Spray passwords, avoid lockouts

en.hackndo.com/password-spraying-lockout

Pixis June 4, 2024

04 Jun 2024 · 19 min

Author : **Pixis**

Password spraying is a well-known technique which consists of testing the same password on several accounts, in the hope that it will work for one of them. This technique is used in many different contexts: On web applications, the Cloud, services like SSH, FTP, and many others. It's also widely used in internal penetration testing with Active Directory. It's the latter that we're going to focus on, because although the technique seems simple, it's not easy to put it into practice without side effects.

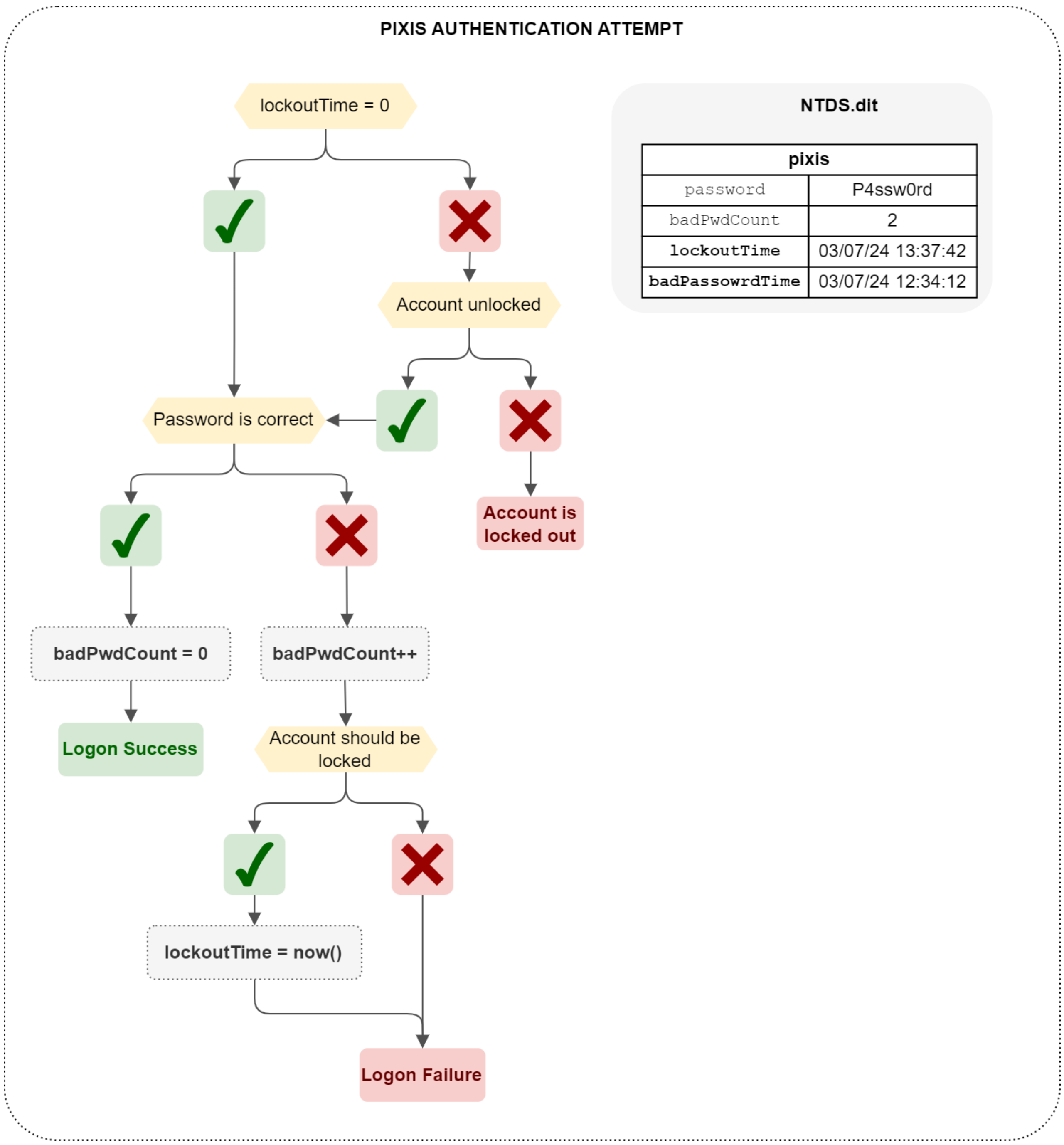
Introduction

This article is not about something new, but rather a report on my research into password policies in an Active Directory environment. Indeed, there are several ways of limiting an attacker by locking accounts. These different levers are very useful when they are understood, which is not always the case (and wasn't the case for me a few weeks ago). This article will, I hope, clarify what password policies allow, how they are applied, and therefore, as a pentester, how to do password spraying while minimizing the risk of locking accounts.

Authentication mechanism

Password spraying on an Active Directory directory consists in choosing a password considered highly likely to be used by at least one user, and testing it on all users in the domain. We won’t go into detail here on how to test the validity of a password, as there are many different ways of doing this. A password can be tested via Kerberos or NTLM, via different services (SMB, LDAP, etc.). In any case, when authentication is tested, the domain controller will first check whether the account is locked by the password policy. If not, it will check the validity of the password. Finally, if the password is valid, the account will be authenticated, and the user’s LDAP `badPwdCount` attribute will be reset to `0`. If, on the other hand, the password is incorrect, then `badPwdCount` will be incremented, and if this failure results in the account being locked, according to password policy criteria, then the account will be marked as locked. This is done by ptoviding the current date and time in the user’s `lockoutTime` LDAP attribute.

That’s quite a verification process, which I’ve tried to summarize in a diagram. Does this make things clearer? I’m not sure. But here it is anyway.



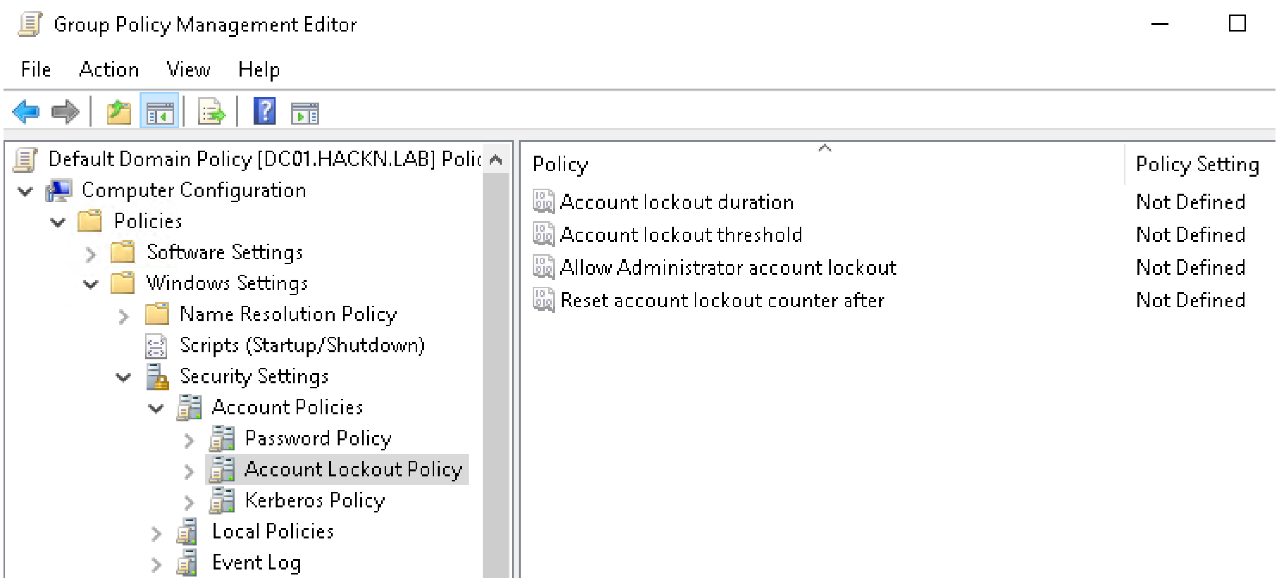
pixis	
password	P4ssw0rd
badPwdCount	2
lockoutTime	03/07/24 13:37:42
badPassowrdTime	03/07/24 12:34:12

Now that we've clarified this logic, we understand that there's one unknown factor that seems rather important: the password policy. Because it's by following the password policy applied to the account that the domain controller is able to know whether or not the account is locked, or whether it should be. Let's take a look at where this default policy can be found, and explain the parameters we're interested in.

The default password policy

When an Active Directory is installed, many things are created and set up by default. Among them are two GPOs: The `Default Domain Policy` linked to the domain root, and the `Default Domain Controller Policy` linked to the `Domain Controllers` OU.

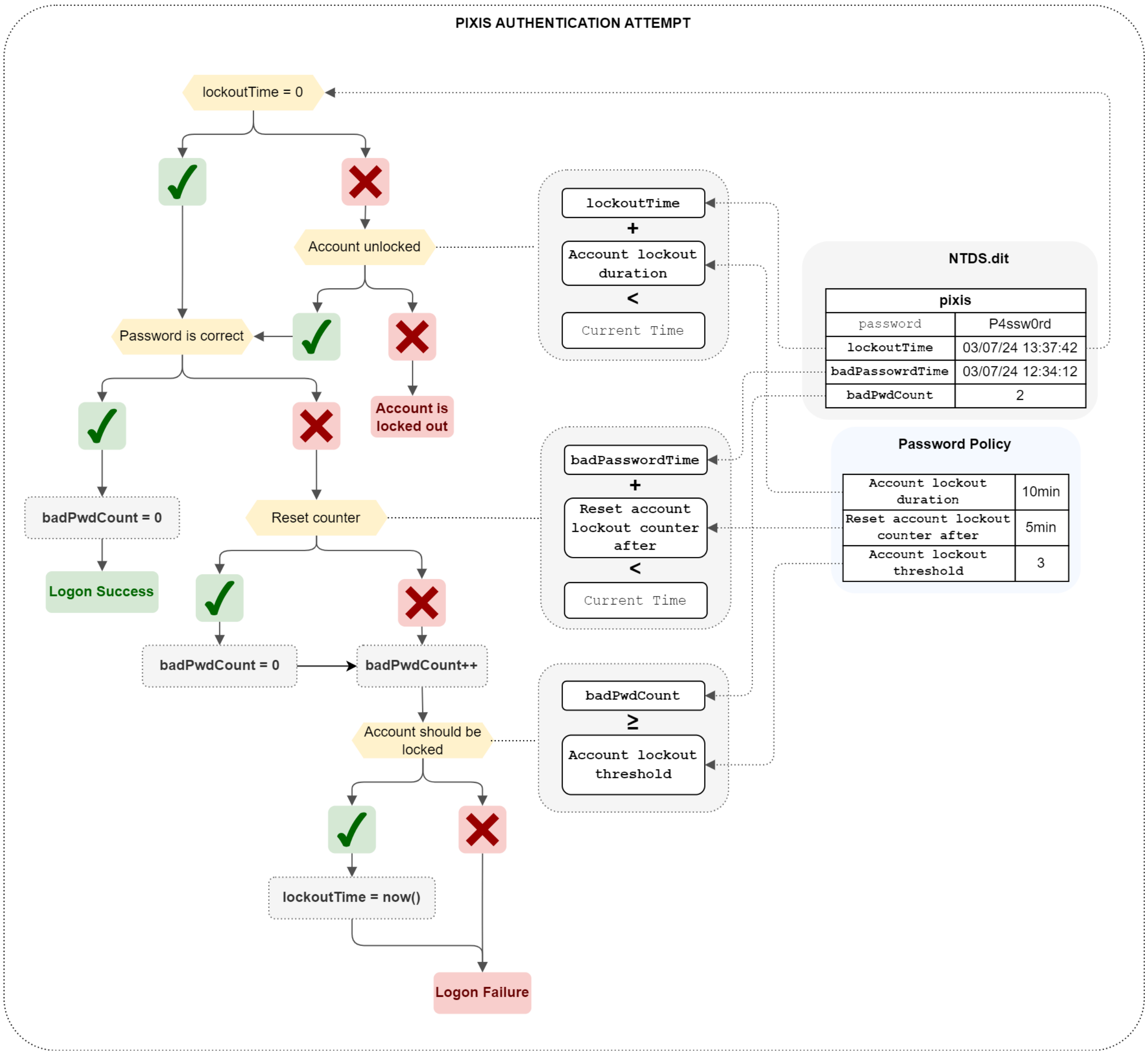
The `Default Domain Policy` contains the following parameters for defining account lockout rules.



Policy	Policy Setting
Account lockout duration	Not Defined
Account lockout threshold	Not Defined
Allow Administrator account lockout	Not Defined
Reset account lockout counter after	Not Defined

- **Account lockout duration** : When an account is locked out, this parameter defines the time during which the account remains locked.
- **Account lockout threshold**: Determines the number of incorrect attempts allowed. If this parameter is `3`, then 3 incorrect attempts will lock the account.
- **Reset account lockout counter after**: As a general rule, when an authentication fails, `badPwdCount` is incremented. However, if the previous failed attempt is older than the time set here, then `badPwdCount` is reset to 0. For example, if this parameter is set to `5 minutes`, then on a first failure, `badPwdCount` is set to **1**. On a 2nd failure a few seconds later, `badPwdCount` is set to **2**. If a 3rd failure occurs, but 5min10 after the 2nd failure, `badPwdCount` will have been reset to **0**, and this failure therefore sets it back to **1**.

These are the parameters that the domain controller takes into account to determine whether an account is locked out. In other terms (and because the previous diagram wasn't complex enough, of course), these values are used in the following way:



As you can see, the bad password counter is calculated by taking the date of the last logon failure, and if the time defined in **Reset account lockout counter after** has passed, then the counter is reset to zero. Otherwise, it is incremented. This counter is then compared with **Account lockout threshold**. If it's equal to (or greater than) this threshold, then the account is locked out, and the `lockoutTime` field is filled in. On a subsequent attempt, if the time defined in **Account lockout duration** has passed (using the date in `lockoutTime` as a starting point) then the account is no longer locked, and the password will be tested.

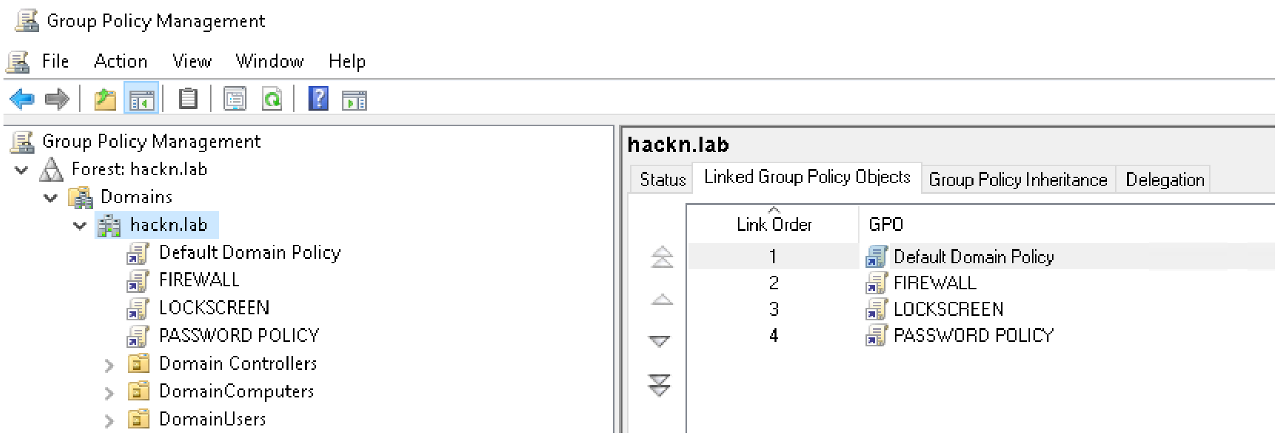
If it's still not clear, then take these explanations from the beginning, and concentrate. I can't explain it any better. All this has to be clear to understand what's coming next. We've already talked about the password policy in the `Default Domain Policy` GPO. But we'll see that this is not the only place where this policy can be defined.

GPOs application order

Personally, I like to create a new, dedicated GPO for every settings (or group of settings) I want to apply to my users or computers. To apply a password policy, I'll create a dedicated GPO, in which there will only be settings related to the password policy. And it'll work just fine. Indeed, we've seen that the password policy can be defined in the `Default Domain Policy`, but they can obviously be set in any GPO.

But how does it work when several GPOs contain different password policies?

There is in fact a **priority order** which defines which GPO takes precedence in the event of a conflict. In the Group Policy Management Console, when you click on an OU (or on the domain), you'll see the list of applied GPOs in the `Linked Group Policy Objects` tab. You'll notice that they are numbered, via the `Link Order` column, which makes it possible to determine the priority of each GPO. Specifically, the last GPO in the list will be applied first, up to GPO number 1. This means that GPO number 1 has the "last word". If its settings conflict with those of other GPOs, its own settings will effectively take effect.



Link Order	GPO
1	Default Domain Policy
2	FIREWALL
3	LOCKSCREEN
4	PASSWORD POLICY

In this example, if one password policy is defined in `PASSWORD POLICY` and another in `Default Domain Policy`, the latter will be applied.

> There is no attribute that allows you to read the GPO number in the list. For each Organizational Unit (and for the domain object), the `gpLink` attribute contains the list of GPOs that are linked to this object, and they are saved in order of application, i.e. in **decreasing** order of `Link Order`. On the `hackn.lab` object, the `PASSWORD POLICY` GPO is listed first in its `gpLink` attribute, then `LOCKSCREEN`, `FIREWALL` and finally `Default Domain Policy`.

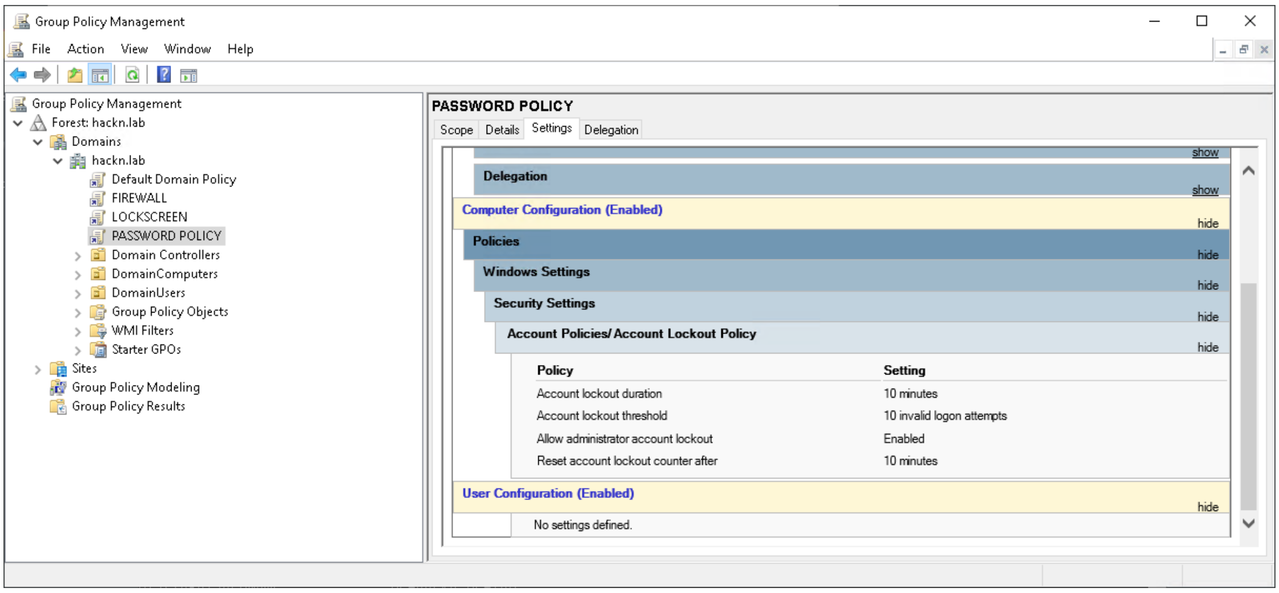
So… By listing the GPOs that are applied to the domain object, we're now able to find out which password policy is actually being applied. Of course, it doesn't stop there. What if we want to apply a different, stronger password policy to the administrators, for example?

Password policy on an Organizational Unit

Before even starting this chapter, I'm announcing that **this won't work**. We'll have to use PSOs to achieve this goal, which we'll see right after.

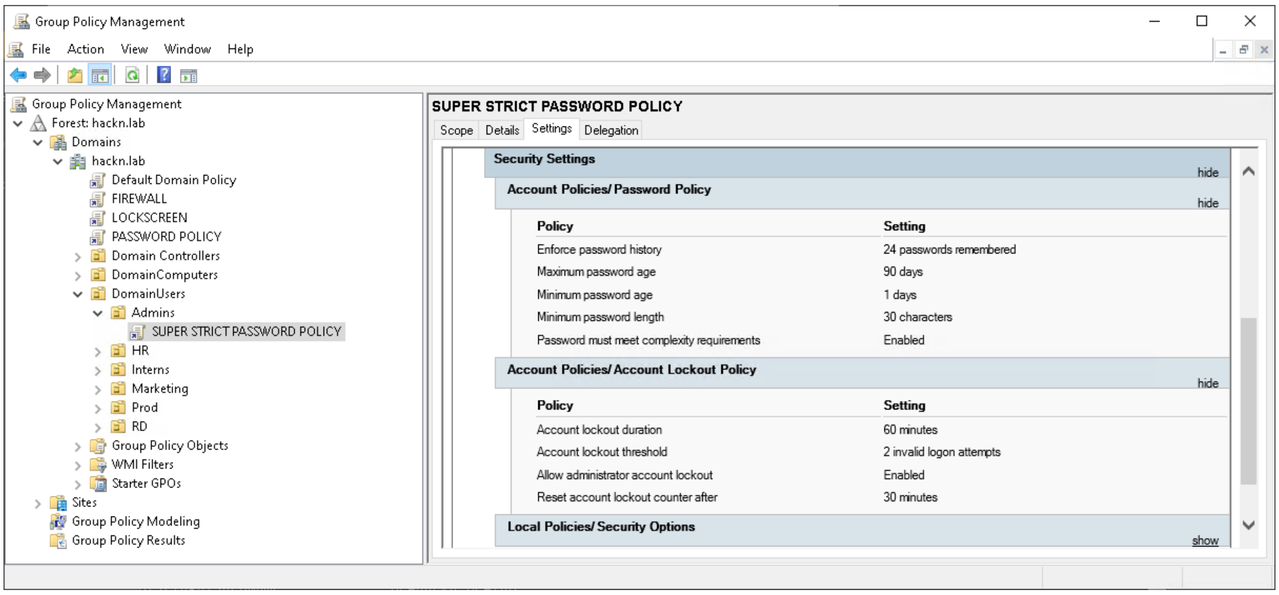
We might be tempted to organize our Active Directory in such a way that our administrators are in a dedicated Organizational Unit, and we'd like to apply a specific password policy to these users. Why not create a GPO dedicated to this OU, in which we define our robust criteria for secure passwords?

Let's take this example. We have a `PASSWORD POLICY` GPO placed at the domain level, which allows you to make 10 mistakes before locking accounts for 10 minutes.



Policy	Setting
Account lockout duration	10 minutes
Account lockout threshold	10 invalid logon attempts
Allow administrator account lockout	Enabled
Reset account lockout counter after	10 minutes

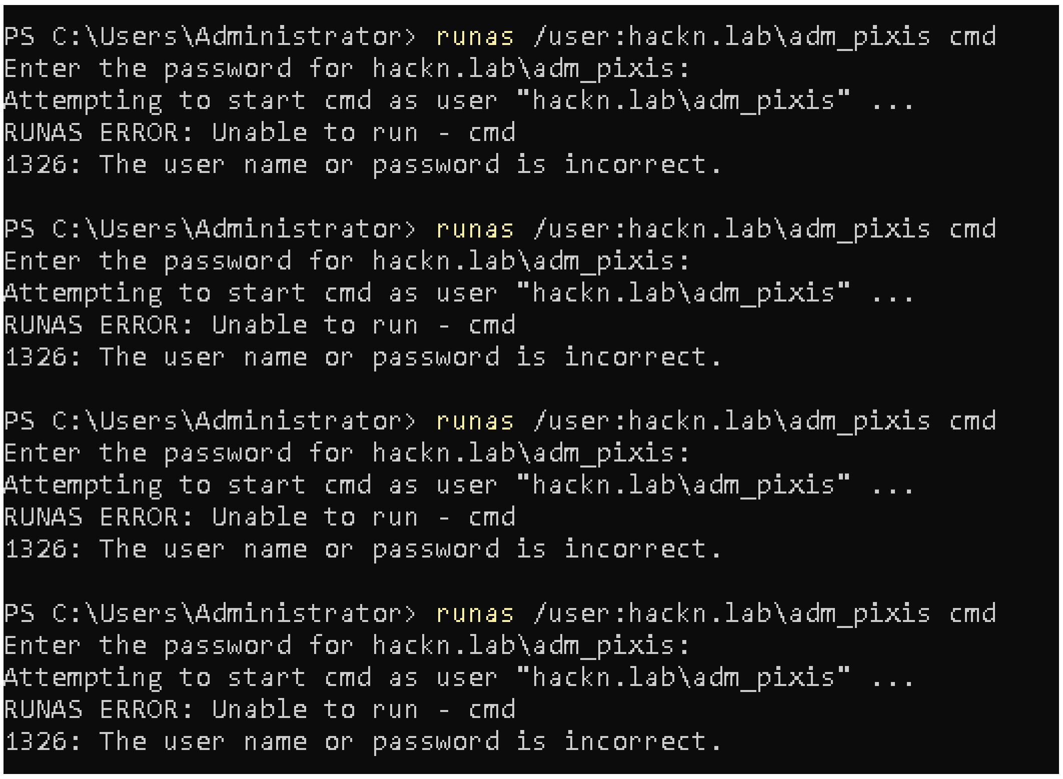
Let's then create a much more robust GPO linked to the `Admins` OU in which the **adm_pixis** account resides. This GPO should lockout an account for 60 minutes after 2 failed attempts.



Policy	Setting
Enforce password history	24 passwords remembered
Maximum password age	90 days
Minimum password age	1 days
Minimum password length	30 characters
Password must meet complexity requirements	Enabled

Policy	Setting
Account lockout duration	60 minutes
Account lockout threshold	2 invalid logon attempts
Allow administrator account lockout	Enabled
Reset account lockout counter after	30 minutes

Now we can try to log in with the user **adm_pixis**, but we deliberately get the password wrong. Once, twice, … three times, four times. The account is still unlocked.



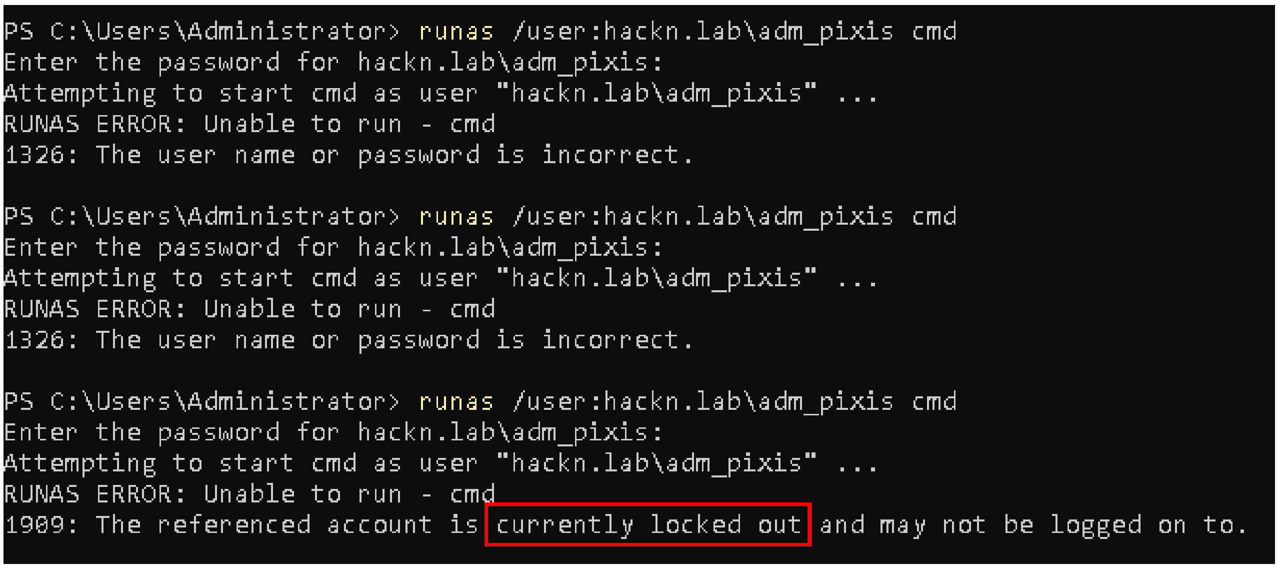
```
PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1326: The user name or password is incorrect.

PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1326: The user name or password is incorrect.

PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1326: The user name or password is incorrect.

PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1326: The user name or password is incorrect.
```

On the 11th try, however, it doesn’t fail: our account is locked.

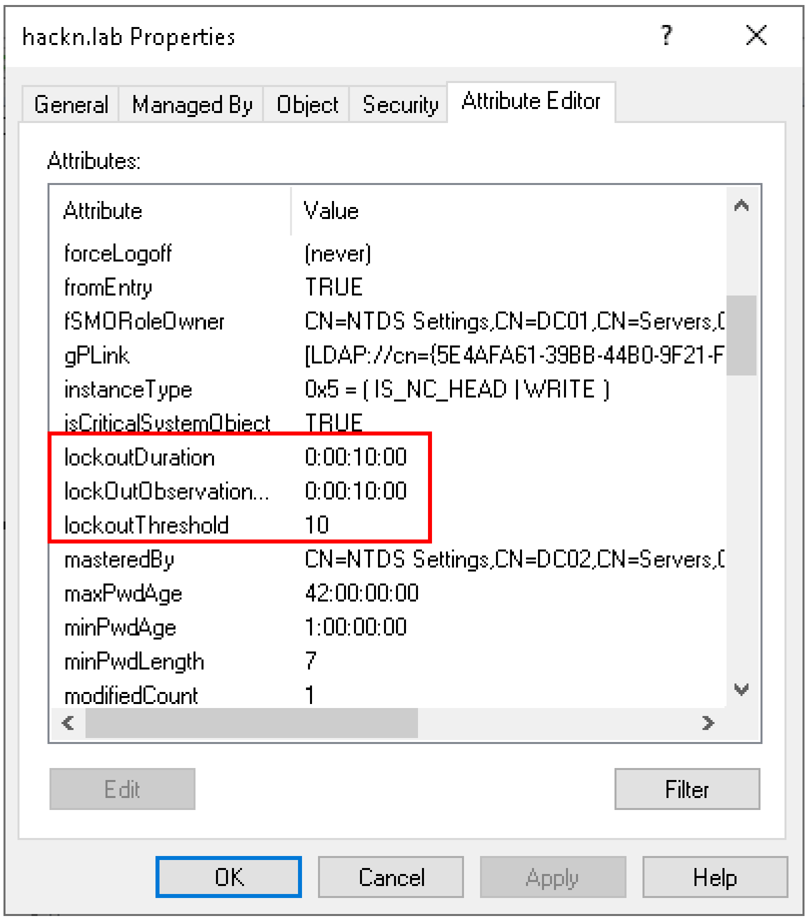


```
PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1326: The user name or password is incorrect.

PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1326: The user name or password is incorrect.

PS C:\Users\Administrator> runas /user:hackn.lab\adm_pixis cmd
Enter the password for hackn.lab\adm_pixis:
Attempting to start cmd as user "hackn.lab\adm_pixis" ...
RUNAS ERROR: Unable to run - cmd
1909: The referenced account is currently locked out and may not be logged on to.
```

Why this black magic? Well, simply because when defining a password policy in a GPO, it will update some specific LDAP attributes of the object to which they are linked, and these attributes **only exist on the domain object**, not on the Organizational Units.


hackn.lab Properties

Attribute	Value
forceLogoff	(never)
fromEntry	TRUE
fSMORoleOwner	CN=NTDS Settings,CN=DC01,CN=Servers,C
gPLink	[LDAP://cn={5E4AFA61-39BB-44B0-9F21-F
instanceType	0x5 = (IS_NC_HEAD | WRITE)
isCriticalSystemObject	TRUE
lockoutDuration	0:00:10:00
lockOutObservation...	0:00:10:00
lockoutThreshold	10
masteredBy	CN=NTDS Settings,CN=DC02,CN=Servers,C
maxPwdAge	42:00:00:00
minPwdAge	1:00:00:00
minPwdLength	7
modifiedCount	1

Here you have the `lockoutDuration` attribute, which corresponds to the **Account lockout duration** parameter, `lockoutObservationWindow` set by **Reset account lockout counter after**, or `lockoutThreshold` which can be defined with **Account lockout threshold**. So, when a password policy is defined by GPO and linked to an OU, this policy will have **no effect** on this Organizational Unit, and therefore no effect on domain accounts in this OU.

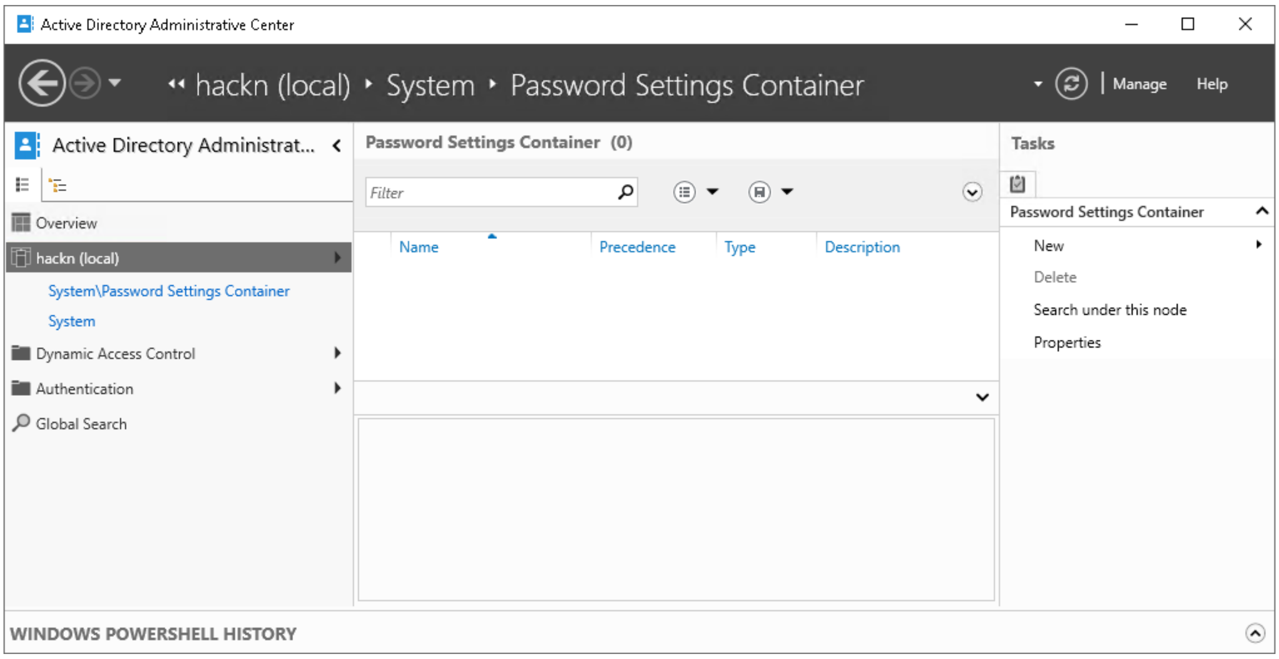
> If you apply a GPO with a password policy to an OU that contains computers (workstations or servers), it has no effect on the domain policy, but it will apply to the **local accounts** of said computers.

All this means that it seems only possible to apply a global password policy to all users. But how do we harden this policy for our administrators? This is where PSOs, or **Password Settings Objects**, come in.

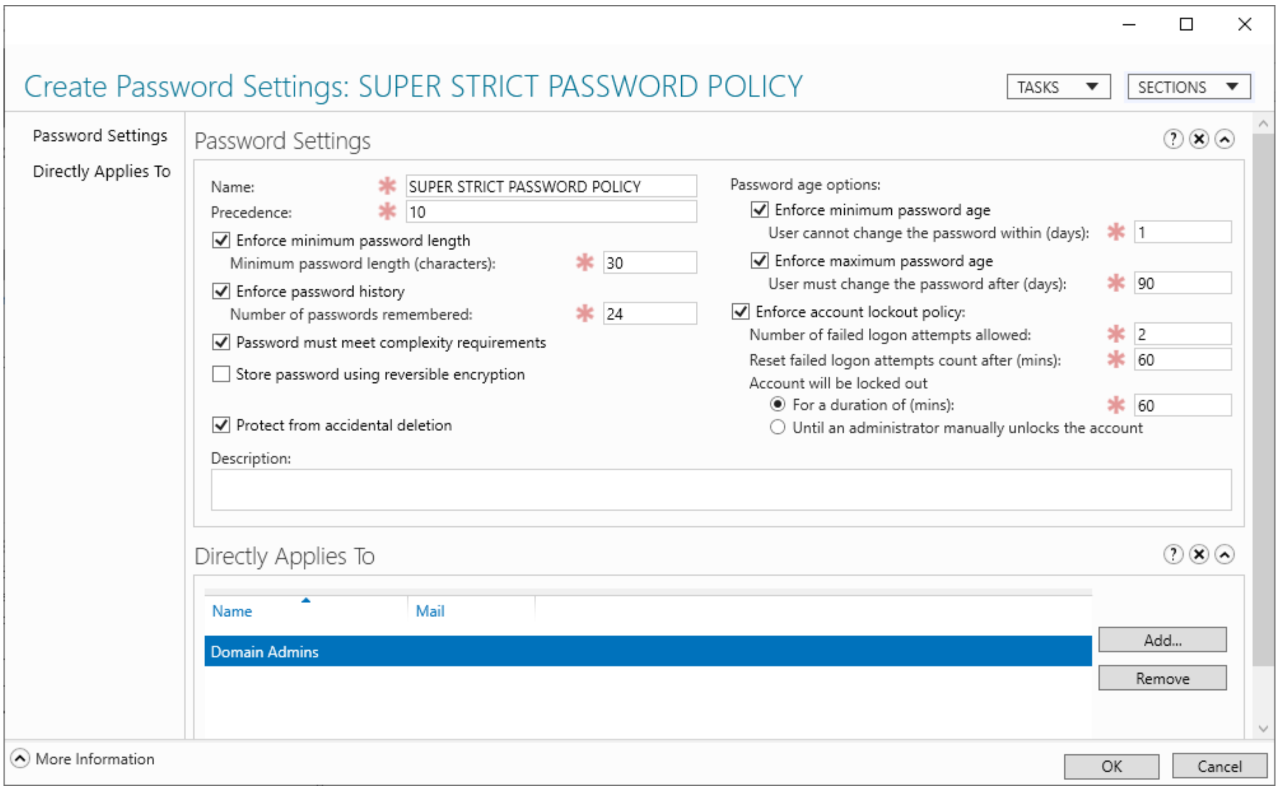
PSOs

Ah, PSOs. The answer to all our problems. PSOs are Active Directory objects that let you define the same password policy settings as those found in GPO, and apply them to users or groups of users. This makes it possible to create **Fine-Grained Password Policies** (FGPP).

These objects must be created in a very specific place, in a container called **Password Settings Container**, itself in the **System** container at the root level. A simple way to create a PSO is to use **Active Directory Administrative Center** tool and navigate to this container.


Active Directory Administrative Center
hackn (local) ‣ System ‣ Password Settings Container
Manage
Help
Active Directory Administrat...
Overview
hackn (local)
System\Password Settings Container
System
Dynamic Access Control
Authentication
Global Search
Password Settings Container (0)
Filter
Name
Precedence
Type
Description
Tasks
Password Settings Container
New
Delete
Search under this node
Properties
WINDOWS POWERSHELL HISTORY

From this console, you can easily create password policies and decide to whom these policies should apply. If we wanted to take over our super secure policy, we could create a PSO with these same parameters and apply it to all members of the **Domain Admins** group.



Create Password Settings: SUPER STRICT PASSWORD POLICY

Password Settings

Name: SUPER STRICT PASSWORD POLICY
Precedence: 10
[✓] Enforce minimum password length
Minimum password length (characters): 30
[✓] Enforce password history
Number of passwords remembered: 24
[✓] Password must meet complexity requirements
[ ] Store password using reversible encryption
[✓] Protect from accidental deletion
Description:

Password age options:
[✓] Enforce minimum password age
User cannot change the password within (days): 1
[✓] Enforce maximum password age
User must change the password after (days): 90
[✓] Enforce account lockout policy:
Number of failed logon attempts allowed: 2
Reset failed logon attempts count after (mins): 60
Account will be locked out
[✓ For a duration of (mins): 60]
Until an administrator manually unlocks the account

Directly Applies To

Name	Mail
Domain Admins	

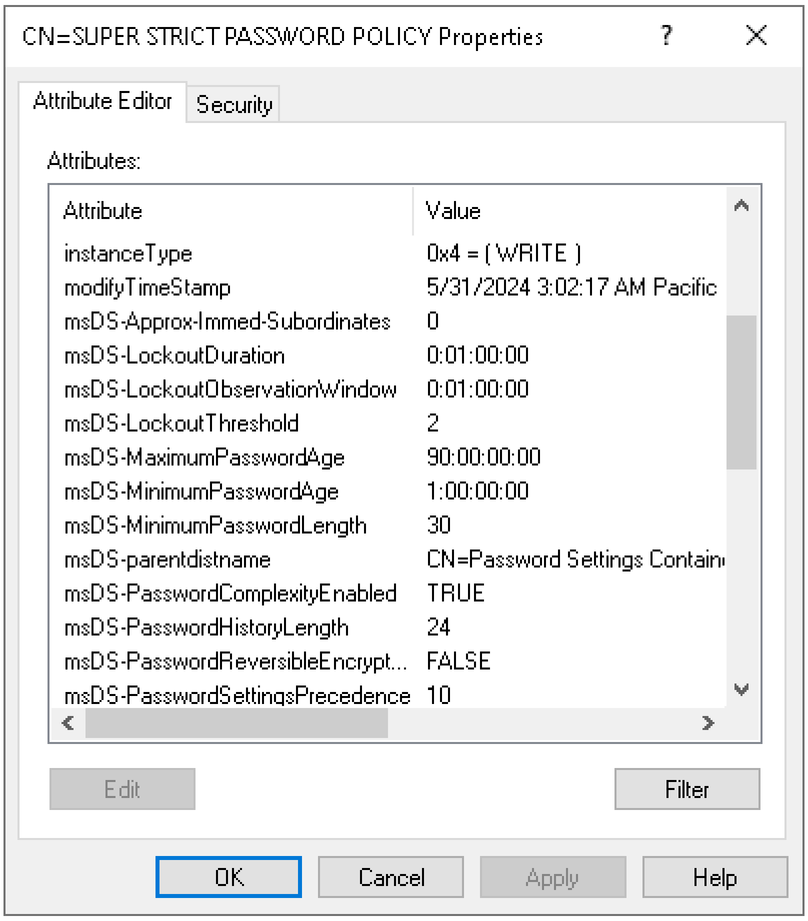
The same question of priority arises as for GPOs. Firstly, PSOs have **priority over the domain's password policy**. But then, if several PSOs are created, and users are affected by different PSOs, which one will be taken into account?

It's the `Precedence` parameter present in the PSO, just after its name, that is used to sort PSOs. Like the `Link Order` for GPOs, PSOs are applied from highest to lowest `Precedence`. This means that the lowest values have priority over the highest, since they also have the last word.

> If two PSOs ever have the same value in `Precedence`, the last PSO created will take precedence.

Let's get a little more technical. We've seen that when GPOs set the password policy, an order of priority will be followed and the attributes of the domain object will be updated to reflect the actual password policy. These domain attributes are readable by all authenticated users, so all you have to do is read the attributes on the domain object to find out the default password policy applied to the domain.

What about PSOs? How do you know if a user is affected by a particular PSO? PSOs can't modify domain attributes, since by their very nature they are designed to have different password policies. In fact, as each PSO is an Active Directory object, password policies are stored in the attributes of the object.

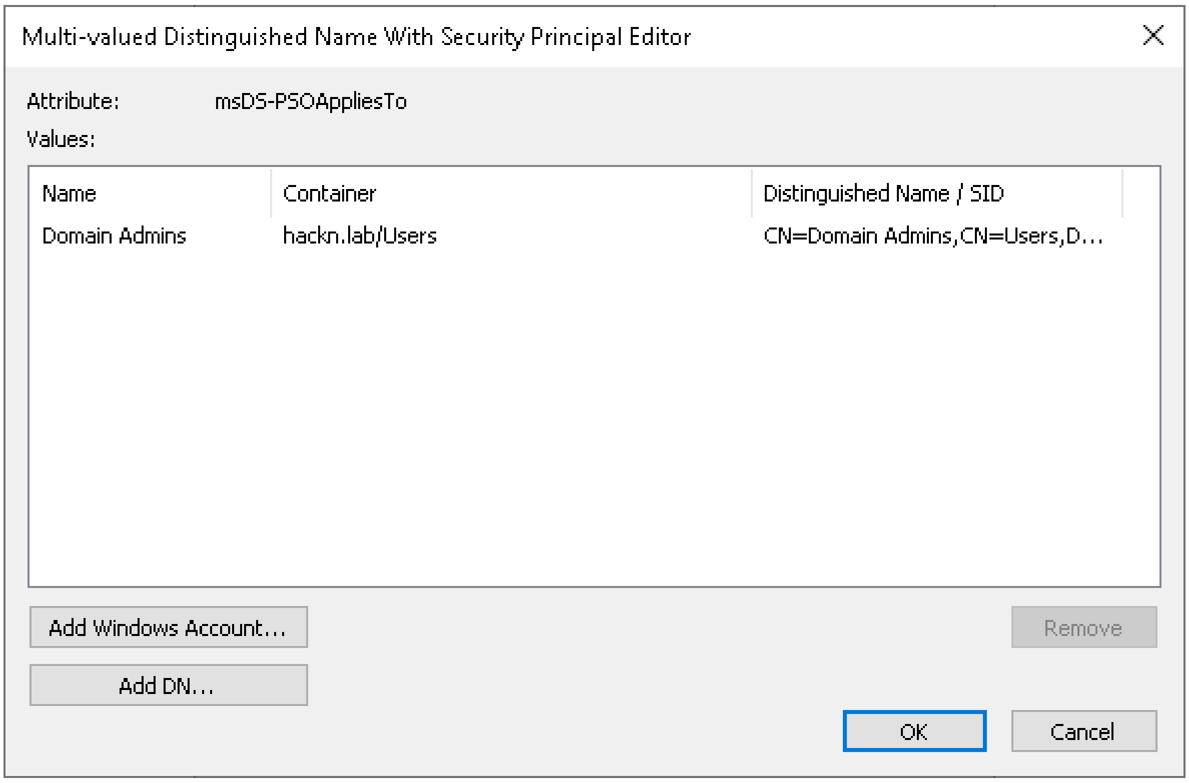

CN=SUPER STRICT PASSWORD POLICY Properties

Attributes:

Attribute	Value
instanceType	0x4 = (WRITE)
modifyTimeStamp	5/31/2024 3:02:17 AM Pacific
msDS-Approx-Immed-Subordinates	0
msDS-LockoutDuration	0:01:00:00
msDS-LockoutObservationWindow	0:01:00:00
msDS-LockoutThreshold	2
msDS-MaximumPasswordAge	90:00:00:00
msDS-MinimumPasswordAge	1:00:00:00
msDS-MinimumPasswordLength	30
msDS-parentdistname	CN=Password Settings Contain
msDS-PasswordComplexityEnabled	TRUE
msDS-PasswordHistoryLength	24
msDS-PasswordReversibleEncrypt...	FALSE
msDS-PasswordSettingsPrecedence	10

The attributes don't have exactly the same names as on the domain, which would be too simple, but they're still there. Still with password spraying in mind, we're particularly interested in `msDS-LockoutThreshold` and `msDS-LockoutObservationWindow`.

Another attribute of great interest to us is `msDS-PSOAppliesTo`. It contains the list of users and/or groups to which the PSO applies.


Multi-valued Distinguished Name With Security Principal Editor

Attribute: msDS-PSOAppliesTo

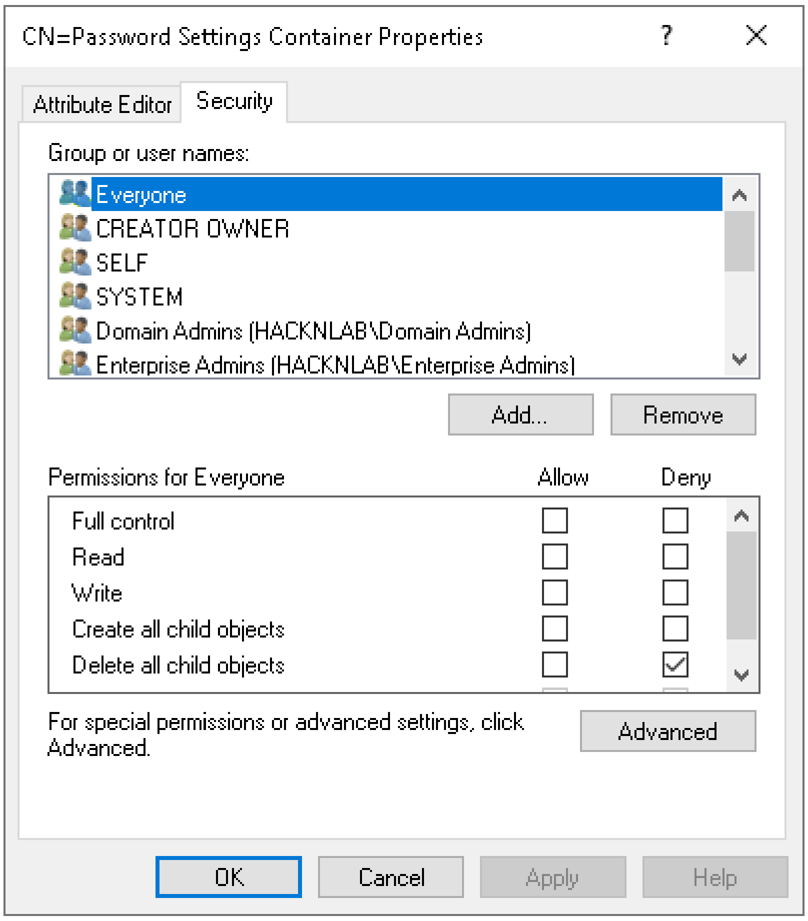
Values:

Name	Container	Distinguished Name / SID
Domain Admins	hackn.lab/Users	CN=Domain Admins,CN=Users,D...

PSO container rights problem

At this stage, you might think that we have all the information we need to know the effective password policy for each user. By default, we take the domain password policy by reading the domain object attributes, and then we list all the PSOs in the right order of priority, list the users in the groups on which each PSO is applied, and we can thus deduce which PSO is applied to which user.

In theory, this would work, but there’s a small hitch in this approach.



CN=Password Settings Container Properties

Attribute Editor | Security

Group or user names:

- Everyone
- CREATOR OWNER
- SELF
- SYSTEM
- Domain Admins (HACKNLAB\Domain Admins)
- Enterprise Admins (HACKNLAB\Enterprise Admins)

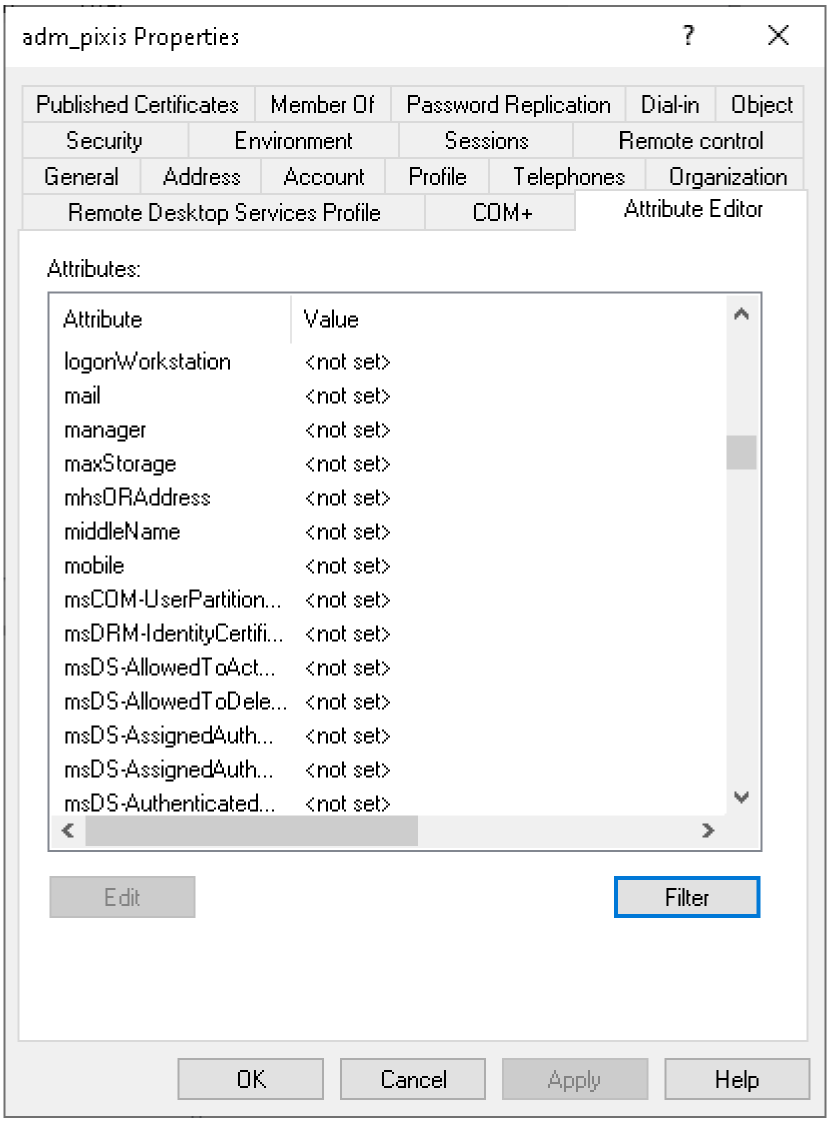
Permissions for Everyone	Allow	Deny
Full control	☐	☐
Read	☐	☐
Write	☐	☐
Create all child objects	☐	☐
Delete all child objects	☐	✓

For special permissions or advanced settings, click Advanced.

Can you see the problem? No ? By default, only administrators have the right to list PSOs. So, as an ordinary user, we have no way of listing the contents of the **Password Settings Container**, and therefore of seeing the PSOs, the policies applied, and to whom they are applied. For our purposes of password spraying, this is quite dangerous. This means we can never be certain of knowing the effective password policies for various domain users, as there is always a risk that a PSO (which we cannot see or read) is applied to one or more users.

Constructed Attributes & Backlinks

It was while facing this wall that I discovered constructed attributes. You probably know that when a user is added to a group, the `member` attribute of the group is updated, adding the user. However, if you simply list a user's attributes, you won't see any LDAP `memberOf` attribute.

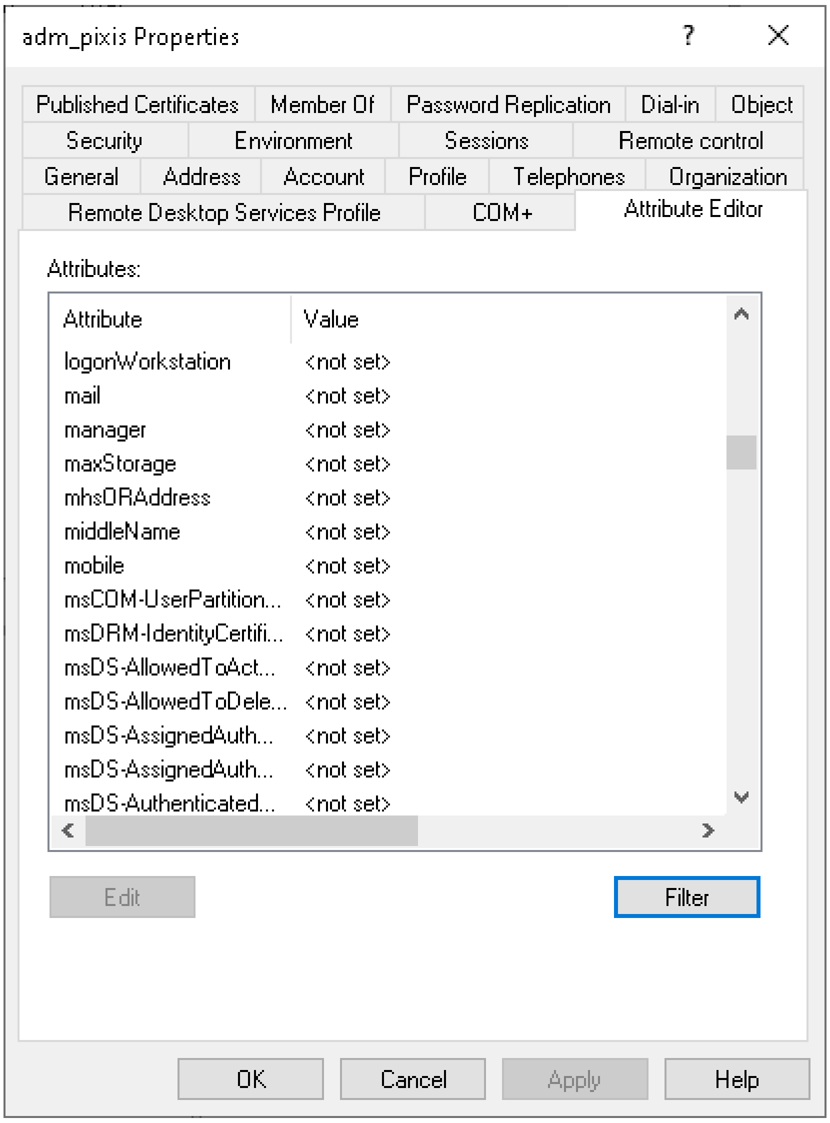

adm_pixis Properties

Attributes:

Attribute	Value
logonWorkstation	<not set>
mail	<not set>
manager	<not set>
maxStorage	<not set>
mhsORAddress	<not set>
middleName	<not set>
mobile	<not set>
msCOM-UserPartition...	<not set>
msDRM-IdentityCertifi...	<not set>
msDS-AllowedToAct...	<not set>
msDS-AllowedToDele...	<not set>
msDS-AssignedAuth...	<not set>
msDS-AssignedAuth...	<not set>
msDS-Authenticated...	<not set>

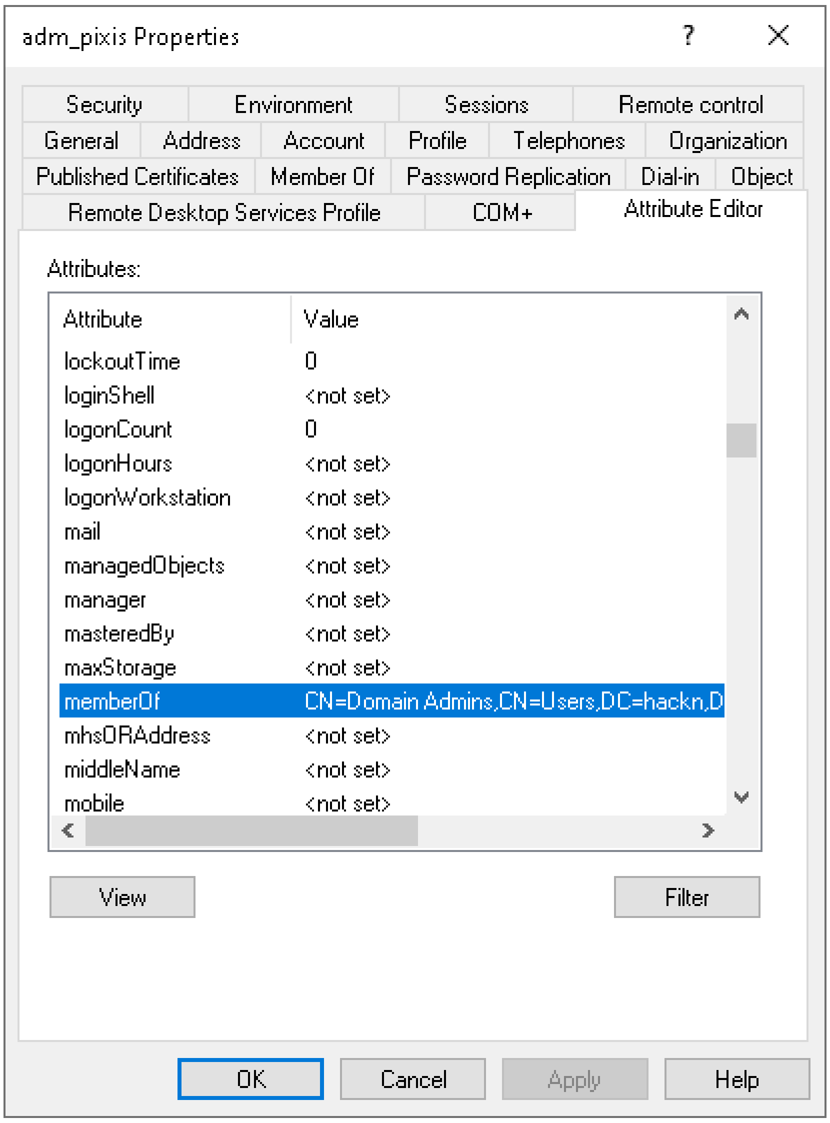
There are many more attributes than those shown by default, but they are automatically managed by Active Directory. You can't change them manually. These are attributes built from other attributes.

Among them, one class of attributes is called **backlinks**. These are attributes that go hand in hand with another attribute, on the same object or on other objects.



Attribute	Value
logonWorkstation	<not set>
mail	<not set>
manager	<not set>
maxStorage	<not set>
mhsORAddress	<not set>
middleName	<not set>
mobile	<not set>
msCOM-UserPartition...	<not set>
msDRM-IdentityCertifi...	<not set>
msDS-AllowedToAct...	<not set>
msDS-AllowedToDele...	<not set>
msDS-AssignedAuth...	<not set>
msDS-AssignedAuth...	<not set>
msDS-Authenticated...	<not set>

For example, the `memberOf` attribute is a backlink that goes hand in hand with the `member` attribute. So, as soon as the `member` attribute of an object is modified (adding or deleting a user or group), this change is automatically reflected in its pair, `memberOf`, of the object concerned. So if we display these backlinks, we can see the list of groups to which a user belongs.

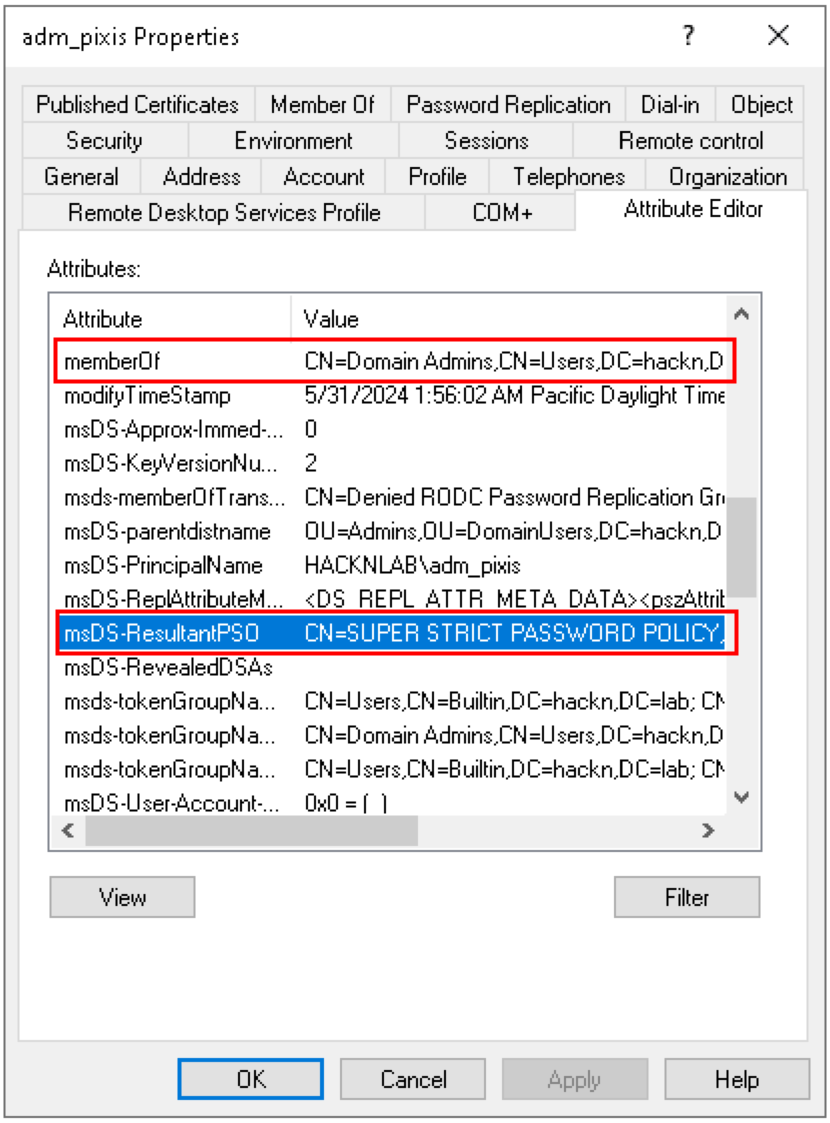


Attribute	Value
lockoutTime	0
loginShell	<not set>
logonCount	0
logonHours	<not set>
logonWorkstation	<not set>
mail	<not set>
managedObjects	<not set>
manager	<not set>
masteredBy	<not set>
maxStorage	<not set>
memberOf	CN=Domain Admins,CN=Users,DC=hackn,D
mhsORAddress	<not set>
middleName	<not set>
mobile	<not set>

Well, guess what, the same applies to PSOs. We're not allowed to list PSOs with a standard user, so we can't read the `msDS-PSOAppliesTo` attribute present on each PSO. But we're in luck, there's a **constructed link** for this attribute, and it's called msDS-ResultantPSO. It's not a **backlink**, because it's a bit smarter. All users in the domain have this attribute, which is potentially updated every time a PSO is applied to users, but also to groups. If a group is added to a PSO, all members of that group will have their `msDS-ResultantPSO` attribute updated, **provided the PSO has priority**. So as well as dynamically resolving group members, this attribute will always contain the name of the **effective** PSO that applies to each user.

And the icing on the cake in all this is that **this attribute can be read on all users in the domain, even with a non-privileged account**.

Remember, we applied a PSO to the **Domain Admins** group. Let's take a look at the **constructed** attributes of the **adm_pixis** account, which is part of this group.


adm_pixis Properties

Attribute	Value
memberOf	CN=Domain Admins,CN=Users,DC=hackn,D
modifyTimeStamp	5/31/2024 1:56:02 AM Pacific Daylight Time
msDS-Approx-Immed-...	0
msDS-KeyVersionNu...	2
msds-memberOfTrans...	CN=Denied RODC Password Replication Gr
msDS-parentdistname	OU=Admins,OU=DomainUsers,DC=hackn,D
msDS-PrincipalName	HACKNLAB\adm_pixis
msDS-ReplAttributeM...	<DS_REPL_ATTR_META_DATA><pszAttrit
msDS-ResultantPSO	CN=SUPER STRICT PASSWORD POLICY,
msDS-RevealedDSAs	
msds-tokenGroupNa...	CN=Users,CN=Builtin,DC=hackn,DC=lab; CN
msds-tokenGroupNa...	CN=Domain Admins,CN=Users,DC=hackn,D
msds-tokenGroupNa...	CN=Users,CN=Builtin,DC=hackn,DC=lab; CN
msDS-User-Account-...	0x0 = ( )

We can clearly see our `memberOf` **backlink**, which contains **Domain Admins**, as well as our `msDS-ResultantPSO` **constructed attribute**, which contains the PSO we created and applied to the **Domain Admins** group.

It's perfect, now we can see whether or not a user's password policy is affected by a PSO. Mind you, **we still don't have the right to read the contents of the PSO**, at least not as a standard user. But from a password spraying point of view, just knowing that a PSO has been applied to a user is extremely valuable. We can simply ignore all accounts for which a PSO is applied, and restrict our tests to users for whom the effective password policy is the default domain policy.

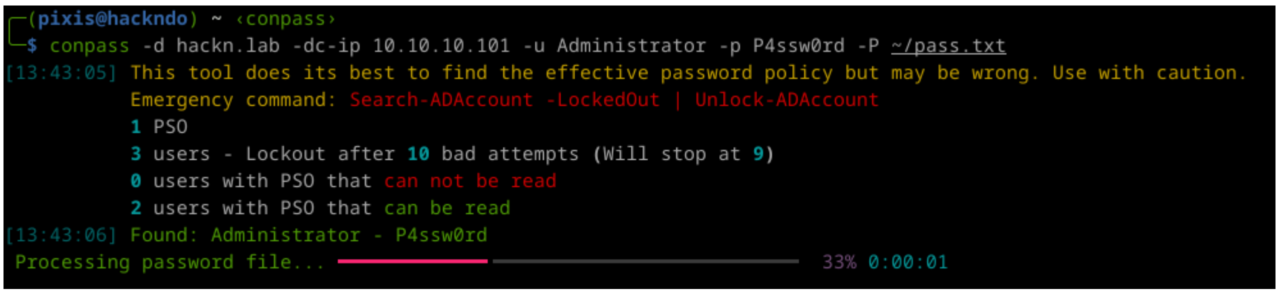
Tools

I followed this white rabbit of password policies by writing a password spraying tool, in an attempt to minimize the risk of locking accounts during my penetration tests. The tool I've developed follows this whole process to determine the list of users and the effective password policy for each of them. If the tool is unable to read the contents of the PSO, it will simply ignore the affected accounts during the spraying.

But I wanted to go a step further. We saw that after a certain period of time, the `badPwdCount` attribute of users was reset. This certain duration, found in the `lockoutObservationWindow` attribute on the domain, or in the `msDS-LockoutObservationWindow` attribute on a PSO, can thus vary from one user to another. The tool I've developed takes as input a list of passwords we wish to test on users, and will try all the passwords on all the users, taking care to follow locking thresholds and waiting times for the counter to be reset to zero before continuing with the tests.

To ensure that everything runs smoothly, the tool starts by retrieving the time from the domain controller so that it's perfectly synchronized, and synchronizes regularly with LDAP (a real user might be trying a wrong password during our password spraying).

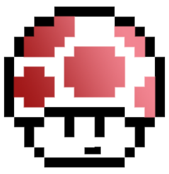
In short, I'm very happy to share with you the conpass tool, which is extremely useful for me in penetration testing, and I hope it will be as useful for you.



```
┌─(pixis@hackndo) ~ ‹conpass›
└─$ conpass -d hackn.lab -dc-ip 10.10.10.101 -u Administrator -p P4ssw0rd -P ~/pass.txt
[13:43:05] This tool does its best to find the effective password policy but may be wrong. Use with caution.
           Emergency command: Search-ADAccount -LockedOut | Unlock-ADAccount
           1 PSO
           3 users - Lockout after 10 bad attempts (Will stop at 9)
           0 users with PSO that can not be read
           2 users with PSO that can be read
[13:43:06] Found: Administrator - P4ssw0rd
 Processing password file...                                          33% 0:00:01
```

Be careful, it worked during my last pentests, but it's still a tool written by a human who makes mistakes, that's going to test passwords. So it's possible that there are bugs and that it crashes, or that it locks accounts. So I'm very keen to get feedbacks from tests in controlled environments, to make it as robust as possible.

Author : Pixis
Blog author, follow me on twitter or discord



Similar posts