Active Directory User and Group Reporting: A practical guide for administrators

Written by SecureHero LLC

30.09.2015

Contents

The importance of Active Directory reporting

It is a simple fact that as the time goes by, Active Directory gets messy. Lack of automated provisioning and de-provisioning practices, employee turnover, Active Directory restructuring due to ongoing M&A activity are just a few reasons why. A good evidence of a messy Active Directory known as the “group sprawl” is when you end up having more groups than you have users.

Not only can it negatively affect the operational state of Active Directory. It might also put a huge toll on the bottom line of the organization itself. Affected companies might face:

- Increased security risks – not knowing what groups are used for and leaving dormant accounts in AD creates opportunities for offenders to break into your network. This becomes a particularly easy target for company ex-employees or administrators that are intimately familiar with the corporate network and structure of Active Directory.
- Affected productivity of end users – when users are members of excessive number of groups they might run into the “token bloat” problem that prevents them from accessing certain resources or conversely yields them access to resources they should not have access to.
- Compliance challenges – how do you prove to an auditor that you keep organization access policies under control when you don’t even know what Active Directory groups are used for? How can you be confident that groups still serve the same purpose they used to serve 5 years ago?

So, where do you start to put things back in order?

What about getting to know your Active Directory first? Start with a thorough assessment of Active Directory users and groups that would spot unnecessary groups and dormant accounts.

In this whitepaper we are going to show Top 10 reports that will help you improve Active Directory hygiene. We will talk about 10 different things you want to know about users and groups in Active Directory and give you a ready-made “do it yourself” recipe to quickly check things off from the Active Directory best practices list.

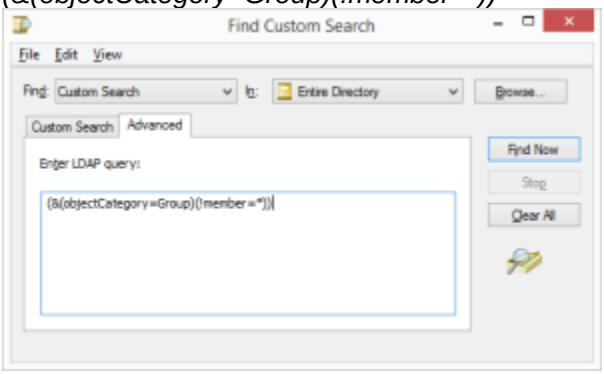
So, if you are tired of coming across stale accounts and unused groups, if “group cleanup” has been keeping you up at night, if you want to see what can be the first step on your way to a well-managed and compliant Active Directory, read on.

#1 Empty groups

In this chapter we'll explore how to discover empty groups in Active Directory. Empty groups are groups that have no members. Obviously, if this condition does not change for a while you might want to consider deleting those groups.

First, let's see how this can be done in ADUC. If you are familiar with custom searches here is a simple LDAP filter that will do the trick:

(&(objectCategory=Group)(!member=))*



The problem with ADUC is that you can't easily select a set of fields you would want to see for each group. This makes it difficult, for instance, to see which of those groups have not been changed for a long time.

This is where PowerShell comes in handy. .

Thanks to the Active Directory module for PowerShell introduced for Windows Server 2008 R2 and later, identifying empty groups is a matter of the two commands:

C:\> $last = (Get-Date) – (new-timespan -days 90)
C:\> get-adgroup -SearchScope Subtree -SearchBase "DC=toronto,DC=local" -Properties whenChanged -filter {(member -notlike "") -and (whenChanged -le $last)} | select-object -property distinguishedName, whenChanged | export-csv c:\test\emptygroups.csv*

In the PowerShell code above we do the following:
1. Set the $last variable to 90 days ago.

2. Find groups with no members using the same LDAP filter used with the correct syntax expected by the Get-ADGroup cmdlet:

(member -notlike “”)*

3. Check the last changed date of each found group and leave only those that have not been changed within the last 90 days:

(whenChanged -le $last)

4. Select the properties we want to see for each resulting group:

select-object -property distinguishedName, whenChanged

5. Save a list of groups into a csv file:

export-csv c:\test\emptygroups.csv

This simple enough PowerShell script can be a huge time savior and it gives the right results with one exception. It does not factor in primary groups.

Primary groups do not store members in its member attribute. Instead their membership is calculated based on the backlink stored in the user object that is included into that group. So, if all users in a group have this group listed as their primary group, than the actual member attribute will appear empty and such group will be picked up by the script above. Usually this is the case with built-in group Domain Users.

#2 Circular nested groups

In this chapter we are going to see how we can uncover another type of toxic and potentially unnecessary Active Directory objects – circular nested groups.

First let's define what circular nested groups are. These are groups that include themselves by a virtue of nested group memberships. Consider this example:

Domain Admins group includes another group called Dallas CSI. The latter in turn happens to include Domain Admins as its member. Now the full membership path will effectively form an infinite "loop":

Domain Admins->Dallas CSI->Domain Admins

Why groups with circular nested membership are bad? While having such groups will not affect the state of Active Directory itself, it is considered a bad Active Directory management practice.

First of all, circular nested groups is a sign of improperly implemented access control policy. The more circular groups you have the more difficult it becomes to follow the "path of access". It also opens a room for errors and unpredictable results when delegating access through such groups. And lastly, never forget about all the scripts and poorly written third party applications that might not handle such groups appropriately. They might end up being stuck in the infinite loop trying to enumerate all members of such groups. You will be surprised how many scripts and applications of this kind are out there.

Fortunately, finding such groups using PowerShell is an easy task. Below you will find the full recipe.

C:\> Get-AdGroup -filter {(member -like "")} -Properties member | %{ Get-AdGroup -filter {(distinguishedName -like $_.DistinguishedName) -and (memberof -recursivematch $_.DistinguishedName)}}*

The code snippet uses the same cmdlets and syntax that we discussed in the previous chapter. The same explanation and parameter choices apply here. However, there is one caveat you should be mindful of.

Unlike with empty groups, this script pulls every group from Active Directory and queries all its members including nested groups. If you don't limit the out most query with the *SearchScope* parameter, it can easily put a strain on your domain controllers for a considerable amount of time.

#3 Groups with same direct members

One way of getting rid of excessive groups is to find which groups have similar membership. The simplest case is when the list of direct members of a group matches another group's list.

Groups with same direct members are not always a bad practice but they definitely need review. As a result of this review, one or a few of such groups might be considered duplicate. In this case all references to the duplicate groups must be replaced with another group with similar members. Not only it will help you simplify on-going management of remaining groups but it will also reduce a risk of so called "token bloat" issues stemming from having too many groups a user is a member of.

Here is a PowerShell code snippet that finds groups with matching direct members.

```
$groups = Get-AdGroup -filter {(member -like "*")} -Properties member
$map = @{}
$groups | %{
  $members = [string]$_.member
  if($map.Contains($members)){
     $map[$members] = $map[$members] + @($_)
  }
  else{
     $map[$members] = @($_)
  }
}
$map.GetEnumerator() | ? { $_.Value.Count -gt 1 } | % { [string]$_.Value }
```

Like in the previous chapter about circular nested groups, this script loads all Active Directory groups first. You can use the *-SearchScope* parameter to limit the scope of groups that will be examined. Depending on the number of groups under investigation the script might consume significant amount of time and CPU.

Once groups have been fetched, the script builds a map of unique group memberships. The sorted list of group members serves as a key to that map. This way, when another group with the same list of direct members is found, the map looks up a matching group by the same key and adds another group into the same map element.

The output of this script shows sets of groups separated by a new line. Each set represents groups that have matching group members. Here is an example of the script output:

CN=Admins,OU=Groups,DC=toronto,DC=local
CN=PrivAccess,OU=Groups,DC=toronto,DC=localCN=All_Admins,OU=Groups,DC=toronto,
DC=local
CN=group1,OU=Groups,DC=toronto,DC=local
CN=distrib_group1,OU=Development,DC=toronto,DC=local

#4 Groups with same nested members

Putting exactly the same members into different groups is not the only way to create duplicate groups. Because Active Directory allows nested groups, two seemingly different groups might end up having same “effective” user members. Consider this simple example:

GroupA GroupB

———>GroupAA ————>GroupBB

————————>UserA ————————->UserB

UserB UserA

GroupA and GroupB have totally different direct members. However, when membership of nested groups GroupAA and GroupBB is expanded, the list of resulting user members starts to match!

Groups that have the same effective members should be closely examined to see if some of them are redundant. This best practice makes your access control policies more transparent and error proof.

Unfortunately, there is no easy way to detect groups with same nested members with native tools. Trying to accomplish this with a PowerShell script is challenging but possible. When devising such a script consider the following:

1. The script has to fetch every Active Directory group along with the full list of its members.

2. For every Active Directory group that has a nested group, another query has to be executed to get the list of the nested group members.

3. While expanding nested group members, make sure to check for circular groups to avoid infinite recursion.

4. Since all groups and their nested members have to be queried, the script might consume a lot of RAM and CPU. Run the script on a computer with adequate system resources.

5. Depending on the number of nested groups, script might issue a lot of LDAP queries to a domain controller. It is recommended to limit the scope of the group comparison to select OUs and run the script during non-business hours to minimize the impact on domain controllers.

#5 Groups without managers

In well managed Active Directory every group has its purpose. As the number of groups grows and Active Directory spreads out globally, it becomes challenging to manage all groups from one central authority. This is where you start thinking about delegation of group management.

In the simplest case, management of groups associated with particular division, business function or location is delegated to lower level administrators or managers. It is considered a good practice to publish group managers in Active Directory. For that purpose every group has the managedBy attribute which is conveniently shown in native AD management tools.

If your Active Directory utilizes managedby attribute for groups then it is a good idea to leverage it for Active Directory cleanup. Here is a simple PowerShell snippet that finds all groups without managers and dumps the resulting list of groups into a csv file.

C:\> get-adgroup -LDAPFilter "(!managedBy=)" -SearchScope Subtree -SearchBase "DC=toronto,DC=local" | select-object -property distinguishedName | export-csv c:\test\groups.csv*

Note that unlike with previous examples, we are specifying the LDAP filter using a different method. This is because managedBy is an extended attribute and regular filter syntax does not work here.

This is the easy part. The not so easy part is what you are going to do with the resulting list of groups. If a group does not have a manager, it does not necessarily mean that it is no longer used. However, it is worth trying to find the probable manager for such groups to ensure its proper management in the future. Here are a few of the ideas taken from the real world that show you how:

1. Examine the list of users included into a group and look up their organizational chart. Most companies publish their organizational structure in Active Directory, so this information is readily available. Users that manage most of the group members can be nominated to the group manager.

2. Look at the group object permissions. Users that have Modify permission on the group object are most likely its managers. Look at the owner attribute of the group permissions as well.

3. If you have been collecting Active Directory object change events into Security log, or have been using third party applications for Active Directory auditing, try looking up who created the group in the first place.

4. Email every user included in the group and ask them about the purpose of this group as well as who might be in charge of its management.

#6 Old groups

Finding Active Directory groups that have not been changed in a long time can reveal other candidates for thorough inspection and possible clean up. Indeed, if a group has not been changed it means that nobody was added or removed from this group either. So, may be this group have been abandoned and is no longer used?

This is for you to find out. Here is the first step showing how to detect old groups:

C:\> $last = (Get-Date) – (new-timespan -days 365)
C:\> get-adgroup -SearchScope Subtree -SearchBase "DC=toronto,DC=local" -Properties whenChanged -filter {(whenChanged -le $last)} | select-object -property distinguishedName, whenChanged | export-csv c:\test\oldgroups.csv

This PowerShell snippet is very much like the one we've used in the chapter about empty groups. We are looking for all groups that have not been changed over the course of the last year to date (365 days). We then select their name and last changed date and dump the resulting list into a csv file.

One thing requires extra attention here. As you can see, we use the whenChanged attribute as an indication of when the group object was changed last time. According to Microsoft, whenChange is a non-replicated attribute. However it gets updated locally on each DC once replication cycle kicks in. This might seem bullet proof and yet it can leave a considerable discrepancy in the attribute values across different DCs. If replication is not working properly, whenChanged might be lagging behind for quite a bit.

That's why it is important that you choose the most up-to-date domain controller such as PDC Emulator when running this script.

#7 Locked Out User Accounts

Now that we've covered what should be a part of your daily Active Directory groups hygiene, let's see what should be the top reports to run on users.

One thing that happens in every Active Directory domain daily is user accounts locking out. It is a good practice to have a reasonable Account Lockout policy in place. Most of the time there is a legit explanation behind every account lockout. For example, it can be user's mail application on mobile device that is using old password. Or a mapped network drive on user's desktop that is trying to connect to a file share every time the desktop operating system reboots. But sometimes locked out user accounts can signify a potential brute force attack happening right in front of your eyes. So, how do you review account lockouts and pinpoint ones that should be given more thorough investigation?

For starters, here is a simple PowerShell command that will return all locked out accounts in the domain:

Search-ADAccount –LockedOut

It is beautifully simple but has one downside. It shows a fixed set of attributes for every locked out user account. This information might not be enough to weigh in the severity of each lockout.

To solve this problem, we are calling out another more flexible way of retrieving user account lockouts using the Get-ADUser command let:

*Get-ADUser -Filter * -Properties SamAccountName,LastBadPasswordAttempt,badPwdCount, LockedOut | Where{$_.LockedOut-eq $true} | Select-Object SamAccountName,LastBadPasswordAttempt,badPwdCount*

In this one liner we have control over the set of attributes that will be shown for every locked user account. In particular we are pulling the following attributes:

1. SamAccountName
2. LastBadPasswordAttempt
3. badPwdCount

The first two attributes are self-explanatory and they just give you some basic context. The badPwdCount attribute is the crown jewel. This attribute keeps counting the number of failed bad password attempts before user successfully logs on. If this attribute value does not go well beyond the max failed password attempts setting in your Account Lockout Policy, then it is probably a poor user that has forgotten his or her password. If, on the other hand, this count keeps growing from one account lockout to another, it can indicate something more serious going on in your network.

Another bit of information that is very helpful in troubleshooting account lockouts is the computer where bad password logon attempts are coming from. Unfortunately, Active Directory does not store this information. To obtain it you can try using scripts that pull this data from event logs or, better yet, rely on products that do that all for you automatically such as SecureHero Logon Reporter.

#8 Disabled User Accounts

According to critical security controls from SANS *"any account that is dormant must be disabled and eventually removed from the system".*

When employees leave the company or contractor accounts are no longer used, it is certainly a wise decision to terminate their access. Most companies do not rush removing accounts of terminated users though. Instead they prefer to disable these accounts and leave them in this "sanitization" state for some time that usually varies from 30 to 90 days. Indeed, when accounts are disabled, you can still preserve and analyze audit trails which might come in handy for forensics investigation of past user activity.

Keeping disabled accounts comes with liability. According to SANS "*Attackers frequently discover and exploit legitimate but inactive user accounts to impersonate legitimate users, thereby making discovery of attacker behavior difficult for network watchers*". That is why it is paramount to keep a close eye on disabled user accounts. In particular, when reviewing a list of currently disabled user accounts you might also want to check the following for each disabled user account:

1. When was the last time the account successfully logged on to the Active Directory managed network?
2. When was the last failed attempt to logon due to bad password?

So, let's see how we can use familiar PowerShell tools to show this information. At first, this command might seem what we are looking for:

Search-ADAccount –AccountDisabled

According to its description, it returns a list of disabled accounts. There are two gotchas here. First, it does not differentiate by account type. So, you will get computer accounts as well. Second, it does not let you specify other attributes you want to see for disabled accounts.

Luckily, just like in the previous example with locked out user accounts, we can use the Get-ADUser cmdlet to address these shortcomings:

*Get-ADUser -Filter * -Properties SamAccountName, LastBadPasswordAttempt, LastLogonDate, Enabled| Where{$_.Enabled-eq $false} | Select-Object SamAccountName,LastBadPasswordAttempt,LastLogonDate*

This command will return a full list of disabled user accounts and a pair of attributes that will effectively tell you when was the last time somebody attempted to use those accounts.

But what if you want to know what exactly these accounts were used for in case they got compromised? Unfortunately, this information is not easy to obtain without appropriate auditing tools in place. If you want to have this kind of insurance in place you might want to take a look at SecureHero Logon Reporter and File System Auditor that will audit and report on access of all users in Active Directory.

It goes without saying that performing this review on a regular basis is the least you can do to safeguard your network from insider attacks and improve overall security posture.

#9 Users that have not logged on in the last X days

One way to detect inactive user accounts is to examine when was the last time they logged on to the Active Directory domain. Without further ado, let's look at the PowerShell snippet that returns all user accounts in the domain that have not logged on during the last 30 days:

```
$30Days = (get-date).adddays(-30)
Get-ADUser -SearchBase "DC=TORONTO,DC=LOCAL" -filter {lastlogondate -notlike "*" -OR lastlogondate -le $30days} -Properties lastlogondate | Select-Object name, lastlogondate
```

The first line subtracts 30 days from the current moment and saves the resulting date and time into the *30days* variable.

The second line uses familiar Get-ADUser cmdlet to return all user objects matching the specified filter. In this case, we are looking at the value of the lastlogondate attribute and pick up only those users that have either logged on longer than 30 days ago or users that have not logged on at all. The latter is determined based on the lastlogondate attribute being empty.

This PowerShell snippet exhibits a subtle difference that can significantly speed up the results. Unlike with the example about Disabled User Accounts, it takes advantage of a more specific filter right in one of its arguments. This way, the underlying LDAP query will return only a subset of user objects as opposed to returning all user objects and then filtering them out on the client side.

Lastlogondate is a tricky attribute. It is a non-replicated attribute and it is only updated on a domain controller that actually authenticates a user. That means that for the most accurate results the script has to examine the value of this attribute on all domain controllers that might have possibly authenticated users. And this is more than just a few lines of PS code…

Microsoft has tried to make this process simpler having introduced the lastlogontimestamp attribute. Although this one is a replicated attribute, it might take up to 14 days for it to get updated, so be careful choosing the time interval that renders user accounts inactive.

If you want a hassle free solution that provides the most accurate data, look at SecureHero Logon Reporter. Logon Reporter hides the complexity of pulling the actual information about last logon time of Active Directory users including the source computer the logon was initiated from.

#10 Users with old passwords

Checking when Active Directory users logged on last should not be the only criteria if you want to pinpoint stale user accounts reliably. Another method that can help with this judgment involves analyzing when user accounts had their password changed.
Indeed, in well-managed Active Directory environment password policies are normally set to anything between 30 and 90 days until user passwords expire. If passwords are expired, users can't logon. Isn't it a sign that such user accounts are no longer used?

Let's take a look at the following PowerShell code snippet:

```
$90Days = (get-date).adddays(-90)
Get-ADUser -SearchBase "DC=TORONTO,DC=LOCAL" -filter {passwordlastset -le $90days}
-Properties  passwordlastset | Select-Object name,  passwordlastset
```

Just like with the lastlogondate example, we search for all user accounts that had their passwordlastset attribute updated longer than 90 days ago. This should pick up a lion share of users with old passwords.

There are a couple of exceptions though. If passwordlastset equals 0, then user account password is expired and user must change password at next logon. If passwordlastset equals -1, then user account password is set to never expire. Depending on the intended use of such accounts you might want to include them into a list of stale user accounts or not.

Combined analysis of lastlogondate and passwordlastset attributes will yield a more reliable conclusion about the status of user accounts and whether or not they should be disabled or removed.

Conclusion

Active Directory hygiene

Your Active Directory is only as reliable and secure as you know it. Daily Active Directory reporting on users and groups is paramount.

Native tools

Built-in operating system tools and PowerShell give you a piece of mind but they are limited, time consuming and unreliable

Focused solutions

Purpose-built solutions automate user and group reporting for a well-managed, secure and optimized Active Directory

Active Directory reporting is a part of IT hygiene

Keeping tabs on user and group accounts is one such thing that like brushing teeth has to be done daily. Identifying toxic or unnecessary groups, verifying locked and disabled user accounts, analyzing duplicate group membership are activities that have to be routinely performed by administrators to prevent Active Directory from spinning out of control and affecting the organizational bottom line.

Native tools can give you a jump start

Equipped with native tools and knowledge of PowerShell Active Directory administrators can probably build most of the reports they need. However, organizations that take on this route often inherit the risk of relying on inaccurate data, making untimely decisions and ultimately missing out on strategic business priorities they could have dedicated their time to instead.

Focused solutions for faster results and increased reliability

For faster results and better informed decisions organizations should look for focused and purpose built solutions such as SecureHero Group Reporter. Group Reporter is the only solution that can hide the complexity and intricacies of Active Directory reporting and empower IT departments to build a better managed, secure and reliable Active Directory.

About SecureHero LLC

We keep your Active Directory growth under control.

SecureHero caters to IT administrators challenged to manage corporate networks based on Microsoft Active Directory. We understand the critical importance of Active Directory and we know how to maintain its security, reliability and performance to satisfy the requirements of the growing business.

We keep Active Directory growth under control. Like traditional disk defragmentation tools that reclaim unused space and optimize performance of file systems, we “defrag” Active Directory by removing unused objects, cleaning up stale permissions and reconciling group membership. We do that based on the actual usage of resources, groups and permissions to avoid system downtime and provide intelligence for confident decision making.

Find out more about SecureHero and its innovative products at www.securehero.com