# Kerberos Delegation: A Wrap Up

csandker.io/2020/02/10/KerberosDelegationAWrapUp.html

February 10, 2020

10 Feb 2020 (Last Updated: 20 Aug 2021)

*There many good posts about Kerberos Delegation, explaining the backgrounds, underlying concepts and terminology of it. This post is is a wrap-up of Kerberos Delegation and unlike other posts it will not dig into where it came from and lay out surrounding concepts.*
*Most important: This wrap-up assume the reader is aware of the environment that Kerberos Delegation takes place in.*
**If you need a fresh-up on this read through Kerberos Authentication: A Wrap Up**

Delegation allows a **server** application to **impersonate a client** when the server connects to other network resources.
In other words: Delegation specifies the client's action to authorize a server in order to allow this server to impersonate itself (the client).

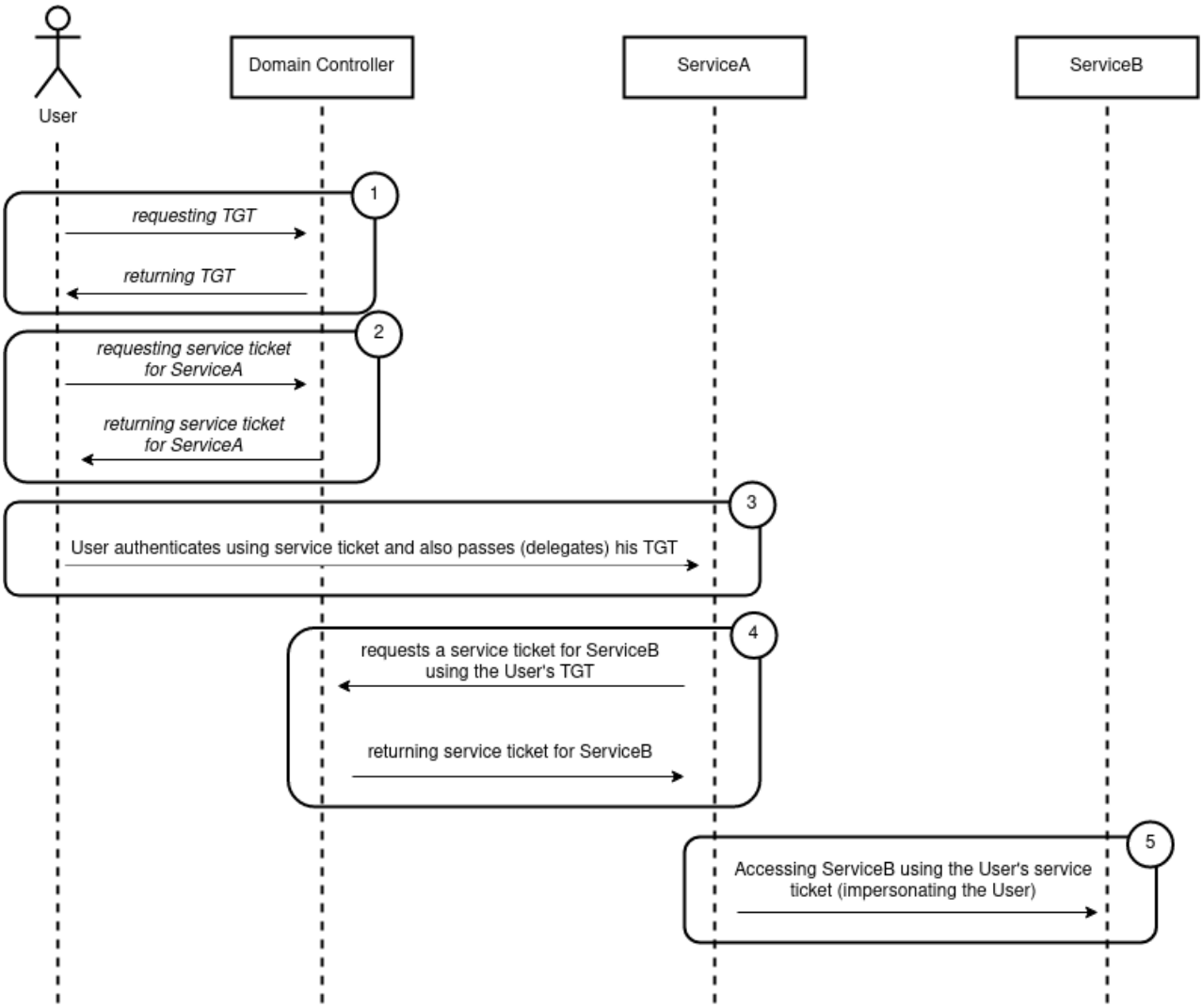
There are 3 Types of Kerberos Delegation:

- Unconstrained
- Constrained
- Resource Based Constrained

## Unconstrained Delegation

Unconstrained Delegation allows a server to impersonate a client against any service the server wishes to. The client gives the server a wildcard allowance for impersonation. Server and client are conceptual terms to stress the idea of delegation, read this as the server being a User account and the client being some other user account.

In technical terms this means a server authorized with unconstrained delegation receives the krbtgt service tickets (TGTs) from connecting clients and uses the clients TGT to create new service tickets (requested from the TGS) for any service on behalf of the connecting client.
**»Take away»:** A connecting client sends his TGT to a server authorized for unconstrained delegation

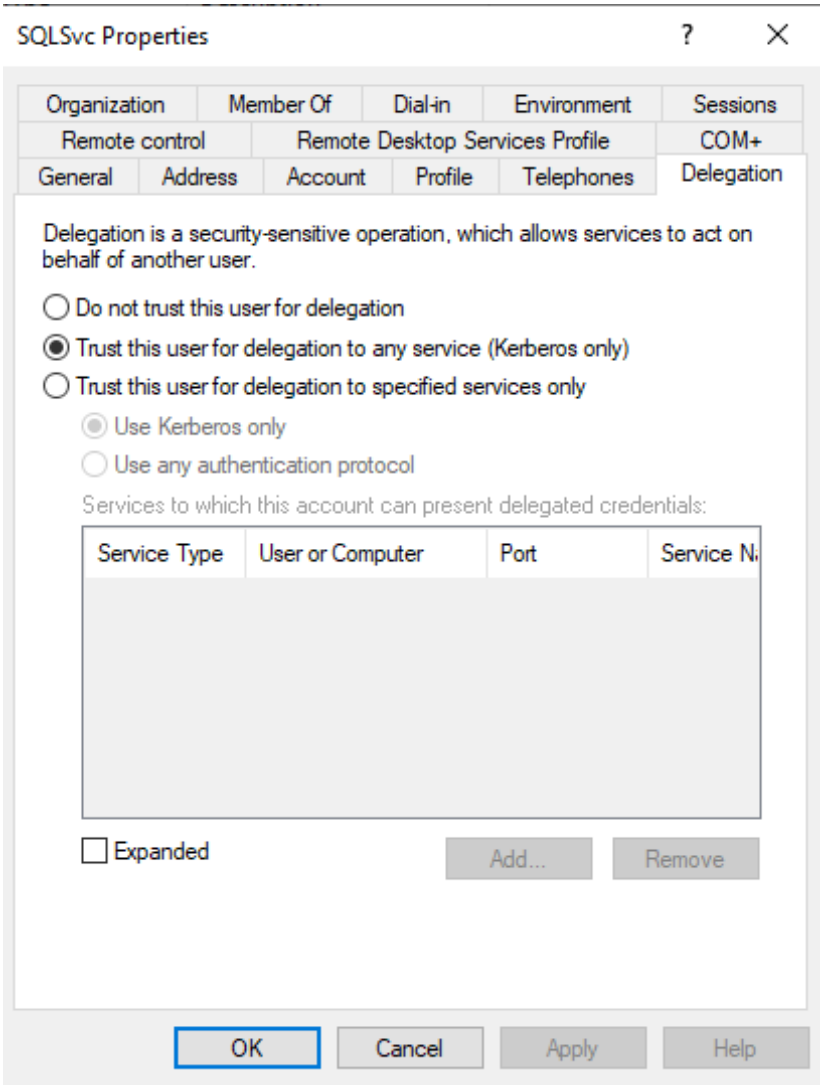



**Who is Authorized For Unconstrained Delegation?**

All Principals that have the **TRUSTED_FOR_DELEGATION** UserAccountControl Attribute
Per default that is only the SYSTEM/Computer-Account on Domain Controller(s)
This UserAccountControl Attribute can be queried using Powershell:

```
([adsisearcher]'(userAccountControl:1.2.840.113556.1.4.803:=524288)').FindAll()
```

The user properties for a user that is allowed for unconstrained delegation is shown below:




**Who is allowed to set the attribute 'TRUSTED_FOR_DELEGATION' for other users?**
All Principals that hold the **SeEnableDelegationPrivilege** privilege.
→ On domain controllers, this right is assigned to the Administrators group by default.

The permission to set the TRUSTED_FOR_DELEGATION attribute for other users is controlled by a user privilege.
User privileges are access rights granted to principals on a local computer (not domain wide access rights). Each system has a database to store which principal has which privileges.
A list of Privileges (user rights) can be found here: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/user-rights-assignment

The most efficient way to spread privileges rights within a domain is by the use of Group Policy Objects (GPO) (could also be set locally using secpol.msc)

**How can it be abused by an attacker?**

First step is to identify if any principal has the **TRUSTED_FOR_DELEGATION** UserAccountControl attribute set (see ADSISearcher snippet above), let' say this is *UserA*.

If there is a user set with TRUSTED_FOR_DELEGATION (*UserA*), an attacker can steal another user's TGT - let's call that user *UserB* - if:

1. The attacker (*UserA*) can make the victim (*UserB*) connect to a service run by *UserA* (who has the TRUSTED_FOR_DELEGATION attribute set)
2. The attacker (*UserA*) can extract *UserB*'s TGT from the computer where *UserA* run his/her service and that *UserB* connected to.

For 1. @tifkin_'s amazing SpoolSample.exe (also known for being the exploit for "The PrinterBug") can be used to trick an arbitrary ComputerAccount (including the DC-Computer account) to authenticate against a chosen target machine:

```
.\SpoolSample.exe <PDC> <Target-Computer>
```

For 2. Easiest way to extract a TGT is by having local admin access to the targeted Computer, and then using either

- Rubeus: `Rubeus.exe monitor`
- Mimikatz: `mimikatz # sekurlsa::tickets /export`

The extracted TGT of the targeted principal (*UserB*) can then in turn be used to connect to arbitrary other services on behalf of the victim, *UserB*.

**An attack chain** could look as follows:
Assume you found found a computer account (ComputerA$) is set with the TRUSTED_FOR_DELEGATION attribute:

> - You managed to compromise ComputerA$
> - You then use SpoolSample.exe to force the Primary Domain Controller (computername: *PDC$*) to connect back to ComputerA$
> - After you imported the TGT of the *PDC$* into your current process you can use that to run a DCSync attack
>   – Mimikatz: `mimikatz # lsadump::dcsync /domain:<DOMAIN.FQDN> /user:X`
>   *X could be any user, e.g. the default (RID 500) admin of the DC, or the krbtgt user*
> - Let's say you set the username (X) to be the default admin (RID 500) account for the *PDC$*
>   This will return you the *PDC$*'s default admin's NTLM hash
>   This NTLM hash can in turn be used to request a TGT of that default admin user
>   – Rubeus: `Rubeus.exe asktgt /user:DA01 /rc4:<NTLM-Hash-From-RID-500-Admin> /ptt`
>   In this command you not only requested the TGT, but also included it in you current process with the **/ptt**-flag
>   Now that you included the *PDC$* default admin's TGT in your current process, you're basically that user now (within this process).
>   You can now use your new user access for example by using PsExec to add a new user to the local admin group on the *PDC$*
> - `C:\> PsExec.exe \\PDC$.FQDN localgroup Administrators <DOMAIN>\<YourUser> /add`

## Constrained Delegation (S4U2Proxy)

Constrained Delegation allows a server to impersonate a client against defined, specified service(s). Server and client are conceptual terms to stress the idea of delegation, read this as the server being a User account and the client being some other user account.

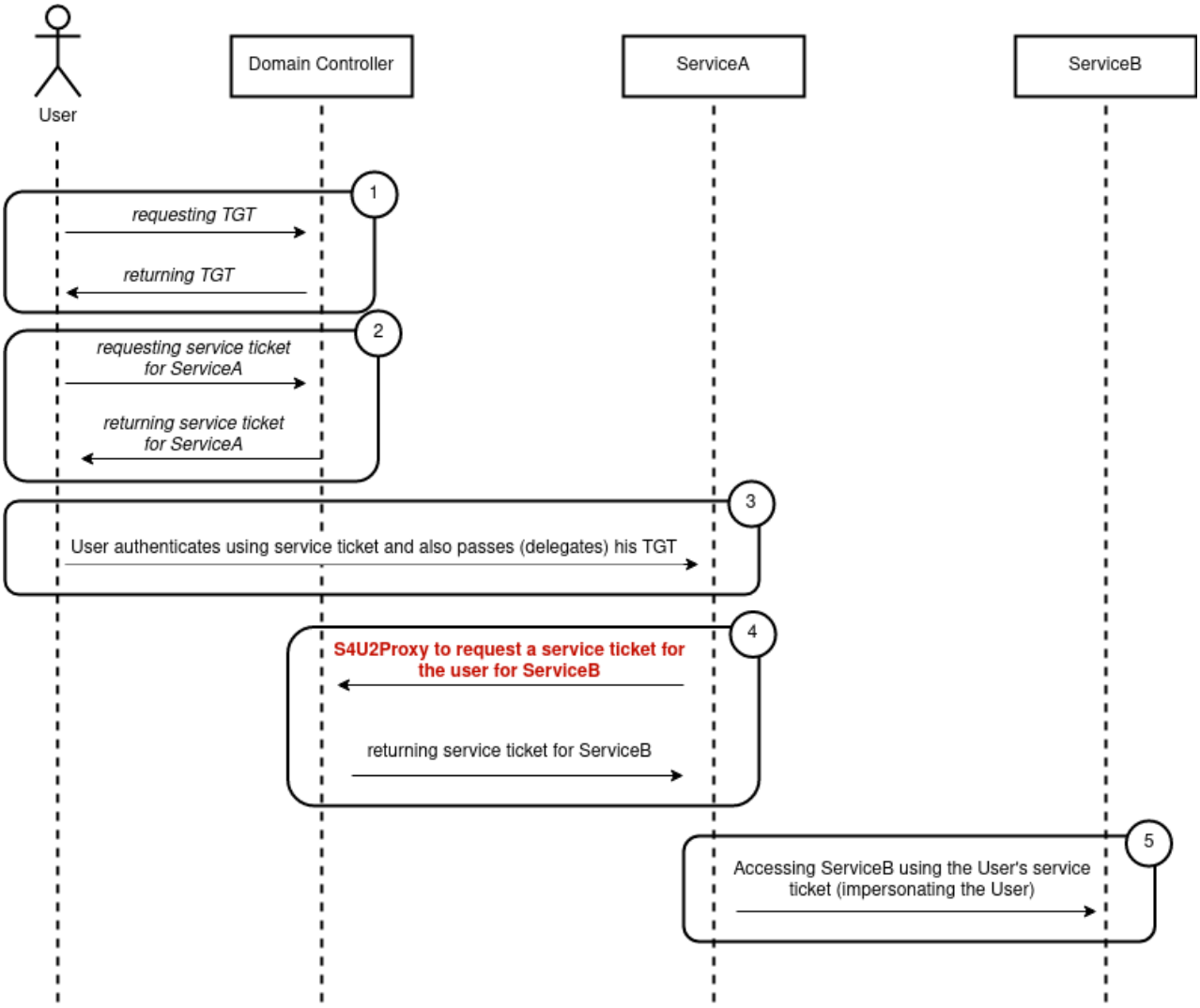
Let's say a user (*UserA*) connects to a service (let's call it *ServiceA*) using his service ticket for *ServiceA*.
Now *ServiceA* needs to connect to another service (let's call that *ServiceB*) to do it's task, but it needs the permissions of *UserA* to do that.
In short: *ServiceA* wants to impersonate *UserA* and delegates his/her access permissions to *ServiceB*.

This is where Constrained Delegation comes into play. Instead of allowing *ServiceA* to completely impersonating the user (against all services in the domain), *ServiceA* should only be allowed to impersonate *UserA* against *ServiceB*.

The way this is done is as follows:

- *UserA* connects to *ServiceA* using his/her service ticket for *ServiceA*

- *ServiceA* uses a special service, called **S4U2Proxy**, to request a service ticket for *ServiceB* on behalf of *UserA*
- The Kerberos Distribution Center (**KDC**) (which in most cases is part of the DomainController) return a service ticket for *ServiceB* (this service tiicket is issued for *UserA*, hence allowing *ServiceA* to impersonate *UserA* towards *ServiceB*)
- *ServiceA* then connects to *ServiceB* with the returned service ticket for *ServiceB*



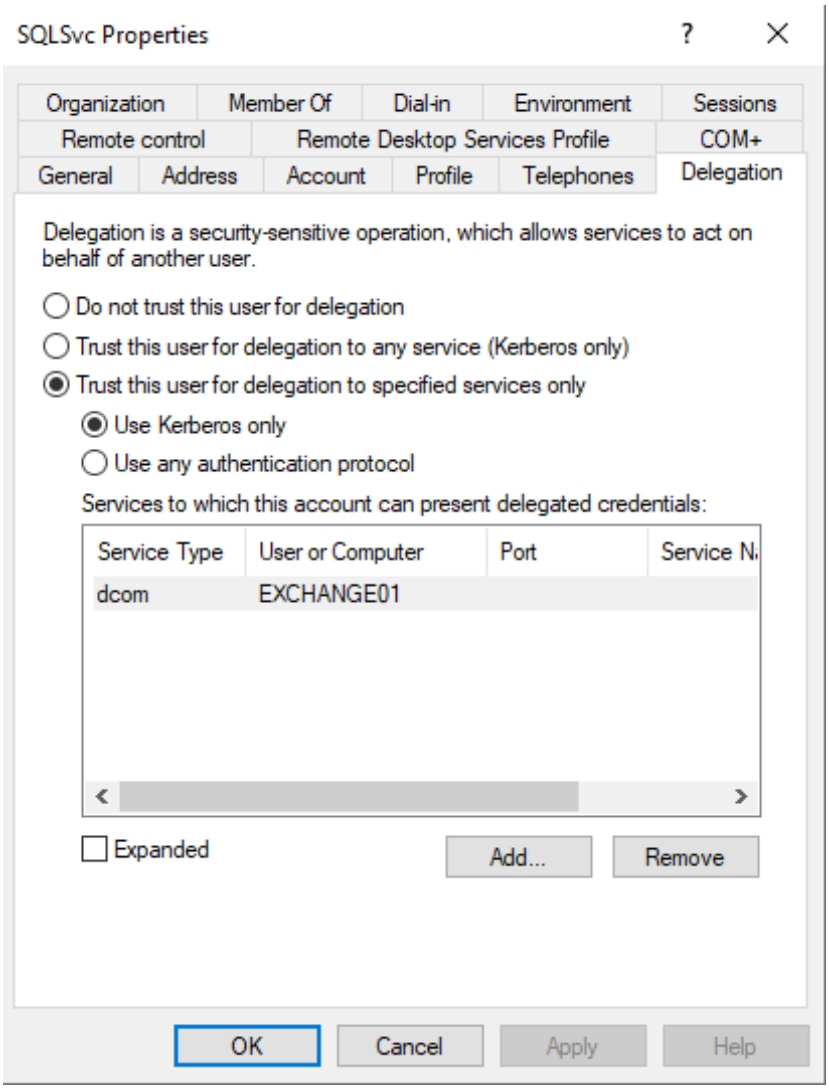

*(Note in red the use of **S4U2Proxy** instead of using the user's TGT, which would require **Unconstrained Delegation**)*

In this case *ServiceA* makes use of the **S4U2Proxy** service in order to request a service for *ServiceB* on behalf of *UserA*.
The S4U2Proxy services requires that the caller (*ServiceA*) presents a valid service ticket for the user that should be impersonated (*UserA*).
This is meant to ensure that the user exists and has really connected to *ServiceA* (otherwise ServiceA would not have the user's service ticket). There might occur cases where ServiceA can't present a service ticket for the User (because *UserA* hadn't connected via Kerberos), for those cases the **Protocol Transition** with the **S4U2Self** service has been created (we'll get to this in a moment further down below).

Note that the service ticket provided in the S4U2Proxy request must have the **FORWARDABLE** flag set.
The FORWARDABLE flag is never set for accounts that are configured as “sensitive for delegation” (the **USER_NOT_DELEGATED** attribute is set to true) or for members of the “Protected Users” group.

In order for this to work *ServiceA* must be authorized for Constrained Delegation (more on this below) and must be specified with a defined set of user accounts that can impersonate (meaning where it can delegate a user authentication to).

**An Example** account configuration could look like this:
In the below image ServiceA would be the user *SQLSvc* and ServiceB would be *EXCHANGE01*.
Note here that ServiceA can not impersonate a user against all services of *EXCHANGE01* (ServiceB), but only against the *DCOM* service.
**»Take away»:** Constrained Delegation is different to Unconstrained Delegation by only allowing impersonation for specified services.




**Who is Authorized For Constrained Delegation?**
All Principals that have the **ms-DS-Allowed-To-Delegate-To** object attribute.
→ Per default no principal has this attribute

The following Powershell snippet can be used to find all user's that have the 'ms-DS-Allowed-To-Delegate-To' object attribute:

```
([adsisearcher]"(msds-allowedtodelegateto=*)").FindAll() | %{$_.Properties['msds-allowedtodelegateto']}
```

**Who is allowed to grant set the ms-DS-Allowed-To-Delegate-To Attribtue for other users?**
All principals that hold the SeEnableDelegationPrivilege privilege.
→ On domain controllers, this right is assigned to the Administrators group by default.

## Constrained Delegation (S4U2Self)

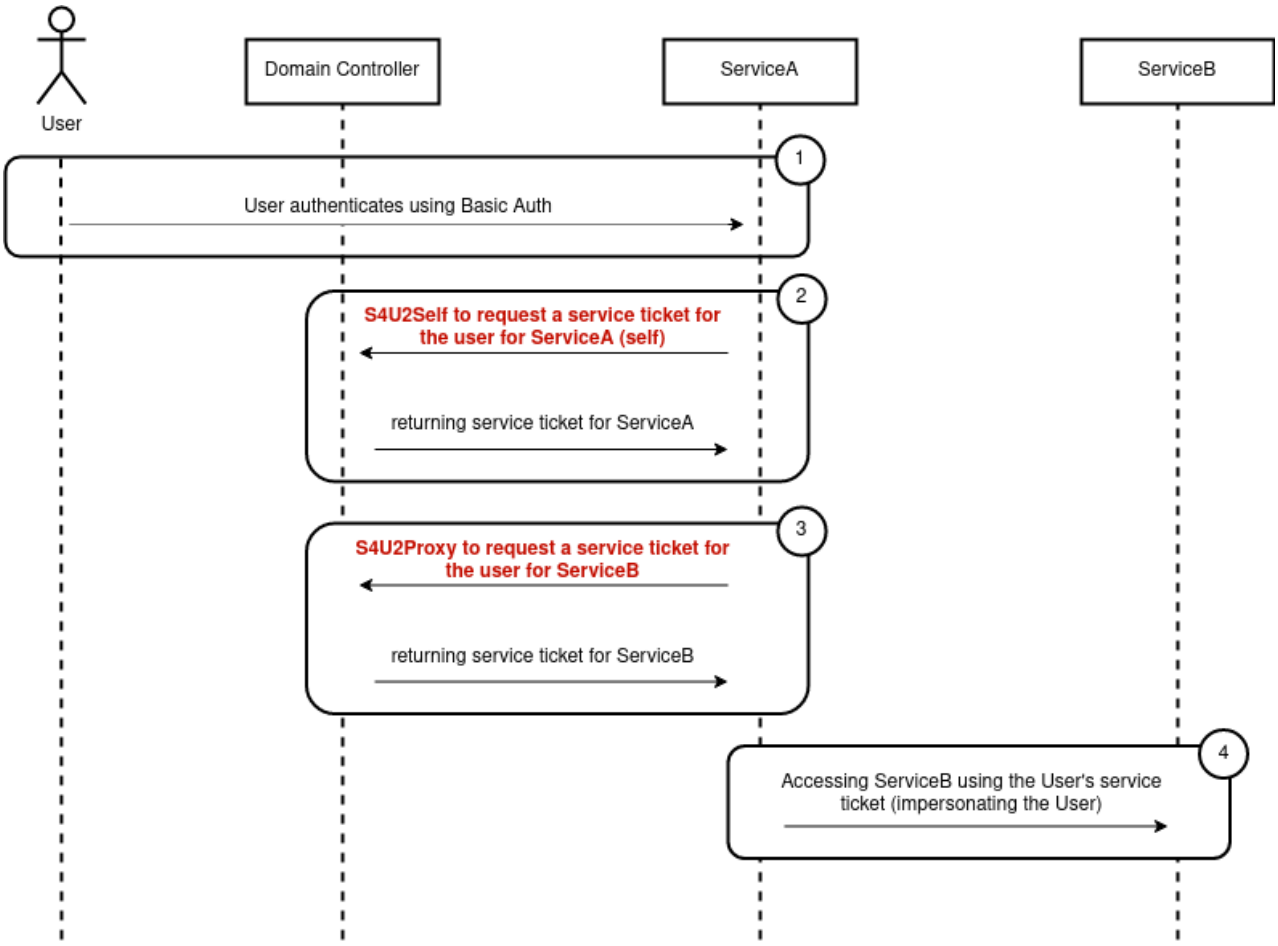
In the "usual" Constrained Delegation process the S4U2Proxy service is required to obtain a service ticket for the user that a service wishes to impersonate. This is meant to ensure that this user exists and has really authenticated to the service that now tries to impersonate him/her.

Let's say *UserA* connected to *ServiceA* and *ServiceA* is allowed to delegate (ms-DS-Allowed-To-Delegate-To) to *ServiceB*.
As *ServiceA* is required to present *UserA*'s service ticket to the S4U2Proxy service, *UserA* is required to connect to *ServiceA* through Kerberos (otherwise there would be no service ticket).
What if the user can't connect to *ServiceA* via Kerberos, but wants to use other authentication mechanisms (NTLM, BasicAuth, …)
This is where **Protocol Transition** and the **S4U2Self** services come into play.

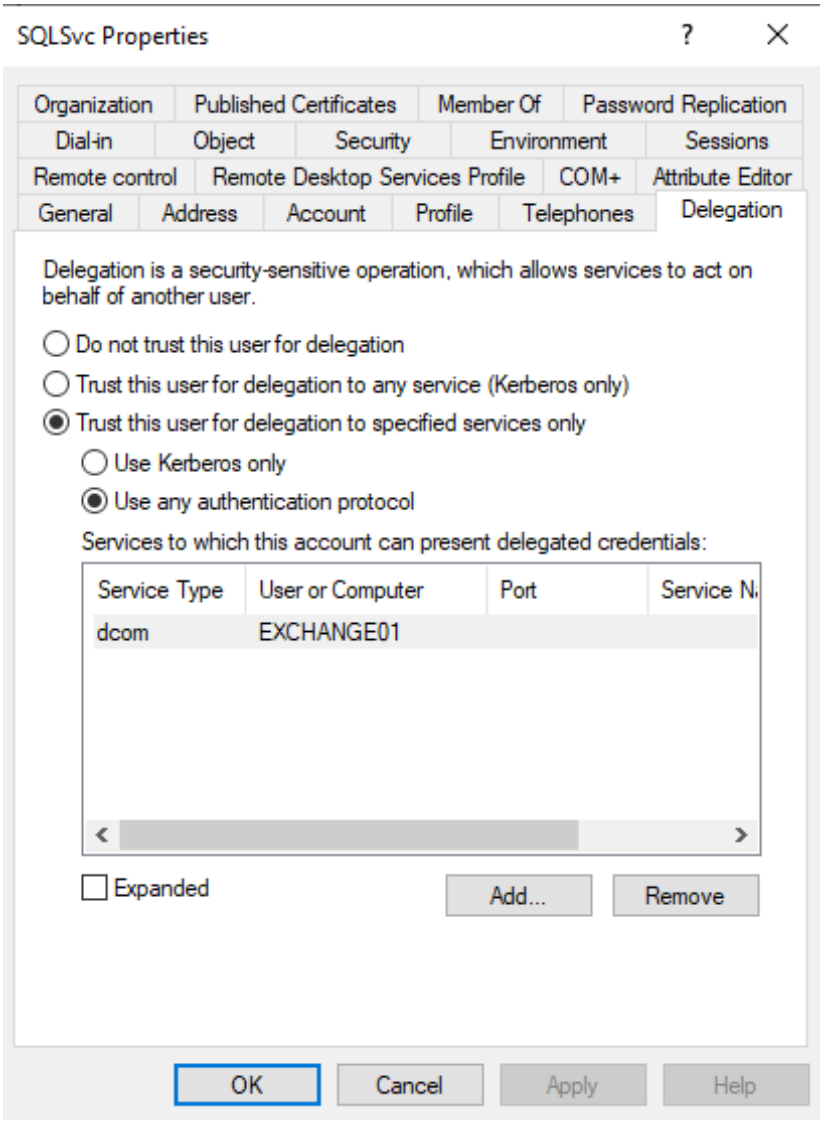
In the cases where *UserA* does not authenticate to *ServiceA* via Kerberos, *ServiceA* can use the S4U2Self service to create a servuce ticket for *UserA*, which it then in turn can use to run S4U2Proxy as usual.
→ **This means ServiceA can request a service ticket for/to itself for any arbitrary user.**

As this is sensitive operation Microsoft created an additional UserAccountControl Attribute **'TRUSTED_TO_AUTH_FOR_DELEGATION'**.
Only principals that hold this UserAccountControl attribute can request service ticket to/for itself for any user from the KDC.

Assume *UserA* connected to *ServiceA* via NTLM, the authentication flow then looks like the following (making use of S4U2Self)




An Example account configuration for Unconstrained Delegation with Protocol Transition could look like this:




**Who is Authorized To Request Service Tickets For Arbitrary Users To Themselfs ?**
All principals that have the **TRUSTED_TO_AUTH_FOR_DELEGATION** UserAccountControl attribute.
→ Per Default no user has this attribute

The following Powershell snippet can be used to find all user's that have the 'TRUSTED_TO_AUTH_FOR_DELEGATION' UserAccountControl attribute:

```
([adsisearcher]'(userAccountControl:1.2.840.113556.1.4.803:=16777216)').FindAll()
```

The TRUSTED_TO_AUTH_FOR_DELEGATION UserAccountControl attribute should only come along with the "right/permission" for Constrained Delegation (ms-DS-Allowed-To-Delegate-To).
The following snippet list all user accounts that are TRUSTED_TO_AUTH_FOR_DELEGATION, but not ms-DS-Allowed-To-Delegate-To:

```
([adsisearcher]'(&(!msds-allowedtodelegateto=*)
(userAccountControl:1.2.840.113556.1.4.80
3:=16777216))').FindAll()
```

## Resource Based Constrained Delegation

For the following view on Resource Based Constrained Delegation let's assume the following:

- *UserA* runs *ServiceA* (let's say a WebServer) on *HostA*
- *UserB* runs *ServiceB* (let's say a SQLServer) on *HostB*
- *ServiceA* wants to delegate to *ServiceB*

That means *ServiceA* wants to impersonate a user connecting to it to *ServiceB*, to run tasks on *ServiceB* as this connecting User.

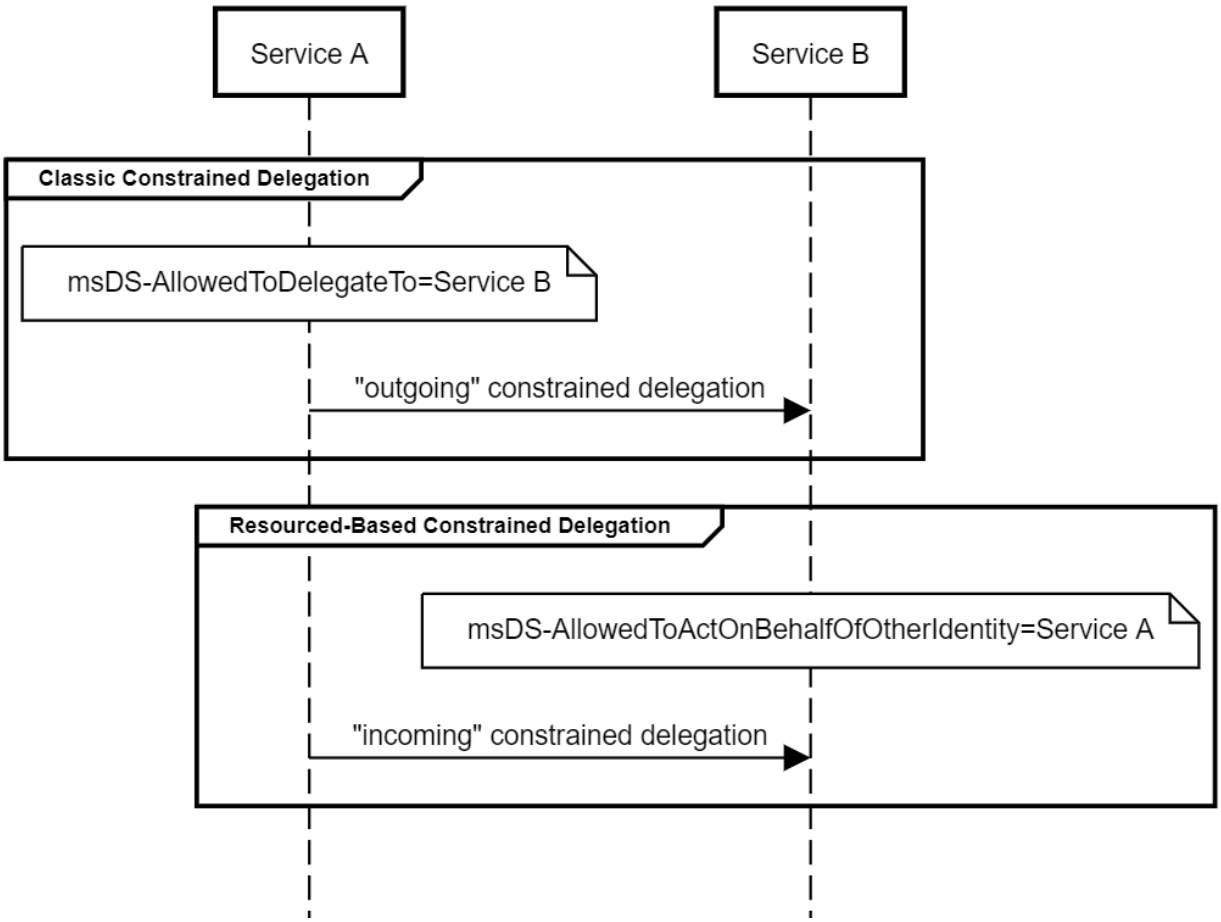
In the "conventional" Constrained Delegation an administrator could configure that one defined user (*UserA*) is allowed to delegate to a defined service (*ServiceB*) on a defined host (*HostB*).
In technical terms that means:
A user account holding the **SeEnableDelegationPrivilege** can set **ms-DS-Allowed-To-Delegate-To** account attribute of *UserA* to be 'ServiceB\HostB'.

Let's refer to this "conventional" Constrained based delegation as "Outbound Delegation" for a second.
This makes sense as the granted authorization by the admin points away from the resource (from *UserA* to *ServiceB**HostB*).

By introducing Resource Based Constrained Delegation Microsoft wanted the delegation concept to be more flexible and allow a resource owner (that is a User) to decide who is allowed to delegate to him/her.
To facilitate that Microsoft created the new account attribute **msDS-AllowedToActOnBehalfOfOtherIdentity**.

Instead of an admin (a user holding SeEnableDelegationPrivilege) setting 'ms-DS-Allowed-To-Delegate-To' on *UserA*, any user with write permissions to *UserB*'s 'msDS-AllowedToActOnBehalfOfOtherIdentity' account attribute can set this attribute to a bitmask representing *UserA*.
→ As *UserB* has write permissions to his/her own account attributes, *UserB* can allow *UserA* to delegate to his/her services without the need of an administrator.

If *UserB*'s 'msDS-AllowedToActOnBehalfOfOtherIdentity' account attribute is set to *UserA*, *UserA* can request a service ticket for any service run by *UserB*.
→ **Note in "conventional" constraint delegation UserA would only be allowed to request a service ticket for ServiceB run by UserB**




**Who is Authorized For Resource Based Constrained Delegation?**
Every User that has his/her identify placed in the 'msDS-AllowedToActOnBehalfOfOtherIdentity' attribute of the user that is targeted for delegation.

You can look for Resource Based Constrained Delegation settings using the following Powershell snippet:

```
PS C:\Users\> ([adsisearcher]"(msds-
AllowedToActOnBehalfOfOtherIdentity=*)").FindAll()

Path                                                    Properties
----                                                    ----------
LDAP://CN=EXCHANGE01,CN=Computers,DC=Lab,DC=local {logoncount, codepage,
objectcategory, iscriticalsystemob.

## The EXCHANGE01 server has the attribtue set (see PS command below)
```

**Who is Authorized To Configure Resource Constrained Delegation?**
As Resource Based Constrained Delegation is controlled by a user account attribute, every user that has write access (*GenericAll*/*GenericWrite*/*WriteDacl*) to that attribute of a target User can authorize any user to delegate to the services run by that target User.
→ By default each user has access writes to it's own 'msDS-AllowedToActOnBehalfOfOtherIdentity' attribute
→ Every user can decide on it's own who can delegate to his/her services

Note: Resource Based Constrained Delegation can not be set via a GUI application, instead Powershell can be used:

```
PS C:\> Set-ADComputer 'EXCHANGE01' -PrincipalsAllowedToDelegateToAccount (Get-
ADComputer 'Client01')
```

## Other Posts