Kerberos (III): How does delegation work?

tarlogic.com/blog/kerberos-iii-how-does-delegation-work

February 17, 2020

Introduction to Kerberos delegation

There are several kinds of delegation implemented by using the Kerberos protocol on Windows and Linux servers. Basically, delegation allows a service to impersonate the client user to interact with a second service, with the privileges and permissions of the client itself. The flavors of delegation are the following:

- **Unconstrained delegation**
- **Constrained delegation**
- **RBCD** (Resource Based Constrained Delegation)

In this article, we will focus on understand how the different kinds of delegation work, including some special cases. Additionally, some scenarios where it could be possible to take advantage of these mechanisms in order to leverage privilege escalation or set persistence in the domain will be introduced. Before starting with the explanations, I will assume that you already understand Kerberos' basic concepts. However, if expressions like TGT, TGS, KDC or Golden ticket sound strange to you, you should definitely check the article "How does Kerberos works?" or any related Kerberos' introduction. Moreover, this is a tricky topic, with several details to consider, so do not be discouraged in case you do not fully understand everything at first sight or you only get a general idea, that is fine, you can review the content as many times as you want. In the same way, if you have any question, do not hesitate and let a comment down below. Now, onto the topic.

Services, Users and Computers

First of all, since Kerberos provides a way for users to authenticate against services of the domain and delegation is about users whom interact with services, which in turn interact with other services, the first logical step is to provide a definition of service, user and how they relate to each other. It is important to be familiar with these topics in order not to get lost in the later explanations of the delegation mechanisms.

Users

An user is an agent which is represented by an user account (or a subclass of it) in Active Directory. In an Active Directory domain different types of user accounts can be found:

It is important to understand that, from the Active Directory perspective, **computers are users**, more specifically, computers are a subclass of users.

Services

A service:

- is identified by a SPN (Service Principal Name) in Active Directory, which indicates the service name and class, the owner and the host computer.
- is executed in a computer (the host of the service) as a process. However, it is not necessary that the service is running in order to get a TGS for it. Furthermore, as a process, it must be understood that the administrators of the host can get total access to the memory of the service and the tickets handled by this one.
- is executed in the context of a domain user (the owner of the service). Usually, services run in the context of the computer account of its host, but this is not always true.
- can be used by many users of the domain (For instance, Kerberos, LDAP, SMB or MSSQL), and any domain user can get a TGS for any service in the domain.

The most important part here is to understand that services (as any process) are running in the context of a user account, and therefore they have the privileges and permissions of that user. On the opposite side, domain users can own services. The SPN's of the services owned by an user are stored in the attribute *ServicePrincipalName* of that account. Additionally, it should be noted that in order to be able to add a SPN to a user account, the permission *Validated-SPN* is required on that account. This permission is not granted for the user account itself by default (except in computer accounts), so usually a Domain Admin or similar role is required to modify the SPN's of a user.

Service in delegation

In relation with Kerberos delegation. Due to a service being executed in the context of its owner user:

- The service can perform delegation only if its owner has permission to do it. In other words, if a user has delegation capabilities, all its services (and processes) have delegation capabilities.
- When the service talks with the KDC, it has the identity of its owner user. In fact, KDC only worries about the user who is talking to, not the process. This means that any process belonging to the same user can perform the same actions in Kerberos, regardless of whether it is a service or not.

Anti-Delegation Measures

Before explaining any specific type of delegation, it should be known that 2 methods exists to avoid any kind delegation for a specific user account:

- Set the *NotDelegated* (or ADS_UF_NOT_DELEGATED) flag, stored in the User-Account-Control attribute of the user account. This flag disables any kind of delegation for the account. By default, no domain account has this flag set.

- Add the user to the group *Protected Users*, which was introduced in Windows Server 2012 R2. By default, this group has no members. Additionally, the members of *Protected Users* are unable to:
    - Use NTLM authentication.
    - Use DES or RC4 encryption in Kerberos pre-authentication.
    - **Be delegated with any kind of Kerberos delegation.**
    - Renew the Kerberos TGT's beyond the initial four-hour lifetime.

In the subsequent sections, it will be assumed that delegation will not work for a user protected against delegation, thus examples will avoid this check for the sake of clarity.

Unconstrained delegation

First of all, Unconstrained Delegation is the simplest type of delegation, which allows a service to impersonate any user that was authenticated against it without limitations. In this sort of delegation, the service acquires a valid TGT for the client user, which is equivalent to become that user in Kerberos' world (and therefore on the domain).

But, how does the service get a TGT for the client user?

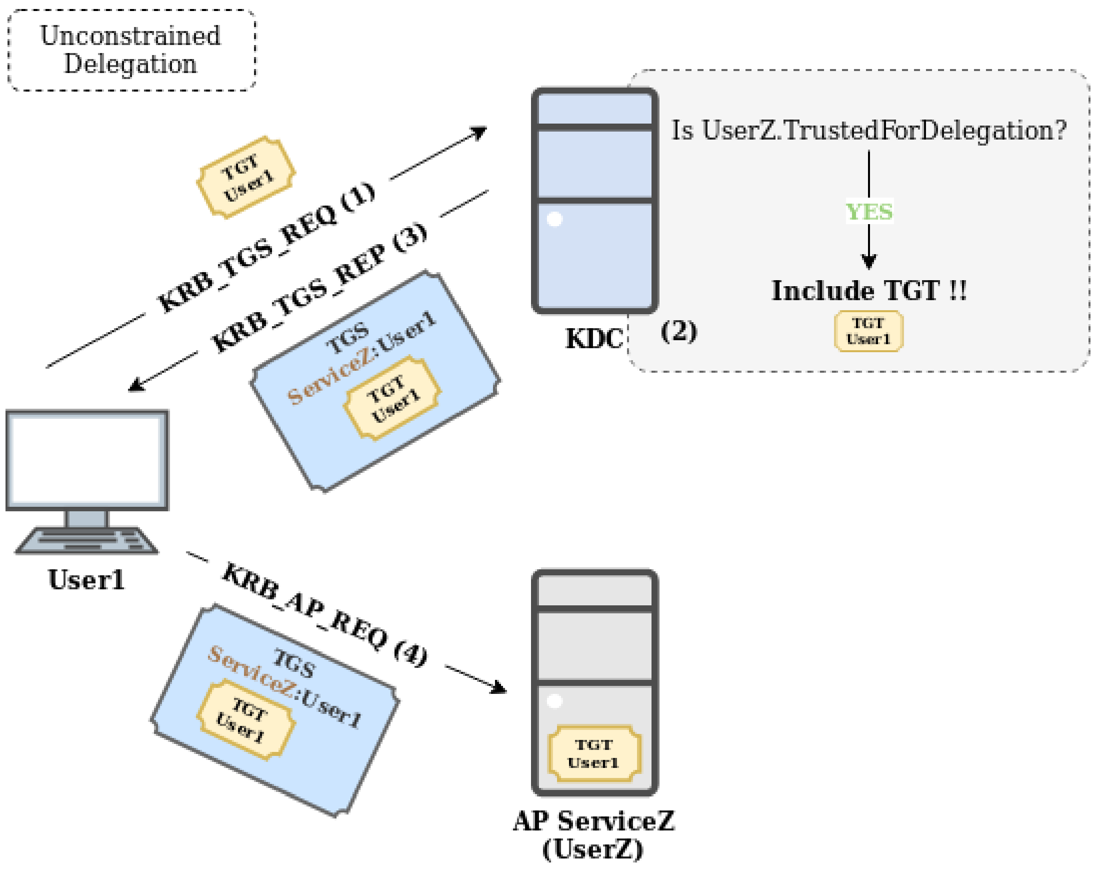
The client sends the TGT to the service. When the client user requests a TGS for a service which has Unconstrained Delegation active, then the KDC includes a TGT (of the client user) inside the TGS. Specifically, it is included in the part of the ticket encrypted with the service owner key. Thus, upon receiving the TGS, it can access the client TGT.

But, how does the KDC know when to include the TGT inside the TGS?

KDC includes the TGT in case the *TrustedForDelegation* (or ADS_UF_TRUSTED_FOR_DELEGATION) flag is set for the owner of the target service. This flag is stored in the User-Account-Control attribute of Active Directory user accounts. Moreover, it should be noted that in order to modify the TrustedForDelegation flag of a user, it is necessary to have the privilege SeEnableDelegationPrivilege on a domain controller. Anyway, if the target user of delegation is protected against delegation because of *Protected Users* group or *NotDelegated* flag, delegation will not work.

OK, Can you give me an example of Unconstrained Delegation?

The following image shows the process followed in Unconstrained Delegation (for sake of clarity it is assumed that User1 is not protected against delegation by *Protected Users* or *NotDelegated*, otherwise the delegation would fail):



Unconstrained Delegation

In this situation:

1. *User1* requests a TGS for *ServiceZ*, of *UserZ*.
2. The KDC checks if *UserZ* has the *TrustedForDelegation* flag set (Yes).
3. The KDC includes a TGT of *User1* inside the TGS for *ServiceZ*.
4. *ServiceZ* receives the TGS with the TGT of *User1* included and stores it for later use.

OK, but how could a pentester take advantage of this situation?

There are a few possible scenarios:

- If a pentester is able to compromise a computer which is hosting services with Unconstrained Delegation, there is a good chance that TGT’s can be found for the clients of those services.
- If a pentester is able to compromise the account of a user which has Unconstrained Delegation permissions, it could be possible to harvest TGT’s from the services belonging to that user.
- In any of the previous scenarios, if the pentester can persuade a privileged account to interact with one of the “controlled” services, then it would be possible to steal its TGT and compromise the domain.
- In case of having the control of a domain, a way of persistence could be granting Unconstrained Delegation to a set of users that the pentester would control.

Constrained Delegation and RBCD

In addition to Unconstrained, there are 2 more kinds of delegation:

- **Constrained Delegation**
- **RBCD** (Resource Based Constrained Delegation) (Introduced in Windows Server 2012)

In any of these types, the delegation is constrained to only some whitelisted third-party services.

OK, how can Kerberos create a whitelist of services?

Kerberos is unable by itself to create special tickets to implement delegation for specific groups of services. For this reason Microsoft implemented 2 Kerberos extensions that allow it to obtain this desired behavior:

- **S4U2Proxy** (Service for User to Proxy)
- **S4U2Self** (Service for User to Self)

S4U2Proxy

OK, what is S4U2Proxy?

S4U2Proxy is an extension that allows a service to use the TGS sent by the client user to order from the KDC a new TGS for a third service, in behalf of the client user.

But, how does the KDC knows if an user/service can order a TGS for a third service using the client TGS?

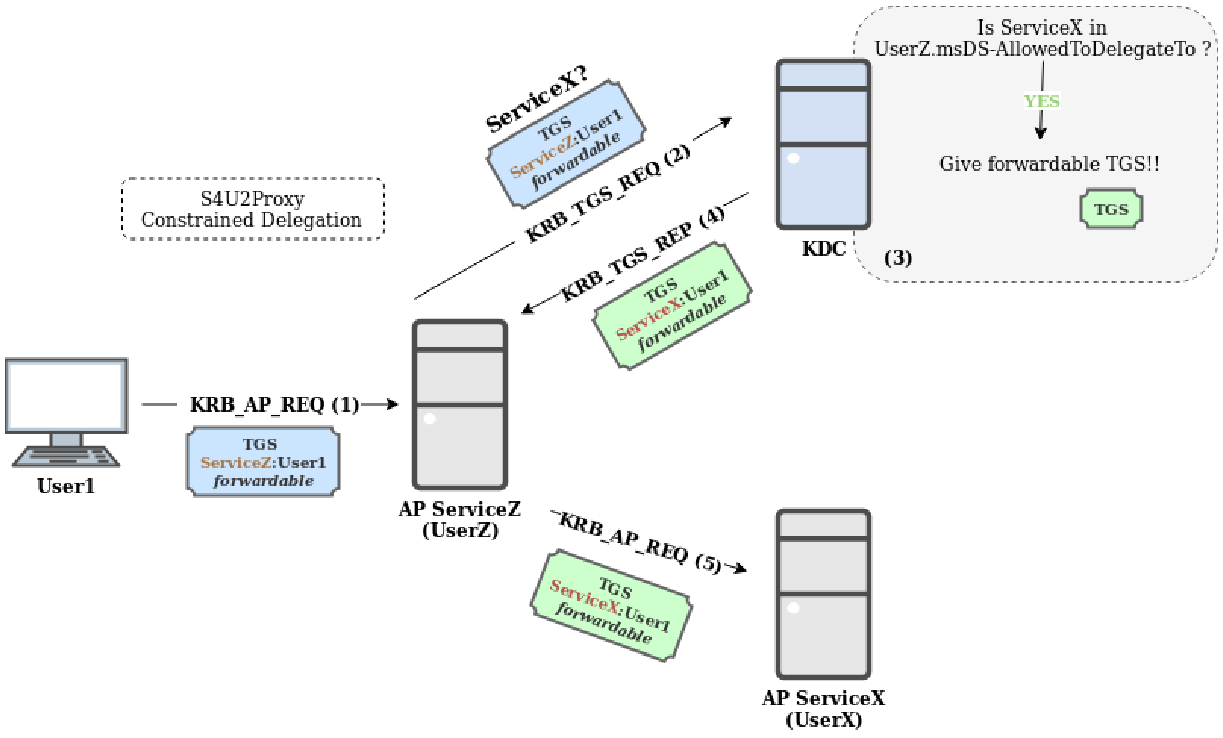
The KDC has to check 2 lists: On one hand, each user has a list of which services it can order a TGS for. This list is stored in the *msDS-AllowedToDelegateTo* attribute of the user account. In order to modify this attribute, the *SeEnableDelegationPrivilege* privilege is required in the domain controller. This is the list used in Constrained Delegation. On the other hand, each user has a list of other users which are allowed to ask a TGS for any of its services. This list is stored in the *msDS-AllowedToActOnBehalfOfOtherIdentity* attribute. The user by itself can edit his own list of allowed users on demand. This is the list used in RBCD. Therefore the KDC will check for the following conditions in order to determine if delegation can be applied:

- If the service client is protected against delegation (member of *Protected Users* group or has the *NotDelegated* flag set), then S4U2Proxy will fail.
- If the requested service is in *msDS-AllowedToDelegateTo*, then the KDC will check:
  - If the sent TGS is *forwardable* (the flag *forwardable* is set), then KDC will return a *forwardable* TGS for the requested service (Constrained Delegation is applied in this case).
  - Else the S4U2Proxy will fail.

- If the user who is requesting the TGS is listed in the *msDS-AllowedToActOnBehalfOfOtherIdentity* attribute of the requested service owner account, then the KDC will return a *forwardable* TGS (RBCD is applied in this case).
- Else the S4U2Proxy will fail.

It should be noted that, in case of RBCD the sent TGS does not need to be *forwardable*. Moreover, if the user can apply Constrained Delegation and RBCD for the same target service, Constrained Delegation will take precedence. That means that in this case S4U2Proxy will fail if the sent TGS is not *forwardable*, without even trying to apply RBCD. Besides, one curious fact is that the TGS returned by S4U2Proxy is always *forwardable*.

Please, can you give me an example of S4U2Proxy in Constrained Delegation?



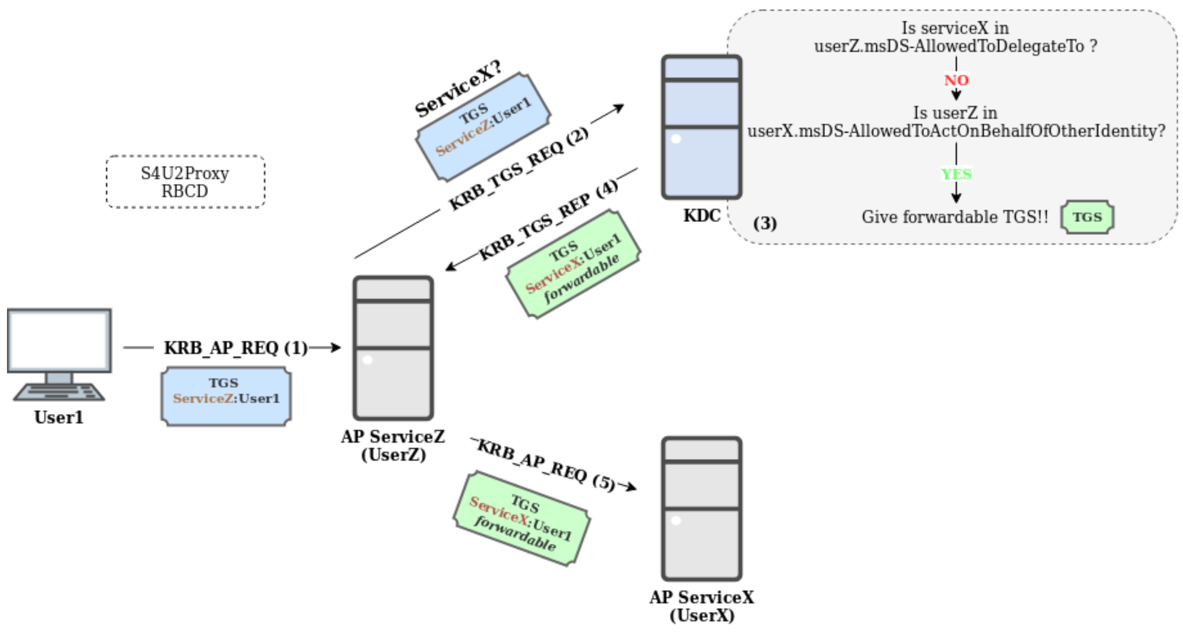
S4U2Proxy Constrained Delegation

In order to clarify the use of S4U2Proxy, the next example is shown where delegation is resolved by applying Constrained Delegation (for sake of clarity it is assumed that *User1* is not protected against delegation by *Protected Users* or *NotDelegated*, otherwise the delegation would fail): In this example:

1. *User1* sends the TGS to *ServiceZ*.
2. *ServiceZ*, owned by *UserZ*, uses this TGS to ask the KDC for a TGS for *ServiceX* on behalf of *User1*.
3. The KDC checks:
    1. If *ServiceX* is listed in *UserZ msDS-AllowedToDelegateTo* (Yes).
    2. If the sent TGS is *forwardable* (Yes). Constrained Delegation is applied.
4. The KDC returns a TGS to *ServiceZ* to interact with *ServiceX* on behalf of *User1*.
5. *ServiceZ* uses the new TGS to interact with *ServiceX*, by impersonating *User1*.

Please, can you give me an example of S4U2Proxy in RBCD?

In order to clarify the use of S4U2Proxy, the next example is shown where delegation is resolved by applying RCBD (for sake of clarity it is assumed that *User1* is not protected against delegation by *Protected Users* or *NotDelegated*, otherwise the delegation would fail):



S4U2Proxy RBCD

In this example:

1. *User1* sends the TGS to *ServiceZ*.
2. *ServiceZ*, owned by *UserZ*, uses this TGS to ask the KDC for a TGS for *ServiceX* on behalf of *User1*.
3. The KDC checks:
    1. If *ServiceX* is listed in *UserZ msDS-AllowedToDelegateTo* (No). Constrained Delegation cannot be applied.
    2. If *UserZ* is listed in *msDS-AllowedToActOnBehalfOfOtherIdentity* of *UserX*, the owner of *ServiceX* (Yes). RBCD can be applied.
4. The KDC returns a TGS to *ServiceZ* to interact with *ServiceX* on behalf of *User1*.
5. *ServiceZ* uses the new TGS to interact with *ServiceX*, by impersonating *User1*.

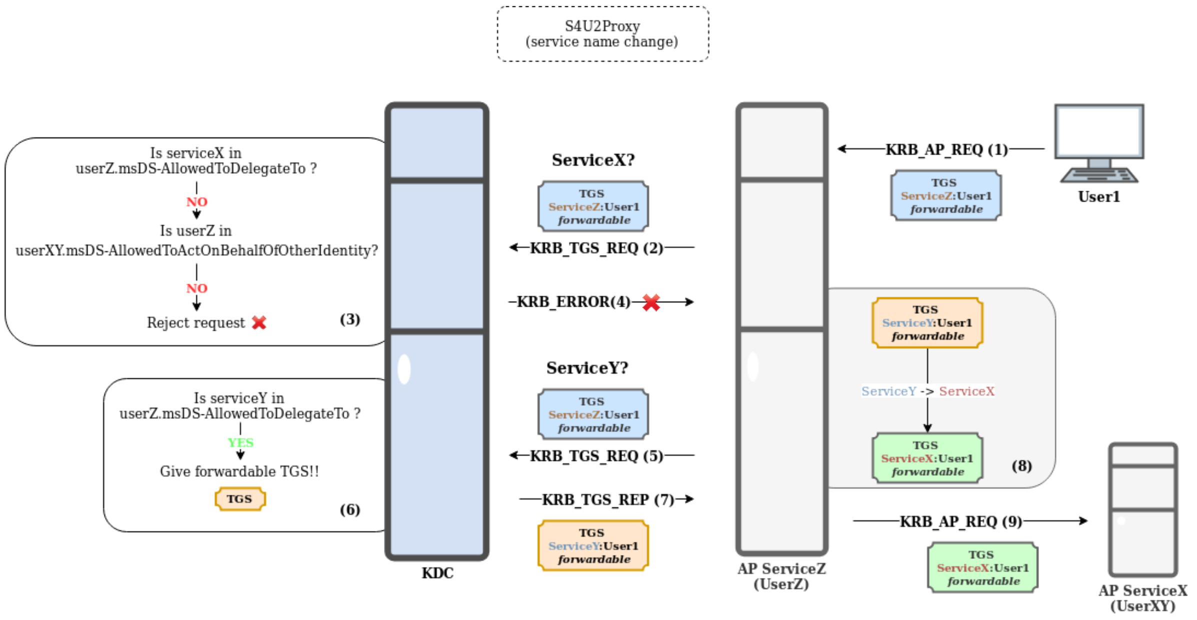
OK, anything else?

Yes. There is a interesting trick using TGS: it is possible to use the same TGS for any service of the same user just by changing the target service name. This is because 2 facts:

- The service name is written in the cleartext part of the ticket, which anyone can modify.
- All the services of the same user share the same Kerberos key, therefore any service can decrypt correctly a TGS for another service of the same user.

This trick can be used to bypass the white list (*msDS-AllowedToDelegateTo*) of services used by Constrained Delegation.

OK, example?


S4U2Proxy
(service name change)
Is serviceX in userZ.msDS-AllowedToDelegateTo ?
NO
Is userZ in userXY.msDS-AllowedToActOnBehalfOfOtherIdentity?
NO
Reject request
(3)
Is serviceY in userZ.msDS-AllowedToDelegateTo ?
YES
Give forwardable TGS!!
TGS
(6)
KDC
ServiceX?
TGS ServiceZ:User1 forwardable
KRB_TGS_REQ (2)
KRB_ERROR(4)
ServiceY?
TGS ServiceZ:User1 forwardable
KRB_TGS_REQ (5)
KRB_TGS_REP (7)
TGS ServiceY:User1 forwardable
AP ServiceZ (UserZ)
KRB_AP_REQ (1)
TGS ServiceZ:User1 forwardable
User1
TGS ServiceY:User1 forwardable
ServiceY -> ServiceX
TGS ServiceX:User1 forwardable
(8)
KRB_AP_REQ (9)
TGS ServiceX:User1 forwardable
AP ServiceX (UserXY)

In this example:

1. *User1* sends the TGS to *ServiceZ*.
2. *ServiceZ*, owned by *UserZ*, uses this TGS to ask the KDC for a TGS for *ServiceX*, owned by *UserXY*, on behalf of *User1*.
3. The KDC checks:
    1. If *ServiceX* is in *UserZ msDS-AllowedToDelegateTo* (No). Constrained Delegation cannot be applied.
    2. If *UserZ* is in *UserXY msDS-AllowedToActOnBehalfOfOtherIdentity* (No). RBCD cannot be applied.
4. The KDC rejects the request made by *ServiceZ*.
5. *ServiceZ* asks the KDC , on behalf of *User1*, for a TGS for *ServiceY*, another service owned by *UserXY*.
6. The KDC checks:
    1. If *ServiceY* is in *msDS-AllowedToDelegateTo* of *UserZ* (Yes).
    2. If the sent TGS is *forwardable* (Yes). Constrained Delegation can be applied.
7. The KDC returns a TGS to *ServiceZ* to interact with *ServiceY* on behalf of *User1*.
8. ***ServiceZ* (*UserZ*) changes the target service of the TGS from *ServiceY* to *ServiceX*.**
9. *ServiceZ* uses the modified TGS to interact with *ServiceX*, by impersonating *User1*.

It should be noted that this is not the normal course of action of a legitimate service, since no service is going to edit the service name of the TGS in order to contact with another service. Actually, this behavior is more typical of an attacker, which tries to use this trick to bypass the restriction imposed by the KDC.

S4U2Self

OK, but if S4U2Proxy can accomplish impersonation, what is the purpose of S4U2Self?

The purpose of S4U2Self is to allow the use of Delegation to services that do not support Kerberos authentication, and therefore, are unable to get a TGS from the client user. To bypass this limitation, S4U2Self allows a service to request the KDC for a TGS for itself, on behalf of another user. This is also known as *Protocol Transition*.

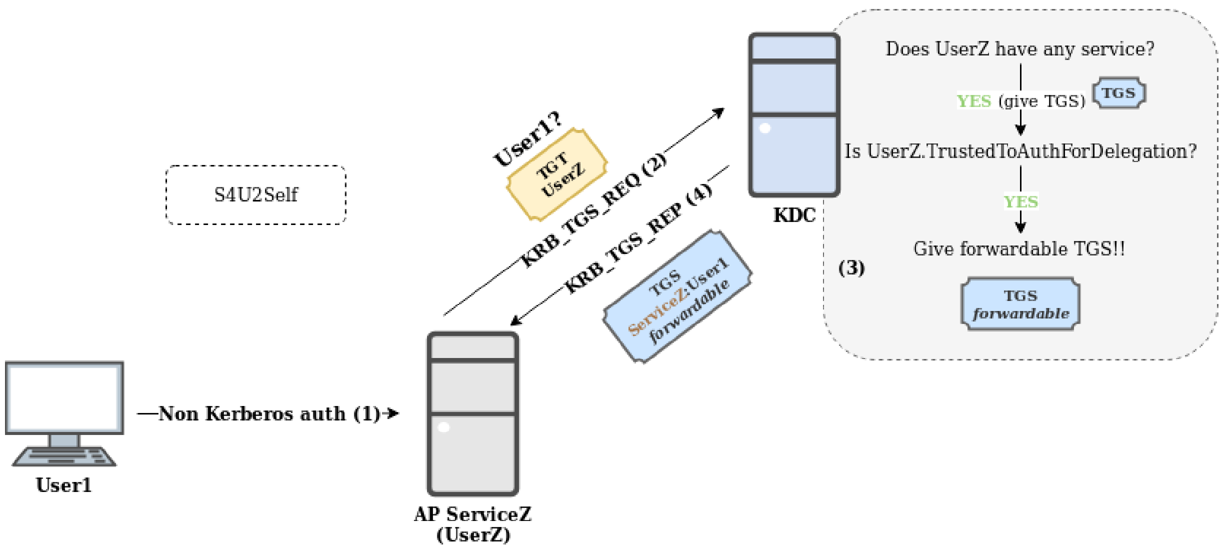
OK, can any user use S4U2Self?

No. Actually, the KDC will respond differently to a S4U2Self request based on some characteristics of the user account, such are the services of the account and the flag TrustedToAuthForDelegation. The *TrustedToAuthForDelegation* (or ADS_UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION) flag is contained in the User-Account-Control attribute of user accounts. To modify this flag it is required, to have the SeEnableDelegationPrivilege privilege in the domain controller. To apply S4U2Self, the KDC will follow the following rules:

- If the user account has no services, then the KDC will not return a TGS.
- If the target user for impersonation is not allowed to be delegated (because is in *Protected Users* group or the *NotDelegated* flag is set), then KDC will return a TGS without the *forwardable* flag. It seems also that the TGS includes an incorrect service name, but this can be modified as seen above.
- If the *TrustedToAuthForDelegation* flag is set for the user, then the KDC will return a TGS with the *forwardable* flag.
- If the TrustedToAuthForDelegation flag is not set for the user, then the KDC will return a TGS without the *forwardable* flag.

Please, can you give me an example of S4U2Self?

The next example shows the use of the S4U2Self extension (for sake of clarity it is assumed that *User1* is not protected against delegation by *Protected Users* or *NotDelegated*, otherwise the KDC would return a non-*forwardable* TGS):



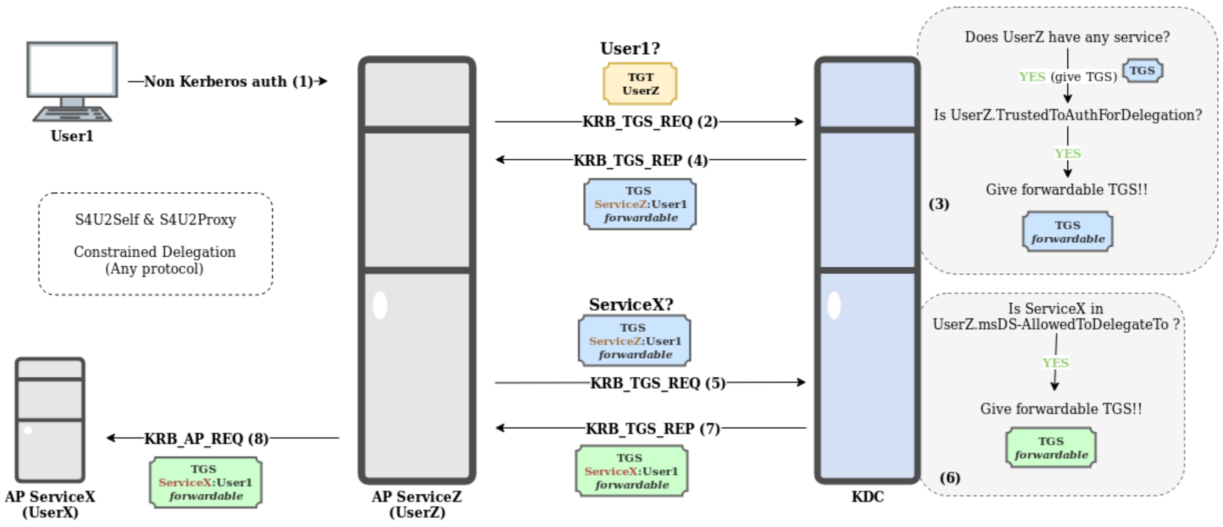
S4U2Self

In this example:

1. *User1* performs authentication against *ServiceZ* without using Kerberos. NTLM or other protocols could be used in this situation.
2. *ServiceZ*, owned by *UserZ*, asks for a TGS for itself on behalf of *User1*.
3. The KDC checks:
    1. If *UserZ* is the owner of any service (Yes).
    2. If *TrustedToAuthForDelegation* flag is set for *UserZ* account (Yes).
4. The KDC returns to *ServiceZ* a *forwardable* TGS for itself, on behalf *User1*, which can be used in S4U2Proxy.

Please please, can you give me an example of S4U2Self and S4U2Proxy used together?

The next example shows the use of S4U2Self and S4U2Proxy together, resolved by applying Constrained Delegation (for sake of clarity it is assumed that *User1* is not protected against delegation by *Protected Users* or *NotDelegated*, otherwise delegation would fail):

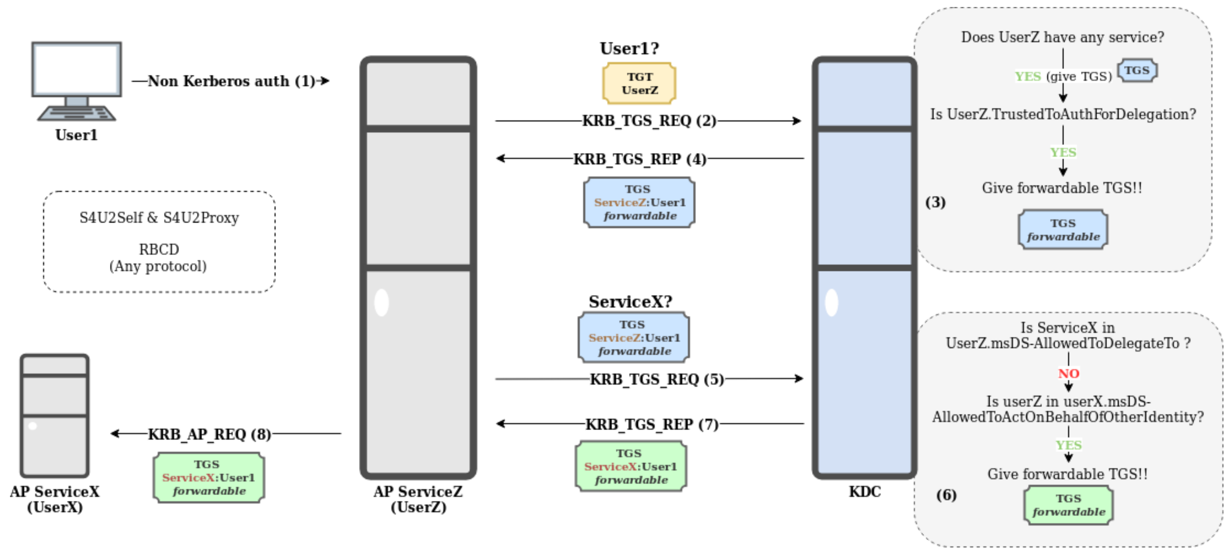


S4U2Self & S4U2Proxy Constrained Delegation

In this example:

1. *User1* performs authentication against *ServiceZ* without using Kerberos. NTLM or other protocols could be used in this situation.
2. *ServiceZ*, of *UserZ*, asks for a TGS for itself on behalf of *User1*.
3. The KDC checks:
    1. If *UserZ* is the owner of any service (Yes).
    2. If *TrustedToAuthForDelegation* flag is set for *UserZ* account (Yes).
4. The KDC returns to *ServiceZ* a *forwardable* TGS for itself, on behalf *User1*.
5. *ServiceZ*, uses this TGS to ask the KDC for a TGS for *ServiceX* on behalf of *User1*.
6. The KDC checks:
    1. If *ServiceX* is listed in *UserZ msDS-AllowedToDelegateTo* (Yes).
    2. If the sent TGS is *forwardable* (Yes). Constrained Delegation is applied.
7. The KDC returns a *forwardable* TGS to *ServiceZ* to interact with *ServiceX* on behalf of *User1*.
8. *ServiceZ* uses the new TGS to interact with *ServiceX*, by impersonating *User1*.

The next example shows the use of S4U2Self and S4U2Proxy together, resolved by applying RBCD (for sake of clarity it is assumed that *User1* is not protected against delegation by *Protected Users* or *NotDelegated*, otherwise delegation would fail):



S4U2Self & S4U2Proxy RBCD

In this example:

1. *User1* performs authentication against *ServiceZ* without using Kerberos. NTLM or other protocols could be used in this situation.
2. *ServiceZ*, owned by *UserZ*, asks for a TGS for itself on behalf of *User1*.
3. The KDC checks:
    1. If *UserZ* is the owner of any service (Yes).
    2. If *TrustedToAuthForDelegation* flag is set for *UserZ* account (Yes).
4. The KDC returns to *ServiceZ* a *forwardable* TGS for itself, on behalf *User1*.
5. *ServiceZ*, uses this TGS to ask the KDC for a TGS for *ServiceX* on behalf of *User1*.

6. The KDC checks:
    1. If *ServiceX* is listed in *UserZ msDS-AllowedToDelegateTo* (No). Constrained Delegation cannot be applied.
    2. If *UserZ* is listed in *msDS-AllowedToActOnBehalfOfOtherIdentity* of *UserX*, the owner of *ServiceX* (Yes). RBCD can be applied.
7. The KDC returns a *forwardable* TGS to *ServiceZ* to interact with *ServiceX* on behalf of *User1*.
8. *ServiceZ* uses the new TGS to interact with *ServiceX*, by impersonating *User1*.

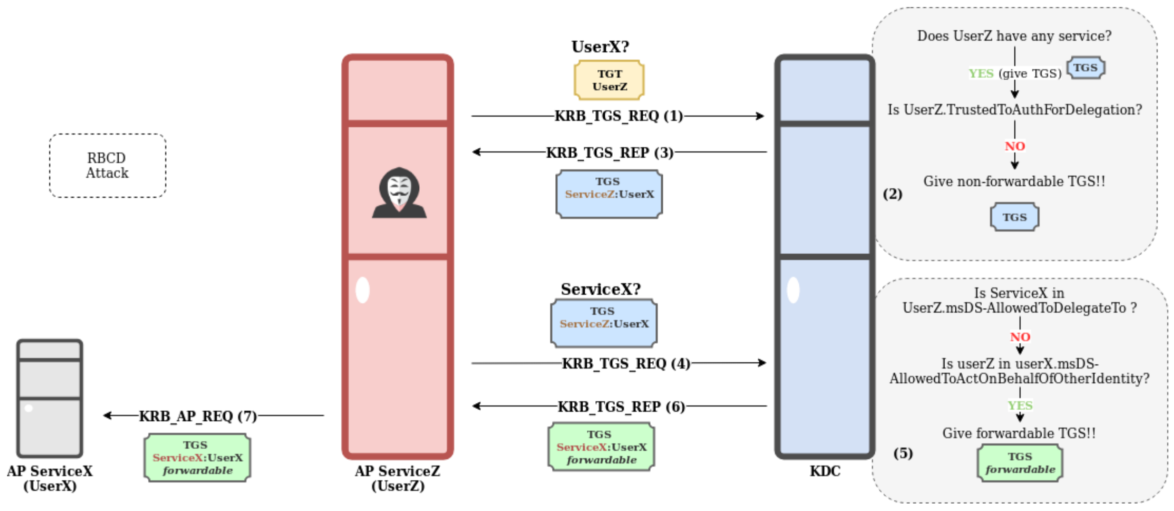
OK, but how could a pentester take advantage of Constrained Delegation?

There are a few possible scenarios:

- If a pentester is able to compromise a computer which is hosting services with Constrained Delegation, it could harvest the TGS's for third-party services of the clients of each service and interact with them. Additionally, those TGS's could be edited to change the target service and interact with other services belonging to the same target user.
- If a pentester is able to compromise the account of a user which has Constrained Delegation permissions, it could collect the TGS's of the clients of services belonging to the compromised account. Moreover, in case the compromised user account has the flag TrustedToAuthForDelegation activated, the pentester itself could use S4U2Self to request TGS's of the clients directly to the KDC.
- Moreover, in any of the previous scenarios, for each TGS it could be possible to modify the service name in order to increase the number of services the pentester could interact.

OK, but how could a pentester take advantage of RBCD?

If a pentester is able to compromise an account *UserZ* which is listed in the *msDS-AllowedToActOnBehalfOfOtherIdentity* attribute of another account named *UserX*, then it would be possible to chain the use of S4U2Self and S4U2Proxy to get access to the services of *UserX* by impersonating it. Here is an example of this scenario where *UserZ* uses S4U2Self and S4U2Proxy to interact with the service *ServiceX* by impersonating its owner *UserX* (it is assumed that *UserX* is not protected against delegation by *Protected Users* or *NotDelegated*):


RBCD Attack
UserX?
TGT UserZ
KRB_TGS_REQ (1)
KRB_TGS_REP (3)
TGS ServiceZ:UserX
Does UserZ have any service?
YES (give TGS)
TGS
Is UserZ.TrustedToAuthForDelegation?
NO
Give non-forwardable TGS!!
(2)
TGS
ServiceX?
TGS ServiceZ:UserX
KRB_TGS_REQ (4)
KRB_TGS_REP (6)
TGS ServiceX:UserX forwardable
Is ServiceX in UserZ.msDS-AllowedToDelegateTo ?
NO
Is userZ in userX.msDS-AllowedToActOnBehalfOfOtherIdentity?
YES
Give forwardable TGS!!
(5)
TGS forwardable
KRB_AP_REQ (7)
TGS ServiceX:UserX forwardable
AP ServiceX (UserX)
AP ServiceZ (UserZ)
KDC

RBCD attack

In this example:

1. *UserZ* (pentester), owner of *ServiceZ*, requests a TGS for *ServiceZ* on behalf for *UserX* (S4U2Self).
2. The KDC checks:
   1. If *UserZ* is the owner of any service (Yes).
   2. If *TrustedToAuthForDelegation* flag is set for *UserZ* account (Yes).
3. The KDC returns to *UserZ* a non-*forwardable* TGS for *ServiceZ* on behalf of *UserX*.
4. *UserZ* uses the new *non-forwardable* TGS to order another TGS (S4U2Proxy) for *ServiceX* on behalf of *UserX*.
5. The KDC checks:
   1. If *ServiceX* is listed in *UserZ msDS-AllowedToDelegateTo* (No). Constrained Delegation cannot be applied.
   2. If *UserZ* is listed in *msDS-AllowedToActOnBehalfOfOtherIdentity* of *UserX*, the owner of *ServiceX* (Yes). RBCD can be applied.
6. The KDC returns a *forwardable* TGS to *ServiceZ* to interact with *ServiceX* on behalf of *UserX*.
7. *UserZ* uses the new TGS to interact with *ServiceX*, by impersonating *UserX*.

Other interesting possibility, as a way of persistence, is to configure RBCD from arbitrary users to the *krbtgt* account. Thus, those arbitrary users will be able to generate TGS's to the *krbtgt* service for any user, which means generating TGT's (since TGT are just TGS's to the *krbtgt* service) for any user in the domain, similarly of the Golden Ticket attack.

Conclusion

As we have seen, Kerberos seems to have and endless surface of attack, now that delegation is on the table too, with its 3 different flavors. Be aware that some of this features could determine the success of an intrusion. Therefore, in the next article, I will show you how to perform delegation attacks in a practical way, as well as the tools that can be used in order to carry out them, or what mitigations can be applied, since the concepts are already in your mind. See you.

Kerberos delegation references

- [MS-SFU]: Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol => https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/3bff5864-8135-400e-bdd9-33b552051d94
- Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory => https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html
- Active Directory Security Risk #101: Kerberos Unconstrained Delegation => https://adsecurity.org/?p=1667
- Trust? Years to earn, seconds to break => https://labs.mwrinfosecurity.com/blog/trust-years-to-earn-seconds-to-break/
- S4U2Pwnage => https://harmj0y.medium.com/s4u2pwnage-36efe1a2777c
- The Most Dangerous User Right You (Probably) Have Never Heard Of => https://blog.harmj0y.net/activedirectory/the-most-dangerous-user-right-you-probably-have-never-heard-of/
- Another Word on Delegation => https://blog.harmj0y.net/redteaming/another-word-on-delegation/
- Kerberos Delegation, SPNs and More… => https://www.secureauth.com/blog/kerberos-delegation-spns-and-more
- Protected Users => https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/dn466518(v=ws.11)