Kerberos and Windows Security: Delegation

medium.com/@robert.broeckelmann/kerberos-and-windows-security-delegation-7b5a3f31d779

Robert Broeckelmann 6 февраля 2021 г.

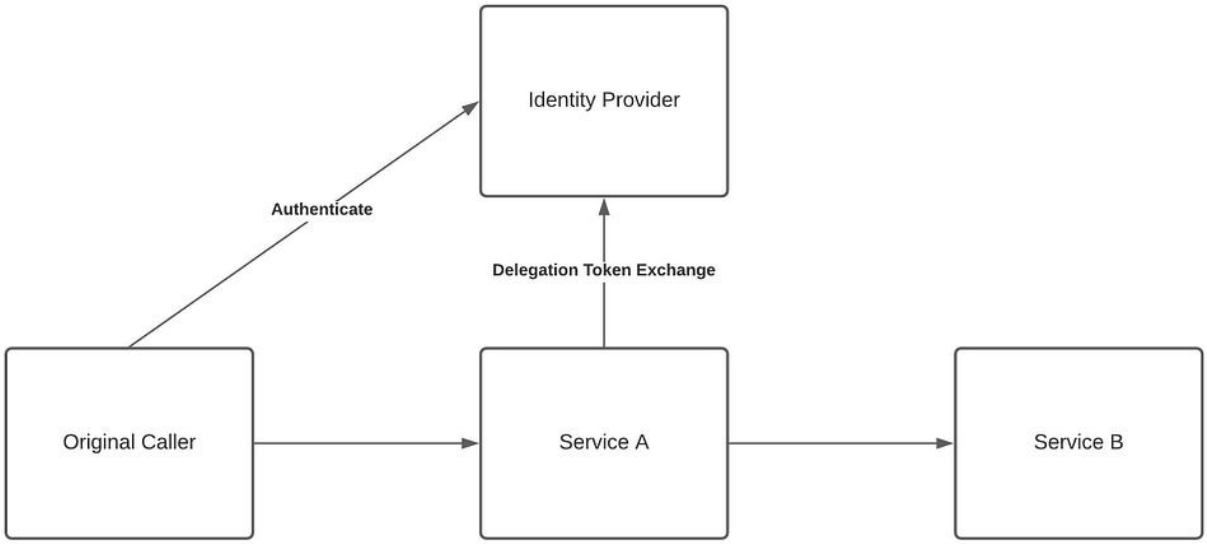
Robert Broeckelmann

Nature Art /

In this next post in the Kerberos and Windows Security Series, we are going to explore a very useful, but abstract feature of the Kerberos Authentication Protocol: Delegation. In particular, we are going to focus on the Windows implementation of this feature. Delegation allows downstream actors to interact with other services on behalf of the original authenticated user without that users credentials having to be provided directly to the downstream actors. This allows complex orchestration: Service (App) A calling Service (App) B calling Service (App) C and so on and so forth to be done in a secure manner within the security context of the original user without the users original (secret) credentials being passed on to each Service (App).

Let’s approach this in a protocol independent way first. Delegation is accomplished by, first, having the original caller authenticate against a trusted third-party (an Identity Provider, IdP, a Key Distribution Center, KDC, for Kerberos) and that trusted third-party issue a token that describes the authenticated user. Next, that token is used to make a call to a Service (we’ll call it Service A) with the token. Service A validates that token as

part of a normal authentication process. Then, maybe, Service A needs to call Service B. In order to do this, Service A needs a token that describes the original caller, but scoped to Service B. Delegation would allow Service A to exchange the token it has received from the caller for a new token that describes the same original caller, but scoped to Service B — note, all tokens are still issued by the IdP. Service A can then call Service B with the permissions of the original caller using the new token it received from the IdP. Service B will validate the token in roughly the same manner that Service A did when it received the intial request. This entire exchange is depicted in the following diagram:



Protocol-Independent Delegation Description

This is called “delegation”. In other words, the “Original Caller” has delegated permissions to “Service A” to call “Service B”. Depending on the protocol, this may also be phrased as “Service A” calls “Service B” on behalf of the original caller. Or, “Service A” impersonates the original caller while calling “Service B”. The various protocols that support this concept are not necessarily consistent in their terminology.

Depending on the product, protocol, and feature sets, there varying levels of security that can be accomplished with this exchange. There may be limits placed on what new scope Service A can request the permissions of the original caller be delegated to it (Service A). There may be limits on what original caller identities that Service A is allowed to obtain a new token for that is scoped to Service B. Service B may restrict what services are or users are allowed to connect to it using delegation. furthermore, there are different types of delegation wherein the IdP and Service B may be aware that Service B is connecting with the permissions of the original or it may only see a token describing the original caller. This is the concept that allows end-to-end secure user identity propagation to occur. I’ve written about the use of this concept many times, but I haven’t dug into the details of how to implement it yet.

Admittedly, delegation introduces complexity, but it also eliminates the use of service (or generic) accounts with administrative or super user privileges that can perform any task on any data set for any user that may interact with the system. This is dangerous, if the

system is ever compromised, the attacker has access to everything. On the other hand, if the attacker could only access data for or perform actions of behalf of the user whose credentials were compromised, the damage isn't nearly as severe.

Now, let's see how Kerberos and Windows Security implement these concepts. Admittedly, Kerberos delegation (or its Windows Kerberos Extensions) is probably not the mechanism that would be used with the REST APIs that I am typically writing about. Nevertheless, these mechanisms are common in the Windows landscape. We'll tie this into API security soon enough.

A more detailed discussion of delegation (and impersonation) in the general case can be found here.

Kerberos Feature: delegation

Kerberos delegation is a feature that allows an application (or service), likely recently invoked itself by the end user (though not necessarily the case), to use the end-user's identity to access resources hosted on a different server on-behalf-of that end user. The Kerberos spec, RFC 4120, refers to this as "proxyable" in section 2.5. The service that is allowed to use the end-user's identity to invoke yet another service must be granted a ticket (a special ticket that contains metadata describing what is being done).

From an old Microsoft Tech Note, we have "Kerberos delegation is actually solving an ancestral problem with multi-tier applications. When you have a front-end server, like a web server, using back-end services like a database, or a SQL reporting, the application running on the front-end is accessing those back-end services on behalf [of the calling user using] a service account.…This is the "double-hop" effect. And this can cause an ugly pop-up in which a disgruntled user will have to type its password again. But if we allow the front-end server (or rather its service account) to do Kerberos delegation, then the service will ask for tickets on behalf the user and the context of the user will be provided end-to-end." I like this explanation of delegation as a concept.

From RFC 4120, we have the following for two types of delegation that the Kerberos v5 spec describes:

```
Traditionally, the Kerberos protocol enabled delegation to be accomplished using
one of two mechanisms: The client can get a ticket for the back-end server and
then give it to the front-end server. Tickets obtained in this way — by a client
for a proxy — are called proxy tickets. The difficulty with proxy tickets is that
the client must have information about the name of the back-end server. The client
can give the front-end server a TGT that the server can use to request tickets as
needed. Tickets obtained in this way — with credentials forwarded by a client —
are called forwarded tickets. This second method overcomes the difficulty (that
the client must have information about the name of the back-end server) presented
by the first method.
```

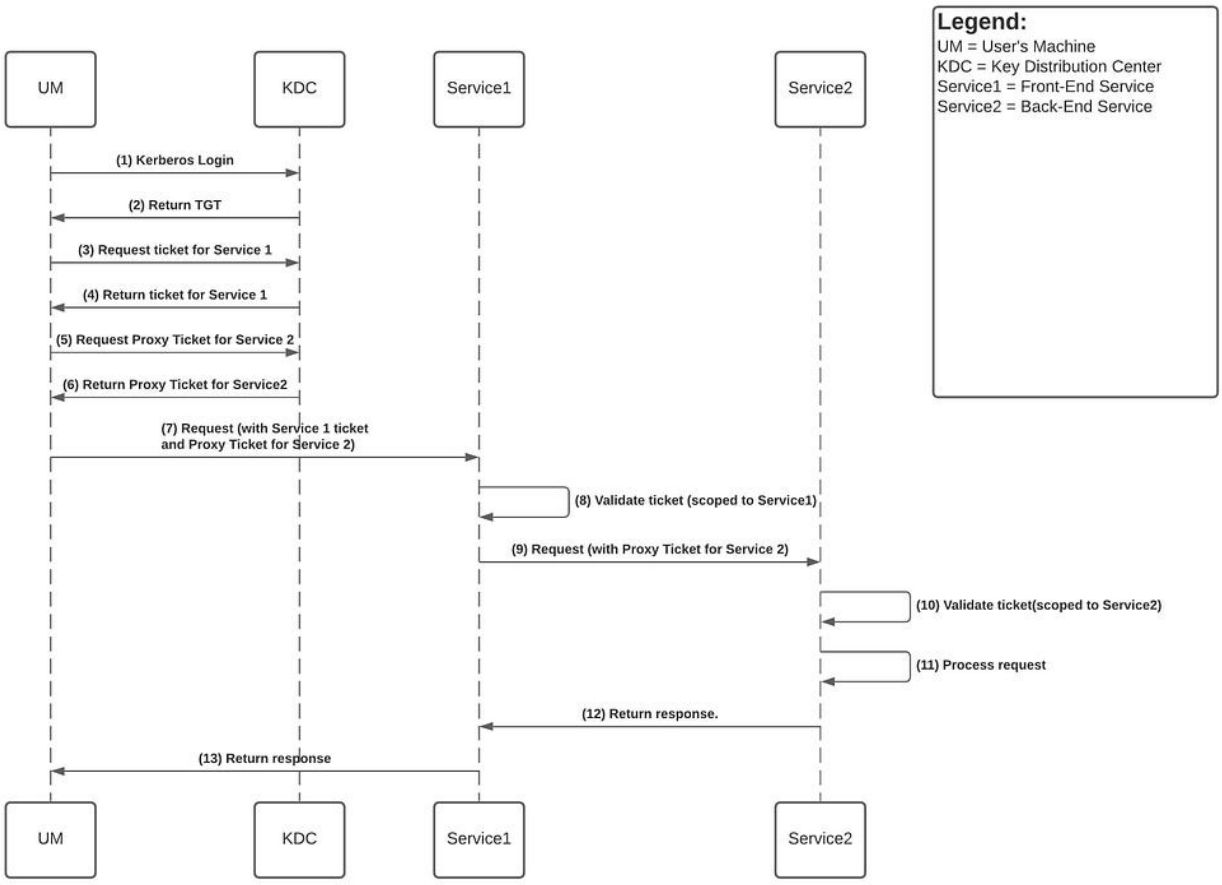
From here, we have "Kerberos, as defined by RFC 1510 [and later in RFC 4210], does provide a constrained delegation feature known as a proxy ticket, but the client must request these tickets on the service's behalf." Obviously, the burden that "Proxy Tickets"

place on the client to know about the back-end architecture makes this option undesirable for many use cases. In fact, I'm having trouble coming up with a situation where I would actually want the client to have to be involved in the back-end details in this manner. The spec does give an option to obtain a proxy ticket that is valid for any network address, but this still requires the client to know about delegation occurring on the back-end, requires the policy to allow it, and would violate the least-privilege principal. So, generally, this is not going to be the desired approach.

The need for the client to know about the backend service is addressed by the second option (Forwarded Tickets), but giving a remote actor the user's Ticket Granting Ticket (TGT) may also be giving more access than the user or developer intended.

Whether either of these will be available to a client is a configurable Kerberos policy. For Windows Kerberos, see here. Regardless, in the Windows Kerberos world, there are other options available for delegation that should be considered.

The protocol steps for a forwarded TGT are described here. The protocol steps for delegation via Proxy Ticket is outlined in the following diagram (I couldn't find this explictly depicted in any other google search result):



Kerberos delegation via Proxy Ticket

From a practical standpoint, one does not usually see these Kerberos protocol features used in the Windows Kerberos implementation. Instead, there are Microsoft Kerberos Extensions that we will explore next.

Windows Kerberos Extensions (For delegation)

From here, we have the following Kerberos Protocol Extensions from Microsoft that were introduced starting in Windows Server 2003:

```
S4U2Self) The protocol transition extension allows a kerberized service (that is,
a service designed to use Kerberos authentication features) to obtain a Kerberos
service ticket on behalf of a Kerberos principal to the service itself without
requiring the principal to initially authenticate to the KDC with a
credential.S4U2Proxy) The constrained delegation extension allows a kerberized
service to obtain service tickets (under the delegated user's identity) to a
subset of other services after it has been presented a service ticket obtained
either through KRB_TGS_REQ or the protocol transition extension.
```

The Protocol Transition Extension (S4U2Self) allows a trusted service to act on-behalf-of other Principals without something having provided those credentials to begin with. This allows a trusted service to authenticate a user using another identity protocol and then receive a Kerberos ticket for itself that describes the original user that is to be impersonated. This ticket can then be fed into another Kerberos call, such as S4U2Proxy to receive a ticket that describes the impersonated user for the desired service. You can see where this would be a foundational building block of a fully-featured Identity Provider. The use of S4U2Self with Active Directory Federation Services (ADFS) is discussed here. In an OAuth2 list server discussion, it was suggested that this referred to as Impersonation as described above, but I've had trouble finding other references to corroborate that — we'll explore that more in the next blog post.

The Constrained Delegation Extension(S4U2Proxy) is another foundational building block of a fully-featured Identity Provider and is critical to the use cases of interest because it limits who or what can be impersonated by the service. Of course, this goes beyond what is required by the spec, but one can also see where most real-world Kerberos implementations would need something similar. An Identity Provider (say ADFS) could use S4U2Proxy to obtain a ticket for a backend service that provides the permissions of the authenticated user in the next step (following S4U2Self) of a complete authentication sequence (if the backend service understands Kerberos).

From here, we have:

```
The Kerberos Security Support Provider (SSP) will first detect whether the
forwarded-TGT delegation mechanism is available (by checking whether there is a
forwarded TGT in the local ticket cache); if no forwarded TGT is available, the
Kerberos SSP will then try to perform the S4U2proxy delegation.
```

So, Microsoft Kerberos will always attempt to use the spec-defined forwarded-TGT delegation mechanism before it attempts to use S4U2proxy constrained delegation.

From here, we have:

The S4U_DELEGATION_INFO structure (section) lists the that have been delegated by this client and subsequent services or servers. The list is meaningful as the feature could be used multiple times in succession from service to service. This is useful for auditing purposes.

So, S4U2proxy uses a ticket that describes the service identity performing the delegation and the identity it is acting on-behalf-of. This is a type of composite token.

From here, we have the following notes about Kerberos Constrained Delegation (S4U2Proxy) before Windows Server 2012:

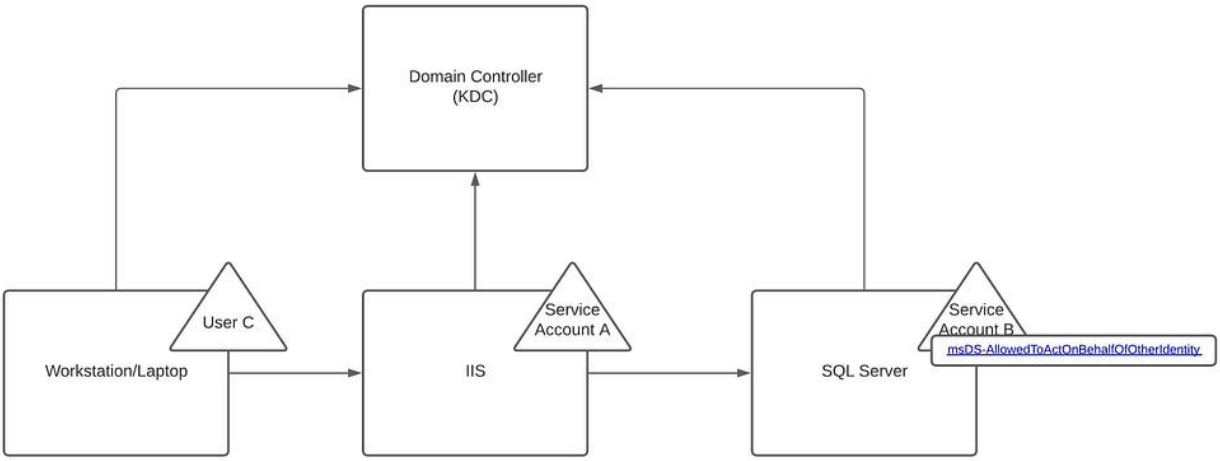
The use of [Kerberos Constrained Delegation] KCD before Windows Server 2012 presented three major issues:Delegation is controlled via modification of the impersonating (service) account and not the account of the resource server. This potentially creates a scenario where the resource owner has no control or knowledge of accounts that have been configured for delegation to services on his or her server.Registration of KCD [Service Principal Names] SPNs requires Domain Admin rights, often preventing delegation of KCD configuration to resource administrators.KCD cannot cross an Active Directory domain or forest boundary. This limitation required that the impersonating (service) account, the impersonated (user) account and the account of the resource server all resided in the same domain. This precluded the use of KCD for typical extranet scenarios where a web server would reside in an extranet or DMZ domain, with a SQL or other resource server residing in an internal domain. Using Kerberos Protocol Transition (KPT) in conjunction with KCD helped to address this issue somewhat. See “KCD Across Domain Boundaries” later in this blog post for info on KPT.

Several years ago, I was working with an organization that was a heavy user of ADFS. We ran into all of these limitations. Especially, in scenarios where we had an Enterprise Service Bus (pseudo-API Gateway) that advertised services that could be called by multiple user communities (Business-to-Employee — B2E, Business-to-Business — B2B, Business-to-Consumer — B2C). Each user community was defined in a different Active Directory domain. The ESB would request new downstream-tokens from ADFS via WS-Trust (with SAML2 Assertions as the response token type). Under-the-covers, ADFS was maintaining a kerberos ticket/session for each authenticated user that it was issuing SAML2 Assertions for. The service account the ESB used to obtain these downstream tokens was defined in the B2E AD domain. So, the WS-Trust delegation calls (and corresponding Kerberos Constrained Delegation calls) worked just fine for B2E users, but didn’t work so well for the other user communities.

From here, we have:

Server 2012 introduces a new kind of Kerberos constrained delegation that addresses many of the shortcomings that exist with the previous constrained delegation model. The new implementation of constrained delegation removes the dependencies on SPNs for delegation configuration, removes the need for domain administrative privileges, enables the resource administrator to own the delegation experience, and increases the scope of delegation.Constrained delegation in Server 2012 introduces the concept of controlling delegation of service tickets using a security descriptor rather than an allow list of SPNs. This change simplifies delegation by enabling the resource to determine which security principals are allowed to request tickets on behalf of another user.

Windows Server 2012 introduced the msDS-AllowedToActOnBehalfOfOtherIdentity attribute that is placed on a resource's security principal (see below). In the diagram below, Service Account B would have this attribute set to list Service Account A which would be performing a Kerberos Constrained Delegation call. The KDC will check this attribute when IIS attempts to obtain the downstream token. If allowed, then Service Account A will be able to obtain a ticket that describes User C, which can be used used when IIS (running as Service Account A) attempts to access SQL Server.



With that update, the Windows Kerberos Constrained Delegation feature begins to look like something that can safely be used. Though, one does still have to be careful with this feature. Exploring ways to exploit this feature of Windows Kerberos will have to be a subject of a future blog post.

Sequence diagrams and the detailed steps of Kerberos delegation (using a forwarded TGT) and the same with the Microsoft Kerberos Extensions can be found here. The sequence diagrams and explanation are thorough; so, I didn't reproduce them here.

Summary

Delegation is powerful concept that enables advanced use cases that are used often. Most notably among these use cases is the end-to-end secure identity propagation.

There are different approaches to delegation; each one results in roughly the same end result (namely how much metadata about the delegation protocol the downstream service and IdP see). There are different nomenclatures spanning protocols, specs, and product vendors. The Kerberos spec defined two mechanisms for implementing delegation. Those mechanisms have their shortcomings; so, Microsoft implemented two Kerberos protocol extensions that are commonly used in the Microsoft product suite. Since Windows Kerberos is so common in the industry, many third-party libraries have implemented support for these extensions.

In the next post, we're going to tie together delegation concepts from several identity protocols to give a comprehensive picture of delegation capabilities.

Image: Nature Art /