Kerberos and Windows Security: Kerberos v5 Protocol

medium.com/@robert.broeckelmann/kerberos-and-windows-security-kerberos-v5-protocol-b9c804e06479

Robert Broeckelmann 16 мая 2018 г.

Robert Broeckelmann

/ Cerberus Greek Monsters by beetroot

In our last post, we looked at the history of Kerberos and its use in Windows Security. This post continues our Kerberos and Windows Security discussion. Here we will look at the Kerberos v5 protocol independent of its use in Microsoft Windows.

Before we jump into the details, consider the following:

- As I noted in the post, Kerberos is the oldest identity protocol in common use today.
- It has had dozens of added to it — both proprietary and spec-defined.
- It is not HTTP-based, unlike most of the recent . So, the data being transferred between actors is not in human readable format without tooling/translation.
- The protocol has been crowbarred into a variety of different scenarios that no one could have envisioned in the mid-1980s.

All of these contribute to the overarching theme of Kerberos being rather complex. That being said, it has benefits over the other identity protocols and is widely used. So, it is wise to understand it.

Kerberos v5

From the Kerberos v5 RFC (RFC 4120), we have

```
Kerberos provides a means of verifying the identities of principals,(e.g., a
workstation user or a network server) on an open(unprotected) network.  This is
accomplished without relying onassertions by the host operating system, without
basing trust on host addresses, without requiring physical security of all the
hosts on the network, and under the assumption that packets traveling along the
network can be read, modified, and inserted at will.  Kerberos performs
authentication under these conditions as a trusted third-party authentication
service by using conventional (shared secret key) cryptography.
```

Before we dive into an explanation of how Kerberos works, I encourage the reader to take the time to go read this fictional dialogue about the creation of an authentication protocol. This was originally written in February, 1997 by Theodore Ts'o (one of the MIT researchers involved in Kerberos' creation). If not entertaining, it builds the thought process and justification of the protocol structure up well enough.

Kerberos predates the SSL/TLS protocols. So, its creators had to implement a secure authentication protocol that didn't have access to the ubiquitous transport layer encryption technology available today.

System Actors

First, let’s describe a few Kerberos actors that we need to know:

Realm: A set of network nodes (hosts, workstations, VMs, servers, etc) that share a common Kerberos database. I like to think of it as all of those nodes plus the KDC (and associated database). This brings it in line with definitions I’ve given for similar concepts across the other identity protocols.

Principal: A unique identity to which Kerberos tickets can be assigned. This could be a client, a user, or a server providing a service. The Principal name in Kerberos v5 is of the form primary/instance@REALM. See here for more information.

Hosts/Clients: A process, host, server, VM, or other network node that makes use of a network service (that understands Kerberos and is part of the same Realm or a trusted-Realm) on behalf of a user.

Server (sometimes called a Service Server — SS): A particular Principal that provides a resource to network Clients. The server is, also, sometimes referred to as the Application Server.

Service: A resource provided to network clients. Typically, provided by more than one server.

Key Distribution Center (KDC): A network service that supplies tickets and temporary session keys; or an instance of that service or the host on which it runs. The KDC services both initial ticket and ticket-granting ticket requests. It is composed of the Authentication Service and Ticket Granting Server (see below).

Authentication Server (AS): The KDC component that handles the initial request and issues a TGT. The AS is responsible for maintaining a database of principals (users and servers) and the associated secret keys.

Ticket Granting Server (TGS): The KDC component that handles the ticket-granting ticket step of the Kerberos protocol. This issues tickets for the requested services.

Environmental Assumptions

The Kerberos v5 specification makes the following assumptions:

- "Denial of service" attacks are not solved with Kerberos. Though, extensions and implementations have put mechanisms in place that do protect against such attacks.
- Principals MUST keep their secret keys secret. If the password is compromised, the system isn’t so secure.
- "Password guessing" attacks are not solved by Kerberos. Again, extensions and implementations have put mechanisms in place that do protect against this.

- Each host on the network MUST have a clock which is “loosely synchronized” to the time of the other hosts. See “Time Synchronization” Section below about time. Most identity protocols in common use today have this requirement.
- Principal identifiers are not recycled on a short-term basis. Don’t reuse user and server names.

Cross-Realm Authentication

Kerberos v5 supports authentication and interaction between actors in different Kerberos Realms. To keep things simple, we will assume that all actors are inside the same Kerberos Realm for this discussion.

Sub-Protocols

The Kerberos authentication protocol defines three sub-protocols (or message exchanges). These include:

- Authentication Service Exchange
- Token Service Exchange
- Client/Server Exchange

In the protocol diagrams below, all three of these are described.

NTP & Time Synchronization

All Kerberos actors need to have the time synchronized to a central time source — like most identity protocols. This is usually done with Network Time Protocol (NTP). If the times are not synchronized, then weird and frustrating problems will occur.

TCP vs UDP

The Kerberos protocol uses port 88 (UCP or TCP, both must be supported) on the KDC when used on an IP network. The spec supports using alternate ports; especially to support multiple Kerberos realms on the same server. Clients must support using TCP, but can use UDP. Some extensions to the Kerberos v5 spec will fail if TCP is not used. The Client can send multiple request messages before it receives a response — though, to keep things simple, we won’t be trying that there.

Since Kerberos can run on unreliable transport protocols such as UDP, Kerberos actors must be prepared to retransmit messages that may have been lost in this scenario. Or, better yet, just use TCP.

Pre-Authentication

Pre-Authentication provides a mechanism for the Authentication Server to identify who the Principal (user or system being authenticated) is before it issues an KRB_AS_RSP message. This prevents certain types of known attacks on Kerberos.

From RFC 6113 (A Generalized Framework for Kerberos Pre-Authentication) we have,

```
The core Kerberos specification [] treats pre-authenticationdata (padata) as an
opaque typed hole in the messages to the keydistribution center (KDC) that may
influence the reply key used to encrypt the KDC reply.  This generality has been
useful: pre-authentication data is used for a variety of extensions to
theprotocol, many outside the expectations of the initial designers.
```

By default, Windows Domains require Pre-Authentication data to be provided in KRB_AS_REQ messages. From here, we have:

```
By default the KDC requires all accounts to use pre-authentication. This is a
security feature which offers protection against password-guessing attacks. The AS
request identifies the client to the KDC in plain text. If pre-authentication is
enabled, a time stamp will be encrypted using the user’s password hash as an
encryption key. If the KDC reads a valid time when using the user’s password hash,
which is available in the Active Directory, to decrypt the time stamp, the KDC
knows that request isn’t a replay of a previous request.
```

Pre-Authentication provides the mechanism that many Kerberos extensions need to integrate with those “unexpected” use cases Kerberos wasn’t originally meant for. The Kerberos spec itself uses the Pre-Authentication header to pass authentication data in the TGS-REQ step.

Tickets

From RFC 4120, a ticket is “a record that helps a client authenticate itself to a server; it contains the client’s identity, a session key, a timestamp, and other information, all sealed using the server’s secret key. It only serves to authenticate a client when presented along with a fresh Authenticator.” We talk about Authenticators below.

There are different types of tickets, including a Ticket Granting Ticket (TGT) and a Service Ticket.

A TGT is a ticket intended for the TGS that can be used to obtain tickets for additional services without the need for the original user password or secret key.

A Service Ticket is a ticket issued by the TGS that can be used to authenticate to a desired service.

Tickets can be renewed. The protocol provides mechanisms for requesting a renewable ticket and certain properties for the renewal.

A Ticket includes the following high-level information:

- ticket flags
- username
- realm
- service name
- client IP address

- timestamp
- ticket lifetime
- session key

Early on while researching the low-level details of Kerberos, I was frustrated by the lack of detail about exactly what fields are in each message given in the usual hits at the top of Google on the topic. Obviously, RFC 4120 has this information, but most of the parameters are optional. So, looking at functioning examples of the Kerberos protocol is the most viable way of obtaining the details. In subsequent posts, we will look at real world examples of Kerberos and the message details. So, in this post, we will stick with the high-level details.

Authenticator

An authenticator is information sent along with a ticket in a Kerberos request to prove that the message originated with the principal to whom the ticket was issued. The authenticator is encrypted with the session key (Client-to-TGS session key or Client-to-Server (Service) Session Key); this proves that the authenticator was generated by a party that has the session key (which should be the the principal described in the ticket). The authenticator also contains a timestamp to prove that the message was generated recently and not intercepted and replayed.

The authenticator contains the following details:

- Timestamp
- client ID
- application-specific checksum
- initial sequence number KRB_SAFE or KRB_PRIV messages)
- session sub-key (used in negotiations for a session key unique to this particular session)

Delegation

Sometimes, a client may need to grant a remote service the authority to act on the clients behalf to invoke other services. This is referred to as delegation. Delegation entails a service taking on the identity (or credentials) of a client that invoked it in order to perform another action such as calling another service (with the same access the original client had). The Kerberos v5 protocol defines two mechanisms for delegating credentials to a service: a proxy ticket or a forwarded TGT. Identity protocol mechanisms that dictate how an actor should allow another actor to securely use its credentials is a fascinating topic that I will address in much greater detail in a future post. In the meantime, note that Kerberos provides support for delegation through two mechanisms that are briefly described below.

Proxy Tickets

The first type of delegation that is defined by the Kerberos spec is called a Proxy Ticket. It must be possible to control exactly what actions the service is allowed to perform using the credentials it is provided. The Proxy Ticket allows the principal the ticket describes to dictate exactly what the service can do on its behalf. Or, to use terminology in the spec, the principal can grant a proxy to the service. This is done with the proxy and proxiable flags in the ticket. These flags allow the principal to provide credentials that can be used with specific services.

From the spec,

```
  The PROXIABLE flag in a ticket is normally only interpreted by the  ticket-
granting service.  It can be ignored by application   servers. When set, this flag
tells the ticket-granting server   that it is OK to issue a new ticket (but not a
TGT) with a   different network address based on this ticket.  This flag is set
if requested by the client on initial authentication.
```

So, the important step here is that a ticket is issued that references a network address other than the one where the client is known to be located. In particular, it has the address of the service that is allowed to use the delegated identity.

The spec provides for the application server being able to require the invoking service to provide information about itself for audit log purposes.

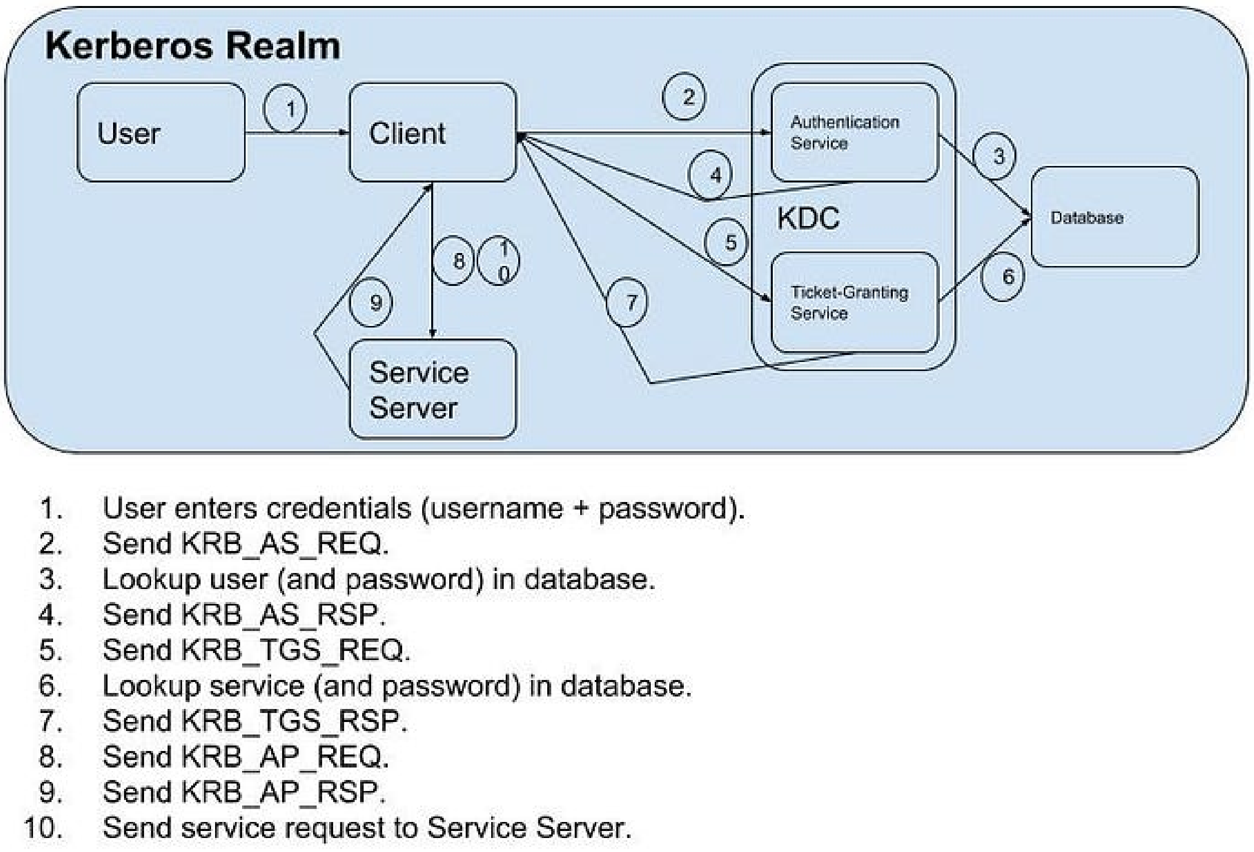
Forwardable Tickets

A forwardable ticket or authentication forwarding is a proxy ticket where the service that's allowed to use the proxy is given complete freedom to use the identity with any remote service the identity would otherwise be able to access.

This concept is implemented by issuing a ticket that doesn't reference any network address for where the request is allowed to come from.

Sign On & Service Access

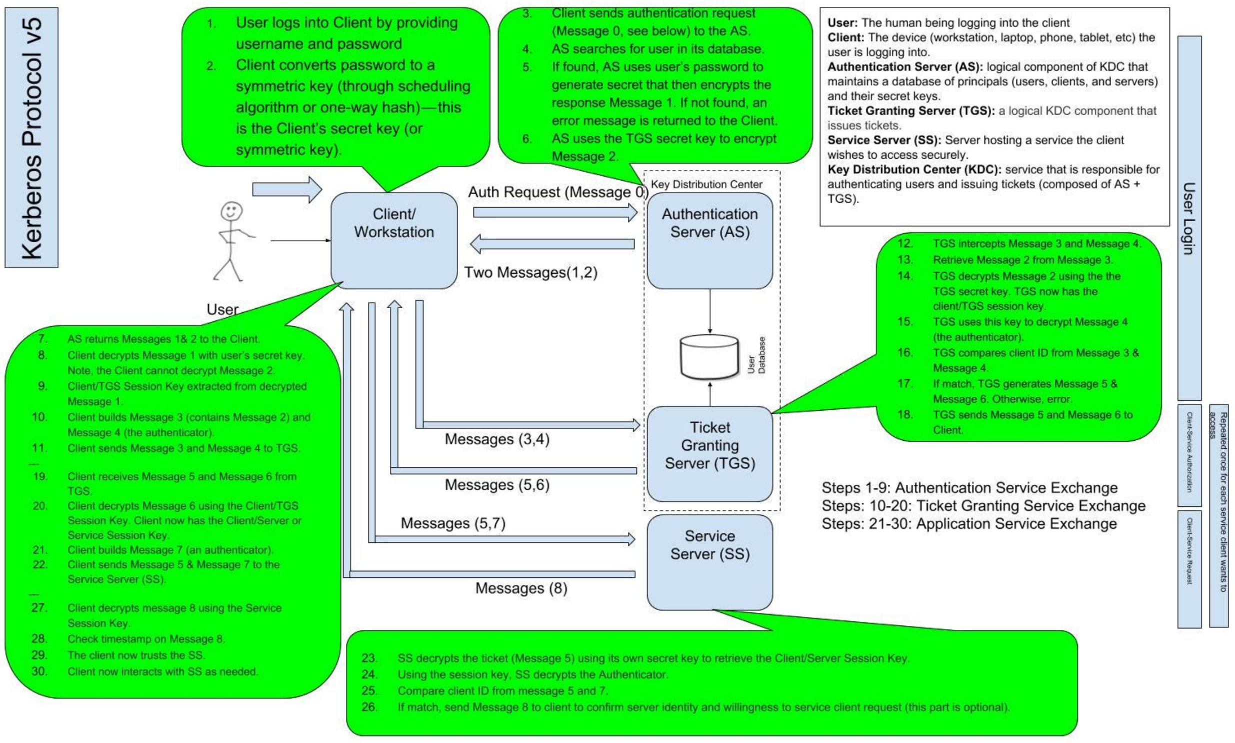
The following diagram shows the high-level interaction of Kerberos actors.



1. User enters credentials (username + password).
2. Send KRB_AS_REQ.
3. Lookup user (and password) in database.
4. Send KRB_AS_RSP.
5. Send KRB_TGS_REQ.
6. Lookup service (and password) in database.
7. Send KRB_TGS_RSP.
8. Send KRB_AP_REQ.
9. Send KRB_AP_RSP.
10. Send service request to Service Server.

Kerberos Protocol High-Level

The following diagram describes the detailed exchanged of messages between the actors.



1. User logs into Client by providing username and password
2. Client converts password to a symmetric key (through scheduling algorithm or one-way hash)—this is the Client's secret key (or symmetric key).

3. Client sends authentication request (Message 0, see below) to the AS.
4. AS searches for user in its database.
5. If found, AS uses user's password to generate secret that then encrypts the response Message 1. If not found, an error message is returned to the Client.
6. AS uses the TGS secret key to encrypt Message 2.

User: The human being logging into the client
Client: The device (workstation, laptop, phone, tablet, etc) the user is logging into.
Authentication Server (AS): logical component of KDC that maintains a database of principals (users, clients, and servers) and their secret keys.
Ticket Granting Server (TGS): a logical KDC component that issues tickets.
Service Server (SS): Server hosting a service the client wishes to access securely.
Key Distribution Center (KDC): service that is responsible for authenticating users and issuing tickets (composed of AS + TGS).

7. AS returns Messages 1& 2 to the Client.
8. Client decrypts Message 1 with user's secret key. Note, the Client cannot decrypt Message 2.
9. Client/TGS Session Key extracted from decrypted Message 1.
10. Client builds Message 3 (contains Message 2) and Message 4 (the authenticator).
11. Client sends Message 3 and Message 4 to TGS.

—

19. Client receives Message 5 and Message 6 from TGS.
20. Client decrypts Message 6 using the Client/TGS Session Key. Client now has the Client/Server or Service Session Key.
21. Client builds Message 7 (an authenticator).
22. Client sends Message 5 & Message 7 to the Service Server (SS).

—

27. Client decrypts message 8 using the Service Session Key.
28. Check timestamp on Message 8.
29. The client now trusts the SS.
30. Client now interacts with SS as needed.

12. TGS intercepts Message 3 and Message 4.
13. Retrieve Message 2 from Message 3.
14. TGS decrypts Message 2 using the the TGS secret key. TGS now has the client/TGS session key.
15. TGS uses this key to decrypt Message 4 (the authenticator).
16. TGS compares client ID from Message 3 & Message 4.
17. If match, TGS generates Message 5 & Message 6. Otherwise, error.
18. TGS sends Message 5 and Message 6 to Client.

Steps 1-9: Authentication Service Exchange
Steps: 10-20: Ticket Granting Service Exchange
Steps: 21-30: Application Service Exchange

23. SS decrypts the ticket (Message 5) using its own secret key to retrieve the Client/Server Session Key.
24. Using the session key, SS decrypts the Authenticator.
25. Compare client ID from message 5 and 7.
26. If match, send Message 8 to client to confirm server identity and willingness to service client request (this part is optional).

Kerberos Authentication Protocol — Detailed View

Authentication Service Exchange

These steps authenticate the user.

1. User logs into Client by providing username and password
2. Client converts password to a symmetric key (through scheduling algorithm or one-way hash) — this is the Client's secret key (or symmetric key).
3. Client sends authentication request (Message 0, see below) to the AS.
4. AS searches for user in its database.
5. If found, AS uses user's password to generate secret that then encrypts the response Message 1. If not found, an error message is returned to the Client.
6. AS uses the TGS secret key to encrypt Message 2.
7. AS returns Messages 1 & 2 to the Client.
8. Client decrypts Message 1 with user's secret key. Note, the Client cannot decrypt Message 2.
9. Client/TGS Session Key extracted from decrypted Message 1.

Some notes of interest:

- At this point, the password the user originally entered is validated and the user can be considered authenticated.
- From , "This exchange is typically used at the initiation of a login session to obtain credentials for a Ticket-Granting Server, which will subsequently be used to obtain credentials for other servers (see Section 3.3) without requiring further use of the client's secret key."
- From , "Without pre-authentication, the authentication server does not know whether the client is actually the principal named in the request. It simply sends a reply without knowing or caring whether they are the same. This is acceptable because nobody but the principal whose identity was given in the request will be able to use the reply."
- Unless otherwise noted, the information is passed in clear text.
- The Principal's secret key is not cached.

Ticket Granting Service Exchange

1. Client builds Message 3 (contains Message 2) and Message 4 (the authenticator).
2. Client sends Message 3 and Message 4 to TGS.
3. TGS intercepts Message 3 and Message 4.
4. Retrieve Message 2 from Message 3.
5. TGS decrypts Message 2 using the the TGS secret key. TGS now has the client/TGS session key.
6. TGS uses this key to decrypt Message 4 (the authenticator).
7. TGS compares client ID from Message 3 & Message 4.
8. If match, TGS generates Message 5 & Message 6. Otherwise, error.
9. TGS sends Message 5 and Message 6 to Client.
10. Client builds Message 3 and Message 4.
11. Client sends Message 3 and Message 4 to TGS.
12. Client receives Message 5 and Message 6 from TGS.

13. Client decrypts Message 6 using the Client/TGS Session Key. Client now has the Client/Server or Service Session Key.

Some notes of interest:

- The TGT is cached for its lifetime and used to request each new Service Ticket that is needed by the Client.
- The Client-to-TGS Session Key is also cached by the client.

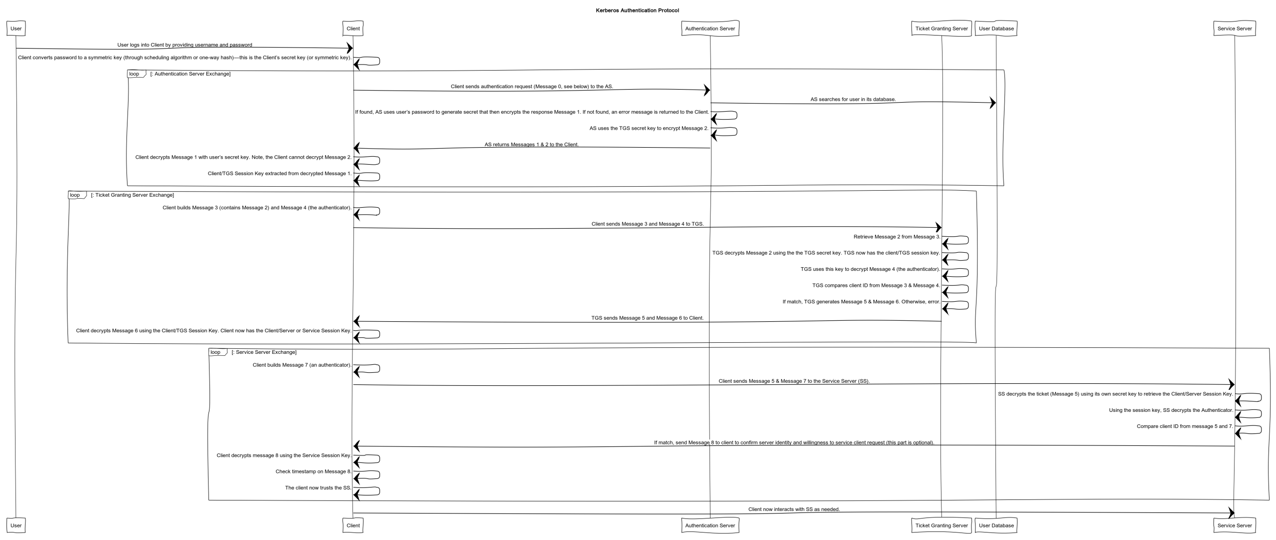
Application Service Exchange

1. Client builds Message 7 (an authenticator).
2. Client sends Message 5 & Message 7 to the Service Server (SS).
3. SS decrypts the ticket (Message 5) using its own secret key to retrieve the Client/Server Session Key.
4. Using the session key, SS decrypts the Authenticator.
5. Compare client ID from message 5 and 7.
6. If match, send Message 8 to client to confirm server identity and willingness to service client request (this part is optional).
7. Client decrypts message 8 using the Service Session Key.
8. Check timestamp on Message 8.
9. The client now trusts the SS.
10. Client now interacts with SS as needed.

Some notes of interest:

> The response from the SS, Message 8, is optional. That is only required if the client wants the SS to authenticate itself (ie, mutual authentication is required. It is interesting to note that the authentication component of TLS considers mutual authentication to include client authentication (the opposite of Kerberos).

The same steps are presented in the following sequence diagram.


Kerberos Authentication Protocol
User
Client
Authentication Server
Ticket Granting Server
User Database
Service Server
User logs into Client by providing username and password
Client converts password to a symmetric key (through scheduling algorithm or one-way hash)—this is the Client's secret key (or symmetric key).
loop [: Authentication Server Exchange]
Client sends authentication request (Message 0, see below) to the AS.
AS searches for user in its database.
If found, AS uses user's password to generate secret that then encrypts the response Message 1. If not found, an error message is returned to the Client.
AS uses the TGS secret key to encrypt Message 2.
AS returns Messages 1 & 2 to the Client.
Client decrypts Message 1 with user's secret key. Note, the Client cannot decrypt Message 2.
Client/TGS Session Key extracted from decrypted Message 1.
loop [: Ticket Granting Server Exchange]
Client builds Message 3 (contains Message 2) and Message 4 (the authenticator).
Client sends Message 3 and Message 4 to TGS.
Retrieve Message 2 from Message 3.
TGS decrypts Message 2 using the the TGS secret key. TGS now has the client/TGS session key.
TGS uses this key to decrypt Message 4 (the authenticator).
TGS compares client ID from Message 3 & Message 4.
If match, TGS generates Message 5 & Message 6. Otherwise, error.
TGS sends Message 5 and Message 6 to Client.
Client decrypts Message 6 using the Client/TGS Session Key. Client now has the Client/Server or Service Session Key.
loop [: Service Server Exchange]
Client builds Message 7 (an authenticator).
Client sends Message 5 & Message 7 to the Service Server (SS).
SS decrypts the ticket (Message 5) using its own secret key to retrieve the Client/Server Session Key.
Using the session key, SS decrypts the Authenticator.
Compare client ID from message 5 and 7.
If match, send Message 8 to client to confirm server identity and willingness to service client request (this part is optional).
Client decrypts message 8 using the Service Session Key.
Check timestamp on Message 8.
The client now trusts the SS.
Client now interacts with SS as needed.
User
Client
Authentication Server
Ticket Granting Server
User Database
Service Server

Kerberos Authentication Protocol Sequence Diagram

Message Formats

The following describes all fields in the Kerberos v5 message formats used in the diagrams and description above.

Message 0: KRB_AS_REQ (see Section 5.4.1 of RFC 4120)

- Pre-authentication data (if used) — in Windows, this will contain an encrypted timestamp
- Client principal name
- Client address
- Realm name
- Nonce (generated randomly by client, used to detect replays and match up the response with the request)
- Requested message expiration time.
- Requested renew-till time (optional, used if requesting a renewal).
- Requested post-dated time (optional, used for post-dated ticket requests).
- Desired encryption type (optional)
- Options (see of) :

> Is this an initial ticket (ie, were credentials presented)?
>
> Is pre-authentication performed?
>
> Is requested ticket renewable?
>
> Is requested ticket proxiable?
>
> Is requested ticket forwardable?
>
> Postdating allowed?
>
> Postdating of derivative tickets allowed?
>
> Renewable ticket allowed in lieu of a non-renewable ticket

KRB_AS_REP (see section 5.4.2 of RFC 4120)

Various meta-data (see examples in subsequent posts)

Message 1: KRB_AS_REP (encrypted with the client's secret key)

- TGS name
- Timestamp
- Lifetime
- TGS Session Key

Message 2: KRB_AS_REP (TGT, encrypted with TGS's secret key)

- Username
- TGS Name
- Timestamp
- Client IP address
- Ticket lifetime
- TGS Session Key

KRB_TGS_REQ

Meta-data

Message 3: KRB_TGS_REQ

- Name of the requested service.
- TGT (Message 2 from above)

Message 4: KRB_TGS_REQ

Authenticator (see earlier description)

KRB_TGS_REP

Meta-data

Message 5: (Service Ticket, encrypted with Service Secret Key)

- Username
- Service Name
- Timestamp
- Client IP address
- Ticket lifetime
- Client-to-Server (or Service) Session Key

Message 6: KRB_TGS_REP (encrypted with Client-TGS Session Key)

Service Session Key

KRB_AP_REQ

Meta-data

Message 5: KRB_TGS_REP (Service Ticket, encrypted with Service Secret Key)

- Username
- Service Name
- Timestamp
- Client IP address
- Ticket lifetime

- Client-to-Server (or Service) Session Key

Message 7: KRB_AP_REQ

Authenticator

KRB_AP_RES

Meta-data

Message 8: KRB_AP_REP

Timestamp from authenticator (encrypted with Client-Server (Service) Session key. Session Key)

There are many other details and edge cases in the base Kerberos spec and even more in the family of extensions, RFCs, and proprietary features that have been added in the thirty years since its initial publication. We will continue to look at these throughout this series.

Image: / Cerberus Greek Monsters by beetroot