September 17: Kerberos

Scribe: Rishabh Poddar

Kerberos is an authentication system in an open network computing environment, where a workstation cannot be trusted to identify its users correctly to network services. In such an environment, Kerberos runs as a trusted third-party service to verify the identities of the users, using credentials-based authentication (as opposed to, say, simple IP-based authentication which was used by services such as rlogin and rsh).

1 Overview

Consider a network consisting of multiple servers providing various services, such as mail, printing, remote login, etc. The network also includes multiple workstations that are either public or private. Users log into their workstations and use the network services; however, not all services are available to everyone, and services are under access control.

Goal. The goal of Kerberos is to authenticate users to services and vice versa, in the setting described above.

Threat model. An adversary in the network may attempt to take control of a workstation, or even a service. The adversary may also intercept communication in the network, and read or forge messages.

Trust model. The third-party Kerberos server is trusted. Neither the network nor any other machine in the network is trusted.

2 Architecture

Kerberos maintains a database of all clients and services in the network, along with their private keys. The private key of a client is derived from a password using, effectively, a hash function. Network services that require authentication, and clients wishing to use those services, register with Kerberos during which the private keys are negotiated and stored in the database. Kerberos uses these keys to achieve mutual authentication between users and services. Thus, users do not need to set up accounts and passwords on each individual server providing a network service. Note that Kerberos does not provide *authorization*, i.e. whether or not a user can access a particular resource. Authorization is the services' prerogative.

Two interfaces are provided to the database: the Kerberos authentication server, and the *ticket granting server*. These two servers may or may not be located on the same machine. The architecture diagram of the system is shown in Figure 1.

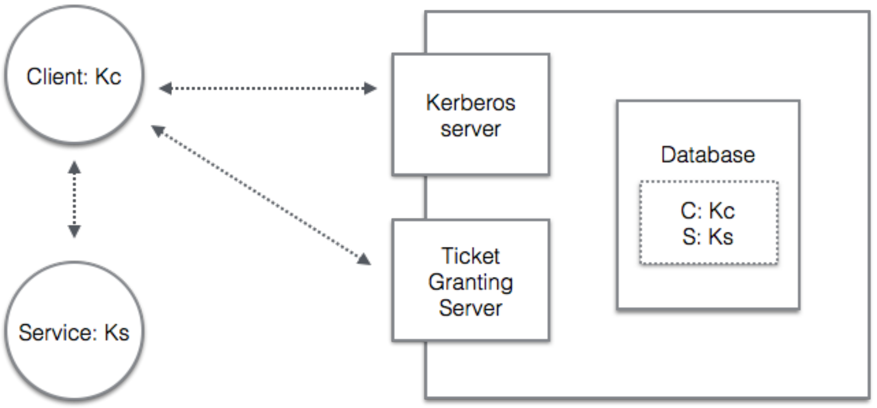


Figure 1: **Kerberos architecture.** c and s refer to the client and service respectively; K_c is the private key of the client, and K_s is the private key of the service.

The authentication server grants the client an initial *ticket* for the ticket granting server. The ticket granting server then issues tickets to the client for network services. Thus, the client doesn't need to store the user's password on the workstation, and can delete it once it receives the initial ticket.

3 Names in Kerberos

In Kerberos, the principals (i.e. the users and services) are named. Each name consists of three parts: a primary name, an instance, and a realm, expressed as *name.instance@realm* (e.g. alice.root@berkeley.edu). The names are used to identify the users and services, and the process of authentication entails verifying that the client is the one named in a request.

In the rest of this document, we use c to refer to the name of the client, and s to refer to the name of the service requested by c.

4 How Kerberos works

A client that requests a network service needs to establish its identity to the service by authenticating itself through Kerberos. To do this, it first obtains credentials to be used in service requests. Next, it requests authentication for a specific service. Finally, it presents the credentials to the requested service.

4.1 Credentials

There are two types of credentials that are used in the authentication process: *tickets* and *authenticators*. Tickets are issued to clients by the authentication server, and are used to securely pass the identity of the client to the end service. Specifically, a ticket $T_{c,s}$ used to identify a client c to a service s, contains the following information:

$$T_{c,s} = \{s, c, \text{addr}, \text{timestamp}, \text{life}, K_{s,c}\}K_s$$

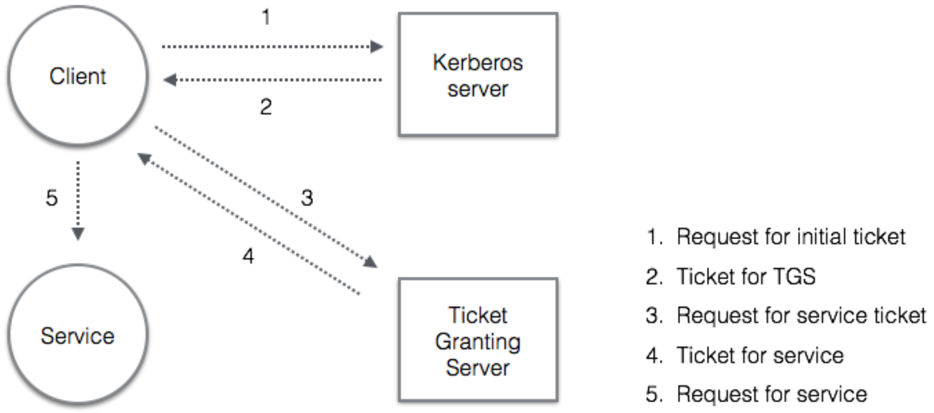


Figure 2: Kerberos authentication protocol

where addr is the client's network address; life refers to the lifetime of the ticket (i.e. validity period); and $K_{s,c}$ is the session key for s and c, i.e. the key with which the communication between s and c will be encrypted. Further, the tuple $< s, c, \text{addr}, \text{timestamp}, \text{life}, K_{s,c} >$ is encrypted with K_s, the private key of s. (We use the notation $\{M\}K$ to mean that a message M is encrypted with key K.) Once a ticket is issued, it may be used multiple times by the client until it expires, based on its lifetime. Note that since the ticket is encrypted with the private key of s, it is safe to assume that the client cannot tamper with its contents.

An authenticator $A_{c,s}$ contains additional information which, when compared against the ticket, proves to s that the client presenting the ticket is the same one to which the ticket was issued. Unlike tickets, authenticators can be used only once, and a new authenticator must be generated each time a client wants to use a service. Specifically, an authenticator contains the following information:

$$A_{c,s} = \{c, \text{addr}, \text{ timestamp}\}K_{c,s}$$

i.e. an authenticator consists of the tuple $< c, \text{addr}, \text{ timestamp} >$ encrypted with the session key $K_{c,s}$, where the timestamp is the current timestamp at the client's workstation. Note that the authenticator can be built by the client once it has the session key.

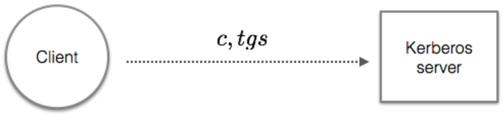
The service decrypts $T_{c,s}$, uses the session key $K_{c,s}$ included in the ticket to decrypt the authenticator $A_{c,s}$, compares the information in the ticket with the authenticator, the IP address from which the request was received, and the current time. If everything matches, it allows the request to proceed. Note that this mechanism also safeguards an adversary that captures an old ticket and attempts to mount a ticket replay attack, because the service checks the timestamp of the authenticator (indicating the current time of use) against its own time. (However, this safeguard depends on how well-synchronized the system times are.)

4.2 Kerberos authentication protocol

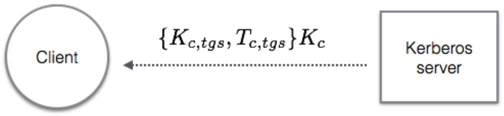
The authentication protocol consists of five steps (see Figure 2). The client first obtains an initial ticket from the authentication server in steps 1 and 2. It uses the initial ticket to request a service, and obtains a service ticket in steps 3 and 4 from the ticket granting server. Finally, it authenticates itself to the service in step 5 using the service ticket.

4.2.1 Getting the initial ticket

The client first sends a request to the authentication server (step 1) consisting of its name c, and the name of the ticket granting server tgs.



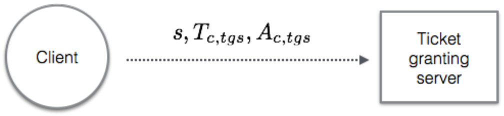
The authentication server looks up c in the database, and verifies that it knows about the client. It then generates a random session key $K_{c,tgs}$ which will be used to encrypt the communication between c and tgs. Next, it creates the initial ticket $T_{c,tgs}$ and encrypts it with K_{tgs}. The session key along with the initial ticket are then encrypted with K_c and returned to the client (step 2).



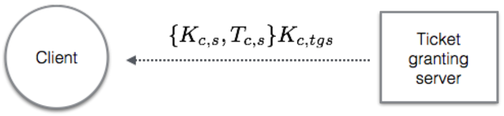
Note that the authentication server does not verify the identity of c. Since the response is encrypted with K_c, no one but c will be able to decrypt it. This also serves as a means of authenticating the Kerberos server to the client.

4.2.2 Getting the service ticket

The client now sends a request to the ticket granting server (step 3), to request a ticket for service s. The request contains the name of the service, the initial ticket $T_{c,tgs}$, and an authenticator $A_{c,tgs}$.

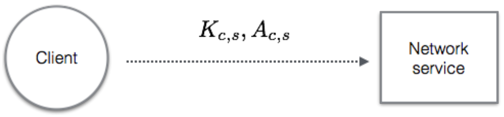


The ticket granting server uses $T_{c,tgs}$ and $A_{c,tgs}$ to authenticate the client c as described in § 4.1. If valid, it generates a new random session key $K_{c,s}$ to be used between c and s, and crafts a new ticket $T_{c,s}$ for the service. It then returns the new session key and ticket to c, encrypting the information with $K_{c,tgs}$ (step 4).



4.2.3 Requesting the service

The client finally uses the service ticket $T_{c,s}$ along with an authenticator $A_{c,s}$ to request the service s (step 5). Authentication of the client by s is done in the manner described earlier.



5 General weaknesses

DES encryption. Each step in the authentication protocol of Kerberos is based on symmetric encryption, which was originally DES. DES, however, is not secure anymore, as its 56-bit key size is too small and can easily be broken with current systems. Moreover, Kerberos assumed that encryption provides message integrity. However, DES does not provide unforgeability guarantees (as opposed to a MAC), i.e. it is possible for an adversary to forge a valid ciphertext corresponding to some plaintext.

Password-guessing attacks. Since the private key K_c of a client is derived from the password, an adversary can attempt to mount an offline dictionary attack to guess the user's password (and hence private key). This can be fixed by using better alternatives such as SRP and PAKE.

Persistent session keys. The session key $K_{c,s}$ is used for all communication between c and s over a long period of time. This has been fixed in Kerberos v5, where fresh session keys are used each time. This is accomplished by keeping the original $K_{c,s}$ locally at c an s, and using it to encrypt temporary session keys.

6 Using Kerberos

Network services register with Kerberos, run Kerberos libraries and choose a private key.

From the perspective of users, Kerberos is transparent except when tickets expire. After a ticket has expired, the user needs to run `kinit` and re-enter the password. A user that wishes to change her password can do so easily by using a password-changing service. Recovering a lost password requires using a new trusted service.

7 Replication

For the purposes of high availability and performance, multiple copies of the Kerberos system are maintained. In addition to a *master* server which houses the main copy of the authentication database, multiple copies of the database are kept on *slave* machines. Hence, if the master server is down, a slave can still be used for the purposes of authentication.

In the original version of Kerberos these slaves were cold replicas of the master. This means that they were updated with the master's copy of the database only at hourly intervals. (As a result, when the master was down, it was possible for a revoked user to continue using the system if the update had not been copied to the slaves. Similarly, an attacker could use a user's old compromised password even if the user changed the password within the last hour.) Moreover, these slaves were read-only copies, enabling the system to avoid complicated consistency problems.