to: Beth Kevles
from: Dan Geer
date: 24 Jan 1990
re: Brief on how Kerberos works

---

1. Definitions

Accounting	Measuring resource usage attributable to a particular client.
Authentication	Verifying the claimed identity of a client or service.
Authorization	Allowing an authenticated client to use a particular service.
Client	A program that makes use of a network service, on behalf of a user.
Kerberos	aside from the 3-headed dog guarding Hades, the name given to the Athena authentication service, the protocol used by that service, and the libraries used to invoke the authentication and authorization services.
Principal	A uniquely named client or server instance that participates in a network communication. *(The term "principal," being somewhat formal, is replaced with the word "user" in this document wherever the context permits that usage without confusion.)*
Principal identifier	The name used to uniquely identify each different client and server.
Private key	An encryption key between a principal and the KKDS, distributed outside the system, with a long lifetime;
Seal	To encipher a record containing several fields, in such a way that the fields cannot be individually replaced without either knowledge of the key or leaving evidence of tampering.
Session key	A temporary encryption key used between two principals, with a lifetime limited to the duration of a single communications "session".

Ticket	A record that authenticates a client to a service; it contains the clients' identity, a session key, and a timestamp, all of which is sealed by encryption using the services' private key.

2. Introduction to Kerberos

Most conventional time-sharing systems require a prospective user to identify him or herself and to authenticate that identity before using its services. In an environment consisting of a network that connects prospective clients with services, a network service has a corresponding need to identify and authenticate its clients. When the client is a user of a time-sharing system, one approach is for the service to trust the authentication that was performed by the time-sharing system. For example, the network applications *lpr* and *rcp* provided with Berkeley 4.3 UNIX trust the user's time-sharing system to reliably authenticate its clients.

In contrast with the time-sharing system, in which a protection wall separates the operating system from its users, a workstation is under the complete control of its user, to the extent that the user can run a private version of the operating system, or even replace the machine itself. As a result, a network service cannot rely on the integrity of the workstation operating system when it (the network service) performs authentication.

Kerberos extends the conventional notions of authentication, authorization, and accounting to the network environment with untrusted workstations. It establishes a trusted third-party service that can perform authentication to the mutual satisfaction of both clients and services. The authentication approach allows for integration with authorization and accounting facilities. The resulting design is also applicable to a mixed time-sharing/network environment in which a network service is not willing to rely on the authentication performed by the client's time-sharing system.

3. Goals of Kerberos

• Authentication - Authentication is not an end in itself, but rather a tool to support both integrity and authorization. Its basic purpose is to prevent fraudulent connection requests. The goal of Kerberos is to support both one-way and mutual authentication of principals, to the granularity of at least an individual user and specific service instance.

• Authorization - Authentication can imply a coarse-grained authorization - for example, some services may allow anyone who can be reliably authenticated by the local Kerberos to use the service. In cases where more selective authorization is needed, the goal of Kerberos is to allow different services to implement different authorization models, and to allow those authorization models to assume that authentication of user identities is reliable.

• Accounting - Given an authenticated client, the goal of accounting is to support either quotas charged against the client (to limit consumption), e.g. disk quota, and/or charges based on consumption, e.g. $.01 per page printed. The goal of Kerberos is to permit modular attachment of an integrated, secure, reliable accounting system.

4. The Kerberos Authentication Model

In response to the requirements and assumptions sketched above, this section describes the Athena Kerberos model for authentication and authorization, with provision for accounting. This model is based on the Needham and Schroeder key distribution protocols, modified with the addition of timestamps. Their paper (listed in the References section) describes the basic protocol; a tutorial paper by Voydock and Kent provides a broader introduction to the topic and explains the timestamp modifications.

The basic approach for Kerberos authentication is the following: to use a service, a client must supply a *ticket* previously obtained from Kerberos. A ticket for a service is a string of bits

with the property that it has been enciphered using the private key for that service. That private key is known only to the service itself and to Kerberos. As a result of that property, the service can be confident that any information found inside the ticket originated from Kerberos. As will be seen, Kerberos will have placed the identity of the client inside the ticket, so the service that receives a ticket has a Kerberos-authenticated opinion of the identity of the client. To help insure that one user does not steal and reuse another user's tickets, the client accompanies the ticket with an authenticator, explained later. (In addition, tickets expire after a specified lifetime, which is usually on the order of several hours.)

The client obtains a ticket by sending a message to Kerberos naming the principal identifier of the desired service, the principal identifier of the (alleged) client, and mentioning the current time of day. Anyone could send such a message or intercept its response; that response, however, is usable only to the client named in the original request, because Kerberos seals the response by enciphering it in the private key of that client. The response contains three parts: the ticket (which itself is further sealed in the private key of the service), a newly-minted key for use in this client-server session, and a copy of the timestamp that was in the original request.

A legitimate user will be able to unseal this message, obtain the ticket and session key, and verify that the timestamp is current (Thereby preventing replays of old responses). No other user, without the named user's private key, can correctly decrypt the reply to produce the sealed tickets and corresponding session key.

Once a client obtains a ticket and sends it to a service, and the service has identified the client, further use of the fact of authentication is specific to the protocol of the service. One application might use the session key (Kerberos seals a copy in the ticket) for secure end-to-end encryption, while at the other extreme, another application might throw everything but the

source network address away and assume that all further requests coming on the connection from this particular network address are from the same user.

The *authenticator* mentioned above is a simple mechanism designed to discourage attempts at unauthorized reuse ("replay") of tickets by someone who notices a ticket going by on the network and makes a copy. The authenticator consists of, among other things, the client's principal identifier, network address, and the current time of day all sealed with the key that Kerberos minted for this session. After the service decrypts the ticket, it uses the session key found in that ticket to decrypt the authenticator. If the principal identifier of the authenticator matches the one in the ticket, the network address in the authenticator is the same as the one that sent the packet, and the time in the authenticator is within the last few minutes, the authenticator is probably not a replay, and the service accepts the associated ticket. It is because authenticators expire in a short time that all the clients and servers in a Kerberos realm need to have their clocks loosely synchronized.

If a private key is compromised, another party may successfully pose as the principal until the private key is changed and all tickets previously issued under it expire. If a session key is compromised, another party may successfully pose as the principal until the previously issued tickets expire.

One more mechanism rounds out the complete Kerberos scenario. If a client uses several services, a distinct ticket is needed for each. Not all the services to be used may be known at the beginning of a login session, but that is when the user provides the password used as a private key to decrypt tickets. To avoid storing the private key in the workstation memory for the entire duration of the session, at login time the user obtains a single ticket, useful only for a service provided by Kerberos itself, the ticket-granting service. Whenever the client goes back to Ker-

beros for an additional, service-specific ticket, the response is actually enciphered in the session key of the ticket-granting service. Thus the private key is needed only for the initial ticket, and the workstation software can immediately destroy its copy of that private key after that single use.

5. User Interface

For the most part, Kerberos is designed to operate under the covers, without separate actions by the user. For network applications that make use of Kerberos authentication there is a library of Kerberos functions that simplify the obtaining of authentication. The primary interface consists of three generic user commands and two generic subroutines that are used by applications.

- User command **kinit**: This command asks the user for a password, obtains a ticket-granting ticket, and destroys the password as soon as it has stored the ticket-granting ticket and associated session key. Note that the function of this command may be combined with the **login** command.
- User command **klist**: Displays the list of tickets obtained so far in this login session.
- User command **kdestroy**: Destroys all tickets. The function of this command may be combined with the **logout** command.

6. User-Initiated Password Change

The basic scenario for changing a password is that the user does it him or herself by invoking the password-changing program at a workstation. This program demands the old and new passwords, uses the old password to create a completely encrypted session with the master Kerberos server, and sends the new password on the encrypted connection. If the user has reason to believe that the old password is so badly compromised that it is not safe to send the new password this way, the user may appeal to the system manager to install a new password.