Kerberos: An Authentication Service for Open Network Systems

Jennifer G. Steiner

Project Athena
Massachusetts Institute of Technology
Cambridge, MA 02139
steiner@ATHENA.MIT.EDU

Clifford Neuman†

Department of Computer Science, FR-35
University of Washington
Seattle, WA 98195
bcn@CS.WASHINGTON.EDU

Jeffrey I. Schiller

Project Athena
Massachusetts Institute of Technology
Cambridge, MA 02139
jis@ATHENA.MIT.EDU

Introduction

This paper gives an overview of *Kerberos*, an authentication system designed by Miller and Neuman[1] for open network computing environments, and describes our experience using it at MIT's Project Athena.[2] In the first section of the paper, we explain why a new authentication model is needed for open networks, and what its requirements are. The second section lists the components of the *Kerberos* software and describes how they interact in providing the authentication service. In Section 3, we describe the *Kerberos* naming scheme.

Section 4 presents the building blocks of *Kerberos* authentication – the *ticket* and the *authenticator*. This leads to a discussion of the two authentication protocols: the initial authentication of a user to *Kerberos* (analogous to logging in), and the protocol for mutual authentication of a potential consumer and a potential producer of a network service.

Kerberos requires a database of information about its clients; Section 5 describes the database, its management, and the protocol for its modification. Section 6 describes the *Kerberos* interface to its users, applications programmers, and administrators. In Section 7, we describe how the Project Athena *Kerberos* fits into the rest of the Athena environment. We also describe the interaction of different *Kerberos* authentication domains, or *realms*; in our case, the relation between the Project Athena *Kerberos* and the *Kerberos* running at MIT's Laboratory for Computer Science.

In Section 8, we mention open issues and problems as yet unsolved. The last section gives

† Clifford Neuman was a member of the Project Athena staff during the design and initial implementation phase of *Kerberos*.

the current status of *Kerberos* at Project Athena. In the appendix, we describe in detail how *Kerberos* is applied to a network file service to authenticate users who wish to gain access to remote file systems.

Conventions. Throughout this paper we use terms that may be ambiguous, new to the reader, or used differently elsewhere. Below we state our use of those terms.

User, Client, Server. By *user*, we mean a human being who uses a program or service. A *client* also uses something, but is not necessarily a person; it can be a program. Often network applications consist of two parts; one program which runs on one machine and requests a remote service, and another program which runs on the remote machine and performs that service. We call those the *client* side and *server* side of the application, respectively. Often, a *client* will contact a *server* on behalf of a *user*.

Each entity that uses the *Kerberos* system, be it a user or a network server, is in one sense a client, since it uses the *Kerberos* service. So to distinguish *Kerberos* clients from clients of other services, we use the term *principal* to indicate such an entity. Note that a *Kerberos* principal can be either a user or a server. (We describe the naming of *Kerberos* principals in a later section.)

Service vs. Server. We use *service* as an abstract specification of some actions to be performed. A process which performs those actions is called a *server*. At a given time, there may be several *servers* (usually running on different machines) performing a given *service*. For example, at Athena there is one BSD UNIX *rlogin* server running on each of our timesharing machines.

Key, Private Key, Password. Kerberos uses private key encryption. Each *Kerberos* principal is assigned a large number, its private key, known only to that principal and *Kerberos*. In the case of a user, the private key is the result of a one-way function applied to the user's *password*. We use *key* as shorthand for *private key*.

Credentials. Unfortunately, this word has a special meaning for both the Sun Network File System and the *Kerberos* system. We explicitly state whether we mean NFS credentials or *Kerberos* credentials, otherwise the term is used in the normal English language sense.

Master and Slave. It is possible to run *Kerberos* authentication software on more than one machine. However, there is always only one definitive copy of the *Kerberos* database. The machine which houses this database is called the *master* machine, or just the *master*. Other machines may possess read-only copies of the *Kerberos* database, and these are called *slaves*.

1. Motivation

In a non-networked personal computing environment, resources and information can be protected by physically securing the personal computer. In a timesharing computing environment, the operating system protects users from one another and controls resources. In order to determine what each user is able to read or modify, it is necessary for the timesharing system to identify each user. This is accomplished when the user logs in.

In a network of users requiring services from many separate computers, there are three approaches one can take to access control: One can do nothing, relying on the machine to which the user is logged in to prevent unauthorized access; one can require the host to prove its identity, but trust the host's word as to who the user is; or one can require the user to prove her/his identity for each required service.

In a closed environment where all the machines are under strict control, one can use the first approach. When the organization controls all the hosts communicating over the network, this is a reasonable approach.

In a more open environment, one might selectively trust only those hosts under organizational control. In this case, each host must be required to prove its identity. The *rlogin* and *rsh* programs use this approach. In those protocols, authentication is done by checking the Internet address from which a connection has been established.

In the Athena environment, we must be able to honor requests from hosts that are not under organizational control. Users have complete control of their workstations: they can reboot them, bring them up standalone, or even boot off their own tapes. As such, the third approach must be taken; the user must prove her/his identity for each desired service. The server must also prove its identity. It is not sufficient to physically secure the host running a network server; someone elsewhere on the network may be masquerading as the given server.

Our environment places several

requirements on an identification mechanism. First, it must be secure. Circumventing it must be difficult enough that a potential attacker does not find the authentication mechanism to be the weak link. Someone watching the network should not be able to obtain the information necessary to impersonate another user. Second, it must be reliable. Access to many services will depend on the authentication service. If it is not reliable, the system of services as a whole will not be. Third, it should be transparent. Ideally, the user should not be aware of authentication taking place. Finally, it should be scalable. Many systems can communicate with Athena hosts. Not all of these will support our mechanism, but software should not break if they did.

Kerberos is the result of our work to satisfy the above requirements. When a user walks up to a workstation s/he ''logs in''. As far as the user can tell, this initial identification is sufficient to prove her/his identity to all the required network servers for the duration of the login session. The security of *Kerberos* relies on the security of several authentication servers, but not on the system from which users log in, nor on the security of the end servers that will be used. The authentication server provides a properly authenticated user with a way to prove her/his identity to servers scattered across the network.

Authentication is a fundamental building block for a secure networked environment. If, for example, a server knows for certain the identity of a client, it can decide whether to provide the service, whether the user should be given special privileges, who should receive the bill for the service, and so forth. In other words, authorization and accounting schemes can be built on top of the authentication that *Kerberos* provides, resulting in equivalent security to the lone personal computer or the timesharing system.

2. What is *Kerberos*?

Kerberos is a trusted third-party authentication service based on the model presented by Needham and Schroeder.[3] It is trusted in the sense that each of its clients believes *Kerberos'* judgement as to the identity of each of its other clients to be accurate. Timestamps (large numbers representing the current date and time) have been added to the original model to aid in the detection of *replay*. Replay occurs when a message is stolen off the network and resent later. For a more complete description of replay, and other issues of authentication, see Voydock and Kent.[4]

2.1. What Does It Do?

Kerberos keeps a database of its clients and their *private keys*. The private key is a large number known only to *Kerberos* and the client it belongs to. In the case that the client is a user, it is an encrypted password. Network services requiring authentication register with *Kerberos*, as do clients wishing to use those services. The private keys are negotiated at registration.

Because *Kerberos* knows these private keys, it can create messages which convince one client that another is really who it claims to be. *Kerberos* also generates temporary private keys, called *session keys*, which are given to two clients and no one else. A session key can be used to encrypt messages between two parties.

Kerberos provides three distinct levels of protection. The application programmer determines which is appropriate, according to the requirements of the application. For example, some applications require only that authenticity be established at the initiation of a network connection, and can assume that further messages from a given network address originate from the authenticated party. Our authenticated network file system uses this level of security.

Other applications require authentication of each message, but do not care whether the content of the message is disclosed or not. For these, *Kerberos* provides *safe messages*. Yet a higher level of security is provided by *private messages*, where each message is not only authenticated, but also encrypted. Private messages are used, for example, by the *Kerberos* server itself for sending passwords over the network.

2.2. Software Components

The Athena implementation comprises several modules (see Figure 1). The *Kerberos* applications library provides an interface for application clients and application servers. It contains, among others, routines for creating or reading authentication requests, and the routines for creating safe or private messages.

- *Kerberos* applications library
- encryption library
- database library
- database administration programs
- administration server
- authentication server
- db propagation software
- user programs
- applications

Figure 1. *Kerberos* Software Components.

Encryption in *Kerberos* is based on DES, the Data Encryption Standard.[5] The encryption library implements those routines. Several methods of encryption are provided, with trade-offs between speed and security. An extension to the DES Cypher Block Chaining (CBC) mode, called the Propagating CBC mode, is also provided. In CBC, an error is propagated only through the current block of the cipher, whereas in PCBC, the error is propagated throughout the message. This renders the entire message useless if an error occurs, rather than just a portion of it. The encryption library is an independent module, and may be replaced with other DES implementations or a different encryption library.

Another replaceable module is the database management system. The current Athena implementation of the database library uses *ndbm*, although Ingres was originally used. Other database management libraries could be used as well.

The *Kerberos* database needs are straightforward; a record is held for each principal, containing the name, private key, and expiration date of the principal, along with some administrative information. (The expiration date is the date after which an entry is no longer valid. It is usually set to a few years into the future at registration.)

Other user information, such as real name, phone number, and so forth, is kept by another server, the *Hesiod* nameserver.[6] This way, sensitive information, namely passwords, can be handled by *Kerberos*, using fairly high security measures; while the non-sensitive information kept by *Hesiod* is dealt with differently; it can, for example, be sent unencrypted over the network.

The *Kerberos* servers use the database library, as do the tools for administering the database.

The *administration server* (or KDBM server) provides a read-write network interface to the database. The client side of the program may be run on any machine on the network. The server side, however, must run on the machine housing the *Kerberos* database in order to make changes to the database.

The *authentication server* (or *Kerberos* server), on the other hand, performs read-only operations on the *Kerberos* database, namely, the authentication of principals, and generation of session keys. Since this server does not modify the *Kerberos* database, it may run on a machine housing a read-only copy of the master *Kerberos* database.

Database propagation software manages replication of the *Kerberos* database. It is possible to have copies of the database on several different machines, with a copy of the authentication server running on each machine. Each of these *slave* machines receives an update of the *Kerberos* database from the *master* machine at given intervals.

Finally, there are end-user programs for logging in to *Kerberos*, changing a *Kerberos* password, and displaying or destroying *Kerberos tickets* (tickets are explained later on).

3. *Kerberos* Names

Part of authenticating an entity is naming it. The process of authentication is the verification that the client is the one named in a request. What does a name consist of? In *Kerberos*, both users and servers are named. As far as the authentication server is concerned, they are equivalent. A name consists of a primary name, an instance, and a realm, expressed as *name.instance@realm* (see Figure 2).

bcn
treese.root
jis@LCS.MIT.EDU
rlogin.priam@ATHENA.MIT.EDU

Figure 2. *Kerberos* Names.

The *primary name* is the name of the user or the service. The *instance* is used to distinguish among variations on the primary name. For users, an instance may entail special privileges, such as the ‘‘root’’ or ‘‘admin’’ instances. For services in the Athena environment, the instance is usually the name of the machine on the which the server runs. For example, the *rlogin* service has different instances on different hosts:

rlogin.priam is the *rlogin* server on the host named priam. A *Kerberos* ticket is only good for a single named server. As such, a separate ticket is required to gain access to different instances of the same service. The *realm* is the name of an administrative entity that maintains authentication data. For example, different institutions may each have their own *Kerberos* machine, housing a different database. They have different *Kerberos* realms. (Realms are discussed further in section 8.2.)

4. How It Works

This section describes the *Kerberos* authentication protocols. The following abbreviations are used in the figures.

c	->	client
s	->	server
addr	->	client's network address
life	->	lifetime of ticket
tgs, TGS	->	ticket-granting server
Kerberos	->	authentication server
KDBM	->	administration server
K_x	->	x's private key
$K_{x,y}$	->	session key for x and y
$\{abc\}K_x$	->	abc encrypted in x's key
$T_{x,y}$	->	x's ticket to use y
A_x	->	authenticator for x
WS	->	workstation

As mentioned above, the *Kerberos* authentication model is based on the Needham and Schroeder key distribution protocol. When a user requests a service, her/his identity must be established. To do this, a ticket is presented to the server, along with proof that the ticket was originally issued to the user, not stolen. There are three phases to authentication through *Kerberos*. In the first phase, the user obtains credentials to be used to request access to other services. In the second phase, the user requests authentication for a specific service. In the final phase, the user presents those credentials to the end server.

4.1. Credentials

There are two types of credentials used in the *Kerberos* authentication model: *tickets* and *authenticators*. Both are based on private key encryption, but they are encrypted using different keys. A ticket is used to securely pass the identity of the person to whom the ticket was issued between the authentication server and the end server. A ticket also passes information that can be used to make sure that the person using the ticket is the same person to which it was issued. The authenticator contains the additional information which, when compared against that in the ticket proves that the client presenting the ticket is the same one to which the ticket was issued.

A ticket is good for a single server and a single client. It contains the name of the server, the name of the client, the Internet address of the client, a timestamp, a lifetime, and a random session key. This information is encrypted using the key of the server for which the ticket will be used. Once the ticket has been issued, it may be used multiple times by the named client to gain access to the named server, until the ticket expires. Note that because the ticket is encrypted in the key of the server, it is safe to allow the user to pass the ticket on to the server without having to worry about the user modifying the ticket (see Figure 3).

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

Figure 3. A *Kerberos* Ticket.

Unlike the ticket, the authenticator can only be used once. A new one must be generated each time a client wants to use a service. This does not present a problem because the client is able to build the authenticator itself. An authenticator contains the name of the client, the workstation's IP address, and the current workstation time. The authenticator is encrypted in the session key that is part of the ticket (see Figure 4).

$$\{c, addr, timestamp\}K_{s,c}$$

Figure 4. A *Kerberos* Authenticator.

4.2. Getting the Initial Ticket

When the user walks up to a workstation, only one piece of information can prove her/his identity: the user's password. The initial exchange with the authentication server is designed to minimize the chance that the password will be compromised, while at the same time not allowing a user to properly authenticate her/himself without knowledge of that password. The process of logging in appears to the user to be the same as logging in to a timesharing system. Behind the scenes, though, it is quite different (see Figure 5).

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.