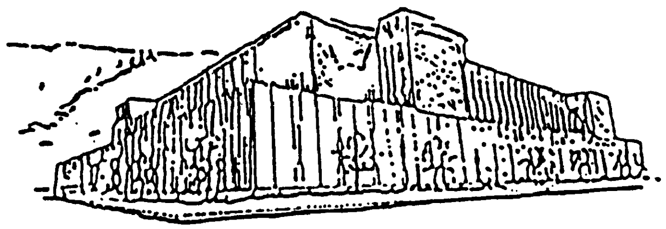
1998

Unifying heterogeneous networks with Kerberos Authentication Server and multithread implementation of Kerberized FTP for Windows 95/NT

Chien Lin
The University of Montana



Unifying Heterogeneous Networks with Kerberos Authentication Server and Multithread Implementation of Kerberized FTP for Windows 95/NT

by

Jian Lin

B.S. Nankai University, 1995

presented in partial fulfillment of the requirements

for the degree of

Master of Science

The University of Montana

May 1998

Approved by:

[illegible]

Chairman

[illegible]

Dean, Graduate School

6-10-98

Date

Lin, Jian, M.S., May 1998 Computer Science

Unifying Heterogeneous Networks with Kerberos Authentication Server and Multithread Implementation of Kerberized FTP for Windows 95/NT (53 pp.)

Director: Alden H. Wright [illegible]

Commercialization of the Internet has triggered an explosion of development aimed at maximizing on-line communication. Multi-platform client-server installations are becoming the enterprise network norm, and market forces are causing new resources to be driven into networks on an almost daily basis. Competitive pressures and a growing acceptance of remote management and telecommuting are fueling the development explosion, and the end is not yet in sight. This has put network administrators in a challenging position: They are being asked to provide wide-scale, open access to heterogeneous networks, but to do so without compromising overall security. The Computer Science Department at the University of Montana may upgrade some older RS6000 workstations (AIX4.1) with Pentium PC workstations (Redhat LINUX). So we need a mechanism to unify the heterogeneous networks (including AIX 4.1, Redhat LINUX, and NT4.0). At the same time, we want to secure our heterogeneous networks as well as possible. The Kerberos Authentication Server is a desirable solution to this situation. Kerberos Authentication Server provides a common security implementation at a very basic level that prevents the weaknesses of one system from compromising the strengths of the other.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Description

In recent years, multi-platform distributed computing installations are becoming more popular. With the use of the Internet becoming more and more extensive, the networks that were previously private are now connected to the Internet. That exposes a lot of security problems on an open network.

This has put network administrators in a challenging position: They are being asked to provide wide-scale, open access to heterogeneous networks and to achieve higher security. Perhaps the most difficult problem is to unify the UNIX/NT/MAC networks. UNIX/NT/MAC administrators have the formidable task of implementing disparate security solutions on fundamentally different operating systems, then combining those solutions to create uniform access and management across the entire network. The risk is that the combined solution can yield a result that is weaker than the weakest link of any single operating system's security alone.

The CS department of the University of Montana may replace some old AIX workstations with LINUX workstations. So the CS department needs a mechanism to maintain a global user account database for the heterogeneous networks (including

AIX 4.1, LINUX, NT 4.0) At the same time, the CS department wants to secure the heterogeneous networks as well as possible. After the replacement, the CS department will have AIX workstations, LINUX workstations, and NT workstations. The goals are:

- A Global user account database, which means that any user must be able to login to the same account with the same password on any workstation
- Users should be able to change their passwords from any workstation
- Any system must work with a shadow password system on both AIX and LINUX
- Users should be able to access any workstation over the Internet (for example, a student or faculty should be able to login to a workstation using a PPP connection to an ISP from home)
- Unencrypted passwords should not be transferred over the network
- The solution should be relatively easy for the system administrator to maintain
- All of the system administrator's tasks and as many as possible of other tasks should be able to be done through an encrypted session
- The solution should be easy for users to use
- The client programs like FTP and TELNET should be easy for CS students to install on personally-owned PCs

1.2 Proposed Solution

The fundamental issue in implementing security for a UNIX/NT environment is to provide a common security implementation at a very basic level that prevents the weaknesses of one system from compromising the strengths of the other. The richer this common implementation is, the more security can be enhanced for both systems.

The Kerberos Authentication Server, which was developed by MIT, IBM and DEC implements a third-party authentication layer that unifies UNIX and NT user administration, authentication, and authorization, and provides functional uniformity for administering.

Cygnus Inc. developed a commercial version of Kerberos, **KerbNet**, which contains Kerberos server and client applications for both UNIX and NT. Using a set of technologies that create a security layer which is implemented and managed identically on both UNIX and NT systems, KerbNet software was announced to conform the following UNIX/NT distinctions:

Since a user-ID in NT is completely alien to UNIX and vice-versa, the most difficult components to unify are the different user authentication models. KerbNet software surmounts this problem by utilizing a single trusted server which other systems access over the network. This trusted server provides a central repository for usernames and passwords, as well as some access control audit information.

KerbNet software authenticates users and application servers and provides administrators with a unified method of managing passwords and access on all platforms in the system. With enterprise-

wide authentication, administrators are assured of the identities of users and servers. With this assurance, single sign-on can be used system-wide at both network and application levels.

From a user standpoint, the KerbNet interface provides single-login convenience and seamless access to network resources. From an application-development standpoint, KerbNet software relieves developers of the burden of having to provide functionality to accommodate new users or groups.

And for administrators, KerbNet software provides a uniform interface from which they can see exactly who users are at any moment and add, delete, or manage identifications and authorizations system-wide, regardless of the operating system platform. [14]

Actually from my point of view, KerbNet doesn't achieve all of the above. If Kerbnet is really used to unify Unix/NT/Mac environment, there is a password synchronization problem. Due to the conflict of the NT native authentication protocol and the Kerberos authentication protocol, in order to achieve single sign-on, a user's NT domain password must be the same as the user's Kerberos password. I will talk about this in detail in chapter 3.

At a recent developer's conference, Microsoft announced that NT version 5.0 will include Kerberos authentication. This endorsement of Kerberos legitimizes the technology as a foundation for network security. With this initiative, Microsoft will also be making available Kerberized file and print services for current Windows 95 installations. Microsoft's direction to provide NT Kerberos services and Kerberized applications ensures that future Windows NT and Windows 95 users will have full access to Internet standard security services.

Chapter 2

Kerberos Authentication Server

2.1 History of Kerberos

In 1983, MIT, in cooperation with IBM and DEC, started an eight-year project designed to integrate computers into the university's undergraduate curriculum. This project was called Project Athena.

At the beginning, nearly 50 traditional time-sharing minicomputers--Digital Equipment Corporation's VAX 11/750 systems running Berkeley 4.2 UNIX, were used by project Athena. Each VAX had a few terminals; when a student or faculty member wanted to use a computer, he/she sat down at one of its terminals.

Within a few years, the project received a lot of funds to update equipment. Hundreds of networked workstations replaced the VAX 11/750s. *The project's goal was to allow any user to sit down at any computer and enjoy full access to his files and to the network* [12].

After the workstations were deployed, the network was accessible from all over the MIT campus. It was impossible to prevent students (or outside intruders) from running network spy programs. It was also nearly impossible to prevent the students from intercepting the root user passwords of the workstations. Even worse, many of the computers on the network were IBM PC/ATs that didn't have even the basic operating system security. *Something had to be done to protect student files in the networked environment to the same degree that they were protected in the time-sharing environment* [12].

Athena's ultimate solution to this security problem was Kerberos. *Kerberos is an authentication system that uses DES cryptography to protect sensitive information such as passwords on an open network. When a user logs in to a workstation running Kerberos, that user is issued a ticket from the Kerberos Authentication Server. The user's ticket can only be decrypted with the user's password; it contains information necessary to obtain additional tickets, From that point on, whenever the user wishes to access a network service, an appropriate ticket for that service must be presented. As all of the information in the Kerberos tickets is encrypted before it is sent over the network, the information is not susceptible to eavesdropping or misappropriation* [12].

2.2 Authentication, Integrity, Confidentiality, and Authorization

Neuman and Ts'o [1] give some general definitions:

- ***A principal*** *is the party whose identity is verified.*
- ***The verifier*** *is the party who demands assurance of the principal's identity.*
- ***Data integrity*** *is the assurance that the data received is the same as generated.*
- ***Authentication*** *is the verification of the identity of a party who generated some data, and of the integrity of the data.*

- ***Confidentiality** is the protection of information from disclosure to those not intended to receive it.*
- ***Authorization** is the process by which one determines whether a principal is allowed to perform an operation* [1].

For example, when a user telnets to remote machine, the remote machine's telnetd daemon is the verifier, the user is the principal, and the user's identity need to be verified by the verifier – the remote telnetd daemon. Authorization is usually performed after the principal has been authenticated. Normally, authentication and authorization occur at the same time.

The section concentrates on authentication for real-time, interactive services that are offered on computer networks. The term real-time is used to mean that a client process is waiting for a response to a query or command so that it can display the results to the user. This class of services includes remote login, file system reads and writes, and information retrieval for applications like web browser.

2.3 How Kerberos works

A series of encrypted messages is used by the Kerberos Authentication System to prove to a verifier (server) that a client is running on behalf of a real claimed user.

The remainder of this section describes the Kerberos protocol. The description is simplified for clarity; additional fields are present in the actual protocol. Readers should consult RFC 1510 [4] for a more thorough description of the Kerberos protocol.

2.3.1 Kerberos Encryption

Kerberos is an authentication system based on the secret key cryptography. In the Kerberos Version 4, the only encryption method supported is the data encryption standard (DES). A property of DES is that if the encrypted data is decrypted with the same key used to encrypt it, the original data appears. If different encryption keys are used for encryption and decryption, or if the encrypted data is modified, the result will be unreadable, and the checksum in the Kerberos message will not match. Due to the U.S. regulation limiting the export of DES, in Kerberos Version 5, an encryption algorithm identifier tag is appended on each Kerberos message, which allows the user to configure Kerberos to use alternative encryption algorithm other than DES. But so far, all implementations of Kerberos Version 5 use DES. A standard for a public key version of Kerberos is under consideration.

2.3.2 What are the Kerberos Tickets

- Each service registered in Kerberos Authentication Server (KAS) needs to share a secret key with the KAS, this secret key is called the **server key.** For example, FTP, TELNET and other services need to share a server key with the KAS respectively.

- **A Service Granting Ticket (SGT)** consists of a set of information that can be used by the client to apply for service provided by a specific application server. Normally, **the Ticket Granting Service (TGS)** issues a SGT. A separate SGT is needed for different service. For example, a FTP ticket is needed for the FTP service and a HOST ticket is needed for the TELNET service.
- **Ticket Granting Ticket (TGT)** is issued by the KAS and used to obtain multiple SGTs from the TGS later within a limited period (normally 8 hours). The TGT is kind of like a general ticket for Disney Land. Once a tourist obtains this general ticket, the tourist can get the access to any service supplied by the Disney Land within a day.

2.3.3 How Kerberos Authentication Works

Under Kerberos Version 5, the user first types in his/her user ID and password. Then the client sends a message to the KAS that includes the user ID, the name of the TGS service, the requested TGT expiration time, and a random number, all encrypted with the user's password (message 1 in figures 2.1 and 2.2). The client uses this message to request a TGT for the user represented by itself. When the KAS receives the message, it finds the claimed user's password from its password database, and tries to decrypt the message using the user's password. If the decryption succeeds, the KAS assumes that the running client really represents the claimed user. Then the KAS generates a one-time encryption key known as the **session key** and a

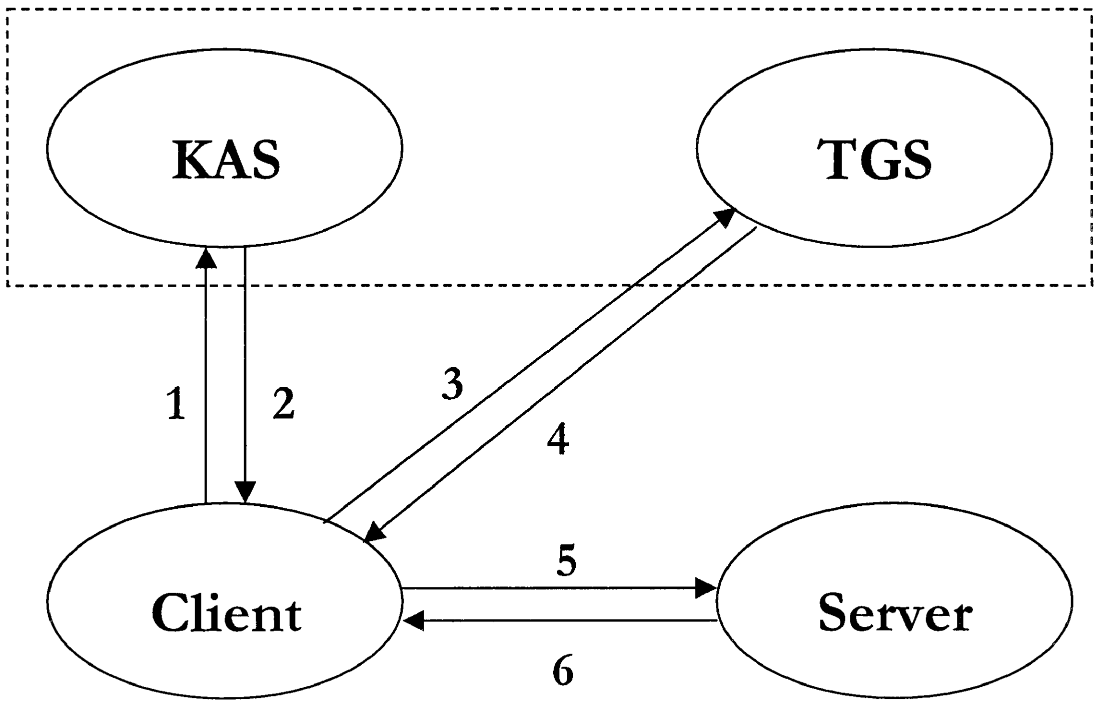
TGT that contains the session key; the TGT is encrypted with the TGS server key and the session key with other information are encrypted with the user's password. This prevents the TGT from being modified and guarantees that only the real user can extract the session key. The KAS sends the encrypted TGT and session key back to the client (message 2 in figures 2.1 and 2.2). The client decrypts the encrypted session key using the user's password, stores the encrypted TGT and the decrypted session key into a temporary file, and forgets the user's password. Next, the client will use the TGT and session key to obtain the SGTs as needed. Through messages 1 and 2 in figures 2.1 and 2.2, the client obtains the TGT and session key without transferring the user's password over the network at all.

When the user wants to use any service, the user needs to obtain a SGT for that requested service. The client sends the encrypted TGT and a request for that service which is encrypted with the TGS session key to the TGS (message 3 in figures 2.1 and 2.2). Upon the receipt of the message 3, the TGS decrypts the encrypted TGT with its server key, extracts the session key and tries to decrypt that request with the session key. So the TGS knows that the client has requested a SGT for that requested service. Then the TGS generates another session key which is shared between the client and the requested service and encrypted with the original TGS session key, and an SGT for that requested service which contains the session key and is encrypted with that requested service's server key. The TGS sends them back to the client (message 4 in figures 2.1 and 2.2).

Upon the receipt of the message 4, the client decrypts the session key with the original TGS session key, and stores the encrypted SGT and the decrypted SGT session key into the same temporary file. Then the client sends the message 5 in figures 2.1 and 2.2 to the requested service. Message 5 consists of two parts: authenticator and encrypted SGT. The authenticator consists of the current time, a checksum, and an optional encryption key, all encrypted with the SGT session key. Upon the receipt of the message 5, the server decrypts the encrypted SGT with its server key, extracts the SGT session key, and tries to decrypt the authenticator. If the same key is used to encrypt and to decrypt on the same authenticator, the checksum in the authenticator will match. So the server can be sure that the client is running to represent the claimed user, since the client knows the session key. In some cases, the client wants to verify the identity of the server. If mutual-authentication is required, the message 6 in figures 2.1 and 2.2 is used.

At this point, the user has access to the service. Later, if the user wants to use other services, the client will go through from messages 3 and 5 (message 6, if needed) to obtain the corresponding SGT for the requested service and to get the access. The user doesn't need to type his/her password anymore. The single sign-on is achieved by obtaining the TGT through messages 1 and 2 in figures 2.1 and 2.2.

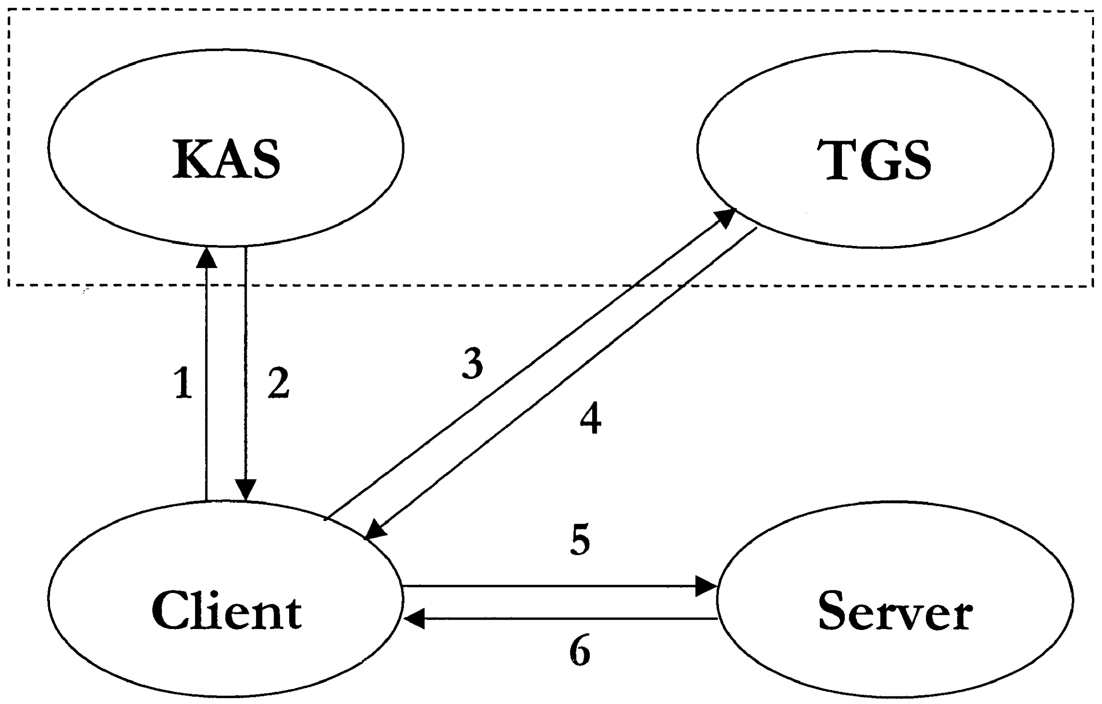
For further details of the Kerberos Authentication Server, please see [1, 2, 3, 4, 15].



1. Client → KAS: C, Tgs, $Time_{exp}$, N
2. KAS → Client: $\{K_{c,Tgs}, Tgs, Time_{exp}, N, \{T_{c,Tgs}\}K_{Tgs}\}K_c$
3. Client → TGS: $\{CT, \ldots\}K_{c,Tgs}$, $\{T_{c,Tgs}\}K_{Tgs}$, $Time_{exp}$, N
4. TGS → Client: $\{K_{c,v}, V, Time_{exp}, N, \ldots\}K_{c,Tgs}$, $\{T_{c,v}\}K_v$
5. Client → Server $\{CT, CK, K_{opt}, \ldots\}K_{c,v}$, $\{T_{c,v}\}K_v$
6. Server → Client $\{CT,\ldots\}K_{c,v}$ (optional)

KAS: Kerberos Authentication Server
TGS: Ticket Granting Service (usually exists in the same machine with KAS)
C: Client's name
V: Verifier's name (Server's name)
Tgs: TGS's name
N: A random number(used to match the authentication response
with the request)
$Time_{exp}$: Requested expiration time for the ticket
CT: Current time
CK: Checksum
K_c: User's password
K_v: Server key (shared between server and KAS)
$K_{c,v}$: Session key (Client with Server)
$K_{c,Tgs}$: Session key (Client with TGS)
$T_{c,Tgs}$: Ticket to TGS $\{K_{c,Tgs}, C, Time_{exp}, \ldots\}$
$T_{c,v}$: Ticket to the Server$\{K_{c,v}$, C, $Time_{exp},\ldots\}$

Figure 2.1 Complete Kerberos V4 Authenticaion Protocol (simplified)



1. Client → KAS: C , {C, Tgs, $Time_{exp}$, N}K_c
2. KAS → Client: {$K_{c,Tgs}$,Tgs, $Time_{exp}$,N}K_c , {$T_{c,Tgs}$}K_{Tgs}
3. Client → TGS: {CT, …}$K_{c,Tgs}$, {$T_{c,Tgs}$}K_{Tgs} , $Time_{exp}$, N
4. TGS → Client: {$K_{c,v}$, V, $Time_{exp}$, N, …}$K_{c,Tgs}$, {$T_{c,v}$}K_v
5. Client → Server {CT, CK, K_{opt} , …}$K_{c,v}$, {$T_{c,v}$}K_v
6. Server → Client {CT,…}$K_{c,v}$ (optional)

KAS: Keberos Authentication Server
TGS: Ticket Granting Service (usually exists in the same machine with KAS)
C: Client's name
V: Verifier's name (Server's name)
Tgs: TGS's name
N: A random number(used to match the authentication response
with the request)
$Time_{exp}$: Requested expiration time for the ticket
CT: Current time
CK: Checksum
K_c: User's password
K_v: Server key (shared between server and KAS)
$K_{c,v}$: Session key (Client with Server)
$K_{c,Tgs}$: Session key (Client with TGS)
$T_{c,Tgs}$: Ticket to TGS {$K_{c,Tgs}$, C, $Time_{exp}$, …}
$T_{c,v}$: Ticket to the Verifier(Server){$K_{c,v}$, C, $Time_{exp}$,…}

Figure 2.2 Complete Kerberos Version 5 Authenticaion Protocol (simplified)

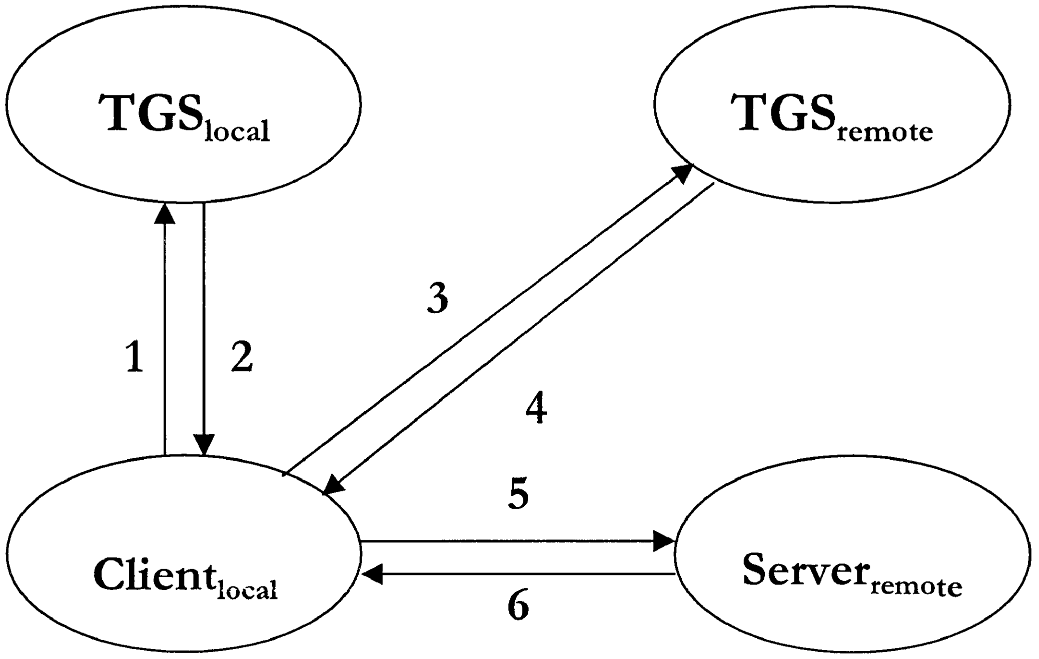
2.3.4 Kerberos Cross-Realm Authentication

A full-service Kerberos environment consisting of a Kerberos server, a number of clients and a number of application servers, requires the following:

- *The Kerberos server must have the User ID and password of all participating users in its database. All users are registered with Kerberos server.*
- *The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.*

Such an environment is referred to as a **realm** [15].

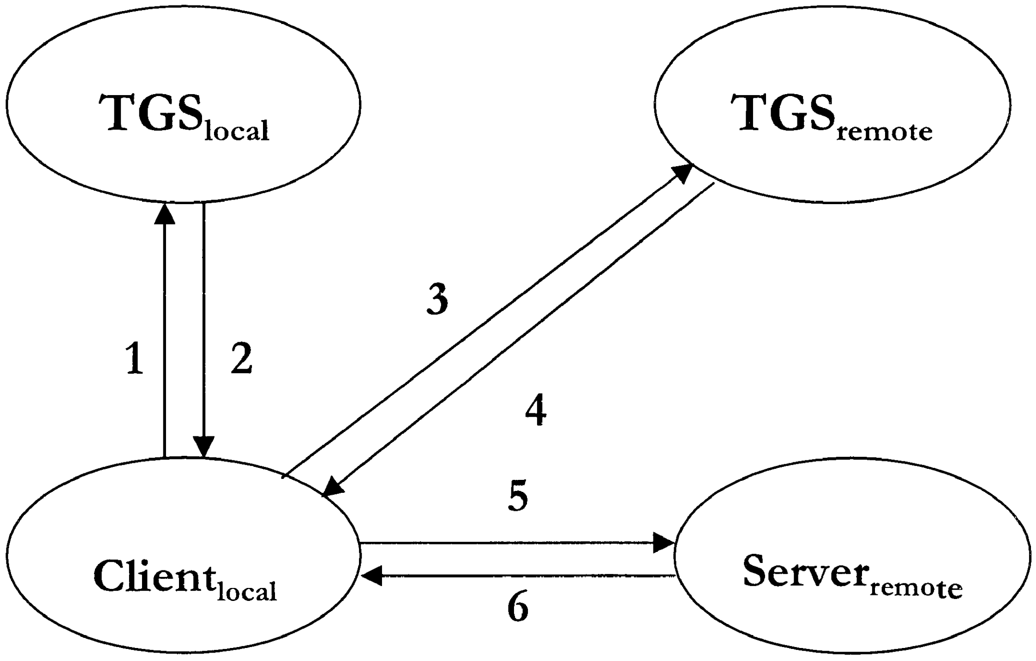
It's impractical for networks in different organizations to register in the same realm. Instead, the networks in different organizations are in different realms. Sometimes, a user registered in one realm needs to use the server registered in another realm, which leads to a cross-realm authentication problem. To enable the cross-realm authentication between two realms, a secret key needs to be generated and shared between the two realms. In order for a user registered in the local realm to use a server registered in a remote realm, the user must obtain a SGT for the remote server. The client uses messages 1 and 2 in figures 2.3 and 2.4 to obtain the TGT for the remote realm. Then the client uses the TGT for the remote realm to request a SGT for the remote server from the remote TGS through the messages 3 and 4 from figures 2.3 and 2.4. As soon as the remote KAS detects that the TGT was issued in a different realm, it finds the cross-realm key, verifies the validity of the TGT, and then generates and issues a SGT and session key to the remote client. The authentication of the local client to the remote server is the same as that in the same realm. For further details of Kerberos cross-realm authentication, please see [1, 3].



1. Client → TGS_{local}: $\{A_c\}K_{c,tgs}$, tgs_{rem} , $\{T_{c,tgs}\}K_{tgs}$
2. TGS_{local} → Client: $\{Kc,tgs_{rem}, \{Tc,tgs_{rem}\}Ktgs_{rem}\}K_{c,tgs}$
3. Client → TGS_{remote}: $\{A_c\}Kc,tgs_{rem}$, $\{Tc,tgs_{rem}\}Ktgs_{rem}$, S_{rem}
4. TGS_{remote} → Client: $\{ Kc,s_{rem}, \{Tc,s_{rem}\}Ks_{rem}\}Kc,tgs_{rem}$
5. Client → $Server_{remote}$: $\{A_c\}$ Kc,s_{rem}, $\{Tc,s_{rem}\}Ks_{rem}$
6. $Server_{remote}$ → Client: $\{A_c\}$ Kc,s_{rem} (optional)

A_c: Authenticator (different for different message)
TGS_{local}: Local Ticket Granting Service
$TGS_{remote:}$ Remote Ticket Granting Service
tgs: TGS_{local}'s name
tgs_{rem}: TGS_{remote}'s name
$K_{c,tgs}$: Session key (Client with TGS_{local})
Kc,tgs_{rem} : Session key(Client with TGS_{remote})
Kc,s_{rem} : Session key (Client with Remote Server)
$T_{c,tgs}$: Ticket to TGS_{local}
Tc,tgs_{rem} : Ticket to TGS_{remote}
K_{tgs}: TGS_{local} key known only by TGS_{local}
$Ktgs_{rem}$: Inter-realm key shared with TGS_{local} and TGS_{remote}
Ks_{rem} : Remote Server key shared with TGS_{remote}

Figure 2.3 Kerberos V4 Inter-realm Authenticaion Protocol (simplified)



1. Client → TGS$_{local}$: $\{A_c\}K_{c,tgs}$, tgs_{rem} , $\{T_{c,tgs}\}K_{tgs}$
2. TGS$_{local}$ → Client: $\{K_{c,tgs_{rem}}\}K_{c,tgs}$, $\{T_{c,tgs_{rem}}\}K_{tgs_{rem}}$
3. Client → TGS$_{remote}$: $\{A_c\}K_{c,tgs_{rem}}$, $\{T_{c,tgs_{rem}}\}K_{tgs_{rem}}$, S_{rem}
4. TGS$_{remote}$ → Client: $\{K_{c,s_{rem}}\}K_{c,tgs_{rem}}$, $\{T_{c,s_{rem}}\}K_{s_{rem}}$
5. Client → Server$_{remote}$: $\{A_c\}$ $K_{c,s_{rem}}$, $\{T_{c,s_{rem}}\}K_{s_{rem}}$
6. Server$_{remote}$ → Client: $\{A_c\}$ $K_{c,s_{rem}}$ (optional)

A_c: Authenticator (different for different message)
TGS$_{local}$: Local Ticket Granting Service
TGS$_{remote:}$ Remote Ticket Granting Service
tgs: TGS$_{local}$'s name
tgs_{rem}: TGS$_{remote}$'s name
$K_{c,tgs}$: Session key (Client with TGS$_{local}$)
$K_{c,tgs_{rem}}$: Session key(Client with TGS$_{remote}$)
$K_{c,s_{rem}}$: Session key (Client with Remote Server)
$T_{c,tgs}$: Ticket to TGS$_{local}$
$T_{c,tgs_{rem}}$: Ticket to TGS$_{remote}$
K_{tgs}: TGS$_{local}$ key known only by TGS$_{local}$
$K_{tgs_{rem}}$: Inter-realm key shared with TGS$_{local}$ and TGS$_{remote}$
$K_{s_{rem}}$: Remote Server key shared with TGS$_{remote}$

Figure 2.4 Kerberos V5 Inter-realm Authenticaion Protocol (simplified)

2.4 Kerberos Version 4 vs. Kerberos Version 5

Initial login:

With Kerberos Version 4, after a user types his/her ID, the client sends a message to the *Kerberos Authentication Server* (KAS) that includes the user's ID, a request for a TGT, and other information. The KAS checks its database, finds the user's password, sends back a credential that is encrypted with the user's password. The client then prompts the user to type in his/her password and finally tries to decrypt the encrypted credential with the password that the user has typed. If the decryption is successful, the client then forgets the user's password, and stores the credential into a temporary file. If the decryption fails, the client knows that the user typed the wrong password and gives the user a chance to try again. This scenario makes Kerberos Version 4 susceptible to offline password-guessing attacks. An attacker could simply modify the login program, ask a KAS for a TGT for a particular user, then copy it into a file, then try to decrypt that ticket with every word in the dictionary. For example, an attacker might write a kerberized "kinit" for Win95/NT that obtains the TGT, but this "kinit" simply stores the encrypted TGT into a file. Since the PC on which the attacker is running the "kinit" is entirely under the control of the attacker, the attacker can apply a password-guessing process at his/her leisure.

With Kerberos Version 5, the client doesn't contact the KAS until the user has typed his/her password. Then the client sends a message to the KAS that consists of the user's ID, a request for a TGT, a random number and etc., all encrypted with the user's password. The KAS looks up the user's ID, finds his/her password, and tries to decrypt the message. If the decryption succeeds, the KAS then creates a credential that is encrypted with the user's password, and sends it back to the user. The KAS can be set up to allow a fixed number of login attempts. This will effectively prevent a password-guessing process.

Inter-realm Authentication:

With Kereros Version 4, in order to connect n realms completely, $\frac{n(n-1)}{2}$ key exchanges are required (see figure 2.5). Even with only a few cooperating realms, the assignment and management of the inter-realm keys is an expensive task.

With Version 5, a hierarchy based on the name of the realm is supported (see figure 2.6). A source realm can communicate directly with a destination realm if it shares an inter-realm key directly with the destination realm. Or if it shares a key with an intermediate realm that shares an inter-realm key with the destination realm, the source realm can communicate with the destination realm through the intermediate realm too. Each realm needs to share a unique inter-realm key with its parent node and each of its child nodes. This arrangement reduces the number of key exchanges to O(log(n)).

For further details, please see [1, 3].

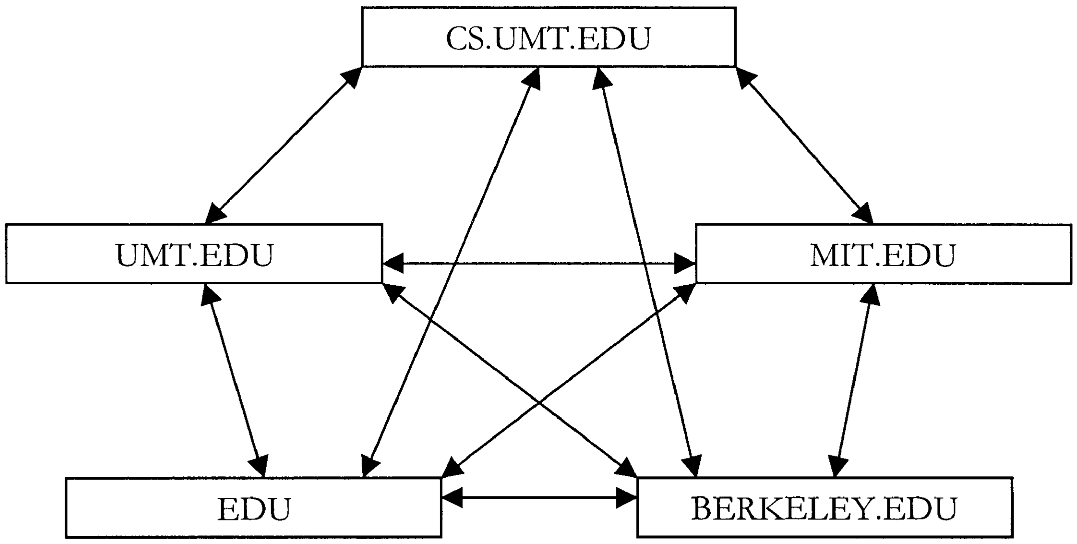


Figure 2.5: Kerberos V4 realm interconnections

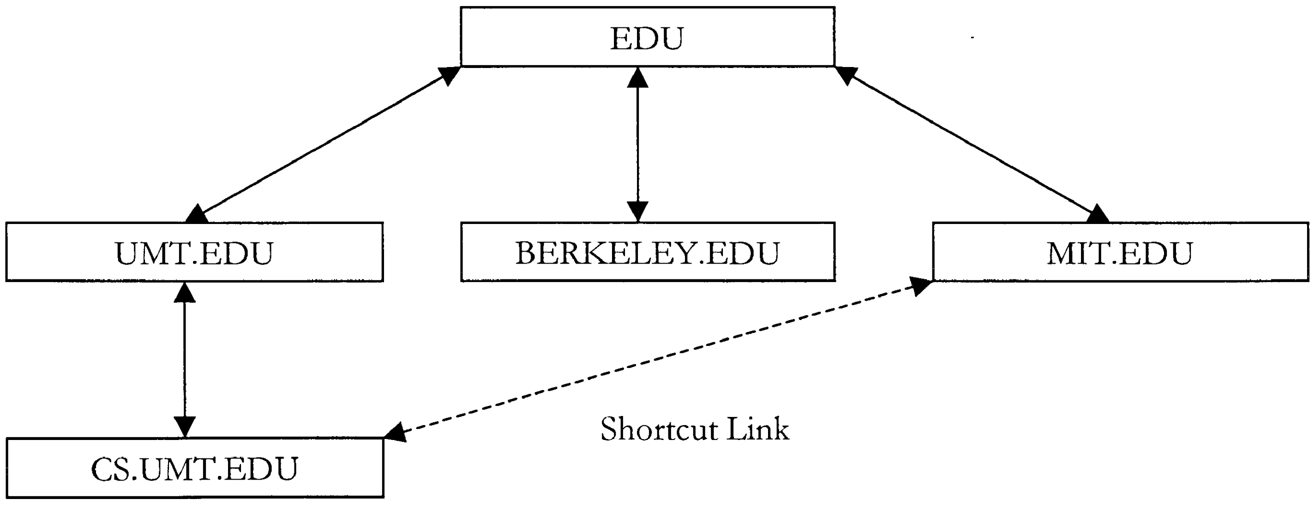


Figure 2.6: Kerberos V5 hierarchy of realms

2.5 Kerberos Limitations

Kerberos is a good solution to solve the security problems on an open network, but it still has several shortcomings:

- ***Every network service must be individually modified for use with Kerberos*** [12]. Due to the design of the Kerberos, each application must be modified to use Kerberos for authentication. The process of modifying an application is called *Kerberizing* the application. To Kerberize an application, the application's source code is required.
- ***Kerberos doesn't work well in a time-sharing environment.*** *Kerberos is designed for an environment in which there is one user per workstation. Because of the difficulty of sharing data between different processes running on the same UNIX computer, Kerberos keeps tickets in the "/tmp" directory. If a user is sharing the computer with several other people, it is possible that the user's tickets can be stolen, that is, copied by an attacker, stolen tickets can then be used to obtain fraudulent service* [12]. This problem is much worse for a shared PC or Mac running an insecure operating system like Windows 95/98 or MacOS. If an attacker has stolen other user's tickets and the attacker is familiar with Kerberos API or GSSAPI and the Kerberizing process, the attacker can write a program to use that user's stolen ticket to impersonate that user.
- ***Kerberos requires a secure Kerberos server.*** *By design, Kerberos requires that there be a secure central server that maintains the master password database. To ensure security, the Kerberos server should be run on a secure dedicated machine* [12]. Any unnecessary services

(such as POP, FTP and R-commands daemons) should be shut down. The Kerberos server must be kept in a physically secure place.

- ***Kerberos requires a continuously available Kerberos server*** [12]. If the Kerberos Authentication Server goes down, the whole Kerberos network can not be used. So normally, a backup Kerberos Authentication Server called slave KDC is needed.
- ***Kerberos stores all passwords encrypted with a single key*** [12]. All passwords stored in the Kerberos server are encrypted with the server's master key, which resides on the same hard disk as the encrypted passwords. This means that, in case the Kerberos server is compromised, all user passwords are compromised.
- ***Kerberos does not protect against modifications to system software (Trojan horses).*** *Kerberos does not have the computer authenticate itself to the user, so there is no way for a user sitting at a computer to determine whether the computer has been compromised* [12]. A knowledgeable attacker can easily exploit this weakness. For example, an attacker could simply modify a workstation's system login program which copies every username/password combination typed into a file. Later, the attacker can impersonate these users using their passwords.
- ***Kerberos may result in a cascading loss of trust.*** *Another problem with Kerberos is that if a server password or a user password is broken or otherwise disclosed, it is possible for an eavesdropper to use that password to decrypt other tickets and use this information to spoof servers and users* [12].

Kerberos is a workable system based on the secret key cryptography for network security, and it is widely used. A system based on public key cryptography would probably be simpler and perhaps more secure. A more secure authentication system would combine:

1. Something that a user knows (e.g. password).
2. Something that a user has (e.g. a smart card or a token).
3. Something that a user is (e.g. fingerprint, face recognition, etc.).

Chapter 3

Implementation of Unification

3.1 Possible Implementations

3.1.1 The KDC servers

To unify the heterogeneous networks (AIX4.1, Redhat LINUX, and NT 4.0) using a Kerberos Authentication server, a master KDC and a slave KDC are needed. Because the more other services that are run on the KDCs, the greater the possibility is that the KDCs could be compromised, the KDCs should be dedicated servers which are only used for authentication. The CS department at the University of Montana doesn't want to spend a lot of money on these servers, since the CS department is not very big and the authentication workload is not very heavy. So a solution might be to use two Intel486 LINUX machines as KDCs. That should be enough for the CS department. The KDCs must be located in a physically secure place. If the KDCs are compromised, the whole network will be compromised. When a user changes his/her Kerberos password, only the master KDC's password database has been updated. Any changes of password only occur on master KDC. In order to let the slave KDC know about password changes, a cron job needs to be set up to make the master KDC to periodically propagate the password database to the slave KDC. Another cron job also needs to be set up, which periodically propagates the current time from the KDC to other machines. If the time difference (clockskew) between clients and KDC is larger than 5 minutes, authentication will fail, even with the correct password.

3.1.2 The Application Servers and Clients

All AIX/LINUX machines except for the KDCs can be both application servers and clients. To use Kerberos authentication server, the Kerberized daemons and clients must replace the existing daemons and clients. The Kerberized daemons include telnetd, ftpd, klogind, kshd…, the Kerberized clients include telnet, ftp, klogin, ksh, kinit, kdestroy, kpasswd… A configuration file "krb5.conf" also needs to be copied into "/etc" subdirectory. Actually, the same copy of "krb5.conf" is shared among all the machines. To obtain a *Ticket Granting Ticket* (TGT) implicitly when a user logs in, the regular login must be replaced with the Kerberized login – "klogin". So when a user logs in, the user has obtained a TGT implicitly, the user doesn't need to use "kinit" to get it explicitly. When the user logs out, all his/her tickets need to be deleted using "kdestroy". In case that the user forgets to do this, "kdestroy" should be added to his/her ".logout" file. The user should lock the screen, if he/she leaves his/her workstation without logging out.

3.1.3 Some Issues about the login process

3.1.3.1 Local Login to Unix workstation(AIX, Redhat LINUX)

After the Kerberos has been deployed on the CS department's LAN, the regular login has been replaced with the Kerberized login—"klogin". When a user sits in front of any UNIX (AIX, Redhat LINUX) workstation and wants to login, the user types in his/her login ID and Kerberos password. The Kerberized login – "klogin" – then encrypts an authenticator using his/her Kerberos password and transfers the authenticator (including the service name TGS –Ticket Granting Service, and a time stamp) and his/her login ID to the KDC. The KDC knows the user's Kerberos password, so the KDC tries to decrypt the authenticator. If the KDC succeeds, the KDC sends back a TGT and "klogin" lets the user login. (A more detailed description of the process of obtaining TGT is in chapter 2) If the login does not succeed, "klogin" will do the same thing as the regular login program namely compare the password the user typed with his/her local password stored in "/etc/passwd" (or "/etc/shadow", if a shadow-password mechanism is used). If the password matches, "klogin" lets the user login without obtaining the TGT. If the user still wants to obtain the TGT, the user has to use "kinit" and types in the correct

Kerberos password to get it. After the user has the TGT, the user can use any Kerberized service of the heterogeneous network without typing a password. It's very convenient and secure.

3.1.3.2 Local Logon to NT Workstation/Server

Cygnus Solutions Inc. provides a freeware Kerberos package for NT, which includes the KDC service, the Kadmin service, and the Kerberized client softwares such as kinit, kpasswd, kerbnet, ktelnet and gina. The KDC service and Kadmin service can only run on NT server/workstation, the Kerberized client software can run on Windows 95 or Windows NT server/workstation. So an NT machine can be configured to be a KDC.

The "gina" package is login software that can replace the regular NT login software. There are two installation options for "gina": Kerberos authentication required, and Kerberos authentication not required.

- **Kerberos authentication not required**

If this option is chosen, When a user logs on, first, "gina" will check the password the user typed in with the Windows NT PDC (Primary Domain Controller) to see if it matches the password stored in the PDC's password database. If it does not match, "gina" gives the user a message box that says something like "password wrong, retry" and won't let the user logon. If the password matches, "gina" will try to obtain a TGT from the KDC using the password the user typed in. If the password the user typed in is the same as his/her Kerberos password, "gina" will obtain a TGT implicitly for the user and let the user logon. If the password the user typed is not the same as his/her Kerberos password, but is same as his/her NT domain password, "gina" will give the user a message box that says something like "Kerberos password wrong and can not obtain TGT" and let the user logon without obtaining the TGT.

- **Kerberos authentication required**

If this option is chosen, the process is almost the same as the above scenario, except that when the user's password matches the PDC's stored password but doesn't match his/her Kerberos password. "gina" still gives the user a message box that says something like "Kerberos password wrong, retry" and won't let the user logon. Thus, in this case, the user's password in the Windows NT PDC must match the Kerberos password to be able to login NT workstation or server.

3.1.3.3 Over-the-network telnet from Windows 95/NT to UNIX (AIX, Redhat LINUX)

- **Using non-Kerberized telnet**

If the application servers have been configured so that they can accept non-Kerberized clients, then a user can use non-Kerberized client software such as regular TELNET, FTP, and etc. Due to the large existing base of non-Kerberized applications, normally, the application servers need to be configured to accept non-

Kerberized client applications. Suppose a user uses a non-Kerberized application, such as regular telnet, to telnet to the remote Kerberos-enhanced AIX or LINUX workstation. When the user types in his/her login ID and password, the user's password will be transferred to the remote machine in clear text. So if someone is eavesdropping on the Internet or CS department Ethernet, he/she can catch the user's password and impersonate the user later. When the user's login ID and password have been transferred to the remote machine's Kerberized login software, the process of obtaining the TGT is same as local login. If the user wants to use other services in the Kerberos realm from the CS department workstation, the user doesn't need to type his/her password anymore, since the user has a TGT for the whole Kerberos realm. That means if the user uses a regular telnet client to access a Kerberos-enhanced network, then it's safer than to use a regular telnet to access a non-Kerberos-enhanced network. For example, assume that Kerberos has been deployed on the CS department's LAN. When a user uses a regular telnet client to telnet to "garnet", the user's password will be transferred over the Internet in clear text once. Then the user has a TGT that resides on "garnet". Then if the user wants to telnet to other machines from "garnet", say, "ninepipe", the user doesn't need to type his/her password again. The user only needs to type "telnet ninepipe", the user will be logged in without supplying a password, since the user is using the Kerberized telnet client that resides in "garnet". If the CS department's LAN is not Kerberos-enhanced, then when the user telnets to "garnet", the user's password will be transferred over the Internet once. When the user wants to telnet to "ninepipe", the user has to type his/her password again, so the user's password will be transferred over the Internet another time. Although it's better, transferring the password in clear text one time is enough for eavesdropper to catch it. So I encourage people to use Kerberized client applications. Cygnus solution Inc. has a free package -- Kerbnet that includes kpasswd, kinit, ktelnet for Windows 95/NT. I wrote a GUI Kerberized FTP for Windows 95/NT.

- **Using Kerberized telnet**

To use Kerberized telnet on Windows 95/NT, "kinit" or "kerbnet" (GUI software which combines "kinit" and "kpasswd" together) is needed to obtain the TGT. Next, the Kerberized telnet can be launched which will automatically log the user in.

3.1.4 Password Synchronization and User Account Setup

The scenarios for logon to NT domain and for login to UNIX (AIX, LINUX) locally are quite different.

When a user logs in to UNIX, "klogin" will first try to match the password the user typed in with his/her Kerberos password, if the match succeeds, "klogin" will let the user login. If it does not succeed, "klogin" works like regular login that checks the password the user typed in with his/her local password. If it matches, "klogin" lets the user login, if not, the user has to retry.

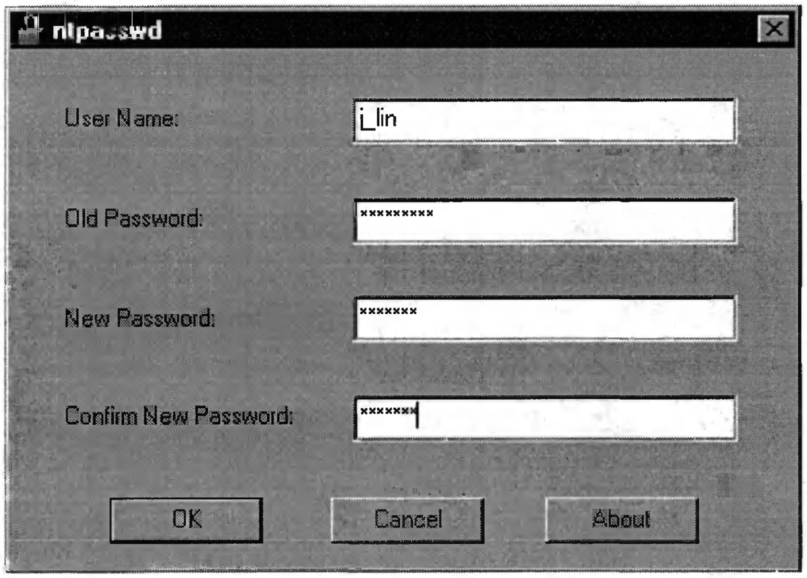
When the user logs on to an NT domain, "gina" will first try to match the password the user typed in with his/her NT domain password. If the password does not match, "gina" let the user retry. If the password does match, "gina" will try to obtain the TGT using the password the user typed in. If succeed, the user will logon and get the TGT. If not, depending on which installation option of "gina" is chosen, either the user can logon without obtaining the TGT or the user cannot logon with a chance to try again.

This leads to an important issue on setting up user accounts and password synchronization. Under the UNIX login scenario, the user only needs to know the Kerberos password, the local password can be generated randomly and the user doesn't need to know it. When users change their passwords, they only need to change the Kerberos password; the local passwords will never be changed. But on NT, due to the "gina" logon scenario, if the user wants to achieve single logon, the user's NT domain password and Kerberos password must be kept the same.

For example, if I'm the system administrator, I need to open a user account for a user, Joe. I can setup an user account "joe" on a workstation and generate a random password for "joe" and propagate this local account to every UNIX workstation. Then I need to setup a Kerberos user account "joe" for him and generate a Kerberos password for "joe", and Joe needs to know this Kerberos password. Then I setup an NT domain user account "joe" for Joe, where the NT domain password is the same as his Kerberos password. I also need to guarantee that the login ID is same for the NT domain and Kerberos.

When Joe wants to change his password, if he changes his password on UNIX, he only needs to change his Kerberos password. After he is done, he might go to an NT machine and want to logon, he can either logon without obtaining the TGT or simply can't logon depending on the "gina" installation option. So a mechanism is needed to synchronize NT domain password changes and Kerberos password changes. I wrote a Win32 GUI program "ntpasswd" which first changes the NT domain password and then makes a system call to call "kpasswd" to change the Kerberos password.

"ntpasswd" first prompts the user to input his/her old password and new password and confirmed new password, then calls the NetUserChangePassword() API function to change the NT domain password. It then detects the environment variables $HOMEDRIVE, $HOMEDIR, $KPASSWD_PATH, and creates a temporary file in the user's home directory. It outputs the user's old password and new password and confirmed new password to this file, then through $KPASSWD_PATH finds "kpasswd.exe", and makes a system call like "kpasswd < tmpfilename", and then deletes the temporary file. Normally, in an NT environment, each regular user may not have write permission on any directory except his/her own home directory, that's why I detect $HOMEDRIVE and $HOMEDIR. This guarantees that the creation of the temporary file will be successful. Normally, only the user has full control on his/her home directory; other users, even the administrator, don't have read rights on the user's home directory. Because the temporary file contains the user's password in clear text, if the system is setup with home directories owned by users, putting this temporary file into the user's home directory is safe. This means that after Kerberos has been deployed, if a user wants to change his/her password, and doesn't want to change it twice (once for Kerberos, once for NT domain), the user needs to go to an NT workstation and uses "ntpasswd" to change his/her password. See figure 3.1 for the interface of "ntpasswd".



User Name: j_lin

Old Password: **********

New Password: *******

Confirm New Password: *******

Figure 3.1 Interface of ntpasswd

3.1.5 The Cost of Deploying Kerberos

The source code and executable of Kerberos on UNIX is freely available from MIT. Cygnus Solution Inc. has an implementation of Kerberos on NT called **KerbNet** that is also freely downloadable. KerbNet contains a Kerberized telnet client for Windows 95/NT. I wrote a Kerberized FTP client for Windows 95/NT. I'm going to make it a freeware. After the Kerberos has been deployed, it should be fairly easy for users to use. It's also fairly easy for the CS students to download, install and use the Kerberized client software (such as Kerberized TELNET and FTP) on their personally owned PCs. Thus, all of the software is free.

3.2 Two Test Network Models

I have setup two test network models.

- **Model one**

I used a Pentium Pro 200 machine ("dillon") running Redhat LINUX 2.0 as a master KDC, a Pentium 120 machine ("glasgow") running Redhat LINUX 2.0 as an application server and client (see figure 3.2).

I installed MIT Kerberos Version 5 on "dillon". I downloaded the source code of MIT Kerberos Version 5 and compiled it to get the executable, since MIT only provides the source code for Kerberos Version 5. I installed Kerbnet Kerberos from

Cygnus solutions Inc. on "glasgow". I installed MIT Kerberos Version 5 on "glendive".

- **Model two**

I used a IBM PowerStation 220 machine – "gaesl" – running AIX4.1 as master KDC, a Pentium 200 machine – "dillon" – which is installed Redhat LINUX 2.0 as slave KDC and application server, a Pentium 120 machine – "glasgow" – which is installed Windows NT server 4.0 as client (see figure 3.3).

I installed Kerbnet Kerberos for AIX on "gaesl". I installed MIT Kerberos Version 5 into "dillon". I installed Kerbnet Kerberos for NT into "glasgow".

Both test network models met the goals I mentioned in Chapter 1, except for goal #2 -- "Users should be able to change their password from any workstation". The MIT Kerberos and Cygnus Kerberos work perfectly together. Actually, Cygnus just uses the MIT Kerberos source code and modifies it to be easy-to-install and more user-friendly. Cygnus also developed an NT Kerberos version. Both networks can accept both non-Kerberized clients and Kerberized clients. This will guarantee that a lot of non-Kerberized clients, especially in a Windows or Mac environment will still work. I tested ntpasswd on "glasgow", it works perfectly. It guarantees the synchronization of NT domain password change and Kerberos password change.

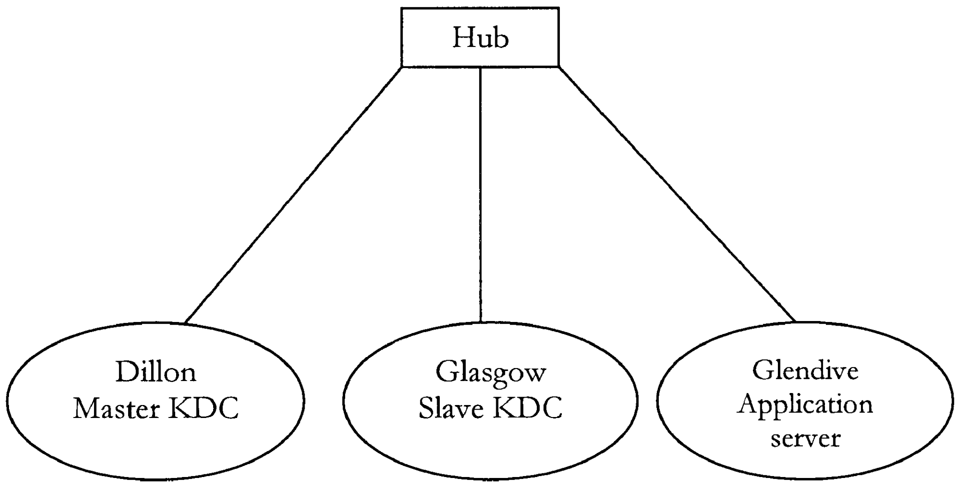


Figure 3.2 Model One

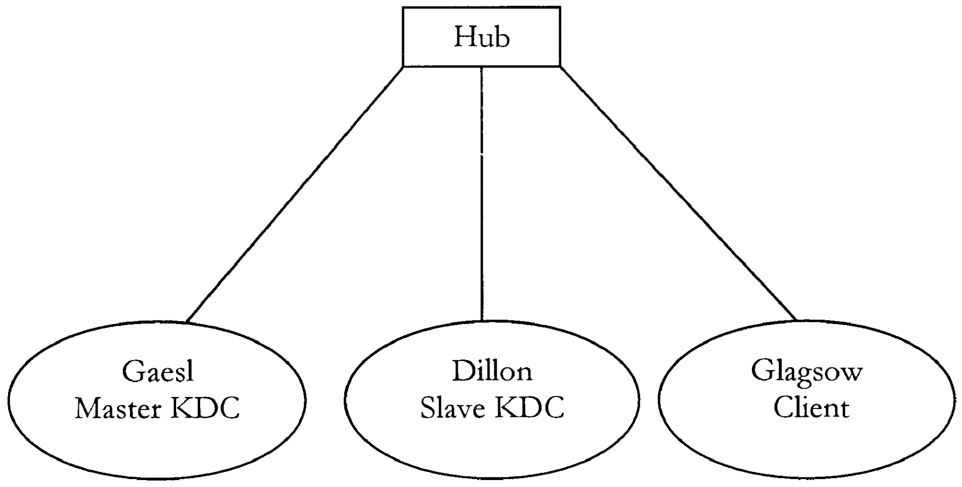


Figure 3.3 Model Two

Chapter 4

A Multi-threaded Implementation of Kerberized FTP for Windows 95/NT

4.1 Using "Kerberized" Applications

Client/server applications must be modified to use Kerberos for authentication. The process of modifying the applications to make it Kerberos-aware is called Kerberizing. Kerberizing an application is the most difficult part of installing Kerberos. Fortunately, the MIT Kerberos implementation includes most of the popular applications for Unix (the Berkeley R-commands, telnet, and POP) that have been Kerberized. Other applications have been Kerberized by vendors and are included in their supported products, such as Cygnus Solutions Inc.'s KerbNet System. The availability of Kerberized applications will improve with time.

It is generally necessary to modify the client/server protocol when Kerberizing an application, unless the protocol designer has already made provisions for an authentication exchange.

More recent implementations of Kerberos provide a Generic Security Services Application Programmer Interface (GSSAPI). *The GSSAPI provides a standard programming interface that is authentication mechanism independent. Using GSSAPI allows the application programmer to design an application and application protocol which can use alternative different authentication technologies, including Kerberos* [6]. The use of the GSSAPI in application programs is recommended wherever possible.

Because it is a generic authentication interface, the GSSAPI does not support all of the functionality provided by Kerberos. For example, Kerberos's notion of user-to-user authentication is not currently supported. Hence, an application programmer will not always be able to use the GSSAPI in all cases, and may have to use the Kerberos API in order to use some features.

4.2 Introduction to GSSAPI

The GSSAPI provides generic security services to its callers, and is intended for implementation on top of alternative underlying authentication protocols. Typically, GSSAPI callers will be application protocols into which security enhancements are integrated through invocation of services provided by the GSSAPI. The GSSAPI allows a caller application to authenticate a principal identity associated with a peer application, to delegate rights to a peer, and to apply security services such as confidentiality and integrity on a per-message basis.

There are four stages to using the GSSAPI:

- *The application acquires a set of credentials with which it may prove its identity to other processes. The application's credentials vouch for its global identity, which may or may not be related to the local username under which it is running.*
- *A pair of communicating applications establish a joint security context using their credentials. The security context is a pair of GSSAPI data structures that contain shared state information, which is required in order that per-message security services may be provided. As part of the establishment of a security context, the context initiator is authenticated to the responder, and may require that the responder is authenticated in turn. The initiator may optionally give the responder the right to initiate further security contexts. This transfer of rights is termed delegation, and is achieved by creating a set of credentials, similar to those used by the originating application, but which may be used by the responder. To establish and maintain the shared information that makes up the security context, certain GSSAPI calls will return a token data structure, which is a cryptographically protected opaque data type. The caller of such a GSSAPI routine is responsible for transferring the token to the peer application, which should then pass it to a corresponding GSSAPI routine which will decode it and extract the information.*
- *Per-message services are invoked to apply either:*

  (a). integrity and data origin authentication, or

  (b). confidentiality, integrity and data origin authentication to application data, which are treated by GSSAPI as arbitrary octet-strings. The application transmitting a message that it wishes to protect will call the appropriate GSSAPI routine (sign or seal) to apply protection, specifying the appropriate security context, and send the result to the receiving application. The receiver will

pass the received data to the corresponding decoding routine (verify or unseal) to remove the protection and validate the data.

- *At the completion of a communications session (which may extend across several connections), the peer applications call GSSAPI routines to delete the security context. Multiple contexts may also be used (either successively or simultaneously) within a single communications association* [6].

4.3 FTP Security Extension

4.3.1 Introduction

The File Transfer Protocol (FTP) currently defined in STD 9, RFC 959[7] uses usernames and passwords passed in clear text to authenticate clients to servers (via the USER and PASS commands), except for services such as "anonymous" FTP archives. This leads to a security risk that passwords can be stolen through monitoring of local and wide-area networks.

Except for the problem of authenticating users in a secure manner, there is also the problem of authenticating servers, protecting sensitive data and/or verifying its integrity. An attacker may be able to access valuable or sensitive data simply by monitoring a network. An active attacker may also initiate spurious file transfers to and from a site of the attacker's choice, and may invoke other commands on the server. FTP does not currently have any provision for the encryption or verification of the authenticity of commands, replies, or transferred data.

4.3.2 The FTP Security Extension Standard

FTP Security Extension Standard addresses (adapted from [8], RFC 2228):

1. How to authenticate users to servers in a secure manner.
2. How to protect command channel.

   Command **MIC** is used to protect the integrity of the command channel (Other people can not modify the command channel); Command **ENC** is used to protect the privacy of the command channel (Other people can not see the contents of the command channel).
3. How to protect data channel.

   Command **PROT** and **PBSZ** are used to protect the data channel.

The following commands are optional, but dependent on each other. They are extensions to the FTP Access Control Commands.

AUTH (Authentication/Security Mechanism)
ADAT (Authentication/Security Data)
PROT (Data Channel Protection Level)
PBSZ (Protection Buffer Size)
CCC (Clear Command Channel)
MIC (Integrity Protected Command)
CONF (Confidentiality Protected Command)
ENC (Privacy Protected Command)
---- from [8]

In my Kerberized FTP, I implemented **AUTH**, **ADAT** (for authentication), **MIC** and **ENC** (for protection of control channel).

4.4 Implementation of Kerberized FTP

The Kerberized FTP client is a GUI, multithreaded application for Windows 95 and Windows NT). This FTP client implements four new FTP extension commands (**AUTH**, **ADAT**, **MIC**, **ENC**) and all the basic regular FTP commands (such as **SEND**, **RECV**, **USER**, **LIST**, etc.).

Some features:

- **Can connect to Kerberized FTP servers and regular FTP servers:** Due to the large base of existing regular FTP server, the Kerberized FTP client can connect to both Kerberized FTP servers and regular FTP servers.
- **Multithreaded:** When sending, receiving, and listing files, a new thread will be launched to guarantee that the transfer of large amount of data won't block the GUI interface. So you can browse the local file system, see online help and "about" information while transferring a big file or a large directory.
- **Uses the Registry to maintain most recently connected hosts:** This uses the system registry database to store the most recently connected host names. So next time, when you want to connect to the host which has been connected before, you can just choose its name from the "host" droplist.

- **Display local and remote file system visually:** This uses two listview windows to display local and remote file system visually. You can browse local and remote file systems just like "WS-FTP".
- **Indicate the progress of file transfer:** This uses a progress bar to indicate the progress of file transfer visually and how much data has been transferred.
- **Beep to indicate file transfer finished:** After a file transfer is done, a beep will remind you that file transfer is done.

4.4.1 Implementation of the Authentication Part

To Kerberize an application, the most important part is the authentication part. The authentication part must be rewritten to support whatever authentication mechanism you want to support. Using GSSAPI will make this job a little easier but cannot achieve all the functionality provided by the underlying authentication protocol. But in general, GSSAPI will be sufficient for most situations. I used GSSAPI to write my Kerberized FTP client.

The following is the state transition diagram for the authentication part:

Two common paths:

- For non-Kerberized FTP server (follow — — — —▶ path).
- For Kerberized FTP server (follow --------> path).

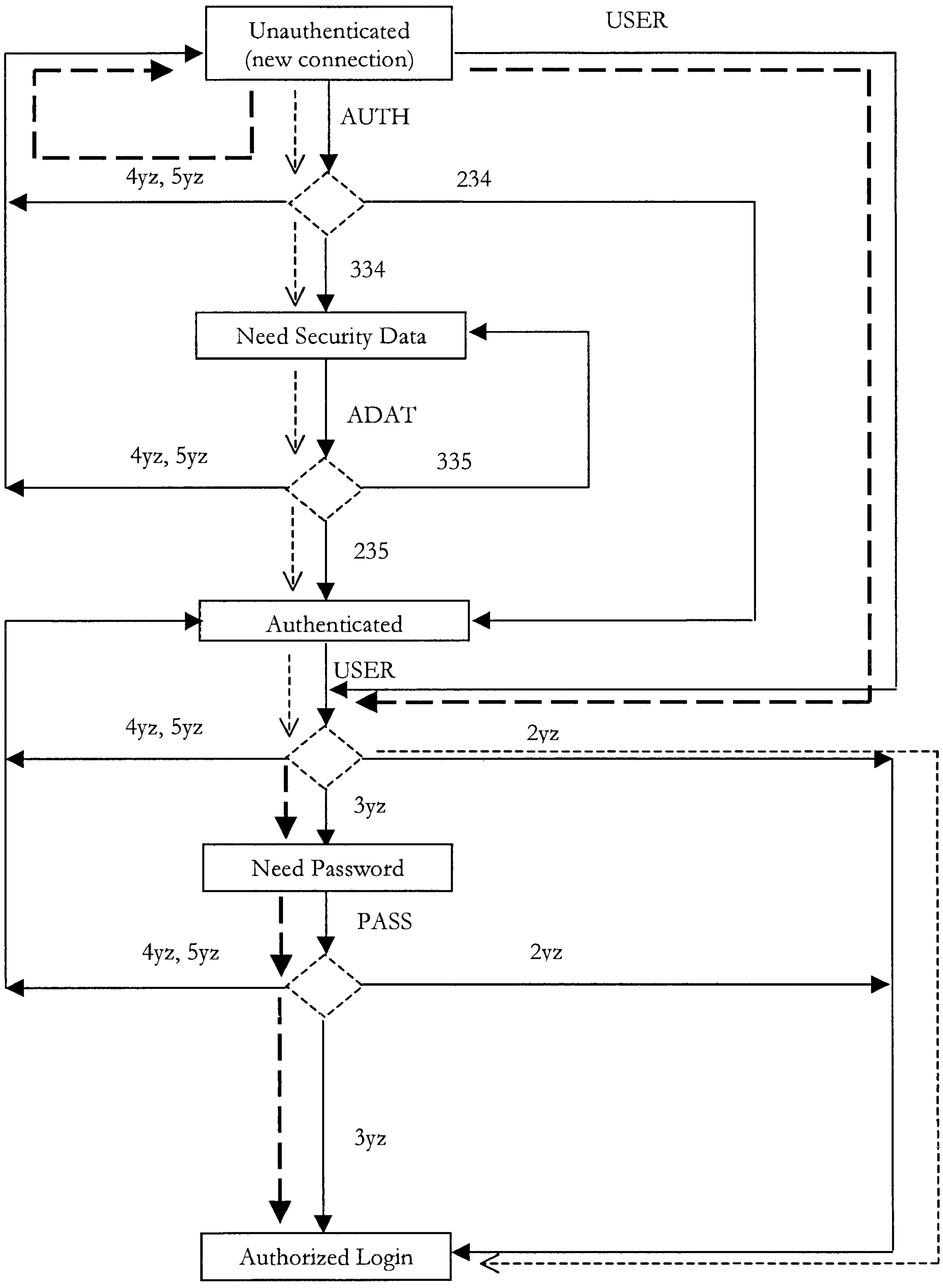


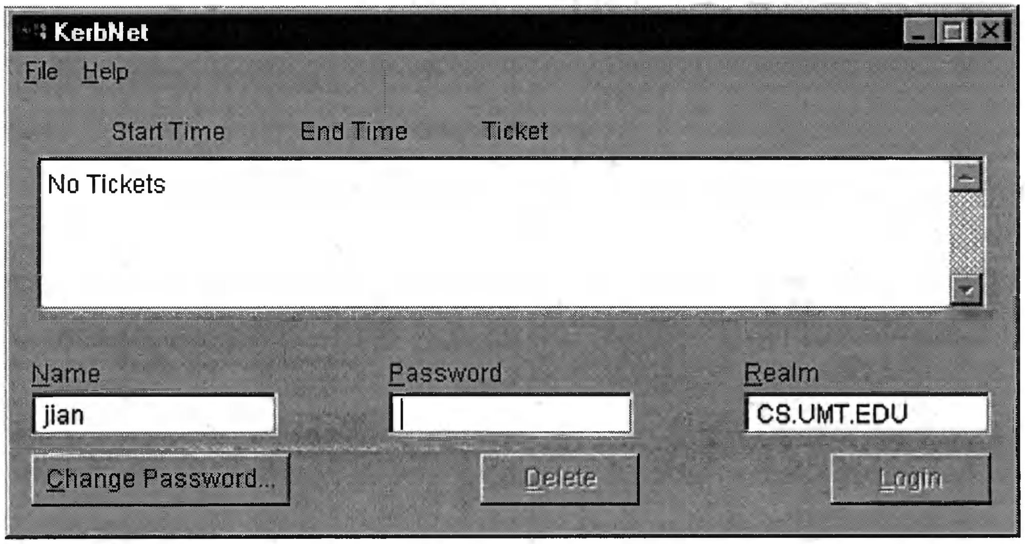
Figure 4.1: Login Authentication State Diagram

4.4.2 How to use the Kerberized FTP

This GUI Kerberized FTP client supports both Kerberized FTP server and regular FTP server. To use the FTP to open a connection to remote Kerberized FTP server, you have to obtain a *Ticket Granting Ticket* first using **Kerbnet System** explicitly, or if your operating system is Windows NT, you can obtain a *Ticket Granting Ticket* through the program **"gina"** which replaces the regular logon and obtains the ticket for you implicitly.

Here is how to use **Kerbnet System** to obtain *Ticket Granting Ticket.*

- After double clicking Kerbnet icon you get :

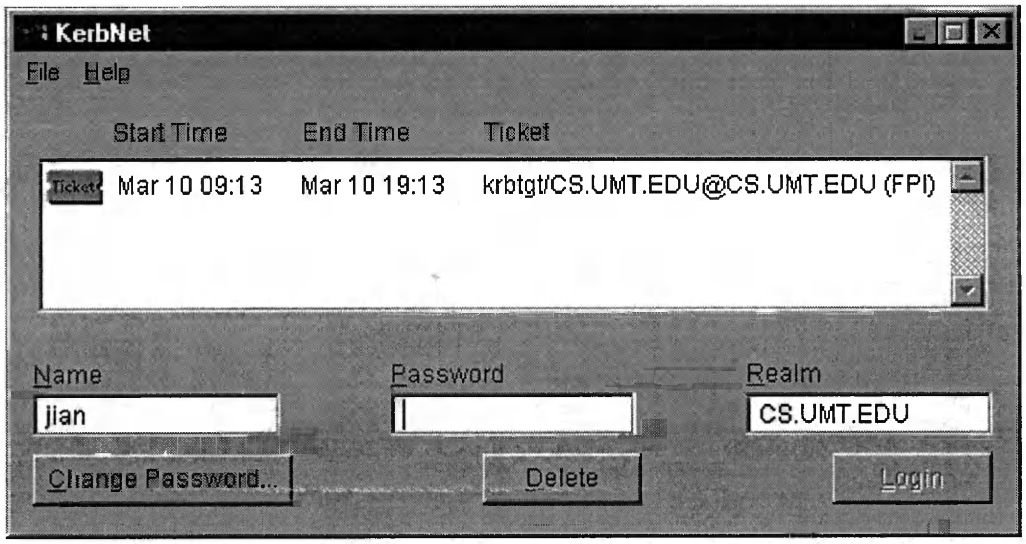


No Tickets

Name: jian
Password:
Realm: CS.UMT.EDU

Figure 4.2. Initial Kerbnet Interface

- After typing UserID, password, realm (only needed the very first time you use Kerbnet) and then click login button, you get:

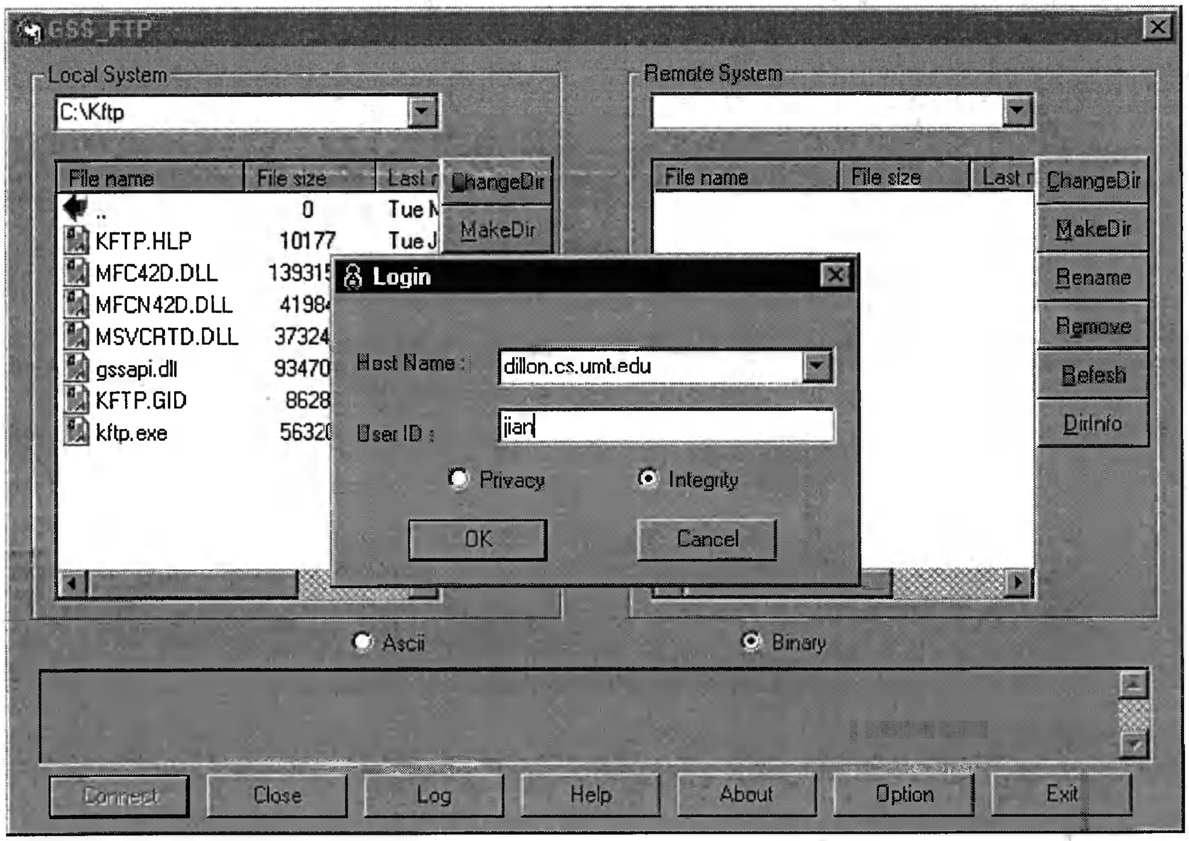


Mar 10 09:13 Mar 10 19:13 krbtgt/CS.UMT.EDU@CS.UMT.EDU (FPI)

Name: jian
Password:
Realm: CS.UMT.EDU

Figure 4.3. After obtaining Ticket Granting Ticket

- After you have obtained the Ticket Granting Ticket, you can launch Kerberized FTP by double clicking on its icon. There are two options in the login window, privacy and integrity. The integrity option is the default option, which guarantees the integrity of the command channel. With this option, intruders can not modify any FTP command. The privacy option encrypts the command channel, which guarantees both the integrity and the privacy. With this option, intruders can not see and modify any FTP command.



C:\Kftp

File name	File size	Last r
..	0	Tue N
KFTP.HLP	10177	Tue J
MFC42D.DLL	13931	
MFCN42D.DLL	4198	
MSVCRTD.DLL	37324	
gssapi.dll	93470	
KFTP.GID	8628	
kftp.exe	5632	

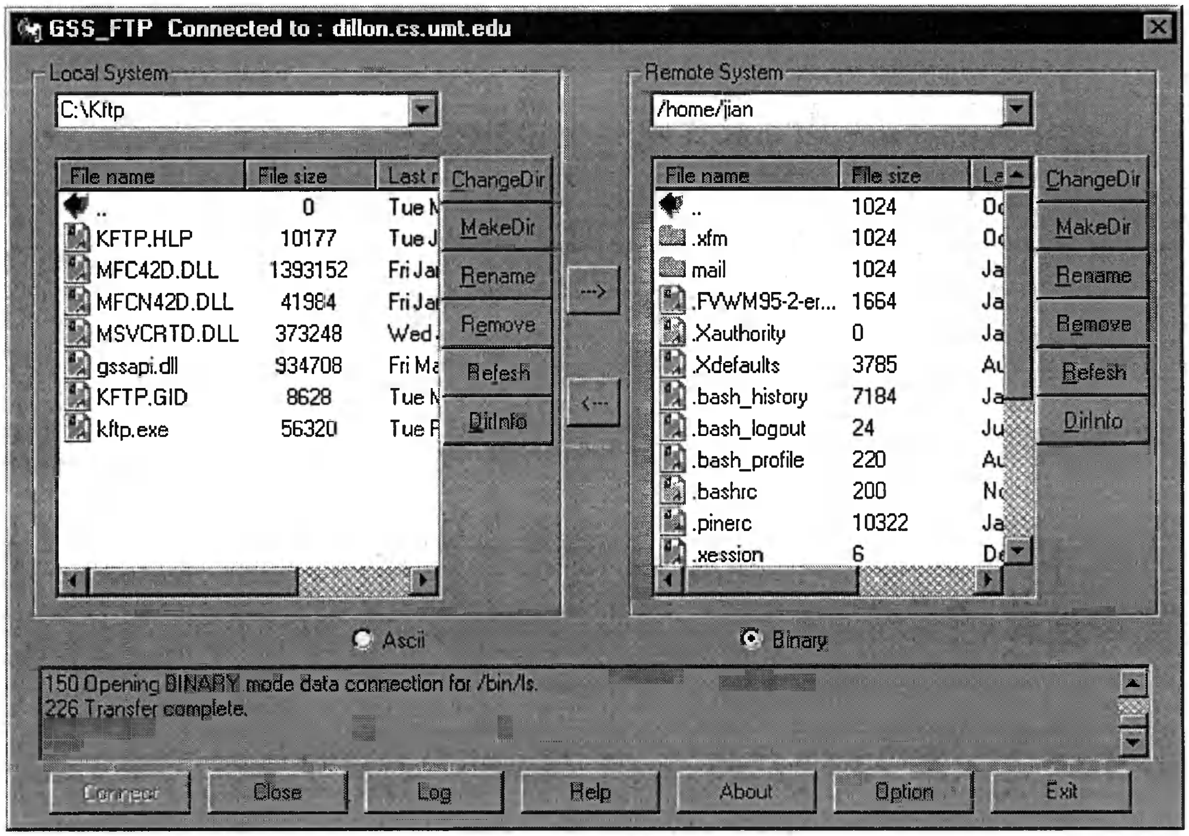
Host Name : dillon.cs.umt.edu

User ID : jian

Privacy [✓ Integrity]

Figure 4.4. GSS_FTP initial Interface

- Type or choose a host name and type your user name and click "OK" button.



C:\Kftp

File name	File size	Last r
..	0	Tue M
KFTP.HLP	10177	Tue J
MFC42D.DLL	1393152	Fri Ja
MFCN42D.DLL	41984	Fri Ja
MSVCRTD.DLL	373248	Wed
gssapi.dll	934708	Fri Ma
KFTP.GID	8628	Tue N
kftp.exe	56320	Tue F

/home/jian

File name	File size	La
..	1024	Oc
.xfm	1024	Oc
mail	1024	Ja
.FVWM95-2-er...	1664	Ja
.Xauthority	0	Ja
.Xdefaults	3785	Au
.bash_history	7184	Ja
.bash_logout	24	Ju
.bash_profile	220	Au
.bashrc	200	Nc
.pinerc	10322	Ja
.xession	6	De

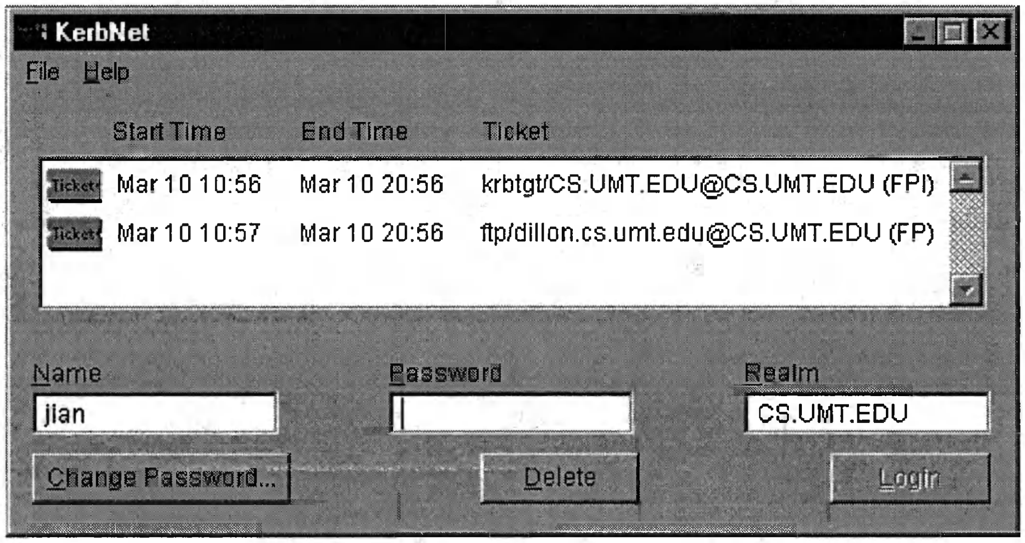
Ascii [✓ Binary]

```
150 Opening BINARY mode data connection for /bin/ls.
226 Transfer complete.
```

Figure 4.5. After Logon

- After you logon, you can recheck the Kerbnet window and see that the Ticket for the FTP server has been obtained.
- Then, you can browse the local and remote file systems through the two listview windows and choose what to transfer.

To know more about how to use this FTP client, please see the online help system of the Kerberized FTP.



	Start Time	End Time	Ticket
Ticket	Mar 10 10:56	Mar 10 20:56	krbtgt/CS.UMT.EDU@CS.UMT.EDU (FPI)
Ticket	Mar 10 10:57	Mar 10 20:56	ftp/dillon.cs.umt.edu@CS.UMT.EDU (FP)

Name: jian
Password:
Realm: CS.UMT.EDU

Figure 4.6. After obtaining FTP service ticket

Chapter 5

Conclusions

This project consists of two parts. The first part is to find a feasible solution to unify the authentication for the heterogeneous networks of the CS department at the University of Montana. The goals include:

- A global user account database, which means that any user must be able to login to the same account with the same password on any workstation
- Users should be able to change their password from any workstation
- Any system must work with a shadow password system on both AIX and LINUX
- Users should be able to access any workstation over the Internet (for example, a student or faculty should be able to login to a workstation using a PPP connection to an ISP from home)
- Unencrypted passwords should not be transferred over the network (including the LAN of CS department and the Internet)
- The solution should be relatively easy to use, maintain, and administrate

investigated and studied a lot about how Kerberos works, how to administrate Kerberos, how to make the use of Kerberos as transparent as possible. Chapter 2 covers most of these topics. I've set up two small test networks to test if Kerberos can really satisfy the CS department's requirements. Chapter 3 covers this. I also wrote a program "ntpasswd" which synchronizes an NT domain password and a Kerberos password. Note that the goal #2 – "Users should be able to change their passwords from any workstation" was not met.

There are Kerberos implementations both on UNIX and on NT which means that a UNIX or a NT machine can be used as the KDC. Due to the small size of the CS department at the University of Montana, the CS department won't want to spend a lot of money on the KDCs. Two Intel 486 machines should be enough for the KDCs. Because NT is not as stable as UNIX and NT's system overhead is bigger than UNIX, the performance of a Intel 486 machine running NT server 4.0 is much poorer than that of a Intel 486 machine running LINUX. So I recommend using two Intel 486 machine running LINUX as the KDCs.

Client/server applications must be modified to use Kerberos for authentication; such Kerberos-aware applications are said to be Kerberized. Kerberizing an application is the most difficult part of installing Kerberos. Fortunately, the MIT reference implementation includes the popular applications (the Berkeley R-commands, telnet, and POP) for a variety of UNIX platforms with support for

Kerberos already added. Cygnus Inc. also provides a free software package **Kerbnet for Win95/NT** that includes a program used for obtaining a Ticket Granting Ticket, a Kerberized telnet, and more. But they didn't provide a Kerberized FTP for Win95/NT. This leads to the second part of this project, developing a Kerberized FTP for Win95/NT. I investigated the FTP protocol, the FTP security extension protocol, GSSAPI and more. I used Visual C++ 5.0 and the Cygnus GSSAPI library to develop this GUI Kerberized FTP for Win95/NT. Chapter 4 covers this software.

Bibliography

[1] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks", *IEEE Communications Magazine, Volume 32, Number 9, pages 33-38, September 1994.*

[2] Jennifer G. Steiner, Clifford Neuman and Jeffrey I. Schiller, "Kerberos: An Authentication Service for Open Network Systems", *In Proceedings of the Winter 1988 Usenix Conference, pages 191-201, February 1988.*

[3] John T. Kohl, B. Clifford Neuman and Theodore Y. Ts'o, "The Evolution of the Kerberos Authentication Service", *In Distributed Open Systems, pages 78-94.* IEEE *Computer Society Press, 1994.*

[4] J. T. Kohl and B. C. Neuman. "The Kerberos network authentication service". *Internet RFC 1510, September 1993.*

[5] J. Linn, "Generic Security Service Application Program Interface, Version 2", *Internet RFC 2078, January 1997.*

[6] J. Wray, "Generic Security Service API : C-bindings", *Internet RFC 1509, September 1993.*

[7] J. Postel and J. Reynolds, "FILE TRANSFER PROTOCOL (FTP)", *Internet RFC 959, October 1985.*

[8] M. Horowitz, "FTP Security Extensions", *Internet RFC 2228, October 1997.*

[9] "Kerberos V5 Installation Guide", *shipped with Kerberos package.*

[10] "Kerberos V5 System Administrator's Guide", *shipped with Kerberos package.*

[11] "Kerberos V5 UNIX User's Guide", *shipped with Kerberos package.*

[12] Simson Garfinkel , Gene Spafford, "Practical Unix & Internet security" ,

O'Reilly & Associates, Inc. ISBN 1-56592-148-8, 1996.

[13] Bob Quinn, Dave Shute, "Windows Sockets Network Programming", *Addison-Wesley Publishing Company, ISBN 0-201-63372-8, 1995.*

[14] "Unifying UNIX and NT Security", *http://www.cygnus.com/product/unifying-security.html* (Now, this page is not available any more).

[15] Frederic J. Cooper, Chris Goggans, John K. Halvey, Larry Hughes, Lisa Morgan, Karanjit Siyan, William Stallings, Peter Stephenson, "Implementing Internet Security", *New Riders Publishing, ISBN 1-56205-471-6, 1995.*