Project Report
on
CALYPSO PLATFORM IN FINANCE DOMAIN

Submitted to the partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

Session 2011-12

UNITED COLLEGE OF ENGINEERING & RESEARCH, GREATER NOIDA (GAUTAM BUDDH TECHNICAL UNIVERSITY, LUCKNOW)

Submitted to: Mr. Jitendra Singh

Submitted by: Nishant Gaur (0819310053 )
B.TECH CS (VII SEM)
BONAFIDE CERTIFICATE

This is to certify that project Report entitled “SECURITY ENHANCEMENT USING KERBEROS”, which is submitted by Raghvendra Singh (10306882), Narendra Singh (10306869) and Piyush Kumar (10306874) in the partial fulfillment of the requirement for the award of degree B. Tech. in Department of Computer Science & Engineering of SRM UNIVERSITY, is a record of the candidate own work carried out by him under my own supervision.

..............................................                                                      ..............................................
(Signature)                                                                                (Signature)
Dr.R.P.Mahapatra                                                                            Mr. Jitendra Singh
HEAD OF THE DEPARTMENT                                                                 SUPERVISOR
C.S.E. DEPARTMAENT                                                                          Asst Professor (C.S.E. DEPT.)
SRM UNIVERSITY                                                                            SRM UNIVERSITY
NCR CAMPUS                                                                                NCR CAMPUS
MODINAGAR, GHAZIABAD                                                                       MODINAGAR, GHAZIABAD

INTERNAL EXAMINER                                                                         EXTERNAL EXAMINER

ACKNOWLEDGEMENT
We would like to express our hearty gratitude to DR. M. K. Pandey, DEAN SRM UNIVERSITY NCR CAMPUS, MODINAGAR, under whose auspices we were able to work on our project work.

Also, we extend our sincere regards and thanks to DR. R. P. Mahapatra, Head of the Department, Computer Science and Engineering SRM UNIVERSITY NCR CAMPUS, MODINAGAR, for his thoughtful and clairvoyant suggestion.

We feel compelled to articulate our thankfulness to MR. M. MOHAN and MR. A. KULOTHUNGAN, PROJECT COORDINATOR, SRM UNIVERSITY NCR CAMPUS, MODINAGAR, for his enlightening dexterity which was a perennial source of inspiration.

We would also like to express our special thanks to MR. Jitendra Singh, Asst Professor and project Supervisor, Mr. M. Mohan, Mr. Mohan Raj of SRM UNIVERSITY NCR CAMPUS, MODINAGAR, for their insight and knowledge on the subject, who imbued us with the feeling to work assiduously.

We are also indebted to all the teaching and non-teaching staff members of our college for helping us directly or indirectly by all means throughout the course of our study and project work.

Finally, we take this opportunity to thank our friends for their moral support and help and extended thanks to our well wishers.

Raghvendra Singh (10306882)
Narendra Singh (10306869)
Piyush Kumar (10306874)
DECLARATION

We, Raghvendra Singh (10306882), Narendra Singh (10306869) and Piyush Kumar (10306874) hereby declare that the work which is being presented in the project report “Security Enhancement Using kerberos” is the record of authentic work carried out by me during the period from January ’10 to May ’10 and submitted by me in partial fulfillment for the award of the degree “Bachelor of Technology” to SRM UNIVERSITY, NCR Campus, Ghaziabad (U.P.). This work has not been submitted to any other University or Institute for the award of any Degree/Diploma.

(Signature)                                       (Signature)                                (Signature)
Raghvendra Singh                        Narendra Singh                          Piyush Kumar
(10306882)                                     (10306869)                           (10306874)

ABSTRACT

iv
This project is aimed at providing a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. A free implementation of this protocol is available from the Massachusetts Institute of Technology. Kerberos is available in many commercial products as well.

In this project, we allow users to log in securely to computers which belong to a network.
This project assumes a network of computers, with one trusted computer or several verifying the identity of users to the computers to which they wish to log in, and supplying session keys so that the communications between the user and that computer can be encrypted.

Beginning a session with another computer under Kerberos requires the exchange of five messages:

- First, a message is sent in the clear to the master security server (also called Kerberos), consisting of the user's login ID and the identity of the security server (called a ticket-granting server) from which the user is requesting authentication to open a session with a computer on the network.
- Second, the security server generates a random session key and sends that to the user, encrypted with the user's secret key. (The user's secret key is a hash of the user's password.) It also sends the user, encrypted with its own permanent key, a "ticket", which includes a timestamp, the user's identity, the random session key divulged to the user previously within this message, and the service the user is requesting (of the security server).
- Third, the user sends an authenticator to the security server, which consists of the user's name and a timestamp, encrypted with the user's secret key. This authenticator is then encrypted with the session key before sending. The ticket, which identifies the user as authorized to request a ticket for a session with another computer, is sent back as well, and it is intended that the security server may safely rely on reading the ticket to find out what the session key is. Also, the name of the other computer the user wants to communicate with is sent.
- Fourth, the security server responds with another random session key, this one intended to be used with the requested computer, encrypted with the current session key for use between itself and the user. And it sends the user another "ticket"; this one includes a timestamp, the user's identity, the random session key for use with the requested computer, and the service requested of the requested computer. This ticket is enciphered with the permanent secret key of the requested computer.
Fifth, the user, now having a "ticket" for use with the requested computer, generates another authenticator, consisting of the user's name and a timestamp, encrypted with the session key for use with that computer, and also sends the "ticket", from which the requested computer can verify the user's identity, and discover what session key is to be used.

This project demonstrates implementation of a basic realm environment in which there is a Kerberos server connecting with a client and a server module to provide a mutual authentication between that client and server.

The project is implemented successfully by using the coding done in ‘Java’ Language for front end and MS-Office Acess for back end support.
List of figures…………………………………………………………… 5

Chapter 1:  Introduction………………………………………………..6-14
  1.1 About Kerberos
  1.2 History Of Kerberos
  1.3 Overview of Kerberos

Chapter 2 : Literature Survey…………………………………………15-20
  2.1 Nature Of Kerberos
  2.2 Modules
  2.3 How Kerberos Works

Chapter 3:  Hardware and Software Specification…………………..21-23

Chapter 4:  Problem Description…………………………………….24-27
  4.1 Existing Problem
  4.2 Proposed technique
    4.2.1 User Client Based Logan
    4.2.2 Client Authentication
    4.2.3 Client service authorization
    4.2.4 Client service request

Chapter 5:  Methodology……………………………………………28-39
  5.1 Project life cycle model
  5.2 Project schedule
  5.3 Data encryption standard

Chapter 6:  Implementation………………………………………….40-45
  6.1 Overall structure
  6.2 Execution Steps

Chapter 7:  Result……………………………………………………..46-55

Chapter 8:  Performance Evaluation…………………………………56-57

Chapter 9:  Conclusion and Future Work…………………………..58-59
  9.1 Conclusion
  9.2 Future Work
Chapter 10: References........................................60-61

LIST OF FIGURES

viii
CHAPTER -1

INTRODUCTION

1.1 About Kerberos
Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. A free implementation of this protocol is available from the Massachusetts Institute of Technology. Kerberos is available in many commercial products as well.

**The Internet is an insecure place.** Many of the protocols used in the Internet do not provide any security. Tools to "sniff" passwords off of the network are in common use by malicious hackers. Thus, applications which send an unencrypted password over the network are extremely vulnerable. Worse yet, other client/server applications rely on the client program to be "honest" about the identity of the user who is using it. Other applications rely on the client to restrict its activities to those which it is allowed to do, with no other enforcement by the server.

Some sites attempt to use firewalls to solve their network security problems. Unfortunately, firewalls assume that "the bad guys" are on the outside, which is often a very bad assumption. Most of the really damaging incidents of computer crime are carried out by insiders. Firewalls also have a significant disadvantage in that they restrict how your users can use the Internet. (After all, firewalls are simply a less extreme example of the dictum that there is nothing more secure than a computer which is not connected to the network --- and powered off!) In many places, these restrictions are simply unrealistic and unacceptable.

### 1.2 History of Kerberos

Kerberos was created by MIT as a **solution to these network security problems**. The Kerberos protocol uses **strong cryptography** so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server has used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.

Kerberos is freely available from MIT, under copyright permissions very similar those used for the BSD operating system and the X Window System. MIT provides Kerberos in source form so that anyone who wishes to use it may look over the code for themselves and assure themselves that the code is trustworthy. In addition, for those who prefer to rely on a professionally supported product, Kerberos is available as a product from many different vendors.

In summary, Kerberos is a solution to your network security problems. It provides the tools of authentication and strong cryptography over the network to help you secure your information systems across your entire enterprise. We
hope you find Kerberos as useful as it has been to us. At MIT, Kerberos has been invaluable to our Information/Technology architecture.

**Kerberos**

The Kerberos protocol, developed at the Massachusetts Institute of Technology as part of Project Athena, is used to allow users to log in securely to computers which belong to a network. This protocol assumes a network of computers, with one trusted computer or several verifying the identity of users to the computers to which they wish to log in, and supplying session keys so that the communications between the user and that computer can be encrypted. Beginning a session with another computer under Kerberos requires the exchange of five messages:

- First, a message is sent in the clear to the master security server (also called Kerberos), consisting of the user's login ID and the identity of the security server (called a ticket-granting server) from which the user is requesting authentication to open a session with a computer on the network.

- Second, the security server generates a random session key and sends that to the user, encrypted with the user's secret key. (The user's secret key is a hash of the user's password.) It also sends the user, encrypted with its own permanent key, a "ticket", which includes a timestamp, the user's identity, the random session key divulged to the user previously within this message, and the service the user is requesting (of the security server).

- Third, the user sends an authenticator to the security server, which consists of the user's name and a timestamp, encrypted with the user's secret key. This authenticator is then encrypted with the session key before sending. The ticket, which identifies the user as authorized to request a ticket for a session with another computer, is sent back as well, and it is intended that the security server may safely rely on reading the ticket to find out what the session key is. Also, the name of the other computer the user wants to communicate with is sent.

- Fourth, the security server responds with another random session key, this one intended to be used with the requested computer, encrypted with the current session key for use between itself and the user. And it sends the user another "ticket"; this one includes a timestamp, the user's identity, the random session key for use with the requested computer, and the service requested of the requested computer. This ticket is enciphered with the permanent secret key of the requested computer.
• Fifth, the user, now having a "ticket" for use with the requested computer, generates another authenticator, consisting of the user's name and a timestamp, encrypted with the session key for use with that computer, and also sends the "ticket", from which the requested computer can verify the user's identity, and discover what session key is to be used.

It should be noted that while the timestamp on a ticket is intended to prevent replay attacks, it is valid for a somewhat extended period of time; the timestamp on an authenticator is only valid for a very brief amount of time, as authenticators are intended to be only used once.

In Kerberos version 4, when a ticket is sent to a user, it is also encrypted under the key known to the user at that stage: by the user's secret key in the second message, and by the session key for use with the security server in the fourth message. In version 5, this encipherment, not being seen as necessary for security, is omitted to save computer time. In version 4, the PCBC mode of DES is used for encryption; in version 5, CBC mode is used, and there is provision for use of block ciphers other than DES. Incidentally, PCBC mode was used without the transmission of an initialization vector: instead, the DES key, known to both parties, served as the initialization vector without being transmitted.

Also, in Version 5, the user's secret key is derived not only from the user's password, but from "salt", consisting of the user's location on the system and a timestamp added to the first message. This makes dictionary attacks on passwords somewhat more difficult, but they are still possible. For this reason, and because there might be applications where security is desired, but there is no secure initial contact for communicating a password secretly to the user or the server, there are several proposals to improve and extend Kerberos by adding steps involving the use of public-key cryptography to it.

1.3 Overview of Kerberos

The following diagram may make it clearer what is happening in Kerberos:
In the first line, the user machine, colored red, sends a message in the clear identifying itself (a red N) and the ticket-granting server it wants to talk to (its name being a blue N) to the Kerberos server. The Kerberos server knows the permanent secret keys of all the users, shown by a red key beside it for the current user, and the permanent secret keys of all the ticket-granting servers, shown by a blue key beside it for the ticket-granting server requested. It responds by providing the user with a session key for communicating with the ticket-granting server, shown as a purple key. This key was generated by the Kerberos server at random, so it knows that key also, shown by a purple key popping out of thin air beside the server. The purple key is in a red box, which means that it is encrypted with the user's secret key. The user also receives a ticket, which is shown as another purple key, plus a red N, in a blue box: the red/blue session key, and the name of the user, encrypted in the ticket-granting server's key.

The user then presents the ticket to the ticket-granting server, along with an authenticator - a copy of its own name, and a timestamp, encrypted in the session key. As well, it sends a green N to the ticket-granting server, to identify the computer for which it wants a ticket. The ticket-granting server can un wrap the ticket, since the blue key is its own secret key, and finding the session key it can verify the authenticator. As well as knowing a secret key of its own, it knows the secret keys of the ordinary computers it serves, shown by a green key, the secret key of the computer requested in the example session.

The ticket-granting server then follows in the footsteps of the Kerberos server, generating a session key (this time shown as brown) for communications.
between the user and the computer the user had originally set out to use. This
session key is given to the user enciphered in the purple session key given to the
ticket-granting server and the user by the Kerberos server; it is also enclosed,
with the user's name, in a ticket, which this time is in a green box, that is,
enciphered in the permanent secret key of the computer the user wanted to use.
Finally, the user presents the ticket, and an authenticator, to the computational
server, thus providing it with the brown key which will protect their productive
interactive session.
Note that in the diagrams, the ticket is shown before the authenticator, since the
receiving server needs to unwrap the ticket before it can verify the
authenticator. Also, it might be noted that the use of a timestamp in the
authenticator is what distinguishes and simplifies Kerberos; a more traditional
protocol might have the server generate a random number, send it to the user,
and the user then sends it back encrypted with the session key inside the ticket
for authentication. The server can be requested to authenticate itself; in
Kerberos, this is done by sending the timestamp plus one in the session key.

**Private key cryptography and trusted third parties**

The cryptographic algorithm typically employed in Kerberos is Data Encryption
Standard (DES), although the modular design of Kerberos allows alternative
encryption libraries to be used. DES is a standard algorithm for "private-key"
cryptography.
The term "private key" cryptography (also called "secret key" or "symmetric
key" cryptography) refers to an approach where the pair of entities exchanging
messages share a single key used to encode and decode messages. This is in
contrast to "public key" (or "asymmetric key") cryptography, where the
encoding and decoding keys are separate and difficult to derive from each other,
allowing one of the keys to be made publicly available. Private-key
cryptography has the advantage of efficiency, but poses the fundamental
problem of initially communicating the key to be used. For this reason, the two
techniques are often combined.
In both private-key and public-key cryptography, there is generally a practical
need to rely on trusted third parties to certify identity. In Kerberos, a central
authentication server certifies the identities of all entities (where an entity may
be a user, a client application operating on behalf of a specific user, or a service
provided by an application server). The functionality of Kerberos completely
relies on the trustworthiness of the authentication server, which must know the
password or key of every entity. It is therefore of paramount importance that the
authentication server itself be completely secure.
Kerberos ticket-granting approach-

An essential conflict exists between the need to communicate the encryption key, and the need to keep the key secret. There is also a trade-off between using a unique key for every message and re-using a single key for an extended session of interactions (this trade-off potentially impacts both efficiency and convenience, vs. the level of protection provided).

The Kerberos system's approach involves several layers of messages. Although these multiple layers may seem overly elaborate at first glance, they represent a reasonable attempt to address the design trade-offs.

The following is a simplified description of the Kerberos message scheme (illustrated in Figures 1-3). In this discussion, the term "client" refers to a user or to a client-side application program operating on behalf of a user. As noted above, the term "application" server refers to a remote computer which provides a shared service. The term "password" refers to the user's password or to a cryptographic key derived from it (since the process of deriving the key from the password is transparent to the user). The term "session" key refers to a cryptographic key issued for use in communication between a client and an application server, which is valid for some defined interval of time.

Two basic elements are used in Kerberos messages: a ticket and an authenticator. A ticket is acquired by a client from the authentication server, and sent to the application server as part of the request for a service. An authenticator is constructed by the client, is sent to the application server along with the ticket, and is used by the application server to verify that the request was sent by the client to whom the ticket was issued.

As noted above, the goals of the system include 1) avoiding repeated re-entry of passwords and 2) sending unencrypted passwords over the network. To address these goals, the initial exchange with the Kerberos authentication server issues the user a "ticket-granting ticket" (TGT) containing a session key to be used for subsequent ticket requests.

Each TGT is good for a fixed life span, typically set to 8 hours. The TGT effectively serves as a proxy for the user with the Ticket Granting Service over the life of the TGT. This prevents the user from having to authenticate themselves every time they wish to access a service on the network.
The initial message from the client to the authentication server is typically sent automatically at the time of login. The initial message contains the user's name (but not password) and request for a TGT (Figure 1). The authentication server replies with a message encrypted with the user's password and containing a TGT (Figure 1). Because the TGT message is encrypted with the user's password, it is protected if intercepted. When received by the client, the TGT message can be decoded to obtain the session key to be used for subsequent ticket requests. This session key for ticket-granting requests is referred to below as the TG key.

Once the client has a TG key, the subsequent requests for a specific service will take the form shown in Figure 2.
The client sends a request to the Ticket Granting Service (TGS) to obtain a ticket for the service. The Ticket Granting Service can compare the client identification information embedded in the message with its record of issuing the TG key. The timestamp helps to protect against any attempt to re-use an intercepted message; typically, a request will be considered invalid if received more than 5 minutes after the embedded timestamp. This prevents messages from being intercepted, cracked offline, and then used at a later time. After confirming the client’s identity, the Ticket Granting Service responds with a message containing a new session key.

The ticket itself contains information identifying the client and the newly generated session key. Note that the client cannot decode or alter the ticket, only forward it to the application server.

Figure 4.

The client then sends a message to the application server containing the ticket and an authenticator (Figure 3).

As noted above, the authenticator is constructed by the client, and contains identifying information and a timestamp. When this message is received by the application server, it can decode the ticket, because it is encrypted with the server’s own key (known only to itself and the authentication server). The ticket contains the session key which is used to decode the authenticator. For the request to be considered valid, the data embedded in the authenticator must match that in the ticket.

Subsequent messages between the client and the application server may be encrypted using the session key.
2.1 Nature of Kerberos

Kerberos mainly works on symmetric key cryptography authentication. This protocol is in three parts each of which is now explained. The protocol involves a user \( U \) and four computer principals: a client \( C \); a server \( S \) with whom \( C \) wishes to communicate; and two trusted servers \( G \) and \( A \). \( G \) is known as a Ticket Granting Server and provides keys for communication between clients such as \( C \) and servers such as \( S \). \( A \) is known as the Key Distribution Centre and provides keys for communication between clients such as \( C \) and ticket granting servers such as \( G \). The full protocol has three parts each consisting of two messages between the client \( C \) and each of the servers in turn as shown in figure 10. In the protocol descriptions that follow shared secret keys are written with subscripts of the principals who share (or who will share) them. Thus, \( K_{cg} \) denotes the key for secure communication between \( C \) and \( G \). We use \( K_u \) to denote the key used to encrypt communications between \( A \) and \( C \) on behalf of the user \( U \). It is a key obtained by hashing \( U \)'s password. \( A \) stores this password, \( C \) will request it from \( U \).

2.2 Modules

Our project mainly consists of three modules

- CLIENT
- KERBEROS (AS & TGS)
- SERVER

In each module we use DES Algorithm for encryption and decryption purpose.

2.3 How Kerberos Works

- A password is shared between the user and KDC.
- Credentials are called Tickets.
- Credentials are saved in cache.
- Initial credentials request is for a special ticket granting ticket (TGT).

The first part of the protocol concerns only \( C \) and \( A \).

1. \( C \rightarrow A : U; G; L_1; N_1 \)
2. \( A \rightarrow C : U; T_{cg}; E(K_u : G; K_{cg}; T_{start}; T_{expire}; N_1) \)
where

\[ T_{cg} = E(K_{ag} : U; C, G; K_{cg}; T_{start}; T_{expire}) \]

In message (1) the client C informs the key distribution centre A that he wishes to communicate on behalf of user U with the ticket granting server G. A lifetime L and a nonce N1 are sent too. A generates a new key K_{cg} for this purpose and encrypts it under the key K_{u} it shares (or will share when U enters his password) with C. It also forms a ’ticket’

\[ \text{Tcg } \]

that contains the user identity, the client identity, the identity of G, the new key K_{cg} together with timestamp information. The T_{start}; T_{expire} limit the interval over which the ticket is considered as valid. This ticket is encrypted using the key K_{ag} shared between A and G. C uses the key K_{u} to decrypt the third component of message (2) and obtains the key K_{cg} which it can now use to communicate with G.

This is carried out in the second part of the protocol described below:

\[(3) \ C \rightarrow G : S; L_2; N_2; T_{cg}; A_{cg} \]

\[(4) \ G \rightarrow C : U, T_{cs}, E(K_{cg} : SK_{cs}; T_{start}; T_{expire}; N_2) \]

where

\[ A_{cg} = E(K_{cg} : C; T) \]

\[ T_{cs} = E(K_{cg} : U, C, S; K_{cs}, T_{start}; T_{expire}) \]

The result of the above is that G issues C with a ticket T_{cs} and a key K_{cs} to communicate with S. The authenticator A_{cg} ensures timeliness of message(3).

In the third part of the protocol C uses the newly obtained key K_{cs} and ticket T_{cs} to obtain the services of S.
(5) \( C \rightarrow S : T_{cs}, A_{cs} \)
(6) \( S \rightarrow C : E(K_{cs} : T') \)

where
\[ A_{cs} = E(K_{cs} : C, T') \]

He forms further authenticator \( A_{cs} \), and sends the result to \( S \) together with the newly acquired encrypted ticket as message (5). \( S \) carries out decryption on the ticket to obtain the session key \( K_{cs} \) and then uses this key to obtain the authentication information. If everything is in order, message (6) is returned.

**Kerberos Third-party Authentication Model**

The Kerberos authentication model relies on a secret-key symmetric encryption scheme (DES in the case of Kerberos IV, DES/IDEA/etc. in the case of Kerberos V) and the concept of dual encryption to provide secure authentication across a possibly insecure network. Authentication tickets are delivered to Kerberos clients encrypted in two keys -- one which both the client user and the ticket granting service know (either the user's password or a session key -- see below) and one which both the ticket-granting service and the target service know.

If a client machine is able to decrypt a ticket encrypted in the user's password, the user of the client may be considered authenticated (since only he and the authentication service know the user's password). If a target service is able to decrypt an encrypted ticket using its own secret key, the service may presume that the user who presented the ticket is authentic, since only the ticket-granting service and the target service have knowledge of the target service's secret key.

This allows authentication to occur under the Kerberos model without any password information passing over a possibly insecure network and without any one system or party in the Kerberized exchange having access to enough information to impersonate any other system or party.

Kerberos authentication can be viewed as a six-step process. Below is our Figure III, again, graphically depicting the steps involved in a typical Kerberized authentication session:
Figure 6- Kerberos third party authentication model.

In Step 1, the user wishing access to an authenticated target service provides his principal (username) and his password to the client system he is using. Note that the client system has no record of the user's principal or password -- it merely accepts a \([\text{principal, password}]\) pair provided by the user.

In Step 2, the client system sends a request to the Kerberos Initial Ticketing Service requesting a ticket-granting ticket for the user whose principal it has been given. This request is totally unauthenticated.

In Step 3, the Initial Ticketing Service creates a unique session key (K session) for later use during the user's authenticated session and sends back to the client a dual-encrypted ticket-granting ticket and the session key in the form:

\[
\{\{\text{Ttgs, Ksession}\} \text{Ktgs, Ksession}\} \text{Kuser}
\]

The client then uses the agreed-upon string_to_key() function to convert the user's password into an encryption key and attempts to decrypt the ticket-granting ticket using that key. If the decryption succeeds, the client can be certain that the user is authentic, and the client records the TGT (\{"Ttgs,Ksession\}Ktgs\) for later use.

In Step 4, when the user attempts to use a particular target service, the client sends a service ticket request to the Kerberos ticket granting service. This request is in the form:

\[
\{\text{TGT}, \{\text{request, client-IP, time}\} \text{Ksession}\}
\]

(where TGT = \{"Ttgs,Ksession\}Ktgs\)

In Step 5, the Kerberos ticket-granting service uses its own secret key (Ktgs) to decrypt the TGT in the request it has received, then uses the session key
(Ksession) in that TGT to decrypt the rest of the request. If the ticket-granting service is able to properly decrypt the ticket, it knows:

- that the TGT was issued by the correct initial ticketing service, since it was encrypted with the ticket-granting service's secret key.
- that the request it received was sent by an authenticated user (and, due to the contents of the Ttgs, what principal the ticket was issued for), since the sender was able to encrypt the request in the secret session key, Ksession.

If the decryption is successful, then, the ticket-granting service accepts the user's authentication, generates a service-session key for later use in encrypting transactions between the client and the target service, and issues a service ticket for the requested target service. This is sent back to the client machine in the form:

`{{Tservice,Kservice-session}Kservice,Kservice-session}Ksession`

In Step 6, the client decrypts the service ticket it has received using the session key (Ksession) provided to it in Step 3 to yield the service-session key (Kservice-session) and an encrypted service ticket (`{Tservice,Kservice-session}Kservice`).

The client then presents the encrypted service ticket to the target service, which in turn decrypts it using its own secret key (Kservice). If the decryption works, the target service may presume that the user is authentic (since only an authenticated user would have access to a decryptable service key). The client and target service may choose to further secure their later communications by encrypting their conversations in the service-session key issued by the ticket-granting service (Kservice-session), since it is a secret known only to the client and the target service. This latter step, encrypting the authenticated conversation, has the added advantage of allowing the client and server to be certain of one another's identities -- an interloper attempting to sabotage the authenticated conversation would need to know the shared encryption key (Kservice-session) in order to properly encrypt data to inject into the conversation.

Steps 4, 5, and 6 may be repeated by the client in order to allow the user access to other Kerberized services without the user's having to re-enter his principal or password information. Kerberos tickets are, in that sense, a reusable proof of authentication.

For more details about the Kerberos authentication model and for a review of the advantages and weaknesses of the model.
CHAPTER 3
**HARDWARE AND SOFTWARE SPECIFICATION**

**General Requirement for Server/Client:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
</table>
| Work Station/Node | 1. Windows XP  
2. Java Run time  
3. MS Office | 1. P-3  
2. RAM -512 MB |
| Database        | 1. MS ACCESSS  | 1. P-3  
2. RAM- 512MB  
3. Hard Disk-40GB |
2. RAM- 1GB  
3. Hard Disk-40GB |

**Development Tools and Technologies**

**Front End**

- JAVA

**Back End**

- MS Access
CHAPTER 4

PROBLEM DESCRIPTION
4.1 Existing Problem:

- Internet protocol dependence: Kerberos requires the use of the internet protocol addresses.

- Ticket Lifetime: Lifetime values in Kerberos are encoded in an 8 bit quantity in units of five minutes. Thus the maximum lifetime that can be expressed is $28 \times 5 = 1280$ minutes. Or little over 21 hours.

- Authentication forwarding: Kerberos does not allow credentials issued to one client to be forwarded to some other host and used by other client.

- Single point of failure: It requires continuous availability of a central server. When the Kerberos server is down, no one can log in. This can be mitigated by using multiple Kerberos servers and fallback authentication mechanisms.

- Kerberos requires the clocks of the involved hosts to be synchronized. The tickets have a time availability period and if the host clock is not synchronized with the Kerberos server clock, the authentication will fail. The default configuration requires that clock times are no more than five minutes apart. In practice Network Time Protocol daemons are usually used to keep the host clocks synchronized.

- The administration protocol is not standardized and differs between server implementation.

- Since all authentication is controlled by a centralized KDC, compromise of this authentication infrastructure will allow an attacker to impersonate any user.

Kerberos Limitations
Assumes running on trusted hosts with an untrusted network
• Attacker can impersonate users who have tickets stored on a machine
• If KDC is compromised, entire realm is compromised
• Offline attack to decrypt TGT with different passwords
• System clock synchronization is needed

4.2 Proposed Technique

Protocol on which Kerberos works –

The following is an intuitive description. The client authenticates itself to the Authentication Server and receives a ticket. (All tickets are time-stamped.) It then contacts the Ticket Granting Server, and using the ticket it demonstrates its identity and asks for a service. If the client is eligible for the service, then the Ticket Granting Server sends another ticket to client. The client then contacts the Service Server, and using this ticket it proves that it has been approved to receive the service. A simplified and more detailed description of the protocol follows. The following abbreviations are used:

- AS = Authentication Server
- SS = Service Server
- TGS = Ticket-Granting Server
- TGT = Ticket-Granting Ticket

The client authenticates to the AS once using a long-term shared secret (e.g. a password) and receives a TGT from the AS. Later, when the client wants to contact some SS, it can (re)use this ticket to get additional tickets from TGS, for SS, without resorting to using the shared secret. These tickets can be used to prove authentication to SS. The phases are detailed below.

4.2.1 User Client-based Logon-

1. A user enters a username and password on the client machine.
2. The client performs a one-way function (hash usually) on the entered password, and this becomes the secret key of the client/user.
4.2.2 Client Authentication-

1. The client sends a clear text message of the user ID to the AS requesting services on behalf of the user. (Note: Neither the secret key nor the password is sent to the AS.) The AS generates the secret key by hashing the password of the user found at the database (e.g. Active Directory in Windows Server).

2. The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
   - Message A: Client/TGS Session Key encrypted using the secret key of the client/user.
   - Message B: Ticket-Granting Ticket (which includes the client ID, client network address, ticket validity period, and the client/TGS session key) encrypted using the secret key of the TGS.

3. Once the client receives messages A and B, it decrypts message A to obtain the Client/TGS Session Key. This session key is used for further communications with the TGS. (Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key.) At this point, the client has enough information to authenticate itself to the TGS.

4.2.3 Client Service Authorization-

1. When requesting services, the client sends the following two messages to the TGS:
   - Message C: Composed of the TGT from message B and the ID of the requested service.
   - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the Client/TGS Session Key.

2. Upon receiving messages C and D, the TGS retrieves message B out of message C. It decrypts message B using the TGS secret key. This gives it the "client/TGS session key". Using this key, the TGS decrypts message D (Authenticator) and sends the following two messages to the client:
   - Message E: Client-to-server ticket (which includes the client ID, client network address, validity period and Client/Server Session Key) encrypted using the service's secret key.
   - Message F: Client/server session key encrypted with the Client/TGS Session Key.

4.2.4 Client Service Request-

1. Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
   - Message E from the previous step (the client-to-server ticket, encrypted using service's secret key).

xxxii
Message G: a new Authenticator, which includes the client ID, timestamp and is encrypted using client/server session key.

2. The SS decrypts the ticket using its own secret key to retrieve the Client/Server Session Key. Using the sessions key, SS decrypts the Authenticator and sends the following message to the client to confirm its true identity and willingness to serve the client:
   - Message H: the timestamp found in client's Authenticator plus 1, encrypted using the Client/Server Session Key.

3. The client decrypts the confirmation using the Client/Server Session Key and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.

4. The server provides the requested services to the client.

Use of DES Algorithm:

Main Paradigms:

Initial permutation IP

DES round structure

DES key schedule

32 BIT swap

Initial inverse permutation.
CHAPTER 5

METHODOLOGY
5.1 Project Life Cycle Model

Software development projects are complex. To deal with these complexities, many developers adhere to a core set of development principles. These principles define the field of software engineering. A major component of this field is the lifecycle model. The lifecycle model describes steps to follow when developing software from the initial concept to the release, maintenance and subsequent upgrading of the software.

Many different lifecycle models currently exist. Each has advantages and disadvantages in terms of time-to-release, quality and risk management. This section describes some of the most common models used in software engineering. Many hybrids of these models exist, so you can customize these models to fit the requirements of a project.

The lifecycle model is a foundation for the entire development process. Good decisions can improve the quality of the software you develop and decrease the time it takes to develop it.

WATERFALL MODEL

The waterfall model is the classic model of software engineering. This is one of the oldest models and is widely used in government projects and in many major companies. Because the model emphasizes planning in the early stages, it can design flaws before they develop. Also, because the model is document and planning intensive, it works well for projects in which quality control is a major concern.

The pure lifecycle consists of several non-overlapping stages, as shown in the following figure. The model begins with establishing system requirements and software requirements and continues with architectural design, detailed design,
coding, testing and maintenance. The waterfall model serves as a baseline for many other lifecycle models.

The following list details the steps for using the waterfall model:

- **System requirements** - Establishes the components for building the system, including the hardware requirements, software tools and other necessary components. Example include decisions on hardware, such as plug in boards (number of channels, acquisition speed and so on), and decisions on external pieces of software, such as databases or libraries.

- **Software requirements** - It establishes the expectations for software functionality and identities which system requirements the software affects. Requirements analysis includes determining interaction needed with other applications and databases, performance requirements, user interface requirements and so on.
• **Architecture design**- It determines the software framework of a system to meet the specified requirements. The design defines the major components and the limitation of those components, but the design does not define the structure of each component. You also determine the external interfaces and tools to use in the project.

• **Detailed design**- It examines the software components defined in the architectural design stage. Produces a specification for how each component is implemented.

• **Coding**- It implements the detailed design specification.

• **Testing**- It determines whether the software meets the specified requirements and find any errors present in the code.

• **Maintenance**- Addresses problems and enhancement requests after the software releases. In some organizations, a change control board maintains the quality of the product by reviewing each change made in the maintenance stage. Consider applying the full waterfall development cycle model when correcting problems or implementing these enhancement requests.

In each stage, you create documents that explain the objectives and describe the requirements for that phase. At the end of each stage, you hold a review to determine whether the project can proceed to the next stage. You also can incorporate prototyping into any stage from the architectural design and after.

Many people believe you cannot apply this model to all situations. For example with the pure waterfall model, you must state the complete design before you
begin coding. There is no overlap between stages. In real world development, however you can discover issues during the design or coding stages that point out errors or gaps in the requirements.

The waterfall method does not prohibit returning to an earlier phase, for example from the design phase to the requirement phase. However this involves costly rework. Each completed phase requires formal reviews and extensive documentation development. Thus oversight made in the requirements phase are expensive to correct later. Because the actual development comes late in the process, you do not see results for a long time. This delay can be dis concerning to management and to the customers.

5.2 PROJECT SCHEDULE

Whenever working on a project, time limit is an important constraint, to fight with any sort of latency, the thing should be scheduled in order to achieve them on time. Here are two techniques used for scheduling, both of them are standard procedures of software engineering.

PERT CHART
On the other hand pert chart follows scheduling in context of different components of a particular project which is being under development. The charts of the schedule have been shown below.
Figure 8—Pert chart
A gantt chart is a procedure, which describes the schedule for various phases of software life cycle with context of standard model that we are following in project making. It follows a general behaviour rather than that of particular software.

5.3 Data Encryption Standard

The Data Encryption Standard (DES) is a block cipher (a form of shared secret encryption) that was selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) for the United States in 1976 and which has subsequently enjoyed widespread use internationally. It is based on a symmetric-key algorithm that uses a 56-bit key. The algorithm was initially controversial with classified design elements, a relatively short key.
length, and suspicions about a National Security Agency (NSA) backdoor. DES consequently came under intense academic scrutiny which motivated the modern understanding of block ciphers and their cryptanalysis. DES is now considered to be insecure for many applications. This is chiefly due to the 56-bit key size being too small; in January, 1999, distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes (see chronology). There are also some analytical results which demonstrate theoretical weaknesses in the cipher, although they are unfeasible to mount in practice. The algorithm is believed to be practically secure in the form of Triple DES, although there are theoretical attacks. In recent years, the cipher has been superseded by the Advanced Encryption Standard (AES). Furthermore, DES has been withdrawn as a standard by the National Institute of Standards and Technology (formerly the National Bureau of Standards).

**DES Algorithm**

- It uses 64 bits plaintext.
- And 56 bits key for encryption and decryption.
- Plaintext is followed by a phase consisting of 16 rounds of same function.
- Each round involves both Permutation and Substitution function.

**IMPLEMENTATION OF ALGORITHM**

xl
Figure 10—DES algorithm model

Algorithm Description
DES is the archetypal block cipher — an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64
bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits, and it is usually quoted as such.

Like other block ciphers, DES by itself is not a secure means of encryption but must instead be used in a mode of operation. FIPS-81 specifies several modes for use with DES. Further comments on the usage of DES are contained in FIPS-74

Overall structure-
The algorithm's overall structure is shown in Figure 1: there are 16 identical stages of processing, termed rounds. There is also an initial and final permutation, termed IP and FP, which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have almost no cryptographic significance, but were apparently included in order to facilitate loading blocks in and out of mid-1970s hardware, as well as to make DES run slower in software.

Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes — the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.

The ⊕ symbol denotes the exclusive-OR (XOR) operation. The F-function scrambles half a block together with some of the key. The output from the F-function is then combined with the other half of the block, and the halves are swapped before the next round. After the final round, the halves are not swapped; this is a feature of the Feistel structure which makes encryption and decryption similar processes.

The Feistel (F) function-
The F-function, depicted in Figure 2, operates on half a block (32 bits) at a time and consists of four stages:
The Feistel function (F-function) of DES

1. **Expansion** — the 32-bit half-block is expanded to 48 bits using the expansion permutation, denoted $E$ in the diagram, by duplicating half of the bits. The output consists of eight 6-bit (8*6=48bits) pieces, each containing a copy of 4 corresponding input bits, plus a copy of the immediately adjacent bit from each of the input pieces to either side.

2. **Key mixing** — the result is combined with a subkey using an XOR operation. Sixteen 48-bit subkeys — one for each round — are derived from the main key using the key schedule (described below).

3. **Substitution** — after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the $S$-boxes, or substitution boxes. Each of the eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table. The S-boxes provide the core of the security of DES — without them, the cipher would be linear, and trivially breakable.

4. **Permutation** — finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the $P$-box. This is designed so that, after expansion, each S-box's output bits are spread across 6 different S boxes in the next round.

The alternation of substitution from the S-boxes, and permutation of bits from the P-box and E-expansion provides so-called "confusion and diffusion" respectively, a concept identified by Claude Shannon in the 1940s as a necessary condition for a secure yet practical cipher.
Figure 13 illustrates the key schedule for encryption — the algorithm which generates the subkeys. Initially, 56 bits of the key are selected from the initial 64 by Permutation Choice 1 (PC-1) — the remaining eight bits are either discarded or used as parity check bits. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one or two bits (specified for each round), and then 48 subkey bits are selected by Permutation Choice 2 (PC-2) — 24 bits from the left half, and 24 from the right. The rotations (denoted by "<<<" in the diagram) mean that a different set of bits is used in each subkey; each bit is used in approximately 14 out of the 16 subkeys.

The key schedule for decryption is similar — the subkeys are in reverse order compared to encryption. Apart from that change, the process is the same as for encryption. The same 28 bits are passed to all rotation boxes.
IMPLEMENTATION

6.1 Overall Structure

Figure 14—overall structure of kerberos
6.2 Execution steps

Message 1:

\[ C \rightarrow AS \quad ID_c \mid ID_{tgs} \mid TS_1 \]

Client request ticket granting ticket.

- \( ID_c \): Tells AS identify of user from the client.
- \( ID_{tgs} \): Tells AS that user request access to TGS
- \( TS_1 \): Allows AS to verify that client clock is synchronized with that of AS

Message 2:

\[ AS \rightarrow C \quad E(K_c[K_{c,tgs} \mid ID_{tgs} \mid TS_2 \mid Lifetime_2 \mid Ticket_{tgs}]) \]

AS returns ticket-granting ticket

\[ Ticket_{tgs} = E(K_{tgs}[K_{c,tgs} \mid ID_c \mid AD_c \mid ID_{tgs} \mid TS_2 \mid Lifetime_2]) \]

Message 1 and 2 compromises for the authentication service exchange to obtain ticket-granting ticket.

- \( K_c \): Encryption is based upon the user’s password, enabling AS and Client to verify password, and protecting contents of message(2).
$K_{c, tgs}$: Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share up a permanent key.

$ID_{tgs}$: Confirms that this ticket is for the TGS.

$TS_2$: Informs client of time this ticket was issued.

$\text{Lifetime}_2$: Informs client of the lifetime of this ticket.

$\text{Ticket}_{tgs}$: Ticket to be used by client to access TGS.

Message 3 and 4 compromises for Ticket generating service exchange to obtain service generating ticket.

**Message 3**: $C \rightarrow TGS \ \ ID_v \ | \ \ Ticket_{tgs} \ | \ \ Authenticator_c$

Client request service generating ticket

$ID_v$: Tells TGS that user request access to server V.

$\text{Ticket}_{tgs}$: Assures TGS that this user has been authenticated by AS.

$\text{Authenticator}_c$: Generated by client to validate ticket.
Message 4: TGS→C  \(E(K_{c,tgs},[K_{c,v} | ID_v | TS_4 | Ticket_v])\)

TGS returns service granting ticket.

\[\text{Ticket}_{tgs}=E(K_{tgs},[K_{c,tgs} | ID_c | AD_c | ID_{tgs} | TS_2 | \text{Lifetime}_2])\]

\[\text{Ticket}_v=E(K_{v},[K_{c,v} | ID_c | AD_c | ID_v | TS_4 | \text{Lifetime}_4])\]

\[\text{Authenticator}_c=E(K_{c,tgs},[ID_c | AD_c | TS_3])\]

\(K_{c,tgs}\) : Key shared only by C and TGS protects contents of message(4).

\(K_{c,v}\) : Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.

\(ID_v\) : Confirms that this ticket is for server V.

\(TS_4\) : Informs client of time this ticket was issued.

\(Ticket_v\) : Ticket to be used by client to access server V.

\(Ticket_{tgs}\) : Reusable so that user does not have to reenter password.

\(K_{tgs}\) : Ticket is encrypted with key known only to AS and TGS, to prevent tampering.

\(K_{c,tgs}\) : Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.

\(ID_c\) : Indicates the rightful owner of this ticket.

\(AD_c\) : Prevents use of ticket from workstation other than one that initially requested the ticket.

\(ID_{tgs}\) : Assures server that it has decrypted ticket properly.

\(TS_2\) : Informs TGS of time this ticket was issued.

\(\text{Lifetime}_2\) : Prevents reply after ticket has expired.

\(\text{Authenticator}_c\) : Assure TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent reply.

\(K_{c,tgs}\) : Authenticator is encrypted with key known only to client and TGS, to prevent tampering.

\(ID_c\) : Must match ID in ticket to authenticate ticket.

\(AD_c\) : Must match address in ticket to authenticate ticket.

\(TS_3\) : Informs TGS of time this authenticator was generated.
Message 5 and 6 for client/server authentication exchange to obtain service.

**Message 5:**

\[ C \rightarrow V \text{ Ticket}_v \mid \text{ Authenticator}_c \]

Client requests service.

Ticket\(_v\) : Assures server that this user has been authenticated by AS.

Authenticator\(_c\) : Generated by Client to validate ticket.

**Message 6:**

\[ V \rightarrow C \ E(K_{c,v},[TS_5+1]) \]( for mutual authentication)

Optional authentication of sever to client.

Ticket\(_v\)=\(E(K_v,[K_{c,v} \mid ID_c \mid AD_c \mid ID_v \mid TS_4 \mid \text{Lifetime}_4])\)

Authenticator\(_c\)=\(E(K_{c,v},[ID_c \mid AD_c \mid TS_5])\)

\(K_{c,v}\) : Assures C that this message is from V.

\(TS_5+1\) : Assures C that this is not a reply of an old reply.

Ticket\(_v\) : Reusable so that the client does not need to request a new ticket from TGS for each access to the same server.

\(K_v\) : Ticket is encrypted with key known only to TGS and server, to prevent tempering.

\(K_{c,v}\) : Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.

\(ID_c\) : Indicates the rightful owner of this ticket.

\(AD_c\) : Prevents use of ticket from workstation other Than one that initially requested the ticket.

\(ID_v\) : Assure server that it has decrypted the ticket properly.

\(TS_4\) : Informs server of time this ticket was issued.

\(\text{Lifetime}_4\) : Prevents reply after ticket has expired.

Authenticator\(_c\) : Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short life time to prevent reply.

\(K_{c,v}\) : Authenticator is encrypted with key known only to
Client and server, to prevent tempering.

**ID_c** : Must match ID in ticket to authenticate ticket.

**AD_c** : Must match address in ticket to authenticate Ticket.

**TS_5** : Informs server of time this authenticator was generated.
CHAPTER 7
RESULT

The results that come up as the Kerberos is implemented are as follow:

Figure 15(a) Snapshot of DOS screen for client source code.

Figure 15(b) Client output
Figure 16(a) - Snapshot of DOS screen for Kerberos source code.

Figure 16(b) — Kerberos output
Figure 17(a) - Snapshot of Dos screen for server source code

Figure 17(b) — server output
Client sending user name to Kerberos.

Password generated for user and TGT generated for client
Encrypted TGT for user

Encrypted and decrypted ticket at client.
### Kerberos

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Life Stamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Stamp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Password, life stamp and time stamp at Kerberos.

lix
Session key, encrypted ticket sent through Kerberos

Server authorized and data sent through the client terminal.
Encrypted packet, session key and authenticator created at client.

Authenticator created at server.
Data received at the server through the client.

Data received at client from server
CLIENT

USERNAME

1. Username is sent to Kerberos

Encrypted Ticket - Granting Ticket (GT)

2. Ticket-Granting Ticket is received from Kerberos encrypted with Client's TGT

3. Ticket-Granting Ticket is decrypted

Decrypted Ticket - Granting Ticket (GT)

4. Packet containing Service_name and TGT is sent to kerberos

Service Name

5. Packet containing Session_key and Service Ticket received from Kerberos

EOMEE

6. Packet is decrypted to obtain the Session Key and Encrypted - Service Ticket

Encrypted Packet

7. Authenticator containing Client name and Workstation add is created, Encrypted with Session Key

8. Packet containing Authenticator and Service Ticket is created

Session Key

Authenticator Created

9. Packet is sent to Server

Packet Created

10. Authentication from Server is received

Send Packet to Server

11. Checking for Server's Authenticity

Server Authenticator

12. Server's Authenticity is proved

Authorized Server

13. Data sent to Server

Data Send

14. Data Received from Server

Data Received

15. Data is proceeded by the Client

Data Processed

Exit
CHAPTER 8
PERFORMANCE EVALUATION

The Competition: SSL

<table>
<thead>
<tr>
<th>SSL</th>
<th>Kerberos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses public key encryption</td>
<td>Uses private key encryption</td>
</tr>
<tr>
<td>Is certificate based (asynchronous)</td>
<td>Relies on a trusted third party (synchronous)</td>
</tr>
<tr>
<td>Ideal for the WWW</td>
<td>Ideal for networked environments</td>
</tr>
<tr>
<td>Key revocation requires Revocation Server to keep track of bad certificates</td>
<td>Key revocation can be accomplished by disabling a user at the Authentication Server</td>
</tr>
<tr>
<td>Certificates sit on a users hard drive (even if they are encrypted) where they are subject to being cracked.</td>
<td>Passwords reside in users' minds where they are usually not subject to secret attack.</td>
</tr>
<tr>
<td>Uses patented material, so the service is not free. Netscape has a profit motive in wide acceptance of</td>
<td>Kerberos has always been open source and freely available.</td>
</tr>
</tbody>
</table>

Firewall vs. Kerberos?

- Firewalls make a risky assumption: that attackers are coming from the outside. In reality, attacks frequently come from within.
- Kerberos assumes that network connections (rather than servers and work stations) are the weak link in network security.
**Difference between Version 4 and Version 5**

Version 5 is integrated to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Kerberos Version 4 was developed for use within the project Athena environment and accordingly, did not fully address the need to be of general purpose. This led to the following environmental shortcomings:

1] Encryption system dependence.
Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5 cipher text is tagged with an encryption type identifier so that any encryption may be used. In version 5 the same key to be used in different algorithm.

2] Internet protocol dependence
Version 4 requires the use of internet protocol address other address types such as the ISO network address are not accommodated. Version 5 network address are tagged with type and length allowing any network address type to be used.

3] Ticket Lifetime
Lifetime values in version 4 are encoded and in an 8 bit quantity in units of 5 minutes thus the maximum life time that can be expressed is $2^8 \times 5 = 1280$ minutes or a little over 21 hours. In version 5 tickets include an explicit start timer and end time allowing tickets with arbitrary life time.
CHAPTER 9
9.1 Conclusion

Kerberos 5 is a very widely used authentication protocol that across Windows, Unix, Mac OS series operating systems. The basic architecture and algorithm among those systems is almost the same, yet, there are several differences according to the specific system and the system company. Another feature is, in the operating systems, network authentication and authorization depends on a lot of protocols and services, which can guarantee the security from all aspects of using the system. So the authentication alone, say Kerberos is just a significant part of the system. Also, public key cryptography can be integrated in Kerberos 5 authentication protocol to eliminate the need of establishing and storing a large number of shared symmetric keys between KDCs, servers, and users. Thus, it can prevent the potential risk of a KDC compromise, in which all the symmetric keys will be exposed to the attacker and need to be reestablished. There have been many proposed extension to Kerberos that supports public cryptography, including PKDA, PKINIT, PKCROSS, PKTAPP, and etc.

9.2 Future Work-

As a mature product, Kerberos has undergone significant revision. The last major revision, Kerberos Version 5, addressed many previously identified limitations:

- Version 5 added support for forwardable, renewable, and post datable tickets. These accommodate long running processes and processes which need to run automatically in the future, in addition to allowing users to use their credentials on a machine other than the one they logged in on.
- Kerberos tickets can now contain multiple IP addresses and addresses for different types of networking protocols. This allows the use of multi-homed machines.
- Replay caches keep track of recently issued tickets and do not allow the same ticket to be used twice in a row. This cuts down on the ability of attackers to hijack cached tickets before they expire.
- There is now support for transitive cross-realm authentication which removes the requirement that each pair of realms that wish to allow authentication must share a secret. In large networks consisting of many realms, the number of secrets can become quite large and is not scalable. Instead, transitive cross-realm authentication allows a path between
secret-sharing realms to be specified so that credentials from the desired realm can be earned by following this path.

Modifications to Kerberos and research on extending Kerberos are still active. For instance, as noted previously, recent work has explored using public-key cryptography for cross-realm authentication, which has the potential to improve the scalability of Kerberos.

Kerberos' many strengths are attested to by its wide adoption as an industry standard. It seems certain to continue to play a role in small networks with strict authentication requirements. In the broader Internet context, however, SSL seems already to have achieved de facto standard status. Even with technical improvements in its scalability, Kerberos may remain restricted to the small network context.
REFERENCES


