BACHELOR THESIS
CAPABILITY OF KERBEROS

MATTHIJS MEKKING

JUNE 2006

Contents

Chapter 1

Introduction

Kerberos is a well-known authentication system created at MIT, that has been adapted in many software applications. However, it was not the intention to build a system that reached world wide use in the first place. Still, they have made a good design and the system combines the right security aspects with nice functionality. Since the beginning of Kerberos, in 1988, there have been many publications about the subject. Those describe the general ideas and the operation of the system. Also, many publications report to have found bugs in the system. Over the years, Kerberos has evolved to become more secure and more accessible, by providing more functionality. But few reports have given an exact overview of what Kerberos is actually capable of. In this study, I shall try to create a list of all the services that Kerberos provides and look at which security measures it takes.

The services can be provided thanks to elements in the Kerberos procotol. For example, the element 'encryption key' will contribute to the service 'confidentiality' and 'integrity' of messages sent over the network. How are the services and security aspects provided? Which elements contribute to which services? For all Kerberos elements, I shall try to find out which service they contribute to.

1.1 Outline

First of all, in chapter 2, a comprehensive introduction to the Kerberos system is given. I shall discuss its origin, how the protocol works, known flaws and bugs and its evolution to the latest Kerberos version.

In chapter 3, a list of services that Kerberos provides is being described. Also, we shall look at the security vulnerabilities it encounters . The protocol will be described again in this chapter, but at a more detailed level. I shall relate every element of the protocol to a service or security measure.

Chapter 4 deals with the vulnerabilities that still exists in the latest version of Kerberos. It will be followed by a small overview of this thesis.

Chapter 2

The Kerberos Protocol

The question I shall try to answer is: 'What services does Kerberos provide?'. However, Kerberos is a rather complex system, and to identify all services out of the blue is quite difficult. That is why I shall first devote a chapter to describe how Kerberos has evolved. We shall mainly look at the protocol, but also some social and implementation aspects will be discussed. So, the focus of this paper lays on protocol level. If implementation features or drawbacks are mentioned, then these refer to the MIT implementation. The most common used versions are Kerberos 4 and Kerberos 5 [Sta03]. Kerberos 5 was developed to improve Kerberos 4, which still had some security flaws. Also different implementations of Kerberos were developed, for example Heimdal, that could communicate with the MIT Kerberos 5 version. The Kerberos 4 flaws and improvements will be pointed out in the section 2.3. Section 2.4 will describe the new elements of Kerberos 5 with respect to version 4.

2.1 Term definitions

In this paper, there are some terms that can be interpreted ambiguously. Therefore, we shall define these terms first:

User and client
Kerberos deals with users that request services. There are two terms that are used for users: 'users' and 'clients'. In general, when this paper states that a client makes contact to a server to access a particular service, it is not the physical person who makes the contact, but a program running on that person's computer. In this paper, the term user refers to the actual person.

Application servers and services
Typically, a user will request a service, for example list his e-mails. This is a service that is handled by an application that runs on a certain server. This server will be called the 'application server'. So when a user is requesting a service, the client is communicating with the application server.

Principals
Kerberos uses the term 'principals' to identify users and application servers. A

principal is a uniquely named client or application server that participates in the Kerberos network communication.

Tickets
Tickets are used to give users access to services. There are two different kind of tickets. One can be used to request services and tells the Kerberos server that the use has been authenticated. This one is called the ticket-granting ticket. The other can be used to access a certain service, and is called the application ticket or the service ticket. Many properties apply to both tickets. Only in this case, the study will refer to both tickets with the term 'tickets'. In all other situations the study will state which type of ticket is being discussed. However, in the protocols the term 'TGT' is used for the ticket-granting ticket, and 'Ticket' is used for the service ticket.

C = client
AS = authentication server
AP = application server
TGS = ticket-granting server
TGT = ticket-granting ticket
$Ticket$ = ticket to prove identity of the user to the application server
$Auth_{x,y}$ = authenticator of x to y
ID_x = identifier of x
$Realm_x$ = The environment in which x is located
P_x = password of x
NA_x = network address of workstation of x
K_x = secret key shared by AS and x
$K_{y,x}$ = secret key shared by y and x
SK = session key that can be used for one connection only
TS_x = Timestamp, where x identifies the timestamp
LT_x = Lifetime, where x identifies the lifetime
$Nonce_x$ = A random value to be repeated, to assure the response is fresh, with x identifying the nonce
Seq = A sequence number

Figure 2.1: Protocol denotations

In the next section there will also be some protocol denotations. They are defined in figure 2.1. You can choose to look at them now, or while reading this thesis, when an abbreviation is not clear.

2.2 Kerberos 4

2.2.1 Motivation

The development of Kerberos started at MIT as part of the Athena project [Koh91]. There was need for an authentication system that is secure in an open environment, with network connections to other machines. In this environment, relying on the host operating system, host addresses and physical security is not sufficient. This is because in such an environment, opponents can easily tamper with data sent over

the network. To solve these problems, their own authentication system needed to be developed. It evolved from version 1 to version 3 for internal use during this project. Version 4 was also primarily designed for use by Project Athena, but has achieved widespread use. The Kerberos model is based in part on Needham and Schroeders trusted third-party authentication protocol [Nee78] and on modifications suggested by Denning and Sacco [Den81]. The idea of Kerberos is that services can verify the identities of users in an open, unprotected network and users can also verify the identity of services.

The first report on Kerberos listed the following requirements for Kerberos, that are essential to succeed the project [Sta03]:

- **Secure**
  For Kerberos, this means that a network eavesdropper, who is unauthorized to access the Kerberized services, should not be able to retrieve information, which can be used to retrieve access or information about the services. This means that no information may be obtained by the opponent that can be used to impersonate a user, or that service requests and responses can't be read. We shall see that this requirement is met by trust on tickets. Tickets are given to a user, so that he is able to prove he is authentic.
- **Reliable**
  Kerberos should be available all the time. If the Kerberos server is not available, then all Kerberized services are also not available. This requirement is met by installing several backup servers, and is not really a protocol issue.
- **Transparent**
  The user should not be aware of the authentication process taking place (with the exception of entering his password). The protocol gives the impression that the user should do many actions, but these actions are being made by a program that runs on the client's workstation. Also this requirement is not a protocol issue.
- **Scalable**
  Kerberos should be capable of supporting a large number of clients and services. Since Kerberos 4 has been widely used, this requirement can be considered met. However, some functionality (for example key exchanges) lead to heavy computation when the number of services and clients grew. With Kerberos 5, some of these efficiency problems have been solved.

The intention is to use Kerberos in a small, well managed environment. With the development of Kerberos 5 and further, some suggestions for adaptation were made, in order to make Kerberos usable for large public environments. However, the protocols are currently not being considerated an Internet standard [Koh93]. To authenticate users, Kerberos makes use of symmetric keys. But the protocol itself has no suggestion to exchange these keys, so key exchange becomes an implementation or organizational decision.

2.2.2 Protocol description

To understand the protocol, we follow the strategy by Bill Bryant of Project Athena [Bry88] by building up the protocol step by step. First, we shall look at a rather simple authentication dialogue (figure 2.2). In this dialogue the risk of impersonation is considered. An attacker can pretend to be another user and obtain unauthorized

(1) $C \rightarrow AS : ID_C + P_C + ID_{AP}$
(2) $AS \rightarrow C : Ticket$
(3) $C \rightarrow AP : ID_C + Ticket$

with

$Ticket = E_{K_{AP}}[ID_C + NA_C + ID_{AP}]$

Threats/problems prevented:
- Authenticate users: APs can check if a user is authentic
- Key management: AS lowers the number of needed keys and passwords

New or remaining threats/problems:
- IP spoofing: The ticket can be stolen, the opponent can adapt the identity of C and alter the IP address to the network address of C
- Usability of tickets: For every new service, a new ticket has to be issued
- Password sniffing: password is sent unencrypted over the network

Figure 2.2: Dialogue 1, a rather simple dialogue

services. This should not be considered as to be the weakest link. One way to counter this threat, is that each server can be adapted in such a way that they can confirm the identity of users. This is however a rather clumsy solution, because then each server needs to store a password file. Imagine that someone needs to change his password, then he has to change it for every server independently. So let's try a different approach. In the first dialogue the authentication server (AS) is introduced. This server knows all passwords of all principals and stores them in a centralized database, and shares a unique secret key with each application server (AP). In the highlighted boxes, I sum the threats and problems that have been taken care of, and also sum new problems that are being introduced by the protocols.

The first dialogue we shall look at starts with the client (C) requesting access to an AP. He authenticates himself to the AS with his identity, his password and the identity of the AP. The AS checks the credentials and whether the user has permission to access the AP. If so, the user gets an application ticket that is encrypted with the secret key shared by the AS and the AP. This prevents the user or an attacker to modify the ticket, and the AP knows that the ticket could only have been made by the AS. To prove that the user is authentic, the application ticket consists of three elements:

- ID_C
  The identity of C, in order to indicate that the ticket is indeed issued on behalf of C.
- ID_{AP}
  The identity of the AP, to make sure that the AP has decrypted the ticket correctly.
- NA_C
  The network address of C. NA_C is included to prevent someone from stealing the ticket, copies the identity of C and present himself to the AP. Without the network address, the attacker would have a valid ticket. However, with

the threat of IP spoofing, this threat still remains.

There are two particular problems that we encounter in dialogue 1. The first is the usability of the ticket. For every different application server, the user needs to ask for a new ticket. This means that the user needs to enter his password again. We would like to minimize this. The second problem is that the password is sent unencrypted over the network in message (1). A new dialogue (figure 2.3) is suggested, that overcomes these two problems.

Once per user logon session
(1) $C \rightarrow AS : ID_C + ID_{TGS}$
(2) $AS \rightarrow C : E_{K_C}[TGT]$
Once per type of service
(3) $C \rightarrow TGS : ID_C + ID_{AP} + TGT$
(4) $TGS \rightarrow C : Ticket$
Once per service session
(5) $C \rightarrow AP : ID_C + Ticket$

with

$TGT = E_{K_{TGS}}[ID_C + NA_C + ID_{TGS} + TS_1 + LT_1]$
$Ticket = E_{K_{AP}}[ID_C + NA_C + ID_{AP} + TS_2 + LT_2]$

Threats/problems prevented:
- Usability of tickets: Password has to be entered only once a working session
- Password sniffing: Password is no longer sent over the network

New or remaining threats/problems:
- Replay attacks: An opponent could steal the ticket and use it to access services unauthorized
- IP spoofing: The TGT or service ticket could still be captured and the IP address could be forged to the network address of C, with a man-in-the-middle attack

Figure 2.3: Dialogue 2, a much more secure dialogue

In the new dialogue, a ticket-granting server (TGS) is introduced. This server issues application tickets to clients who have been authenticated to the AS. In this case, the user logs in, requesting a ticket-granting ticket (TGT). He receives the TGT encrypted with a key derived from the client's password. The user can only generate the key to decrypt the ticket, if he knows the right password, which is known to the AS and himself only. This once per user logon session prevents sending the password over the network.

With the TGT, we can minimize the number of password requests. Because the TGT verifies that the user is authenticated, he does not need to give his password anymore, and the TGS can provide him with a ticket for the AP. The user does this by sending his identity, the identity of the AP and his TGT. The TGS checks the TGT, and if it is valid, C will receive a ticket for the AP. Now the user can request a service (5) the same way he did in figure 2.2.

The tickets are extended with a timestamp and a lifetime to prevent an attacker from

stealing one ticket and reuse it after the legitimate user has left the workstation. Tickets would have a typical lifetime of the length of a working day (e.g., eight hours).

The second dialogue is already much more secure, but still problems remain. First of all, although the tickets are provided with a timestamp and lifetime, replay attacks remain a threat. Someone could steal the TGT and forge the network address and he will receive application tickets unauthorized. Also, he could capture the application ticket and uses it before expiration, giving him access to the corresponding service. We could reduce this risk by shortening the lifetime. However, a short lifetime will introduce the problem of many password requests. The user will need to enter his password again, if the lifetime has exceeded. The second problem is something we mentioned in the beginning: with both dialogues, the AP still cannot prove its identity to the user. Kerberos 4 takes care of these problems.

For the Kerberos protocol, it is necessary to have server clocks and client clocks that register the same time. The Network Time Protocol (NTP) [Mil89] can be used to synchronize the clocks.

Authentication Service
(1) $C \rightarrow AS : ID_C + ID_{TGS} + TS_1$
(2) $AS \rightarrow C : E_{K_C}[K_{C,TGS} + ID_{TGS} + TS_2 + LT_2 + TGT]$
Ticket-granting Service
(3) $C \rightarrow TGS : ID_{AP} + TGT + Auth_{C,TGS}$
(4) $TGS \rightarrow C : E_{K_{C,TGS}}[K_{C,AP} + ID_{AP} + TS_4 + Ticket]$
Client/Server Service
(5) $C \rightarrow AP : Ticket + Auth_{C,AP}$
(6) $AP \rightarrow C : E_{K_{C,AP}}[TS_5 + 1]$

with

$TGT = E_{K_{TGS}}[K_{C,TGS} + ID_C + NA_C + ID_{TGS} + TS_2 + LT_2]$
$Ticket = E_{K_{AP}}[K_{C,AP} + ID_C + NA_C + ID_{AP} + TS_4 + LT_4]$
$Auth_{C,TGS} = E_{K_{C,TGS}}[ID_C + NA_C + TS_3]$
$Auth_{C,AP} = E_{K_{C,AP}}[ID_C + NA_C + TS_5]$

Threats/problems prevented:
- Replay attacks: An opponent can no longer get access with a stolen ticket
- Prove identity of APs: Users can prove the identity of APs
- IP spoofing: The authenticators prevent attackers to spoof meaningful transmissions

New threats/problems:
- This protocol still has some vulnerabilities, which are being discussed in section 2.3

Figure 2.4: Dialogue 3, the Kerberos 4 protocol

Let us look at the protocol adjustments, message by message. The protocol is shown in figure 2.4. First, the client needs to authenticate himself to the AS in the authentication service exchange part. This way, the client can obtain a TGT. In message

(1), the client requests access to the TGS, the same way as in figure 2.3. One extra element is added, TS_1, to verify that the client's clock is synchronized with the clock of the AS. Message (2) introduces also some new elements. The first is the session key $K_{C,TGS}$, that permits secure exchange between the TGS and the client. The other elements, ID_{TGS}, TS_2 and LT_2, are used to provide the client with information about the TGT, such as the identity of the TGS, when the ticket was issued and when it expires.

Now that the client has a TGT and a session key, he can obtain an application ticket in the ticket-granting service exchange part. In message (3), the client requests for a service, almost the same way as in figure 2.3. But instead of sending his identity, he sends a so-called authenticator. The authenticator prevents replay attacks to the TGS. It is encrypted with the session key of C and the TGS, so that only these two parties can read it. It contains the identity of C, the network address of C and a timestamp, to inform when the authenticator was created. The TGS sends a ticket to the client in message (4), which has been constructed similarly as the TGT. Again, the client is provided with a session key (this time for the client and the AP), and information about the application ticket (the identity of AP, timestamp when the ticket was issued and expiration time).

That brings us to the final part of the protocol, the client/server authentication exchange. In message (5), the client requests for a service, comparable with message (5) in dialogue 2, but again the identity of C is replaced by an authenticator. This time the authenticator is encrypted with the session key of C and the AP. Message (6) is added to permit mutual authentication.

The replay attack problem is solved by using a session key $E_{K_{C,TGS}}$, a secret piece of information shared by the client and the TGS, that is provided by the AS in message (2). The session key can only be read by the AS and the client, because it is encrypted with E_{K_C}, the secret key between C and AS. With this session key the client can prove its identity to the TGS, by encrypting an authenticator with the session key, proving that he is the only principal who could have made it. The authenticator contains information that verifies the identity of the client. The TGS is able to decrypt the authenticator, because the session key is also provided in the ticket. The second problem, that AP can't prove its identity to the user, is solved by message (6). The AP sends back the timestamp of the authenticator, incremented by 1, encrypted with the session key $E_{K_{C,AP}}$. The encryption assures that the incremented timestamp could only have been created by the application server.

2.2.3 Realms

Now we have described the Kerberos 4 version, we can identify its components:

- **Clients**
  Entities on the network that can get tickets from Kerberos, in order to use Kerberized programs on application servers.
- **Kerberos server**
  The Kerberos server consists of an AS, which checks the service access permissions for clients and issues the TGTs, and a TGS, which issues the tickets for the desired services.

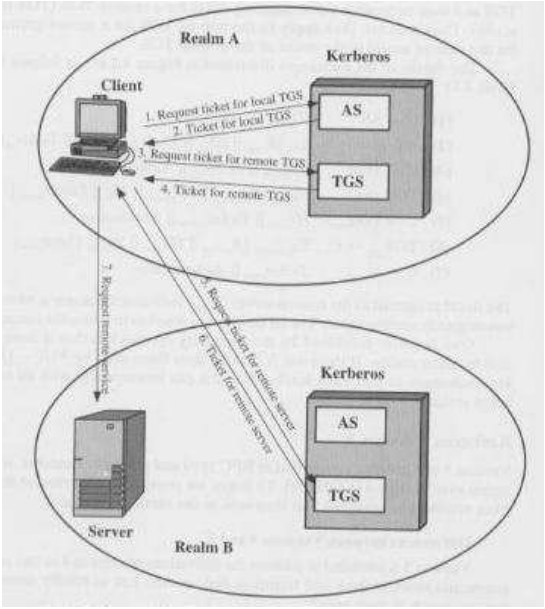

Figure 2.5: Interaction between the components of Kerberos

- **Application servers**
  The servers where the Kerberized applications run.

These three components form a complete Kerberos environment, called a *realm*. Usually an organization operates within one realm. But these days, cross-organizational operations are quite common. That means that a user in organization A can have access to certain permissions in organization B. But with the Kerberos protocol discussed so far, this user cannot authenticate himself in a different realm, and thus cannot access the services in that realm. Kerberos provides support for this cross-realm authentication, by sharing a secret key between the two servers in the inter-operating realm. Therefore, the two Kerberos servers must trust each other to authenticate users, and the application servers must trust the Kerberos server in the other realm. This is visually described in 2.5. If we deal with inter-operating realms, some new messages must be introduced to the protocol. Instead of requesting for an application ticket, the client asks the TGS for a remote TGT, that can be used in the other realm. With this remote TGT, the client can request a remote application ticket. The changed protocol is shown in figure 2.6, with R referring to the cooperating realm.

2.3 Limitations of Kerberos 4

Kerberos 4 was originally developed for use by Project Athena only. However, it became a popular authentication system that reached world wide use. A first prototype version of Kerberos 4 was released in September of 1986. But limitations and problems were identified in the following three years, which leaded to the development of a new Kerberos version. Work on Kerberos 5 started in 1989. This version

Authentication Service
(1) $C \rightarrow AS : ID_C + ID_{TGS} + TS_1$
(2) $AS \rightarrow C : E_{K_C}[K_{C,TGS} + ID_{TGS} + TS_2 + LT_2 + TGT]$
Remote Access Service
(3) $C \rightarrow TGS : ID_{TGS_R} + TGT_R + Auth_{C,TGS}$
(4) $TGS \rightarrow C : E_{K_{C,TGS}}[K_{C,TGT_R} + ID_{TGT_R} + TS_4 + TGT_R]$
Ticket-granting Service
(3) $C \rightarrow TGS_R : ID_{AP_R} + TGT_R + Auth_{C,TGS_R}$
(4) $TGS_R \rightarrow C : E_{K_{C,TGS_R}}[K_{C,AP_R} + ID_{AP_R} + TS_6 + Ticket_R]$
Client/Server Service
(5) $C \rightarrow AP : Ticket + Auth_{C,AP_R}$
(6) $AP \rightarrow C : E_{K_{C,AP}}[TS_6 + 1]$

with

$TGT = E_{K_{TGS}}[K_{C,TGS} + ID_C + NA_C + ID_{TGS} + TS_2 + LT_2]$
$Ticket = E_{K_{AP}}[K_{C,AP} + ID_C + NA_C + ID_{AP} + TS_4 + LT_4]$
$Auth_{C,TGS} = E_{K_{C,TGS}}[ID_C + NA_C + TS_3]$
$Auth_{C,TGS_R} = E_{K_{C,TGS_R}}[ID_C + NA_C + TS_5]$
$Auth_{C,AP_R} = E_{K_{C,AP_R}}[ID_C + NA_C + TS_6]$

Introduced:
- Cross-realm authentication

Figure 2.6: Dialogue 4, the Kerberos 4 protocol with cross-realm authentication

has been adapted in many systems as well, for example in the Windows operating system (since Windows NT).

The limitations of Kerberos 4 can be addressed in two areas: environmental shortcomings and technical deficiencies [Koh91]. Bear in mind, that there are differences between the security of the protocol itself and its implementation. In the next two subsections, security weaknesses in both areas are addressed. However, in the section about Kerberos 5, the focus will be back at protocol level.

2.3.1 Environmental shortcomings

Kerberos 4 was developed for use by Project Athena, and did not fully address the need to be of general purpose. This caused some environmental shortcomings:

- **Encryption system dependence**
  Kerberos 4 uses only DES to encrypt messages. First of all, the export of DES from the USA was restricted until the year 2000. Second, the strength of the DES algorithm raises concerns as the DES computations can now be resolved in reasonable time. To avoid these problems in the future, Kerberos 5 tags the cipher text with an encryption type identifier. Any type of encryption can be used, or replaced when needed. This way, providing sufficient integrity protection is the responsibility of the encryption technique.
- **Internet protocol dependence**
  Kerberos 4 requires the use of IP addresses, but nowadays different address

types are used, such as the ISO network address. Kerberos 5 solves this the same way as with the encryption dependence problem: network addresses are tagged with type and length.

- **Message byte ordering**
  In Kerberos 4, the sender of a message chooses its own byte ordering and tags the message to indicate if the least or most significant byte is in the lowest address. The receiver must then convert this byte order to its own native order. This simplifies the communication between two hosts with the same byte order. However, the protocol does not follow established conventions. In Kerberos 5, all message structures are defined using Abstract Syntact Notation One (ASN.1) and Basic Encoding Rules (BER). With these standards, byte ordering is no longer ambiguous.
- **Ticket lifetime**
  The lifetime of a ticket in Kerberos 4 could not exceed $21\frac{1}{3}$ hours, which may not be enough in some environments. The lifetime was encoded by a UNIX timestamp for the issue date and an 8-bit quantity in units of five minutes for the lifespan ($\frac{2^8 * 5}{60} = 21\frac{1}{3}$). In Kerberos 5, tickets contain an explicit start and end time, so tickets can have lifetimes without restriction on time.
- **Authentication forwarding**
  Kerberos 4 is not allowing credentials issued to a client for a certain host to be forwarded to a different host and used by another client. But this functionality could be useful if an intermediate principal needs access to a particular service with the rights of the client. Kerberos 5 supports this functionality.
- **Principal naming**
  In Kerberos 4, principals have three components: name, instance, and realm. They each can have a maximum length of 39 characters, which seems to be insufficient in some environments. Also, because of implementation reasons, the period (.), which sometimes is used in account names, was excluded from the character set. Also, Kerberos 4 assumed that the account name match the name portion of the principal identifier, which is not acceptable in environments that accept non-unique account names. These problems are solved in Kerberos 5.
- **Cross-realm authentication**
  As we have seen, Kerberos 4 provides cross-realm authentication by letting the inter-operating Kerberos servers sharing a secret key. The key exchange simplifies the implementation of inter-realm ticket requests and verification, but the number of key exchanges grows exponentially ($O(n^2)$). In Kerberos 5, support for cross-realm authentication is implemented, which lowers the number of key exchanges to $O(log(n))$.

2.3.2 Technical deficiencies

Next to the environmental shortcomings, some technical limitations of the Kerberos 4 protocol have been identified:

- **Double encryption**
  Within message (2) and (4) in figure 2.4, tickets that are provided to the client are encrypted twice: once with the key known by the target server, and once with the key known to the client. Because the ticket is already encrypted once, the integrity of the ticket can't be breached. The second encryption does not

provide extra security, and thus is unnecessary. [Mer90]

- **PCBC encryption**
  Kerberos 4 makes use of plain- and cipher-block-chaining (PCBC) DES. This mode tries to provide data encryption and integrity in one operation. But it has been demonstrated that an opponent can modify a message without the recipient being able to detect it [Koh89]. Kerberos 5 uses explicit integrity mechanisms, together with the standard cipher-block-chaining (CBC) encryption.
- **Authenticators and replay detection**
  Timestamps in Kerberos 4 are used to verify the freshness of messages, in order to prevent replay attacks. A version 4 authenticator has a short lifetime, to counter replay attacks and may only be used once. However, a list of unexpired authenticators which have already been used is not maintained in Kerberos 4, so that replay attacks remain a threat. This is an implementation deficiency, rather than a limitation of the protocol. The threat is rather limited, because of the short lifetime of the authenticators.
- **Password attacks**
  Passwords cannot be sniffed in Kerberos 4, but anyone can make an authentication request. Message (1) can be copied and replayed to the AS. The opponent gets a message back, encrypted with a key based on the password of the client. Now, he can try infinitly many passwords to decrypt it. When the message is decrypted correctly, the content will make sense. Nobody is logging the number of tries to decrypt the message. In Kerberos 5, a mechanism called pre-authentication is provided which makes this attack more difficult, but does not prevent it. This mechanism will be described in chapter 3.
- **Session keys**
  Tickets include a session key that enables the client to encrypt the authenticator. The session key can also be used to protect the messages during a session. Since clients may use a ticket multiple times during a user's session, it is possible for an attacker to replay messages from a previous connection to clients or servers (if they do not protect themselves properly). With Kerberos 5 it is possible for clients and servers to negotiate a subsession key, which is used for one connection only. When a client tries to access the server again, a new subsession key is needed.
- **Cryptographic checksum**
  If the basic encryption algorithm itself does not provide for integrity protection (like DES in PCBC mode), then some form of verifiable MAC or checksum must be included. The cryptographic checksum used in Kerberos 4 is based on the quadratic algorithm, is not performed as described by Jueneman, Matyas and Meyer [Jue85]. The Kerberos version 4 checksum is not proven suitable, and that can be considered as an implementation deficiency. This is corrected in Kerberos 5.

2.4 Kerberos 5

Kerberos 5 has been developed to address the limitations and weaknesses of version 4. This has resulted in changes in the protocol and in the tickets. First, the ticket changes will be discussed. After that, the protocol will be described.

2.4.1 Ticket flags

The major change in tickets is that it contains a set of flags, to indicate certain attributes of that ticket.

INITIAL
The INITIAL flag indicates that a ticket was issued using the AS request and not issued by the TGS. It represents a TGT obtained from the AS. A server may require that this flag is set, for example with the password changing protocol (see section 2.4.3) wants to know if the password was recently used.

HW-AUTHENT
The HW-AUTHENT flag indicates if a hardware device was used to alter the encryption key. For example, a smart card that transforms the user's password, so that actually arbitrary passwords are used to alter the key shared by the client and the AS. This prevents an attack on easy passwords.

PRE-AUTHENT
If the pre-authentication flag is set, the AS will require the client to be authenticated before issuing a ticket. This should make password guessing attacks more complicated. The exact form is left unspecified.

INVALID
The INVALID flag indicates that a ticket is invalid. When this flag is set, servers must reject access to the clients. Tickets with the INVALID flag set, can be validated by Kerberos. The client must send a request with the option VALIDATE (message (3), figure 2.7). Only if the start time of the ticket has passed, the ticket will be validated. This check assures that tickets that have been stolen before their start time will be marked invalid permanently through a hotlist mechanism.

RENEWABLE
Some applications desire to hold tickets which can be valid for quite a long period. But this raises a problem of potential credential theft, and the stolen ticket can be used until this long lifetime ticket expires. With the RENEWABLE flag set, the normal "short lived" ticket can be replaced by a new one, without the client having to store the password (remember that the authentication was already taken care of in message (1) and (2) of figure 2.4). Therefore, renewable tickets need to hold two expiration times. The client can renew his ticket by presenting the ticket to Kerberos until the final expiration time has been reached, together with the option RENEW.

POSTDATED
Sometimes client applications need to obtain tickets that they can use later on. For example, a batch submission system would like to have a valid ticket at the time a batch job is serviced. However, holding a valid ticket for a longer time is more prone to theft. Postdated tickets support such functionality, without the extra risk of theft. Postdated tickets have also the INVALID flag set. When the batch job is serviced, the postdated ticket is activated and validated by Kerberos. With this approach, the client does not have to repeatedly use its TGT to obtain an application ticket.

MAY-POSTDATE
A client with a ticket with the MAY-POSTDATE flag set, can request a ticket that

is POSTDATED and INVALID.

PROXIABLE
Sometimes a client wants to allow a service to perform an action on its behalf. This means that the services must be able to take the identity of the client, but only for a particular purpose. This is possible if the client grants the service a proxy. This is allowed, if the PROXIABLE flag is set. This option is only interpreted by the TGS, and this functionality can thus only be used for application tickets. With this ticket, the service can request a ticket with a different network address.

PROXY
When a service requests an application ticket on behalf of the client, this ticket will have the PROXY flag set.

FORWARDABLE
The forwardable concept is an unlimited version of the proxy case. With this flag set, also TGTs can be issued with different network addresses.

FORWARDED
The FORWARDED flag is set if a principal presented a FORWARDABLE ticket.

2.4.2 Protocol description

In figure 2.7 we see the (simplified) protocol for Kerberos 5. However, not all elements are shown in this figure. The figure would become too complex. The new elements and the most important elements not shown will be discussed. The detailed information is provided by RFC 1510, The Kerberos Network Authentication Service (V5) [Koh93]. However, the protocol is still based on the same ideas which where used to design Kerberos 4.

These are the new elements shown in figure 2.7.

- **Options**
  The option field can be used to request tickets where certain flags are set. Generally, the name of the option can be the same as that of the ticket flag, so if you need a FORWARDABLE ticket, you set the FORWARDABLE option. Some other options are needed to provide information about when to use the pre-authentication and authorization data.
- **Client name and realm**
  Just as in Kerberos 4, the identifier of the client is included. Extra information about the realm of the client is added, to support inter-realm authentication.
- **Service name**
  Also like in Kerberos 4, the identifier of the service is included. In the AS request, the service name is that of the TGS.
- **Times**
  Instead of a timestamp, more information about the request time can be stored, such as a skew in which the request is valid, and a field to be able to request a 'renew until' time.
- **Nonce**
  A nonce is added to indicate that the request is fresh.

The following elements are not shown in figure 2.7, but are part of the Kerberos 5

protocol:

- **Protocol version number**
  This field speficies the protocol version number. In this case it will indicate version 5.
- **Message type**
  The structures of the AS request and the TGS request have a lot in common. That's why the structure is being reused. The message type indicates if the message is involved in the AS request or the TGS request.
- **Pre-authentication data**
  If the client requests a ticket with pre-authentication, this field is used to store the necessary information to execute pre-authentication. It can be used against password guessing attacks.
- **Encryption type**
  This field is added to indicate which encryption has been used.
- **Addresses**
  If a proxy ticket is requested, this fields can hold the addresses that are allowed to use the ticket.

The client receives a TGT in message (2). The structure of the AS reply again looks like that of the TGS reply. Also this time, the structure contains an element *message type* to indicate if the reply came from the AS or from the TGS. This message also has a protocol version number, room for pre-authentication data and the realm and identity of the client. Furthermore, it consists of the TGT and an encrypted part to detect replay attacks. The ticket has been extracted from the encrypted part, to overcome the problem of double encryption. A part of the ticket is revealed to simplify inter-realm authentication. The version number, the realm for who the ticket was issued for and the identity of the client are shown in plain text. The rest of the ticket is encrypted and contains the flags, the session key, the realm, network address and identity of the client, some information about time, authorization data and a field that lists the Kerberos realms that took part in authenticating the user. The encrypted part (not part of the TGT) also contains the session key. Furthermore it holds the last request by the user, a response nonce, the key expiration time and some elements that are also in the encrypted part of the ticket. This information can be helpful to the user to verify that the element matches the intended request and to assist in proper ticket caching. These elements can also be helpful to detect credential theft.

Now let us look at the changes in the ticket-granting service exchange. The TGS request in message (3) has the same elements as the AS request. Only the identifier and the realm of the client are left out here. They are only needed at the authentication request. One extra field is used, called Authorization data. This encrypted field is added if services have to be accessed by other principals on behalf of the owner of the ticket. The actual TGT and the authenticator are placed in the pre-authentication data field. Furthermore, a field 'Additional tickets', which is left empty in the AS request, is used. In this field, additional tickets can be stored to configure the dialogue. For example, a session key stored in an additional ticket can be used, instead of the session key provided by the server.

The TGT reply has the same structure as the AS reply, and the application ticket is built the same way as the TGT. The flags indicate if it is a ticket-granting ticket

or a service-granting ticket. The only part that is still unclear in this part of the protocol is the authenticator. But this is essentially the same as the authenticator used in Kerberos 4. Instead of the network address, the realm of the client is stored and again the protocol version number is added.

Finally, we can look at the client/server authentication exchange. The AP request in message (5) also contains the protocol version number and a message type. Also in this request, the client can set some options, to indicate which key has been used to encrypt the application ticket and if mutual authentication is required. Of course, the ticket and the authenticator is also sent. This time, the authenticator has two extra fields: a subkey, to be used only this application session and a sequence number to provide mutual authentication. In message (6), the service can authenticate himself presenting the timestamp and the subkey issued. The sequence number can be used to track messages in long communications between the client and AP.

Authentication Service
(1) $C \rightarrow AS : Options + ID_C + Realm_C + ID_{TGS} + Times + Nonce_1$
(2) $AS \rightarrow C : ID_C + Realm_C + TGT + E_{K_C}[K_{C,TGS} + ID_{TGS} + Realm_{TGS} + Times + Nonce_1]$
Ticket-granting Service
(3) $C \rightarrow TGS : Options + ID_{AP} + Times + Nonce_2 + TGT + Auth_{C,TGS}$
(4) $TGS \rightarrow C : ID_C + Realm_C + Ticket + E_{K_{C,TGS}}[K_{C,AP} + ID_{AP} + Realm_{AP} + Times + Nonce_2]$
Client/Server Service
(5) $C \rightarrow AP : Options + Ticket + Auth_{C,AP}$
(6) $AP \rightarrow C : E_{K_{C,AP}}[TS_2 + Subkey + Seq]$

with

$TGT = E_{K_{TGS}}[K_{C,TGS} + Flags + ID_C + Realm_C + NA_C + Times]$
$Ticket = E_{K_{AP}}[K_{C,AP} + Flags + ID_C + Realm_C + NA_C + Times]$
$Auth_{C,TGS} = E_{K_{C,TGS}}[ID_C + Realm_C + TS_1]$
$Auth_{C,AP} = E_{K_{C,AP}}[ID_C + Realm_C + TS_2 + Subkey + Seq]$

Threats/problems prevented:
- Fixes the most of the discussed limitations of version 4

New threats/problems:
- This protocol still has vulnerabilities [Koh91]

Figure 2.7: Dialogue 5, the Kerberos 5 protocol

2.4.3 Side protocols

We have now seen the main Kerberos protocol, but there are more protocols necessary in order to work wit Kerberos. For example, there should be an administration protocol to add new principals or change principal's credentials. This is done using a protocol between the client and a third Kerberos server, the Kerberos Administration Server (KADM). Note that the first Kerberos server is the AS and the second Kerberos server is the TGS. Furthermore, there are specific messages between the

client and an AP for detecting modifications, providing confidentiality or for transferring Kerberos credentials. However, in this study these additional protocols are not being discussed.

2.4.4 Compatibility support for Kerberos 4

To facilitate the upgrade from Kerberos 4 to 5, backward compatibility is supported in version 5. This means that existing Kerberos applications should be able to interoperate with the new implementation. The new Kerberos server may have a compatibility mode enabled, to be able to accept version 4 format requests and respond with version 4 format tickets and messages, next to the version 5 tickets and messages. This allows programmers to upgrade the Kerberos installation very slowly. However, with the backward compatibility mode enabled, the problems that occurred in version 4 will remain. That is why this mode should be turned off after some period of time. In 2000, MIT announced in a news group that support for Kerberos version 4 stopped, so they can fully concentrate on the development of version 5.

Chapter 3

Kerberos services

Now that we have seen a not so short introduction to the Kerberos protocol, we can look at the services that Kerberos provides. In the first section, a list of these services will be given and described. Next, the protocol elements of the previous chapter are being related to the services. It should clarify which elements are necessary to provide a certain service. This means that we need to look at the protocol from a different point of view. Each element of the protocol will be observed more carefully. This will introduce some new elements.

3.1 List of Services

We have seen the Kerberos protocol in detail, now we can look at a list of services and security aspects that are provided by Kerberos 5. First, let us look at the services:

sso **Single Sign On**
Single Sign On refers to a single identity that is shared across multiple systems. In Kerberos, a client will only have to authenticate himself once, and can make use of all Kerberized applications, without filling in his password again (for a certain amount of time).

cro **Cross-organizational authentication**
With the introduction of Kerberos realms, cross-organizational authentication is possible, so that users from one organization can use services from another organization.

mut **Mutual authentication**
Because Kerberos provides a secure communication between two principals, the two parties involved might like to ensure that the other principal is also authorized. This means that not only the client should authenticate himself, also the service principal should authenticate himself to the client (if necessary). Kerberos provides mutual authentication, the process of two principals proving their identities to each other.

u2u **User-to-user authentication**
Until now, we have seen Kerberos using only user-to-host authentication: The user makes a request for its credentials, and will eventually receive an application ticket to access a particular service. This will only work on small, managed work environments. When users configure their desktop servers with

a long-lived key, this long-lived key becomes a very attractive target for theft. So, this type of authentication cannot be used on public workstations. Those workstations are vulnerable to privacy attacks and hence cannot securely hold a long-lived secret. User-to-user authentication provides support for the use of Kerberos on public workstations. With this approach, the service principal and the client principal will share a secret. Because the public workstation's secret could be compromised when unattended, the secret should have a short lifetime. A session key would be a good secret. The service principal can request a TGT, with the session key stored in it. It gives his TGT to the client principal and he can provide it as an additional ticket when requesting a service ticket, requesting the service ticket should be encrypted with the session key in the additional ticket. For more information on this subject you can read [Dav89].

bat **Support batch jobs**
Kerberos tickets usually may not have a long lifetime, because that enlarges the threat of credential theft. This makes running batch jobs harder. Fortunately, Kerberos 5 provides mechanisms to facilitate these actions.

lon **Support jobs with long lifetime**
The same problem occurs for jobs that take a long time to execute. Also these kind of jobs are supported by Kerberos 5.

fwd **Authentication forwarding**
Sometimes it is necessary for a principal to allow a service to perform an action on its behalf. But normally, tickets are only valid from network addresses that are included in the ticket, because of security reasons. Kerberos 5 provides functionality that allows you to forward your authenticated identity to a different network address.

Kerberos deals with a couple of stakeholders. Clients would like to make sure that their credentials are safe. Also, they don't want that the application could tamper with the client's assets (stored on the client's machine). Furthermore they assume that the Kerberos system is always available. Application servers would like to make sure that only authorized clients can access their services, and that no one could tamper with the assets on their server. Also these stakeholders would like to see that their credentials are safe. Also the owner of the Kerberos server is a stakeholder. The system may not have access to its personal data or storage.

An attacker could be someone outside the system, trying to gain access to assets for which he has no authorization. But the attacker could also be a principal or even the owner of the Kerberos server. However, Kerberos was intended to be used in small, manageable environments, so in this case it is not very likely that an administrator or authorized user has bad intentions. Therefore I shall only look at possible attacks from an opponents that have no administrator rights and are no valid users.

pwd **Password attacks**
Kerberos has implemented various ways that make password attacks more difficult. For example, Kerberos will not send the password over the network, so it cannot be eavesdropped. However, if an opponent receives a TGT, he can try a password attack to decode the password-based key. One measure against this attack in version 5 is pre-authentication (optional), where the client (or opponent) does not receive a TGT before he is authenticated.

net **Network attacks**
The protocol is designed in such a way, that the network does not have to be trusted. Messages could be eavesdropped, changed or stolen. Therefore, all sensitive data in the protocol must be encrypted and may not leak.

rep **Replay attacks**
One special case of network attacks is a replay attack. When an intruder was able to eavesdrop or alter a message, he could try to perform a replay attack. This way, he can try to obtain authentication credentials, and to gain access to services that he was not authorized for. Making the Kerberos system time critical, replay attacks should be more difficult.

cre **Credential theft**
A replay attack could already be considered credential theft. An attacker tries to capture someone's authentication credentials. When performing a replay attack, the attacker would usually not be able to read sensitive data. Another form of credential theft can occur by breaking the confidentiality of the messages.

mal **Distribution of malicious Kerberos services**
When a client finally has gained access to a particular service, he must be able to verify the identity of the service. If this is not possible, an attacker could imitate a service and try to intercept some credentials. With the introduction of mutual authentication, this attack should not be possible anymore.

3.2 The necessary elements

The Kerberos dialogue consists of three parts, as shown in figure 2.7. All elements will be related to services or security properties (to counter a threat). These services or security properties are listed in the previous section. In the following subsections, we describe each element in short and show the relation with the service.

3.2.1 The authentication part

The first message

In the first message of the protocol as shown in figure 2.7, the client wishes to obtain authentication credentials. He generates a request to the AS. This request consists of a protocol version number, a message type which in this case indicates the request is an authentication request, possibly some pre-authentication data and a request body. Let us look at each element more precisely.

The protocol version number is added for compatibility reasons. Recall that Kerberos 5 was compatible with Kerberos 4. The version number determines which actions should be taken, and how the response should look like.

The message type is included because of efficiency reasons. In the previous chapter we saw that the structure of the authentication request looks a lot like the ticket-granting request, so the same structure for both requests is used. The message type and the protocol version number are both elements that can be considered unnecessary for any provided service or to counter an attack. Both are introduced to facilitate the implementation.

The pre-authentication data field is left empty in most authentication requests. It may contain information needed to initially verify the client's identity before a response is given. The field can also be used to help the AS or client to select the key needed for generating or decrypting the response. This form of pre-authentication data is useful for supporting the use of hardware, such as smart cards. These uses of the pre-authentication data make it more difficult (although not impossible) to do password guessing attacks. A user now first has to perform authentication, before the TGT will be handed to him. However, this mechanism is left open and can be implemented in many different ways [Koh93].

The request body contains many elements: some options, the name and realm of the client, the identity of the TGS, some time fields, a nonce, a field that indicates which encryption is used for the private parts of the message, and a list of addresses from where the requested TGT can be used. Let us first look at which options can be set. Note that there is a difference between options and flags. Options occur in requests, to configure the requested ticket. Flags are set in the requested tickets, depending on which options were set.

The POSTDATED option is used to obtain a POSTDATED ticket. With a POSTDATED ticket, no services can be accessed yet. The ticket will also have the flag INVALID set. A POSTDATED ticket also contains a start time (in the future), that indicates when the service should be accessed. The POSTDATED ticket must be validated (with the option VALIDATE, described in the next subsection) before use by presenting it to the TGS after the start time has been reached. The POSTDATED option supports executing batch jobs. It is possible to obtain a POSTDATED TGT, but this option is more useful for service tickets.

The ALLOW-POSTDATE option is used to obtain a MAY-POSTDATE ticket. This option can only be set on the initial request (to the AS), or in a request to the TGS if the TGT has its MAY-POSTDATE flag set. This flag is only interpreted by the TGS. It tells the TGS that POSTDATED tickets may be issued. Without a MAY-POSTDATE TGT, no POSTDATED service tickets can be issued, so this option also contributes to the support of executing batch jobs.

The RENEWABLE option can be used to obtain a RENEWABLE ticket. Kerberos tickets with a long lifetime are more vulnerable to theft. But sometimes a service needs to be accessed for a long time. To support this functionality without increasing the possibility of ticket theft, RENEWABLE tickets can be used. the RENEWABLE flag is only interpreted by the TGS and can be used to obtain a replacement ticket that expires at a later date. With such a ticket, you can access a server for a longer time, but also can renew a TGT for more than a day.

The RENEWABLE-OK option can be set to indicate that a RENEWABLE ticket will be accepted if the requested ticket lifetime cannot be provided. It works exactly like the RENEWABLE option, but with this option the client does not prefer such a ticket. If this option is not set and the client requests a ticket with an expiration time too far in the future, the maximum expiration time for tickets is used instead.

The PROXIABLE option can be set to obtain a PROXIABLE ticket. It may only be set in the initial request or in a request to the TGS when presenting

a PROXIABLE ticket. A PROXIABLE flag is only interpreted by the TGS and allows a principal to issue service tickets from a different network address. Therefore, the PROXY option must be set in the following request (to the TGS). The valid network addresses are included in the TGT. Now, a service can perform an action on behalf of the principal.

The FORWARDABLE option is used to obtain a FORWARDABLE ticket. It also may only be set in the initial request or in a request to the TGS when presenting a FORWARDABLE ticket. A FORWARDABLE flag is only interpreted by the TGS and allows a principal to issue tickets from a different network address. There is a slight difference with the PROXIABLE flag, which is used to forward only service tickets. With the FORWARDED option in the request to the TGS, you can transfer your whole identity to a different machine.

Not all options have been explained yet. Some options can only be used when granting service tickets, so we don't have to bother about them yet. They will be described in the next subsection. First, we still need to look at the remaining elements of the request body.

The name of the client is submitted to show the AS which user is requesting a TGT. The name of the client is defined as a principal name and consists of two fields, a type and a name. The type indicates if the principal is a user or a service. There are some more types defined, indicating a special or unique principal. However, because this subfield will probably be left out in the future [Koh93], we shall not discuss these. The second subfield, the name, is a string which holds the actual name of the client.

The realm of the client forms the principal identifier together with the principal name. It is encoded as a string. Although you can technically choose any name for a realm, interoperability across realm boundaries requires agreement on how realm names are to be assigned, and what information they imply. There are currently two important styles of realm names: domain names and X.500 names. Other realm names fall into the category 'other' or 'reserved'. Reserved names are unlikely to be used, unless there is a very strong argument for not using the 'other' category.

The name of the requested service, in this case the name of the TGS. Since a ticket can only be issued within its own realm, the realm name of the TGS is the same of the client and it is not necessary to include it twice. The AS can determine the identity of the TGS and can select the correct key to encrypt the TGT.

The above three elements contribute to the TGT generation. Remember that with the introduction of ticket-granting, Single Sign On becomes possible. The three elements just listed contribute to this functionality.

The start time is an optional field to request some time settings in the TGT. This field indicates the desired start time for the requested ticket. If the start time is invalid (for example it is in the past or has the wrong structure), the ticket's start time is set to the current time of the AS. If the requested start time is in the future, but the POSTDATED option is not set, an error is returned. Time fields in tickets are important because they make it harder to steal tickets. However, the configuration of these time fields are introduced for usability. Tickets can be configured easier for long or batch jobs.

The expiration time is used to set the desired expiration time of the TGT. If the requested expiration time minus the start time is less than some determined minimum lifetime, the AS returns an error. If no (valid) expiration time has been issued, the ticket's expiration time will be set to the start time added with the maximum lifetime associated with the client, the server or the policy file. The policy file will be used if no maximum lifetimes are associated with the client and the server.

The renewal time is an optional field which sets the renewal time of the ticket if the RENEWABLE (and sometimes the RENEWABLE-OK) option has been set. The ticket's renewal time will be the requested value. If this is an invalid value, the start time plus the renewal time associated with the principal or with the policy file is used.

A nonce, which holds a random number generated by the client, is added to assure that the response will be fresh. The same number (encrypted) is included in the response.

The encryption type specifies the desired encryption algorithm to be used in the response. There are a few encryption algorithms provided. If the client chooses another encryption type, an error is returned. This field facilitates the choice of encryption being used. However, it is the actual encryption that contributes to the security of the protocol.

A list of network addresses is included in the initial request. It specifies the addresses from which the requested ticket is to be considered valid. Normally it contains only the addresses associated with the client's host. If a PROXY is requested, this field will also contain other addresses.

We have now described every element of the first message of the protocol that is shown in figure 2.7. These are related to a service or a way to counter an attack. These relations are summarized in Table 3.1. The second, third and fourth column depict the provided services. The fifth column shows how the element is denotated in figure 2.7.

The second message

If all checks have succeeded, the AS will generate a response, which includes the requested TGT, some encrypted elements and some unencrypted elements. We shall describe the TGT first.

The ticket version number can be compared with the protocol version number. It just determines which ticket format is used. In this case it will contain the number 5. This element will also not contribute to any service or to counter an attack, but rather for efficiency reasons.

The realm is included to specify for which realm the ticket has been issued. It also serves to identify the realm part of the identity of the principal. Note that it will have the same values as the realm of the client in the first line of the protocol.

The name of the TGS, together with the realm, completes the identity of the TGS. The client needs to verify that its identity is the same as in the request. Before decrypting the encrypted part of the message (not the TGT), he can assume that he received the ticket he requested, by verifying the identity of the TGS.

Elements from message (1)

Element	Service	Attack	Other	Notation
Protocol version number	-	-	Efficiency reasons	-
Message type	-	-	Efficiency reasons	-
Pre-authentication data	-	pwd	-	-
Option POSTDATED	bat	-	-	*Options*
Option MAY-POSTDATE	bat	-	-	*Options*
Option RENEWABLE	lon	-	-	*Options*
Option RENEWABLE-OK	lon	-	-	*Options*
Option PROXIABLE	fwd	-	-	*Options*
Option FORWARDABLE	fwd	-	-	*Options*
Name of the client	sso	-	-	ID_C
Realm of the client	sso	-	-	$Realm_C$
Name of the TGS	sso	-	-	ID_{TGS}
Start time field	sso	-	-	$Times$
Expiration time field	lon	-	-	$Times$
Renewal time field	lon	-	-	$Times$
Nonce	-	rep	-	$Nonce_1$
Encryption type	-	-	Efficiency reasons	-
List of network addresses	fwd	-	-	-

Table 3.1: Elements from message (1) of the Kerberos 5 protocol

Because the realm and the name of the TGS also come back in the encrypted part of the message, this information might seem redundant. But these elements do have a significant meaning in a later stage of the protocol. To support user-to-user authentication, it is possible that the TGS encrypts the service ticket with a provided session key (that is shared between the TGS and AP). With cross-realm authentication, the AP could be registered in multiple realms, with different keys in each. The unencrypted realm field specifies which key it should use to decrypt the ticket. With the name of the TGS, the server can decide if the party is trustworthy.

The TGT also has an encrypted part. Its elements will be described next.

Flags are used to indicate how the ticket has been configured. There are eleven different flags: FORWARDABLE, FORWARDED, PROXIABLE, PROXY, MAY-POSTDATE, POSTDATED, INVALID, RENEWABLE, INITIAL, PRE-AUTHENT and HW-AUTHENT. These flags have already been explained in the previous chapter and most of them have an obvious link with the options. For example, it is easy to see that the RENEWABLE flag is set by the RENEWABLE option in the request. One extra note on the INVALID flag has to be mentioned: It is clear that the flag is set when a POSTDATED ticket is requested. So it contributes to support executing batch jobs. But also when a check by the AS or the TGS fails, this flag will be set. A check fails if the ticket has some unexpected behavior, for example an expired lifetime, or an invalid network address. This could be the cause of an opponent requesting unauthorized access to a service. Thus the INVALID flag also helps against credential theft attacks.

The session key is included to provide a secure communication between the client and the TGS. With this key, the TGS can verify that the encrypted

part of the message could only be generated by the client. We shall see that the session key is also a measure against replay attacks, because it encrypts the authenticator. I shall elaborate on that in section 3.2.2.

The identity of the client, which consists of the realm and the name of the client, is included. This way, the TGS can verify that the ticket really belongs to the person who presented it, without prompting the client for his password.

The list of the client's addresses is copied from the initial request. With this list, the TGS can verify if the ticket is used from a valid location. If this field is empty, the ticket can be used from anywhere. Normally, the client's host address will be in this field. Kerberos can be configured in such a way that an empty list of addresses may be refused. Using an address policy reduces the risk that an attacker can successfully use a stolen ticket.

The transited field is used when the client performs cross-realm authentication. In some cases, where there are many realms, it is inefficient to register each realm in every other realm, so a different hierarchy is used. This means that in order to contact a service in another realm, it is sometimes necessary to contact the remote TGS in one or more intermediate realms. These realms are called the transited realms, and their names are recorded in this field. This is to make sure that the end service knows all of the intermediate realms that were transited, in order to decide whether or not to accept the authentication (e.g. if all realms are trustworthy).

The authentication time indicates the time of initial authentication for the specified principal. It is included in the ticket to provide additional information to the end service. An end service could refuse tickets for which the authentication time is too far in the past, which might point out credential theft or a replay attack.

The start time indicates from when the ticket is valid. If this field is empty, the authentication time is considered the start time.

The expiration time indicates when a ticket loses its validity. Together with the start time, it forms the lifetime of the ticket. Sometimes services may have their own limits on the ticket's lifetime. Therefore this field is actually an upper bound of the expiration time.

The renew until time is only present when we deal with a RENEWABLE ticket. This indicates the maximum end time, including all the renewals. When a ticket is renewed, this field is copied into the new ticket.

Authorization data is the data passed through by the client. This can only happen in the TGS request. It can be used to facilitate proxy services, as we shall see in the subsequent request.

Additional tickets may be provided to request for renewal, proxy or forwarded cases. This field will hold for example the ticket to be renewed. Also a TGT of the service principal could be stored in this field, to support user-to-user authentication.

Besides the TGT, the AS will give the client some more information. The client will see some unencrypted elements, for example his identity. He verifies that this is the same as in his request. This can be compared with the address on an envelope: If the address is incorrect, you don't open the envelope. If any pre-authentication data is set, they may be used to derive the proper secret key to decrypt the message. Also the message will contain a protocol version number and message type again. The encrypted elements are however more interesting to look at.

The identity of the TGS is used to verify if the requested ticket is for the correct service.

A session key is also included outside the TGT to provide a secure communication between the client and the TGS. The TGT cannot be read by the client, so it is necessary to add the session key in the TGT, as well in the encrypted part of the message.

A time field indicating the client's last request is added to aid the user in discovering unauthorized use of its identity.

The nonce is returned, so the client can be assured of the freshness of the message.

A key-expiration field is included to indicate how long the client's secret key is still valid. Expiration might be the result of password aging. This will strongly recommend the user to change or renew its password. This field does nothing else than showing information, but because it causes the client to change or renew its password, it can be seen as a countermeasure against password attacks.

The flags, authentication time, start time, expiration time, renew until time and the list of the client's network addresses are duplicates of those in the TGT, provided so the client may verify they match the intended request.

Now we have discussed all elements of the second message of the protocol. Their relations with provided services or countering attacks are summarized in Table 3.2.

3.2.2 Granting tickets

The third message

At this point, we have discussed many elements already. But from this part on it only gets better, because the request and the reply in the third and fourth message have almost the same structure as the ones in message 1 and 2. We can skip the following elements, because they have no different purpose in the third message: Protocol version number, message type, the options POSTDATED, ALLOW-POSTDATE, RENEWABLE, RENEWABLE-OK, PROXIABLE, FORWARDABLE, the time fields start time, expiration time, renewal time, the nonce, the encryption type and the list of network addresses. Also, the service ticket will have exactly the same elements as the TGT. The only difference is that it has not its INITIAL flag set. The name and realm of the client will also not be needed in this request, since they are already stored in the TGT. And instead of requesting a ticket for the TGS, the client now provides the name of the service to request a service ticket.

Let us now look at the new options that can be set in this request:

The PROXY option can be set to obtain a PROXY service ticket. Together with a PROXIABLE ticket-granting ticket, it allows a principal to request a service from a different network address.

The FORWARDED option can be used set to obtain a FORWARDED service ticket. Together with a FORWARDABLE ticket-granting ticket, it allows a principal to request a service from a different network address. A client can also request a FORWARDED ticket-granting ticket, so that he can transfer its whole identity to a different machine.

Elements from message (2)

Elements	Service	Attack	Other	Notation
Protocol version number	-	-	Efficiency reasons	-
Message type	-	-	Efficiency reasons	-
Pre-authentication data	-	pwd	-	-
Name of the client	-	-	Verification	ID_C
Realm of the client	-	-	Verification	$Realm_C$
Ticket-granting ticket	sso	-	-	TGT
Encrypted part	-	net	-	$E_{K_C}[...]$
Encrypted part elements	Service	Attack	Other	Notation
Name of the TGS	-	-	Verification	ID_{TGS}
Realm of the TGS	-	-	Verification	$Realm_{TGS}$
Session key	-	net, rep	-	$K_{C,TGS}$
Last request	-	cre	-	-
Nonce	-	rep	-	$Nonce_1$
Key-expiration field	-	pwd	Notification	-
Flags	-	-	Verification	-
Authentication time	-	-	Verification	$Times$
Start time	-	-	Verification	$Times$
Expiration time	-	-	Verification	$TImes$
Renew until time	-	-	Verification	$Times$
List of client's addresses	-	-	Verification	-
Ticket elements	Service	Attack	Other	Notation
Ticket version number	-	-	Efficiency reasons	-
Realm of the TGS	u2u	-	-	-
Name of the TGS	u2u	-	-	-
Encrypted ticket elements	Service	Attack	Other	Notation
INITIAL flag	-	-	Efficiency reasons	$Flags$
HW-AUTHENT flag	-	pwd	-	$Flags$
PRE-AUTHENT flag	-	pwd	-	$Flags$
INVALID flag	bat	rep, cre	-	$Flags$
RENEWABLE flag	lon	-	-	$Flags$
POSTDATED flag	bat	-	-	$Flags$
MAY-POSTDATE flag	bat	-	-	$Flags$
PROXIABLE flag	fwd	-	-	$Flags$
PROXY flag	fwd	-	-	$Flags$
FORWARDABLE flag	fwd	-	-	$Flags$
FORWARDED flag	fwd	-	-	$Flags$
Session key	-	net, rep	-	$K_{C,TGS}$
Name of the client	sso	-	-	ID_C
Realm of the client	sso	-	-	$Realm_C$
List of client's addresses	fwd	rep, cre	-	NA_C
Transited field	cro	-	-	-
Authentication time	-	rep, cre	-	$Times$
Start time	bat	rep, cre	-	$Times$
Expiration time	bat, lon	rep, cre	-	$Times$
Renew until time	lon	-	-	$Times$
Authorization data	fwd	-	-	-
Additional tickets	bat, lon, fwd, u2u	-	-	-

Table 3.2: Elements from message (2) of the Kerberos 5 protocol

The RENEW option can be used in a request to the TGS to renew a ticket. The ticket must have its RENEWABLE flag set and its renew until not expired.

The VALIDATE option indicates that the client wants to validate a POST-DATED ticket. The request is only honoured if the ticket has its POST-DATED and INVALID flag set. Also, a ticket cannot be validated before its starttime.

The ENC-TKT-IN-SKEY option points out that the service ticket should be encrypted in the session key from the additional ticket provided. This option supports user-to-user authentication.

The pre-authentication data field is now used to store the authenticator and the ticket. The field is constructed in such a way that it can provide different functionalities. In the first message it makes password guessing attacks harder. In the third message it facilitates the ticket request and authentication to the TGS.

Authorization data is specific to the end service and must be encrypted with the subsession key (provided in the authenticator). If the subsession key is not present, it should be encrypted with the session key shared between the client and the TGS. It is expected that the field will contain the names of service specific objects. Authorization data from the principal can be passed through using the ticket field 'Authorization data'. It can be used to issue a proxy that is valid for a specific purpose. Consider the following example: A client wishes print a file from a certain server. The printer however is on a different server. By specifying the file name in the authorization data, the server that holds the file knows that the print server can only use the client's credentials when accessing the particular file to be printed. The print server cannot access other files. So, the authorization data field can be used to facilitate proxy services.

The pre-authentication field contains the authenticator and the TGT. The TGS can verify if the TGT is addressed to him (think of the envelope again). The authenticator is used to prevent replay attacks. With the authenticator, the client shows that he knows the encryption key in the ticket. It contains the following elements:

Authentication version number to identify which authenticator version is used. In this case it will hold the value 5.

Identity of the client to check if it matches with the identity provided in the ticket and that of the sender.

A timestamp when the authenticator was generated, a timestamp is used because it must have a short lifetime. If the timestamp is too far in the past, probably it is being replayed and the TGS can refuse the request.

A subsession key may be included to use an alternative encryption key. Each ticket contains a session key that is used by the client to encrypt the authenticator. The session key may be used subsequently during that session. This means that the same ticket may be used repeatedly to gain service from a particular server. This leads to the risk that an opponent will replay the messages from an old session to the client or the server. A subsession key can be used only for that one connection. A new access will lead to the use of a new subsession key.

A sequence number may be included as a way to counter replay attacks. The sequence numbers should be random and non-repeating to be successive.

A checksum of the data may be used if the AP is still suspicious about the request.

Elements from message (3)

Elements	Service	Attack	Other	Notation
Name of the service	sso	-	-	ID_{AP}
Pre-authentication data	-	-	Efficiency reasons	-
Ticket-granting ticket	sso	-	-	TGT
Authenticator	-	rep	-	$Auth_{C,TGS/AP}$
Option PROXY	fwd	-	-	$Options$
Option FORWARDED	fwd	-	-	$Options$
Option RENEW	lon	-	-	$Options$
Option VALIDATE	bat	-	-	$Options$
Option ENC-TKT-IN-SKEY	u2u	-	-	$Options$
Authenticator elements	Service	Attack	Other	Notation
Authentication version number	-	-	Efficiency reasons	-
Name of the client	-	rep	-	ID_C
Realm of the client	-	rep	-	$Realm_C$
Timestamp	-	rep	-	$TS_{1/2}$
Subsession key	-	net, rep,	-	$Subkey$
Sequence number	-	rep	-	Seq
Checksum	-	rep	-	-

Elements from message (4)

Element	Service	Attack	Other	Notation
Service ticket	sso	-	-	$Ticket$

Table 3.3: Elements from message (3) and (4) of the Kerberos 5 protocol

Since it is infeasible to find two plaintexts that have the same checksum, the server can verify that the service ticket really belongs to this client.

The fourth message

The fourth message in the protocol uses exactly the same structure as the second message. No new elements are introduced. Instead of receiving a TGT, the client will receive a service ticket. Sometimes the client will receive a TGT from the TGS, if a request for another realm was made. The client will provide the remote TGT to the TGS in the corresponding realm. The service ticket will also contribute to Single Sign On. It has a shorter lifetime than the TGT, but thanks to the TGT, a new service ticket can be requested within the lifetime of the TGT. The service tickets are the last link of the Single Sign On process.

All new elements of message (3) and (4) are listed in Table 3.3.

3.2.3 Accessing the application

The fifth message

The last elements are introduced in accessing the AP. In the fifth message, some new options can be set. Furthermore, the service ticket and authenticator, which we already have described, are provided. Also this message contains fields for the version number and the message type.

Elements from message (5)

Elements	Service	Attack	Other	Notation
Option USE-SESSION-KEY	u2u	-	-	*Options*
Option MUTUAL-REQUIRED	mut	mal	-	*Options*
Elements from message (6)				
Timestamp	mut	mal	-	$TS_{1/2}$
Subsession key	-	net, rep	-	*Subkey*
Sequence number	-	rep	-	*Seq*

Table 3.4: Elements from message (5) and (6) of the Kerberos 5 protocol

The USE-SESSION-KEY option can be set to let the AP know that the application ticket is encrypted with the session key from the APs TGT. Normally the ticket is encrypted with the service's secret key, but when the client requested the TGS an application ticket with the ENC-TKT-IN-SKEY option set, the TGS will use the session key that was stored in the additional ticket. With this option, the client can let the AP know that the request is part of the user-to-user authentication.

The MUTUAL-REQUIRED option can be set if the client also wants to authenticate the AP. Because only the AP could decrypt the application ticket containing the session key (shared between the client and AP), the client can be sure of the identity of the AP.

The sixth message

The sixth message is only sent if in the fifth message the MUTUAL-REQUIRED option was set. The service will send back a message containing the protocol version number, the message type and an encrypted part which is required to support mutual authentication. This part consists of the following elements:

A timestamp is included to show to the client that it has the identical value of the timestamp in the authenticator. This way, the client can verify the identity of the AP.

The subsession key may be included if the AP desires to negotiate a different subkey.

The sequence number is an optional field that can be used by the server for messages sent to the client during this session. Messages are than sequenced numbered and hinders replay attacks.

All elements of the last two messages are listed in Table 3.4.

Chapter 4

Kerberos limitations

In the previous chapter we have looked at the services and security aspects of Kerberos 5 and which elements of the protocol contribute to these services. This gives us a good picture of what Kerberos is capable of. In this chapter, we shall see which vulnerabilities Kerberos 5 still has, and how they can be improved. Most of the problems have also been discussed in [Bel91]. Other problems were already known at MIT, and are considered assumptions [Koh93]. This thesis will end with some suggestions for further improvement of Kerberos and a short overview of what was presented.

4.1 Kerberos vulnerabilities

4.1.1 Kerberos assumptions

The following vulnerabilities are known to the developers of Kerberos. Denial of service (DOS) attacks, for example, are not solved with Kerberos. In the protocol, an intruder can prevent an application from participating in the proper authentication steps, by intercepting the messages (with a man-in-the-middle attack). Because this is possible, the requirement that the system should always be available is not entirely met. Password guessing attacks are also not solved with Kerberos. The system does provide ways to make these attacks more difficult, but this still does not make offline guessing attacks impossible. Also, there is no mechanism that prevents the user to choose a poor password.

Furthermore, Kerberos makes no provisions for host security. It assumes that it is running on trusted hosts with an untrusted network. Although this is not a protocol weakness, it is important to mention. If your host security is compromised, then Kerberos is compromised as well. The impact depends on the compromised host. All tickets stored on that machine can be used by the attacker to gain access to services, but only until those tickets expire. However, if the attacker could intercept the password, he could impersonate the user at any time for all services.

Another assumption is that the Kerberos server, that holds all the principals passwords in that realm, can be trusted. If this host security is compromised, the entire realm is also compromised.

4.1.2 Replay attacks

According to Bellovin and Merrit [Bel91], Kerberos is not as resistant to penetration as it should be. The authenticator for instance, is not sufficient enough to prevent replay attacks. In Kerberos 4 the problem with authenticators was the lifetime of the object. The assumption was that replays are unlikely to be made within the lifetime of the authenticator (about five minutes). Together with the included network address in the ticket and authenticator, it would be able to prevent replay attacks. Morris described an attack based on the slow increment rate of the initial sequence number counter in some TCP implementations [Mor85]. This made it possible to spoof one half of a pre-authenticated TCP connection without ever seeing any responses from the targeted host. This attack would work in Kerberos if the opponent is accompanied with a stolen authenticator (that is used within the correct time skew).

In Kerberos 5, a list of used authenticators is maintained. However, Bellovin and Merrit state that caching authenticators could not solve this problem. This is because there are problems to store authenticators at TCP-based servers. UDP-based servers have problems with retransmissions of the client's request if the answer was lost, because UDP does not provide guaranteed delivery.

4.1.3 Time attacks

We have seen that authenticators rely on machines' clocks being synchronized. If a host can be misled about the correct time, such an authenticator can be replayed easily. An alternative approach could be considered by replacing the authenticator by a challenge/response mechanism. In this situation, the client would present a ticket, but no authenticator. Instead, the server responds with a nonce encrypted with the session key shared between the server and the client. The client can respond the challenge, proving that he knows the right session key. If the server has validated the response, he provides the client with the requested service (or a service ticket, in case of the TGS).

A time attack would not be possible anymore, but does increase the protocol costs. While making requests to the TGS and different APs, more messages must be exchanged. Also it brings new problems along with respect to UDP-based query servers able to retain state in order to complete the authentication process [Bel91].

4.1.4 Scope of tickets

Kerberos 5 introduces ticket-forwarding, but this introduces a new problem of cascading trust. A host may be willing to trust an intermediary host, and this intermediary host trusts the end host, but the initial host may not be willing to accept tickets originally created on the end host, because it believes it is insecure. Kerberos sets a flag to indicate that a ticket was forwarded, but does not include the original source.

However, to include the network address is not so useful. Given the assumption that the network is insecure, no extra security is gained by relying on the network address.

4.2 Overview

In this thesis I presented what Kerberos is capable of, and how these services were achieved. Primary functionalities are Single Sign On, cross-organizational authentication and mutual authentication. Kerberos was implemented with in mind that the network cannot be trusted. This has its consequences for the protocol. With the use of tickets and authenticators, eavesdropping and replay attacks should be countered. Unfortunately, some forms of replay attacks are still possible, according to Bellovin and Merrit [Bel91]. Also, some more problems and limitations have been discovered.

In the eighties, Kerberos 4 became a popular system, but still had some vulnerabilities and limitations. That is why Kerberos 5 was developed. It provided better security and more functionality, such as executing batch jobs and long jobs, user-to-user authentication and forwarding credentials. However, some security problems remained and the functionality introduced some new problems. At this moment, Kerberos is still a good SSO authentication system, that prevents many network attacks. But considering the attacks that are still possible, it is not ready for use in large, public networks.

Bibliography

[Bel91] Bellovin, S. and Merritt, M. Limitations of the Kerberos Authentication System. In *USENIX Conference Proceedings*, pages 253–267, Dallas, TX, Winter 1991. USENIX.

[Bry88] Bryant, B. Designing an Authentication System: a Dialogue in Four Scenes, february 1988.

[Dav89] Davis D. and Swick, R. Workstation Services and Kerberos Authentication at Project Athena. Technical report, MIT, march 1989.

[Den81] Denning, D. and Sacco, G. Timestamps in Key Distributed Protocols. *Communication of the ACM*, 24(8):533–536, 1981.

[Jue85] Jueneman, R. and Matyas, S. and Meyer, C. Message Authentication. *IEEE Communications Magazine*, 23(9):29–40, september 1985.

[Koh89] Kohl, J. The use of encryption in Kerberos for network authentication. In *Proceedings, Crypto '89*. Springer-Verlag, 1989.

[Koh91] Kohl, J. and Neuman, C. and Ts'o, Y. The Evolution of the Kerberos Authentication Service. In *Proceedings of the Spring 1991 EurOpen Conference*, 1991.

[Koh93] Kohl, J. and Neuman, C. The Kerberos Network Authentication Service (V5). Technical report, ISI, september 1993.

[Mer90] Merkle, R. Fast Software Encryption Functions. In *Crypto '90 COnference Proceedings*, Santa Barbara, CA, August 1990.

[Mil89] Mills, D. Network Time Protocol (NTP). Network Working Group Request for Comments: 958, 1989.

[Mor85] Morris, R. A weakness in the 4.2BSD UNIX TCP/IP Software. In *Computing Science Technical Report No. 117*, Murray Hill, New Jersey, February 1985.

[Nee78] Needham, R. and Schroeder, M. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, december 1978.

[Sta03] Stallings, W. *Network Security Essentials: Applications and Standards*, chapter 4: Authentication Applications, pages 88–105. Alan R. Apt, second edition, 2003.