A

Seminar report

On

Kerberos

Submitted in partial fulfillment of the requirement for the award of degree of Bachelor of Technology in Computer Science

SUBMITTED TO:
www.studymafia.org

SUBMITTED BY:
www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Kerberos;** I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to …………..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Table of Contents

Abstract

Kerberos is a security system that helps prevent people from stealing information that gets sent across the wires from one computer to another. Usually, these people are after your password.

The name "Kerberos" comes from the mythological three-headed dog whose duty it was to guard the entrance to the underworld. The Kerberos security system, on the other hand, guards electronic transmissions that get sent across the Internet. It does this by scrambling the information -- encrypting it -- so that only the computer that's supposed to receive the information can unscramble it. In addition, it makes sure that your password itself never gets sent across the wire: only a scrambled "key" to your password.

Kerberos is necessary because there are people who know how to tap the lines between computers and listen for your password. They do this with programs called "sniffers", and the only way to stop them would be to physically guard every inch of the Internet ... computers, cables and all. This, of course, is impossible. As long as there are physically insecure networks in the world, we'll need something like Kerberos to maintain the integrity and security of our electronic communications

Introduction

Kerberos gets its name from Greek mythology. Cerberus, also known as Kerberos, was a three headed beast that guarded the Underworld and kept the living from entering the world of the dead. Kerberos protocol design began in the late 1980s at the Massachusetts Institute of Technology (MIT), as part of project Athena. It is a secure authentication mechanism designed for distributed severs, which assumes the network is unsafe. It enables a client and a server to mutually authenticate before establishing a connection. The first public release was Kerberos version 4, which leads to the actual version (v5) in 1993 after a wide public review. It followed the IETF standard process and its specifications are defined in Internet RFC 1510. Originally designed for UNIX, it is now available for all major operating systems, freely from MIT or also through commercial versions.

The problem that the Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a work station cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a reply attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public -key encryption.

Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88]is still widely used. Version 5[KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 1510).

Benefits of using Kerberos

Nothing is easier today than to catch credentials over a network. If we try to run a sniffer in our environment we will see that we certainly get a login/password combination within a few minutes. This could lead to an unauthorized use of our network services and would certainly compromise all data present in our environment; even protected confidential data, as most users are using only one password for every application. Authentication is critical to security. Too many applications use a weak authentication mechanism, like clear text passwords or, even worse, rely on the "honesty" of client applications, known as authentication by assertion. However, it is not the primary role of an application to manage security. Consider a mail server: its role is to deliver email messages over the network to the appropriate recipients, but not to verify the user's identity! This is where Kerberos comes in. It has the advantage to manage secure authentication from a central location, and for many applications. For each application that requires this service, it is a reliable, simple and easy to manage solution to use Kerberos. Furthermore, it unloads application servers from this time consuming authentication task and allows concentrating on their primary function.

Motivation

If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users are served by a central time sharing system, the time sharing operation must provide the security. The operating system can enforce access control policies based on user identity and use the logon procedure to identify users.

Today, neither of these scenarios is typical. More common is a distributed architecture consisting of dedicated user work stations (clients) and distributed or centralized servers. In this environment, three approaches of security can be envisioned:

1. Rely on each individual client workstations to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
2. Requires that client systems authenticate themselves to servers, but trust the client system concerning the identity of the user.
3. Requires the user to prove identity fro each service invoked. Also requires that severs prove their identity to clients.

In a small, closed environment, in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice. But in a more open environment, in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed by the server. The third approach is supported b Kerberos. Kerberos assumes distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

The first published report on Kerberos [STEI88] listed the following requirements for Kerberos:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all the services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large number of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78]. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. It is a property of DES that if cipher text (encrypted data) is decrypted with the same key used to encrypt it, the plaintext (original data) appears. If different encryption keys are used for encryption and decryption, or if the cipher text is modified, the result will be unintelligible, and the checksum in the Kerberos message will not match the data. This combination of encryption and the checksum provides integrity and confidentiality for encrypted Kerberos messages. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena (BRYA88) and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/ server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner.

What does Kerberos do?

How does Kerberos allow you to authenticate yourself? Let us try and co-relate this to how does one authenticate himself in real life? Typically, you show your driver's license--or ID card, if you're not of driving age

What does this show? It shows that there is an agency (the one that issued the license or card) that has linked a given identity to a physical likeness. This physical likeness usually consists of a photo and some physical stats, and is considered to be uncopiable.

The identity consists of a name and an address, and some other information, such as a birth date. In addition, there may be some restrictions on what the named person can do; for instance, he or she may be required to wear corrective lenses while driving. (In many cases, this restriction is implicit: one can't drink until the age of 21, based on the birth date on the card.) Finally, the identification has a limited lifetime, represented by the expiration date on the card.

Note that this demonstration of identity is contingent on a number of things. First of all, the card must not be tampered with (such as changing the birth date, or the name, or the photo). Secondly, the person performing the authentication must accept the agency that issued the card. (Many supermarkets won't accept out-of-state or even out-of-town ID's when verifying checks.) There are some other concerns, such as that the person being authenticated really *hasn't* changed in appearance or name, or that the card is stolen, and so forth.

Kerberos works in basically the same way. It's typically used when a user on a network is attempting to make use of a network service, and the service wants assurance that the user is who he says he is. To that end, the user presents a *ticket* that is issued by the Kerberos ***authentication server*** **(AS)**, much as a driver's license is issued by the RTO. The service then examines the ticket to verify the identity of the user. If all checks out, then the user is accepted.

Therefore, this ticket must contain information linking it in a fool proof manner to the user. Since the user and the service don't meet face to face, a photo is of no use. Therefore, the ticket must demonstrate that the bearer knows something only its intended user would know, such as a password.

Furthermore, there must be safeguards against an attacker stealing the ticket, and using it later.

Assumptions Kerberos Makes

Kerberos does make some assumptions about the environment it lives in. It assumes that users won't make poor choices for passwords. If a user selects a password like ``password" or ``nothing," then an attacker who intercepts a few encrypted messages will be able to mount a *dictionary attack*, in which he tries password after password to see if it decrypts messages correctly. Success means that the user's password has been guessed and that the attacker can now impersonate the user to any verifier.

Similarly, Kerberos assumes that the workstations or machines are more or less secure, that only the network connections are vulnerable to compromise. In other words, Kerberos assumes that there is no way for an attacker to position himself between the user and the client in order to obtain the password in that manner.

The (High Level) Details

Let's take a look at how Kerberos goes about things. Both the user and the service are required to have keys registered with the Authentication Server. The user's key is derived from a password that he chooses; the service key is a randomly selected key (since no person is available to type in a password).

Let us imagine that messages are written on paper (instead of being electronic), and are ``encrypted" by being locked in a strongbox by means of a key. In this ``box world," principals are initialized by making a physical key and registering a copy of the key with the AS.

1. First the user sends a message to the AS: ``I, Samit Mehra User, would like to talk to Yahoo Server."
2. When the AS receives this message, it makes up two copies of a brand new key. This is called the ***session** key*. It will be used in the direct exchange between user and service.
3. It puts one of the session keys in Box 1, along with a piece of paper with the name ``Yahoo Server" written on it. It locks this box with the user's key.

Why is this piece of paper here? Let us recall that this box is really just an encrypted message, and that the session key is really just a sequence of random bytes. If Box 1 only contained the session key, then the user wouldn't be able to tell whether the response came back from the AS, or whether the decryption was successful. By putting in ``Yahoo Server,'' the user (or more precisely, the user's program) will be able to verify both that the box comes from the AS, and that the decryption was successful.

4. It puts the other session key in a Box 2, along with a piece of paper with the name ``Samit Mehra User'' written on it. It locks this box with the service's key.
5. It returns both boxes to the user.
6. The user unlocks Box 1 with his key, extracting the session key and the paper with ``Yahoo Server'' written on it.
7. The user can't open Box 2 (since it's locked with the service's key). Instead, he puts a piece of paper with the current time written on it in Box 3, and locks it with the session key. He then hands both boxes to the service.
8. The service opens the Box 2 with its own key, extracting the session key and the paper with ``Samit Mehra User'' written on it. It then opens Box 3 with the session key to extract the piece of paper with the current time on it. These items demonstrate the identity of the user.

   The *timestamp* is put in Box 3 to prevent someone else from copying Box 2 and using it to impersonate the user at a later time. Because clocks don't always work in perfect synchrony, a small amount of leeway (about five minutes is typical) is given between the timestamp and the current time. In addition, the service maintains a list of recently sent authenticators, to make sure that they aren't resent in quick order.

   You may wonder how the service is able to open Box 2, if there isn't anyone ``back there'' to type in a password. Well, the service key isn't derived from a password. Instead, it's randomly generated, then stored in a special file called a *service key file*. This file is assumed to be secure, so that no one can copy the file and impersonate the service to a legitimate user.

In Kerberos parlance, Box 2 is called the ***ticket,*** and Box 3 is called the ***authenticator***. The authenticator typically contains more information than what is listed here. Some of this added information arises from the fact that this is an electronic message (for example, there is a checksum). There may also be an encryption key in the authenticator to provide for privacy in future communications between the user and the service.

Service Authentication

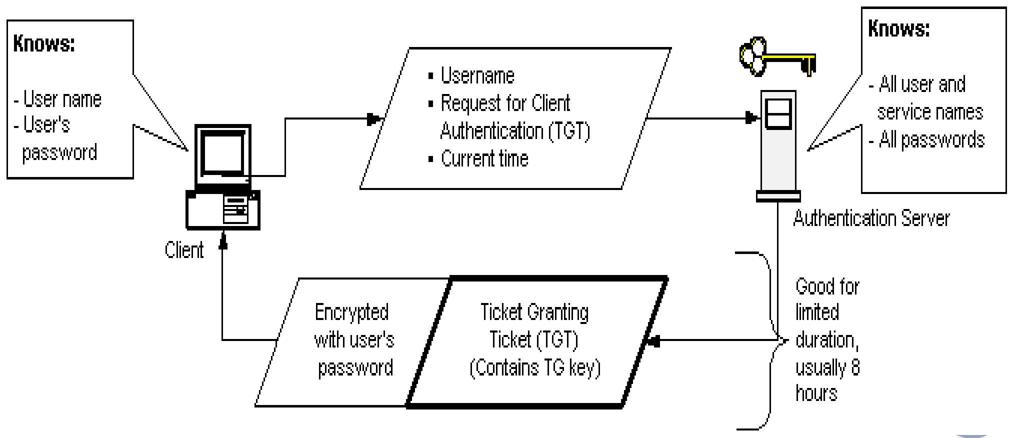
Sometimes, the user may want the service to be authenticated in return. To do so, the service takes the timestamp from the authenticator (Box 3), places it in Box 4, along with a piece of paper with ``Yahoo Server'' written on it, locks it with the session key, and returns it to the user. (Clearly, it must include *something* with the timestamp; otherwise, it could simply return Box 3!)

The Ticket Granting Server

There is a subtle problem with the above exchange. It is used every time a user wants to contact a service. But notice that he then has to enter in a password (unlock Box 1 with the key) each time. The obvious way around this is to cache the key derived from the password. But caching the key is dangerous. With a copy of this key, an attacker could impersonate the user at any time (until the password is next changed).

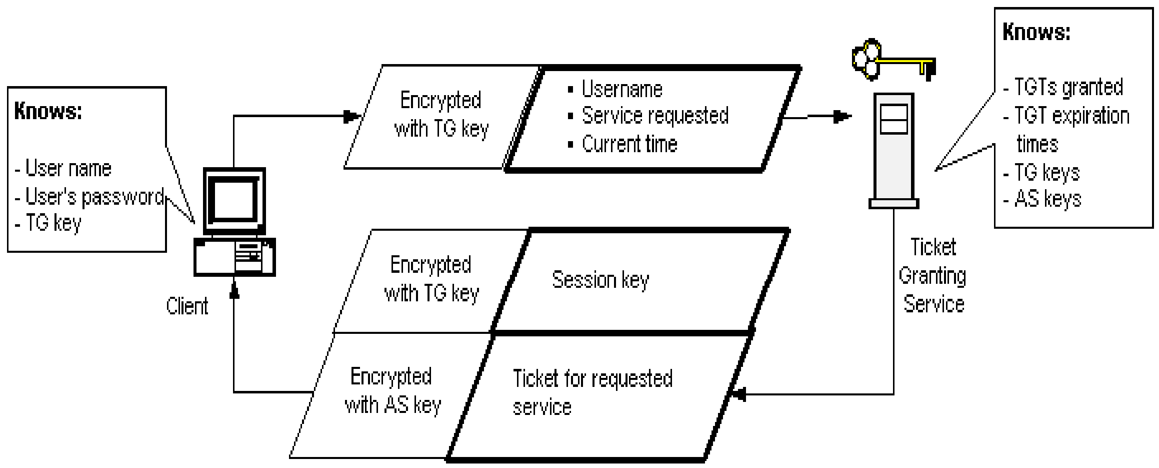
Kerberos resolves this problem by introducing a new agent, called the ***ticket granting server*** **(TGS)**. The TGS is logically distinct from the AS, although they may reside on the same physical machine. (They are often referred to collectively as the KDC--the Key Distribution Center. The function of the TGS is as follows. Before accessing any regular service, the user requests a ticket to contact the TGS, just as if it were any other service. This ticket is called the ***ticket granting ticket*** **(TGT)**.

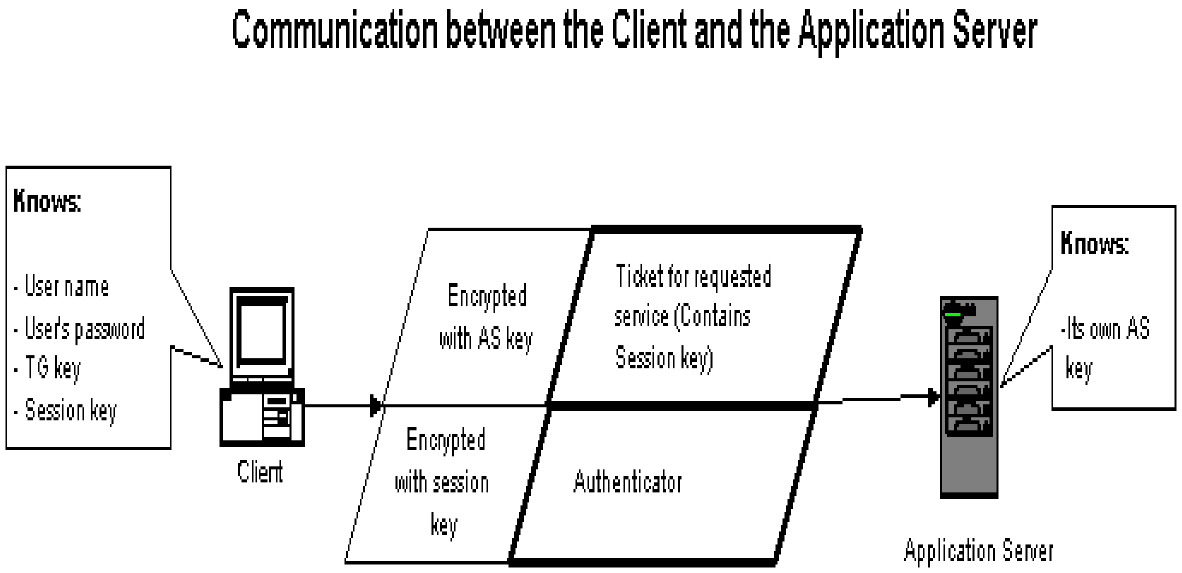
Initial Issuance of Ticket Granting Ticket


Knows:
- User name
- User's password
Client
- Username
- Request for Client Authentication (TGT)
- Current time
Knows:
- All user and service names
- All passwords
Authentication Server
Encrypted with user's password
Ticket Granting Ticket (TGT) (Contains TG key)
Good for limited duration, usually 8 hours

After receiving the TGT, any time that the user wishes to contact a service, he requests a ticket not from the AS, but from the TGS. Furthermore, the reply is encrypted not with the user's secret key, but with the session key that the AS provided for use with the TGS. Inside that reply is the new session key for use with the regular service. The rest of the exchange now continues as described above.

Susequent Requests for Services from the Ticket Granting Service




Communication between the Client and the Application Server
Knows:
- User name
- User's password
- TG key
- Session key
Client
Encrypted with AS key
Ticket for requested service (Contains Session key)
Encrypted with session key
Authenticator
Knows:
-Its own AS key
Application Server

It's sort of like when you visit some workplaces. You show your regular ID to get a guest ID for the workplace. Now, when you want to enter various rooms in the workplace, instead of showing your regular ID over and over again, which might make it vulnerable to being dropped or stolen, you show your guest ID, which is only valid for a short time anyway. If it were stolen, you could get it invalidated and be issued a new one quickly and easily, something that you couldn't do with your regular ID.

The advantage this provides is that while passwords usually remain valid for months at a time, the TGT is good only for a fairly short period, typically eight hours. Afterwards, the TGT is not usable by anyone, including the user or any attacker. This TGT, as well as any tickets that you obtain using it, are stored in the *credentials cache*. There are a number of commands that you can use to manipulate your own credentials cache, which we'll get to in a moment.

Limitations of Kerberos

Kerberos is not effective against password guessing attacks; if a user chooses a poor password, then an attacker guessing that password can impersonate the user. Similarly, Kerberos requires a trusted path through which passwords are entered. If the user enters a password to a program that has already been modified by an attacker (a Trojan horse), or if the path between the user and the initial authentication program can be monitored, then an attacker may obtain sufficient information to impersonate the user. Kerberos can be combined with other techniques, as described later, to address these limitations.

To be useful, Kerberos must be integrated with other parts of the system. It does not protect all messages sent between two computers; it only protects the messages from software that has been written or modified to use it. While it may be used to exchange encryption keys when establishing link encryption and network level security services, this would require changes to the network software of the hosts involved.

Kerberos does not itself provide authorization, but V5 Kerberos passes authorization information generated by other services. In this manner, Kerberos can be used as a base for building separate distributed authorization services.

Conclusion

Authentication is critical for the security of computer systems. Without knowledge of the identity of a principal requesting an operation, it's difficult to decide whether the operation should be allowed.

Traditional authentication methods are not suitable for use in computer networks where attackers monitor network traffic to intercept passwords. The use of strong authentication methods that do not disclose passwords is imperative.

The Kerberos authentication system is well suited for authentication of users in such environments

Reference

- www.google.com
- www.wikipedia.org
- www.studymafia.org