ISSN:0975-9646

Pushkar Bhadle et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (2) , 2014, 1930-1933

Kerberos Authentication System using Public key Encryption

Pushkar Bhadle[1], Sonal Gugale[2], Sakshi Trar[3], Harjot Kaur[4], Shital Salve[5]

[1,2,3,4]*B.E. Student,* [5]*Associate Professor*
Imperial College of Engineering and Research
Pune.

Abstract - **Kerberos is a security System that allows secure communication between computers by preventing third person to steal information sent across wires. The kerberos System Guards Electronic transmission that takes place over a network or internet. To do this it firstly encrypts the information i.e. codes it in such a way that the computer that is going to receive it can only decode or decrypt it. Kerberos also ensures that your password is never sent across wires instead password encrypted with keys are sent. This kerberos is necessary to prevent people from tapping the lines and listening all the information specially passwords. so we use Kerberos System to maintain the integrity and security of our electronic communication taking place over insecure network.**

Keywords - Authentication; Kerberos; Public-Key Cryptography;

I. INTRODUCTION

Kerberos gets its name from ancient mythology. Kerberos which was called Cerberous in Greek mythology was a three headed beast more or less like a dog that guarded the underworld and kept others from entering the dead. Kerberos protocol design started in late 1980's as a part of project Athena at MIT. It is designed for distributed systems and is a secure authentication mechanism which assumes the network is not safe. Because of Kerberos authentication the client and server authenticate each other before establishing a connection. Kerberos version 4 was the first public release which later on lead to the actual version 5 in 1995. It was released after wide public reviews. Its specification are defined in internet RFC 1510 [4] and it followed the IETF standard process. Kerberos was originally designed for UNIX, but now it is available for all major operating systems. Many commercial versions are also present foe use for different purposes that can be freely taken from MIT [6]. The problem that Kerberos deals with is: Assume a client/user wants a service from a server located anywhere in the network, then following three threat exists.

- A user may gain access to a workstation and pretend to be the another user/client form the same workstation.
- A user may even impersonate the other user, change the network address and send the messages to server impersonating the original user.
- A user may keep a watch on the exchanges of information and later use that information to gain entrance to a server. It is called replay attack.

In any of the above cases, the unauthorized user may gain access to services and data that he/she is not authorized for. In general, it is difficult to build a authentication protocol at each server in the world. So MIT thought of an centralized server which authenticates users to servers and servers to users. Kerberos uses symmetric encryption but we will use public key encryption

II. KERBEROS BACKGROUND

We saw the problem that kerberos deals with in previous paragraph. To deal with the problem we would like to authenticate request for service restrict services to authorized users. Unauthorized used should not be entertained by the server[1]. For this we cannot fully trust a workstation to identify its users correctly to network services.

Kerberos version 4 and kerberos version 5 are commonly in use . Version 4 implementation is not yet completed. It's not fully developed. Version 5 corrects all the deficiencies found in version 4 and has been issued as a proposed internet standard[3].

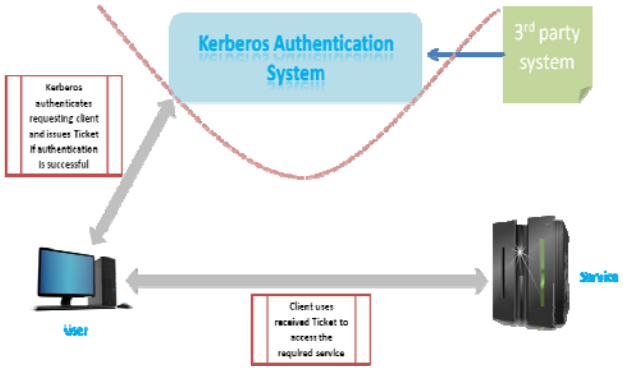

Fig 1 Kerberos as Black Box

III. MOTIVATION

In an environment where each user has its own dedicated personal computer with no network connection , then its data and resources are protected by physically securing each personal computer. But if we use time sharing environment then the time sharing operating system must provide the security. To identify users the operating system can enforce a policy such as username and password logon procedure. It will help provide the security. Today, none of the above scenarios is typical. Most commonly we have distributed architecture which consists of clients and distributed or centralized servers. In this environment

following three approaches to security should be envisioned.

- Rely on each individual workstation to assume the identity of its users and also rely on each server to enforce a policy based on user identification[1].
- Requires the user to prove that the user is authenticated and authorized for each service invoked and also requires the server to prove its identity to the client[1].

In a small environment with few computers as clients and servers first two approaches will work but with open environment the third approach will work. It will help protect the data ,information and resources housed at the server.

IV. EXISTING SYSTEM

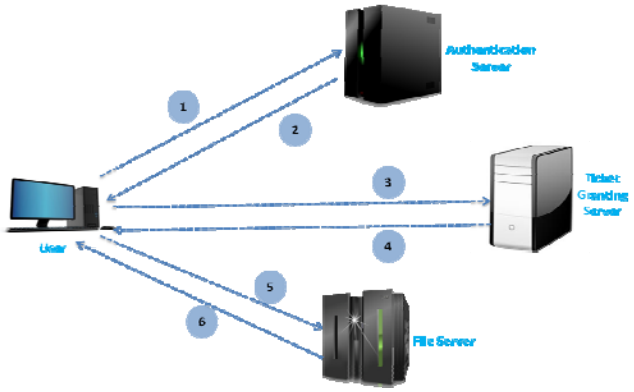

Fig. 2 Existing System Working

AUTHENTICATION DIALOGUE Table summarizes the basic version dialogue.

1. C->AS Options||ID_C||$Realm_c$||ID_{tgs}||times||nonce

2. AS->C
$Realm_c$||ID_C||$Ticket_{tgs}$||$E(K_c,[K_{c,tgs}$||times||Nonce1||$Realm_{tgs}$||$ID_{tgs}]$

$Ticket_{tgs} = E(K_{tgs},[flags||K_{c,tgs}||Realm_c||ID_c||AD_c||Times])$

(a) Authentication Service Exchange to obtain TGT

3. C->TGS Options||ID_v||Times||$Ticket_{tgs}$||$Authenticator_c$

4.TGS->C
$Realm_c$||ID_c||$Ticket_v$||$E(K_{c,tgs}[K_{c,v}$||Times||$Nonce_2$||$Realm_v$||$ID_v]$

$Ticket_{tgs} = E(Ktgs[Flags||Kc,tgs||Realmc||Id_c||AD_c||Times])$

$Ticket_v = E(K_v[Flags||K_{c,v}||Realm_c||ID_c||AD_c||Times])$

$Ticket_v = E(K_v[Flags||K_{c,v}||Realm_c||AD_c||Times])$

$Authenticator_c = E(K_{c,tgs}[IDc||Realm_c||TS_1])$

(b)TGS Exchange to obtain service-granting ticket

5. C->V Options||$Ticket_v$||$Authenticator_c$

6.V->C $E_{Kc,v}$ $[TS_2$||Subkey||Seq!=]

$Ticket_v = E(K_v,[Flag||K_{c,v}||Realm_c||ID_c||AD_c||Times])$

$Authenticator_c = E(K_{c,v}[ID_c||Realm_c||TS_2||Subkey||Seq!=])$

(c) client/server authentication exchange to obtain service

First we will see the **authentication service exchange**. In Message (1), a client is requesting for a ticket-granting ticket(TGT). It includes the ID of the user and the TGS. The following new parameters are added:

- **Realm:** Indicates the realm (working area) of user[6].
- **Options:** Certain flags are set in the returned ticket[6].
- **Times:** Used by the client to do the following time settings in the ticket[6].

—**from**: the desired start time for the requested ticket[6].
—**till**: the requested expiration time for the requested ticket[6].
—**rtime**: requested renew-till time[6].

• **Nonce:** A random value that is repeated in message (2) to assure that the response is fresh and has not been replaced by an opponent or hacker[6]. Message (2) returns a ticket-granting ticket(TGT). The block includes the session key used between the client and the TGS, times specifications, the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. All the above stated flags introduce new functionality to version 5.Let us now compare the **ticket-granting service exchange** for versions 4 and 5.We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition to version 4, version 5 includes requested times and options for the ticket and a nonce. The authenticator itself is same as the one used in version 4. Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key that is now shared by the client and the TGS. Therefore, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required .The authenticator includes several new fields:

- **Sub key:** It is the client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, then the session key from the ticket ( ) is used for further operation.
- **Sequence number:** It is an optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be numbered sequentially to detect replays. If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. In version 4, the timestamp was incremented by one. Incrimination of timestamp is not necessary in version 5, because the message's formats nature is such that it is not possible for an opponent to create message (6) without the knowledge of the appropriate encryption keys. The Subkey field, if present, overrides the Subkey field that is present in message (5).

V. NEW KERBEROS USING PUBLIC KEY ENCRYPTION

A new direction for Kerberos is public key cryptography. Public key cryptography eases key distribution a lot. Using only symmetric cryptography KDC and client must share a key; using asymmetric cryptography the client can present the public key, which can be used to encrypt messages for

it. This is used for email communication by the program Pretty Good Privacy (PGP). The big advantage for Kerberos is that the key distribution centre does not have to save the keys client keys in his database any longer. To obtain a ticket granting ticket, the client has to present his public key. The KDC uses this key to encrypt the ticket and session key[7]. As everybody is able to create a key pair for public key cryptography, additional infrastructure is needed. A trusted certification authority (CA) has to sign every valid public key. The client can present his key which is signed by the trusted authority. Integration in Kerberos is easy due to the fact that only interaction with the authentication service has to be changed to use asymmetric cryptography; everything else can remain as it is. If the client presents his public key, the authentication service checks, whether it has a valid signature from a trusted authority and return a session key afterwards. The client decrypts the session key with the private key of his key pair. Following communication is handled like in Kerberos without public key cryptography support.

Algorithm used: RSA Algorithm

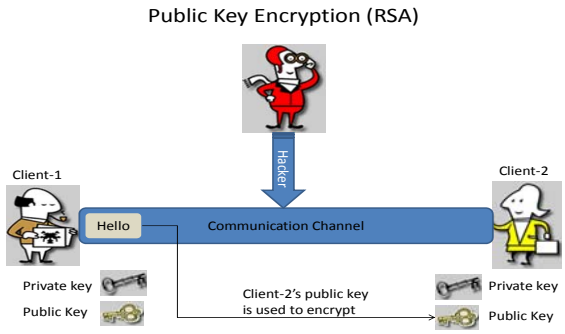

Fig. 3 Public key Encryption using RSA.

The research done by Diffie and Hellman introduced a new approach to cryptography and also challenged cryptologists to come up with a crypto graphical algorithm that met the requirements for public-key systems. Numbers of algorithms have been proposed for public-key cryptography. Some of these provide very successful and secure results. Ron Rivest, Adi Shamir, and Len Adleman were the first three persons who got successful responses from the challenges that were raised in 1977. The Rivest-Shamir-Adleman (RSA) scheme was the most widely accepted and implemented general-purpose approach to public-key encryption. The **RSA** scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n - 1 for some n. The typical size for n is 1024 bits, or 309 decimal digits. That means that n is less than 21024.Now we examine RSA in this section in some detail. Then we examine some of the computational and crypt analytical implications of RSA[1].

Description of the Algorithm

RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, where each block is having a binary value less than some number let say n. The block size must be less than or equal to log2 (n) + 1; in practice, the block size is i bits, where $2^i\ n \leq 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and cipher text block $C[3]$.

$C = M^e \bmod n$

$M = C^d \bmod n = 1M^{ed} \bmod n = M^{ed} \bmod n$

Both sender and receiver must know the value of n. The sender knows the value of e, and the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$. Below are the following requirements that must be met for this algorithm:_

1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$.[5]

2. It is relatively easy to calculate

$M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$.[5]

3. It is infeasible to determine d given e and n.[5]

First, we focus on the first requirement. We need to find a relationship of the form

$$M^{ed} \bmod n = M$$

The preceding relationship holds if e and d are multiplicative inverses modulo $\varphi(n)$, where $\varphi(n)$ is the Euler totient function $\varphi(pq) = (p - 1)(q - 1)$. The relationship between e and d can be expressed as

$$ed \bmod \varphi(n) = 1$$

From this we can say that e and d are multiplicative inverses mod f(n). According to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to f(n). Equivalently, gcd(f(n), d) = 1

Key Generation	
Select p,q	p and q both prime, p!=q
Calculate n = p*q	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n),e)=1;1<e<\phi(n)$
Calculate d	$d = e-1 \pmod{\phi(n)}$
Public key	PU = [e,n]
Private key	PR = [d,n]

Encryption using Public Key	
Plaintext	M < n
Cipher text	$C = M^e \bmod n$

Decryption using Public Key	
Plaintext	C
Cipher text	$M = C^d \bmod n$

VI. CONCLUSION

The Kerberos Authentication protocol is widely used by many big or small scale companies as a trusted authentication protocol. Mutual authentication between the clients and the server is the main advantage of this protocol. The passwords are never sent across the network. The addition of public key in the system would certainly add up the authentication and security level. Kerberos is entirely based on open Internet standards. A number of well-tested and widely-understood reference implementations and RFCs are available free of charge to the Internet community[3][4][5]. Commercial implementations based on the accepted standards are also available.

ACKNOWLEDGEMENT

We express our sincerest and deepest regards to our project guide Prof. Shital Salve for her valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism .We deeply express our sincere thanks to our Head of Department of Computers Prof. Vinod Wadne for encouraging and allowing us to present the project We this thank all our lecturers who have directly or indirectly helped our project.

REFERENCES

[1] *Sufyan T. Faraj Al-Janabi and Mayada Abdul-salam Rasheed,* "Public-Key Cryptography Enabled Kerberos Authentication", IEEE communication magazine, (2011) **.**

[2] *Sufyan Alan H. Harbitter PEC Solutions, Inc.,* "Performance of Public-Key-Enabled Kerberos Authentication in Large Networks", IEEE communication magazine, (2001)

[3] *C. Numen and k.Raeburn*"The kerberos network authentication service(V5)", RFC 4120.

[4] *J. Kohl, and C. Neuman,* "The Kerberos network authentication service (V5)", RFC 1510.

[5] *L. Zhu B. Tung,* "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT),"RFC 4556.

[6] www.mit.edu/kerberos

[7]*P. Leach and k. jaganathan,* "Kerberos Cryptosystem Negotiation Extension", RFC 4537.