ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

A Secure Process –Emergence and Implementation of Kerberos Functionality in a Client/Server

K.Sandhya Rani, B.Santosh Kumar
rashmi.chandrika13@gmail.com

Abstract— ***Kerberos is a computer network authentication protocol which works on the basis of "tickets" to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Its designers aimed primarily at a client–server, and it provides mutual authentication — both the user and the server verify each other's identity. Kerberos protocol messages are protected against eavesdropping and replay attacks. Kerberos builds on symmetric key cryptography and requires a trusted third party, and optionally may use public-key cryptography by utilizing asymmetric key cryptography during certain phases of authentication. With Kerberos, by exchanging time-sensitive tickets, we can make transactions secure without sending passwords in plaintext over the network. For a client program to take advantage of Kerberos, it must be Kerberized, which means that it can obtain tickets from the Kerberos server and negotiate with a Kerberos-aware service. Most programs can be Kerberized, including web browsers, telnet applications, POP email clients, and print utilities. Similarly, services that can be made Kerberos-aware include web sites; printers, file servers, and POP mail servers. Though it's a fairly complex protocol, following are a few basic characteristics: Every user and every service has a password. Only the owner of the password and the Kerberos server know this password. Passwords must remain confidential, as Kerberos provides no inherent protection against those that are stolen. When we use a client program that makes an initial ticket request to the Kerberos server, it will ask you for your Kerberos username and password. The program will then send a ticket request to the Kerberos server. The server will respond by sending you a ticket-granting ticket that it encrypts by plugging your password into an encryption algorithm. Because only you and the Kerberos server know what your password is, only you will be able to decrypt and use the ticket-granting ticket. This ticket-granting ticket normally expires eight hours after it is issued. Once we have a ticket-granting ticket, you may then use Kerberized programs to request services from Kerberos-aware servers. The Kerberized program sends your ticket-granting ticket to a ticket-granting server (usually the Kerberos server itself) with a request to transact with a specific service (e.g., a printer, a POP email server). The server gives you a ticket that lets you conduct a transaction with the service and also ensures that both you and the service are who you say you are. Kerberos gives us the option to encrypt data sent over the network. This means that the entire transaction between you and a Kerberos-aware service will be in unreadable cipher text rather than plaintext.***

Index Terms— **Encryption, decryption, cipher-text, plain-text, servers, clients, tickets, transactions.**

I. INTRODUCTION

MIT (Massachusetts Institute of Technology) developed Kerberos to protect network services provided by Project Athena. The protocol was named after the Greek mythological character Kerberos (or *Cerberus*), known in Greek mythology as being the monstrous three-headed guard dog of Hades. Several versions of the protocol exist; versions 1–3 occurred only internally at MIT. Steve Miller and Clifford Newman, the primary designers of Kerberos version 4, published that version in the late 1980s, although they had targeted it primarily for Project Athena. Version 5, designed by John Kohl and Clifford Newman, appeared as RFC 1510 in 1993 (made obsolete by RFC 4120 in 2005), with the intention of overcoming the limitations and security problems of version 4. MIT makes an implementation of Kerberos freely available, under copyright permissions similar to those used for BSD. In 2007, MIT formed the Kerberos Consortium to foster continued development. Founding sponsors include vendors such as Oracle, Apple Inc., Google, Microsoft, Centrify Corporation and TeamF1 Inc.,, and academic institutions such as KTH-Royal Institute of Technology, Stanford University, MIT and vendors such as Cyber-Safe offering commercially supported versions. Authorities in the United States classified Kerberos as auxiliary military technology and banned its export because it used the DES encryption algorithm (with 56-bit keys).[1] A non-US Kerberos 4 implementation, KTH-KRB developed at the Royal Institute of Technology in Sweden, made the system available outside the US before the US changed its cryptography export regulations (*circa* 2000). The Swedish implementation was based on a limited version called eBones which is based on the exported MIT Bones release (stripped of both the encryption functions and the calls to them) on version Kerberos 4 patch-level 9. Many UNIX and operating systems, including FreeBSD, Apple's Mac OS X, Red Hat Enterprise Linux 4, Oracle's Solaris, IBM's AIX, HP's OpenVMS, and others, include software for Kerberos authentication of users or services. Embedded implementation of the Kerberos V authentication protocol for client agents and network services running on embedded platforms is also available from companies such as TeamF1, Inc.As of 2005, the IETF Kerberos working group is updating the specifications. Recent updates include: Encryption and Checksum Specifications" (RFC (Request for Reference)-3961) Advanced Encryption Standard (AES) Encryption for Kerberos 5 (RFC-3962)[2].

A new edition of the Kerberos V5 specification "The Kerberos Network Authentication Service (V5)" (RFC-4120) this version obsoletes RFC-1510 clarifies aspects of the protocol and intended use in a more detailed and clearer

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

explanation. A new edition of the GSS-API specification "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2." (RFC-4121)Kerberos uses as its basis the symmetric Needham-Schroeder protocol. It makes use of a trusted third party, termed a key distribution center (KDC), which consists of two logically separate parts: an Authentication Server (AS) and a Ticket Granting Server (TGS) [4].

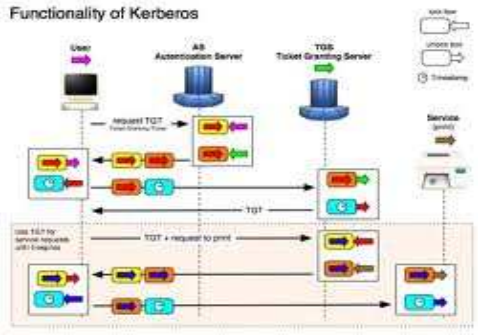

Fig. 1. Functionality of Kerberos

The KDC maintains a database of secret keys; each entity on the network whether a client or a server shares a secret key known only to itself and to the KDC. Knowledge of this key serves to prove an entity's identity. For communication purposes the KDC generates a session key which communicating parties use to encrypt their transmissions.

II. RELATED WORK

A. Kerberos Authentication Work:

We may not know it, but our network is probably unsecured right now. Anyone with the right tools could capture, manipulate, and add data between the connections you maintain with the internet. The security cat and mouse game isn't one sided, however. Network administrators are currently taking advantage of Kerberos to help combat security concerns. Project Athena was initiated in 1983, when it was decided by the Massachusetts Institute of Technology that security in the TCP/IP model just wasn't good enough. A total of 8 long years of research passed before Kerberos, named after the three-headed Greek mythological dog known as Cerberus, was officially complete. The result of MIT's famous research became widely used as default authentication methods in popular operating systems. If you are running Windows 2000 or later, you are indeed running Kerberos by default. Other operating systems such as the Mac OS X also carry the Kerberos protocol. Kerberos isn't just limited to operating systems, however, since it is employed by many of Cisco's routers and switches.

B. Ticket Authentication:

A ticket is something a client presents to an application server to demonstrate the authenticity of its identity. Tickets are issued by the authentication server and are encrypted using the secret key of the service they are intended for. Since this key is a secret shared only between the authentication server and the server providing the service, not even the client which requested the ticket can know it or change its contents. The main information contained in a ticket includes:

- The requesting user's principal (generally the username);
- The principal of the service it is intended for;
- The IP address of the client machine from which the ticket can be used. In Kerberos 5 this field is optional and may also be multiple in order to be able to run clients under NAT or multihomed.
- The date and time (in timestamp format) when the tickets validity commences;
- The ticket's maximum lifetime
- The session key (this has a fundamental role which is described below);

Each ticket has expiration 10 hours is essential since the authentication server no longer has any control over an already issued ticket. Even though the realm administrator can prevent the issuing of new tickets for a certain user at any time, it cannot prevent users from using the tickets they already possess. This is the reason for limiting the lifetime of the tickets in order to limit any abuse over time. Tickets contain a lot of other information and flags which characterize their behavior, but we won't go into that here. We'll discuss tickets and flags again after seeing how the authentication system works.

C. Encryption in Kerberos:

As you can see Kerberos often needs to encrypt and decrypt the messages (tickets and authenticators) passing between the various participants in the authentication. It is important to note that Kerberos uses only symmetrical key encryption (in other words the same key is used to encrypt and decrypt). Certain projects are active for introducing a public key system in order to obtain the initial user authentication through the presentation of a private key corresponding to a certified public key, but since there is no standard we'll skip this discussion for now.

1. Encryption Type in Kerberos:

Kerberos 4 implements a single type of encryption which is DES at 56 bits. The weakness of this encryption plus other protocol vulnerabilities has made Kerberos 4 obsolete. Version 5 of Kerberos, however, does not predetermine the number or type of encryption methodologies supported. It is the task of each specific implementation to support and best negotiate the various types of encryption. However, this flexibility and expandability of the protocol has accentuated interoperability problems between the various implementations of Kerberos 5. In order for clients and application and authentication servers using different implementations to interoperate, they must have at least one encryption type in common. The difficulty related to the interoperability between UNIX implementations of Kerberos 5 and the one present in the Active Directory of Windows is a classic example of this. Indeed, Windows Active Directory supports a limited number of encryptions and only had DES at 56 bits in common with UNIX. This required keeping the latter enabled, despite the risks being well known, if interoperability had to be guaranteed. The problem was

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

subsequently solved with version 1.3 of MIT Kerberos 5. This version introduced RC4-HMAC support, which is also present in Windows and is more secure than DES. Among the supported encryptions (but not by Windows) the triple DES (3DES) and newer AES128 and AES256 are worth mentioning.

2. *Encryption Key in Kerberos:*

As stated above, one of the aims of the Kerberos protocol is to prevent the user's password from being stored in its unencrypted form, even in the authentication server database. Considering that each encryption algorithm uses its own key length, it is clear that, if the user is not to be forced to use a different password of a fixed size for each encryption method supported, the encryption keys cannot be the passwords. For these reasons the string2key function has been introduced, which transforms an unencrypted password into an encryption key suitable for the type of encryption to be used. This function is called each time a user changes password or enters it for authentication. The string2key is called a hash function, meaning that it is irreversible: given that an encryption key cannot determine the password which generated it unless by brute force. Famous hashing algorithms are MD5 and CRC32.

III. IMPLEMENTATION OF KERBEROS

Kerberos operates describing each of the packets which go between the client and KDC and between client and application server during authentication. At this point, it is important to underline that an application server never communicates directly with the Key Distribution Center: the service tickets, even if packeted by TGS, reach the service only through the client wishing to access them. The messages we will discuss are listed below (see also the figure below):

- *AS_REQ* is the initial user authentication request is made with kinit this message is directed to the KDC component known as Authentication Server (AS).
- *AS_REP* is the reply of the Authentication Server to the previous request. Basically it contains the TGT (encrypted using the TGS secret key) and the session key encrypted using the secret key of the requesting user.
- *TGS_REQ* is the request from the client to the Ticket Granting Server (TGS) for a service ticket. This packet includes the TGT obtained from the previous message and an authenticator generated by the client and encrypted with the session key.
- *TGS_REP* is the reply of the Ticket Granting Server to the previous request. Located inside is the requested service ticket (encrypted with the secret key of the service) and a service session key generated by TGS and encrypted using the previous session key generated by the AS.
- *AP_REQ* is the request that the client sends to an application server to access a service. The components are the service ticket obtained from TGS with the previous reply and an authenticator again generated by the client, but this time encrypted using the service session key (generated by TGS);
- *AP_REP* is the reply that the application server gives to the client to prove it really is the server the client is expecting. This packet is not always requested. The client requests the server for it only when mutual authentication is necessary.

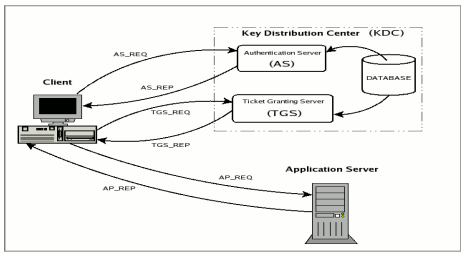

Fig. 2. Application process of Kerberos

Now each of the previous phases is described in greater detail with reference to Kerberos 5, but pointing out the differences with version 4. Nevertheless, it should be borne in mind that the Kerberos protocol is rather complicated and this document is not intended as a guide for those who wish to know the exact operating details (in any case, these are already written up in RFC1510). The discussion below has been left intentionally abstract, but sufficient for those who examine the KDC logs to understand the various authentication transitions and any problems which occur. Note: the subsequent paragraphs enclose unencrypted data in round brackets (), and encrypted data in curly brackets {}: (x, y, z) means that x, y, z are unencrypted; { x, y, z }K indicates that x, y, z are encrypted all together using the symmetrical key K. It is also important to note that the order in which the components are listed in a packet has nothing to do with the real order found in the actual messages (UDP or TCP). This discussion is very abstract. Should you wish further details, please refer to RFC1510 having a good background on the descriptive protocol ASN.1.

A. *Authentication Request for Kerberos:*

In this phase, known as the initial authentication request, the client (kinit) asks the KDC (more specifically the AS) for a Ticket Granting Ticket. The request is completely unencrypted and looks like this:

AS_REQ = ($Principal_{Client}$, $Principal_{Service}$, IP_list , Lifetime) where: $Principal_{Client}$ is the principal associated with the user seeking authentication (e.g. pippo@EXAMPLE.COM); $Principal_{Service}$ is the principal associated to the service the ticket is being asked for and thus is the string "krbtgt/REALM@REALM" (see note*); IP_list is a list of IP addresses that indicate the host where it is possible to use the ticket which will be issued (see note **); Lifetime is the maximum validity time (requested) for the ticket to be issued.

B. *Authentication Reply for Kerberos:*

When the previous request arrives, the AS checks whether $Principal_{Client}$ and $Principal_{Service}$ exist in the KDC database: if at least one of the two does not exist an error message is sent

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

to the client, otherwise the Authentication Server processes the reply as follows:

- It randomly creates a session key which will be the secret shared between the client and the TGS. Let's say SK_{TGS};
- It creates the Ticket Granting Ticket putting it inside the requesting user's principal, the service principal(it is generally krbtgt /REALM@REALM, but read the note* for the previous paragraph), the IP address list (these first three pieces of information are copied as they arrive by the AS_REQ packet), date and time (of the KDC) in timestamp format, lifetime (see note*) and lastly the session key. SK_{TGS}; the Ticket Granting Ticket thus appears as follows: TGT=($Principal_{Client}$, krbtgt/REALM@REALM, IP_list, Timestamp , Lifetime , SK_{TGS})
- It generates and sends the reply containing: the ticket created previously, encrypted using the secret key for the service (let's call it K_{TGS}); the service principal, timestamp, lifetime and session key all encrypted using the secret key for the user requesting the service (let's call it K_{User}). In summary:
- AS_REP = {$Principal_{Service}$, Timestamp, Lifetime, SK_{TGS}} KUser {TGT} KTGS.

It may seem that this message contains redundant information ($Principal_{Service}$, timestamp, lifetime and session key). But this is not the case: since the information present in the TGT is encrypted using the secret key for the server, it cannot be read by the client and needs to be repeated. At this point, when the client receives the reply message, it will ask the user to enter the password. The salt is concatenated with the password and then the string2key function is applied: with the resulting key an attempt is made to decrypt the part of the message encrypted by the KDC using the secret key of the user stored in the database. If the user is really who he/she says, and has thus entered the correct password, the decrypting operation will be successful and thus the session key can be extracted and with the TGT (which remains encrypted) stored in the user's credential cache.

C. *Ticket Granting Request for Kerberos:*

At this point, the user who has already proved to be who he/she says (thus in his/her credential cache there is a TGT and session key SK_{TGS} and wants to access the service but does not yet have a suitable ticket, sends a request (TGS_REQ) to the Ticket Granting Service constructing it as follows:

- Create an authenticator with the user principal, client machine timestamp and encrypt everything with the session key shared with the TGS, i.e.: Authenticator = { $Principal_{Client}$, Timestamp }SK_{TGS}
- Create a request packet containing: the service principal for which the ticket is needed and lifetime encrypted; the Ticket Granting Ticket which is already encrypted with the key of the TGS; and the authenticator just created. In summary: TGS_REQ = ($Principal_{Service}$, Lifetime , Authenticator) { TGT }K_{TGS} .

D. *Ticket Granting Reply for Kerberos:*

When the previous request arrives, the TGS first verifies that the principal of the requested service ($Principal_{Service}$) exists in the KDC database: If it exists, it opens the TGT using the key for krbtgt/REAM@REALM and extracts the session key (SK_{TGS})

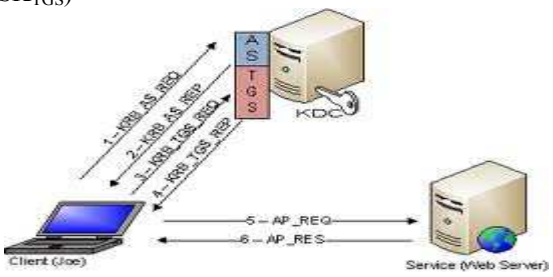

Fig. 3. Client-servers Process

Which it uses to decrypt the authenticator. For the service ticket to be issued it checks that the following conditions have a positive outcome:

- TGT has not expired;
- The $Principal_{Client}$ present in the authenticator matches the one present in the TGT;
- The authenticator is not present in the replay cache and has not expired;
- If IP_list is not null it checks that the source IP address of the request packet (TGS_REQ) is one of those contained in the list; The previously checked conditions prove that the TGT really belongs to the user who made the request and therefore the TGS starts to process the reply as follows:
- It randomly creates a session key which will be the secret shared between the client and the service. Let's say $SK_{Service}$;
- It creates the service ticket, putting inside the requesting user's principal, the service principal, the list of IP addresses, the date and time (of the KDC) in timestamp format, the lifetime (as the minimum between the lifetime of the TGT and that associated with the service principal) and lastly the session key $SK_{Service}$. Known as $T_{Service}$ the new ticket is: $T_{Service}$ = ($Principal_{Client}$, $Principal_{Service}$, IP_list , Timestamp , Lifetime , $SK_{Service}$)
- It sends the reply message containing: the previously created ticket encrypted using the service secret key (let's call it $K_{Service}$); the service principal, timestamp, lifetime and new session key all encrypted using the session key extracted from TGT. In summary:TGS_REP = { $Principal_{Service}$, Timestamp , Lifetime , $SK_{Service}$ }SK_{TGS} { $T_{Service}$ }$K_{Service}$

When the client receives the reply, having in the credential cache the session key SK_{TGS}, it can decrypt the part of the message containing the other session key and memorize it

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

together with the service ticket $T_{Service}$ which, however, remains encrypted.

E. Application Authentication Request:

The client, having the credentials to access the service (i.e. the ticket and related session key), can ask the application server for access to the resource via an AP_REQ message. It should be borne in mind that, unlike the previous messages where the KDC was involved, the AP_REQ is not standard, but varies depending on the application. Thus, the application programmer has the job of establishing the strategy with which the client will use its credentials to prove its identity to the server. However, we can consider the following strategy by way of example:

- The client creates an authenticator containing the user principal and timestamp and encrypts everything with the session key $SK_{Service}$ that it shares with the application server, i.e.: Authenticator = { $Principal_{Client}$, Timestamp }$SK_{Service}$.
- It creates a request packet containing the service ticket $T_{Service}$ which is encrypted with its secret key and the authenticator just created. In summary: AP_REQ = Authenticator { $T_{Service}$ }$K_{Service}$ When the previous request arrives, the application server opens the ticket using the secret key for the requested service and extracts the session key $SK_{Service}$ which it uses to decrypt the authenticator. To establish that the requesting user is authentic and thus grant access to the service, the server verifies the following conditions:
- ticket has not expired;
- The $Principal_{Client}$ present in the authenticator matches the one present in the ticket;
- The authenticator is not present in the replay cache and has not expired;
- If `IP_list` (extracted from the ticket) is not null it checks that the source IP address of the request packet (AP_REQ) is one of those contained in the list;

1) *Pre-Authentication:*

As seen in the description of the Authentication Server Reply (AS_REP), before distributing a ticket the KDC simply checks that the principal of the requesting user and service provider exist in the database. Then, particularly if it involves a request for a TGT, it is even easier, because krbtgt/REALM@REALM certainly exists and thus it is sufficient to know that a user's principal exists to be able to obtain a TGT with a simple initial authentication request. Obviously, this TGT, if the request comes from an illegitimate user, cannot be used because they do not know the password and cannot obtain the session key for creating a valid authenticator. However, this ticket, obtained in such a easy way can undergo a brute-force attack in an attempt to guess the long-term key for the service the ticket is intended for. Obviously, guessing the secret of a service is not any easy thing even with current processing powers, however, with Kerberos 5, a pre-authentication concept has been introduced to reinforce security. Thus if the KDC policies (configurable) request pre-authentication for an initial client request, the Authentication Server replies with an error packet indicating the need to pre-authenticate. The client, in light of the error, asks the user to enter the password and resubmit the request but this time adding the timestamp encrypted with the user long term key, which, as we know, is obtained by applying the string2key to the unencrypted password after having added the salt, if there is one. This time, the KDC, since it knows the secret key of the user, attempts to decrypt the timestamp present in the request and if it is successful and the timestamp is in line, i.e. included within the established tolerance, it decides that the requesting user is authentic and the authentication process continues normally.
It is important to note that pre-authentication is a KDC policy and thus the protocol does not necessarily require it. In terms of implementation, MIT Kerberos 5 and Heimdal have pre-authentication disabled by default, while Kerberos within Windows Active Directory and the AFS kaserver (which is a pre-authenticated Kerberos 4) request it.

2) *Cross Authentication :*

We have already mentioned the possibility for a user belonging to a certain realm to authenticate and access the services of another realm. This characteristic known as cross-authentication is based on the assumption that there is a trust relationship between the realms involved. This may be mono-directional, meaning that the users of realm a can access the services of realm B but not vice versa, or bi-directional, where, as one might expect, the opposite is also possible. In the following paragraphs we will look at cross authentication, breaking down the trust relationships into direct, transitive and hierarchical.

F. Direct Trust Relationships:

This type of trust relationship is elementary and is the basis of cross-authentication and is used to construct the other two types of relationships we will look at later. It occurs when the KDC of realm B has direct trust in the KDC of realm A, thus allowing the users of the latter realm to access its resources. From a practical point of view, a direct trust relationship is obtained by having the two involved KDCs share a key (the keys become two if a bi-directional trust is desired). To do this the concept of a remote Ticket Granting Ticket is introduced which, in the example of the two realms A and B, assumes the form krbtgt/B@A and is added to both the KDCs with the same key. This key is the secret which will guarantee the trust between the two realms. Obviously, to make it bi-directional (i.e. that A also trusts B), it is necessary to create the remote TGT krbtgt/A@B in both KDCs, associating them with another secret key.

As we'll see shortly in the following example, the introduction of the remote TGTs makes cross authentication a natural generalization of normal intra-realm authentication: this underlines that the previous description of Kerberos operation continues to be valid as long as it is accepted that the TGS of one realm can validate the remote TGTs issued by the TGS of another. Note the formal anomaly arising when the remote TGTs are not issued by the AS, as happens for the local ones, but by the local Ticket Granting Server upon

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

presentation of the local TGT. Now let's look at an example to clarify all this. Let's suppose that the user pippo of the realm EXAMPLE.COM, whose associated principal is pippo@EXAMPLE.COM, wishes to access the pluto.test.com server belonging to the TEST.COM.

- If Pippo does not already have a TGT in the realm EXAMPLE.COM he makes an initial authentication request (kinit). Obviously, the reply comes from the AS of his realm;
- He gives the ssh pippo@pluto.test.com command which should open the remote shell on pluto.test.com without having to re-enter the password;
- the ssh client makes two queries to DNS: it works out the IP of pluto.test.com and on the just obtained address carries out the reverse in order to obtain the hostname (FQDN) in canonical form (in this case it coincides with pluto.test.com);
- ssh client then realizes, thanks to the previous result, that the destination does not belong to the user's realm and thus asks the TGS of the realm EXAMPLE.COM (note that it asks the TGS of its realm for this) for the remote TGT krbtgt/TEST.COM@EXAMPLE.COM;
- With the remote TGT it asks the TGS of the realm TEST.COM for the host/pluto.test.com@TEST.COM service ticket;
- When the TEST.COM Ticket Granting Service receives the request, it checks for the existence of the principal krbtgt/TEST.COM@EXAMPLE.COM in its database with which it can verify the trust relationship. If this verification is positive the service ticket (encrypted with the key of host/pluto.test.com@TEST.COM) is finally issued which pippo will send to the host pluto.test.com to obtain the remote shell.

G. Transitive Trust Relationship:

When the number of realms in which cross-authentication must be possible increases, the number of keys to exchange increases quadratically. For example, if there are 5 realms and the relationships must be bi-directional, the administrators must generate 20 keys (double the combinations of 5 elements by 2 by 2). To get around this problem, Kerberos 5 has introduced transitivity in the trust relationship: if realm a trusts realm B and realm B trusts realm C then A will automatically trust C. This relationship property drastically reduces the number of keys (even if the number of authentication passages increases).However, there is still a problem: the clients cannot guess the authentication path (capath) if it is not direct. So they must be informed of the correct path by creating a special stanza ([capaths]) in the configuration of each of the clients. These paths must also be known to the KDCs which will use them to check the transits.

H. Hierarchical Trust Relationship:

If, within organizations, the convention of naming realms with the name of DNS domains in upper case letters is used (highly recommended choice) and if the latter belong to a hierarchy, then Kerberos 5 will support adjacent realms (hierarchically) having a trust relationship (naturally this assumed trust must be supported by the presence of appropriate keys) and will automatically construct (without the need for capaths) the transitive authentication paths. However, administrators can alter this automatic mechanism (for example for reasons of efficiency) by forcing the capaths in the client configuration.

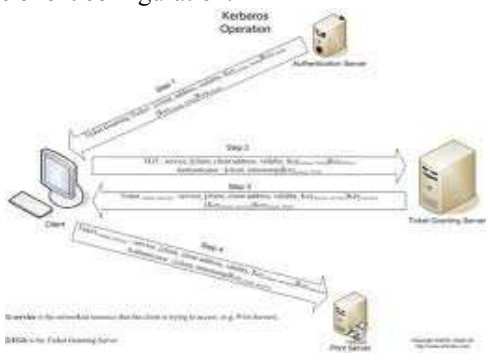

Fig. 4. Operations of Kerberos

Kerberos operates by encrypting data with a symmetric key. A symmetric key is a type of authentication where both the client and server agree to use a single encryption/decryption key for sending or receiving data. When working with the encryption key, the details are actually sent to a key distribution center, or KDC, instead of sending the details directly between each computer. The entire process takes a total of eight steps, as shown below.

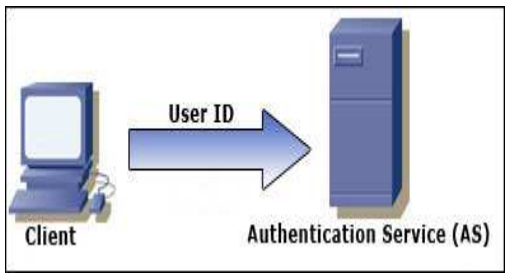

Fig .5. Client server authentication

1) The **authentication service**, or AS, receives the request by the client and verifies that the client is indeed the computer it claims to be. This is usually just a simple database lookup of the user's ID.
2) Upon verification, a **timestamp** is created. This puts the current time in a user session, along with an expiration date. The default expiration date of a timestamp is 8 hours. The encryption key is then created. The timestamp ensures that when 8 hours is up, the encryption key is useless. (This is used to make sure a hacker doesn't intercept the data, and try to crack the key. Almost all keys are able to be cracked, but it will take a lot longer than 8 hours to do so)

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

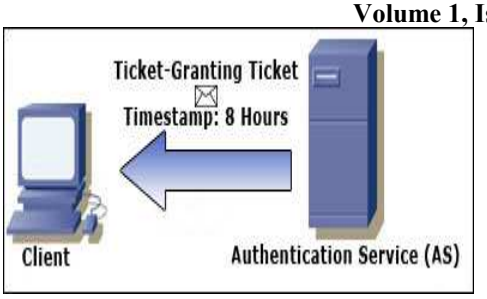

Fig.6.Client Authentication Service System

3) The key is sent back to the client in the form of a **ticket-granting ticket**, or TGT. This is a simple ticket that is issued by the authentication service. It is used for authenticating the client for future reference.

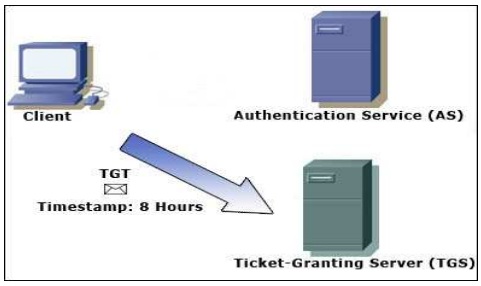

Fig.7. Client server ticket granting Authentication

4) The client submits the ticket-granting ticket to the **ticket-granting server**, or TGS, to get authenticated.

Fig.8. Client Server Ticket Granting

5) The TGS creates an encrypted key with a timestamp, and grants the client a service ticket.

6) The client decrypts the ticket, tells the TGS it has done so, and then sends its own encrypted key to the service.

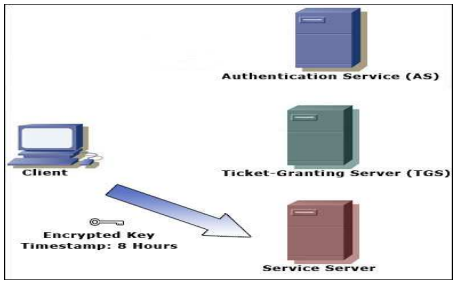

Fig.9. Encrypted Authentications

7) The service decrypts the key, and makes sure the timestamp is still valid. If it is, the service contacts the key distribution center to receive a session that is returned to the client

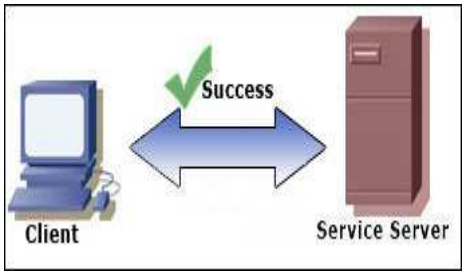

Fig .10. Client service server authentications

8) The client decrypts the ticket. If the keys are still valid, communication is initiated between client and server Is all that back-and-forth communication really necessary? When concerning speed and reliability, it is entirely necessary. After the communication is made between the client and server, no further need of transmitting logon information is needed. The client is authenticated until the session expires.

I. Some More Authentications:

The authentication method described above seems a little one-sided. Kerberos provides support for mutual authentication, for a more secure protection against man in the middle attacks. Remember how the client no longer needs to send logon information after the authentication takes place? Well it sure would ruin everything if a hacker just intercepted our communication to the server and pretended to be us. This type of authentication is fairly easy to understand, since it only involves two systems.

- The first system creates a challenge code made up of random numbers.
- This code is sent to the second system, which generates a response to the received code. This response and a challenge code of its own are then sent back to the first system.
- The first system verifies the response of the second system, and then sends a response to the challenge code it received.
- When the second system receives the response, it is verified. If all is well, it notifies the first system that they are indeed mutually authenticated.

IV. REQUIREMENTS OF KERBEROS

Communicating over a non-secure network to prove their identity to one another in a secure manner. Although it is a widely used protocol, it has the following drawbacks: Kerberos requires continuous availability of a central server. When the Kerberos server is down, no one can log in. This can be mitigated by using multiple Kerberos servers and fallback authentication mechanisms. Kerberos requires the clocks of the involved hosts to be synchronized. The tickets have a time availability period and if the host clock is not synchronized with the Kerberos server clock, the authentication will fail. The default configuration requires that clock times are no more than 10 minutes apart. In practice

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

Network Time Protocol daemons are usually used to keep the host clocks synchronized. The administration protocol is not standardized and differs between server implementations. Password changes are described in RFC 3244. Since the secret keys for all users are stored on the central server, a compromise of that server will compromise all users' secret keys. A compromised client will compromise the user's password

A. Dynamic Password with Kerberos:

In the original Kerberos system, the reasons caused the security problems mentioned as the above, mainly include the following: In the user authentication stage, user's identification is only on the basis of his static password, and there are not mutual authentication between users and AS .In the service access stage, the communications between client and application server are lack of authentication mechanism. So, based on the thought of dynamic password, the improved Kerberos scheme can be drawn up. During the user initial authentication, the user authentication is not simply based on his password, but a big random number (called as salt) is used together to generate a security password, thus can prevent the Password Guessing. And using exchanging random number between the client and the server, the mutual authentication can be achieved. In addition, in the communication period between the client and application server, a connection authentication factor can be used to prevent the replay attacks. Dynamic Password (DP) is also called One Time Password (OTP), which is used to solve the traditional problems which appear when the static Password authentication cannot cope with eavesdropping and replaying, forging, guessing, etc. By means of DP, uncertainties will be considered in authentication information during the process of lodging

B. The feasibility analysis of the improved scheme:

The improved scheme keeps the authentication main structure of the original one; it only adds some security protection fields in the data structure of the first, second and the fifth step. All added fields do not change the process of the original scheme, so the improved scheme is feasible. Comparing to the original scheme, the amounts of computing have increased in the improved scheme, which are shown in Table 1. But in practice, it is very fast to do Hash, thus cannot cause an obvious decrease in authentication

Data Encrypted - The Data Encryption Standard (DES) is widely recognized as weak. The krb5-1.7 release contains measures to encourage sites to migrate away from using single-DES cryptosystems. Among these is a configuration variable that enables "weak" encrypts, which now defaults to "false" beginning with krb5-1.8. This is primarily a bug-fix release – major changes in 1.9.2.

- Improve KDC performance by fully its disabling replay cache.
- Fix MITKRB5-SA-2011-006 KDC denial of service vulnerabilities [CVE-2011-1527 CVE-2011-1528 CVE-2011-1529]. This is primarily a bug-fix release. Fix vulnerabilities – major changes in 1.9.1.
- Kpropd denial of service [MITKRB5-SA-2011-001 CVE-2010-4022].
- KDC denial of service attacks [MITKRB5-SA-2011-002 CVE-2011-0281 CVE-2011-0282 CVE-2011-0283] .
- KDC double-free when PKINIT enabled [MITKRB5-SA-2011-003CVE-2011-0284] .
- kadmind frees invalid pointer [MITKRB5-SA-2011-004 CVE-2011-0285]
- Interoperability:
- Don't reject AP-REQ messages if their PAC doesn't validate; suppress the PAC instead.
- Correctly validate HMAC-MD5 checksums that use DES keys Code quality – *1.9.*
- Fix MITKRB5-SA-2010-007 checksum vulnerabilities (CVE-2010-1324 and others).
- Add a Python-based testing framework.
- Perform DAL cleanup.
- Developer experience.
- Add NSS crypto back end.
- Improve PRNG modularity.
- Add a Fortuna-like PRNG back end. Performance.
- Account lockout performance improvements -- allow disabling of some account lockout functionality to reduce the number of write operations to the database during authentication.
- Add support for multiple KDC worker processes.
- Administrator experience
- Add Trace logging support to ease the diagnosis of configuration problems.
- Add support for purging old keys (e.g. from "cpw -randkey -keepold").
- Add plugin interface for password sync -- based on proposed patches by Russ Allbery that support his krb5-sync package.
- Add plugin interface for password quality checks -- enables pluggable password quality checks similar to Russ Allbery's krb5-strength package.
- Add a configuration file validator script.
- Add KDC support for Secured pre authentication -- this is the old SAM-2 protocol, implemented to support existing deployments, not the in-progress FAST-OTP work.
- Add "cheat" capability for kinit when running on a KDC host Protocol evolution.
- Add support for IAKERB -- a mechanism for tunneling Kerberos KDC transactions over GSS-API, enabling clients to authenticate to services even when the clients cannot directly reach the KDC that serves the services.
- Add support for Camellia encryption (experimental; disabled by default).
- Add GSS-API support for implementers of the SASL GS2 bridge mechanism.

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 1,January 2012

V. CONCLUSION

Kerberos has a couple of main flaws that system administrators need to take into account. First and foremost is the need of the Kerberos server. This server will handle all the functions required for authentication. If this server goes down, no one can get authenticated, and thus- the network is down. A total network crash can be prevented by using more than one Kerberos server, but that is more costly than some people would like to think. Next, we have the issue of clock synchronization. Since Kerberos uses timestamps to handle all activity, the clocks on all host machines must be within 10 minutes of the Kerberos server's clock. Since not all clocks are perfect, the host clock and server clock will eventually be misaligned enough to cause a failure. This can usually be remedied by keep clocks up to date, or use a Network Time Protocol, or NTP. Kerberos isn't the only encryption protocol available. There are multiple ways to encrypt data, and this holds true for many types of different applications. Email encryption protocols, for example, are a breed all of their own. With a product that has been researched and developed for over 8 years, it is generally expected that the product should be well polished. Kerberos doesn't fail to deliver, and this can be seen by looking at all the vendors who use it. Cisco, Microsoft, Apple, and many others rely on this faithful three-headed dog for network security. As Greek mythology goes, you could get around Cerberus by gently lulling him to sleep with honey cakes. Rest assured it will take a lot more than that to get past the famous Kerberos security.

REFERENCES

[1] "ISO/IEC 9594: The Directory."X.500 Blue Book Recommendations. ISO, 1991.

[2] Albitz, P. and Liu, C. DNS and BIND. 4th ed., Sebastopol, CA., O'Reilly & Associates, 2001.

[3] Garfinkel, S. and Spafford, G. Practical UNIX & Internet Security. Sebastopol, CA, O'Reilly & Associates, 1991.

[4] Larson, M. and Liu, C. DNS on Windows 2000. Sebastopol, CA, O'Reilly & Associates, 2001.

[5] Mann, S. and Mitchell, E.L. Linux System Security. Prentice Hall PTR, 1999.

[6] Miller, S.P., Neuman, C., Schiller, J.I., and Saltzer, J.H. "Kerberos Authentication and Authorization System." Project Athena Technical Plan, Section E.2.1, Massachusetts Institute of Technology, October, 1988.

[7] Needham, R.M. and Schroder, M.D. "Using Encryption for Authentication in Large Networks of Computers." Communications of the ACM, 21(12) 1978, 993-999.

[8] Toxen, B. Real World Linux Security. 2nd ed., Prentice Hall PTR, 2002.

[9] SDK Team. "Microsoft Kerberos (Windows)". MSDN Library.http://msdn.microsoft.com/en-us/library/aa378747(VS.85).aspx.

[10] B. Clifford Neuman and Theodore Ts'o (September 1994). "Kerberos: An Authentication Service for Computer Networks". IEEE Communications **32** (9): 33–8. doi:10.1109/35.312841.http://gost.isi.edu/publications/kerberos-neuman-tso.html.

[11] "Domain Name System (DNS) Center Knowledge Base Articles" at http://www.microsoft.com/windows2000/technologies/communications/dns/dnskbs.asp.

[12] "How to migrate an existing DNS infrastructure from a BIND-based server to a Windows Server 2003-based DNS" at http://support.microsoft.com/default.aspx?scid=kb;en-us;323419.

[13] Links to technologies documented in the Platform SKD at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdkintro/sdkintro/contents_of_the_platform_sdk.asp.

[14] "Restricting Active Directory replication traffic to a specific port," a Microsoft Knowledge Base article, at http://support.microsoft.com/default.aspx?scid=224196.

[15] "Public Key Infrastructure for Windows Server 2003" at http://go.microsoft.com/fwlink/?LinkId=19936.

K.Sandhya Rani B.Tech IT from Jayamukhi Institute of Science & Technology M.Tech Software Engineering from CVSR College of Engineering currently working as Asst Prof at Samskruti College of Engineering & Technology having five years of experience in academic has guided many UG students. Her research areas include Databases Web Technology Multimedia Application Development.

B.Santosh Kumar M.Tech Computer Science & Engineering from Hyderabad Institute of Science & Technology having several years of experience in Academic has guided many UG& PG students. Currently he is working as Asst Prof at Samskruti College of Engineering, his areas of interest include Unix Operating System, and Object oriented Analysis & Design, Distributed Databases.