Installing and configuring Kerberos

Koncz Szabolcs
Szabó Csilla

Dept. of Technology of Information
Petru Maior University, Tg. Mures, Romania

2010

Abstract

Kerberos negotiates authenticated and optionally encrypted communications between two points anywhere on the Internet, providing a layer of security that is not dependent on which side of a firewall either client is on. Since studies have shown that half of the computer security breaches in industry happen from inside firewalls, Kerberos V5 from MIT will play a vital role in the security of our network.

In this paper we will present the Kerberos authentication mechanism, we will describe how can be installed and configured the newest available release, Kerberos V5 release 1.7 on a UNIX system, and also how can be configured the Secure Shell to work with Kerberos to ensure secure and reliable communication over the network.

Table of Contents

Introduction

In the real world, identification is something we, as human beings, do naturally, through physical appearance or voice patterns. It is based on the assumption that those attributes are unique, and that they can be trusted. This ability provides us with the possibility to distinguish one person from another.

However, when put in a situation where we're not able to use those attributes to identify someone, as in a phone call for example, we're left with finding some other means to prove our identities. We sometimes identify ourselves with what is called a "shared secret", where one party asks the other party to prove his identity through information that is known only by them, like a password. When we add a computer to this mechanism, with an identification that needs to be provided over a network, things are going a little more complex. Sending this "shared secret", or password, over an unsecured network can be compared to shouting it in a crowded room.

Many authentication mechanisms were developed during the last decade to solve those problems, Kerberos being one of them. In Greek mythology, Kerberos is the three-headed dog that guards the entrance to the underworld, but in the computing world is a network authentication protocol designed at MIT to provide strong authentication for client - server applications by using secret-key cryptography.

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. It differs from asymmetric systems, where a public key is known by virtually everybody, while the private key remains secret, and it is stored on the server.

Kerberos aims

Before describing the elements that make up the Kerberos authentication system and looking at its operation, some of the aims the protocol wishes to achieve are listed below:

- The user's password must never travel over the network;
- The user's password must never be stored in any form on the client machine: it must be immediately discarded after being used;

- The user's password should never be stored in an unencrypted form even in the authentication server database;
- The user is asked to enter a password only once per work session. Therefore users can transparently access all the services they are authorized for without having to re-enter the password during this session. This characteristic is known as **Single Sign-On**;
- Authentication information management is centralized and resides on the authentication server. The application servers must not contain the authentication information for their users. This is essential for obtaining the following results:
    - ➔ The administrator can disable the account of any user by acting in a single location without having to act on the several application servers providing the various services;
    - ➔ When a user changes its password, it is changed for all services at the same time;
    - ➔ There is no redundancy of authentication information which would otherwise have to be safeguarded in various places;
- Not only do the users have to demonstrate that they are who they say, but, when requested, the application servers must prove their authenticity to the client as well. This characteristic is known as **Mutual authentication**;
- Following the completion of authentication and authorization, the client and server must be able to establish an encrypted connection, if required. For this purpose, Kerberos provides support for the generation and exchange of an encryption key to be used to encrypt data.

Definitions of the components and terms

Client – It is an entity that can obtain a ticket, usually either a host or a user.

Host – It is a computer that can be accessed over a network.

Service – Any program or computer you access over a network.

Ticket – A temporary set of electronic credentials that verify the identity of a client for a particular service. A ticket is a kind of encrypted data, stored on the client side, that is presented to an application server to demonstrate the authenticity of the clients identity. The Kerberos communication is based around these tickets.

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
- 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);

Keytab – It is a key table file containing one or more keys. A host or service uses a keytab file in much the same way as a user uses his or her password.

Encryption – 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).

Principal – It is a string that names a specific entity to which a set of credentials may be assigned, the name used to refer to the entries in the authentication server database. A principal is associated with each user, host or service of a given realm. A principal in Kerberos is of the following type:

component1/component2/.../componentN@REALM , but in practice a maximum of two components are used.

It generally has three parts:

- **primary –** the username or the name of the service
- **instance –** hostname or user group type, some kind of qualification
- **realm –** the logical network served by a single Kerberos database and a set of Key Distribution Centers, it indicates an authentication administrative domain.

For a user the principal has the following type *Name[/Instance]@REALM* , where the *instance* is optional and is normally used to better qualify the type of user (user1/admin@REALM). In the case of services the principal looks as following: *Service/Hostname@REALM* . The first component is the name of the service, for example *ftp, ssh.* The second component is the complete hostname (FQDN) of the machine providing the requested service.

Key Distribution Center (KDC) – The machine that issues Kerberos tickets, this is the central part of a Kerberos network.

It consists of three parts:

- an **Authentication Server (AS)**, which answers requests for Authentication issued by clients. The **AS** replies to the initial authentication request from the client, when the user, not yet authenticated, must enter the password**.** Here, we're in the **AS_REQUEST** and **AS_REPLY** challenging part (see below for details), where the client gets a **Ticket Granting Ticket** (**TGT**). If the users are actually who they say they are they can use the TGT to obtain other service tickets, without having to re-enter their password.
- a **Ticket Granting Server**, which issues **Ticket Granting Service** (**TGS**) to a client. This is the **TGS_REQUEST** and **TGS_REPLY** part, where a client gets a **TGS** that allows him to authenticate to a service accessible on the network.
- a **database**, that is the container for entries associated with users and services, it stores all the secret keys (clients and services ones), as well as some information relating to Kerberos accounts (creation date, policies, etc.).

Ticket Granting Ticket (TGT) – A special Kerberos ticket that permits the client to obtain additional Kerberos tickets within the same Kerberos realm.

How does Kerberos work?

Authentication mechanism is the first step to be done in a Kerberos environment. It provides the user with a **Ticket Granting Ticket** (TGT), that serves post-authentication for later access to specific services, Single Sign On. 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 that are going to be discussed are listed below:

- **AS_REQ** is the initial user authentication request (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.

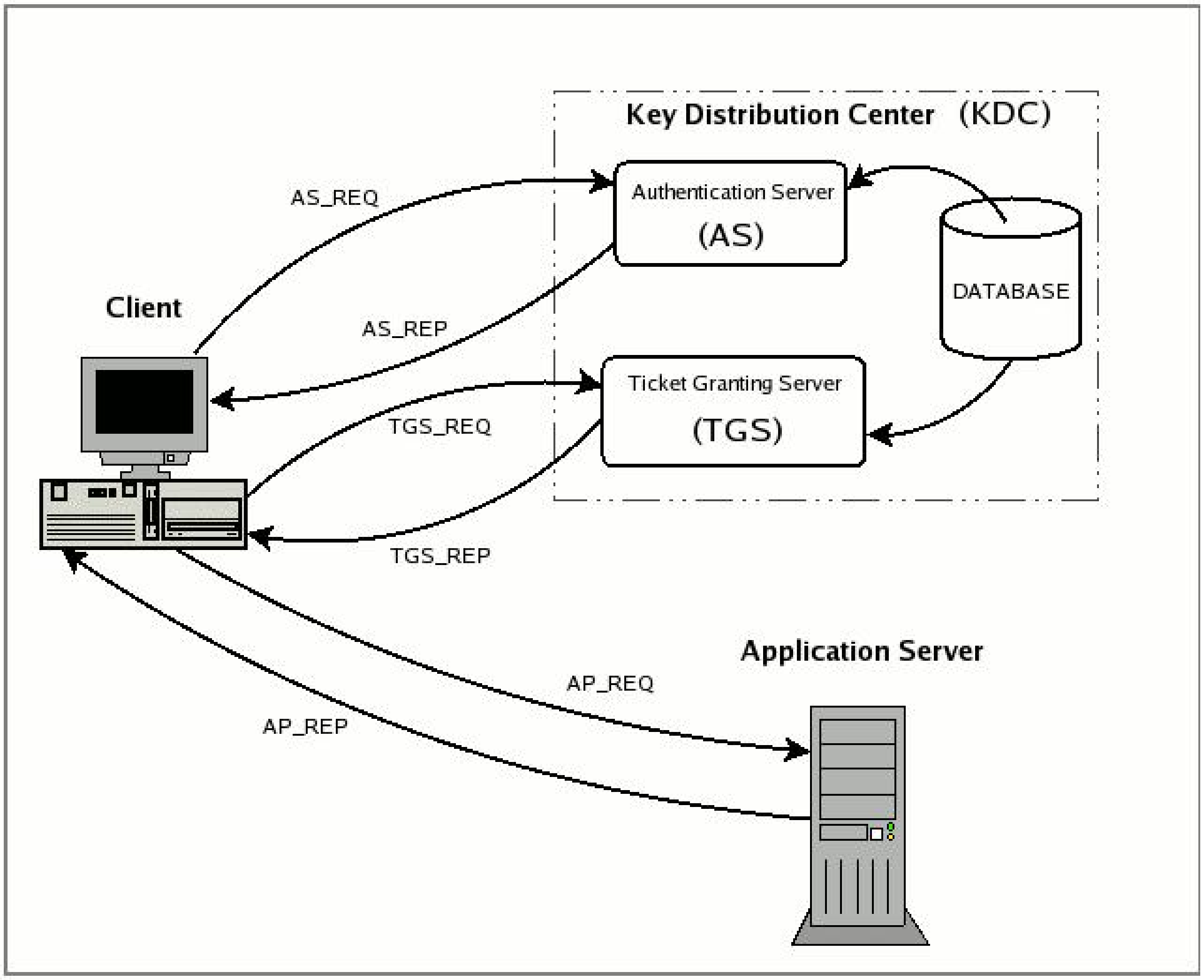

Key Distribution Center (KDC)
Authentication Server (AS)
DATABASE
Ticket Granting Server (TGS)
Client
AS_REQ
AS_REP
TGS_REQ
TGS_REP
Application Server
AP_REQ
AP_REP

Illustration 1: Kerberos operation, http://www.zeroshell.net/eng/kerberos/Kerberos-operation/

The first message, the **AS_REQ** is sent to the KDC in plain text, where the client asks the KDC (more specifically the AS) for a Ticket Granting Ticket and it contains:

- the client's principal name,
- the Ticket Granting Server's principal (termed "krbtgt principal", needed to obtain further TGS
- the client timestamp
- the requested ticket lifetime (usually 8 to 10 hours long)

The KDC receives this message, checks if the client's principal has a match in the database, and if the timestamp between client's machine and KDC are close enough (3 to 5 min.), this is just a way of warning the user from on incorrect time synchronization, before going any further into authentication.

Upon checking, the Authentication Server generates a random session key ("short term" key). The KDC makes two copies of it, one is for the client and it is added to the **AS_REP** message, the second copy remains available for the Ticket Granting Server. This key is mainly used for later negotiations for other tickets concerning kerberized services.

If the client succeeded in his authentication, the **KDC** returns an **AS_REP** message, containing the **Ticket Granting Ticket,** which will be stored in some kind of credential cache for future use.

The **AS_REP** message is formed of two layers; the first one is encrypted with the user's key, while the second layer is the **TGT** itself, first encrypted with the *Ticket Granting Server's* key, then re-encrypted with the user's key.

The content of the **AS_REP** message is the following:

- encrypted with user's key
    - copy of session key for user
    - ticket lifetime
    - krbgt principal name
- first encrypted with *Ticket Granting Server*'s key, then user's key. This is the **TGT:**
    - copy of session key
    - effective ticket lifetime
    - KDC timestamp
    - client principal
    - client IP address

Although the **TGT** is decrypted and cached onto the client, its content cannot be read on the client's side. It is effectively encrypted with the *Ticket Granting Server*'s key, which is only
known by **Ticket Granting Server**.

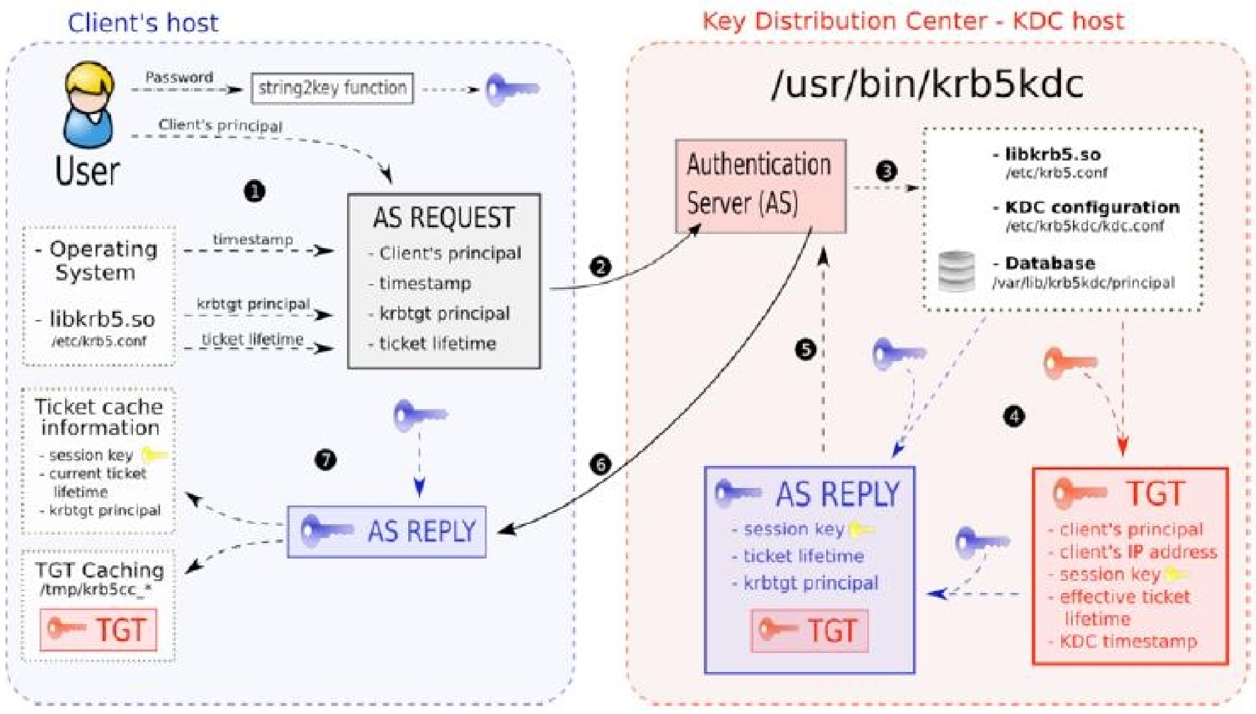


Illustration 2: TGT delivery [Migeon]

The mechanism of *Authentication* is presented in the previous diagram. When the **AS_REP,** comes back to the user 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 entered the correct password, the decrypting operation will be successful and the session key will be extracted, and the **TGT** will be stored in the user's credential cache.

Supposing that the client has already gone through the *authentication mechanism,* and has a *TGT (Ticket Granting Ticket)* and a session key, now he can access particular service over the network and for this he/she requires a **TGS (Ticket Granting Service).** This request is also separated into two steps, **TGS_REQ** and **TGS_REP.** Both messages are encrypted for security reasons.

When the user wishes to access a kerberized service, he must authenticate himself to it, this means a separate connection to the **Ticket Granting Server,** the **TGE_REQ** message. This message is composed of :

- the **TGS** request itself, containing the service principal and the requested lifetime
- the **TGT** acquired earlier, with a successful authentication
- an authenticator

The authenticator is a message encrypted with the session key acquired during the **AS** process and contains the user's principal and a timestamp. This way the **KDC** ensures that this message is coming from the right person, by checking the temporary session key, and the timestamp. Upon a successful request (this means a valid **TGT,** and a correct *authenticator*), the **Ticket Granting Server** returns the **TGS**.

At this stage the server generates a new set of session keys. The reply message from the server is encrypted with the session key acquired through **AS** process so only the client that effectively identified himself some time ago to **KDC** is able to read its contentand to extract the **TGS** from it. The **TGS_REP** message contains the following:

- encrypted with session key acquired through **AS** process
  - copy of the new session key for the user
  - effective ticket lifetime
  - service's principal name
- first encrypted with service's long term key, than with the actual session key, this is the **TGS**:
  - copy of the new session key for service
  - effective ticket lifetime
  - KDC timestamp
  - client principal
  - client IP address

When the client receives the reply, having in the credential cache the session key, it can decrypt the part of the message containing the other session key and the *TGS,* but this remain encrypted. In the next diagram are presented visually these steps, for requiring this **TGS**.

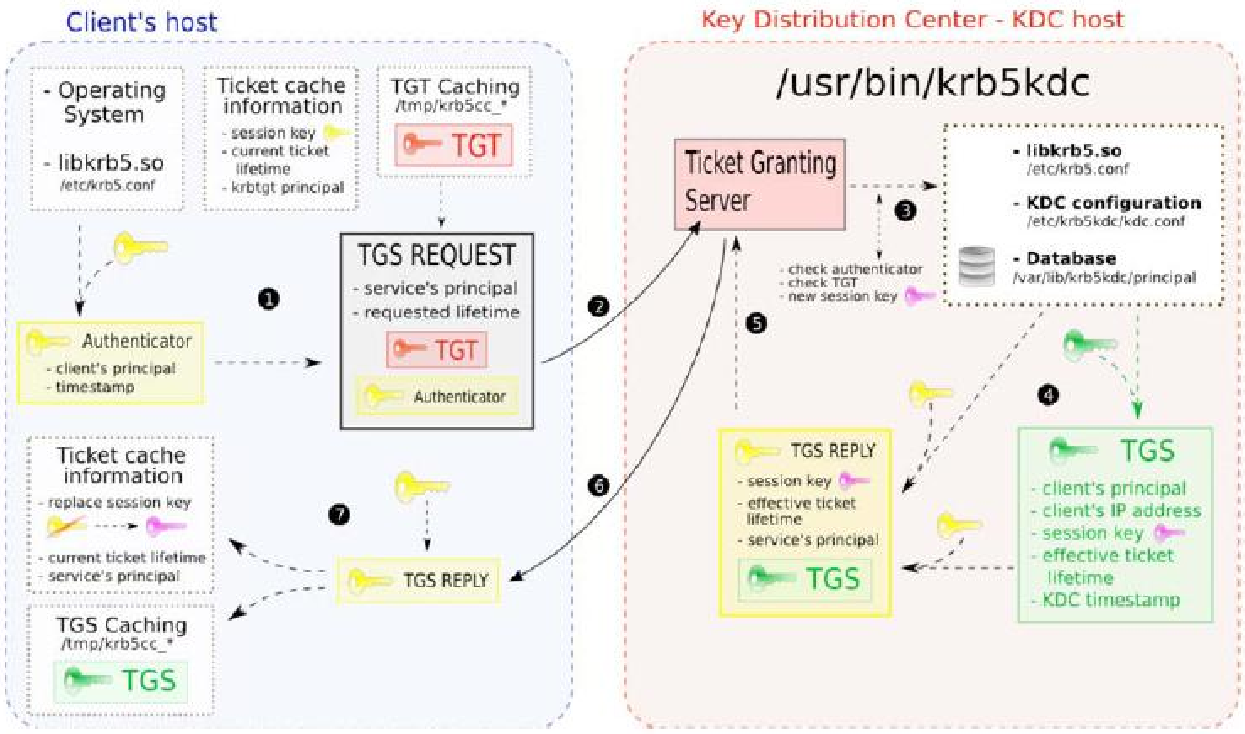


Illustration 3: TGS delivery [Migeon]

Once the client obtained its **TGS**, he will use it to authenticate himself to the requested service directly, via an **AP_REQ** message. The service has access to its keytab, a file that stores its long term key. This key will allow the service to decrypt the **TGS** sent by the client, and get all the information needed to identify user and create security context. The **AP_REQ** message it is not standard, unlike the previous messages where the **KDC** was involved, it depends on the application. The **AP_REQ** contains:

- an encrypted authenticator with the session key, containing
  - timestamp
  - user principal
- the *TGS* acquired earlier

When the previous request arrives, the application server opens the ticket using the secret for the requested service and extracts the session key, 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:

- the ticket has not expired

- the user principal matches the one present in the ticket
- the ip address of the **AP_REQ** matches the one in the ticket

As a conclusion we can say that Kerberos protocol can be divided into three main steps:

1. *Authentication process,* where the user (and host) obtain a **Ticket Granting Ticket (TGT)** as authentication token
2. *Service request process*, where the user obtain a **Ticket Granting Service (TGS)** to access a service
3. *Service access*, where the user use **TGS** to authenticate and access a specific service.

Installation of Kerberos V5

System requirements

In order to build Kerberos V5, it will need approximately 60-70 megabytes of disk space. The exact amount varies depending on the platform and whether the distribution is compiled with debugging symbol tables or not.

Downloading the Kerberos V5 release 1.7 source code

On the http://web.mit.edu/Kerberos/dist/index.html#krb5-1.7 MIT Kerberos Distribution Page can be found the newest Kerberos release's source code – the Kerberos V5 release 1.7 – packed in a tar file with a PGP signature, named krb5-1.7-signed.tar.

Unpacking the sources

After we download this package from the given MIT page, we unpack it: this way we get two files: krb5-1.7.tar.gz, which contains all the source codes and also documentations, and krb5-1.7.tar.gz.asc, which is a PGP signature for the source tree. MIT highly recommends that you verify the integrity of the source code using this signature.

We unpack the compressed krb5-1.7.tar.gz file in some directory, for example /home/student/krb5-1.7.

Building Kerberos

We can choose from different options regarding the building of Kerberos: if we will use it on one platform we can use a single directory tree which contains both the source files and the object files. If we need to maintain Kerberos on multiple platforms, we may use separate build trees for each platform.

We have chosen the building within a single tree.

Building within a single tree

In a terminal (in root mode) we must type:

```
>>cd /home/student/krb5-1.7/src
>>./configure
>>make
>>make check
>>make install
```

The **./configure** command configures the installation: checks, if the system has all the required libraries for a proper function of Kerberos. There must be a C compiler, tgetent(curses/ncurses libraries) and yacc, g++ command handlers.

Note: the C compiler must conform to ANSI C (ISO/IEC 9899:1990, c89). Some operating systems do not have ANSI C compiler, or their default compiler requires extra command-line options to enable ANSI C conformance.

The **make check** command tests the build: if appears the **krb5-config tests pass** line, we can continue with the installation.

The **make install** command installs the binaries.

Building with separate build directories

If you wish to keep separate build directories for each platform, for example you wish to build a directory for pmax binaries you might use the following procedure:

```
>>mkdir /home/student/krb5-1.7/pmax
>>cd /home/student/krb5-1.7/pmax
>>..src/configure
>>make
```

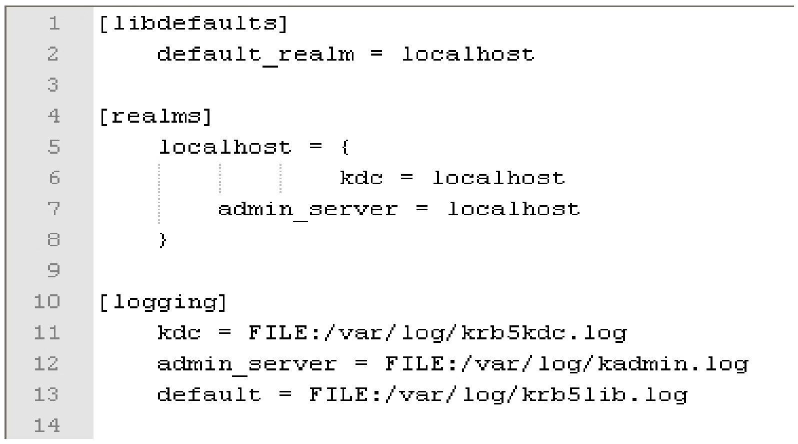
Configurations of Kerberos V5

Installing KDCs

The Key Distribution Centers (KDCs) issue Kerberos tickets. Each KDC contains a copy of the Kerberos database. The master KDC contains the master copy of the database, here are made all database changes, such as password changes. Slave KDCs provide Kerberos ticket-granting services.

1. Firstly we must edit the configuration files: */etc/krb5.conf* and */usr/local/var/krb5kdc/kdc.conf* to reflect the correct information for our realm.

   krb5.conf:



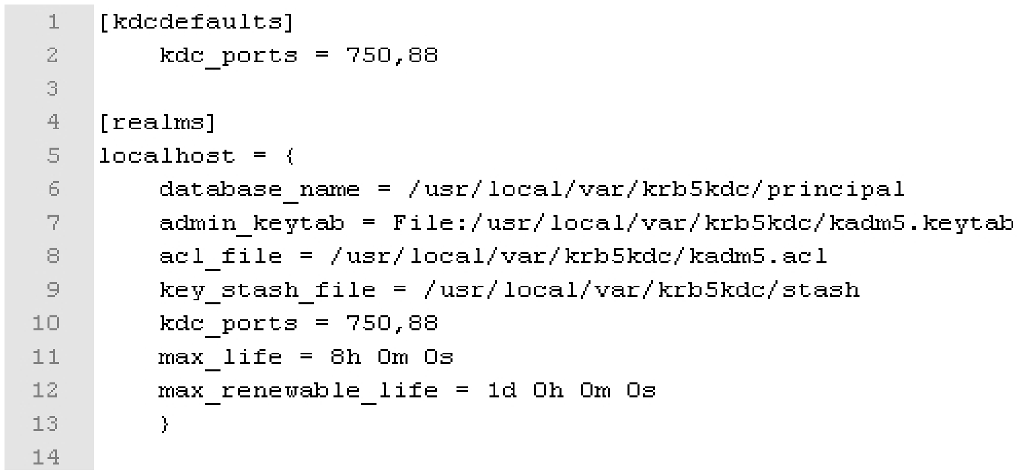
```
[libdefaults]
    default_realm = localhost

[realms]
    localhost = {
                kdc = localhost
        admin_server = localhost
    }

[logging]
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmin.log
    default = FILE:/var/log/krb5lib.log

```

The *krb5.conf* file contains Kerberos configuration information, including the locations of KDCs and admin servers for the Kerberos realms, defaults for the current realm and for Kerberos applications. It also contains mappings of hostnames onto Kerberos realms.

This file is set up in the style of a Windows INI file: sections are headed by the section name in square brackets.

- **libdefaults**: contains default values used by the Kerberos V5 library.
- **realms**: contains subsections keyed by Kerberos realm names. Each section describes realm-specific information, including where to find the Kerberos servers for that realm.
- **logging**: contains relations which determine how Kerberos programs are to perform logging.

kdc.conf:



```
[kdcdefaults]
    kdc_ports = 750,88

[realms]
localhost = {
    database_name = /usr/local/var/krb5kdc/principal
    admin_keytab = File:/usr/local/var/krb5kdc/kadm5.keytab
    acl_file = /usr/local/var/krb5kdc/kadm5.acl
    key_stash_file = /usr/local/var/krb5kdc/stash
    kdc_ports = 750,88
    max_life = 8h 0m 0s
    max_renewable_life = 1d 0h 0m 0s
    }

```

The *kdc.conf* file contains KDC configuration information, including defaults used when issuing Kerberos tickets. This file is set up in the same format as the *krb5.conf* file.

The file contains the following sections:

- kdcdefaults: contains default values for overall behavior of the KDC, for example the ports used by Kerberos.
- realms: contains subsections keyed by Kerberos realm names. Each subsection describes realm-specific information, including where to find the Kerberos servers for that realm.

Create the database

The **kdb5_util** command will be used on the master KDC to create the Kerberos database and the optional stash file. The stash file is a local copy of the master key that resides in encrypted form on the KDC's local disk, and is used to authenticate the KDC itself automatically before starting the **kadmind** and **krb5kdc** daemons. If we do not create it, the system will ask for this master key each time we start the Kerberos daemons.

The creation of the Kerberos database and stash file can be done by using the following command:

>>**/usr/local/sbin/kdb5_util create -r localhost -s**

After typing the master key two times, in the directory specified in the *kdc.conf* file (*/usr/local/var/krb5kdc*) will be created five files: two Kerberos database files (*principal.db* and *principal.ok*), the Kerberos administrative database file (*principal.kadm5*), the administrative database lock file (*principal.kadm5.lock*) and the stash file (*.k5stash*). The **-s** option creates the stash file.

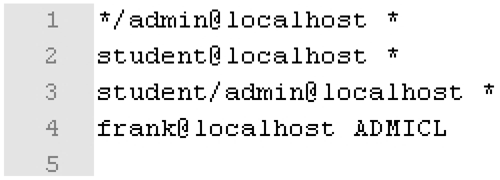
Add administrators to the acl file

We need to create an Access Control List (acl) file, and put the Kerberos principals into it. This file is used by the *kadmind* daemon to control which principals may view and modify the Kerberos database files. This file's name is the one given in the *kdc.conf* file.

The format of the file is:

Kerberos_principal permissions [target_principal] [restrictions]

kadm5.acl:



```
*/admin@localhost *
student@localhost *
student/admin@localhost *
frank@localhost ADMICL
```

A common use of an admin instance is so you can grant separate permissions to a separate Kerberos principal.

The permissions are represented by single letters, upper-case letters representing negative permissions (disallow):

- a: allows the addition of principals or policies in the database
- d: allows the deletion of principals or policies in the database
- m: allows the modification of principals or policies in the database
- c: allows the changing of the passwords for principals in the database
- i: allows inquiries to the database
- l: allows the listing of principals or policies in the database

- s: allows the explicit setting of the key for a principal.
- *: all privileges (admcil)
- x: all privileges (like *)

Add administrators to the Kerberos database

Next step is to add administrative principals to the Kerberos database. These administrative principals are the ones added to the acl file.

```
>>/usr/local/sbin/kadmin.local
kadmin.local: addprinc student@localhost
```

We must enter then re-enter a password for this principal, and if we succeed a “Principal “student@localhost” created” message will appear.

Create a kadmind keytab

This is an optional operation. The kadmind keytab is the key that the legacy administration daemons kadmind4 and v5passwdd will use to decrypt administrators' or clients' Kerberos tickets to determine whether or not they should have access to the given database.

If we need to create the kadmind keytab with entries for the principals *student* and *student/changepw* (these principals are placed in the Kerberos database automatically, when we create them), we type the following command:

```
>>kadmin.local
kadmin.local: ktadd -k /usr/local/var/krb5kdc/kadm5.keytab student student/changepw
```

The *ktadd* will save the extracted keytab as */usr/local/var/krb5kdc/kadm5.keytab*, like it was specified in the *kdc.conf file.*

If we do not use slave KDCs, at this point the installation and configuration is finished, and Kerberos is ready to be used.

Further configurations

Install the slave KDCs

We are now ready to start configuring the slave KDCs. Assuming we are setting the KDCs up so that we can easily switch the master KDC with one of the slaves, we should perform each of these steps on the master KDC as well as the slave KDCs.

Create host keys for the slave KDCs

Each KDC needs a host principal in the Kerberos database. We can enter these from any host, once the `kadmind` daemon is running. For example, if the realm is ATHENA.MIT.EDU, our master KDC were called kerberos.mit.edu, and we had two KDC slaves named kerberos-1.mit.edu and kerberos-2.mit.edu, we would type the following:

```
shell% /usr/local/sbin/kadmin
kadmin: addprinc -randkey host/kerberos.mit.edu
NOTICE: no policy specified for "host/kerberos.mit.edu@ATHENA.MIT.EDU";
assigning "default"
Principal "host/kerberos.mit.edu@ATHENA.MIT.EDU" created.
kadmin: addprinc -randkey host/kerberos-1.mit.edu
NOTICE: no policy specified for "host/kerberos-1.mit.edu@ATHENA.MIT.EDU";
assigning "default"
Principal "host/kerberos-1.mit.edu@ATHENA.MIT.EDU" created.
kadmin: addprinc -randkey host/kerberos-2.mit.edu
NOTICE: no policy specified for "host/kerberos-2.mit.edu@ATHENA.MIT.EDU";
assigning "default"
Principal "host/kerberos-2.mit.edu@ATHENA.MIT.EDU" created.
kadmin:
```

Extract host keytabs for the KDCs

Each KDC (including the master) needs a keytab to decrypt tickets. Ideally, we should extract each keytab locally on its own KDC. If this is not feasible, we should use an encrypted session to send them across the network. To extract a keytab on a KDC called kerberos.mit.edu, we would execute the following command:

```
kadmin: ktadd host/kerberos.mit.edu
kadmin: Entry for principal host/kerberos.mit.edu@ATHENA.MIT.EDU with
```

```
        kvno 1, encryption type DES-CBC-CRC added to keytab
        WRFILE:/etc/krb5.keytab.
    kadmin:
```

Set up the slave KDCs for database propagation

The database is propagated from the master KDC to the slave KDCs via the `kpropd` daemon.

To set up propagation, we create a file on each KDC, named `/usr/local/var/krb5kdc/kpropd.acl`, containing the principals for each of the KDCs. For example, if the master KDC were `kerberos.mit.edu`, the slave KDCs were `kerberos-1.mit.edu` and `kerberos-2.mit.edu`, and the realm were `ATHENA.MIT.EDU`, then the file's contents would be:

```
host/kerberos.mit.edu@ATHENA.MIT.EDU
host/kerberos-1.mit.edu@ATHENA.MIT.EDU
host/kerberos-2.mit.edu@ATHENA.MIT.EDU
```

Then, we add the following lines to `/etc/inetd.conf` file on each KDC (the line beginning with => is a continuation of the previous line):

```
krb5_prop stream tcp nowait root /usr/local/sbin/kpropd kpropd
eklogin   stream tcp nowait root /usr/local/sbin/klogind
=> klogind -k -c -e
```

The first line sets up the `kpropd` database propagation daemon. The second line sets up the `eklogin` daemon, allowing Kerberos-authenticated, encrypted rlogin to the KDC.

We also need to add the following lines to `/etc/services` on each KDC:

```
kerberos        88/udp      kdc       # Kerberos authentication (udp)
kerberos        88/tcp      kdc       # Kerberos authentication (tcp)
krb5_prop       754/tcp               # Kerberos slave propagation
kerberos-adm    749/tcp               # Kerberos 5 admin/changepw (tcp)
kerberos-adm    749/udp               # Kerberos 5 admin/changepw (udp)
eklogin         2105/tcp              # Kerberos encrypted rlogin
```

Propagate the database to each slave KDC

First, we create a dump of the database on the master KDC, as follows:

```
shell% /usr/local/sbin/kdb5_util dump /usr/local/var/krb5kdc/slave_datatrans
```

Next, we need to manually propagate the database to each slave KDC, as in the following example. (The lines beginning with => are continuations of the previous line.):

```
/usr/local/sbin/kprop -f /usr/local/var/krb5kdc/slave_datatrans
=> kerberos-1.mit.edu
/usr/local/sbin/kprop -f /usr/local/var/krb5kdc/slave_datatrans
=> kerberos-2.mit.edu
```

We will need a script to dump and propagate the database. The following is an example of a bourne shell script that will do this. (Note that the line that begins with => is a continuation of the previous line)

```
#!/bin/sh

kdclist = "kerberos-1.mit.edu kerberos-2.mit.edu"

/usr/local/sbin/kdb5_util "dump
=> /usr/local/var/krb5kdc/slave_datatrans"

for kdc in $kdclist
do
/usr/local/sbin/kprop -f /usr/local/var/krb5kdc/slave_datatrans $kdc
done
```

Create stash files on the slave KDCs

Create stash files, by issuing the following commands on each slave KDC:

```
shell% kdb5_util stash
kdb5_util: Cannot find/read stored master key while reading master key
kdb5_util: Warning: proceeding without master key
Enter KDC database master key:  <= Enter the database master key.
shell%
```

Add Kerberos principals to the database

Once our KDCs are set up and running, we are ready to use `kadmin` to load principals for our users, hosts, and other services into the Kerberos database. The keytab is generated by running `kadmin` and issuing the `ktadd` command.

Connecting through Kerberos

Start the Kerberos daemons on the master KDC

To start the Kerberos daemons, we need to type the following commands in a terminal in root mode:

>>**/usr/local/sbin/krb5kdc**

>>**/usr/local/sbin/kadmind**

Each daemon will fork and run in the background.

Assuming we want these daemons to start up automatically at boot time, we can add them to the KDC's `/etc/rc` or `/etc/inittab` file. We need to have a stash file in order to do this.

We can verify that they started properly by checking for their startup messages in the logging locations defined in `/etc/krb5.conf`. For example:

```
shell% tail /var/log/krb5kdc.log
Dec 02 12:35:47 beeblebrox krb5kdc[3187](info): commencing operation
shell% tail /var/log/kadmin.log
Dec 02 12:35:52 beeblebrox kadmind[3189](info): starting
```

Any errors the daemons encounter while starting will also be listed in the logging output.

OpenSSH

For administration, it is quite common to bounce from one host to another, mainly for maintenance tasks. As a consequence, having to retype a full password each time you need to login can be quite annoying.

OpenSSH provides a mechanism to avoid typing in a password to authenticate. It is a challenge-based negotiation, built around asymmetric keys.

Before we can connect with ssh and Kerberos, we need to configure ssh to use it with Kerberos. For this we need to take the following steps.

Configure OpenSSH

Edit the ssh server configuration file [by default, in */etc/ssh/sshd_config*] and ssh client configuration file [by default, in */etc/ssh/ssh_config*].

GSSAPIAuthentication yes

KerberosAuthentication yes

After these, we need to restart the ssh server:

>>**/etc/init.d/sshd restart**

Connecting

We make sure, that the Kerberos daemons are started by typing the following command:

>>**ps -A |grep krb5kdc**

If they aren't started, we give the following commands:

>>**/usr/local/sbin/krb5kdc**

>>**/usr/local/sbin/kadmind**

If the user, who wants to connect with ssh and Kerberos to localhost, doesn't exist in the Kerberos database yet, we need to create it.

We can do this, by typing the following commands in a terminal in root mode:

>>**kadmin.local**

kadmin.local: **ank student**

We will be prompted to enter and re-enter a password for the user student, thus the principal student@localhost will be created.

Now, we exit the Kerberos shell by typing quit:

kadmin:local: **quit**

Now that the user is created, it can authenticate to Kerberos. To do this the following command must be given:

```
>>kinit student
```

After we give the password we should be now authenticated to Kerberos as student@localhost (we now have student's TGT). To verify, we execute klist, and check that we have been properly authenticated:

```
>>klist
```

We should see the starting and expiring date of our ticket.

At this point, we can connect to localhost through ssh and Kerberos:

```
>>ssh -v localhost -l student
```

We type the Kerberos password and this is it: we are connected to localhost as student.

Conclusions

Kerberos V5 release 1.7 is based on the Kerberos authentication system developed at MIT. Under Kerberos, a client sends a request for a ticket to the Key Distribution Center (KDC). The KDC creates a ticket-granting ticket (TGT), which - if is successfully decrypted - indicates proof of the client's identity and permits the client to obtain additional tickets, which give permission for specific services. The requesting and granting of these additional tickets is user-transparent.

One of these specific services is connecting with SSH to different servers, to which Kerberos provides an authentication mechanism for additional security.

The installation and configuration of the Kerberos is a relatively simple and fast procedure and proves to be highly efficient with Single Sign On.

References

http://web.mit.edu/kerberos/#what_is

http://web.mit.edu/kerberos/www/krb5-1.2/krb5-1.2.6/doc/user-guide.html#SEC3

http://h71000.www7.hp.com/openvms/products/kerberos/kerberos_doc.html

http://www.softwareonline.hu/Article/View.aspx?id=2662

http://technet.microsoft.com/en-us/library/bb742516.aspx

http://learn-networking.com/network-security/how-kerberos-authentication-works

http://www.linuxtopia.org/online_books/linux_system_administration/kerberos_guides/kerberos-5.15_installation_guide/index.html

http://www.zeroshell.net/eng/kerberos/

http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci212437,00.html

http://www.kerberos.org/software/tutorial.html

http://www.zeroshell.net/eng/kerberos/Kerberos-operation/

http://www.visolve.com/security/ssh_kerberos.php

Kerberos V5 Installation Guide

Kerberos V5 System Administrator's Guide

Kerberos V5 UNIX User's Guide

[Migeon] Jean-Yves Migeon: The MIT Kerberos Administrator’s How-to Guide Protocol, Installation and Single Sign On