Kerberos in a production environment

Table of contents

[Types of Single Sign-on 4](#_Toc262140516)

[Client based 4](#_Toc262140517)

[Server based 6](#_Toc262140518)

[Kerberos Authentication Protocol 7](#_Toc262140519)

[Kerberos terminology 7](#_Toc262140520)

[How does Kerberos work? 8](#_Toc262140521)

[Schematic representation 10](#_Toc262140522)

[Installation prelude 13](#_Toc262140523)

[Setting up a DNS-Server 14](#_Toc262140524)

[DNS-Lookup process 15](#_Toc262140525)

[Installing Bind9 17](#_Toc262140526)

[apt-get 17](#_Toc262140527)

[source 17](#_Toc262140528)

[Configuring Bind 18](#_Toc262140529)

[named.conf 18](#_Toc262140530)

[named.conf.default-zones 18](#_Toc262140531)

[named.conf.local 18](#_Toc262140532)

[named.conf.options 19](#_Toc262140533)

[Zone files 20](#_Toc262140534)

[Reverse zones 22](#_Toc262140535)

[resolv.conf 23](#_Toc262140536)

[Starting Bind 24](#_Toc262140537)

[Setting up a Kerberos Server 25](#_Toc262140538)

[Installing an MIT Kerberos v5 Server 25](#_Toc262140539)

[apt-get 25](#_Toc262140540)

[source 25](#_Toc262140541)

[Configuring Kerberos 26](#_Toc262140542)

[krb5.conf 26](#_Toc262140543)

[kadm5.acl 28](#_Toc262140544)

[Administering Kerberos 28](#_Toc262140545)

[Initializing the Kerberos realm 28](#_Toc262140546)

[kadmin.local 28](#_Toc262140547)

[listprincs 28](#_Toc262140548)

[addprinc 29](#_Toc262140549)

[kadmin 29](#_Toc262140550)

[Adding users 29](#_Toc262140551)

[Setting up Kerberos clients 30](#_Toc262140552)

[Obtaining tickets manually 30](#_Toc262140553)

[Obtaining tickets automatically 31](#_Toc262140554)

[libpam-krb5 31](#_Toc262140555)

[common-auth 31](#_Toc262140556)

[common-session 31](#_Toc262140557)

[result 31](#_Toc262140558)

[Kerberized services 32](#_Toc262140559)

[telnet 33](#_Toc262140560)

[Server principal/keytab 34](#_Toc262140561)

[connecting 35](#_Toc262140562)

[Remote Shell 36](#_Toc262140563)

[Server principal/keytab 36](#_Toc262140564)

[Connecting 36](#_Toc262140565)

[Secure Shell 37](#_Toc262140566)

[sshd\_config 37](#_Toc262140567)

[ssh\_config 37](#_Toc262140568)

[Server principal/keytab 38](#_Toc262140569)

[Connecting 38](#_Toc262140570)

[File Transfer Protocol 39](#_Toc262140571)

[Server principal/keytab 39](#_Toc262140572)

[Connecting 39](#_Toc262140573)

[Sourcelist 40](#_Toc262140574)

Kerberos in a production environment

Single sign-on, abbreviated as “SSO”, is a way to work around the hassle of constantly having to authenticate into applications. Many employees and students will agree that constantly getting prompted to enter their password(s) over and over is more than just a nuisance and we can all agree that this proves to be detrimental for the overall efficiency of a company

When looking at Single Sign-on systems we can divide them in two main group, namely the client based and the server based ones.

# Types of Single Sign-on

## Client based

Client Based Single Sign-on systems prompt the user for his password the first time he wants to log in to a new website or application. The program then stores the password on the local computer and subsequently it will automatically enter these credentials whenever the user gets prompted for them.

Unfortunately this way of authentication requires some user input which makes room for human error. Another disadvantage to this is that in most cases all the data is stored on the local hard drive. It would be possible to work around this by for example bringing your authentication data with you everywhere you go on a stick, but this just creates further *complications*.

One big advantage of this method, however, is that once you have configured the application you can immediately gain access to multiple sites and applications that are not interconnected.

In the end this method of Single Sign-on authentication simply lacks the flexibility needed to be efficient in a production environment. This does prove to be a good method for simplifying authentication on a personal computer.

On the following page you can find an example of the Novell SecureLogin Single Sign-on tool. Several screenshots are depicted on which you can see the prompts presented to the end-user on the first time he authenticates to a new site.



http://www.novell.com/products/securelogin/media.html



http://www.novell.com/products/securelogin/media.html

## Server based

The key component in a server based single sign-on environment is one central authentication server within a certain production environment which takes care of all the authentication requests. Once you have logged in to the authentication server you will possess a ticket, whenever you will try to log in to a service you will send that ticket to the service. On its turn the service will crosscheck the ticket with the central authentication server to see if you have access to that service. If you do you will be authenticated automatically.

It can pretty much be compared to a ski resort, once you paid for your skipass you will get access to all the slopes you paid for.

Once the Single Sign-on system is in place an employee can for instance log into his workstation and from that point on he will possess a ticket which will allow him to authenticate easily via the central authentication server. When he opens outlook his login credentials will be passed on to the mail server and he will be authenticated without having to do a thing. He will just as well get logged in with a smooth transition when he connects to the intranet, a local fileserver, etc.

Implementing this will of course have a positive influence on the efficiency of the production environment and it will result in a rather fast Return on Investment.

Some Single Sign-on systems prompt the user for a smartcard instead of his credentials, he will then have to enter the card into a reader and his credentials will be loaded by the card reader. It is also possible to use a smartcard on top of normal login credentials to increase security.

The most secure authentication method right now uses OTP-tokens. OTP stands for One Time Password. The user has a small hardware device which generates a new password every time. These are either based on a mathematical algorithm or on time synchronization with the authentication server.

This paper will further discuss the technical details of the Kerberos authentication protocol and show how to set up and configure a Kerberos server using normal login credentials in order to achieve an efficient Enterprise Single Sign-on system.

Kerberos

# Kerberos Authentication Protocol

Kerberos is an authentication protocol which allows users to easily log into network services over an insecure network in a secure manner. It was developed in the early 80’s by MIT and has since then known great popularity, mainly thanks to the incorporation of the protocol within the Windows server environments.

The Open Source nature of Kerberos also allows cross platform authentication which we will address later on in this paper.

## Kerberos terminology

Before we begin learning about how exactly Kerberos works it might not be a bad idea to learn the basic terminology concerning Kerberos. Here is a list where you can easily look up certain definitions. Most terms will be elaborated upon during the explanation of how the protocol works.

**Authentication Service (AS)** - Performs authentication and is a part of the Key Distribution Center (KDC).  
   
**Key Distribution Center (KDC)** - Holds secret keys (the crytographic keys) for *"principals"*; provides authentication; creates and distributes session keys (crytographic keys). Session keys and secret keys are crytographic keys. The KDS utilizes symmetric cryptography. A KDC has a Ticket Granting Service (see TGS) and the Authentication Service.  
   
**Principal** - Any object such as user, application, service, or resource which utilizes Kerberos authentication is referred to as principal. Collectively, the objects using Kerberos are principals. A Key Distribution Center (KDC) is responsible for one or more *"realms"* of principals. Any principal must "trust" the KDC. Principals do not directly trust each other. Only the KDC is supposed to have a copy of each principals "secret key".  
   
**Realm** - The group or set of principals which are grouped together logically by a network administrator is called a realm. Again, a Key Distribution Center (KDC) is responsible for one or more realms.  
   
**TGS (Ticket Granting Service)** - That part of the Key Distribution Center (KDS) which creates and distributes tickets to the objects (principals) containing session keys.  
   
**Ticket** - Simply a digital authentication token sent from the Authentication Service (AS). The first ticket sent **from** the AS to a principal (user, application, service or resource) is called the Ticket Granting Ticket (TGT).  
   
**Secret keys and Session keys** - Symmetric cryptography keys used for both authentication and/or data encryption.

http://www.hitmill.com/computers/kerberos.html

## How does Kerberos work?

We will explain first the basics of Kerberos authentication, if this is the first time you deal with two- way authentication we advise you to take the schematic representation you can find later on in this chapter alongside you as you read through this explanation.

As we said said before Keberos relies on trusted third parties to work and it utilizes tickets controlled by those parties.

There usually are 3 key players in the world of Kerberos authentication:

* the client that needs a service
* the server providing the service
* the key distribution center

The first two are pretty self explanatory, we will however elaborate a bit on the latter: the key distribution center. The key distribution center is in fact the heart of all Kerberos operations, it is composed of the authentication server which crosschecks any credentials with a user database and the ticket granting server which is responsible for checking the validity of tickets and helping the client establish a connection with the server. This will all become clear in a few moments.

Once the user enters his credentials the client will generate a client secret key which is a one way hash based on the username and password.

When the client wants to use a service it sends a clear text message to the authentication server. It says “user x wants to use service y”. Next the authentication server will check whether or not the user is in its database and it will also generate a client secret key based on the found username and password.

The authentication server then sends back two messages to the client. The first message is a Client/TGS session key. This ticket is an identifying value that will be used between the client and the Ticket Granting Server later on. This message is encrypted with the client secret key.

The second message sent is called a Ticket Granting Ticket. In this message you can find the client ID, the client network address, the validity period of the ticket and also the Client/TGS session key. This message is encrypted with the Ticket Granting Server secret key.

When the client receives the first message it can decrypt this with his client secret key and it will obtain the Client/TGS session key. It is however still unable to decrypt message B because the client does not have the Ticket Granting Server secret key.

Next the client will send two messages to the Ticket Granting Server.

The first one is a message containing the (still encrypted) Ticket Granting Ticket it received earlier. It will also contain a file service id which describes which service the client is requesting access to.

The second message will contain an authenticator composed of the client ID and timestamp, this message is encrypted with the Client/TGS session key it obtained earlier from the authentication server.

The Ticket Granting Server will decrypt the first message sent by the client and he will get the Ticket Granting Ticket, which contains the client ID, the client network address, the validity period of the ticket and the Client/TGS session key.

From now on the client and the Ticket Granting Server can talk to each other in a secure manner because now they both have the Client/TGS session key. Now the Ticket Granting Server will decrypt the authenticator message from the client using the Client/TGS session key and it will get the client ID and the timestamp.

The Ticket Granting Server will now check if the Client ID from the Ticket Granting Ticket and the client ID from the client authenticator message are the same, it will also look at the timestamp to see whether or not the ticket has exceeded the ticket validity period.

If the right conditions are met the Ticket Granting Server will send the client a Client-to-Service ticket which contains the client ID, the network address, the validity period (of the Client-to-Service ticket) and last but not least: the Client/Server session key. This message will be encrypted with the Service Secret key – which is different for every service you request access to.

The Service Secret Key is also sent to the client encrypted with the Client/TGS session key. The client decodes this message with the Client/TGS session key and it obtains the Service Secret Key.

The client then sends two messages to the service server. It the earlier obtained Client-to-Service ticket which contains the client ID, the client network address, the validity period of the ticket and also the Client/Server session key. This message is encrypted with the Service Secret Key.

Next it also sends an authenticator once again composed of a client ID and a timestamp, this time it is encrypted with the Client/Server session key.

The service server will then decrypt the first message it will receive with the service secret key and it will obtain the client ID, the client network address, the validity period of the ticket and the Client/Server session key. It will then decrypt the authenticator message with the Client/Server session key.

Now the service server will check whether the client ID’s match and whether or not the ticket validity period has been exceeded. If the right conditions are met the Service Server will send the timestamp + 1 to the Client to confirm that he is in fact the correct server. This message is encrypted with the Client/Server session key. The client will decrypt this message, if it contains the correct timestamp (+1) it will henceforth trust the fileserver. It will then issue a service request to the service server and the server will service that request.

As you can see the Kerberos protocol utilizes many different keys in order to maintain security at all times. The only vulnerability in Kerberos is a brute-force attack, but the limited ticket validity period protects quite well against this.

## Schematic representation

If this is the first time you come in touch with two-way authentication, it might all be a bit difficult to comprehend. Therefore you can find the entire process depicted in the following diagrams.





Client sends authentication request saying which user wants to use which service



Authentication Server generates and sends the Client TGS Session Key, encrypts it with the Client secret key  
Authentication Server generates and sends the Ticket Granting Ticket, encrypts it with the TGS secret key



Client Decrypts the Client/TGS Session Key with the Client secret Key.  
Client sends Authenticator to the Ticket Granting Server, encrypts it with the Client/TGS session Key  
Client forwards the Ticket Granting Ticket to the Ticket Granting Server



Ticket Granting Server decrypts the Ticket Granting Ticket with the TGS secret Key  
Ticket Granting Server decrypts the Authenticator with the Client/TGS session Key  
Ticket Granting Server compares contents of TGT with the authenticator  
Ticket Granting Server sends Client to Service ticket to the client, encrypted with the service secret key  
Ticket Granting Server sends Client Server Session key to the client, encrypted with Client/TGS Session key.

  
Client decrypts the Client Server Session key with the Client/TGS session key.  
Client forwards the Service ticket to the Service Server.  
Client sends Authenticator to the Service Server, encrypts it with the Client Server Session Key.

  
Service Server decrypts the Service Ticket with the service secret key and acquires the client/service session key.  
Service Server decrypts the authenticator with the client/service session key.  
Service Server compares the contents of the service ticket with the authenticator, if the right conditions are met it will send the authenticator’s timestamp +1 back to the client to validate its authenticity.



Client Issues Service Request   
Service server supplies the service

# Installation prelude

Not only will we be installing a Kerberos server but we will also be installing a dns-server. A Kerberos server will always work in cooperation with a dns-server.

When installing the following services we will be building everything from source considering this is more efficient and that this is universal for all linux distributions. It is however possible to acquire the services by using software manage tools such as yast, zypper, aptitude-get, etc depending on the linux distribution you are running.

For this paper we will be using the Ubuntu Server 9.10 distribution.

To compile the services from source the gcc package management tool and the make utility has to be installed. These should be installed if you did a basic ubuntu installation.

On the OpenSuse 11.2 distribution where we did some earlier testing these packages were not yet installed. You can install them easily by issuing the following commands:

sudo zypper install gcc  
sudo zypper install make

# Setting up a DNS-Server

Even though dns is not the main focus of this paper Kerberos does rely **heavily** on a fully functional forward and reverse dns. In this section we will be configuring a Berkeley Internet Name Domain server, more commonly known as bind. This is one of the de facto standards on Unix-like systems.

The domain we set up in this section will be used later on when configuring the Kerberos server and other network services.

We will however only be covering the basics of dns and bind, in any case enough to help you set up a server and understand what is going on. The DNS protocol is quite extensive and we could easily devote an entire paper to just this subject.

If you are interested in learning more about dns refer to the following link which contains a summary of all dns related RFC’s: <http://www.dns.net/dnsrd/rfc/>

## DNS-Lookup process

Before starting on the installation of the server it is useful to have a general understanding of what exactly is going on during the DNS-lookup process.

Let us assume that we want to go to the address <http://toledo.khm.be>

When a dns enabled client tries to go to that address its system will contact one of the root servers. These servers are authoritive for the dns root zone. There are 13 root server clusters, when taking a closer look at the nameserver status we can see that there are 13 zones detected, these zones refer to the root servers.

Checking for nameserver BIND  
version: 9.6.1-P3  
CPUs found: 1  
worker threads: 1  
number of zones: 13  
debug level: 0  
xfers running: 0  
xfers deferred: 0  
soa queries in progress: 0  
query logging is OFF  
recursive clients: 0/0/1000  
tcp clients: 0/100  
server is up and running

When reading dns addresses we read them from back to front. One of the root servers will take a look at the address and it will see that the first part is “be”.

It will then return a message to your computer which will forward you to one of the top level domain servers. Top-level domain dns servers are servers which are very high in the dns hierarchy and are often responsible for namespaces such as .com, .net, .edu, …

In our case the top-level domain server we will be forwarded to is the server responsible for the .be zone. This server knows about the khm zone and will send the address of the khm dns server back to the client.

The client will then contact the khm dns server which will finally supply us with an address record.

If we issue the dig command we can see how this process works

#dig toledo.khm.be  
  
; <<>> DiG 9.5.1-P2.1 <<>> toledo.khm.be  
;; global options: printcmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17431  
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 0  
  
;; QUESTION SECTION:  
;toledo.khm.be. IN A  
  
;; ANSWER SECTION:  
toledo.khm.be. 3600 IN CNAME athens.khm.be.  
athens.khm.be. 3600 IN A 193.191.150.39  
  
;; AUTHORITY SECTION:  
[khm.be](http://khm.be). 86400 IN NS dns.khm.be.  
[khm.be](http://khm.be). 86400 IN NS ns.belnet.be.  
  
;; Query time: 373 msec  
;; SERVER: 127.0.0.1#53(127.0.0.1)  
;; WHEN: Tue May 18 13:23:53 2010  
;; MSG SIZE rcvd: 110

In green you can see the top level domain server which supplied us with the khm nameserver which is colored in yellow. In red you can see the answer (ip-address record) we got from that nameserver. The dig command can prove to be particularly useful for troubleshooting any dns problems you may encounter.

The most important things you need to remember about the dns protocol is that it follows a highly hierarchical tree-like architecture and that it approaches addresses from back to front.



## Installing Bind9

### apt-get

If you are running ubuntu like us, you can easily install the bind server by issueing the following command as root.

sudo apt-get install bind9

### source

To install the package by compiling it from source start by downloading the latest version of bind at <http://www.bind9.net/>  
Once you reeled in the package you can extract it by navigating to the Download directory and issuing the following command:

cd /home/fre/Download  
tar –zxvf bind-9.0.0.tar.gz

Next you will need to navigate to the newly made directory. There we will build the source libraries, generate the binaries and install the software. Do note that the last command must be issued as root.

cd bind-9.0.0  
make  
sudo make install

## Configuring Bind

### named.conf

In the directory /etc/bind/ you can find the configuration files for bind. The most important configuration file is the named.conf file, this file is however “split up” in several smaller files to provide us with a better overview.

If you installed bind correctly the named.conf file should look something like this:

#cat /etc/bind/named.conf

include “/etc/bind/named.conf.options”;  
include “/etc/bind/named.conf.local”;  
include “/etc/bind/named.conf.default-zones”;

### named.conf.default-zones

This file contains all the default zones that the dns-server uses. It is not necessary to edit this file.

### named.conf.local

In this file we will be specifying the zone for which our dns-server will be responsible. A zone refers to a specific portion of the global dns namespace. It represents the boundaries of the authority a dns server has.

Open the named.conf file as root with a text editor suchs as vi, nano, gedit, etc. For the unexperienced user we would like to recommend gedit because it ressembles the windows notepad application.

sudo vi /etc/bind/named.conf.local

Our named.conf file looks like this:

zone "khm.lan" in {  
 type master;  
 file "khm.zone";  
};

As you can clearly see the zone “khm.lan” is specified on the first line. You can call the zone anything you want, just make sure you do not use any dns-specific keywords. Calling the zone “khm.local” will for instance confuse the server.

The type master parameter makes this a master dns-server which gets its information from a local source. The local source is specified right underneath after the file parameter. This refers to the zone file which contains the information to perform dns-services.

### named.conf.options

The named.conf.options file is used for specifying other configuration parameters that are unrelated to the zone specification.

In our case we want to configure our dns-server so that it will forward dns queries it can’t handle itself to another dns server, therefore we will add a forwarder in the file, in our case this looks like the following:

forwarders {

192.168.0.1;

};

In our test topology the dns-queries that cannot be answered by our own dns-server will now be forwarded to the address 192.168.0.1. In our test topology this is a simple D-Link router which on its turn will forward any queries it cannot answer to the isp’s dns-server. We will learn more about the dns lookup process later in this chapter.

### Zone files

You will have to create the zone file yourself. To create the zone file first navigate to the right directory and issue the touch command.

cd /var/cache/bind  
sudo touch khm.zone

Next open the empty file you just created with a text editor.

sudo vi khm.zone

We configured our zone file like this:

$TTL 86400 ; max TTL  
$ORIGIN khm.lan.  
@ IN SOA ns.khm.lan. root.khm.lan. (  
 2010040701 ;serial number  
 28800 ;refresh after 8 hours  
 7200 ;retry after 2 hours  
 604800 ;expire after a week  
 3600 ) ;minimum TTL of 1 hour  
@ IN A 192.168.0.152  
@ IN NS khm.lan.  
@ IN MX 10 mail.khm.lan.  
www IN CNAME khm.lan.  
 IN CNAME khm.lan.

#### SOA

The first record in the zonefile is the SOA or the Start of Authority record. This record first lists the nameserver of the zone and next the e-mail address of the administrator of this zone. Note that the ‘@’ character has been replaced by a period.

The next parameter of the SOA record is a serial number. The serial number helps slave servers identify which version of the zone file they have saved locally. For that reason the serial number needs to be increased every time a zone file is updated. It is best practice to use a date here.

All the other parameters of the SOA file are all expressed in seconds, you can find the explanation in the file itself.

#### A

The A or address record supplies the dns-server with an ip-address. Because there is just an @ this A record will refer to khm.lan.  
It is ofcourse possible to assign multiple addresses within the same zone in the following way:

@ IN A 192.168.0.152  
Firewall IN A 192.168.0.1

Switch IN A 192.168.0.3

Replaytv IN A 192.168.0.200

In this example firewall.khm.lan for instance will refer to 192.168.0.1

#### MX

The MX record is used for mailservers. It allows you to specify a priority in front of the alias you will be using for the name server. This way you can prioritize your mail traffic if you have more than one mailserver.

#### NS

Specifies the name server to used to look up a domain.

#### CNAME

The CNAME or Canonical name is used to assign an alias to already existing A records. In our example “www.khm.lan” will refer to the same address record as “khm.lan”.

www IN CNAME khm.lan.  
alias1 IN CNAME khm.lan.  
alias2 IN CNAME khm.lan.  
alias3 IN CNAME khm.lan.

“alias1.khm.lan”, “alias2.khm.lan”, “alias3.khm.lan” and “www.khm.lan” would all refer to “khm.lan.” in this example.

### Reverse zones

Reverse zones are zones that help services find back the dns-name linked to a certain address.  
This is – as the name clearly implies – a *reversed* dns lookup.

These reverse zones are of vital importance for running Kerberos because the tickets sent across the network are linked to the FQDN (Fully Qualified Domain Name) of the Kerberos server.

To create a reverse zone we will have to create an “in-addr.arpa” domain, to do this we reverse the network part of our ip-address and add the “in-addr.arpa” zone.

Considering we have a /24 class C network address this will become 0.168.192.in-addr.arpa.

We added the following entry to “*/etc/bind/named.conf.local”.*

zone "0.168.192.in-addr.arpa" {  
        type master;  
        notify no;  
        file "reverse-0.168.192.zone";  
};

Next we created the following zone file we specified earlier in “/var/cache/bind”.

$TTL 86400      ;max TTL  
@       IN      SOA     ns.khm.lan. admin.khm.lan. (  
                        2010051201      ;serial number  
                        28800           ;refresh after 8 hours  
                        7200            ;retry after 2 hours  
                        604800          ;expire after 1 week  
                        3600 )          ;minimum ttl of 1 hour   
                        NS      ns.khm.lan.  
152             IN      PTR     krb.khm.lan.

#### PTR

The PTR or pointer record is the record used for reverse lookups. The record links an ip-address to a fully qualified domain name. As you can see only the host part of the ip-address needs to be specified because the network part is already processed into the zone name.

### resolv.conf

The resolv.conf file tells your linux system where it has to send its dns queries. This needs to be configured correctly on all dns clients. We will however also configure this on our server because we want to be capable of querying our own server.

Start by opening the resolv.conf file with a text editor.

sudo vi /etc/resolv.conf

After editing, our resolv.conf file looked like this:

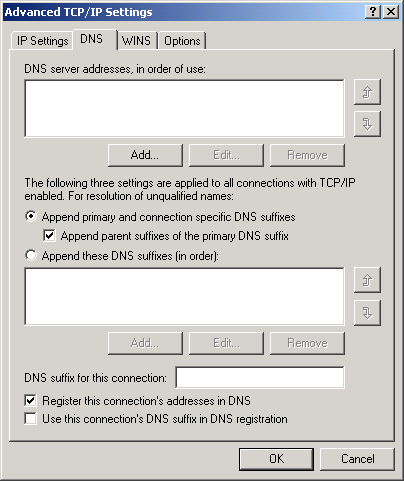
domain khm.lan  
search khm.lan  
nameserver 192.168.0.152

We added the one line telling the system to send dns queries on ip 192.168.0.152

We decided to leave the other search parameters there as well, this is because our dns server normally speaking has no knowledge about other domains on the internet, only about our internal network.

This , however, is not a necessity if you have the forwarders in named.conf configured properly.

The resolv.conf file corresponds with the following interface in Windows, which might seem more familiar to some users.



## Starting Bind

Now we will issue a reload command to test the zone files:

rndc reload

Next we will start the bind server, this can be done by running the script in the /etc/init.d folder.

/etc/init.d/bind9 start

If everything is working as it should now you should be capable of pinging “www.khm.lan”.

ping www.khm.lan  
  
PING khm.lan (192.168.0.152) 56(84) bytes of data.  
64 bytes from 192.168.0.152: icmp\_seq=1 ttl=64 time=0.028 ms  
64 bytes from 192.168.0.152: icmp\_seq=2 ttl=64 time=0.051 ms  
64 bytes from 192.168.0.152: icmp\_seq=3 ttl=64 time=0.050 ms

# Setting up a Kerberos Server

## Installing an MIT Kerberos v5 Server

We will be installing the Kerberos V5 software suite on an Ubuntu Server 9.10 system. This tutorial will of course also work on other Linux systems. We will refrain from using platform specific commando’s.

Some basic knowledge of Linux is recommended, this tutorial will however be really descriptive. The main idea of this tutorial is not to just set up a Kerberos server, but to help you learn and understand more about it.

### apt-get

Once again the easiest way to install Kerberos is by using aptitude-get. To download and install the latest stable Kerberos version this way issue the following commands:

apt-get install krb5-admin-server

apt-get install krb5-kdc

### source

To compile the Kerberos server from source, start by going to the MIT Kerberos distribution center. Once there, download the latest release. The “krb5-1.8.tar.gz” tarball file will need to be extracted. Open a terminal window and then go to the directory where the file is located, you can unpack the file by issuing the following commando:

cd /home/fre/Download  
tar –zxvf krb5-1.8.tar.gz

Next we will build the source libraries and create the binaries and of course install the software binaries. The last command must be issued as root.

cd /krb5-1.8/src  
./configure  
make  
sudo make install

## Configuring Kerberos

### krb5.conf

The krb5.conf file contains Kerberos configuration information concerning the locations of KDCs and admin servers for the Kerberos realms of interest, defaults for the current realm and for Kerberos applications, and mappings of hostnames onto Kerberos realms. It in fact identifies the different realms to the Kerberos server.

You will notice that the krb5.conf file is already quite extensive because it contains quite some default information about the standard MIT realms.

Underneath you can see what we added to our krb5.conf, the file can be found in the /etc map.

[libdefaults]  
default\_realm = KHM.LAN  
  
[realms]  
  
        KHM.LAN = {  
                kdc = krb.khm.lan  
                admin\_server = krb. khm.lan  
                default\_domain = khm.lan  
}  
  
  
[domain\_realm]  
.khm.lan = KHM.LAN  
khm.lan = KHM.LAN  
  
  
[logging]  
    kdc = FILE:/var/log/krb5/krb5kdc.log  
    admin\_server = FILE:/var/log/krb5/kadmind.log  
    default = FILE:/var/log/krb5/krb5lib.log

Do note that Kerberos realms are case sensitive. It is best practice to name your Kerberos realm after your domain, but in full caps. There are different sections in this file all headed by the name of the section in square brackets, similar to the structure of a windows INI file.

#### [libdefaults]

The libdefaults section contains default values used by the Kerberos library, we specified the default realm to be KHM.LAN  
Note that unlike the dns namespace the Kerberos realms are not hierarchical, you cannot make “subrealms” like accounting.khm.lan and expect that they inherit the principals of khm.lan. In that case you would have to make multiple realms.

#### [realms]

The realms section contains subsections containing information about the different realms. In our example we only have one realm specified.

The subsection specifies that the kdc and the admin server can both be found under krb.khm.lan.  
It is unnecessary to specify the entire path for the kdc and admin\_server parameter if the default\_domain has been set.

#### [domain\_realm]

The domain\_realm section contains relations which map domain names and subdomains onto Kerberos realm names. In our case we linked the khm.lan domain the the KHM.LAN realm.

#### [logging]

The logging section will already be there as soon as you open the file. This simply specifies where Kerberos will keep its logs. By default Kerberos logs to syslog. Keeping things separated can supply you with a better overview.

### kadm5.acl

kadm5.acl contains the access list Kerberos looks at when authenticating users. It defines the user access rights within Kerberos. There is no need to specify entries in the access lists for normal users because they only need the standard privileges.

For admin users, however, we want to grant all priveleges. Edit the kadm5.acl file so that it resembles the one depicted below. The file can be found in /var/lib/Kerberos/krb5kdc/

\*/admin@KHM.LAN \*

The access list abides by a fixed syntax. The general naming syntax is spec@realm, where the spec is composed of different components separated by a “/”. The first component defines the username and the second – if specified – defines the user role. Because of this an admin can also log in with normal user privileges.

The asterisk is once again used as a joker here, in our example we give all users with the role admin all permissions in the realm KHM.LAN.

## Administering Kerberos

### Initializing the Kerberos realm

If you have configured the krb5.conf file correctly you can initialize the Kerberos realm by simply running the following command, it will initialize the realm you specified as the default realm.

**krb5\_newrealm**

You will be prompted for a password, enter one and confirm it. Be sure to remember this as it is the main administrative password for the realm.

### kadmin.local

The kadmin.local command is a command that can only be used on the administrative server itself. Instead of accessing the Kerberos server over the network with the Kerberos protocol to authenticate the kadmin.local directly reads the Kerberos database present in the local filesystem. This is of course only possible as a user with sufficient rights on the local machine.

### listprincs

We will issue the listprincs command to test the basic functionality of our server. This command lists all the principals in the database.

fre@fre:~$ sudo kadmin.local  
Authenticating as principal root/admin@KHM.LAN with password.  
kadmin.local:  listprincs  
K/M@KHM.LAN  
kadmin/admin@KHM.LAN  
kadmin/changepw@KHM.LAN  
kadmin/history@KHM.LAN  
krbtgt/KHM.LAN@KHM.LAN

### addprinc

We will now add the principal which we will be using later for remote administration with the addprinc command.

kadmin.local:  addprinc root/admin  
WARNING: no policy specified for root/admin@KHM.LAN; defaulting  
to no policy  
Enter password for principal "root/admin@KHM.LAN":  
Re-enter password for principal "root/admin@KHM.LAN":  
Principal "root/admin@KHM.LAN" created.

Note that we specified in the kadm5.acl file that principals with the role admin have all priveledges.

### kadmin

Now that we have an administrative principal created we can log in with it by using kadmin instead of kadmin.local. As we said before this authenticates using the Kerberos protocol and will therefore prompt you for a password. Note that Kerberos will try to authenticate as the user that initiates the authentication, in this example it will log in as root because this command is issued as superuser.

root@fre:/# kadmin  
Authenticating as principal root/admin@KHM.LAN with password.  
Password for root/admin@ KHM.LAN:  
  
kadmin:  listprincs  
K/M@ KHM.LAN  
kadmin/admin@KHM.LAN  
kadmin/changepw@ KHM.LAN  
kadmin/history@ KHM.LAN  
krbtgt/KHM.LAN@ KHM.LAN  
root/admin@ KHM.LAN

We issued the listprincs command here again, and as you can see the principal root/admin has been added to the list.

### Adding users

As we mentioned before in this paper Kerberos does not keep any information about the user itself, its only task is to authenticate user principals. Applications are in most cases configured in such a way to look at a database to determine the privileges a certain user has.

Therefore we will create a principal with the same name as our user.

root@fre:/# kadmin  
Authenticating as principal root/admin@ KHM.LAN with password.  
Password for root/admin@ KHM.LAN:  
  
kadmin:  addprinc fre

# Setting up Kerberos clients

## krb5.conf

On clients the krb5.conf must also be configured, but not as extensively as on the server itself. In a setup with one KDC the client must simply be aware of the default domain and realm, and the location of the KDC.

[libdefaults]  
default\_realm = KHM.LAN  
  
[realms]  
  
        KHM.LAN = {  
                kdc = krb.khm.lan  
                default\_domain = khm.lan  
}

## Obtaining tickets manually

First of all we will use the klist command to observe which tickets we have.

fre@fre:~$ klist  
klist: No credentials cache found (ticket cache FILE:/tmp/krb5cc\_1000)

This output is normal considering we did not yet request any tickets as user fre. To acquire a ticket we will use the following command:

fre@fre:~$ kinit  
Password for fre@KHM.LAN:

Once we issue the klist command again we will notice that we have obtained a krbtgt – a ticket granting ticket – from the ticket granting server.

fre@fre:~$ klist  
Ticket cache: FILE:/tmp/krb5cc\_1000  
Default principal: fre@KHM.LAN  
  
Valid starting     Expires            Service principal  
04/26/10 11:13:31  04/27/10 11:13:31  krbtgt/KHM.LAN@KHM.LAN

To remove all tickets issue the kdestroy command:

fre@fre:~$ kdestroy

Note that it is possible to kinit as any user by simply issueing their user principal name after the kinit command. If a kinit is initiated without a username as a parameter Kerberos will by default attempt to kinit with the username of the user who issued the command.Obtaining tickets automatically

Ideally a kinit will be done when the user logs in so he himself no longer needs to supply anymore passwords. This is however only advised for single access machines because the tickets a user receives are stored locally and can be copied and used on another machine.

### libpam-krb5

In order to have the system perform a kinit on startup we will have to edit certain PAM files. PAM is a mechanism which integrates multiple authentication systems in one centralized API. We will not discuss the functionalities of PAM in-depth because that is beyond the scope of this thesis, we do however urge you to take a look at the supplied documentation to get a general understanding of what is going on.

We will begin by installing the PAM module “libpam-krb5” by issueing the following command:

sudo apt-get install libpam-krb5

This packet assures the interconnectivity between Kerberos and PAM.

### common-auth

The common-auth file contains a list of the authentication modules that define the central authentication scheme for use on the system. In order to achieve our goal the following line must be present:

auth [success=2 default=ignore] pam\_krb5.so minimum\_uid=1000

### common-session

The common-session file contains information on the tasks that should be performed at the start and at the end of a session. In order to pass on the user password to Kerberos add the following line:

session optional pam\_krb5.so minimum\_uid=1000 use\_first\_pass

### result

Now when you log out and log back in you should be granted a Ticket Granting Ticket, this will however only work if your user principal name and password correspond to the name and password supplied at the user login.

Test this by logging in an issuing a klist.

# Setting up Kerberized services

It is noteworthy that Kerberos is an all or nothing approach when we approach it from a security point of view. It has little use to for instance secure all your remote login and then still have a mailserver sending insecure plain text passwords across the network. Because all the authentication information is centralized there are risks involved and security holes like that should be avoided.

When Kerberos is configured properly the KDC will be the single point of failure.

In this section you will learn how to configure multiple kerberized services, the first example will be explained more extensively to give you a better insight in what is actually going on.

## telnet

To begin the installation of Kerberized telnet you must first acquire the krb5-telnetd packet by issueing the following command:

fre@fre: sudo apt-get install krb5-telnetd

Remember that you need to acquire a ticket before using the kerberized service. First acquire the ticket using the kinit command and then use the telnet.krb5 command to telnet into the server.

fre@fre: kinit  
Password for fre@KHM.LAN:  
fre@fre:~$ telnet.krb5 www.khm.lan  
Trying 192.168.0.152...  
Connected to www.khm.lan (192.168.0.152).  
Escape character is '^]'.  
telnetd: Authorization failed.  
Connection closed by foreign host.

As you can see there is still a connection error, the authorization failed. If we go check our logfile (/var/log/krb5/krb5kdc.log) we can see that the server is unknown to Kerberos.

fre@fre: cat /var/log/krb5/krb5kdc.log

…

TGS\_REQ (7 etypes {18 17 1623 1 3 2}) 192.168.0.152: UNKNOWN\_SERVER: authtime 0,fre@KHM.LAN for host/khm.lan@KHM.LAN

…

### Server principal/keytab

If you think of the way the Kerberos protocol works you will remember that the KDC must know a few things about the service in order to generate the client-to-service key to establish the client/service communication. Therefore we must add the service as a principal.

As you can also see in the log the telnet service is known to Kerberos as “host”, therefore we will be adding the following principal:

fre@fre: sudo kadmin.local  
Authenticating as principal fre/admin@KHM.LAN with password.

kadmin.local: addprinc -randkey host/krb.khm.lan

Contrary to the user principals that have a USER/ROLE@REALM layout the service principals have a SERVICE/SERVER@REALM layout. Note how we used the –randkey parameter to generate a random key. There is no need for the key to be known by us because it only needs to be handled by the KDC and the telnet server.

This shared secret between the KDC and the telnet server still have to be added to the keytab file. We do this by issueing the ktadd command:

fre@fre: sudo kadmin.local  
Authenticating as principal fre/admin@KHM.LAN with password.

kadmin: ktadd host/krb.khm.lan

If we take a look at the keytab file we will see that the correct entries have been added. You cannot open the keytab file with a text editor, you will need to issue the following command:

fre@fre:~$ sudo klist -k /etc/krb5.keytab  
Keytab name: WRFILE:/etc/krb5.keytab  
KVNO Principal  
---- --------------------------------------------------------------------------  
  3 host/krb.khm.lan@KHM.LAN  
  3 host/krb.khm.lan@KHM.LAN  
  3 host/krb.khm.lan@KHM.LAN  
  3 host/krb.khm.lan@KHM.LAN

### connecting

As of now it should be possible to authenticate into the Kerberos server. Do note that you need to telnet to krb.khm.lan considering this is considered the Kerberos server in krb5.conf.

Note that the –a parameter needs to be issued as well in order to enforce automatic authentication, or the –x parameter to ensure encrypted authentication. The server will otherwise refuse the insecure connection.

fre@fre:/var/log/krb5$ telnet.krb5 -a krb.khm.lan

Trying 192.168.0.152...  
Connected to krb.khm.lan (192.168.0.152).  
Escape character is '^]'.

[ Kerberos V5 accepts you as ``fre@KHM.LAN'' ]  
Last login: Thu Apr 22 10:53:23 on tty2  
Linux fre 2.6.31-20-generic-pae #58-Ubuntu SMP Fri Mar 12 06:25:51 UTC 2010 i686

We have logged in without having to supply an additional password, telnet is now properly kerberized.

## Remote Shell

To run a remote shell server install the following package:

sudo apt-get install krb5-rsh-server

### Server principal/keytab

Remote login uses the same “host” principal as telnet to authenticate. Issue a klist and a listprincs to check whether or not this data is present, if you already set up telnet they should already be added. If not you can add them by following the aforementioned steps.

### Connecting

To connect issue the rlogin command and bes ure to add the “–x” parameter to ensure encrypted authentication.

fre@fre:~$ kinit  
fre@fre:~$ rlogin -x krb.khm.lan  
This rlogin session is encrypting all data transmissions.  
Last login: Thu May 20 16:23:03 from krb  
Linux fre 2.6.31-14-generic-pae #48-Ubuntu SMP Fri Oct 16 15:22:42 UTC 2009 i686

To access official Ubuntu documentation, please visit:  
http://help.ubuntu.com/  
System information as of Thu May 20 15:56:43 CEST 2010

System load: 0.0 Memory usage: 35% Processes: 150  
Usage of /: 5.2% of 52.50GB Swap usage: 0% Users logged in: 1

Graph this data and manage this system at https://landscape.canonical.com/

fre@fre:~$

We logged in without being prompted for a password, this service is now properly kerberized.

## Secure Shell

Secure Shell is preferred over telnet and rlogin considering the security provided by it. There is no pre-configured kerberized package for SSH, therefore we will have to install the standard SSH daemon and reconfigure it for Kerberos use.

Begin by reeling in the openssh daemon:

sudo apt-get install openssh-server

### sshd\_config

The sshd\_config configuration file is responsible for the server configuration, we will have to change some parameters here in order for the daemon to participate in Kerberos authentication.

sudo nano /etc/ssh/sshd\_config

Once in the file you will have to add or uncomment the following lines to enable Kerberos authentication.

# Kerberos options  
KerberosAuthentication yes  
KerberosTicketCleanup yes

# GSSAPI options  
GSSAPIAuthentication yes  
GSSAPICleanupCredentials yes

These parameters specify that GSSAPI and Kerberos authentication will be enabled, users will thus henceforth be validated through the KDC.

The parameters also specify that the user’s ticket and credential cache files will automatically be destroyed upon logging out.

### ssh\_config

The ssh\_config file needs to be reconfigured on every client that will be using kerberized ssh. Begin by editing the file and setting the following parameters:

GSSAPIAuthentication yes  
GSSAPIDelegateCredentials yes

These parameters specify that GSSAPI authentication will be used and that the user credentials will be delegated to the server.

### Server principal/keytab

SSH uses the same “host” principal and keytab as telnet and rlogin to authenticate. Issue a klist and a listprincs to check whether or not this data is present, if you already set up telnet they should already be added. If not you can add them by following the aforementioned steps.

### Connecting

In this example we will obtain a ticket as beerend and then ssh into the server, specifying we are ssh’ing into beerend’s profile. As you can see the “-l” parameter specifies the user account.

fre@fre:~$ kinit beerend  
Password for beerend@KHM.LAN:   
fre@fre:~$ ssh krb.khm.lan -l beerend  
Linux fre 2.6.31-14-generic-pae #48-Ubuntu SMP Fri Oct 16 15:22:42 UTC 2009 i686  
  
To access official Ubuntu documentation, please visit:  
http://help.ubuntu.com/  
  
  System information as of Thu May 20 14:35:02 CEST 2010  
  
  System load: 0.0               Memory usage: 33%   Processes:       154  
  Usage of /:  5.2% of 52.50GB   Swap usage:   0%    Users logged in: 1  
  
  Graph this data and manage this system at https://landscape.canonical.com/  
  
63 packages can be updated.  
32 updates are security updates.  
  
Last login: Wed May 19 12:39:49 2010 from ubuntu-2.local  
$

As you can see we managed to SSH into the server without having to supply an additional password, we can conclude that this service is now properly kerberized.

## File Transfer Protocol

There is a kerberized version of the ftp daemon you can download by reeling in the following package:

sudo apt-get install krb5-ftpd

After installing this package no more changes need to be made to the server.

### Server principal/keytab

The principal and keytab that need to be added for the ftp are the following:

fre@fre:~$ sudo kadmin.local  
[sudo] password for fre:  
Authenticating as principal fre/admin@KHM.LAN with password.

kadmin.local: addprinc –randkey ftp/krb.khm.lan  
kadmin.local: ktadd ftp/krb.khm.lan

### Connecting

To connect simply ftp to the server.

fre@fre:~$ kinit  
fre@fre:~$ ftp krb.khm.lan  
Connected to krb.khm.lan.  
220 fre FTP server (Version 5.60) ready.  
334 Using authentication type GSSAPI; ADAT must follow  
GSSAPI accepted as authentication type  
GSSAPI authentication succeeded  
Name (krb.khm.lan:fre):  
232 GSSAPI user fre@KHM.LAN is authorized as fre  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp>

As you can see it is possible to log in without having to supply a password, this service is now properly kerberized.

# Sourcelist

[1] Novel, “Novel securelogin”, <http://www.novell.com/products/securelogin/media.html>, 2010

###### [2] Cheryl Gribble, “Tutorial for Kerberos”, <http://www.hitmill.com/computers/kerberos.html>, 1 februari 2009

###### [3]MIT, “Kerberos: The network authentication protocol”, <http://web.mit.edu/Kerberos/>, 8 april 2010

###### [4] Ninja, “Ninja lezingendavond over internetinfrastructuur”, <http://ninja4ru.wordpress.com/2008/02/25/ninja-lezingenavond-over-internet-infrastructuur/>, 25 februari 2008

###### [5]Docelic, “MIT Kerberos installation on Debian”, <http://www.debian-administration.org/articles/570#krb-install>, 5 december 2007

[6]MIT, “Install the master KDC”, <http://web.mit.edu/Kerberos/krb5-1.3/krb5-1.3.5/doc/krb5-install.html#Install%20the%20Master%20KDC>, 1995-2002

[7]OpenSuse community, “Example Configuration of Kerberos 5 Server on OpenSUSE 10.2”, <http://en.opensuse.org/Howto_KRB_server>, 25 april 2009

[8] Romain Lenglet, “How to install Kerberos 5”, <http://www.berabera.info/oldblog/lenglet/howtos/kerberoshowto/index.html>, 8 november 2005

[9] Raxxal, “DNS Server with Suse Linux 10.0”, <http://en.opensuse.org/DNS_server_with_SUSE_Linux_10.0>, 13 januari 2006

[10] Lars M. Hansen, “Writing zone files”, <http://www.hansenonline.net/Networking/zonefile.html>, 1999-2005

[11]Wiki community, “Domain Name System”, <http://en.wikipedia.org/wiki/Domain_Name_System>

[12]zytrax.com, “Chapter 5: OpenLDAP Samples”, <http://www.zytrax.com/books/ldap/ch5/index.html#step1-ldif>

[13] Stone, “Automatic kinit upon login”, <http://yoten.blogspot.com/2009/04/automatic-kinit-upon-login.html>, 22 april 2009

[14] Tillman Hodgson; “Kerberos5”, <http://www.freebsd.org/doc/en/books/handbook/kerberos5.html>

[15]Troy Dawson, “Configure your ssh daemon”, <http://fermilinux.fnal.gov/documentation/security/ssh-daemon/>, 20 november 2008

[16] atq, “Kerberos on Ubuntu”, <http://www.alittletooquiet.net/text/kerberos-on-ubuntu/>