Debian GNU and Ubuntu: Setting up MIT Kerberos 5

Davor Ocelic

SPINLOCK— advanced GNU/Linux and Unix solutions for commercial and education sectors.
<docelic@spinlocksolutions.com>

Copyright (c) 2006-2014 Davor Ocelic

Last update: Jan 07, 2014. — Up to date information, Ubuntu support

This documentation is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Abstract

The purpose of this article is to give you a straight-forward, Debian/Ubuntu-friendly way of installing and configuring MIT Kerberos 5.

By the end of this guide, you will have a functional Kerberos environment and one Kerberized service — the ability to login remotely to other machines in the network in a secure, encrypted and transparent way, without the need for typing in any passwords.

This article is part of Spinlock Solutions's practical 5-piece introductory series to infrastructure-based Unix networks, containing Debian GNU Guide, MIT Kerberos 5 Guide, OpenLDAP Guide, OpenAFS Guide and FreeRADIUS Guide.

Table of Contents

Introduction

Kerberos is a service that has been traditionally captivating system administrators' and advanced users' interest, but its (seemingly or not) high entry barrier and infrastructure requirements have been preventing many from using it.

Kerberos has already been the topic of numerous publications. Here, we will present only the necessary summary; enough information to establish the context and to achieve practical results.

You do not need to follow any external links; however, the links have been provided both throughout the article and listed all together at the end, to serve as pointers to more precise technical treatment of individual topics.

The role of Kerberos within a network

Kerberos is intended to centrally authenticate users, hosts and services on the network by verifying them against entries in the Kerberos database.

These entries (called "principals") consist of principal names, secret keys, key aging information and Kerberos-specific data. They're created or modified using a Kerberos-specific administrative tool.

When users type in their principal name and password anywhere on the network (within a Kerberos realm), their input is authenticated against the Kerberos database. In case of a successful authentication, the KDC ("`Key Distribution Center`") will *issue* users a "confirmation", called the TGT ("`Ticket-Granting Ticket`"). From that point on, and until their ticket expires, users will be transparently granted access to all network services they'll wish to use. (Here, the TGT will not grant access by itself — instead, it will be used in place of a password to automatically create further, service-specific tickets. Hence its name, the "Ticket-granting Ticket").

While the idea of a centralized network authentication is not unique, let's quickly identify Kerberos-specific elements in the authentication process:

- Kerberos is not in any way related to traditional system usernames or other data; Kerberos identity (or tickets) are obtained using a separate, Kerberos-specific mechanism. Arbitrary system user can obtain arbitrary Kerberos identity (provided they know the correct password).

  Often times, however, the Kerberos identity is obtained during log-in to the system and, for convenience, an assumption is made that the person's system login name matches the Kerberos principal name.

- The Kerberos database only contains the information necessary for Kerberos authentication; it does not (and can not) contain any other information, such as people's real names, Unix user and group IDs etc. This makes Kerberos well-defined and easy to fit in a network infrastructure.

  When a central directory is required for users' real names, IDs, meta information and other network information, OpenLDAP is often used in combination and installed after Kerberos as explained in another article from the series, the OpenLDAP Guide.

- Thanks to the design of the protocol, users' passwords never travel the wire in any form; Kerberos thus allows for secure authentication in and over untrusted networks.

- Kerberos requires mutual authentication of users and services, preventing stealing of information.

  To achieve this, Kerberos uses its database to store host and service principals alongside the "real", person-owned principals. This is normal behavior and indeed, the host and service principals will account for the majority of output when you list database entries for the first time.

- As users are only required to type in their password once (after which the TGT is used in place of the password to create further tickets), Kerberos offers a true SSO ("`Single Sign-On`") network solution.

You can find the complete Kerberos documentation at the MIT Kerberos website. Their on-line documentation is, however, only generated in multi-page HTML format — other more convenient formats (such as PostScript) are available within Kerberos release tarballs.

Glue layer: integrating Kerberos with system software

On all GNU/Linux-based platforms, Linux-PAM is available for service-specific authentication configuration. Linux-PAM is an implementation of PAM ("`Pluggable Authentication Modules`") from Sun Microsystems.

Network services, instead of having hard-coded authentication interfaces and decision methods, invoke PAM through a standard, pre-defined interface. It is then up to PAM to perform any and all authentication-related work, and report the result back to the application.

Exactly how PAM reaches the decision is none of the service's business. In traditional set-ups, that is most often done by asking and verifying usernames and passwords. In advanced networks, that could be retina scans or — Kerberos tickets.

PAM will allow for inclusion of Kerberos into the authentication path of all services, regardless of whether they natively support Kerberos or not.

You can find the proper introduction (and complete documentation) on the Linux-PAM website. Pay special attention to the PAM Configuration File Syntax page. Also take a look at the Linux-PAM(7) and pam(7) manual pages.

Conventions

Let's agree on a few conventions before going down to work:

- Our platform of choice, where we will demonstrate a practical setup, will be Debian GNU. The setup will also work on Ubuntu, and if any notable differences exist they will be noted.

- If using Debian GNU, install and configure sudo. On Ubuntu sudo will already be installed and configured to work for your regular user account.

  Sudo is a program that will allow you to carry out system administrator tasks from your normal user account. All the examples in this article requiring root privileges use sudo, so you will be able to copy-paste them to your shell.

  To install sudo on Debian GNU only, run:

  ```
  su -c 'apt-get install sudo'
  ```

If asked for a password, type in the root user's password.

To configure sudo on Debian GNU only, run the the following, replacing `USERNAME` with your login name:

```
su -c 'echo "USERNAME ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers'
```

- Packages that we will install during the complete procedure will ask us a series of questions through the so-called *debconf* interface. To configure debconf to a known state, run:

  ```
  sudo dpkg-reconfigure debconf
  ```

  When asked, answer *interface*=`Dialog` and *priority*=`low`.

- Monitoring log files is crucial in detecting problems. The straight-forward, catch-all routine to this is opening a terminal and running:

  ```
  cd /var/log; sudo tail -F daemon.log sulog user.log auth.log debug kern.log syslog dmesg messages kerberos/{krb5kdc,kadmin,krb5
  ```

  The command will keep printing log messages to the screen as they arrive.

- Our test system will be called `monarch.spinlock.hr` and have an IP address of `192.168.7.12`. Both the server and the client will be installed on the same machine. However, to differentiate between client and server roles, the client will be referred to as `monarch.spinlock.hr` and the server as `krb1.spinlock.hr`. The following addition will be made to `/etc/hosts` to completely support this scheme:

  ```
  192.168.7.12    monarch.spinlock.hr monarch krb1.spinlock.hr krb1
  ```

  > **Caution**
  >
  > Note that in some installations the system's network hostname is assigned to the localhost address `127.0.0.1`. This can and will cause problems for network operations, so make sure that your "localhost" entry in `/etc/hosts` looks exactly like the following (nothing more, nothing less):
  >
  > ```
  > 127.0.0.1  localhost
  > ```

  Finally, test that the network setup is as expected. Pinging the hostnames should report proper FQDNs and IPs as shown:

  ```
  ping -c1 localhost
  PING localhost (127.0.0.1) 56(84) bytes of data.
  ....

  ping -c1 monarch
  PING monarch.spinlock.hr (192.168.7.12) 56(84) bytes of data.
  ....

  ping -c1 krb1
  PING krb1.spinlock.hr (192.168.7.12) 56(84) bytes of data.
  ....
  ```

Kerberos 5

Now when everything has been properly prepared, let's move forward.

Server installation

Kerberos server installation basically consists of just two packages — the KDC (Key Distribution Center), which takes care of handling authentication requests and issuing Kerberos tickets, and kadmind (Kerberos master server), which allows *remote administration access* to the Kerberos database and carrying out of administrative tasks.

```
sudo apt-get install krb5-{admin-server,kdc}
```

Here are the Debconf answers for reference. The listing here includes all questions; some were asked in Kerberos 1.6 packages and some are asked only in Kerberos 1.7 and newer, and their order has changed a little as well. In any case, it's no problem — just answer the subset of questions you are asked:

```
Default Kerberos version 5 realm? SPINLOCK.HR
# (Your domain name in uppercase - a standard for naming Kerberos realms)

Add locations of default Kerberos servers to /etc/krb5.conf? Yes
# (Adding entries to krb.conf instead of DNS, for simplicity)

Kerberos servers for your realm: krb1.spinlock.hr
# (Make sure your DNS resolves krb1.spinlock.hr to
# the NETWORK IP of the server, NOT 127.0.0.1!). Hint is given in
# the section called “Conventions”.

Administrative server for your Kerberos realm: krb1.spinlock.hr
# (Make sure your DNS resolves krb1.spinlock.hr to
# the NETWORK IP of the server, NOT 127.0.0.1!). Same hint as above.

Create the Kerberos KDC configuration automatically? Yes

Run the Kerberos V5 administration daemon (kadmind)? Yes
```

As soon as the installation is done, the Kerberos admin server (**kadmind**) and the KDC will start. Kadmind will fail as, initially, there are no Kerberos realms created, which is fine.

Initial configuration

To create the Kerberos realm, invoke:

```
sudo krb5_newrealm

This script should be run on the master KDC/admin server to initialize
a Kerberos realm.  It will ask you to type in a master key password.
This password will be used to generate a key that is stored in
/etc/krb5kdc/stash.  You should try to remember this password, but it
is much more important that it be a strong password than that it be
remembered.  However, if you lose the password and /etc/krb5kdc/stash,
you cannot decrypt your Kerberos database.
Loading random data
Initializing database '/var/lib/krb5kdc/principal' for realm 'SPINLOCK.HR',
master key name 'K/M@SPINLOCK.HR'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.

Enter KDC database master key: PASSWORD

Re-enter KDC database master key to verify: PASSWORD
```

Note that the command may pause for a significant amount of time after printing "Loading random data".
To speed up the process and allow the kernel to generate enough random data to continue, log in to the machine in another terminal and execute a couple commands, such as `find /` and type random text into the terminal.
Once enough random data has been collected, the command execution will continue.

Now that the realm has been created, we need to adjust the Kerberos config file, `/etc/krb5.conf`. That file should to be the same on all Kerberos servers and clients belonging to the same realm.

`/etc/krb5.conf` is split into sections; you should search for section "`[domain_realm]`" (not "`[realms]`") and append your definitions:

```
.spinlock.hr = SPINLOCK.HR
spinlock.hr = SPINLOCK.HR
```

At the bottom of the file, you should add the logging section:

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

To create the logging directory and set up permissions, run:

```
sudo mkdir /var/log/kerberos
sudo touch /var/log/kerberos/{krb5kdc,kadmin,krb5lib}.log
sudo chmod -R 750  /var/log/kerberos
```

You do not need to restart the log monitoring command you ran earlier (see the section called “Conventions”) — the **tail -F** command will pick up new log files from the the `kerberos/` directory automatically.

To apply changes to the Kerberos server, run:

```
sudo invoke-rc.d krb5-admin-server restart
sudo invoke-rc.d krb5-kdc restart
```

Initial test

It is already the time to test the installation. We assume that both the admin server and the KDC can be restarted with no errors (which should be no problem to determine if you're monitoring the log files as advised).

As the first test, we will run command **kadmin.local** on the server. The **kadmin** command ordinarily requires principal name and password before letting anyone access the administrative interface. However, **kadmin.local** is a variant of the command that must be run locally on the same machine as the KDC, and with administrator privileges. It is then able to open the Kerberos database file directly (taking advantage of Unix file permissions), without requiring extra privileges and without using the **kadmind** (Kerberos master server) daemon.

The purpose of our running **kadmin.local** will be to print out the list of existing principals (user, host and service accounts) in the database using the command `listprincs`. The whole session should look like this:

```
sudo kadmin.local
Authenticating as principal root/admin@SPINLOCK.HR with password.

kadmin.local:  listprincs

K/M@SPINLOCK.HR
kadmin/admin@SPINLOCK.HR
kadmin/changepw@SPINLOCK.HR
kadmin/krb1.SPINLOCK.HR@SPINLOCK.HR
krbtgt/SPINLOCK.HR@SPINLOCK.HR

kadmin.local:  quit
```

Note

If your output does not say `kadmin/krb1.SPINLOCK.HR@SPINLOCK.HR` but it says `kadmin/YOUR_HOSTNAME.SPINLOCK.HR@SPINLOCK.HR`, then edit `/etc/hosts` to verify and make sure that *YOUR_HOSTNAME* —if it is listed there — appears associated to a network IP of the machine, and not to its local IP (127.*).

In other words, something like:

```
127.0.0.1       localhost
127.0.1.1       ubuntu.spinlock.hr ubuntu
192.168.7.12    krb1.spinlock.hr krb1 monarch.spinlock.hr monarch
```

Would need to be adjusted to:

```
127.0.0.1       localhost
192.168.7.12    ubuntu.spinlock.hr ubuntu krb1.spinlock.hr krb1 monarch.spinlock.hr monarch
```

Principal Names

In the test step above, you might have noticed principal names similar to `kadmin/admin@SPINLOCK.HR`. The general naming syntax for principals is *SPEC*`@`*REALM*, where *SPEC* by convention consists of components separated by "`/`", and the default *REALM* can be omitted.

In the case of principals related to system users, the first component identifies the user name, and the second component, if present, identifies user role. For regular users, there will usually be one principal with no special role, named simply *USERNAME*. But when administrative or other roles are required, there will be no need to condense them all to one "`root`" prinicpal — each user can simply be given conveniently named additional principals with special privileges, such as *USERNAME*`/admin`.

In the case of principals related to system services, the components will be used to identify service and hostname, such as `host/`*monarch.spinlock.hr* or `ldap/`*monarch.spinlock.hr*. ("host" is somewhat of a misnomer from today's perspective — it has nothing to do with host per-se, but is actually the service name for all remote shell protocols, such as rsh, rlogin and ssh).

Access rights

Let's take a look at the `/etc/krb5kdc/kadm5.acl` file; it defines user access rights for the Kerberos database. For regular users with no special privileges, no action will be required. For admin users, we will want to grant all privileges, as hinted earlier in the section called “Principal Names”. To do this, make sure the following line is present in the file and enabled (that is, without the comment '`#`' character at the beginning):

```
*/admin *
```

(While the above syntax might remind you of shell globbing, it does not work that way. The only matching character supported is the asterisk ("`*`"), it does not match multiple components, and it can only be used in form of "*component1*`/*`" or "`*/`*component2*".)

Make sure to restart the admin server to apply `/etc/krb5kdc/kadm5.acl` changes:

```
sudo invoke-rc.d krb5-admin-server restart
```

Kerberos policies

Kerberos "policies" offer an elegant way to sort principals into a kind of categories and to automatically apply corresponding defaults onto newly created principals.

Let's create four basic policies: for admins, hosts, services and users. In this example, each policy will define minimum password strength (measured in number of character classes present in the password, from 1 to 5), but a few other options can be set — run **addpol** (the supported abbreviation of add_policy) if you're curious.

```
sudo kadmin.local
Authenticating as principal root/admin@SPINLOCK.HR with password.

kadmin.local:  add_policy -minlength 8 -minclasses 3 admin
kadmin.local:  add_policy -minlength 8 -minclasses 4 host
kadmin.local:  add_policy -minlength 8 -minclasses 4 service
kadmin.local:  add_policy -minlength 8 -minclasses 2 user

kadmin.local:  quit
```

Creating first privileged principal

As you might have noticed, the **kadmin.local** command identified us as the principal `root/admin`. Still, that principal does not actually exist in the database so we might as well create it now. Once the principal is actually there, we'll be able to connect to the administrative server using **kadmin** from any machine within the Kerberos realm, and not just by using **kadmin.local** on the Kerberos server.

Creating a principal based on your regular identity (such as *USERNAME*`/admin`) is preferred over creating one called `root/admin`, but we will create `root/admin` for simplicity:

```
sudo kadmin.local
Authenticating as principal root/admin@SPINLOCK.HR with password.

kadmin.local:  addprinc -policy admin root/admin

Enter password for principal "root/admin@SPINLOCK.HR":  PASSWORD
Re-enter password for principal "root/admin@SPINLOCK.HR":  PASSWORD
Principal "root/admin@SPINLOCK.HR" created.

kadmin.local:  quit
```

Kadmin test

Now that the `root/admin` principal exists in the Kerberos database, we should be able to use **kadmin** just as we used **kadmin.local**. The only exception, of course, is that **kadmin** will prompt for a password to connect to the Kerberos admin server.

Double-check that all the permissions are granted to admin roles in the `/etc/krb5kdc/kadm5.acl` (as explained in the section called “Access rights”), and that the admin server has been restarted to read the new configuration; then proceed to test **kadmin** connection:

```
sudo kadmin -p root/admin
Authenticating as principal root/admin@SPINLOCK.HR with password.

Password for root/admin@SPINLOCK.HR:  PASSWORD
```

```
kadmin:  listprincs

K/M@SPINLOCK.HR
root/admin@SPINLOCK.HR
kadmin/admin@SPINLOCK.HR
kadmin/changepw@SPINLOCK.HR
kadmin/history@SPINLOCK.HR
kadmin/krb1.SPINLOCK.HR@SPINLOCK.HR
krbtgt/SPINLOCK.HR@SPINLOCK.HR

kadmin:  quit
```

If there is a noticeable delay present before the kadmin password prompt appears, or if you notice a "SERVER_NOT_FOUND" warning printed to /var/log/kerberos/krb5kdc.log, look up the section called “Error: SERVER_NOT_FOUND” for a solution.

Creating first unprivileged principal

Let's add a principal that will correspond to your regular, unprivileged user account. In our example, the username will be called "mirko". We've essentially performed this procedure for the root/admin principal above, but we'll repeat it here for your regular user account, using a different policy, and replacing *mirko* with your username.

```
sudo kadmin -p root/admin
Authenticating as principal root/admin@SPINLOCK.HR with password.

Password for root/admin@SPINLOCK.HR: PASSWORD

kadmin:  addprinc -policy user mirko

Enter password for principal "mirko@SPINLOCK.HR": PASSWORD
Re-enter password for principal "mirko@SPINLOCK.HR": PASSWORD
Principal "mirko@SPINLOCK.HR" created.

kadmin:  quit
```

Obtaining Kerberos ticket

As hinted in the introduction, each user is expected to type in the password once, to obtain the initial TGT (Ticket-granting Ticket). Obtained tickets are saved to a so-called *ticket cache*, which is most commonly a file named /tmp/krb5cc_*, stored on the user's workstation.

Let's run the **klist** command to inspect our ticket cache (run this command under your regular, non-privileged username). As one might guess, since we did not obtain any tickets yet, the cache will be empty:

```
klist -f

klist: No credentials cache found (ticket cache FILE:/tmp/krb5cc_0)
```

Let's use **kinit** now to obtain the ticket, and then re-inspect the ticket cache. If the command seemingly "hangs" and does nothing, wait a few seconds — DNS misconfiguration may cause a delay.

```
kinit mirko

Password for mirko@SPINLOCK.HR: PASSWORD
```

```
klist -f

Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: mirko@SPINLOCK.HR

Valid starting     Expires            Service principal
11/22/06 22:30:36  11/23/06 08:30:33  krbtgt/SPINLOCK.HR@SPINLOCK.HR
renew until 11/23/06 22:30:34, Flags: FPRIA
```

If you remember the story from the beginning, you will recognize the "krbtgt" to be the Ticket-granting Ticket.

The meanings of each flag letter produced by the **klist** switch -f are not important at this stage, but long-term it is useful to get into the habit of using -f, and the flag descriptions can be looked up in the manpage klist(1).

All great. Let's run **kdestroy** to terminate the ticket now.

Installing kerberized services

To actually use Kerberos, we need to install kerberized versions of the standard services.

Each service may support Kerberos authentication either by having native Kerberos support, or by delegating the authentication work to the PAM subsystem (and since all relevant services support PAM, this means it is possible to Kerberize all network services).

In the APT repository you will find packages like krb-ftpd, krb5-telnetd and krb5-rsh-server. Those are replacement services for ftp, telnet and rsh with native Kerberos (and encryption) support.

Caution

The mentioned krb-ftpd, krb5-telnetd and krb5-rsh-server are still part of Kerberos for traditional reasons, but they contain known implementation flaws and their removal has been discussed a couple of times. We will use still use krb5-rsh-server here because that is the most straight-forward during learning phase, but removing krb5-rsh-server and setting up ssh is covered in later chapters of this Guide.

So let's install krb5-rsh-server and ensure that it runs. It is started from **inetd**, so **inetd** must be installed and running as well, and the kshell and eklogin services need to be enabled in /etc/inetd.conf:

```
sudo apt-get install openbsd-inetd krb5-rsh-server

sudo update-rc.d -f openbsd-inetd remove
sudo update-rc.d openbsd-inetd defaults

sudo update-inetd --enable kshell
sudo update-inetd --enable eklogin

sudo invoke-rc.d openbsd-inetd restart
```

To connect to a certain service, the service must have a corresponding principal in the Kerberos database. This is because the Kerberos server acts as a trusted 3rd party and performs mutual authentication of the user and the service as explained in the section called “The role of Kerberos within a network”. The generic service name for telnet, rsh, ssh and related protocols is "`host`", so let's create the necessary principal with a randomly-generated password. Make sure that you export the key to a keytab file as shown (that is, within the same invocation of **kadmin**) to save yourself from getting an error about the "Key version number" being incorrect:

```
sudo kadmin -p root/admin
Authenticating as principal root/admin@SPINLOCK.HR with password.

Password for root/admin@SPINLOCK.HR: PASSWORD

kadmin:  addprinc -policy service -randkey host/monarch.spinlock.hr

Principal "host/monarch.spinlock.hr@SPINLOCK.HR" created.

kadmin:  ktadd -k /etc/krb5.keytab -norandkey host/monarch.spinlock.hr

Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/etc/krb5.k
Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type arcfour-hmac added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type des3-cbc-sha1 added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type des-cbc-crc added to keytab WRFILE:/etc/krb5.keytab.

kadmin:  quit
```

Finally, ensure that this host's keys are **also** present on the rsh server machine:
If the rsh server and the Kerberos server are the same machine, no additional key export is necessary.
If they are separate machines, log in to the rsh server and connect to **kadmind** to export the key into the local keytab:

```
kadmin -p root/admin

ktadd -k /etc/krb5.keytab -norandkey host/monarch.spinlock.hr

Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/etc/krb5.k
Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type arcfour-hmac added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type des3-cbc-sha1 added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/monarch.spinlock.hr with kvno 2, encryption type des-cbc-crc added to keytab WRFILE:/etc/krb5.keytab.
```

Installing kerberized clients

Let's install kerberized versions of the basic client programs:

```
sudo apt-get install krb5-clients
```

One of the client programs, **krb5-rsh**, will allow you connect to the secure rsh server, automatically and without asking for any user names or passwords. The connection will also be encrypted as long as the `-x` switch is used.

As we have taken care of all the pre-requisites, we can try connecting:

Obtain Kerberos ticket:

```
kinit mirko

Password for mirko@SPINLOCK.HR: PASSWORD
```

Connect:

```
krb5-rsh -x -PN krb1.spinlock.hr

This rlogin session is encrypting all data transmissions.
Last login: Mon Nov 27 16:49:49 from monarch
Linux monarch 2.4.27-2-686 #1 Mon May 16 17:03:22 JST 2005 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.

logout
Connection closed.
```

Congratulations! You have a working Kerberos setup.

If anything is not working, proceed right to the section called “Troubleshooting Kerberos connection” — it contains an extensive list of possible errors and the corresponding solutions!

If everything is working, then you can skip that section for now and head directly to the section called “PAM configuration”.

Troubleshooting Kerberos connection

Error: Trying krb4 rlogin... krb_sendauth failed: You have no tickets cached

```
krb5-rsh -x -PN krb1.spinlock.hr

Trying krb4 rlogin...
krb_sendauth failed: You have no tickets cached
```

You have no valid Kerberos tickets, which can be verified by running **klist** (the output will either be empty or show expired tickets). Obtain a new ticket using **kinit**:

```
kinit PRINCIPAL_NAME
```

Error: Connection Refused

```
krb5-rsh -PN krb1.spinlock.hr

connect to address 192.168.7.12: Connection refused
Trying krb4 rlogin...
connect to address 192.168.7.12: Connection refused
trying normal rlogin (/usr/bin/netkit-rlogin)
exec: No such file or directory
```

Let's take a look at this. First of all, you can see that **krb5-rsh** has some fallbacks built-in. It first tries to connect using the Kerberos 5 protocol, then Kerberos 4, and then using the normal, non-kerberized rsh. We are only interested in the krb5 result. If any of the other two methods succeed (the krb4 or plain rsh), it's still not what we want (and you will probably want to disable them somehow, because no one setting up a new Kerberos realm in the 21st century should be running either krb4 or unprotected rsh).

So where's the problem? Assuming that you did everything right (installed krb5-rsh-server and restarted inetd), the problem is very simple. Namely, by default, kerberized servers in Debian do not accept unencrypted connections! So, on next attempt, add `-x` on the command line.

```
krb5-rsh -PN -x krb1.spinlock.hr
```

Error: Server not found in Kerberos database

```
krb5-rsh -PN -x krb1.spinlock.hr

error getting credentials: Server not found in Kerberos database
```

As explained in the section called “The role of Kerberos within a network”, both the users and the services must have an appropriate principal entry in the Kerberos database. While users are in form of *NAME/ROLE*, services are in form *SERVICE-NAME/HOSTNAME*. So we need to add a principal for service "host" (common name for all shell services), on host where the service is provided — *monarch.spinlock.hr*. (Strictly, the service is provided on the server, on krb1.*spinlock.hr*, but in a single-machine setup, the hostname's FQDN returns *monarch.spinlock.hr* so we must use that).

Within the same session, you will almost always want to export that principal's key to a keytab file. Exporting will not work as intended if the key was not created in a single **kadmin** session, so the below solution deletes the existing key (if any), creates it anew and exports it to a file. As to what you need to do with the keytab file after creation — you need to move it from the Kerberos server onto the machine providing the service. If that is the same machine, no moving is necessary.

As most of the errors really boil down to this step, we also take care of re-initializing the ticket properly, to minimize the chance of a mistake:

```
rm /etc/krb5.keytab

kdestroy

sudo kadmin.local
Authenticating as principal root/admin@SPINLOCK.HR with password.

kadmin.local:  delprinc host/monarch.spinlock.hr
Are you sure you want to delete the principal "host/monarch.spinlock.hr@SPINLOCK.HR"? (yes/no): yes
Principal "host/monarch.spinlock.hr@SPINLOCK.HR" deleted.
Make sure that you have removed this principal from all ACLs before reusing.

kadmin.local:  addprinc -randkey host/monarch.spinlock.hr

WARNING: no policy specified for host/monarch.spinlock.hr@SPINLOCK.HR; defaulting to no policy
Principal "host/monarch.spinlock.hr@SPINLOCK.HR" created.

kadmin.local:  ktadd -k /etc/krb5.keytab -norandkey host/monarch.spinlock.hr

kadmin.local: quit

kinit root/admin
```

Error: No such file or directory

```
krb5-rsh -PN -x krb1.spinlock.hr

Couldn't authenticate to server: Server rejected authentication (during sendauth exchange)
Server returned error code 60 (Generic error (see e-text))
Error text sent from server: No such file or directory
```

The above error indicates that we should pay attention to the "e-text" (error text returned to the client). The error text tells us, in kind of a confusing way (since — you see — there is no filename reported), that the `/etc/krb5.keytab` file on the rsh server is missing altogether. This file needs to exist and contain the service key. The way to obtain the file and the key is to follow the recipe from the section called “Error: Server not found in Kerberos database”.

Error: Key table entry not found

```
krb5-rsh -PN -x krb1.spinlock.hr

Couldn't authenticate to server: Server rejected authentication (during sendauth exchange)
Server returned error code 60 (Generic error (see e-text))
Error text sent from server: Key table entry not found
```

The server did accept the connection, but the e-text "Key table entry not found" indicates that the service principal (created earlier, `host/monarch.spinlock.hr`) is not listed in the keytab file on rsh server. Follow the recipe in the section called “Error: Server not found in Kerberos database”.

Error: Key version number for principal in key table is incorrect

```
krb5-rsh -PN -x krb1.spinlock.hr

Couldn't authenticate to server: Server rejected authentication (during sendauth exchange)
Server returned error code 60 (Generic error (see e-text))
Error text sent from server: Key version number for principal in key table is incorrect
```

The service key has changed on the Kerberos server, and the service did not succeed in proving its identity to the Kerberos server — the file `/etc/krb5.keytab` on the service did not contain the correct key. Have in mind that the key changes if you run `ktadd` from within the **kadmin** shell, and the only way to prevent that from happening is to use **kadmin.local** interface and use `ktadd -norandkey` in it. If curious, read up on `ktadd` behavior in kadmin(8). Follow the recipe in the section called “Error: Server not found in Kerberos database”.

Error: Client not found in Kerberos database while getting initial credentials

```
kinit root/admin

kinit(v5): Client not found in Kerberos database while getting initial credentials
```

This is Kerberos way of saying "User not found". You either misspelled the principal name ("`root/admin`" in this case), or you didn't add the principal to the kerberos database in the first place. Adding a principal is performed using the **addprinc** command as shown in the section called “Creating first privileged principal” or the section called “Creating first unprivileged principal”.

Error: Client not found in Kerberos database while initializing kadmin interface

```
sudo kadmin -p root/admin

kadmin: Client not found in Kerberos database while initializing kadmin interface
```

This is Kerberos way of saying "User not found". You either misspelled the principal name ("`root/admin`" in this case), or you didn't add the principal to the kerberos database in the first place. Adding a principal is performed using the **addprinc** command as shown in the section called “Creating first privileged principal” or the section called “Creating first unprivileged principal”.

Error: Decrypt integrity check failed

```
krb5-rsh -PN -x krb1.spinlock.hr

Couldn't authenticate to server: Server rejected authentication (during sendauth exchange)
Server returned error code 31 (Decrypt integrity check failed)
Error text sent from server: Decrypt integrity check failed
```

This is Kerberos way of saying "Password incorrect". In this case, it means that the service key changed on the server, and your your ticket cache no longer contains the ticket with the correct key. Running `kdestroy; kinit` should obtain a new ticket and solve the problem.

Error: Unsupported key table format version number while adding key to keytab

```
kadmin: ktadd -k /etc/krb5.keytab host/monarch.spinlock.hr

kadmin: Unsupported key table format version number while adding key to keytab
```

This usually happens when the local file to which you want to export the key (`/etc/krb5.keytab`) is in an incorrect format.

The most common reason why this would happen is if you have tried to create an empty file (using **touch** or similar commands) beforehand, and then export the key into it.

To verify that this is indeed the case, try running **klist** on the existing file to which you are attempting to export the key:

```
sudo klist -k /etc/keytab

klist: Unsupported key table format version number while starting keytab scan
```

The solution is to delete the incorrectly created keytab file, or to choose a different keytab file to which the intended key should be exported.

Error: Wrong principal in request

```
krb5-rsh -x -PN krb1.spinlock.hr

Couldn't authenticate to server: Server rejected authentication (during sendauth exchange)
Server returned error code 60 (Generic error (see e-text))
Error text sent from server: Wrong principal in request
```

Error: SERVER_NOT_FOUND

```
sudo kadmin -p root/admin

==> kerberos/krb5kdc.log <==
Jan 07 01:47:35 ubuntu krb5kdc[20837](info): AS_REQ (4 etypes {18 17 16 23}) 192.168.7.12: SERVER_NOT_FOUND: root/admin@SPINLOCK.HR
```

This error is emitted in the krb5kdc log file when the principal reported (`kadmin/krb1.spinlock.hr@SPINLOCK.HR`) is missing in the Kerberos database.
It usually happens when you are setting up a Kerberos server using a chosen hostname that does not match the hostname reported by the system command **hostname**.

Add the missing kadmin principal as follows:

```
sudo kadmin.local
Authenticating as principal root/admin@SPINLOCK.HR with password.

kadmin.local:  addprinc -randkey -requires_preauth -allow_tgs_req  kadmin/krb1.spinlock.hr

WARNING: no policy specified for kadmin/krb1.spinlock.hr@SPINLOCK.HR; defaulting to no policy
Principal "kadmin/krb1.spinlock.hr@SPINLOCK.HR" created.

kadmin.local:  quit
```

Error: SERVER_NOT_FOUND

```
sudo kadmin -p root/admin

==> kerberos/krb5kdc.log <==
Jan 07 01:47:35 ubuntu krb5kdc[20837](info): AS_REQ (4 etypes {18 17 16 23}) 192.168.7.12: SERVER_NOT_FOUND: root/admin@SPINLOCK.HR
```

This error is emitted in the krb5kdc log file when the principal reported (`kadmin/krb1.spinlock.hr@SPINLOCK.HR`) is missing in the Kerberos database.
It usually happens when you are setting up a Kerberos server using a chosen hostname that does not match the hostname reported by the system command **hostname**.

Add the missing kadmin principal as follows:

```
sudo kadmin.local
Authenticating as principal root/admin@SPINLOCK.HR with password.

kadmin.local:  addprinc -randkey -requires_preauth -allow_tgs_req  kadmin/krb1.spinlock.hr

WARNING: no policy specified for kadmin/krb1.spinlock.hr@SPINLOCK.HR; defaulting to no policy
Principal "kadmin/krb1.spinlock.hr@SPINLOCK.HR" created.

kadmin.local:  quit
```

Error: klogind: not authorized to login to account

```
krb5-rsh -x -PN krb1.spinlock.hr

klogind: User root/admin@SPINLOCK.HR is not authorized to login to account root.
```

This error is emitted when the Kerberos principal name ("`root/admin`") does not exactly match the name of the user account to which it wants to log in to ("`root`"), and when the login allowance for that principal has not been added to file `~/.k5login`.

To add the permission, add the principal's full name to the file `~/.k5login` in the target account's home directory:

```
echo 'root/admin@SPINLOCK.HR' >> ~root/.k5login
```

PAM configuration

The final step in this article pertains to integrating Kerberos into the system authentication procedure. We want Kerberos tickets to be issued for users as they log in, without the need to run **kinit** manually after login.

On GNU/Linux and derivatives, this is done by simply altering Linux-PAM configuration in `/etc/pam.d/` on all machines where the users are logging in.

As we have explained in the section called "The role of Kerberos within a network", Kerberos alone does not help replace the usual password files (`/etc/passwd`, `/etc/shadow` or `/etc/group`). For now, your "kerberized" users will have to be present in both system password files and in Kerberos. (For a solution to that problem, see the next article in the series, the OpenLDAP Guide.)

Our Linux-PAM configuration will be defined so that *either* the usual password authentication *or* Kerberos authentication will need to succeed for the user to log in. This way, both users that will have no Kerberos entry (the system ones, such as `root`, `daemon`, `bin`, `sync`, `sys`, ...) and those that will (regular user accounts), will be able to log in.

System password in `/etc/shadow` will be tried first. If you want Kerberos tickets to be issued, this type of authentication **must fail** for regular users (otherwise their "system login" would succeed — resulting in the Kerberos part being skipped altogether and no tickets issued).

The most common way to make regular users have only one password (and that one being in Kerberos) is to replace their system password in `/etc/shadow` with a literal "`*K*`", which is not a valid password and also by spoken convention indicates that the "real" password is stored in Kerberos. This password can be set either by editing `/etc/shadow` file directly (i.e. with `sudo vipw -s`) or by invoking `sudo usermod -p '*K*' USERNAME`.

Let's install the necessary Kerberos PAM module:

```
sudo apt-get install libpam-krb5
```

Let's configure Linux-PAM. PAM configuration is quite fragile, so use the provided examples that have been verified to work. For any modifications, you will want to look at PAM Configuration File Syntax and pay special attention to seemingly insignificant variations — with PAM, they often make a whole world of difference.

To minimize the chance of locking yourself out of the system during PAM configuration phase, ensure right now that you have at least one root terminal window open and a copy of the files available *before* starting on PAM configuration changes. To do so, execute the following in a cleanly started shell and leave the terminal open:

```
sudo su -
cd /etc
cp -a pam.d pam.d,orig
```

> **Note**
>
> If you break logins with an invalid PAM configuration, the above will allow you to simply revert to a known-good state by using the open root terminal and executing:
>
> ```
> cp -a pam.d,orig/* pam.d/
> ```

The following PAM examples are complete; you should replace your existing configuration with the one shown below:

/etc/pam.d/common-account

```
account [success=1 new_authtok_reqd=done default=ignore]        pam_unix.so
account requisite                       pam_deny.so
account required                        pam_permit.so
account required                        pam_krb5.so minimum_uid=1000
```

/etc/pam.d/common-auth

```
auth    [success=2 default=ignore]      pam_krb5.so minimum_uid=1000
auth    [success=1 default=ignore]      pam_unix.so nullok_secure try_first_pass
auth    requisite                       pam_deny.so
auth    required                        pam_permit.so
```

/etc/pam.d/common-password

```
password        [success=2 default=ignore]      pam_krb5.so minimum_uid=1000
password        [success=1 default=ignore]      pam_unix.so obscure use_authtok try_first_pass sha512
password        requisite                       pam_deny.so
password        required                        pam_permit.so
```

/etc/pam.d/common-session

```
session [default=1]                     pam_permit.so
session requisite                       pam_deny.so
session required                        pam_permit.so
session optional                        pam_krb5.so minimum_uid=1000
session required        pam_unix.so
```

If you have edited PAM configuration manually, restart the services you will be connecting to. This isn't strictly necessary, but it ensures that the services can start properly and will certainly re-read the PAM configuration.

Conclusion

At this point, you have a functional Kerberos installation!

You can rely on either system login or manually running **kinit** in obtaining Kerberos tickets and accessing Kerberized services. One of those services is the passwordless, Kerberos-secured rsh login that we've demonstrated in this guide.

With a good foundation we've built, for further information on Kerberos, please refer to other available resources:

- Official documentation: http://web.mit.edu/kerberos/krb5-1.8/

- Mailing lists: http://web.mit.edu/kerberos/mail-lists.html
- IRC: channel #kerberos at the FreeNode network (irc.freenode.net)
- For commercial consultation and infrastructure-based networks containing Kerberos, contact Spinlock Solutions.

Remember that, as explained in this Guide, your user accounts still need to be created locally on all hosts the users wish to access. To solve that problem and achieve true centralized logins, follow the next article in the series, the OpenLDAP Guide.

If you have followed the OpenLDAP Guide first and have come here to set up Kerberos as an afterthought, run `sudo dpkg-reconfigure libpam-ldap` to choose "Unix authentication" and "Kerberos authentication" instead of "LDAP Authentication", and re-visit the OpenLDAP Guide to verify that the resulting PAM configuration files have actually been re-generated and look like the Kerberos-related examples shown there.

If you have followed this MIT Kerberos 5 Guide only as a pre-requisite for installing OpenAFS and do not want to use LDAP in combination, proceed to another article in the series, the OpenAFS Guide.

Links

Platforms:
GNU
Debian GNU

Kerberos:
MIT Kerberos
Heimdal Kerberos
Kerberos consortium

Kerberos specifics:
Kerberos release
Kerberos database
realm
KDC
principal
secret key
TGT

Glue layer:
Linux-PAM
PAM Configuration File Syntax

Related infrastructural technologies:
OpenLDAP
OpenAFS
FreeRADIUS

Commercial support:
Spinlock Solutions

Misc:
DocBook