NAME

krb5_admin - kerberos administration tool

SYNOPSIS

krb5_admin [**-Mdlv**] [**-D** *kdb*] [**-S** *sqlitedb*] [**-h** *hostspec*] [**-r** *REALM*] *command arg* [*arg ...*]

DESCRIPTION

krb5_admin is used to make modifications to the Kerberos database, either remotely via the krb5_admind(8) daemon, or locally (with the **-l** flag.)

The options are as follows:

-D *kdb* specifies the location of the Kerberos DB when running against a local database. Defaults to the location built into the Kerberos libriares. Implies **-l**.

-M connect to the master KDC rather than a slave.

-S *sqlitedb* specifies the location of the sqlite3 adjunct database where **krb5_admin** stores its additional schemas. Can only be used if running against a local database.

-d print debugging output.

-h *hostspec* connect to *hostspec* instead of searching for KDCs using the usual method. A *hostspec* has a format of [*service@*]*hostname*[:*port*].

-r *REALM* connect to one of the KDCs for realm *REALM*.

-l operate on the local Kerberos DB.

-v make the output a bit more verbose.

If no *command* is provided on the command line, **krb5_admin** will prompt for commands to process. (This behaviour is deprecated as the quoting is sub-optimal.)

The remaining arguments to **krb5_admin** are accepted from the shell and no further quoting is performed. The first argument is considered to be the command and the remaining arguments are passed to the command. Some commands accept a list of key value pairs starting after a particular position. Some keys are denoted as ‘‘set-based’’ meaning that they manage a set of values rather than an individual setting. Set-based keys take a comma-separated list of values (interpreted as a set) and can also take plus equals (+=) or minus equals (-=) to add or subtract elements from the set.

Authorisation to make a change is checked both before and after any action. If the owernsip is changed such that the actor may not revert the change, then the change is not allowed. This means that you cannot relinquish access to any of the objects that **krb5_admin** manages. This has the nice property that the user can always undo any action that they just performed. The downside is that transferring ownership becomes a two step process where the original owner adds a new owner and then the new owner removes the old owner.

General commands:

master
outputs the name of the master KDC. Note, this will actually connect to the master KDC and will fail if the master KDC is unavailable.

Principals and Appids

Principals are simply principals stored in the Kerberos DB. Appids are non-human identities used to run batch jobs. Appids are both a Kerberos principal and some ancilliary data which is used to enable their management.

The following keys are defined for all principals: princ_expire_time, pw_expiration, max_life, max_renewable_life, and attributes. The following keys are also defined for appids: owner (set-based), desc, cstraint (set-based).

Commands that operate on principals and appids:

list [*glob*]
will list the principals that match the *glob*.

query *princ*
will display the principal *princ*.

mquery *glob* [*glob* ...]
Outputs a list of principals.

remove *princ*
will remove the principal *princ*.

enable *princ*
will enable the principal *princ* by removing the attribute **-allow_tix**.

disable *princ*

will disable the principal *princ* by adding the attribute **-allow_tix**.

create_appid *princ* [*key=val* ...]
will create a principal for use by automated processes. This principal will have no assigned passwd and will have a default owner set to the calling user. The key value pairs are the same as for *modify*.

create_user *princ*
will create a principal with a random temporary password which is printed to stdout. The password must be reset by the user via kpasswd(1).

reset_passwd *princ*
will reset the password of a principal to a random temporary password which is printed to stdout. The password must be reset by the user via kpasswd(1).

modify *princ* [*key=val* ...]
will modify a principal or its associated attributes.

is_owner *princ appid*
tests whether *princ* is one of the owners of *appid*. This test follows group membership. There is no output, the return value is set.

Policies

Commands that operate on policies:

listpols [*glob*]
will list the policies that match *glob*.

Subjects and Groups

Subjects are simply a list of allowable principals which can be used in ownerships relationships. **krb5_admin** ensures that all users and appids that are created will have an associated krb5 subject in the database, but if you want to refer to principals in foreign realms in your ownership relationships then you will need to add them using *create_subject*.

Groups are just groups of subjects. Groups can contain either krb5 principals or other groups nested to sixteen (16) levels. Groups are simply subjects where the type is "group" and they are allowed to have "members". The commands *create_group*, *modify_group*, etc. are equivalent to their _subject variants except that they set type=group automatically.

The following keys are defined for subjects: type, owner (set-based), and member (set-based).

Commands that operate on subjects:

create_subject *subj* type=*type* [key=*val* ...]
will create a subject *subj* of type *type*.

list_subject [key=*val* ...]
will list all of the subjects which satisfy the conditions specified.

modify_subject *subj* [key=*val* ...]
will modify the attributes of a subject.

query_subject *subj* [field ...]
will display the fields of *subj*. If the optional field parameters are supplied then only the fields specified will be displayed.

remove_subject *subj*
will remove *subj*.

Commands that operate on groups:

create_group *group* type=*type* [key=*val* ...]
will create a group *group* of type *type*.

list_group [key=*val* ...]
will list all of the groups which satisfy the conditions specified.

modify_group *group* [key=*val* ...]
will modify the attributes of a group.

query_group *group* [field ...]
will display the fields of *group*. If the optional field parameters are supplied then only the fields specified will be displayed.

remove_group *group*
will remove *group*.

Hosts

krb5_admin needs to keep track of all of the hosts in an environment. This information is used to support bootstrapping of initial credentials and for the deployment of prestashed tickets.

The following keys are defined for all hosts: realm, ip_addr, bootbinding, and owner (set-based).

Commands that operate on hosts:

create_host *host* realm=*REALM* [key=val ...]
Create a host in the krb5_admin database with the given realm and bootbinding. The realm is used for prestashed ticket access control and is a required parameter. The remaining key value pairs are the same as for *modify_host*.

create_logical_host
Create a logical host. This commands works the same as *create_host* except the host created is marked as a logical host which means that it is either an alias to an existing host or a cluster of hosts.

bind_host *host principal*
Bind an existing *host* to the given *principal*, this entitles the host to negotiate its initial keys. This function can also be accomplished using "**krb5_admin** *modify_host host bootbinding=princ*". This function may have different authorisation rules, though.

remove_host *host*
Remove *host*.

modify_host *host* [key=val ...]
will modify the attributes of a host.

Labels

Lables are placed on hosts to help constrain where prestashed tickets are allowed to be placed. When tickets are asked to be placed on a host via krb5_prestash, it is required that the host has a label matching each of the "cstraints" defined for the appid. These commands are simply to manage the list of acceptable labels, to actually set labels on hosts see the "Hosts" sub-section and to set the "cstraints" on an appid see the "Principals and Appids" sub-section.

The following commands work on labels:

add_label
adds *label*.

del_label
removes *label*.

list_labels

lists all of the valid labels.

Features

Features are simply a set of flags defined in the **krb5_admin** database which can be tested by clients to determine if certain features have been enabled at a site. No generic features have been defined, yet.

Commands that operate on ‘‘features’’:

add_feature *feature*
Adds a ‘‘feature’’ flag.

del_feature *feature*
Deletes a ‘‘feature’’ flag.

has_feature *feature*
Check to see if ‘‘feature’’ is present, i.e. has been added.

SACLS

SACLS are Simple Access Control Lists. SACLS are usually used to provide administrative access to certain functions in **krb5_admin**. When a SACL is set for a principal, the principal can execute the command with any arguments. Because of this, these are a heavy hammer which should be used with some level of discretion and they are designed mainly for either administrators or synchronisation jobs which source information such as host names from an upstream source.

Commands that operate on SACLS:

sacls_add *verb actor*
grants *verb* to *actor*.

sacls_del *verb actor*
revokes *verb* from *actor*.

sacls_query
lists all of the SACLS.

EXIT STATUS

The **krb5_admin** utility normally exits 0 on success, and exits 1 on failure.

EXAMPLES

To create an appid *webserver*:

```
$ krb5_admin create_appid webserver
```

To add *elric@IMRRYR.ORG* to the list of owners of the appid *webserver*:

```
$ krb5_admin modify webserver owner+=elric@IMRRYR.ORG
```

To list all of the principals that begin with web:

```
$ krb5_admin list web
```

Show a host:

```
$ krb5_admin query_host foo.example.com
```

To change the owners of a host:

```
$ krb5_admin modify_host foo.example.com owner+=elric@IMRRYR.ORG
$ krb5_admin modify_host foo.example.com owner-=yyrkoon@IMRRYR.ORG
```

SEE ALSO

knc(1), krb5_admind(8), krb5_keytab(8), krb5_prestash(1).