Kerberos Concepts

Release 1.11.5

MIT

CONTENTS

CHAPTER

ONE

CREDENTIAL CACHE

A credential cache (or "ccache") holds Kerberos credentials while they remain valid and, generally, while the user's session lasts, so that authenticating to a service multiple times (e.g., connecting to a web or mail server more than once) doesn't require contacting the KDC every time.

A credential cache usually contains one initial ticket which is obtained using a password or another form of identity verification. If this ticket is a ticket-granting ticket, it can be used to obtain additional credentials without the password. Because the credential cache does not store the password, less long-term damage can be done to the user's account if the machine is compromised.

A credentials cache stores a default client principal name, set when the cache is created. This is the name shown at the top of the *klist(1) -A* output.

Each normal cache entry includes a service principal name, a client principal name (which, in some ccache types, need not be the same as the default), lifetime information, and flags, along with the credential itself. There are also other entries, indicated by special names, that store additional information.

1.1 ccache types

The credential cache interface, like the *keytab* and *replay cache* interfaces, uses *TYPE:value* strings to indicate the type of credential cache and any associated cache naming data to use.

There are several kinds of credentials cache supported in the MIT Kerberos library. Not all are supported on every platform. In most cases, it should be correct to use the default type built into the library.

1. **API** is only implemented on Windows. It communicates with a server process that holds the credentials in memory for the user, rather than writing them to disk.
2. **DIR** points to the storage location of the collection of the credential caches in *FILE:* format. It is most useful when dealing with multiple Kerberos realms and KDCs. For release 1.10 the directory must already exist. In post-1.10 releases the requirement is for parent directory to exist and the current process must have permissions to create the directory if it does not exist. See *Collections of caches* for details. New in release 1.10.
3. **FILE** caches are the simplest and most portable. A simple flat file format is used to store one credential after another. This is the default ccache type.
4. **KEYRING** is Linux-specific, and uses the kernel keyring support to store credential data in unswappable kernel memory where only the current user should be able to access it. The following residual forms are supported:
    - KEYRING:name
    - KEYRING:process:name - process keyring
    - KEYRING:thread:name - thread keyring

Starting with release 1.12 the *KEYRING* type supports collections. The following new residual forms were added:

- KEYRING:session:name - session keyring
- KEYRING:user:name - user keyring
- KEYRING:persistent:uidnumber - persistent per-UID collection. Unlike the user keyring, this collection survives after the user logs out, until the cache credentials expire. This type of ccache requires support from the kernel; otherwise, it will fall back to the user keyring.

See *Collections of caches* for details.

5. **MEMORY** caches are for storage of credentials that don't need to be made available outside of the current process. For example, a memory ccache is used by *kadmin(1)* to store the administrative ticket used to contact the admin server. Memory ccaches are faster than file ccaches and are automatically destroyed when the process exits.
6. **MSLSA** is a Windows-specific cache type that accesses the Windows credential store.

1.2 Collections of caches

Some credential cache types can support collections of multiple caches. One of the caches in the collection is designated as the *primary* and will be used when the collection is resolved as a cache. When a collection-enabled cache type is the default cache for a process, applications can search the specified collection for a specific client principal, and GSSAPI applications will automatically select between the caches in the collection based on criteria such as the target service realm.

Credential cache collections are new in release 1.10, with support from the **DIR** and **API** ccache types. In release 1.12, the **KEYRING** ccache type also supports collections.

1.2.1 Tool alterations to use cache collection

- *kdestroy(1) -A* will destroy all caches in the collection.
- If the default cache type supports switching, *kinit(1) princname* will search the collection for a matching cache and store credentials there, or will store credentials in a new unique cache of the default type if no existing cache for the principal exists. Either way, kinit will switch to the selected cache.
- *klist(1) -l* will list the caches in the collection.
- *klist(1) -A* will show the content of all caches in the collection.
- *kswitch(1) -p princname* will search the collection for a matching cache and switch to it.
- *kswitch(1) -c cachename* will switch to a specified cache.

1.3 Default ccache name

The default credential cache name is determined by the following, in descending order of priority:

1. The **KRB5CCNAME** environment variable. For example, `KRB5CCNAME=DIR:/mydir/`.
2. The **default_ccache_name** profile variable in *libdefaults*.
3. The hardcoded default, *DEFCCNAME*.

CHAPTER

TWO

KEYTAB

A keytab (short for "key table") stores long-term keys for one or more principals. Keytabs are normally represented by files in a standard format, although in rare cases they can be represented in other ways. Keytabs are used most often to allow server applications to accept authentications from clients, but can also be used to obtain initial credentials for client applications.

Keytabs are named using the format *type*:*value*. Usually *type* is `FILE` and *value* is the absolute pathname of the file. Other possible values for *type* are `SRVTAB`, which indicates a file in the deprecated Kerberos 4 srvtab format, and `MEMORY`, which indicates a temporary keytab stored in the memory of the current process.

A keytab contains one or more entries, where each entry consists of a timestamp (indicating when the entry was written to the keytab), a principal name, a key version number, an encryption type, and the encryption key itself.

A keytab can be displayed using the *klist(1)* command with the `-k` option. Keytabs can be created or appended to by extracting keys from the KDC database using the *kadmin(1) ktadd* command. Keytabs can be manipulated using the *ktutil(1)* and *k5srvutil(1)* commands.

2.1 Default keytab

The default keytab is used by server applications if the application does not request a specific keytab. The name of the default keytab is determined by the following, in decreasing order of preference:

1. The **KRB5_KTNAME** environment variable.
2. The **default_keytab_name** profile variable in *libdefaults*.
3. The hardcoded default, *DEFKTNAME*.

2.2 Default client keytab

The default client keytab is used, if it is present and readable, to automatically obtain initial credentials for GSSAPI client applications. The principal name of the first entry in the client keytab is used by default when obtaining initial credentials. The name of the default client keytab is determined by the following, in decreasing order of preference:

1. The **KRB5_CLIENT_KTNAME** environment variable.
2. The **default_client_keytab_name** profile variable in *libdefaults*.
3. The hardcoded default, *DEFCKTNAME*.

CHAPTER

THREE

REPLAY CACHE

A replay cache (or "rcache") keeps track of all authenticators recently presented to a service. If a duplicate authentication request is detected in the replay cache, an error message is sent to the application program.

The replay cache interface, like the credential cache and *keytab* interfaces, uses *type:value* strings to indicate the type of replay cache and any associated cache naming data to use.

3.1 Background information

Some Kerberos or GSSAPI services use a simple authentication mechanism where a message is sent containing an authenticator, which establishes the encryption key that the client will use for talking to the service. But nothing about that prevents an eavesdropper from recording the messages sent by the client, establishing a new connection, and re-sending or "replaying" the same messages; the replayed authenticator will establish the same encryption key for the new session, and the following messages will be decrypted and processed. The attacker may not know what the messages say, and can't generate new messages under the same encryption key, but in some instances it may be harmful to the user (or helpful to the attacker) to cause the server to see the same messages again a second time. For example, if the legitimate client sends "delete first message in mailbox", a replay from an attacker may delete another, different "first" message. (Protocol design to guard against such problems has been discussed in **RFC 4120**.)

Even if one protocol uses further protection to verify that the client side of the connection actually knows the encryption keys (and thus is presumably a legitimate user), if another service uses the same service principal name, it may be possible to record an authenticator used with the first protocol and "replay" it against the second.

The replay cache mitigates these attacks somewhat, by keeping track of authenticators that have been seen until their five-minute window expires. Different authenticators generated by multiple connections from the same legitimate client will generally have different timestamps, and thus will not be considered the same.

This mechanism isn't perfect. If a message is sent to one application server but a man-in-the-middle attacker can prevent it from actually arriving at that server, the attacker could then use the authenticator (once!) against a different service on the same host. This could be a problem if the message from the client included something more than authentication in the first message that could be useful to the attacker (which is uncommon; in most protocols the server has to indicate a successful authentication before the client sends additional messages), or if the simple act of presenting the authenticator triggers some interesting action in the service being attacked.

3.2 Default rcache type

There is currently only one implemented kind of replay cache, called **dfl**. It stores replay data in one file, occasionally rewriting it to purge old, expired entries.

The default type can be overridden by the **KRB5RCACHETYPE** environment variable.

The placement of the replay cache file is determined by the following:

1. The **KRB5RCACHEDIR** environment variable;
2. If KRB5RCACHEDIR is unspecified, on UNIX, the library will fall back to the environment variable **TMPDIR**, and then to a temporary directory determined at configuration time such as */tmp* or */var/tmp*; on Windows, it will check the environment variables *TEMP* and *TMP*, and fall back to the directory C:\.

3.3 Performance issues

Several known minor performance issues that may occur when replay cache is enabled on the Kerberos system include: delays due to writing the authenticator data to disk slowing down response time for very heavily loaded servers, and delays during the rewrite that may be unacceptable to high-performance services.

For use cases where replays are adequately defended against for all protocols using a given service principal name, or where performance or other considerations outweigh the risk of replays, the special replay cache type "none" can be specified:

```
KRB5RCACHETYPE=none
```

It doesn't record any information about authenticators, and reports that any authenticator seen is not a replay.

CHAPTER

FOUR

STASH FILE

The stash file is a local copy of the master key that resides in encrypted form on the KDC's local disk. The stash file is used to authenticate the KDC to itself automatically before starting the *kadmind(8)* and *krb5kdc(8)* daemons (e.g., as part of the machine's boot sequence). The stash file, like the keytab file (see *keytab_file*) is a potential point-of-entry for a break-in, and if compromised, would allow unrestricted access to the Kerberos database. If you choose to install a stash file, it should be readable only by root, and should exist only on the KDC's local disk. The file should not be part of any backup of the machine, unless access to the backup data is secured as tightly as access to the master password itself.

Note: If you choose not to install a stash file, the KDC will prompt you for the master key each time it starts up. This means that the KDC will not be able to start automatically, such as after a system reboot.

CHAPTER

FIVE

SUPPORTED DATE AND TIME FORMATS

5.1 Time duration

This format is used to express a time duration in the Kerberos configuration files and user commands. The allowed formats are:

Format	Example	Value
h:m[:s]	36:00	36 hours
NdNhNmNs	8h30s	8 hours 30 seconds
N (number of seconds)	3600	1 hour

Here *N* denotes a number, *d* - days, *h* - hours, *m* - minutes, *s* - seconds.

Note: The time interval should not exceed 2147483647 seconds.

Examples:

```
Request a ticket valid for one hour, five hours, 30 minutes
and 10 days respectively:

  kinit -l 3600
  kinit -l 5:00
  kinit -l 30m
  kinit -l "10d 0h 0m 0s"
```

5.2 getdate time

Some of the kadmin and kdb5_util commands take a date-time in a human-readable format. Some of the acceptable date-time strings are:

	Format	Example
Date	mm/dd/yy month dd, yyyy	07/27/12 Jul 27, 2012
	yyyy-mm-dd	2012-07-27
Absolute time	HH:mm[:ss]pp hh:mm[:ss]	08:30 PM 20:30
Relative time	N tt	30 sec
Time zone	Z z	EST -0400

(See *Abbreviations used in this document*.)

Examples:

```
Create a principal that expires on the date indicated:
    addprinc test1 -expire "3/27/12 10:00:07 EST"
    addprinc test2 -expire "January 23, 2015 10:05pm"
    addprinc test3 -expire "22:00 GMT"
Add a principal that will expire in 30 minutes:
    addprinc test4 -expire "30 minutes"
```

5.3 Absolute time

This rarely used date-time format can be noted in one of the following ways:

Format	Example	Value
yyyymmddhhmmss yyyy.mm.dd.hh.mm.ss	20141231235900 2014.12.31.23.59.00	
yymmddhhmmss	141231235900	One minute before 2015
yy.mm.dd.hh.mm.ss	14.12.31.23.59.00	
dd-month-yyyy:hh:mm:ss	31-Dec-2014:23:59:00	
hh:mm:ss hhmmss	20:00:00 200000	8 o'clock in the evening

(See *Abbreviations used in this document*.)

Example

```
Set the default expiration date to July 27, 2012 at 20:30
default_principal_expiration = 20120727203000
```

5.3.1 Abbreviations used in this document

month : locale's month name or its abbreviation;
dd : day of month (01-31);
HH : hours (00-12);
hh : hours (00-23);
mm : in time - minutes (00-59); in date - month (01-12);
N : number;
pp : AM or PM;
ss : seconds (00-60);
tt : time units (hours, minutes, min, seconds, sec);
yyyy : year;

yy : last two digits of the year;
Z : alphabetic time zone abbreviation;
z : numeric time zone;

Note:

- If the date specification contains spaces, you may need to enclose it in double quotes;
- All keywords are case-insensitive.

INDEX

R