

---

# Building MIT Kerberos

*Release 1.22-final*

**MIT**



# CONTENTS

1	Prerequisites	3
2	Obtaining the software	5
3	Contents	7



This section details how to build and install MIT Kerberos software from the source.



## **PREREQUISITES**

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

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

If you wish to keep a separate build tree, which contains the compiled \*.o file and executables, separate from your source tree, you will need a make program which supports **VPATH**, or you will need to use a tool such as Indir to produce a symbolic link tree for your build tree.





## OBTAINING THE SOFTWARE

The source code can be obtained from MIT Kerberos Distribution page, at <https://kerberos.org/dist/index.html>. The MIT Kerberos distribution comes in an archive file, generally named `krb5-VERSION-signed.tar`, where *VERSION* is a placeholder for the major and minor versions of MIT Kerberos. (For example, MIT Kerberos 1.9 has major version “1” and minor version “9”.)

The `krb5-VERSION-signed.tar` contains a compressed tar file consisting of the sources for all of Kerberos (generally named `krb5-VERSION.tar.gz`) and a PGP signature file for this source tree (generally named `krb5-VERSION.tar.gz.asc`). MIT highly recommends that you verify the integrity of the source code using this signature, e.g., by running:

```
tar xf krb5-VERSION-signed.tar
gpg --verify krb5-VERSION.tar.gz.asc
```

Unpack `krb5-VERSION.tar.gz` in some directory. In this section we will assume that you have chosen the top directory of the distribution the directory `/u1/krb5-VERSION`.

Review the README file for the license, copyright and other specific to the distribution information.



**CONTENTS**

## 3.1 Organization of the source directory

Below is a brief overview of the organization of the complete source directory. More detailed descriptions follow.

<code>appl</code>	Kerberos application client and server programs
<code>ccapi</code>	Credential cache services
<code>clients</code>	Kerberos V5 user programs (See <code>user_commands</code> )
<code>config</code>	Configure scripts
<code>config-files</code>	Sample Kerberos configuration files
<code>include</code>	include files needed to build the Kerberos system
<code>kadmin</code>	Administrative interface to the Kerberos database: <code>kadmin(1)</code> , <code>kdb5_util(8)</code> , <code>ktutil(1)</code> .
<code>kdc</code>	Kerberos V5 Authentication Service and Key Distribution Center
<code>lib</code>	Libraries for use with/by Kerberos V5
<code>plugins</code>	Kerberos plugins directory
<code>po</code>	Localization infrastructure
<code>prototype</code>	Templates files containing the MIT copyright message and a placeholder for the title and description of the file.
<code>kprop</code>	Utilities for propagating the database to replica KDCs <code>kprop(8)</code> and <code>kpropd(8)</code>
<code>tests</code>	Test suite
<code>util</code>	Various utilities for building/configuring the code, sending bug reports, etc.
<code>windows</code>	Source code for building Kerberos V5 on Windows (see <code>windows/README</code> )

### 3.1.1 `lib`

The `lib` directory contain several subdirectories as well as some definition and glue files.

- The `apputils` directory contains the code for the generic network servicing.
- The `crypto` subdirectory contains the Kerberos V5 encryption library.
- The `gssapi` library contains the Generic Security Services API, which is a library of commands to be used in secure client-server communication.
- The `kadm5` directory contains the libraries for the KADM5 administration utilities.
- The Kerberos 5 database libraries are contained in `kdb`.
- The `krb5` directory contains Kerberos 5 API.
- The `rpc` directory contains the API for the Kerberos Remote Procedure Call protocol.

### 3.1.2 util

The util directory contains several utility programs and libraries.

- the programs used to configure and build the code, such as autoconf, Indir, kbuild, reconf, and makedepend, are in this directory.
- the profile directory contains most of the functions which parse the Kerberos configuration files (krb5.conf and kdc.conf).
- the Kerberos error table library and utilities (et);
- the Sub-system library and utilities (ss);
- database utilities (db2);
- pseudo-terminal utilities (pty);
- bug-reporting program send-pr;
- a generic support library support used by several of our other libraries;
- the build infrastructure for building lightweight Kerberos client (collected-client-lib)
- the tool for validating Kerberos configuration files (confvalidator);
- the toolkit for kernel integrators for building krb5 code subsets (gss-kernel-lib);
- source code for building Kerberos V5 on MacOS (mac)
- Windows getopt operations (windows)

## 3.2 Doing the build

### 3.2.1 Building within a single tree

If you only need to build Kerberos for one platform, using a single directory tree which contains both the source files and the object files is the simplest. However, if you need to maintain Kerberos for a large number of platforms, you will probably want to use separate build trees for each platform. We recommend that you look at OS Incompatibilities, for notes that we have on particular operating systems.

If you don't want separate build trees for each architecture, then use the following abbreviated procedure:

```
cd /u1/krb5-VERSION/src
./configure
make
```

That's it!

### 3.2.2 Building with separate build directories

If you wish to keep separate build directories for each platform, you can do so using the following procedure. (Note, this requires that your make program support VPATH. GNU's make will provide this functionality, for example.) If your make program does not support this, see the next section.

For example, if you wish to store the binaries in `tmpbuild` build directory you might use the following procedure:

```
mkdir /u1/tmpbuild
cd /u1/tmpbuild
/u1/krb5-VERSION/src/configure
make
```

### 3.2.3 Building using Indir

If you wish to keep separate build directories for each platform, and you do not have access to a make program which supports VPATH, all is not lost. You can use the `Indir` program to create symbolic link trees in your build directory.

For example, if you wish to create a build directory for solaris binaries you might use the following procedure:

```
mkdir /u1/krb5-VERSION/solaris
cd /u1/krb5-VERSION/solaris
/u1/krb5-VERSION/src/util/indir `pwd` ../src
./configure
make
```

You must give an absolute pathname to `Indir` because it has a bug that makes it fail for relative pathnames. Note that this version differs from the latest version as distributed and installed by the XConsortium with X11R6. Either version should be acceptable.

### 3.2.4 Installing the binaries

Once you have built Kerberos, you should install the binaries. You can do this by running:

```
make install
```

If you want to install the binaries into a destination directory that is not their final destination, which may be convenient if you want to build a binary distribution to be deployed on multiple hosts, you may use:

```
make install DESTDIR=/path/to/destdir
```

This will install the binaries under *DESTDIR/PREFIX*, e.g., the user programs will install into *DESTDIR/PREFIX/bin*, the libraries into *DESTDIR/PREFIX/lib*, etc. *DESTDIR* must be an absolute path.

Some implementations of make allow multiple commands to be run in parallel, for faster builds. We test our Makefiles in parallel builds with GNU make only; they may not be compatible with other parallel build implementations.

### 3.2.5 Testing the build

The Kerberos V5 distribution comes with built-in regression tests. To run them, simply type the following command while in the top-level build directory (i.e., the directory where you sent typed make to start building Kerberos; see *Building within a single tree*):

```
make check
```

On some operating systems, you have to run `make install` before running `make check`, or the test suite will pick up installed versions of Kerberos libraries rather than the newly built ones. You can install into a prefix that isn't in the system library search path, though. Alternatively, you can configure with `--disable-rpath`, which renders the build tree less suitable for installation, but allows testing without interference from previously installed libraries.

There are additional regression tests available, which are not run by `make check`. These tests require manual setup and teardown of support infrastructure which is not easily automated, or require excessive resources for ordinary use. The procedure for running the manual tests is documented at [https://k5wiki.kerberos.org/wiki/Manual\\_Testing](https://k5wiki.kerberos.org/wiki/Manual_Testing).

### 3.2.6 Cleaning up the build

- Use `make clean` to remove all files generated by running make command.
- Use `make distclean` to remove all files generated by running `./configure` script. After running `make distclean` your source tree (ideally) should look like the raw (just un-tarred) source tree.

### 3.2.7 Using autoconf

(If you are not a developer, you can ignore this section.)

In the Kerberos V5 source directory, there is a configure script which automatically determines the compilation environment and creates the proper Makefiles for a particular platform. This configure script is generated using autoconf, which you should already have installed if you will be making changes to `src/configure.in`.

Normal users will not need to worry about running autoconf; the distribution comes with the configure script already prebuilt.

The autoconf package comes with a script called `autoreconf` that will automatically run `autoconf` and `autoheader` as needed. You should run `autoreconf` from the top source directory, e.g.:

```
cd /u1/krb5-VERSION/src
autoreconf --verbose
```

## 3.3 Options to *configure*

There are a number of options to configure which you can use to control how the Kerberos distribution is built.

### 3.3.1 Most commonly used options

#### **--help**

Provides help to configure. This will list the set of commonly used options for building Kerberos.

#### **--prefix=PREFIX**

By default, Kerberos will install the package's files rooted at `/usr/local`. If you desire to place the binaries into the directory *PREFIX*, use this option.

#### **--exec-prefix=EXECPREFIX**

This option allows one to separate the architecture independent programs from the host-dependent files (configuration files, manual pages). Use this option to install architecture-dependent programs in *EXECPREFIX*. The default location is the value of specified by **--prefix** option.

#### **--localstatedir=LOCALSTATEDIR**

This option sets the directory for locally modifiable single-machine data. In Kerberos, this mostly is useful for setting a location for the KDC data files, as they will be installed in `LOCALSTATEDIR/krb5kdc`, which is by default `PREFIX/var/krb5kdc`.

#### **--with-netlib[=libs]**

Allows for suppression of or replacement of network libraries. By default, Kerberos V5 configuration will look for `-lnsl` and `-lsocket`. If your operating system has a broken resolver library or fails to pass the tests in `src/tests/resolv`, you will need to use this option.

#### **--enable-dns-for-realm**

Enable the use of DNS to look up a host's Kerberos realm, if the information is not provided in `krb5.conf(5)`. See `mapping_hostnames` for information about using DNS to determine the default realm. DNS lookups for realm names are disabled by default.

#### **--with-system-et**

Use an installed version of the error-table (et) support software, the `compile_et` program, the `com_err.h` header file and the `com_err` library. If these are not in the default locations, you may wish to specify `CPPFLAGS=-I/some/dir` and `LDLAGS=-L/some/other/dir` options at configuration time as well.

If this option is not given, a version supplied with the Kerberos sources will be built and installed along with the rest of the Kerberos tree, for Kerberos applications to link against.

#### **--with-system-ss**

Use an installed version of the subsystem command-line interface software, the `mk_cmds` program, the `ss/ss.h` header file and the `ss` library. If these are not in the default locations, you may wish to specify `CPPFLAGS=-I/some/dir` and `LDLAGS=-L/some/other/dir` options at configuration time as well. See also the **SS\_LIB** option.

If this option is not given, the `ss` library supplied with the Kerberos sources will be compiled and linked into those programs that need it; it will not be installed separately.

#### **--with-system-db**

Use an installed version of the Berkeley DB package, which must provide an API compatible with version 1.85. This option is unsupported and untested. In particular, we do not know if the database-rename code used in the dumpfile load operation will behave properly.

If this option is not given, a version supplied with the Kerberos sources will be built and installed. (We are not updating this version at this time because of licensing issues with newer versions that we haven't investigated sufficiently yet.)

### 3.3.2 Environment variables

#### **CC=COMPILER**

Use *COMPILER* as the C compiler.

#### **CFLAGS=FLAGS**

Use *FLAGS* as the default set of C compiler flags.

#### **CPP=CPP**

C preprocessor to use. (e.g., CPP='gcc -E')

#### **CPPFLAGS=CPPOPTS**

Use *CPPOPTS* as the default set of C preprocessor flags. The most common use of this option is to select certain `#define`'s for use with the operating system's include files.

#### **DB\_HEADER=headername**

If `db.h` is not the correct header file to include to compile against the Berkeley DB 1.85 API, specify the correct header file name with this option. For example, DB\_HEADER=db3/db\_185.h.

#### **DB\_LIB=libs...**

If `-ldb` is not the correct library specification for the Berkeley DB library version to be used, override it with this option. For example, DB\_LIB=-ldb-3.3.

#### **DEFCCNAME=ccachename**

Override the built-in default credential cache name. For example, DEFCCNAME=DIR:/var/run/user/%{USERID}/ccache See `parameter_expansion` for information about supported parameter expansions.

#### **DEFCKTNAME=keytabname**

Override the built-in default client keytab name. The format is the same as for *DEFCCNAME*.

#### **DEFKTNNAME=keytabname**

Override the built-in default keytab name. The format is the same as for *DEFCCNAME*.

#### **LD=LINKER**

Use *LINKER* as the default loader if it should be different from C compiler as specified above.

#### **LDLFLAGS=LDOPTS**

This option informs the linker where to get additional libraries (e.g., `-L<lib dir>`).

#### **LIBS=LDNAME**

This option allows one to specify libraries to be passed to the linker (e.g., `-l<library>`)

#### **PKCS11\_MODNAME=library**

Override the built-in default PKCS11 library name.

#### **SS\_LIB=libs...**

If `-lss` is not the correct way to link in your installed `ss` library, for example if additional support libraries are needed, specify the correct link options here. Some variants of this library are around which allow for Emacs-like line editing, but different versions require different support libraries to be explicitly specified.

This option is ignored if `--with-system-ss` is not specified.

#### **YACC**

The 'Yet Another C Compiler' implementation to use. Defaults to the first program found out of: `'bison -y'`, `'byacc'`, `'yacc'`.

#### **YFLAGS**

The list of arguments that will be passed by default to `$YACC`. This script will default `YFLAGS` to the empty string to avoid a default value of `-d` given by some make applications.



### 3.3.3 Fine tuning of the installation directories

**--bindir=DIR**

User executables. Defaults to EXECPREFIX/bin, where *EXECPREFIX* is the path specified by **--exec-prefix** configuration option.

**--sbindir=DIR**

System admin executables. Defaults to EXECPREFIX/sbin, where *EXECPREFIX* is the path specified by **--exec-prefix** configuration option.

**--sysconfdir=DIR**

Read-only single-machine data such as krb5.conf. Defaults to PREFIX/etc, where *PREFIX* is the path specified by **--prefix** configuration option.

**--libdir=DIR**

Object code libraries. Defaults to EXECPREFIX/lib, where *EXECPREFIX* is the path specified by **--exec-prefix** configuration option.

**--includedir=DIR**

C header files. Defaults to PREFIX/include, where *PREFIX* is the path specified by **--prefix** configuration option.

**--datarootdir=DATAROOTDIR**

Read-only architecture-independent data root. Defaults to PREFIX/share, where *PREFIX* is the path specified by **--prefix** configuration option.

**--datadir=DIR**

Read-only architecture-independent data. Defaults to path specified by **--datarootdir** configuration option.

**--localedir=DIR**

Locale-dependent data. Defaults to DATAROOTDIR/locale, where *DATAROOTDIR* is the path specified by **--datarootdir** configuration option.

**--mandir=DIR**

Man documentation. Defaults to DATAROOTDIR/man, where *DATAROOTDIR* is the path specified by **--datarootdir** configuration option.

### 3.3.4 Program names

**--program-prefix=PREFIX**

Prepend *PREFIX* to the names of the programs when installing them. For example, specifying **--program-prefix=mit-** at the configure time will cause the program named *abc* to be installed as *mit-abc*.

**--program-suffix=SUFFIX**

Append *SUFFIX* to the names of the programs when installing them. For example, specifying **--program-suffix=-mit** at the configure time will cause the program named *abc* to be installed as *abc-mit*.

**--program-transform-name=PROGRAM**

Run `sed -e PROGRAM` on installed program names. (*PROGRAM* is a sed script).

### 3.3.5 System types

**--build=*BUILD***

Configure for building on *BUILD* (e.g., --build=x86\_64-linux-gnu).

**--host=*HOST***

Cross-compile to build programs to run on *HOST* (e.g., --host=x86\_64-linux-gnu). By default, Kerberos V5 configuration will look for “build” option.

### 3.3.6 Optional features

**--disable-option-checking**

Ignore unrecognized --enable/--with options.

**--disable-*FEATURE***

Do not include *FEATURE* (same as --enable-*FEATURE*=no).

**--enable-*FEATURE*[=*ARG*]**

Include *FEATURE* [*ARG*=yes].

**--enable-maintainer-mode**

Enable rebuilding of source files, Makefiles, etc.

**--disable-delayed-initialization**

Initialize library code when loaded. Defaults to delay until first use.

**--disable-thread-support**

Don't enable thread support. Defaults to enabled.

**--disable-rpath**

Suppress run path flags in link lines.

**--enable-athena**

Build with MIT Project Athena configuration.

**--disable-kdc-lookaside-cache**

Disable the cache which detects client retransmits.

**--disable-pkinit**

Disable PKINIT plugin support.

**--disable-aesni**

Disable support for using AES instructions on x86 platforms.

**--enable-asan[=*ARG*]**

Enable building with asan memory error checking. If *ARG* is given, it controls the -fsanitize compilation flag value (the default is “address”).

**--enable-ossfuzz**

Enable building fuzzing targets with OSS-Fuzz build support.

### 3.3.7 Optional packages

**--with-PACKAGE[=ARG]**

Use *PACKAGE* (e.g., `--with-imap`). The default value of *ARG* is *yes*.

**--without-PACKAGE**

Do not use *PACKAGE* (same as `--with-PACKAGE=no`) (e.g., `--without-libedit`).

**--with-size-optimizations**

Enable a few optimizations to reduce code size possibly at some run-time cost.

**--with-system-et**

Use the `com_err` library and `compile_et` utility that are already installed on the system, instead of building and installing local versions.

**--with-system-ss**

Use the `ss` library and `mk_cmds` utility that are already installed on the system, instead of building and using private versions.

**--with-system-db**

Use the Berkeley `db` utility already installed on the system, instead of using a private version. This option is not recommended; enabling it may result in incompatibility with key databases originating on other systems.

**--with-netlib=LIBS**

Use the resolver library specified in *LIBS*. Use this variable if the C library resolver is insufficient or broken.

**--with-hesiod=path**

Compile with Hesiod support. The *path* points to the Hesiod directory. By default Hesiod is unsupported.

**--with-ldap**

Compile OpenLDAP database backend module.

**--with-lmdb**

Compile LMDB database backend module.

**--with-vague-errors**

Do not send helpful errors to client. For example, if the KDC should return only vague error codes to clients.

**--with-crypto-impl=IMPL**

Use specified crypto implementation (e.g., `--with-crypto-impl=openssl`). The default is the native MIT Kerberos implementation `builtin`. The other currently implemented crypto backend is `openssl`. (See `mitK5features`)

**--without-libedit**

Do not compile and link against `libedit`. Some utilities will no longer offer command history or completion in interactive mode if `libedit` is disabled.

**--with-readline**

Compile and link against GNU `readline`, as an alternative to `libedit`.

**--with-system-vert**

Use an installed version of `libverto`. If the `libverto` header and library are not in default locations, you may wish to specify `CPPFLAGS=-I/some/dir` and `LDFLAGS=-L/some/other/dir` options at configuration time as well.

If this option is not given, the build system will try to detect an installed version of `libverto` and use it if it is found. Otherwise, a version supplied with the Kerberos sources will be built and installed. The built-in version does not contain the full set of back-end modules and is not a suitable general replacement for the upstream version, but will work for the purposes of Kerberos.

Specifying `--without-system-vert` will cause the built-in version of `libverto` to be used unconditionally.

**--with-krb5-config=PATH**

Use the `krb5-config` program at *PATH* to obtain the build-time default credential cache, keytab, and client keytab

names. The default is to use `krb5-config` from the program path. Specify `--without-krb5-config` to disable the use of `krb5-config` and use the usual built-in defaults.

**--without-keyutils**

Build without `libkeyutils` support. This disables the `KEYRING` credential cache type.

### 3.3.8 Examples

For example, in order to configure Kerberos on a Solaris machine using the `suncc` compiler with the optimizer turned on, run the configure script with the following options:

```
% ./configure CC=suncc CFLAGS=-O
```

For a slightly more complicated example, consider a system where several packages to be used by Kerberos are installed in `/usr/foobar`, including Berkeley DB 3.3, and an `ss` library that needs to link against the `curses` library. The configuration of Kerberos might be done thus:

```
./configure CPPFLAGS=-I/usr/foobar/include LDFLAGS=-L/usr/foobar/lib \  
--with-system-et --with-system-ss --with-system-db \  
SS_LIB='-lss -lcurses' DB_HEADER=db3/db_185.h DB_LIB=-ldb-3.3
```

## 3.4 osconf.hin

There is one configuration file which you may wish to edit to control various compile-time parameters in the Kerberos distribution:

```
include/osconf.hin
```

The list that follows is by no means complete, just some of the more interesting variables.

**DEFAULT\_PROFILE\_PATH**

The pathname to the file which contains the profiles for the known realms, their KDCs, etc. The default value is `/etc/krb5.conf`.

**DEFAULT\_KEYTAB\_NAME**

The type and pathname to the default server keytab file. The default is `DEFKTNNAME`.

**DEFAULT\_KDC\_ENCTYPE**

The default encryption type for the KDC database master key. The default value is `aes256-cts-hmac-sha1-96`.

**RCTMPDIR**

The directory which stores replay caches. The default is `/var/tmp`.

**DEFAULT\_KDB\_FILE**

The location of the default database. The default value is `LOCALSTATEDIR/krb5kdc/principal`.