Build Moonshot from source on macOS (no UI)

The Moonshot source code is available from our GIT repository and it all can be built by hand relatively easily, assuming you have all of the prerequisite packages installed. This page has instructions for building the software itself.

Contents

macOS versions

These instructions have been tested on Mac OS X 10.10 Yosemite, Mac OS X 10.11 El Capitan, macOS 10.12 Sierra and macOS 10.13 High Sierra.

1. System Preparation

1.1. Requirements

To build all of the Moonshot components, you need various packages installed. To install all of these, see below.

1.1.1. Get Xcode for macOS

To get all of the requirements on your macOS platform, you will need to install Xcode and the Xcode command-line extensions:

1. Install Xcode from the Mac App Store.
2. Open a Terminal, then install the Xcode Command Line Tools. You will be prompted with a dialog to install the Command Line Tools after a 130MB download.

```
$ xcode-select --install
```

3. If you have never launched Xcode before, do so at least once, or run the following command in your Terminal window.

```
$ sudo xcodebuild -license
```

1.1.2. Install the GNU tools for macOS

You will need to install several GNU tools:

1. Install the GNU tools that are required for Moonshot by following the instructions at How to install Autoconf, Automake and related tools on Mac OS X from source?
2. Additionally, install GNU GetText:

```
$ curl -OL http://ftpmirror.gnu.org/gettext/gettext-latest.tar.gz
$ tar xfz gettext-latest.tar.gz
$ cd gettext-*
gettext-latest $ ./configure
gettext-latest $ make && sudo make install && cd ..
```

1.1.3. Install MacPorts and Makedepend

Makedepend is available from MacPorts. Install MacPorts:

1. Download the latest install package from MacPorts.org, then update it:

```
$ sudo port -v selfupdate
```

2. Install Makedepend from MacPorts:

```
$ sudo port install makedepend
```


MacPorts

If you prefer to not install MacPorts, install Makedepend manually as follows:

1. Install pkg-config:

```
$ curl -OL http://pkgconfig.freedesktop.org/releases/pkg-config-0.28.tar.gz
$ tar xfz pkg-config-0.28.tar.gz
$ cd pkg-config-*
pkg-config-0.28 $ ./configure --with-internal-glib
pkg-config-0.28 $ make && sudo make install && cd ..
```

2. Install util-macros:

```
$ curl -OL https://www.x.org/releases/individual/util/util-macros-1.19.1.tar.gz
$ tar xfz util-macros-1.19.1.tar.gz
$ cd util-macros-*
util-macros-1.19.1 $ ./configure
util-macros-1.19.1 $ make && sudo make install && cd ..
```

3. Install xproto:

```
$ curl -OL https://www.x.org/archive/individual/proto/xproto-7.0.31.tar.gz
$ tar xfz xproto-7.0.31.tar.gz
$ cd xproto-*
xproto-7.0.31 $ ./configure
xproto-7.0.31 $ make && sudo make install && cd ..
```

4. Install makedepend:

```
$ curl -OL https://www.x.org/releases/individual/util/makedepend-1.0.5.tar.gz
$ tar xfz makedepend-1.0.5.tar.gz
$ cd makedepend-*
makedepend-1.0.5 $ ./configure
makedepend-1.0.5 $ make && sudo make install && cd ..
```

1.1.4. Install JSON from CPAN

1. Update CPAN and install JSON:

```
$ sudo cpan install JSON
```

2. Setting build parameters and locations

Just like on Linux, build and installation locations matter, with one vital difference. On macOS, the `/usr` tree itself is locked down and inaccessible, even for the privileged (root) user. However, locations like `/usr/local` are open, and with newer versions of the OS, expect this to change.

For the purposes of this set of instructions, we recommend the following:

1. For all the Moonshot dependencies, including Moonshot itself, but excluding Heimdal, the `--prefix` parameter should be set to `/usr/local/moonshot`.
   If you decide to change this location, you should appropriately change the locations in the commands in Sections 3 and 5 to your preference.
2. For Heimdal, the `--prefix` parameter should be set to `/usr/local/heimdal`. This is because we are using Heimdal only for the header files that the Heimdal build generates, not for any library linking. It makes the eventual distribution easier.
3. We recommend that you build all libraries with the `-rpath` parameter enabled for all libraries to avoid any clashes with other libraries (such as the older version of OpenSSL that macOS ships for compatibility reasons). We have been assured by macOS developers that the `clang` and `libtool` tools for macOS support this.
4. We do NOT recommend using the Apple-provided sources for some libraries (such as Heimdal) as they have various customisations that may negatively impact how Moonshot works, and because Apple categorically **WILL NOT** support any of their own source sets (we've tried through a Platinum support path and had the support ticket closed and refunded).
   If you DO try using Apple's OpenSource sources and find that things build and function fine, please let us know by commenting on this document (with instructions that we can update this document with).
5. These instructions should generally be backward-compatible.

3. Download and build the required external dependencies

3.1.1. OpenSSL

1. Create a directory called `openssl`.
2. Download the OpenSSL build tree from Apple's OpenSource site. Some scripts that Apple provides will be needed, but we will not build it.

```
$ cd openssl && curl -OL https://opensource.apple.com/tarballs/OpenSSL098/OpenSSL098-59.60.1.tar.gz
```

3. Download the latest OpenSSL build from the OpenSSL website. We will build this version.

```
$ curl -OL https://www.openssl.org/source/old/1.0.2/openssl-1.0.2l.tar.gz
```

4. Extract `OpenSSL098-59.60.1.tar.gz`, copy its 'bin' directory into the openssl directory, then delete the extracted source.
5. Edit the `extract_source.sh` script in the `bin` directory:
   a. Comment out the IDEA removal and patch lines (lines 39-49).
   b. Add the following parameters to each of the three `./Configure` lines: `no-ssl2 enable-ec_nistp_64_gcc_128`
   c. Change the `--openssldir` parameter to your appropriate directory. We recommend **`/usr/local/moonshot/bin`**
   d. Change the `--prefix` parameter from `/usr` to **`/usr/local/moonshot`**
   e. Comment out the line `'rm -f Makefile'`
   f. Find the line `'rm -f x86_64.h i386.h'`, and insert the following below it: **`ln -s crypto/idea/idea.h include/openssl/idea.h`**
6. From the openssl directory, run the following:

```
openssl$ bin/extract_source.sh .
```

7. In the `src` directory, edit the `Makefile` file:
   a. Add the `-DNO_IDEA` parameter to the `CFLAG` line
   b. Add the `-DNO_IDEA` parameter to the `DEPFLAG` line
8. Run the following commands:

```
openssl/src$ make depend
openssl/src$ make
openssl/src$ sudo make install_sw
```

3.1.2. Heimdal

Heimdal requires OpenSSL. Once OpenSSL has built successfully, build Heimdal.

1. Download Heimdal:

```
$ curl -OL https://github.com/heimdal/heimdal/releases/download/heimdal-7.3.0/heimdal-7.3.0.tar.gz
```

2. Extract Heimdal:

```
$ tar xfz heimdal-7.3.0.tar.gz
```

3. Build Heimdal:

```
$ cd heimdal-7.3.0
heimdal-7.3.0$ ./autogen.sh
heimdal-7.3.0$ ./configure --prefix=/usr/local/heimdal --with-openssl=/usr/local/moonshot
heimdal-7.3.0$ make
heimdal-7.3.0$ sudo make install
```

4. Note down both the location in which you built Heimdal, as well as where the Heimdal libraries are installed to (if you changed the `--prefix` parameter to something else). You will need a binary from the Heimdal build for the Moonshot build in Section 6, and you will need to set the `--with-krb5` parameter of the Moonshot `./configure` command in Section 6 to the location where you installed Heimdal.

3.1.3. LibConfuse

1. Clone the latest Libconfuse repository:

```
$ git clone --recursive https://github.com/martinh/libconfuse
```

2. Build Libconfuse:

```
$ cd libconfuse
libconfuse$ ./autogen.sh
libconfuse$ LDFLAGS=" -L/usr/local/moonshot/lib -Wl,-rpath,/usr/local/moonshot/lib " ./configure --
prefix=/usr/local/moonshot
libconfuse$ make
libconfuse$ sudo make install
```

3.1.4. LibEvent

Libevent requires OpenSSL. Once OpenSSL has built successfully, build Libevent.

1. Clone the latest Libevent repository:

```
$ git clone --recursive https://github.com/libevent/libevent
```

2. Build Libevent:

```
$ cd libevent
libevent$ ./autogen.sh
libevent$ CFLAGS=" -I/usr/local/moonshot/include " LDFLAGS=" -L/usr/local/moonshot/lib -Wl,-rpath,/usr
/local/moonshot/lib " ./configure \
--prefix=/usr/local/moonshot
libevent$ make
libevent$ sudo make install
```

4. Checkout the Moonshot source

The Moonshot source code is all stored in a GIT repository. To fetch it, issue the following command.

```
$ git clone --recursive git://git.project-moonshot.org/moonshot.git
$ cd moonshot/moonshot && git checkout master && cd -
```

5. Build Moonshot's internal dependencies

The Moonshot repository contains a set of internal dependencies. These need to be built before you can build the Moonshot mechanism itself.

5.1.1. Libradsec

Libradsec is used by the Moonshot libraries.

1. Configure the Libradsec build:

```
$ cd moonshot/libradsec
libradsec$ chmod +x autogen.sh
libradsec$ ./autogen.sh
libradsec$ LDFLAGS=" -L/usr/local/moonshot/lib -Wl,-rpath,/usr/local/moonshot/lib " CFLAGS=" -I/usr/local
/moonshot/include " ./configure \
--prefix=/usr/local/moonshot
```

2. Edit the `Makefile`:
   a. Find the **`AM_CFLAGS`** line, remove the **`-Werror`** parameter.
   b. Find the **`libradsec_la_CFLAGS`** line, remove the **`-Werror`** parameter
   c. Save the file.
3. Build Libradsec:

```
libradsec$ make
libradsec$ sudo make install
libradsec$ cd -
```

5.1.2. Jansson

Jansson is used by the Moonshot libraries.

1. Configure and build Jansson:

```
$ cd moonshot/jansson
jansson$ libtoolize --force --automake
jansson$ aclocal -I m4 $ACLOCAL_FLAGS
jansson$ autoheader
jansson$ automake --add-missing
jansson$ autoconf
jansson$ LDFLAGS=" -L/usr/local/moonshot/lib -Wl,-rpath,/usr/local/moonshot/lib " CFLAGS=" -I/usr/local
/moonshot/include " ./configure \
--prefix=/usr/local/moonshot --with-sysroot=/usr/local/moonshot
jansson$ make
jansson$ sudo make install
jansson$ cd -
```

6. Build Moonshot

1. Before building Moonshot, check what the HEAD of your Git tree is:

```
$ cd moonshot && git rev-parse --short HEAD
```

2. Check the output.
3. If your head revision is **`dfa044b5b`**, please download the Heimdal patch first, then follow Step 4 to apply it. If your head revision is **`3fbcdec`** or newer, go directly to Step 5.
   heimdal-patch.patch
4. Apply the Heimdal patch:

```
moonshot$ git apply < heimdal-patch.patch
```

5. Configure the Moonshot build:

```
moonshot$ ./autogen.sh
moonshot$ LDFLAGS=" -L/usr/local/moonshot/lib -Wl,-rpath,/usr/local/moonshot/lib " \
LIBS="-framework GSS -F/System/Library/PrivateFrameworks -framework Heimdal -L/usr/local/moonshot/lib" \
COMPILE_ET="/Users/admin/Desktop/build/heimdal-7.3.0/lib/com_err/compile_et" ./configure --with-krb5=/usr
/local/heimdal \
--with-radsec=/usr/local/moonshot --with-opensaml=no --with-shibresolver=no --with-shibsp=no --with-
openssl=/usr/local/moonshot \
--with-jansson=/usr/local/moonshot --sysconfdir=/etc
```

> ⚠ **Configure script parameters**
>
> There are several parameters in the command above that rely on locations noted down previously:
>
> `LIBS` contains explicit library location references to the macOS-native Heimdal and GSS frameworks as well as the Moonshot libraries.
>
> `COMPILE_ET` contains the full path to the `compile_et` binary that will be in your Heimdal build tree. You noted this down in the last step of Section 3.1.2.
>
> `--with-krb5` contains the location where the Heimdal libraries and headers were installed. You noted this down in the last step of Section 3.1.2.

6. Edit the top-level `Makefile`:
   a. Remove the parameter **`-L/usr/local/heimdal/lib`** from the `KRB5_LDFLAGS` line.
   b. Remove the parameters **`-L/usr/local/heimdal/lib -lgssapi`** from the `KRB5_LIBS` line in the file.
   c. Save the file.
7. Edit the `Makefile` in the `libeap` directory:
   a. Remove the parameter **`-L/usr/local/heimdal/lib`** from the `KRB5_LDFLAGS` line.
   b. Remove the parameters **`-L/usr/local/heimdal/lib -lgssapi`** from the `KRB5_LIBS` line in the file.
   c. Save the file.
8. Edit the `Makefile` in the `mech_eap` directory:
   a. Remove the parameter **`-L/usr/local/heimdal/lib`** from the `KRB5_LDFLAGS` line.
   b. Remove the parameters **`-L/usr/local/heimdal/lib -lgssapi`** from the `KRB5_LIBS` line in the file.
   c. Remove the parameters **`-L/usr/local/heimdal/lib -lgssapi`** from the `mech_eap_la_LIBADD` line in the file.
   d. Remove the parameter **`-Werror`** from the `mech_eap_la_CFLAGS` line.
   e. Remove the parameter **`-Werror`** from the `mech_eap_la_CXXFLAGS` line.
   f. Save the file.
9. Build Moonshot:

```
moonshot$ make
moonshot$ sudo make install
```

You should now have a `mech_eap.so` file in `/usr/local/lib/gss`.

7. Test Moonshot

To test this build of Moonshot, you will need to make some privileged changes to the system you built this on:

1. In `/etc`, create a `gss` directory:

```
moonshot$ sudo mkdir -p /etc/gss
```

2. Copy the `mech` file from the Moonshot `mech_eap` build directory to `/etc/gss`

```
moonshot$ sudo cp mech_eap/mech /etc/gss/
```

3. As the privileged user, edit the `/etc/gss/mech` file:
   a. Change the `mech_eap.so` entry on each line to the full path of the library, e.g. `/usr/local/lib/gss/mech_eap.so`
   b. Save the file.

4. Create a `.gss_eap_id` file in your user's home directory:
    a. On the first line of the file, put your Moonshot credential in email form, e.g. `username@moonshot.realm`.
    b. On the second line of the file, put your password for the Moonshot credential in the first line.
    c. Save the file.
5. Run an SSH command to a Moonshot-enabled system that the credentials you added in the previous step will be valid for:

```
ssh -Kv user@moonshot-host.realm
```

> **Jisc Assent**
>
> If you have an identity provider on the Jisc Assent network, you can use `ssh -Kv moonshot@ssh.test.moonshot.ja.net` to test whether your macOS Moonshot mechanism worked successfully.

6. You should successfully connect to the service, and you should see several lines like this in the output:

```
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-mic,password
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Authentication succeeded (gssapi-with-mic).
Authenticated to ssh.test.moonshot.ja.net ([212.219.179.184]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Sending environment.
debug1: Sending env LANG = en_GB.UTF-8
```

> **Jisc Assent**
>
> On the Jisc Assent Test SSH Service, the final output for success will be this:
>
> ```
> debug1: Entering interactive session.
> debug1: Sending environment.
> debug1: Sending env LANG = en_GB.UTF-8
> *** JISC Moonshot Test SSH Server ***
> You have successfully logged in with Moonshot.
> You are user: moonshot
> [moonshot@ssh ~]$
> ```

8. Distribute and install Moonshot

To distribute this binary set, you will need to trim down the binaries you have built to include only the dynamic libraries and only bare essentials needed to run the mechanism:

1. Make a copy of the `/usr/local` directory into a second location as the privileged user.
2. Once the duplication process is complete, change to your duplicate location's `usr/local` directory and delete all directories except the following:
    a. `lib`
    b. `moonshot`
3. In the `lib` directory, delete everything except the following:
    a. `gss`
    b. `libintl.8.dylib`
    c. `libintl.dylib`
4. In the `moonshot` directory, delete everything except the following:

    a. `lib`
5. In the `moonshot/lib` directory, delete:
    a. all `*.a` files
    b. all `*.la` files
    c. the `pkgconfig` directory
6. Now use tar to package up the contents of your distribution directory. You should have a tarball around 2 MB in size.
7. On your target machine, extract the tarball in such a way that the files end up in the `/usr/local/lib` and `/usr/local/moonshot` directories.
8. To finish the deployment, create the `/etc/gss/mech` file, then try to run the SSH test (section 7, steps 4 to 6) to check that the mechanism is correctly loaded.

9. Issues

Current issues with this build include that the macOS SSH client abandons any `gssapi-with-mic` conversations if the first mechanism it chooses, fails.

In a domain environment, this usually involves a Kerberos interaction, i.e. where you have received a Kerberos ticket before by logging in or by running `kinit`. Other ssh clients (or a custom build of the ssh client) may not exhibit this behaviour.

On macOS Sierra and later, the native SSH client is sandboxed when run from its default location in `/usr/bin`. Making a copy of the binary in `/usr/local/bin` enables it to authenticate with Moonshot. Adjust `/etc/paths` to load binaries in `/usr/local/bin` first, then restart your sessions.