OpenSSL

OpenSSL is a widely used, but occasionally problematic package. Since it's widely used, it generally comes with many systems and builds cleanly on most others, but it can be tricky to build correctly at times, is versioned improperly and inconsistently when it comes to shared libraries, and does not support debug builds on Windows.

The trickiest issue is that on platforms where OpenSSL is not native, you may run into multiple versions and find that different pieces of software used by or with Shibboleth are linked to different versions. This will **NOT** work, and you will get errors or crashes.

On platforms with a version included with the OS, it is strongly suggested that you stick with that version. If you're relying on mechanisms like the Mac ports project or one of Solaris' many open source initiatives, you will have to use a uniform set of software that is provided by those channels, including things like libcurl and Apache. So it's all or nothing, essentially, don't try and mix things.

Non-Windows

OpenSSL uses a front-end to the autoconf process that sometimes has trouble detecting the right platform and compiler, particularly on Solaris. It's critical to double-check the output of the initial step to make sure it picked the right settings.

A typical command to autodetect the platform might be:

```
./Configure threads shared
```

On Solaris, you often have to specify the platform and compiler.

For Sun's compiler:

```
./Configure solaris-sparcv9-cc threads shared
```

And for gcc:

```
./Configure solaris-sparcv9-gcc threads shared
```

Windows

On Windows, the docs can be a little sketchy at times, and the default build is essentially broken. It doesn't supply a proper debug build makefile, it doesn't name its libraries so as to avoid version conflicts, and doesn't support any of the native Windows assembly mechanisms for avoiding conflicts. Given that, a compromise is to modify the default makefiles during the build set up process to create versioned library names to avoid conflicts.

My system includes ActiveState Perl. The directions below are for the latest release (1.0.0 at time of writing). A given generation of lettered versions should be able to share a set of filenames so that you can drop in patched versions at runtime.

First I configure the package from source as follows (latest version at time of topic creation is shown). These steps generate starting makefiles and DEF files that are customized later.

```
C:> cd \openssl-1.0.0
perl Configure VC-WIN32
ms\do_ms.bat
perl util\mk1mf.pl debug dll no-asm VC-WIN32 1>ms\ntdlldebug.mak
copy ms\libeay32.def ms\libeay32d.def
copy ms\ssleay32.def ms\ssleay32d.def
```

Now edit the DEF files to adjust the module names embedded inside the libraries we're building. The module name is specified near the top in the **LIBRARY** command. Modify as follows:

- `ms\libeay32.def: LIBEAY32_1_0_0`
- `ms\libeay32d.def: LIBEAY32_1_0_0D`
- `ms\ssleay32.def: SSLEAY32_1_0_0`
- `ms\ssleay32d.def: SSLEAY32_1_0_0D`

Now modify the default makefiles (`ms/ntdll.mak` and `ms/ntdlldebug.mak`) to change output information and adjust some settings. If you're using VS 2005/2008, you have to remove the /WX option from CFLAGS because some warnings are being generated by the compiler.

In `ms/ntdll.mak`:

- Remove `/WX` from **CLAG** (VS 2005/2008 only)
- Edit the **O_SSL/O_CRYPTO** and **L_SSL/L_CRYPTO** macros around line 78 or so to reflect the corrected filenames:
    - `O_SSL= $(LIB_D)\$(SSL)_1_0_0.dll`
    - `O_CRYPTO= $(LIB_D)\$(CRYPTO)_1_0_0.dll`

    - `L_SSL= $(LIB_D)\$(SSL).lib`
    - `L_CRYPTO= $(LIB_D)\$(CRYPTO).lib`
- Near the bottom of the file, edit the link commands to set the import library filenames by adding `/implib:$(L_CRYPTO)` and `/implib:$(L_SSL)` to the respective links. For VS 6 only, also remove unicows.lib from the link command.

In `ms/ntdlldebug.mak`:

- Remove `/WX` from **CLAG** (VS 2005/2008 only)
- Edit the **O_SSL/O_CRYPTO** and **L_SSL/L_CRYPTO** macros around line 78 or so to reflect the corrected filenames:
    - `O_SSL= $(LIB_D)\$(SSL)_1_0_0D.dll`
    - `O_CRYPTO= $(LIB_D)\$(CRYPTO)_1_0_0D.dll`
    - `L_SSL= $(LIB_D)\$(SSL)D.lib`
    - `L_CRYPTO= $(LIB_D)\$(CRYPTO)D.lib`
- Near the bottom of the file, edit the link commands that reference the DEF files to refer to the correct debug filenames (`LIBEAY32D.DEF` and `SSLEAY32D.DEF`) and set the import library filenames by adding `/implib:$(L_CRYPTO)` and `/implib:$(L_SSL)` to the respective links. For VS 6 only, also remove unicows.lib from the link command.

With those changes made you can run `nmake` against those makefiles to generate debug and release builds with properly isolated filenames. You'll get an `openssl.exe` command line linked to them as well.

The other big advantage is that you can stick the library path to both `out32dll` and `out32dll.dbg` directly in your Visual Studio global library directory list since the link library names are now distinct.