B

Adding SSL Support to IMAP

Many users use their web browser's built-in email client to read their IMAP mail. Many browsers, such as Netscape and IE, are SSL-enabled, but because the current distribution versions of UW and Cyrus servers do not support SSL, the users can't take advantage of their browser's SSL support. There is, however, a workaround—you can use the freely available OpenSSL SSL toolkit and *stunnel*, an SSL encryption wrapper, to wrap IMAP in SSL. The procedure for adding SSL support to IMAP is documented in this appendix. It involves downloading some free software, building and installing it, generating a certificate, and modifying your *inetd* configuration.

UW 2000 and the 2.0 release of Cyrus will support native SSL and are expected to be released before this book is published.

Get the Software

The first step is to get the latest versions of the sources for OpenSSL and *stunnel*.

OpenSSL

http://www.openssl.org/

OpenSSL is a free implementation of Netscape's Secure Socket Layer—the software encryption protocol behind the Netscape Secure Server and the Netscape Navigator Browser. OpenSSL implements Secure Sockets Layer *SSLv2* (Version 2) and *SSLv3* (Version 3) and Transport Layer Security (TLSv1).

Download the latest OpenSSL source distribution (Version 0.9.5a as of this writing), and unpack it where you normally build free software. The URL given at the beginning of this section is the master location for OpenSSL sources. At that URL

you will also find useful documentation. A good, concise source of information is the OpenSSL FAQ, complete with instructions on generating keys, pass phrases, and certificates, that is included in the OpenSSL sources.

stunnel

http://mike.daewoo.com.pl/computer/stunnel

stunnel opens an encrypted pipeline between the SSL client and the IMAP server. It decrypts the data the client sends through the pipeline and passes it over to the IMAP server in cleartext over the loopback interface, where it's safe from being snooped from the network. Download the latest version of the source distribution (which as we write this appendix, is *stunnel-2.4a.tar.gz*) to the area on your system where you normally build free software, and proceed to the next section for instructions on building *stunnel*.

Put It All Together

Now that you've got the sources, the next step is to build and install the software, create a certificate, and tweak your *inetd* configuration file.

Install OpenSSL

OpenSSL supports RSA encryption, which most web browsers use in SSL sessions. Inside the United States, however, RSA.com holds a patent on the RSA encryption algorithms, and that makes it illegal to use OpenSSL with its standard RSA support. For legal use inside the United States, OpenSSL must be built to use the RSAREF encryption libraries, which are included with the OpenSSL.* The simplest way to build OpenSSL is using the "RSAglue" method. Using this method, you'll need to build support for OpenSSL into your application, such as *stunnel*, by including the header file *RSAref.h* in your application's Makefile at build time. An example is given in the instructions for building *stunnel* a little bit later in this section.

OpenSSL installs under */usr/local/ssl* by default (not in */usr/local* as the documentation might indicate). If you don't want it installed there, then read the *INSTALL* file provided with the distribution for instructions on how to change the install location.

---

* RSAREF is no longer available for download from RSA.com's FTP site. If you want to obtain RSAREF, it is available from *ftp.eecs.umich.edu/pub/EECS/crypt/rsaref.tar.gz*

Unpack the distribution:

```
% zcat openssl-0.9.5a.tar.gz | tar xvf -
```

In the top level of the unpacked distribution, run the *config* script:

```
% ./config
```

Next, build OpenSSL:

```
% make
% make test
# make install
```

If you are inside the United States, there is one final step—you will need to copy the *rsaref.h* header file and *libRSAglue.a* library into */usr/local/ssl/*. Applications into which you build SSL support will reference the RSAglue library to include RSAREF support.

```
% cp rsaref/rsaref.h /usr/local/ssl/include
% cp libRSAglue.a /usr/local/ssl/lib
```

Finally, edit */usr/local/ssl/lib/openssl.cnf* as needed. Comments in that file are self-explanatory.

Install stunnel

Unpack the *stunnel* distribution:

```
% zcat stunnel-2.4a.tar.gz | tar xvf -
```

Run the configure script:

```
% ./configure
```

Edit the *Makefile*. Change the `LIBS` definition so that it includes the RSAglue library (RSAglue should be linked after the *ssl* library and before the *crypto* library). The following example is from a Solaris 7 system:

```
LIBS=-lpthread -lsocket -lnsl -L/usr/local/ssl/lib -lssl -lRSAglue -lcrypto
```

Then, build *stunnel*:

```
% rm config.cache
% make clean
% make
# make install
```

Note that using *stunnel* with threads has a small footprint. However, you will end up with as many IMAP sessions as you have file descriptors, so make sure to increase the number of file descriptors on a large system.

Create a Certificate

The next step is to generate a new certificate for *stunnel*. It's sufficient to build a dummy certificate for *stunnel*—private key and certificate authority are not required:

```
# cd /usr/local/ssl/certs
# ../bin/req -new -x509 -nodes -out stunnel.pem -keyout \
   stunnel.pem -days 999
# chown cyrus:mail stunnel.pem
# chmod 0600 stunnel.pem
```

On Cyrus servers, the certificate must be owned by the *cyrus* user, or the client will not be able to verify the certificate. Note also that *stunnel* might ship with a copy of *stunnel.pem*—if your *stunnel* distribution comes with a certificate, delete it and build a new one.

Modify Services

Edit */etc/services*, and add the following line:

```
simap      993/tcp      simap      # SSL enabled IMAP
```

Listen for Secure IMAP Connections

Port 993 is the standard port that listens for requests to open secure IMAP (*simap*) connections. There are two ways to listen for *simap* connections: either by running *stunnel* out of *inetd* or by running it as a standalone daemon.

Running stunnel standalone

This is the preferred method of running *stunnel*. Running *stunnel* out of *inetd* adds some overhead to your system, because SSL has to be initialized on every connection. Additionally, when you run *stunnel* out of *inetd*, *stunnel* does not support session caching and will use more memory when the system is carrying a heavy load.

To run *stunnel* standalone, run one of the following commands. For Cyrus servers:

```
# stunnel -d 993 -l /usr/cyrus/bin/imapd -r imapd
```

For UW servers:

```
# stunnel -d 993 -l /usr/local/sbin/imapd -r imapd
```

stunnel must be started as *root* in order to bind to port 993. Cyrus servers will still perform all IMAP operations as the user defined for the *imapd* service in */etc/inetd.conf* (user *cyrus* in most cases).

Running stunnel out of inetd

Edit */etc/inetd.conf*. If your server is a Cyrus server, add the line:

```
simap stream tcp nowait cyrus /usr/local/bin/stunnel stunnel -l \
   /usr/cyrus/bin/imapd imapd
```

The *–l* argument tells *stunnel* which *inetd*-type program to run.

For UW servers, use:

```
simap stream tcp nowait root /usr/local/bin/stunnel stunnel -l \
  /usr/local/sbin/imapd imapd
```

And finally, restart *inetd*:

```
# ps -ef | grep inetd
root   150     1   TS  48   Apr 21 ?     0:01 /usr/sbin/inetd -s
# kill -hup 150
```

The *stunnel* FAQ, which comes with the *stunnel* source distribution, has many suggestions for testing and troubleshooting your *stunnel* installation.