

Windows Eudora SSL Architecture

February 16, 2006

1 Introduction

Windows Eudora contains its own DLL to manage SSL negotiation. This DLL uses OpenSSL as its engine and serves as a wrapper around the engine to simplify using the SSL functionality.

2 Basic SSL Functionality

Windows Eudora uses a fairly basic subset of the SSL functionality. It receives and processes the server certificate but offers no support for client certificates.

2.1 Server Certificate Processing

The limited processing of the server certificate is implemented in the callback function `QCCertificateUtils::CertificateCallback()`. Many minor certificate problems are simply ignored, others are reported to the user and the user is given the option of trusting the certificate despite the problems. Matching of the server name in the certificate against the actual server name is also done on a limited basis.

2.2 Root Certificates

Windows Eudora maintains a store of certificates that are trusted by default. This store is in the `rootcerts.p7b` file and contains certificates from various authorities that are considered trustworthy and generally useful. Reasonable effort is made to accommodate sites that request additional certificates be included in the default store. We are more inclined to add certificates that are of use to a wide variety of sites than certificates specific to only a single site. Individual sites are free to modify the root certificate store to add their own certificates. Our experience suggests that the vast majority of sites using SSL work by default with our current store.

2.3 User Trusted Certificates

Windows Eudora also maintains a store of certificates that for one reason or another were not considered trusted by default but that the user has chosen to trust. This store is in the `usercerts.p7b` file. When Eudora performs an SSL negotiation numerous checks are performed by OpenSSL and Eudora on the certificate. Some minor problems are ignored, but if the certificate fails any of the more important checks a dialog is presented

to the user showing the certificate in question and explaining the reason the certificate was not trusted. The user is given the option of trusting the certificate despite the problem. If the user chooses to trust the certificate it is added to the store of user trusted certificates. On any subsequent negotiations this certificate will be found in the user trusted store and will be trusted regardless of any problems with the certificate.

3 The QCSSL Project

The QCSSL project compiles into QCSSL.dll. The POP, SMTP and IMAP protocols all use this DLL to perform SSL negotiation. (The LDAP protocol does not use this DLL, but rather relies on OpenLDAP.) The QCSSL project in turn uses the QCSocket project to do reading and writing over the network and uses the OpenSSL project as the engine to do the actual negotiation.

The QCSSL project has two components that are of interest: the interface that code uses to call into the DLL and the wrapper around OpenSSL.

3.1 The DLL Interface

The DLL interface is found in QCSSL.CPP and QCSSL.h. The interface into this DLL is implemented via a set of function pointers. There are two general categories of interface functions: connection management and certificate management.

3.1.1 Connection Management

The following functions are used to manage a connection which uses SSL.

FPNQCSSLClean – Called to clear out session specific data to begin a new session. The current architecture does not use this function, but simply calls FPNQCSSLBeginSession and FPNQCSSLEndSession.

FPNQCSSLBeginSession – Called to begin a new SSL session.

FPNQCSSLEndSession – Called to end an SSL session.

FPNQCSSLWrite – Called to write data across an SSL encrypted connection.

FPNQCSSLRead – Called to read data across an SSL encrypted connection.

FPNQCSSLGetConnectionInfo – Called to obtain the data structure containing information about a specific SSL session.

3.1.2 Certificate Management

All user interface portions relating to SSL are handled outside of this DLL so we need to provide a means for other portions of the code to view and modify certificate data.

FPNQCSSLAddTrustedCertFromFile – Called to add an untrusted certificate to the user trusted certificate store.

FPNQCSSLGetRootCertList – Called to obtain a list of the certificates in the root store.

FPNQCSSLGetUserCertList – Called to obtain a list of the certificates in the user trusted store.

FPNQCSSLAddTrustedUserCert – Called to add a certificate to the user trusted store.

FPNQCSSLDeleteTrustedUserCert – Called to remove a certificate from the user trusted store.

3.2 The OpenSSL Wrapper

Any portion of the Eudora code which needs access to SSL functionality calls into the QCSSL DLL via the functions described above. There is a one-to-one correspondence between those functions and the public functions in the QCSSLContext.cpp file and QCSSL functions use the corresponding QCSSLContext functions. The QCSSLContext functions in turn call into the OpenSSL library.

An instance of the QCSSLReference data structure is created for each connection that uses SSL. This data structure holds the relevant information about the connection and the instance of this object is passed between the various functions.

The original author of this code (not the author of this document) chose to implement this functionality as a data structure passed among C functions which (to the author of this document) seems to be a textbook case for making this a C++ class. Perhaps there was a good reason at that time to use C for this implementation rather than C++ but there seems (to this author) to be no reason to continue this limitation other than a question of how much developer time to spend making a change to code that works fine as it currently stands (which is why this author never bothered to make that change).

4 The OpenSSL Project

The OpenSSL project contains the source files as extracted from the OpenSSL archive. As of this writing the OpenSSL code is compiled into a static library and that library is linked into the QCSSL.dll library. We have considered compiling OpenSSL into its own DLL file and this is probably another reasonable approach. It would allow updating of

Eudora to the latest SSL simply by updating the OpenSSL.dll library assuming no relevant portion of the OpenSSL interface changed. It is perhaps arguably a little safer to update the entire QCSLL.dll since that interface is not at all likely to change. In any case, we do not currently update Eudora by changing select binaries so this is not a pressing issue.

We modify one small portion of the OpenSSL make file but otherwise use OpenSSL unmodified from the standard package. The one change we made was to prevent lots of unnecessary executables from being built each time.

We replaced this line:

```
all: banner $(TMP_D) $(BIN_D) $(TEST_D) $(LIB_D) $(INCO_D) headers lib  
exe
```

With the following:

```
# Use this line to build executables as well and libraries:  
# all: banner $(TMP_D) $(BIN_D) $(TEST_D) $(LIB_D) $(INCO_D) headers  
lib exe  
# Use this line to build libraries only:  
all: banner $(TMP_D) $(BIN_D) $(TEST_D) $(LIB_D) $(INCO_D) headers lib
```

It should be noted that doing a “Clean Solution” on Eudora will not actually remove the binaries for OpenSSL. It is debatable as to whether or not this would be expected or desired. If considered desirable this would require only a small change to the OpenSSL project inside of Developer Studio. Currently, cleaning the OpenSSL binaries must be done either via the OpenSSL make tools or manually.