Accessing Sun™ Mainframe Transaction Processing Software Through Secure Sockets

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 816-7255-10
June 2002, Revision A

Send comments about this document to: `docfeedback@sun.com`

Contents

Secure Sockets Layer (SSL)

This document describes how to configure and customize your region to work with SSL clients.

What is SSL?

The secure sockets layer (SSL) allows applications to use sockets for authenticated, tamper-proof, and encrypted communications. It was designed for exchanging secure information over an insecure network such as the internet. It allows an SSL-enabled server to authenticate itself to an SSL-enabled client, the client to authenticate itself to the server, and both sides to establish an encrypted connection.

SSL server authentication allows a client to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate is valid and has been issued by a Certificate Authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

SSL client authentication allows a server to confirm a client’s identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client’s certificate is valid and has been issued by a Certificate Authority (CA) listed in the server’s list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient’s identity.

An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties in any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering – that is, for automatically determining whether the data has been altered in transit.

Prerequisite Software

To use the SSL interface with the Sun™ Mainframe Transaction Processing software (Sun MTP), you must install Network Security Services (NSS) software. Download the software from The Mozilla Organization at `www.mozilla.org`.

- NSS binaries:

  `ftp.mozilla.org/pub/security/nss/releases/NSS_3_4_1_RTM/SunOS5.6_OPT.OBJ/`

- Netscape™ Portable Runtime (NSPR) binaries, which are required by the NSS software:

  `ftp.mozilla.org/pub/nspr/releases/v4.1.2/SunOS5.6_OPT.OBJ/`

Configuring a Region

Client applications can communicate with Sun MTP regions using SSL. The listening server process, `unikixssl`, listens on a pre-defined port for incoming requests. The `unikixssl` process routes requests through the `unikixsock` socket listener process, which must be running in your region.

▼ To Configure a Region to Accept SSL Requests

1. **Determine the listening port for the TCP/IP socket listener process,** `unikixsock`**.**

   You will need to use the `-p` *port_number* option to `unikixmain` when starting the region to start the `unikixsock` listener process.

2. **Determine the listening port for the SSL listener process,** `unikixssl`**.**

3. **Install the NSS and NSPR software.**

4. **Set up the certificate databases.**

5. **Create or update the** `unikixrc.cfg` **file with the SSL entries.**

6. **Ensure that the location of the NSS and NSPR libraries are specified in the `LD_LIBRARY_PATH` environment variable in your region's setup file.**

7. **Add the `$UNIKIX/lib` directory to the `LD_LIBRARY_PATH` environment variable so that `unikixssl` can find the SSL user exit library (`libkxsslxit.so`).**

Setting Up the Certificate Databases

When you set up your region, you must specify the certificate database directory where the SSL server certificate and the list of trusted certificate authorities are stored. For information on setting up the database directory, refer to the NSS *SSL Reference* document, specifically the "Setting up the Certificate and Key Databases" section in Chapter 2, Getting Started with SSL. You should also refer to the sample setup script `$UNIKIX/src/socket/sslsetup`, which shows how to set up the certificate database used in the SSL example.

Setting Up the Communications Manager for `unikixssl`

Communications Manager (`unikixCommMgr`) is automatically started by `unikixmain`. It reads a configuration file on startup called `$KIXSYS/unikixrc.cfg`. The `unikixrc.cfg` file contains information about the SSL server, `unikixssl`, which is started by `unikixCommMgr`. Each region that supports an SSL server must have its own `unikixrc.cfg` file.

▼ To Configure `unikixCommMgr` to Start `unikixssl`

1. **Copy the `unikixrc` file from the `$UNIKIX/lib` directory to the `$KIXSYS` directory of the region you are configuring.**

2. **Edit the `SslServer*` entries. and save the file as `unikixrc.cfg`.** TABLE 1 **describes each entry.**

   When modifying the `unikixrc.cfg` file, the following syntax rules apply:

   - The exclamation point (!) denotes a comment; any text appearing on a line after the ! is ignored.
   - Keywords and non-quoted values can contain any combination of uppercase and lowercase letters.
   - Values in quotes, such as path and file names, are case-sensitive.

Note – All the cipher suites default to False (disabled). You must enable at least one cipher suite (set to True), or all client requests are rejected.

TABLE 1 `unikixrc.cfg` File –- SSL Server Entries *(1 of 3)*

`SslServer*Active`	If `True`, activate the `unikixssl` Server. If `False` (default), the `unikixssl` Server is not started and all subsequent `SslServer` keywords are ignored.
`SslServer*Debug`	Save `unikixssl` trace information to a file. The values are `True` or `False` (default). Set this value to `False` unless you are working with a technical support representative and you are asked to enable this feature.
`SslServer*Host`	The host to listen for client connections on. You can specify a host name, an IP address, `any` (all IP addresses - `INADDR_ANY`) or `loopback` (local host - `INADDR_LOOPBACK`). If you specify an IP address, it must be enclosed in quotes; for example, "123.45.67.89". If you do not include this entry, the default is `any`. This is appropriate in most cases.
`SslServer*Port`	The SSL port to listen for client connections on.
`SslServer*Sockhost`	The `unikixsock` host. This is the host you specified on the `unikixmain` command with the `-h` option. You can specify a host name, an IP address, `any` (all IP addresses - `INADDR_ANY`) or `loopback` (local host - `INADDR_LOOPBACK`). If you specify an IP address, it must be enclosed in quotes; for example, "123.45.67.89". If you do not include this entry, the default is `any`. This is appropriate in most cases.
`SslServer*Sockport`	The `unikixsock` listen port. This must match the port number specified on the `-p` option on the `unikixmain` command line.
`SslServer*Certdir`	The pathname of the certificate database containing the `unikixssl` server certificate.
`SslServer*Nickname`	The nickname of the `unikixssl` server certificate.
`SslServer*Password`	The `unikixssl` server certificate database password.

TABLE 1 `unikixrc.cfg` File –- SSL Server Entries *(2 of 3)*

`SslServer*Clientcertrequested`	If `True`, the `unikixssl` server requests certificates from clients. Default. If `False`, clients are not asked to provide certificates.
`SslServer*Clientcertrequired`	Has meaning if `SslServer*Clientcertrequested` entry is set to `True`. If `True`, clients must provide a valid certificate to establish a connection. If `False`, clients can optionally provide a certificate. If the client provides a valid certificate or no certificate, a connection will be established.
`SslServer*SSL_EN_RC4_128_WITH_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_EN_RC4_128_EXPORT40_WITH_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_EN_RC2_128_CBC_WITH_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_EN_RC2_128_CBC_EXPORT40_WITH_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_EN_DES_64_CBC_WITH_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_EN_DES_192_EDE3_CBC_WITH_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_WITH_NULL_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_EXPORT_WITH_RC4_40_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.

TABLE 1 `unikixrc.cfg` File –- SSL Server Entries *(3 of 3)*

`SslServer*SSL_RSA_WITH_RC4_128_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_WITH_DES_CBC_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_WITH_3DES_EDE_CBC_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_FORTEZZA_DMS_WITH_NULL_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_FORTEZZA_DMS_WITH_RC4_128_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*SSL_RSA_FIPS_WITH_DES_CBC_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.
`SslServer*TLS_RSA_EXPORT1024_WITH_RC4_56_SHA`	Cipher suite. If `True`, the cipher suite is enabled. If `False`, the cipher suite is disabled.

▼ To Start the Region and the SSL Server

1. **Make sure you have the appropriate values in the `unikixrc.cfg` file.**
2. **Source your region environment.**

   Make sure that the `LD_LIBRARY_PATH` environment variable contains the location of the NSS and NSPR libraries and `$UNIKIX/lib`.
3. **Start the region.**
   - You must specify the `-p` option on the `unikixmain` (`kixstart`) command to start the socket listener. The port number must match the `SslServer*Sockport` value in the `unikixrc.cfg` file.
   - You can optionally specify the `-h` option (to specify a host to which the socket listener is bound). If you use the `-h` option, its value must match the `SslServer*Sockhost` value in the `unikixrc.cfg` file.

   Refer to the *Sun Mainframe Transaction Processing Software Reference Manual* for information about the `-h` option.

Customizing the SSL User Exit

The SSL client certificate verification user exit resides in the shared library `$UNIKIX/lib/libkxsslxit.so`. This user exit allows you to write code that accepts or rejects a client certificate based, for example, on your own certificate revocation check.

The source code for the user exit and a `makefile` to build the shared library are located in the `$UNIKIX/src/socket` directory. The source file is `kxsslxit.c` and the makefile is `makefile.xit`. You can modify the source file to customize the user exit for your environment and use the makefile to build a new shared library.

The function in the shared library that performs the client certificate verification is `Exit_VerifyCertificate()`. During the SSL handshake, `unikixssl` calls this function passing in a pointer (`CERTCertificate *cert`) to the client certificate. The function returns `SECSuccess` if the client certificate is accepted, or `SECFailure` if the client certificate is rejected.

▼ To Customize the SSL User Exit

1. **Back up the existing `$UNIKIX/lib/libkxsslxit.so` shared library; for example:**

```
$ mv $UNIKIX/lib/libkxsslxit.so $UNIKIX/lib/libkxsslxit.so.bak
```

2. **Change directory to the location of the SSL user exit source file:**

```
$ cd $UNIKIX/src/socket
```

3. **Open the source file `kxsslxit.c` and make your changes to the `Exit_VerifyCertificate()` function.**

4. **Save the file.**

5. **Build the shared library:**

```
$ make -f makefile.xit
```

   This creates `libkxsslxit.so` in the `$UNIKIX/src/socket` directory.

6. **Copy the new shared library to the `$UNIKIX/lib` directory:**

```
$ cp libkxsslxit.so $UNIKIX/lib
```

7. **If you wish, you can perform a check to determine where `unikixssl` is finding the shared library.**

```
$ cd $UNIKIX/bin
$ ldd unikixssl
```

   The output shows the list of shared libraries `unikixssl` uses and their locations. Make sure that `libkxsslxit.so` is listed in `$UNIKIX/lib`.

`unikixssl` Server Process

When client applications communicate with a remote Sun MTP region using SSL, the SSL server process, `unikixssl`, listens on a pre-defined port for incoming requests. Before a request is forwarded to the transaction server to be serviced, an SSL handshake occurs. The handshake allows the client and server to validate each other and negotiate an encryption technique. If the handshake is successful the request is forwarded to the transaction server. All data that flows between the client and server is encrypted using the encryption technique negotiated during the handshake.

The `unikixssl` server process works as follows:

1. When a well-defined port number is specified on the `unikixmain` command line with the `-p` option, `unikixmain` starts the `unikixsock` server process, which binds to the specified port number and issues a listen call for that port. For information on `unikixmain`, refer to the *Sun Mainframe Transaction Processing Software Reference Manual*. `unikixssl` uses `unikixsock` to establish a connection to a transaction server.

2. `unikixssl` is controlled by entries in the `unikixrc.cfg` file. `SslServer*Active` should be set to `True` to start the `unikixssl` process. `SslServer*Port` specifies the port on which `unikixssl` listens for client connections. `SslServer*Sockport` specifies the `unikixsock` listen port.

3. When an SSL client program connects to `unikixssl`, an SSL handshake occurs. The client program authenticates `unikixssl'`s certificate, `unikixssl` authenticates the client certificate and an encryption technique is negotiated. If the handshake is successful, `unikixssl` passes the client socket to a transaction server for processing.

4. The transaction server calls the socket user exit, which reads the socket message from the client. If the message conforms to the standard IBM format, the unmodified user exit parses it successfully. If the message is in a non-standard format, the user exit must be customized to parse the message. See “Customizing the SSL User Exit” on page 7.

5. After the message is received and parsed, the transaction server starts the requested transaction and passes to it in the `COMMAREA`, the client socket file descriptor and any user data received.

Note – The remote address passed in the `COMMAREA` for SSL clients is the address of `unikixssl` and not that of the client.

At this point, the SSL client program should wait for data from the transaction program, which ensures that the transaction has started successfully. From this point on, the transaction program and the SSL client program can exchange data using the socket connection. The transaction program uses the standard UNIX socket library calls. Because these are C language calls, the transaction program must use COBOL `CALL` statements. See the `SSLSOCK0.cl2` program in the `$UNIKIX/src/socket` directory for examples. For a complete description of this example refer to the `$UNIKIX/src/socket/README.ssl` file.

Sending a Message

`unikixssl` requires that the client program send the first message in a prescribed format. After the first transmission, the client program should wait for a response before sending any subsequent transmissions.

The input format for the first transmission is as follows:

TRANID`[,`*User-Data*`][,`*XX* `[,`*HHMMSS*`]]`

where:

TRANID	Transaction identifier, 1 to 4 characters, which must exist in the Sun MTP PCT.
User-Data	Optional text; up to 35 characters. Because commas are interpreted as field separators, the data cannot include a comma in either character or binary format. Decimal 44 [hex 2C] are interpreted as commas.
XX	Optional startup type: `IC` or `ic`: Interval Control `TD` or `td`: Transient Data Note that if this field is left blank, startup is immediate.
HHMMSS	Optional field used for hours, minutes and seconds for the interval time when the transaction is started using Interval Control.

CODE EXAMPLE 1 shows the code in `$UNIKIX/src/socket/sslsock00.c` that builds the initial message. For this code to execute without error, you must configure the `SSL0` transaction in the Sun MTP region. Refer to the `$UNIKIX/src/socket/README.ssl` file for instructions on configuring the `SSL0` transaction.

CODE EXAMPLE 1 Building the Initial Message

```
char ibuffer[4+1+35];  /* transid                                 4 bytes
                                            ','              1 byte
                                            data             35
bytes */

/* build initial socket message */
memset(ibuffer, 0, sizeof(ibuffer));
memcpy(ibuffer,   trans,  4);
memcpy(ibuffer+4, ",",    1);
memcpy(ibuffer+5, data,  35);

/* Initial send to invoke transaction */
if (PR_Write(sock, ibuffer, sizeof(ibuffer)) <= 0) {
      exitErr("PR_Write");
}
```

Receiving Messages

The output area must be defined in the transaction program as illustrated in the following example. This code is from the sample program `$UNIKIX/src/socket/SSLSOCK0.cl2`.

CODE EXAMPLE 2 Defining the Output Area

```
 01  DFHCOMMAREA.
         05 GIVE-TAKE-SOCKET         PIC 9(8) COMP.
         05 LSTN-NAME                PIC X(8).
         05 LSTN-SUBNAME             PIC X(8).
         05 CLIENT-IN-DATA           PIC X(36).
         05 SOCKADDR-IN-PARM.
           15 SIN-FAMILY             PIC 9(4) COMP.
           15 SIN-PORT               PIC 9(4) COMP.
           15 SIN-ADDRESS            PIC 9(8) COMP.
           15 SIN-ZERO               PIC X(8).
```

The initial client data is passed in `CLIENT-IN-DATA`. The sample program also includes code for receiving a message, displaying data, and sending a message. The following example shows the code in `SSLSOCK0.cl2` for receiving a message.

CODE EXAMPLE 3 Receiving a Message – SSL *(1 of 2)*

```
working-storage section.
      *
      * program buffers
      *
       77 ws-recv-msg-size           pic s9(8) comp value 4096.
       77 ws-recv-buf                pic x(4096).
       77 ws-recv-total              pic s9(8) comp value 0.
       77 ws-recv-left               pic s9(8) comp value 0.
       77 ws-flags                   pic s9(8) comp value 0.
...

      *
      * set up the receive buffer
      *
              move low-values to ws-recv-buf.
              set ws-recv-total to zero.
              compute ws-recv-left = ws-recv-msg-size.
      *
      * receive data
      *
       recv-1.
              call "recv" using by value GIVE-TAKE-SOCKET,
                 by reference ws-recv-buf(1+ws-recv-total:ws-recv-left),
                 by value ws-recv-left,
                 by value ws-flags.
      *
      * test what was received and decide what we should do
      *
              if return-code < zero
                 display 'SSLSOCK0:recv error ',
                 go to socket-error.

              if return-code = zero
                 display 'SSLSOCK0:client disconnected',
                 go to socket-error.
      *
```

CODE EXAMPLE 3 Receiving a Message – SSL *(2 of 2)*

```
        * have we received all the data yet?
        *
             compute ws-recv-total = ws-recv-total + return-code.
             compute ws-recv-left = ws-recv-msg-size - ws-recv-total.
        *
        * not yet
        *
             if ws-recv-left > 0 go to recv-1.
        *
        * received all the data
        *
             display 'SSLSOCK0:receive buffer   =', ws-recv-buf(1:50).
```

Client certificate information can be obtained using the `EXEC CICS EXTRACT CERTIFICATE` API call. See “API Support” on page 14.

Server Certificate Common Name Issues

During the handshake, SSL clients check to make sure that the common name (CN) of the server certificate matches the host name that the client is using to connect to the server. This must be a textual match. If there is not a match, the client will reject the server certificate with the following error:

```
-12276 SSL_ERROR_BAD_CERT_DOMAIN
```

For example, a `unikixssl` server is running on a host named *saturn*, whose IP address is 123.45.67.89. The server certificate's common name could be *saturn* or *saturn*’s IP address. When a client connects to the `unikixssl` server, it should use the host name or the IP address, whichever matches the server certificate's common name. If a client uses *saturn*’s IP address when the certificate's common name is *saturn*, a connection will be made but the handshake will fail, because the IP address is not a textual match with the server certificate's common name.

API Support

Sun MTP supports the `EXTRACT CERTIFICATE` command for use with SSL clients.

```
EXTRACT CERTIFICATE(ptr-ref)
                 [LENGTH(data-area)]
                 OWNER | ISSUER
                 [COMMONNAME(ptr-ref)]
                 [COMMONNAMELEN(data-area)]
                 [COUNTRY(ptr-ref)]
                 [COUNTRYLEN(data-area)]
                 [STATE(ptr-ref)]
                 [STATELEN(data-area)]
                 [LOCALITY(ptr-ref)]
                 [LOCALITYLEN(data-area)]
                 [ORGANIZATION(ptr-ref)]
                 [ORGANIZATLEN(data-area)]
                 [ORGUNIT(ptr-ref)]
                 [ORGUNITLEN(data-area)]
```

`EXTRACT CERTIFICATE` allows the application to obtain information from the X.509 certificate that was received from an SSL client during an SSL handshake with the SSL server (`unikixssl`). The certificate contains the fields that identify the owner (or subject) of the certificate, and fields that identify the Certificate Authority that issued the certificate. You can select the fields that you require by specifying the `OWNER` or `ISSUER` option. You cannot retrieve both `OWNER` and `ISSUER` fields with one command.

`CERTIFICATE`	Specifies a pointer reference to be set to the address of the full certificate (distinguished name) received from the client. The string `no certificate` is returned if the client did not provide a certificate. If `no certificate` is returned, the other options return zero length.
`LENGTH`	Specifies a fullword binary data area to be set to the length of the full certificate (distinguished name).
`OWNER`	Indicates that the values returned by this command refer to the owner of the certificate.
`ISSUER`	Indicates that the values returned by this command refer to the certificate of the Certificate Authority that issued this certificate.
`COMMONNAME`	Specifies a pointer reference to be set to the common name from the client certificate.

`COMMONNAMLEN`	Specifies a fullword binary data area to be set to the length of the common name from the client certificate.
`COUNTRY`	Specifies a pointer reference to be set to the address of the country from the client certificate.
`COUNTRYLEN`	Specifies a halfword binary data area to be set to the length of the country from the client certificate.
`STATE`	Specifies a pointer reference to be set to the address of the state or province from the client certificate.
`STATELEN`	Specifies a halfword binary data area to be set to the length of the state or province from the client certificate.
`LOCALITY`	Specifies a pointer reference to be set to the address of the locality from the client certificate.
`LOCALITYLEN`	Specifies a halfword binary data area to be set to the length of the locality from the client certificate.
`ORGANIZATION`	Specifies a pointer reference to be set to the address of the organization from the client certificate.
`ORGANIZATLEN`	Specifies a halfword binary data area to be set to the length of the organization from the client certificate.
`ORGUNIT`	Specifies a pointer reference to be set to the address of the organization unit from the client certificate.
`ORGUNITLEN`	Specifies a halfword binary data area to be set to the length of the organization unit from the client certificate.

The following options are not supported:

`SERIALNUM` `SERIALNUMLEN` `USERID`

If a transaction that uses the `EXTRACT CERTIFICATE` API was not started by an SSL client (through `unikixssl`), an `INVREQ RESP` value is returned.