SWI-Prolog SSL Interface

Jan van der Steen
Diff Automatisering v.o.f

Jan Wielemaker
SWI, University of Amsterdam
The Netherlands
E-mail: jan@swi-prolog.org

June 29, 2004

Abstract

This document describes the SWI-Prolog SSL library, a set of predicates which provides secure sockets to Prolog applications, for example to run a secure HTTPS server, or access websites using the `https` protocol. It can also be used to provide authetification and secure data exchange between Prolog processes over the network.

Contents

1 Introduction

Raw TCP/IP networking is dangerous for two reasons. It is hard to tell whether to body you think you are talking to is indead the right one and anyone with access to a subnet through which your data flows can 'tap' the wire and listen for sensitive information such as passwords, creditcard numbers, etc. Secure Socket Layer (SSL) deals with both problems. It uses certificates to establish the identity of the peer and encryption to make it impossible to tap into the wire. SSL allows agent to talk in private and create secure web services.

The SWI-Prolog `ssl` library provides an API very similar to `socket` for raw TCP/IP connections that provides SSL server and client sockets.

2 About SSL

The SWI-Prolog SSL interface is built on top of the OpenSSL library. This library is commonly provided as a standard package in many Linux distributions. The MS-Windows version is built using a binary distribution available from `http://www.slproweb.com/products/Win32OpenSSL.html`.

A good introduction on key- and certificate handling for OpenSSL can be found at `http://www.tldp.org/HOWTO/SSL-Certificates-HOWTO/`

3 Overview of the Prolog API

An SSL server and client can be built with the following (abstracted) predicate calls:

SSL server	SSL client
`ssl_init/3`	`ssl_init/3`
`ssl_accept/3`	
`ssl_open/4`	`ssl_open/3`
...	...
`ssl_exit/1`	`ssl_exit/1`

What follows is a description of each of these functions and the arguments they accept.

ssl_init(-*SSL*, +*Server*, +*Options***)**

Server with legal values `server` or `client` denotes whether the SSL socket will have a server or client role in the established connection. With *Options* various properties of the TCP/IP + SSL connection can be defined, some of which required, some optional. An overview is given below. The handle of the connection is returned in *SSL*.

Below is an overview of the *Options* argument. Some options are only required by the client (C), some are required by the server (marked S), some by both server as client (marked CS).

host(+*HostName***)**

[C] The host to connect to by the client or identified by the server. Both IP addresses and hostnames can be supplied here. This option is required for the client and optional for the server.

port(+*Integer*)

[CS] The port to connect or listen to. This option is required since no default port can sensibly be defined for an abstract layer. The webserver *https* protocol uses port 443.

certificate_file(+*FileName*)

[S] Specify where the certificate file can be found. This can be the same as the key file (see next option).

key_file(+*FileName*)

[S] Specify where the private key can be found. This can be the same as the certificate file.

password(+*Text*)

Specify the password the private key is protected with (if any). If you do not want to store the password you can also specify an application defined handler to return the password (see next option).

pem_password_hook(:*PredicateName*)

In case a password is required to access the private key the supplied function will be called to fetch it. The function has the following prototype: `function`(+*SSL*, -*Password*)

cacert_file(+*FileName*)

Specify a file containing certificate keys which will thus automatically be verified as trusted. You can also install an application defined handler to verify certificates (see next option).

cert_verify_hook(:*PredicateName*)

In case a certificate cannot be verified or has some properties which make it invalid (invalid validity date for example) the supplied function will be called to ask its opinion about the certificate. The predicate is calles as follows: `function`(+*SSL*, +*Certificate*, +*Error*). Access will be granted iff the predicate succeeds.

cert(+*Boolean*)

Trigger the sending of our certificate as specified using the option `certificate_file` described earlier. For a server this option is automatically turned on.

peer_cert(+*Boolean*)

Trigger the request of our peer's certificate while establishing the SSL layer. This option is automatically turned on in a client SSL socket.

ssl_accept(+*SSL*, -*Socket*, -*Peer*)

This predicate is used in the server to accept new connections. *Socket* is unified to a socket that must be passed to `ssl_open/4` and *Peer* is unified with a term `ip/4` denoting the IP address of the peer. The format is compatible to the format used by the TCP/IP library `socket.pl` provided by the clib package.

ssl_open(+*SSL*, +*Socket*, -*In*, -*Out*)

Server side. Attach a Prolog input and output stream to the socket. The *Socket* is automatically closed after both streams are closed. Please note that the output stream is fully buffered and therefore `flush_output/1` is required to actually send the data to the peer.

ssl_open(+*SSL*, -*In*, -*Out*)

Client side. Establish the connection based on the data passed to `ssl_init/3` and create the Prolog I/O streams.

ssl_exit(+*SSL***)**

Clean up all resources related to the SSL + TCP/IP socket layers.

4 Bugs

The current implementation uses blocking (socket) I/O, which implies it is not possible to deal with timeouts and (XPCE) event dispatching. It is adviced to use the SSL library from Prolog threads to prevent the toplevel from blocking, especially on MS-Windows.

We plan to merge the non-blocking socket based implementation used by library `socket` with OpenSSL.

5 Acknowledgments

The development of the SWI-Prolog SSL interface has been sponsored by Scientific Software and Systems Limited.