An Introduction to OpenSSL Programming, Part II of II

Oct 09, 2001 By Eric Rescorla (/user/800894)
in

Part two of the series that started in the September 2001 issue.

The quickest and easiest way to secure a TCP-based network application is with SSL. If you're working in C, your best choice is probably to use OpenSSL (www.openssl.org (http://www.openssl.org)). OpenSSL is a free (BSD-style license) implementation of SSL/TLS based on Eric Young's SSLeay package. Unfortunately, the documentation and sample code distributed with OpenSSL leave something to be desired. Where they exist, the manual pages are pretty good, but they often miss the big picture, as manual pages are intended as a reference, not a tutorial.

Here, we provide an introduction to OpenSSL programming. The OpenSSL API is vast and complicated, so we don't attempt to provide complete coverage. Rather, the idea is to teach you enough to work effectively from the manual pages. In the first part, published in the September issue of *Linux Journal*, we introduced the basic features of OpenSSL. In this article we show how to use a number of advanced features such as session resumption and client authentication.

Source Code

For space reasons, this article only includes excerpts from the source code. The complete source code is available in machine-readable format from the author's web site at www.rtfm.com/openssl-examples (http://www.rtfm.com/openssl-examples) .

Our Programs

For most of this article we'll be extending the simple client/server pair (wclient and wserver) we presented in Part 1 to create two new programs: wclient2 and wserver2. Like wclient, wclient2 is a simple HTTPS (see RFC 2818) client. It initiates an SSL connection to the server and then transmits an HTTP request over that connection. It then waits for the response from the server and prints it to the screen. This is a vastly simplified version of the functionality found in programs like fetch and cURL.

wserver2 is a simple HTTPS server: it waits for TCP connections from clients, and when it accepts one it negotiates an SSL connection. Once the connection is negotiated, it reads the client's HTTP request. It then transmits the HTTP response to the client. Once the response is transmitted it closes the connection.

Towards the end of this article we'll show a more interactive client (sclient) that is usable for debugging or simple remote login.

Session Resumption

When a client and server establish an SSL connection for the first time, they need to establish a shared key called the master_secret. The master_secret is then used to create all the bulk encryption keys used to protect the traffic. The master_secret is almost invariably established using one of two public key algorithms: RSA or Diffie-Hellman (DH). Unfortunately, both of these algorithms are quite slow--on my Pentium II/400 a single RSA operation takes 19 ms. DH can be even slower.

An operation that takes 19 ms may not sound that expensive, but if it has to be done for every connection, it limits the server's throughput to less than 50 connections/second. Without SSL, most web servers can handle hundreds of connections a second. Thus, having to do a key exchange for every client seriously degrades the performance of a web server. In order to improve performance, SSL contains a "session resumption" feature that allows a client/server pair to skip this time consuming step if they have already established a master_secret in a previous connection.

The performance of RSA is highly asymmetric. Operations performed with the private key (such as when the server decrypts the shared key) are much slower than operations performed with the public key. Thus, in most situations most of the computational load is on the server.

What's a Session?

SSL makes a distinction between a connection and a session. A connection represents one specific communications channel (typically mapped to a TCP connection), along with its keys, cipher choices, sequence number state, etc. A session is a virtual construct representing the negotiated algorithms and the master_secret . A new session is created every time a given client and server go through a full key exchange and establish a new master_secret.

Multiple connections can be associated with a given session. Although all connections in a given session share the same master_secret, each has its own encryption keys. This is absolutely necessary for security reasons because reuse of bulk keying material can be extremely dangerous. Resumption allows the generation of a new set of bulk keys and IVs from a common master_secret because the keys depend on the random values, which are fresh for each connection. The new random values are combined with the old master_secret to produce new keys.

Comments

Comment viewing options

Threaded list - expanded | Date - newest first | 50 comments per page | Save settings

Select your preferred way to display the comments and click "Save settings" to activate your changes.

Block socket with SSL_MODE_AUTO_RETRY (/article/5487#comment-1300)
Submitted by Anonymous on Oct 04, 2004.

If I use block socket with SSL_MODE_AUTO_RETRY
The flag SSL_MODE_AUTO_RETRY will cause read/write operations to only return after the handshake and successful completion.

Do I have to handle the retrying in SSL_Read and SSL_Write? Isn't it easier to do this way? I know it hurts throughput a little, but is it a big deal?

Post new comment

Subject:

Comment: *