Red Hat Enterprise Linux Release 9.2 Manual Pages on 'DTLSv1_listen.3ossl' command

$ man DTLSv1_listen.3ossl

DTLSV1_LISTEN(3ossl) OpenSSL DTLSV1_LISTEN(3ossl)

NAME

SSL_stateless, DTLSv1_listen - Statelessly listen for incoming connections

SYNOPSIS

```
#include <openssl/ssl.h>

int SSL_stateless(SSL *s);
int DTLSv1_listen(SSL *ssl, BIO_ADDR *peer);
```

DESCRIPTION

SSL_stateless() statelessly listens for new incoming TLSv1.3 connections. DTLSv1_listen() statelessly listens for new incoming DTLS connections. If a ClientHello is received that does not contain a cookie, then they respond with a request for a new ClientHello that does contain a cookie. If a ClientHello is received with a cookie that is verified then the function returns in order to enable the handshake to be completed (for example by using SSL_accept()).

NOTES

Some transport protocols (such as UDP) can be susceptible to amplification attacks. Unlike TCP there is no initial connection setup in UDP that validates that the client can actually receive messages on its advertised source address. An attacker could forge its source IP address and then send handshake initiation messages to the server. The server would then send its response to the forged source IP. If the response messages are larger than the original message then the amplification attack has succeeded.

If DTLS is used over UDP (or any datagram based protocol that does not validate the source IP) then it is susceptible to this type of attack. TLSv1.3 is designed to operate over a stream-based transport protocol (such as TCP). If TCP is being used then there is no need to use SSL_stateless(). However, some stream-based transport protocols (e.g. QUIC) may not validate the source address. In this case a TLSv1.3 application would be susceptible to this attack.

As a countermeasure to this issue TLSv1.3 and DTLS include a stateless cookie mechanism. The idea is that when a client attempts to connect to a server it sends a ClientHello message. The server responds with a HelloRetryRequest (in TLSv1.3) or a HelloVerifyRequest (in DTLS) which contains a unique cookie. The client then resends the ClientHello, but this time includes the cookie in the message thus proving that the client is capable of receiving messages sent to that address. All of this can be done by the server without allocating any state, and thus without consuming expensive resources.

OpenSSL implements this capability via the SSL_stateless() and DTLSv1_listen() functions. The ssl parameter should be a newly allocated SSL object with its read and write BIOs set, in the same way as might be done for a call to SSL_accept(). Typically, for DTLS, the read BIO will be in an "unconnected" state and thus capable of receiving messages from any peer.

When a ClientHello is received that contains a cookie that has been verified, then these functions will return with the ssl parameter updated into a state where the handshake can be continued by a call to (for example) SSL_accept(). Additionally, for DTLSv1_listen(), the BIO_ADDR pointed to by peer will be filled in with details of the peer that sent the ClientHello. If the underlying BIO is unable to obtain the BIO_ADDR of the peer (for example because the BIO does not support this), then *peer will be cleared and the family set to AF_UNSPEC. Typically user code is expected to "connect" the underlying socket to the peer and continue the handshake in a connected state.

Prior to calling DTLSv1_listen() user code must ensure that cookie generation and verification callbacks have been set up using SSL_CTX_set_cookie_generate_cb(3) and SSL_CTX_set_cookie_verify_cb(3) respectively. For SSL_stateless(), SSL_CTX_set_stateless_cookie_generate_cb(3) and SSL_CTX_set_stateless_cookie_verify_cb(3) must be used instead.

Since DTLSv1_listen() operates entirely statelessly whilst processing incoming ClientHellos it is unable to process fragmented messages (since this would require the allocation of state). An implication of this is that DTLSv1_listen() only supports ClientHellos that fit inside a single datagram.

For SSL_stateless() if an entire ClientHello message cannot be read without the "read" BIO becoming empty then the SSL_stateless() call will fail. It is the application's responsibility to ensure that data read from the "read" BIO during a single SSL_stateless() call is all from the same peer.

SSL_stateless() will fail (with a 0 return value) if some TLS version less than TLSv1.3 is used.

Both SSL_stateless() and DTLSv1_listen() will clear the error queue when they start.

RETURN VALUES

For SSL_stateless() a return value of 1 indicates success and the ssl object will be set up ready to continue the handshake. A return value of 0 or -1 indicates failure. If the value is 0 then a HelloRetryRequest was sent. A value of -1 indicates any other error. User code may retry the SSL_stateless() call.

For DTLSv1_listen() a return value of >= 1 indicates success. The ssl object will be set up ready to continue the handshake. the peer value will also be filled in.

A return value of 0 indicates a non-fatal error. This could (for example) be because of nonblocking IO, or some invalid message having been received from a peer. Errors may be placed on the OpenSSL error queue with further information if appropriate. Typically user code is expected to retry the call to DTLSv1_listen() in the event of a non-fatal error.

A return value of <0 indicates a fatal error. This could (for example) be because of a failure to allocate sufficient memory for the operation.

For DTLSv1_listen(), prior to OpenSSL 1.1.0, fatal and non-fatal errors both produce return codes <= 0 (in typical implementations user code treats all errors as non-fatal), whilst return codes >0 indicate success.

SEE ALSO

SSL_CTX_set_cookie_generate_cb(3), SSL_CTX_set_cookie_verify_cb(3),

SSL_CTX_set_stateless_cookie_generate_cb(3), SSL_CTX_set_stateless_cookie_verify_cb(3), SSL_get_error(3),

SSL_accept(3), ssl(7), bio(7)

HISTORY

The SSL_stateless() function was added in OpenSSL 1.1.1.

The DTLSv1_listen() return codes were clarified in OpenSSL 1.1.0. The type of "peer" also changed in OpenSSL 1.1.0.

COPYRIGHT