NAME

SSL_read_ex, SSL_read, SSL_peek_ex, SSL_peek - read bytes from a TLS/SSL connection

SYNOPSIS

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

int SSL_read_ex(SSL *ssl, void *buf, size_t num, size_t *readbytes);
int SSL_read(SSL *ssl, void *buf, int num);

int SSL_peek_ex(SSL *ssl, void *buf, size_t num, size_t *readbytes);
int SSL_peek(SSL *ssl, void *buf, int num);
```

DESCRIPTION

SSL_read_ex() and **SSL_read()** try to read **num** bytes from the specified **ssl** into the buffer **buf**. On success **SSL_read_ex()** will store the number of bytes actually read in ***readbytes**.

SSL_peek_ex() and **SSL_peek()** are identical to **SSL_read_ex()** and **SSL_read()** respectively except no bytes are actually removed from the underlying BIO during the read, so that a subsequent call to **SSL_read_ex()** or **SSL_read()** will yield at least the same bytes.

NOTES

In the paragraphs below a "read function" is defined as one of **SSL_read_ex()**, **SSL_read()**, **SSL_peek_ex()** or **SSL_peek()**.

If necessary, a read function will negotiate a TLS/SSL session, if not already explicitly performed by **SSL_connect**(3) or **SSL_accept**(3). If the peer requests a re-negotiation, it will be performed transparently during the read function operation. The behaviour of the read functions depends on the underlying BIO.

For the transparent negotiation to succeed, the **ssl** must have been initialized to client or server mode. This is being done by calling **SSL_set_connect_state**(3) or **SSL_set_accept_state()** before the first invocation of a read function.

The read functions work based on the SSL/TLS records. The data are received in records (with a maximum record size of 16kB). Only when a record has been completely received, can it be processed (decryption and check of integrity). Therefore, data that was not retrieved at the last read call can still be buffered inside the SSL layer and will be retrieved on the next read call. If **num** is higher than the number of bytes buffered then the read functions will return with the bytes buffered. If no more bytes are in the buffer, the read functions will trigger the processing of the next record. Only when the record has been received and processed completely will the read functions return reporting success. At

most the contents of one record will be returned. As the size of an SSL/TLS record may exceed the maximum packet size of the underlying transport (e.g. TCP), it may be necessary to read several packets from the transport layer before the record is complete and the read call can succeed.

If **SSL_MODE_AUTO_RETRY** has been switched off and a non-application data record has been processed, the read function can return and set the error to **SSL_ERROR_WANT_READ**. In this case there might still be unprocessed data available in the **BIO**. If read ahead was set using **SSL_CTX_set_read_ahead**(3), there might also still be unprocessed data available in the **SSL**. This behaviour can be controlled using the **SSL_CTX_set_mode**(3) call.

If the underlying BIO is **blocking**, a read function will only return once the read operation has been finished or an error occurred, except when a non-application data record has been processed and **SSL_MODE_AUTO_RETRY** is not set. Note that if **SSL_MODE_AUTO_RETRY** is set and only non-application data is available the call will hang.

If the underlying BIO is **nonblocking**, a read function will also return when the underlying BIO could not satisfy the needs of the function to continue the operation. In this case a call to **SSL_get_error**(3) with the return value of the read function will yield **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. As at any time it's possible that non-application data needs to be sent, a read function can also cause write operations. The calling process then must repeat the call after taking appropriate action to satisfy the needs of the read function. The action depends on the underlying BIO. When using a nonblocking socket, nothing is to be done, but **select()** can be used to check for the required condition. When using a buffering BIO, like a BIO pair, data must be written into or retrieved out of the BIO before being able to continue.

SSL_pending(3) can be used to find out whether there are buffered bytes available for immediate retrieval. In this case the read function can be called without blocking or actually receiving new data from the underlying socket.

RETURN VALUES

SSL_read_ex() and **SSL_peek_ex()** will return 1 for success or 0 for failure. Success means that 1 or more application data bytes have been read from the SSL connection. Failure means that no bytes could be read from the SSL connection. Failures can be retryable (e.g. we are waiting for more bytes to be delivered by the network) or non-retryable (e.g. a fatal network error). In the event of a failure call **SSL_get_error**(3) to find out the reason which indicates whether the call is retryable or not.

For **SSL_read()** and **SSL_peek()** the following return values can occur:

> 0 The read operation was successful. The return value is the number of bytes actually read from the TLS/SSL connection.

<= 0

The read operation was not successful, because either the connection was closed, an error occurred or action must be taken by the calling process. Call **SSL_get_error**(3) with the return value **ret** to find out the reason.

Old documentation indicated a difference between 0 and -1, and that -1 was retryable. You should instead call **SSL_get_error()** to find out if it's retryable.

SEE ALSO

SSL_get_error(3), **SSL_write_ex**(3), **SSL_CTX_set_mode**(3), **SSL_CTX_new**(3), **SSL_connect**(3), **SSL_accept**(3) **SSL_set_connect_state**(3), **SSL_pending**(3), **SSL_shutdown**(3), **SSL_set_shutdown**(3), **ssl**(7), **bio**(7)

HISTORY

The **SSL_read_ex()** and **SSL_peek_ex()** functions were added in OpenSSL 1.1.1.

COPYRIGHT