NAME

SSL_read - read bytes from a TLS/SSL connection

SYNOPSIS

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

int SSL_read(SSL *ssl, void *buf, int num);
```

DESCRIPTION

SSL_read() tries to read **num** bytes from the specified **ssl** into the buffer **buf**.

NOTES

If necessary, *SSL_read()* 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 *SSL_read()* operation. The behaviour of *SSL_read()* 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 call to an *SSL_read()* or *SSL_write(3)* function.

SSL_read() works based on the SSL/TLS records. The data are received in records (with a maximum record size of 16kB for SSLv3/TLSv1). Only when a record has been completely received, it can be processed (decryption and check of integrity). Therefore data that was not retrieved at the last call of *SSL_read()* can still be buffered inside the SSL layer and will be retrieved on the next call to *SSL_read()*. If **num** is higher than the number of bytes buffered, *SSL_read()* will return with the bytes buffered. If no more bytes are in the buffer, *SSL_read()* will trigger the processing of the next record. Only when the record has been received and processed completely, *SSL_read()* will return reporting success. At most the contents of the 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 *SSL_read()* can succeed.

If the underlying BIO is **blocking**, *SSL_read()* will only return, once the read operation has been finished or an error occurred, except when a renegotiation take place, in which case a SSL_ERROR_WANT_READ may occur. This behaviour can be controlled with the SSL_MODE_AUTO_RETRY flag of the *SSL_CTX_set_mode(3)* call.

If the underlying BIO is **non-blocking**, *SSL_read()* will also return when the underlying BIO could not satisfy the needs of *SSL_read()* to continue the operation. In this case a call to *SSL_get_error(3)* with the return value of *SSL_read()* will yield **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. As at any time a re-negotiation is possible, a call to *SSL_read()* can also cause write operations! The calling process then must repeat the call after taking appropriate action to satisfy the needs of *SSL_read()*. The action depends on the underlying BIO. When using a non-blocking 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 *SSL_read()* can be called without blocking or actually receiving new data from the underlying socket.

WARNING

When an *SSL_read()* operation has to be repeated because of **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**, it must be repeated with the same arguments.

RETURN VALUES

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(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(3)*, *bio(3)*

COPYRIGHT