NAME

SSL_write - write bytes to a TLS/SSL connection

SYNOPSIS

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

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

DESCRIPTION

SSL_write() writes **num** bytes from the buffer **buf** into the specified **ssl** connection.

NOTES

If necessary, *SSL_write()* 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_write()* operation. The behaviour of *SSL_write()* 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(3)* or*SSL_write()* function.

If the underlying BIO is **blocking**, *SSL_write()* will only return, once the write 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_write()* will also return, when the underlying BIO could not satisfy the needs of *SSL_write()* to continue the operation. In this case a call to *SSL_get_error(3)* with the return value of *SSL_write()* will yield **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. As at any time a re-negotiation is possible, a call to *SSL_write()* can also cause read operations! The calling process then must repeat the call after taking appropriate action to satisfy the needs of *SSL_write()*. 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_write() will only return with success, when the complete contents of **buf** of length **num** has been written. This default behaviour can be changed with the SSL_MODE_ENABLE_PARTIAL_WRITE option of *SSL_CTX_set_mode(3)*. When this flag is set,*SSL_write()* will also return with success, when a partial write has been successfully completed. In this case the *SSL_write()* operation is considered completed. The bytes are sent and a new *SSL_write()* operation with a new buffer (with the already sent bytes removed) must be started. A partial write is performed with the size of a message block, which is 16kB for SSLv3/TLSv1.

WARNING

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

When calling *SSL_write()* with num=0 bytes to be sent the behaviour is undefined.

RETURN VALUES

The following return values can occur:

> 0 The write operation was successful, the return value is the number of bytes actually written to the TLS/SSL connection.

<= 0
: The write 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()* 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_read(3)*, *SSL_CTX_set_mode(3)*, *SSL_CTX_new(3)*, *SSL_connect(3)*, *SSL_accept(3)* *SSL_set_connect_state(3)*, *ssl(3)*, *bio(3)*

COPYRIGHT