NAME

SSL_set_max_early_data, SSL_CTX_set_max_early_data, SSL_get_max_early_data, SSL_CTX_get_max_early_data, SSL_set_recv_max_early_data, SSL_CTX_set_recv_max_early_data, SSL_get_recv_max_early_data, SSL_CTX_get_recv_max_early_data, SSL_SESSION_get_max_early_data, SSL_SESSION_set_max_early_data, SSL_write_early_data, SSL_read_early_data, SSL_get_early_data_status, SSL_allow_early_data_cb_fn, SSL_CTX_set_allow_early_data_cb, SSL_set_allow_early_data_cb - functions for sending and receiving early data

SYNOPSIS

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

int SSL_CTX_set_max_early_data(SSL_CTX *ctx, uint32_t max_early_data);
uint32_t SSL_CTX_get_max_early_data(const SSL_CTX *ctx);
int SSL_set_max_early_data(SSL *s, uint32_t max_early_data);
uint32_t SSL_get_max_early_data(const SSL *s);

int SSL_CTX_set_recv_max_early_data(SSL_CTX *ctx, uint32_t recv_max_early_data);
uint32_t SSL_CTX_get_recv_max_early_data(const SSL_CTX *ctx);
int SSL_set_recv_max_early_data(SSL *s, uint32_t recv_max_early_data);
uint32_t SSL_get_recv_max_early_data(const SSL *s);

uint32_t SSL_SESSION_get_max_early_data(const SSL_SESSION *s);
int SSL_SESSION_set_max_early_data(SSL_SESSION *s, uint32_t max_early_data);

int SSL_write_early_data(SSL *s, const void *buf, size_t num, size_t *written);

int SSL_read_early_data(SSL *s, void *buf, size_t num, size_t *readbytes);

int SSL_get_early_data_status(const SSL *s);


typedef int (*SSL_allow_early_data_cb_fn)(SSL *s, void *arg);

void SSL_CTX_set_allow_early_data_cb(SSL_CTX *ctx,
                                     SSL_allow_early_data_cb_fn cb,
                                     void *arg);
void SSL_set_allow_early_data_cb(SSL *s,
                                 SSL_allow_early_data_cb_fn cb,
                                 void *arg);
```

DESCRIPTION

These functions are used to send and receive early data where TLSv1.3 has been negotiated. Early data can be sent by the client immediately after its initial ClientHello without having to wait for the server to complete the handshake. Early data can be sent if a session has previously been established with the server or when establishing a new session using an out-of-band PSK, and only when the server is known to support it. Additionally these functions can be used to send data from the server to the client when the client has not yet completed the authentication stage of the handshake.

Early data has weaker security properties than other data sent over an SSL/TLS connection. In particular the data does not have forward secrecy. There are also additional considerations around replay attacks (see "REPLAY PROTECTION" below). For these reasons extreme care should be exercised when using early data. For specific details, consult the TLS 1.3 specification.

When a server receives early data it may opt to immediately respond by sending application data back to the client. Data sent by the server at this stage is done before the full handshake has been completed. Specifically the client’s authentication messages have not yet been received, i.e. the client is unauthenticated at this point and care should be taken when using this capability.

A server or client can determine whether the full handshake has been completed or not by calling **SSL_is_init_finished**(3).

On the client side, the function **SSL_SESSION_get_max_early_data()** can be used to determine if a session established with a server can be used to send early data. If the session cannot be used then this function will return 0. Otherwise it will return the maximum number of early data bytes that can be sent.

The function **SSL_SESSION_set_max_early_data()** sets the maximum number of early data bytes that can be sent for a session. This would typically be used when creating a PSK session file (see **SSL_CTX_set_psk_use_session_callback**(3)). If using a ticket based PSK then this is set automatically to the value provided by the server.

A client uses the function **SSL_write_early_data()** to send early data. This function is similar to the **SSL_write_ex**(3) function, but with the following differences. See **SSL_write_ex**(3) for information on how to write bytes to the underlying connection, and how to handle any errors that may arise. This page describes the differences between **SSL_write_early_data()** and **SSL_write_ex**(3).

When called by a client, **SSL_write_early_data()** must be the first IO function called on a new connection, i.e. it must occur before any calls to **SSL_write_ex**(3), **SSL_read_ex**(3), **SSL_connect**(3), **SSL_do_handshake**(3) or other similar functions. It may be called multiple times to stream data to the server, but the total number of bytes written must not exceed the value returned from

SSL_SESSION_get_max_early_data(). Once the initial **SSL_write_early_data()** call has completed successfully the client may interleave calls to **SSL_read_ex**(3) and **SSL_read**(3) with calls to **SSL_write_early_data()** as required.

If **SSL_write_early_data()** fails you should call **SSL_get_error**(3) to determine the correct course of action, as for **SSL_write_ex**(3).

When the client no longer wishes to send any more early data then it should complete the handshake by calling a function such as **SSL_connect**(3) or **SSL_do_handshake**(3). Alternatively you can call a standard write function such as **SSL_write_ex**(3), which will transparently complete the connection and write the requested data.

A server may choose to ignore early data that has been sent to it. Once the connection has been completed you can determine whether the server accepted or rejected the early data by calling **SSL_get_early_data_status()**. This will return SSL_EARLY_DATA_ACCEPTED if the data was accepted, SSL_EARLY_DATA_REJECTED if it was rejected or SSL_EARLY_DATA_NOT_SENT if no early data was sent. This function may be called by either the client or the server.

A server uses the **SSL_read_early_data()** function to receive early data on a connection for which early data has been enabled using **SSL_CTX_set_max_early_data()** or **SSL_set_max_early_data()**. As for **SSL_write_early_data()**, this must be the first IO function called on a connection, i.e. it must occur before any calls to **SSL_write_ex**(3), **SSL_read_ex**(3), **SSL_accept**(3), **SSL_do_handshake**(3), or other similar functions.

SSL_read_early_data() is similar to **SSL_read_ex**(3) with the following differences. Refer to **SSL_read_ex**(3) for full details.

SSL_read_early_data() may return 3 possible values:

SSL_READ_EARLY_DATA_ERROR

This indicates an IO or some other error occurred. This should be treated in the same way as a 0 return value from **SSL_read_ex**(3).

SSL_READ_EARLY_DATA_SUCCESS

This indicates that early data was successfully read. This should be treated in the same way as a 1 return value from **SSL_read_ex**(3). You should continue to call **SSL_read_early_data()** to read more data.

SSL_READ_EARLY_DATA_FINISH

This indicates that no more early data can be read. It may be returned on the first call to

SSL_read_early_data() if the client has not sent any early data, or if the early data was rejected.

Once the initial **SSL_read_early_data()** call has completed successfully (i.e. it has returned SSL_READ_EARLY_DATA_SUCCESS or SSL_READ_EARLY_DATA_FINISH) then the server may choose to write data immediately to the unauthenticated client using **SSL_write_early_data()**. If **SSL_read_early_data()** returned SSL_READ_EARLY_DATA_FINISH then in some situations (e.g. if the client only supports TLSv1.2) the handshake may have already been completed and calls to **SSL_write_early_data()** are not allowed. Call **SSL_is_init_finished**(3) to determine whether the handshake has completed or not. If the handshake is still in progress then the server may interleave calls to **SSL_write_early_data()** with calls to **SSL_read_early_data()** as required.

Servers must not call **SSL_read_ex**(3), **SSL_read**(3), **SSL_write_ex**(3) or **SSL_write**(3) until **SSL_read_early_data()** has returned with SSL_READ_EARLY_DATA_FINISH. Once it has done so the connection to the client still needs to be completed. Complete the connection by calling a function such as **SSL_accept**(3) or **SSL_do_handshake**(3). Alternatively you can call a standard read function such as **SSL_read_ex**(3), which will transparently complete the connection and read the requested data. Note that it is an error to attempt to complete the connection before **SSL_read_early_data()** has returned SSL_READ_EARLY_DATA_FINISH.

Only servers may call **SSL_read_early_data()**.

Calls to **SSL_read_early_data()** may, in certain circumstances, complete the connection immediately without further need to call a function such as **SSL_accept**(3). This can happen if the client is using a protocol version less than TLSv1.3. Applications can test for this by calling **SSL_is_init_finished**(3). Alternatively, applications may choose to call **SSL_accept**(3) anyway. Such a call will successfully return immediately with no further action taken.

When a session is created between a server and a client the server will specify the maximum amount of any early data that it will accept on any future connection attempt. By default the server does not accept early data; a server may indicate support for early data by calling **SSL_CTX_set_max_early_data()** or **SSL_set_max_early_data()** to set it for the whole SSL_CTX or an individual SSL object respectively. The **max_early_data** parameter specifies the maximum amount of early data in bytes that is permitted to be sent on a single connection. Similarly the **SSL_CTX_get_max_early_data()** and **SSL_get_max_early_data()** functions can be used to obtain the current maximum early data settings for the SSL_CTX and SSL objects respectively. Generally a server application will either use both of **SSL_read_early_data()** and **SSL_CTX_set_max_early_data()** (or **SSL_set_max_early_data()**), or neither of them, since there is no practical benefit from using only one of them. If the maximum early data setting for a server is nonzero then replay protection is automatically enabled (see "REPLAY PROTECTION" below).

If the server rejects the early data sent by a client then it will skip over the data that is sent. The maximum amount of received early data that is skipped is controlled by the recv_max_early_data setting. If a client sends more than this then the connection will abort. This value can be set by calling **SSL_CTX_set_recv_max_early_data()** or **SSL_set_recv_max_early_data()**. The current value for this setting can be obtained by calling **SSL_CTX_get_recv_max_early_data()** or **SSL_get_recv_max_early_data()**. The default value for this setting is 16,384 bytes.

The recv_max_early_data value also has an impact on early data that is accepted. The amount of data that is accepted will always be the lower of the max_early_data for the session and the recv_max_early_data setting for the server. If a client sends more data than this then the connection will abort.

The configured value for max_early_data on a server may change over time as required. However, clients may have tickets containing the previously configured max_early_data value. The recv_max_early_data should always be equal to or higher than any recently configured max_early_data value in order to avoid aborted connections. The recv_max_early_data should never be set to less than the current configured max_early_data value.

Some server applications may wish to have more control over whether early data is accepted or not, for example to mitigate replay risks (see "REPLAY PROTECTION" below) or to decline early_data when the server is heavily loaded. The functions **SSL_CTX_set_allow_early_data_cb()** and **SSL_set_allow_early_data_cb()** set a callback which is called at a point in the handshake immediately before a decision is made to accept or reject early data. The callback is provided with a pointer to the user data argument that was provided when the callback was first set. Returning 1 from the callback will allow early data and returning 0 will reject it. Note that the OpenSSL library may reject early data for other reasons in which case this callback will not get called. Notably, the built-in replay protection feature will still be used even if a callback is present unless it has been explicitly disabled using the SSL_OP_NO_ANTI_REPLAY option. See "REPLAY PROTECTION" below.

NOTES

The whole purpose of early data is to enable a client to start sending data to the server before a full round trip of network traffic has occurred. Application developers should ensure they consider optimisation of the underlying TCP socket to obtain a performant solution. For example Nagle’s algorithm is commonly used by operating systems in an attempt to avoid lots of small TCP packets. In many scenarios this is beneficial for performance, but it does not work well with the early data solution as implemented in OpenSSL. In Nagle’s algorithm the OS will buffer outgoing TCP data if a TCP packet has already been sent which we have not yet received an ACK for from the peer. The buffered data will only be transmitted if enough data to fill an entire TCP packet is accumulated, or if the ACK is received from the peer. The initial ClientHello will be sent in the first TCP packet along with any data from the first call to **SSL_write_early_data()**. If the amount of data written will exceed the size of a single TCP packet, or if there are more calls to **SSL_write_early_data()** then that additional data will be sent in subsequent TCP packets which will be buffered by the OS and not sent until an ACK is received for the first packet containing the ClientHello. This means the early data is not actually sent until a complete round trip with the server has occurred which defeats the objective of early data.

In many operating systems the TCP_NODELAY socket option is available to disable Nagle's algorithm. If an application opts to disable Nagle's algorithm consideration should be given to turning it back on again after the handshake is complete if appropriate.

In rare circumstances, it may be possible for a client to have a session that reports a max early data value greater than 0, but where the server does not support this. For example, this can occur if a server has had its configuration changed to accept a lower max early data value such as by calling **SSL_CTX_set_recv_max_early_data()**. Another example is if a server used to support TLSv1.3 but was later downgraded to TLSv1.2. Sending early data to such a server will cause the connection to abort. Clients that encounter an aborted connection while sending early data may want to retry the connection without sending early data as this does not happen automatically. A client will have to establish a new transport layer connection to the server and attempt the SSL/TLS connection again but without sending early data. Note that it is inadvisable to retry with a lower maximum protocol version.

REPLAY PROTECTION

When early data is in use the TLS protocol provides no security guarantees that the same early data was not replayed across multiple connections. As a mitigation for this issue OpenSSL automatically enables replay protection if the server is configured with a nonzero max early data value. With replay protection enabled sessions are forced to be single use only. If a client attempts to reuse a session ticket more than once, then the second and subsequent attempts will fall back to a full handshake (and any early data that was submitted will be ignored). Note that single use tickets are enforced even if a client does not send any early data.

The replay protection mechanism relies on the internal OpenSSL server session cache (see **SSL_CTX_set_session_cache_mode**(3)). When replay protection is being used the server will operate as if the SSL_OP_NO_TICKET option had been selected (see **SSL_CTX_set_options**(3)). Sessions will be added to the cache whenever a session ticket is issued. When a client attempts to resume the session, OpenSSL will check for its presence in the internal cache. If it exists then the resumption is allowed and the session is removed from the cache. If it does not exist then the resumption is not allowed and a full handshake will occur.

Note that some applications may maintain an external cache of sessions (see **SSL_CTX_sess_set_new_cb**(3) and similar functions). It is the application's responsibility to ensure that any sessions in the external cache are also populated in the internal cache and that once removed from the internal cache they are similarly removed from the external cache. Failing to do this could result in an application becoming vulnerable to replay attacks. Note that OpenSSL will lock the internal cache while a session is removed but that lock is not held when the remove session callback (see **SSL_CTX_sess_set_remove_cb**(3)) is called. This could result in a small amount of time where the session has been removed from the internal cache but is still available in the external cache. Applications should be designed with this in mind in order to minimise the possibility of replay attacks.

The OpenSSL replay protection does not apply to external Pre Shared Keys (PSKs) (e.g. see **SSL_CTX_set_psk_find_session_callback**(3)). Therefore, extreme caution should be applied when combining external PSKs with early data.

Some applications may mitigate the replay risks in other ways. For those applications it is possible to turn off the built-in replay protection feature using the **SSL_OP_NO_ANTI_REPLAY** option. See **SSL_CTX_set_options**(3) for details. Applications can also set a callback to make decisions about accepting early data or not. See **SSL_CTX_set_allow_early_data_cb()** above for details.

RETURN VALUES

SSL_write_early_data() returns 1 for success or 0 for failure. In the event of a failure call **SSL_get_error**(3) to determine the correct course of action.

SSL_read_early_data() returns SSL_READ_EARLY_DATA_ERROR for failure, SSL_READ_EARLY_DATA_SUCCESS for success with more data to read and SSL_READ_EARLY_DATA_FINISH for success with no more to data be read. In the event of a failure call **SSL_get_error**(3) to determine the correct course of action.

SSL_get_max_early_data(), **SSL_CTX_get_max_early_data()** and **SSL_SESSION_get_max_early_data()** return the maximum number of early data bytes that may be sent.

SSL_set_max_early_data(), **SSL_CTX_set_max_early_data()** and **SSL_SESSION_set_max_early_data()** return 1 for success or 0 for failure.

SSL_get_early_data_status() returns SSL_EARLY_DATA_ACCEPTED if early data was accepted by the server, SSL_EARLY_DATA_REJECTED if early data was rejected by the server, or SSL_EARLY_DATA_NOT_SENT if no early data was sent.

SEE ALSO

SSL_get_error(3), **SSL_write_ex**(3), **SSL_read_ex**(3), **SSL_connect**(3), **SSL_accept**(3), **SSL_do_handshake**(3), **SSL_CTX_set_psk_use_session_callback**(3), **ssl**(7)

HISTORY

All of the functions described above were added in OpenSSL 1.1.1.

COPYRIGHT