NAME

SSL_CTX_set_generate_session_id, SSL_set_generate_session_id, SSL_has_matching_session_id, GEN_SESSION_CB - manipulate generation of SSL session IDs (server only)

SYNOPSIS

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

typedef int (*GEN_SESSION_CB)(const SSL *ssl, unsigned char *id,
unsigned int *id_len);

int SSL_CTX_set_generate_session_id(SSL_CTX *ctx, GEN_SESSION_CB cb);
int SSL_set_generate_session_id(SSL *ssl, GEN_SESSION_CB, cb);
int SSL_has_matching_session_id(const SSL *ssl, const unsigned char *id,
unsigned int id_len);
```

DESCRIPTION

SSL_CTX_set_generate_session_id() sets the callback function for generating new session ids for SSL/TLS sessions for **ctx** to be **cb**.

SSL_set_generate_session_id() sets the callback function for generating new session ids for SSL/TLS sessions for **ssl** to be **cb**.

SSL_has_matching_session_id() checks, whether a session with id **id** (of length **id_len**) is already contained in the internal session cache of the parent context of **ssl**.

NOTES

When a new session is established between client and server, the server generates a session id. The session id is an arbitrary sequence of bytes. The length of the session id is between 1 and 32 bytes. The session id is not security critical but must be unique for the server. Additionally, the session id is transmitted in the clear when reusing the session so it must not contain sensitive information.

Without a callback being set, an OpenSSL server will generate a unique session id from pseudo random numbers of the maximum possible length. Using the callback function, the session id can be changed to contain additional information like e.g. a host id in order to improve load balancing or external caching techniques.

The callback function receives a pointer to the memory location to put **id** into and a pointer to the maximum allowed length **id_len**. The buffer at location **id** is only guaranteed to have the size **id_len**. The callback is only allowed to generate a shorter id and reduce **id_len**; the callback **must never** increase **id_len** or write to the location **id** exceeding the given limit.

The location **id** is filled with 0x00 before the callback is called, so the callback may only fill part of the possible length and leave **id_len** untouched while maintaining reproducibility.

Since the sessions must be distinguished, session ids must be unique. Without the callback a random number is used, so that the probability of generating the same session id is extremely small (2^256 for SSLv3/TLSv1). In order to assure the uniqueness of the generated session id, the callback must call *SSL_has_matching_session_id()* and generate another id if a conflict occurs. If an id conflict is not resolved, the handshake will fail. If the application codes e.g. a unique host id, a unique process number, and a unique sequence number into the session id, uniqueness could easily be achieved without randomness added (it should however be taken care that no confidential information is leaked this way). If the application can not guarantee uniqueness, it is recommended to use the maximum **id_len** and fill in the bytes not used to code special information with random data to avoid collisions.

SSL_has_matching_session_id() will only query the internal session cache, not the external one. Since the session id is generated before the handshake is completed, it is not immediately added to the cache. If another thread is using the same internal session cache, a race condition can occur in that another thread generates the same session id. Collisions can also occur when using an external session cache, since the external cache is not tested with *SSL_has_matching_session_id()* and the same race condition applies.

The callback must return 0 if it cannot generate a session id for whatever reason and return 1 on success.

EXAMPLES

The callback function listed will generate a session id with the server id given, and will fill the rest with pseudo random bytes:

```
const char session_id_prefix = "www-18";

#define MAX_SESSION_ID_ATTEMPTS 10
static int generate_session_id(const SSL *ssl, unsigned char *id,
unsigned int *id_len)
{
unsigned int count = 0;
do {
RAND_pseudo_bytes(id, *id_len);
/*
* Prefix the session_id with the required prefix. NB: If our
* prefix is too long, clip it - but there will be worse effects
* anyway, eg. the server could only possibly create 1 session
* ID (ie. the prefix!) so all future session negotiations will
* fail due to conflicts.
*/
memcpy(id, session_id_prefix,
(strlen(session_id_prefix) < *id_len) ?
strlen(session_id_prefix) : *id_len);
}
while (SSL_has_matching_session_id(ssl, id, *id_len) &&
(++count < MAX_SESSION_ID_ATTEMPTS));
if (count >= MAX_SESSION_ID_ATTEMPTS)
return 0;
return 1;
}
```

RETURN VALUES

SSL_CTX_set_generate_session_id() and *SSL_set_generate_session_id()* always return 1.

SSL_has_matching_session_id() returns 1 if another session with the same id is already in the cache.

SEE ALSO

ssl(7) , *SSL_get_version(3)*

COPYRIGHT