NAME

BIO_f_ssl, BIO_set_ssl, BIO_get_ssl, BIO_set_ssl_mode, BIO_set_ssl_renegotiate_bytes, BIO_get_num_renegotiates, BIO_set_ssl_renegotiate_timeout, BIO_new_ssl, BIO_new_ssl_connect, BIO_new_buffer_ssl_connect, BIO_ssl_copy_session_id, BIO_ssl_shutdown - SSL BIO

SYNOPSIS

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

BIO_METHOD *BIO_f_ssl(void);

#define BIO_set_ssl(b,ssl,c) BIO_ctrl(b,BIO_C_SET_SSL,c,(char *)ssl)
#define BIO_get_ssl(b,sslp) BIO_ctrl(b,BIO_C_GET_SSL,0,(char *)sslp)
#define BIO_set_ssl_mode(b,client) BIO_ctrl(b,BIO_C_SSL_MODE,client,NULL)
#define BIO_set_ssl_renegotiate_bytes(b,num) \
BIO_ctrl(b,BIO_C_SET_SSL_RENEGOTIATE_BYTES,num,NULL);
#define BIO_set_ssl_renegotiate_timeout(b,seconds) \
BIO_ctrl(b,BIO_C_SET_SSL_RENEGOTIATE_TIMEOUT,seconds,NULL);
#define BIO_get_num_renegotiates(b) \
BIO_ctrl(b,BIO_C_SET_SSL_NUM_RENEGOTIATES,0,NULL);

BIO *BIO_new_ssl(SSL_CTX *ctx,int client);
BIO *BIO_new_ssl_connect(SSL_CTX *ctx);
BIO *BIO_new_buffer_ssl_connect(SSL_CTX *ctx);
int BIO_ssl_copy_session_id(BIO *to,BIO *from);
void BIO_ssl_shutdown(BIO *bio);

#define BIO_do_handshake(b) BIO_ctrl(b,BIO_C_DO_STATE_MACHINE,0,NULL)
```

DESCRIPTION

BIO_f_ssl() returns the SSL BIO method. This is a filter BIO which is a wrapper round the OpenSSL SSL routines adding a BIO ‘‘flavour’’ to SSL I/O.

I/O performed on an SSL BIO communicates using the SSL protocol with the SSLs read and write BIOs. If an SSL connection is not established then an attempt is made to establish one on the first I/O call.

If a BIO is appended to an SSL BIO using *BIO_push()* it is automatically used as the SSL BIOs read and write BIOs.

Calling *BIO_reset()* on an SSL BIO closes down any current SSL connection by calling *SSL_shutdown()*. *BIO_reset()* is then sent to the next BIO in the chain: this will typically disconnect the underlying transport. The SSL BIO is then reset to the initial accept or connect state.

If the close flag is set when an SSL BIO is freed then the internal SSL structure is also freed using *SSL_free()*.

BIO_set_ssl() sets the internal SSL pointer of BIO **b** to **ssl** using the close flag **c**.

BIO_get_ssl() retrieves the SSL pointer of BIO **b**, it can then be manipulated using the standard SSL library functions.

BIO_set_ssl_mode() sets the SSL BIO mode to **client**. If **client** is 1 client mode is set. If **client** is 0 server mode is set.

BIO_set_ssl_renegotiate_bytes() sets the renegotiate byte count to **num**. When set after every **num** bytes of I/O (read and write) the SSL session is automatically renegotiated. **num** must be at least 512 bytes.

BIO_set_ssl_renegotiate_timeout() sets the renegotiate timeout to **seconds**. When the renegotiate timeout elapses the session is automatically renegotiated.

BIO_get_num_renegotiates() returns the total number of session renegotiations due to I/O or timeout.

BIO_new_ssl() allocates an SSL BIO using SSL_CTX **ctx** and using client mode if **client** is non zero.

BIO_new_ssl_connect() creates a new BIO chain consisting of an SSL BIO (using **ctx**) followed by a connect BIO.

BIO_new_buffer_ssl_connect() creates a new BIO chain consisting of a buffering BIO, an SSL BIO (using **ctx**) and a connect BIO.

BIO_ssl_copy_session_id() copies an SSL session id between BIO chains **from** and **to**. It does this by locating the SSL BIOs in each chain and calling *SSL_copy_session_id()* on the internal SSL pointer.

BIO_ssl_shutdown() closes down an SSL connection on BIO chain **bio**. It does this by locating the SSL BIO in the chain and calling *SSL_shutdown()* on its internal SSL pointer.

BIO_do_handshake() attempts to complete an SSL handshake on the supplied BIO and establish the SSL connection. It returns 1 if the connection was established successfully. A zero or negative value is returned if the connection could not be established, the call *BIO_should_retry()* should be used for non blocking connect BIOs to determine if the call should be retried. If an SSL connection has already been established this call has no effect.

NOTES

SSL BIOs are exceptional in that if the underlying transport is non blocking they can still request a retry in exceptional circumstances. Specifically this will happen if a session renegotiation takes place during a *BIO_read()* operation, one case where this happens is when step up occurs.

In OpenSSL 0.9.6 and later the SSL flag SSL_AUTO_RETRY can be set to disable this behaviour. That is when this flag is set an SSL BIO using a blocking transport will never request a retry.

Since unknown *BIO_ctrl()* operations are sent through filter BIOs the servers name and port can be set using *BIO_set_host()* on the BIO returned by *BIO_new_ssl_connect()* without having to locate the connect BIO first.

Applications do not have to call *BIO_do_handshake()* but may wish to do so to separate the handshake process from other I/O processing.

RETURN VALUES

TBA

EXAMPLE

This SSL/TLS client example, attempts to retrieve a page from an SSL/TLS web server. The I/O routines are identical to those of the unencrypted example in *BIO_s_connect(3)*.

```
BIO *sbio, *out;
int len;
char tmpbuf[1024];
SSL_CTX *ctx;
SSL *ssl;

ERR_load_crypto_strings();
ERR_load_SSL_strings();
OpenSSL_add_all_algorithms();

/* We would seed the PRNG here if the platform didn't
* do it automatically
*/

ctx = SSL_CTX_new(SSLv23_client_method());

/* We'd normally set some stuff like the verify paths and
* mode here because as things stand this will connect to
* any server whose certificate is signed by any CA.
*/
```

```
sbio = BIO_new_ssl_connect(ctx);

BIO_get_ssl(sbio, &ssl);

if(!ssl) {
fprintf(stderr, "Can't locate SSL pointer\n");
/* whatever ... */
}

/* Don't want any retries */
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);

/* We might want to do other things with ssl here */

BIO_set_conn_hostname(sbio, "localhost:https");

out = BIO_new_fp(stdout, BIO_NOCLOSE);
if(BIO_do_connect(sbio) <= 0) {
fprintf(stderr, "Error connecting to server\n");
ERR_print_errors_fp(stderr);
/* whatever ... */
}

if(BIO_do_handshake(sbio) <= 0) {
fprintf(stderr, "Error establishing SSL connection\n");
ERR_print_errors_fp(stderr);
/* whatever ... */
}

/* Could examine ssl here to get connection info */

BIO_puts(sbio, "GET / HTTP/1.0\n\n");
for(;;) {
len = BIO_read(sbio, tmpbuf, 1024);
if(len <= 0) break;
BIO_write(out, tmpbuf, len);
}
BIO_free_all(sbio);
BIO_free(out);
```

Here is a simple server example. It makes use of a buffering BIO to allow lines to be read from the SSL BIO using BIO_gets. It creates a pseudo web page containing the actual request from a client and also echoes the request to standard output.

```
BIO *sbio, *bbio, *acpt, *out;
int len;
char tmpbuf[1024];
SSL_CTX *ctx;
SSL *ssl;

ERR_load_crypto_strings();
ERR_load_SSL_strings();
OpenSSL_add_all_algorithms();

/* Might seed PRNG here */
```

```
ctx = SSL_CTX_new(SSLv23_server_method());

if (!SSL_CTX_use_certificate_file(ctx,"server.pem",SSL_FILETYPE_PEM)
|| !SSL_CTX_use_PrivateKey_file(ctx,"server.pem",SSL_FILETYPE_PEM)
|| !SSL_CTX_check_private_key(ctx)) {

fprintf(stderr, "Error setting up SSL_CTX\n");
ERR_print_errors_fp(stderr);
return 0;
}

/* Might do other things here like setting verify locations and
* DH and/or RSA temporary key callbacks
*/

/* New SSL BIO setup as server */
sbio=BIO_new_ssl(ctx,0);

BIO_get_ssl(sbio, &ssl);

if(!ssl) {
fprintf(stderr, "Can't locate SSL pointer\n");
/* whatever ... */
}

/* Don't want any retries */
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);

/* Create the buffering BIO */

bbio = BIO_new(BIO_f_buffer());

/* Add to chain */
sbio = BIO_push(bbio, sbio);

acpt=BIO_new_accept("4433");

/* By doing this when a new connection is established
* we automatically have sbio inserted into it. The
* BIO chain is now 'swallowed' by the accept BIO and
* will be freed when the accept BIO is freed.
*/

BIO_set_accept_bios(acpt,sbio);

out = BIO_new_fp(stdout, BIO_NOCLOSE);

/* Setup accept BIO */
if(BIO_do_accept(acpt) <= 0) {
fprintf(stderr, "Error setting up accept BIO\n");
ERR_print_errors_fp(stderr);
return 0;
}
```

```
/* Now wait for incoming connection */
if(BIO_do_accept(acpt) <= 0) {
fprintf(stderr, "Error in connection\n");
ERR_print_errors_fp(stderr);
return 0;
}

/* We only want one connection so remove and free
* accept BIO
*/

sbio = BIO_pop(acpt);

BIO_free_all(acpt);

if(BIO_do_handshake(sbio) <= 0) {
fprintf(stderr, "Error in SSL handshake\n");
ERR_print_errors_fp(stderr);
return 0;
}

BIO_puts(sbio, "HTTP/1.0 200 OK\r\nContent-type: text/plain\r\n\r\n");
BIO_puts(sbio, "\r\nConnection Established\r\nRequest headers:\r\n");
BIO_puts(sbio, "--------------------------------------------------\r\n");

for(;;) {
len = BIO_gets(sbio, tmpbuf, 1024);
if(len <= 0) break;
BIO_write(sbio, tmpbuf, len);
BIO_write(out, tmpbuf, len);
/* Look for blank line signifying end of headers*/
if((tmpbuf[0] == '\r') || (tmpbuf[0] == '\n')) break;
}

BIO_puts(sbio, "--------------------------------------------------\r\n");
BIO_puts(sbio, "\r\n");

/* Since there is a buffering BIO present we had better flush it */
BIO_flush(sbio);

BIO_free_all(sbio);
```

BUGS

In OpenSSL versions before 1.0.0 the *BIO_pop()* call was handled incorrectly, the I/O BIO reference count was incorrectly incremented (instead of decremented) and dissociated with the SSL BIO even if the SSL BIO was not explicitly being popped (e.g. a pop higher up the chain). Applications which included workarounds for this bug (e.g. freeing BIOs more than once) should be modified to handle this fix or they may free up an already freed BIO.

SEE ALSO

TBA