NAME

SSL_CTX_set_options, SSL_set_options, SSL_CTX_clear_options, SSL_clear_options, SSL_CTX_get_options, SSL_get_options, SSL_get_secure_renegotiation_support - manipulate SSL options

SYNOPSIS

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

uint64_t SSL_CTX_set_options(SSL_CTX *ctx, uint64_t options);
uint64_t SSL_set_options(SSL *ssl, uint64_t options);

uint64_t SSL_CTX_clear_options(SSL_CTX *ctx, uint64_t options);
uint64_t SSL_clear_options(SSL *ssl, uint64_t options);

uint64_t SSL_CTX_get_options(const SSL_CTX *ctx);
uint64_t SSL_get_options(const SSL *ssl);

long SSL_get_secure_renegotiation_support(SSL *ssl);
```

DESCRIPTION

SSL_CTX_set_options() adds the options set via bit-mask in **options** to **ctx**. Options already set before are not cleared!

SSL_set_options() adds the options set via bit-mask in **options** to **ssl**. Options already set before are not cleared!

SSL_CTX_clear_options() clears the options set via bit-mask in **options** to **ctx**.

SSL_clear_options() clears the options set via bit-mask in **options** to **ssl**.

SSL_CTX_get_options() returns the options set for **ctx**.

SSL_get_options() returns the options set for **ssl**.

SSL_get_secure_renegotiation_support() indicates whether the peer supports secure renegotiation. Note, this is implemented via a macro.

NOTES

The behaviour of the SSL library can be changed by setting several options. The options are coded as bit-masks and can be combined by a bitwise **or** operation (|).

SSL_CTX_set_options() and **SSL_set_options()** affect the (external) protocol behaviour of the SSL library. The (internal) behaviour of the API can be changed by using the similar **SSL_CTX_set_mode**(3) and **SSL_set_mode()** functions.

During a handshake, the option settings of the SSL object are used. When a new SSL object is created from a context using **SSL_new()**, the current option setting is copied. Changes to **ctx** do not affect already created SSL objects. **SSL_clear()** does not affect the settings.

The following **bug workaround** options are available:

SSL_OP_CRYPTOPRO_TLSEXT_BUG
: Add server-hello extension from the early version of cryptopro draft when GOST ciphersuite is negotiated. Required for interoperability with CryptoPro CSP 3.x.

SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS
: Disables a countermeasure against a SSL 3.0/TLS 1.0 protocol vulnerability affecting CBC ciphers, which cannot be handled by some broken SSL implementations. This option has no effect for connections using other ciphers.

SSL_OP_SAFARI_ECDHE_ECDSA_BUG
: Don’t prefer ECDHE-ECDSA ciphers when the client appears to be Safari on OS X. OS X 10.8..10.8.3 has broken support for ECDHE-ECDSA ciphers.

SSL_OP_TLSEXT_PADDING
: Adds a padding extension to ensure the ClientHello size is never between 256 and 511 bytes in length. This is needed as a workaround for some implementations.

SSL_OP_ALL
: All of the above bug workarounds.

It is usually safe to use **SSL_OP_ALL** to enable the bug workaround options if compatibility with somewhat broken implementations is desired.

The following **modifying** options are available:

SSL_OP_ALLOW_CLIENT_RENEGOTIATION
: Client-initiated renegotiation is disabled by default. Use this option to enable it.

SSL_OP_ALLOW_NO_DHE_KEX
: In TLSv1.3 allow a non-(ec)dhe based key exchange mode on resumption. This means that there

will be no forward secrecy for the resumed session.

SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION
: Allow legacy insecure renegotiation between OpenSSL and unpatched clients or servers. See the **SECURE RENEGOTIATION** section for more details.

SSL_OP_CIPHER_SERVER_PREFERENCE
: When choosing a cipher, use the server's preferences instead of the client preferences. When not set, the SSL server will always follow the clients preferences. When set, the SSL/TLS server will choose following its own preferences.

SSL_OP_CISCO_ANYCONNECT
: Use Cisco's version identifier of DTLS_BAD_VER when establishing a DTLSv1 connection. Only available when using the deprecated **DTLSv1_client_method()** API.

SSL_OP_CLEANSE_PLAINTEXT
: By default TLS connections keep a copy of received plaintext application data in a static buffer until it is overwritten by the next portion of data. When enabling SSL_OP_CLEANSE_PLAINTEXT deciphered application data is cleansed by calling **OPENSSL_cleanse**(3) after passing data to the application. Data is also cleansed when releasing the connection (e.g. **SSL_free**(3)).

  Since OpenSSL only cleanses internal buffers, the application is still responsible for cleansing all other buffers. Most notably, this applies to buffers passed to functions like **SSL_read**(3), **SSL_peek**(3) but also like **SSL_write**(3).

SSL_OP_COOKIE_EXCHANGE
: Turn on Cookie Exchange as described in RFC4347 Section 4.2.1. Only affects DTLS connections.

SSL_OP_DISABLE_TLSEXT_CA_NAMES
: Disable TLS Extension CA Names. You may want to disable it for security reasons or for compatibility with some Windows TLS implementations crashing when this extension is larger than 1024 bytes.

SSL_OP_ENABLE_KTLS
: Enable the use of kernel TLS. In order to benefit from kernel TLS OpenSSL must have been compiled with support for it, and it must be supported by the negotiated ciphersuites and extensions. The specific ciphersuites and extensions that are supported may vary by platform and kernel version.

The kernel TLS data-path implements the record layer, and the encryption algorithm. The kernel will utilize the best hardware available for encryption. Using the kernel data-path should reduce the memory footprint of OpenSSL because no buffering is required. Also, the throughput should improve because data copy is avoided when user data is encrypted into kernel memory instead of the usual encrypt then copy to kernel.

Kernel TLS might not support all the features of OpenSSL. For instance, renegotiation, and setting the maximum fragment size is not possible as of Linux 4.20.

Note that with kernel TLS enabled some cryptographic operations are performed by the kernel directly and not via any available OpenSSL Providers. This might be undesirable if, for example, the application requires all cryptographic operations to be performed by the FIPS provider.

SSL_OP_ENABLE_MIDDLEBOX_COMPAT

If set then dummy Change Cipher Spec (CCS) messages are sent in TLSv1.3. This has the effect of making TLSv1.3 look more like TLSv1.2 so that middleboxes that do not understand TLSv1.3 will not drop the connection. Regardless of whether this option is set or not CCS messages received from the peer will always be ignored in TLSv1.3. This option is set by default. To switch it off use **SSL_clear_options()**. A future version of OpenSSL may not set this by default.

SSL_OP_IGNORE_UNEXPECTED_EOF

Some TLS implementations do not send the mandatory close_notify alert on shutdown. If the application tries to wait for the close_notify alert but the peer closes the connection without sending it, an error is generated. When this option is enabled the peer does not need to send the close_notify alert and a closed connection will be treated as if the close_notify alert was received.

You should only enable this option if the protocol running over TLS can detect a truncation attack itself, and that the application is checking for that truncation attack.

For more information on shutting down a connection, see **SSL_shutdown**(3).

SSL_OP_LEGACY_SERVER_CONNECT

Allow legacy insecure renegotiation between OpenSSL and unpatched servers **only**. See the **SECURE RENEGOTIATION** section for more details.

SSL_OP_NO_ANTI_REPLAY

By default, when a server is configured for early data (i.e., max_early_data > 0), OpenSSL will switch on replay protection. See **SSL_read_early_data**(3) for a description of the replay protection feature. Anti-replay measures are required to comply with the TLSv1.3 specification. Some applications may be able to mitigate the replay risks in other ways and in such cases the built in

OpenSSL functionality is not required. Those applications can turn this feature off by setting this option. This is a server-side option only. It is ignored by clients.

SSL_OP_NO_COMPRESSION
: Do not use compression even if it is supported. This option is set by default. To switch it off use **SSL_clear_options()**.

SSL_OP_NO_ENCRYPT_THEN_MAC
: Normally clients and servers will transparently attempt to negotiate the RFC7366 Encrypt-then-MAC option on TLS and DTLS connection.

  If this option is set, Encrypt-then-MAC is disabled. Clients will not propose, and servers will not accept the extension.

SSL_OP_NO_EXTENDED_MASTER_SECRET
: Normally clients and servers will transparently attempt to negotiate the RFC7627 Extended Master Secret option on TLS and DTLS connection.

  If this option is set, Extended Master Secret is disabled. Clients will not propose, and servers will not accept the extension.

SSL_OP_NO_QUERY_MTU
: Do not query the MTU. Only affects DTLS connections.

SSL_OP_NO_RENEGOTIATION
: Disable all renegotiation in TLSv1.2 and earlier. Do not send HelloRequest messages, and ignore renegotiation requests via ClientHello.

SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION
: When performing renegotiation as a server, always start a new session (i.e., session resumption requests are only accepted in the initial handshake). This option is not needed for clients.

SSL_OP_NO_SSLv3, SSL_OP_NO_TLSv1, SSL_OP_NO_TLSv1_1, SSL_OP_NO_TLSv1_2, SSL_OP_NO_TLSv1_3, SSL_OP_NO_DTLSv1, SSL_OP_NO_DTLSv1_2
: These options turn off the SSLv3, TLSv1, TLSv1.1, TLSv1.2 or TLSv1.3 protocol versions with TLS or the DTLSv1, DTLSv1.2 versions with DTLS, respectively. As of OpenSSL 1.1.0, these options are deprecated, use **SSL_CTX_set_min_proto_version**(3) and **SSL_CTX_set_max_proto_version**(3) instead.

SSL_OP_NO_TICKET

SSL/TLS supports two mechanisms for resuming sessions: session ids and stateless session tickets.

When using session ids a copy of the session information is cached on the server and a unique id is sent to the client. When the client wishes to resume it provides the unique id so that the server can retrieve the session information from its cache.

When using stateless session tickets the server uses a session ticket encryption key to encrypt the session information. This encrypted data is sent to the client as a "ticket". When the client wishes to resume it sends the encrypted data back to the server. The server uses its key to decrypt the data and resume the session. In this way the server can operate statelessly - no session information needs to be cached locally.

The TLSv1.3 protocol only supports tickets and does not directly support session ids. However, OpenSSL allows two modes of ticket operation in TLSv1.3: stateful and stateless. Stateless tickets work the same way as in TLSv1.2 and below. Stateful tickets mimic the session id behaviour available in TLSv1.2 and below. The session information is cached on the server and the session id is wrapped up in a ticket and sent back to the client. When the client wishes to resume, it presents a ticket in the same way as for stateless tickets. The server can then extract the session id from the ticket and retrieve the session information from its cache.

By default OpenSSL will use stateless tickets. The SSL_OP_NO_TICKET option will cause stateless tickets to not be issued. In TLSv1.2 and below this means no ticket gets sent to the client at all. In TLSv1.3 a stateful ticket will be sent. This is a server-side option only.

In TLSv1.3 it is possible to suppress all tickets (stateful and stateless) from being sent by calling **SSL_CTX_set_num_tickets**(3) or **SSL_set_num_tickets**(3).

SSL_OP_PRIORITIZE_CHACHA

When SSL_OP_CIPHER_SERVER_PREFERENCE is set, temporarily reprioritize ChaCha20-Poly1305 ciphers to the top of the server cipher list if a ChaCha20-Poly1305 cipher is at the top of the client cipher list. This helps those clients (e.g. mobile) use ChaCha20-Poly1305 if that cipher is anywhere in the server cipher list; but still allows other clients to use AES and other ciphers. Requires **SSL_OP_CIPHER_SERVER_PREFERENCE**.

SSL_OP_TLS_ROLLBACK_BUG

Disable version rollback attack detection.

During the client key exchange, the client must send the same information about acceptable SSL/TLS protocol levels as during the first hello. Some clients violate this rule by adapting to the

server's answer. (Example: the client sends a SSLv2 hello and accepts up to SSLv3.1=TLSv1, the server only understands up to SSLv3. In this case the client must still use the same SSLv3.1=TLSv1 announcement. Some clients step down to SSLv3 with respect to the server's answer and violate the version rollback protection.)

The following options no longer have any effect but their identifiers are retained for compatibility purposes:

SSL_OP_NETSCAPE_REUSE_CIPHER_CHANGE_BUG
SSL_OP_MICROSOFT_BIG_SSLV3_BUFFER
SSL_OP_SSLEAY_080_CLIENT_DH_BUG
SSL_OP_TLS_D5_BUG
SSL_OP_TLS_BLOCK_PADDING_BUG
SSL_OP_MSIE_SSLV2_RSA_PADDING
SSL_OP_SSLREF2_REUSE_CERT_TYPE_BUG
SSL_OP_MICROSOFT_SESS_ID_BUG
SSL_OP_NETSCAPE_CHALLENGE_BUG
SSL_OP_PKCS1_CHECK_1
SSL_OP_PKCS1_CHECK_2
SSL_OP_SINGLE_DH_USE
SSL_OP_SINGLE_ECDH_USE
SSL_OP_EPHEMERAL_RSA
SSL_OP_NETSCAPE_CA_DN_BUG
SSL_OP_NETSCAPE_DEMO_CIPHER_CHANGE_BUG

SECURE RENEGOTIATION

OpenSSL always attempts to use secure renegotiation as described in RFC5746. This counters the prefix attack described in CVE-2009-3555 and elsewhere.

This attack has far reaching consequences which application writers should be aware of. In the description below an implementation supporting secure renegotiation is referred to as *patched*. A server not supporting secure renegotiation is referred to as *unpatched*.

The following sections describe the operations permitted by OpenSSL's secure renegotiation implementation.

Patched client and server

Connections and renegotiation are always permitted by OpenSSL implementations.

Unpatched client and patched OpenSSL server

The initial connection succeeds but client renegotiation is denied by the server with a **no_renegotiation** warning alert if TLS v1.0 is used or a fatal **handshake_failure** alert in SSL v3.0.

If the patched OpenSSL server attempts to renegotiate a fatal **handshake_failure** alert is sent. This is because the server code may be unaware of the unpatched nature of the client.

If the option **SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION** is set then renegotiation **always** succeeds.

Patched OpenSSL client and unpatched server

If the option **SSL_OP_LEGACY_SERVER_CONNECT** or **SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION** is set then initial connections and renegotiation between patched OpenSSL clients and unpatched servers succeeds. If neither option is set then initial connections to unpatched servers will fail.

Setting the option **SSL_OP_LEGACY_SERVER_CONNECT** has security implications; clients that are willing to connect to servers that do not implement RFC 5746 secure renegotiation are subject to attacks such as CVE-2009-3555.

OpenSSL client applications wishing to ensure they can connect to unpatched servers should always **set SSL_OP_LEGACY_SERVER_CONNECT**

OpenSSL client applications that want to ensure they can **not** connect to unpatched servers (and thus avoid any security issues) should always **clear SSL_OP_LEGACY_SERVER_CONNECT** using **SSL_CTX_clear_options()** or **SSL_clear_options()**.

The difference between the **SSL_OP_LEGACY_SERVER_CONNECT** and **SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION** options is that **SSL_OP_LEGACY_SERVER_CONNECT** enables initial connections and secure renegotiation between OpenSSL clients and unpatched servers **only**, while **SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION** allows initial connections and renegotiation between OpenSSL and unpatched clients or servers.

RETURN VALUES

SSL_CTX_set_options() and **SSL_set_options()** return the new options bit-mask after adding **options**.

SSL_CTX_clear_options() and **SSL_clear_options()** return the new options bit-mask after clearing **options**.

SSL_CTX_get_options() and **SSL_get_options()** return the current bit-mask.

SSL_get_secure_renegotiation_support() returns 1 is the peer supports secure renegotiation and 0 if it does not.

SEE ALSO

ssl(7), **SSL_new**(3), **SSL_clear**(3), **SSL_shutdown**(3) **SSL_CTX_set_tmp_dh_callback**(3), **SSL_CTX_set_min_proto_version**(3), **openssl-dhparam**(1)

HISTORY

The attempt to always try to use secure renegotiation was added in OpenSSL 0.9.8m.

The **SSL_OP_PRIORITIZE_CHACHA** and **SSL_OP_NO_RENEGOTIATION** options were added in OpenSSL 1.1.1.

The **SSL_OP_NO_EXTENDED_MASTER_SECRET** and **SSL_OP_IGNORE_UNEXPECTED_EOF** options were added in OpenSSL 3.0.

The **SSL_OP_** constants and the corresponding parameter and return values of the affected functions were changed to "uint64_t" type in OpenSSL 3.0. For that reason it is no longer possible use the **SSL_OP_** macro values in preprocessor "#if" conditions. However it is still possible to test whether these macros are defined or not.

COPYRIGHT