NAME

ssl_ct_validation_cb, SSL_enable_ct, SSL_CTX_enable_ct, SSL_disable_ct, SSL_CTX_disable_ct, SSL_set_ct_validation_callback, SSL_CTX_set_ct_validation_callback, SSL_ct_is_enabled, SSL_CTX_ct_is_enabled - control Certificate Transparency policy

SYNOPSIS

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

 typedef int (*ssl_ct_validation_cb)(const CT_POLICY_EVAL_CTX *ctx,
                                     const STACK_OF(SCT) *scts, void *arg);

 int SSL_enable_ct(SSL *s, int validation_mode);
 int SSL_CTX_enable_ct(SSL_CTX *ctx, int validation_mode);
 int SSL_set_ct_validation_callback(SSL *s, ssl_ct_validation_cb callback,
                                    void *arg);
 int SSL_CTX_set_ct_validation_callback(SSL_CTX *ctx,
                                        ssl_ct_validation_cb callback,
                                        void *arg);
 void SSL_disable_ct(SSL *s);
 void SSL_CTX_disable_ct(SSL_CTX *ctx);
 int SSL_ct_is_enabled(const SSL *s);
 int SSL_CTX_ct_is_enabled(const SSL_CTX *ctx);
```

DESCRIPTION

SSL_enable_ct() and **SSL_CTX_enable_ct()** enable the processing of signed certificate timestamps (SCTs) either for a given SSL connection or for all connections that share the given SSL context, respectively. This is accomplished by setting a built-in CT validation callback. The behaviour of the callback is determined by the **validation_mode** argument, which can be either of **SSL_CT_VALIDATION_PERMISSIVE** or **SSL_CT_VALIDATION_STRICT** as described below.

If **validation_mode** is equal to **SSL_CT_VALIDATION_STRICT**, then in a full TLS handshake with the verification mode set to **SSL_VERIFY_PEER**, if the peer presents no valid SCTs the handshake will be aborted. If the verification mode is **SSL_VERIFY_NONE**, the handshake will continue despite lack of valid SCTs. However, in that case if the verification status before the built-in callback was **X509_V_OK** it will be set to **X509_V_ERR_NO_VALID_SCTS** after the callback. Applications can call **SSL_get_verify_result**(3) to check the status at handshake completion, even after session resumption since the verification status is part of the saved session state. See **SSL_set_verify**(3), <**SSL_get_verify_result**(3)>, **SSL_session_reused**(3).

If **validation_mode** is equal to **SSL_CT_VALIDATION_PERMISSIVE**, then the handshake continues, and the verification status is not modified, regardless of the validation status of any SCTs. The application can still inspect the validation status of the SCTs at handshake completion. Note that with session resumption there will not be any SCTs presented during the handshake. Therefore, in applications that delay SCT policy enforcement until after handshake completion, such delayed SCT checks should only be performed when the session is not resumed.

SSL_set_ct_validation_callback() and **SSL_CTX_set_ct_validation_callback()** register a custom callback that may implement a different policy than either of the above. This callback can examine the peer's SCTs and determine whether they are sufficient to allow the connection to continue. The TLS handshake is aborted if the verification mode is not **SSL_VERIFY_NONE** and the callback returns a non-positive result.

An arbitrary callback data argument, **arg**, can be passed in when setting the callback. This will be passed to the callback whenever it is invoked. Ownership of this context remains with the caller.

If no callback is set, SCTs will not be requested and Certificate Transparency validation will not occur.

No callback will be invoked when the peer presents no certificate, e.g. by employing an anonymous (aNULL) cipher suite. In that case the handshake continues as it would had no callback been requested. Callbacks are also not invoked when the peer certificate chain is invalid or validated via **DANE-TA**(2) or **DANE-EE**(3) TLSA records which use a private X.509 PKI, or no X.509 PKI at all, respectively. Clients that require SCTs are expected to not have enabled any aNULL ciphers nor to have specified server verification via **DANE-TA**(2) or **DANE-EE**(3) TLSA records.

SSL_disable_ct() and **SSL_CTX_disable_ct()** turn off CT processing, whether enabled via the built-in or the custom callbacks, by setting a NULL callback. These may be implemented as macros.

SSL_ct_is_enabled() and **SSL_CTX_ct_is_enabled()** return 1 if CT processing is enabled via either **SSL_enable_ct()** or a non-null custom callback, and 0 otherwise.

NOTES

When SCT processing is enabled, OCSP stapling will be enabled. This is because one possible source of SCTs is the OCSP response from a server.

The time returned by **SSL_SESSION_get_time()** will be used to evaluate whether any presented SCTs have timestamps that are in the future (and therefore invalid).

RESTRICTIONS

Certificate Transparency validation cannot be enabled and so a callback cannot be set if a custom client extension handler has been registered to handle SCT extensions

(**TLSEXT_TYPE_signed_certificate_timestamp**).

RETURN VALUES

SSL_enable_ct(), **SSL_CTX_enable_ct()**, **SSL_CTX_set_ct_validation_callback()** and **SSL_set_ct_validation_callback()** return 1 if the **callback** is successfully set. They return 0 if an error occurs, e.g. a custom client extension handler has been setup to handle SCTs.

SSL_disable_ct() and **SSL_CTX_disable_ct()** do not return a result.

SSL_CTX_ct_is_enabled() and **SSL_ct_is_enabled()** return a 1 if a non-null CT validation callback is set, or 0 if no callback (or equivalently a NULL callback) is set.

SEE ALSO

ssl(7), <**SSL_get_verify_result**(3)>, **SSL_session_reused**(3), **SSL_set_verify**(3), **SSL_CTX_set_verify**(3), **SSL_SESSION_get_time**(3)

COPYRIGHT