Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'X509_STORE_CTX_verify_cb.3ossl' command

$ man X509_STORE_CTX_verify_cb.3ossl

X509_STORE_CTX_SET_VERIFY_CB(3ossl) OpenSSLX509_STORE_CTX_SET_VERIFY_CB(3ossl)

NAME

X509_STORE_CTX_get_cleanup, X509_STORE_CTX_get_lookup_crls,

X509_STORE_CTX_get_lookup_certs, X509_STORE_CTX_get_check_policy,

X509_STORE_CTX_get_cert_crl, X509_STORE_CTX_get_check_crl,

X509_STORE_CTX_get_get_crl, X509_STORE_CTX_get_check_revocation,

X509_STORE_CTX_get_check_issued, X509_STORE_CTX_get_get_issuer,

X509_STORE_CTX_get_verify_cb, X509_STORE_CTX_set_verify_cb,

X509_STORE_CTX_verify_cb, X509_STORE_CTX_print_verify_cb - get and set

X509_STORE_CTX components such as verification callback

SYNOPSIS

```
#include <openssl/x509_vfy.h>

typedef int (*X509_STORE_CTX_verify_cb)(int, X509_STORE_CTX *);
int X509_STORE_CTX_print_verify_cb(int ok, X509_STORE_CTX *ctx);

X509_STORE_CTX_verify_cb X509_STORE_CTX_get_verify_cb(X509_STORE_CTX *ctx);

void X509_STORE_CTX_set_verify_cb(X509_STORE_CTX *ctx,
                                  X509_STORE_CTX_verify_cb verify_cb);
```

X509_STORE_CTX_get_issuer_fn X509_STORE_CTX_get_get_issuer(X509_STORE_CTX *ctx);

X509_STORE_CTX_check_issued_fn X509_STORE_CTX_get_check_issued(X509_STORE_CTX *ctx);

X509_STORE_CTX_check_revocation_fn X509_STORE_CTX_get_check_revocation(X509_STORE_CTX *ctx);

X509_STORE_CTX_get_crl_fn X509_STORE_CTX_get_get_crl(X509_STORE_CTX *ctx);

X509_STORE_CTX_check_crl_fn X509_STORE_CTX_get_check_crl(X509_STORE_CTX *ctx);

X509_STORE_CTX_cert_crl_fn X509_STORE_CTX_get_cert_crl(X509_STORE_CTX *ctx);

X509_STORE_CTX_check_policy_fn X509_STORE_CTX_get_check_policy(X509_STORE_CTX *ctx);

X509_STORE_CTX_lookup_certs_fn X509_STORE_CTX_get_lookup_certs(X509_STORE_CTX *ctx);

X509_STORE_CTX_lookup_crls_fn X509_STORE_CTX_get_lookup_crls(X509_STORE_CTX *ctx);

X509_STORE_CTX_cleanup_fn X509_STORE_CTX_get_cleanup(X509_STORE_CTX *ctx);

DESCRIPTION

X509_STORE_CTX_set_verify_cb() sets the verification callback of ctx to verify_cb overwriting any existing callback.

The verification callback can be used to customise the operation of certificate verification, for instance by overriding error conditions or logging errors for debugging purposes.

However, a verification callback is not essential and the default operation is often sufficient.

The ok parameter to the callback indicates the value the callback should return to retain the default behaviour. If it is zero then an error condition is indicated. If it is 1 then no error occurred. If the flag X509_V_FLAG_NOTIFY_POLICY is set then ok is set to 2 to indicate the policy checking is complete.

The ctx parameter to the callback is the X509_STORE_CTX structure that is performing the verification operation. A callback can examine this structure and receive additional information about the error, for example by calling X509_STORE_CTX_get_current_cert(). Additional application data can be passed to the callback via the ex_data

mechanism.

X509_STORE_CTX_print_verify_cb() is a verification callback function that, when a certificate verification has failed, adds an entry to the error queue with code X509_R_CERTIFICATE_VERIFICATION_FAILED and with diagnostic details, including the most relevant fields of the target certificate that failed to verify and, if appropriate, of the available untrusted and trusted certificates.

X509_STORE_CTX_get_verify_cb() returns the value of the current callback for the specific ctx.

X509_STORE_CTX_get_get_issuer(), X509_STORE_CTX_get_check_issued(), X509_STORE_CTX_get_check_revocation(), X509_STORE_CTX_get_get_crl(), X509_STORE_CTX_get_check_crl(), X509_STORE_CTX_get_cert_crl(), X509_STORE_CTX_get_check_policy(), X509_STORE_CTX_get_lookup_certs(), X509_STORE_CTX_get_lookup_crls() and X509_STORE_CTX_get_cleanup() return the function pointers cached from the corresponding X509_STORE, please see X509_STORE_set_verify(3) for more information.

WARNINGS

In general a verification callback should NOT unconditionally return 1 in all circumstances because this will allow verification to succeed no matter what the error. This effectively removes all security from the application because any certificate (including untrusted generated ones) will be accepted.

NOTES

The verification callback can be set and inherited from the parent structure performing the operation. In some cases (such as S/MIME verification) the X509_STORE_CTX structure is created and destroyed internally and the only way to set a custom verification callback is by inheriting it from the associated X509_STORE.

RETURN VALUES

X509_STORE_CTX_set_verify_cb() does not return a value.

EXAMPLES

Default callback operation:

```
int verify_callback(int ok, X509_STORE_CTX *ctx) {
    return ok;
}
```

Simple example, suppose a certificate in the chain is expired and we wish to continue after this error:

```
int verify_callback(int ok, X509_STORE_CTX *ctx) {
    /* Tolerate certificate expiration */
    if (X509_STORE_CTX_get_error(ctx) == X509_V_ERR_CERT_HAS_EXPIRED)
        return 1;
    /* Otherwise don't override */
    return ok;
}
```

More complex example, we don't wish to continue after any certificate has expired just one specific case:

```
int verify_callback(int ok, X509_STORE_CTX *ctx)
{
    int err = X509_STORE_CTX_get_error(ctx);
    X509 *err_cert = X509_STORE_CTX_get_current_cert(ctx);

    if (err == X509_V_ERR_CERT_HAS_EXPIRED) {
        if (check_is_acceptable_expired_cert(err_cert)
            return 1;
```

```
    }
    return ok;
}
```

Full featured logging callback. In this case the bio_err is assumed to
be a global logging BIO, an alternative would to store a BIO in ctx
using ex_data.

```
 int verify_callback(int ok, X509_STORE_CTX *ctx)
{
    X509 *err_cert;
    int err, depth;

    err_cert = X509_STORE_CTX_get_current_cert(ctx);
    err = X509_STORE_CTX_get_error(ctx);
    depth = X509_STORE_CTX_get_error_depth(ctx);

    BIO_printf(bio_err, "depth=%d ", depth);
    if (err_cert) {
        X509_NAME_print_ex(bio_err, X509_get_subject_name(err_cert),
                           0, XN_FLAG_ONELINE);
        BIO_puts(bio_err, "\n");
    }
    else
        BIO_puts(bio_err, "<no cert>\n");
    if (!ok)
        BIO_printf(bio_err, "verify error:num=%d:%s\n", err,
                   X509_verify_cert_error_string(err));
    switch (err) {
    case X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT:
        BIO_puts(bio_err, "issuer= ");
        X509_NAME_print_ex(bio_err, X509_get_issuer_name(err_cert),
                           0, XN_FLAG_ONELINE);
```

```
            BIO_puts(bio_err, "\n");
            break;
        case X509_V_ERR_CERT_NOT_YET_VALID:
        case X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD:
            BIO_printf(bio_err, "notBefore=");
            ASN1_TIME_print(bio_err, X509_get_notBefore(err_cert));
            BIO_printf(bio_err, "\n");
            break;
        case X509_V_ERR_CERT_HAS_EXPIRED:
        case X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD:
            BIO_printf(bio_err, "notAfter=");
            ASN1_TIME_print(bio_err, X509_get_notAfter(err_cert));
            BIO_printf(bio_err, "\n");
            break;
        case X509_V_ERR_NO_EXPLICIT_POLICY:
            policies_print(bio_err, ctx);
            break;
        }
        if (err == X509_V_OK && ok == 2)
            /* print out policies */

        BIO_printf(bio_err, "verify return:%d\n", ok);
        return(ok);
    }
```

SEE ALSO

X509_STORE_CTX_get_error(3) X509_STORE_set_verify_cb_func(3)
X509_STORE_CTX_get_ex_new_index(3)

HISTORY

The X509_STORE_CTX_get_get_issuer(), X509_STORE_CTX_get_check_issued(),
X509_STORE_CTX_get_check_revocation(), X509_STORE_CTX_get_get_crl(),
X509_STORE_CTX_get_check_crl(), X509_STORE_CTX_get_cert_crl(),

X509_STORE_CTX_get_check_policy(), X509_STORE_CTX_get_lookup_certs(), X509_STORE_CTX_get_lookup_crls() and X509_STORE_CTX_get_cleanup() functions were added in OpenSSL 1.1.0.

X509_STORE_CTX_print_verify_cb() was added in OpenSSL 3.0.

COPYRIGHT