NAME

openssl−verification−options − generic X.509 certificate verification options

SYNOPSIS

openssl *command* [*options* ...] [*parameters* ...]

DESCRIPTION

There are many situations where X.509 certificates are verified within the OpenSSL libraries and in various OpenSSL commands.

Certificate verification is implemented by **X509_verify_cert**(3). It is a complicated process consisting of a number of steps and depending on numerous options. The most important of them are detailed in the following sections.

In a nutshell, a valid chain of certificates needs to be built up and verified starting from the *target certificate* that is to be verified and ending in a certificate that due to some policy is trusted. Verification is done relative to the given *purpose*, which is the intended use of the target certificate, such as SSL server, or by default for any purpose.

The details of how each OpenSSL command handles errors are documented on the specific command page.

DANE support is documented in **openssl−s_client**(1), **SSL_CTX_dane_enable**(3), **SSL_set1_host**(3), **X509_VERIFY_PARAM_set_flags**(3), and **X509_check_host**(3).

Trust Anchors

In general, according to RFC 4158 and RFC 5280, a *trust anchor* is any public key and related subject distinguished name (DN) that for some reason is considered trusted and thus is acceptable as the root of a chain of certificates.

In practice, trust anchors are given in the form of certificates, where their essential fields are the public key and the subject DN. In addition to the requirements in RFC 5280, OpenSSL checks the validity period of such certificates and makes use of some further fields. In particular, the subject key identifier extension, if present, is used for matching trust anchors during chain building.

In the most simple and common case, trust anchors are by default all self-signed ‘‘root’’ CA certificates that are placed in the *trust store*, which is a collection of certificates that are trusted for certain uses. This is akin to what is used in the trust stores of Mozilla Firefox, or Apple’s and Microsoft’s certificate stores, ...

From the OpenSSL perspective, a trust anchor is a certificate that should be augmented with an explicit designation for which uses of a target certificate the certificate may serve as a trust anchor. In PEM encoding, this is indicated by the `TRUSTED CERTIFICATE` string. Such a designation provides a set of positive trust attributes explicitly stating trust for the listed purposes and/or a set of negative trust attributes explicitly rejecting the use for the listed purposes. The purposes are encoded using the values defined for the extended key usages (EKUs) that may be given in X.509 extensions of end-entity certificates. See also the ‘‘Extended Key Usage’’ section below.

The currently recognized uses are **clientAuth** (SSL client use), **serverAuth** (SSL server use), **emailProtection** (S/MIME email use), **codeSigning** (object signer use), **OCSPSigning** (OCSP responder use), **OCSP** (OCSP request use), **timeStamping** (TSA server use), and **anyExtendedKeyUsage**. As of OpenSSL 1.1.0, the last of these blocks all uses when rejected or enables all uses when trusted.

A certificate, which may be CA certificate or an end-entity certificate, is considered a trust anchor for the given use if and only if all the following conditions hold:

- It is an an element of the trust store.
- It does not have a negative trust attribute rejecting the given use.
- It has a positive trust attribute accepting the given use or (by default) one of the following compatibilty conditions apply: It is self-signed or the **−partial_chain** option is given (which corresponds to the **X509_V_FLAG_PARTIAL_CHAIN** flag being set).

Certification Path Building

First, a certificate chain is built up starting from the target certificate and ending in a trust anchor.

The chain is built up iteratively, looking up in turn a certificate with suitable key usage that matches as an issuer of the current ''subject'' certificate as described below. If there is such a certificate, the first one found that is currently valid is taken, otherwise the one that expired most recently of all such certificates. For efficiency, no backtracking is performed, thus any further candidate issuer certificates that would match equally are ignored.

When a self-signed certificate has been added, chain construction stops. In this case it must fully match a trust anchor, otherwise chain building fails.

A candidate issuer certificate matches a subject certificate if all of the following conditions hold:

- Its subject name matches the issuer name of the subject certificate.
- If the subject certificate has an authority key identifier extension, each of its sub-fields equals the corresponding subject key identifier, serial number, and issuer field of the candidate issuer certificate, as far as the respective fields are present in both certificates.
- The certificate signature algorithm used to sign the subject certificate is supported and equals the public key algorithm of the candidate issuer certificate.

The lookup first searches for issuer certificates in the trust store. If it does not find a match there it consults the list of untrusted (''intermediate'' CA) certificates, if provided.

Certification Path Validation

When the certificate chain building process was successful the chain components and their links are checked thoroughly.

The first step is to check that each certificate is well-formed. Part of these checks are enabled only if the **−x509_strict** option is given.

The second step is to check the extensions of every untrusted certificate for consistency with the supplied purpose. If the**−pur pose** option is not given then no such checks are done except for SSL/TLS connection setup, where by default `sslserver` or `sslclient`, are checked. The target or ''leaf'' certificate, as well as any other untrusted certificates, must have extensions compatible with the specified purpose. All certificates except the target or ''leaf'' must also be valid CA certificates. The precise extensions required are described in more detail in ''CERTIFICATE EXTENSIONS'' in **openssl−x509** (1).

The third step is to check the trust settings on the last certificate (which typically is a self-signed root CA certificate). It must be trusted for the given use. For compatibility with previous versions of OpenSSL, a self-signed certificate with no trust attributes is considered to be valid for all uses.

The fourth, and final, step is to check the validity of the certificate chain. For each element in the chain, including the root CA certificate, the validity period as specified by the `notBefore` and `notAfter` fields is checked against the current system time. The **−attime** flag may be used to use a reference time other than ''now.'' The certificate signature is checked as well (except for the signature of the typically self-signed root CA certificate, which is verified only if the **−check_ss_sig** option is given). When verifying a certificate signature the keyUsage extension (if present) of the candidate issuer certificate is checked to permit digitalSignature for signing proxy certificates or to permit keyCertSign for signing other certificates, respectively. If all operations complete successfully then certificate is considered valid. If any operation fails then the certificate is not valid.

OPTIONS

Trusted Certificate Options

The following options specify how to supply the certificates that can be used as trust anchors for certain uses. As mentioned, a collection of such certificates is called a*trust stor e*.

Note that OpenSSL does not provide a default set of trust anchors. Many Linux distributions include a system default and configure OpenSSL to point to that. Mozilla maintains an influential trust store that can be found at <https://www.mozilla.org/en−US/about/governance/policies/security−group/certs/>.

The certificates to add to the trust store can be specified using following options.

−CAfile *file*
: Load the specified file which contains a certificate or several of them in case the input is in PEM or PKCS#12 format. PEM-encoded certificates may also have trust attributes set.

−no−CAfile
: Do not load the default file of trusted certificates.

−CApath *dir*
: Use the specified directory as a collection of trusted certificates, i.e., a trust store. Files should be named with the hash value of the X.509 SubjectName of each certificate. This is so that the library can extract the IssuerName, hash it, and directly lookup the file to get the issuer certificate. See **openssl−rehash** (1) for information on creating this type of directory.

−no−CApath
: Do not use the default directory of trusted certificates.

−CAstore *uri*
: Use *uri* as a store of CA certificates. The URI may indicate a single certificate, as well as a collection of them. With URIs in the `file:` scheme, this acts as **−CAfile** or **−CApath**, depending on if the URI indicates a single file or directory. See**ossl_stor e−file** (7) for more information on the `file:` scheme.

  These certificates are also used when building the server certificate chain (for example with **openssl−s_server** (1)) or client certificate chain (for example with **openssl−s_time** (1)).

−no−CAstore
: Do not use the default store of trusted CA certificates.

Verification Options

The certificate verification can be fine-tuned with the following flags.

−verbose
: Print extra information about the operations being performed.

−attime *timestamp*
: Perform validation checks using time specified by *timestamp* and not current system time. *timestamp* is the number of seconds since January 1, 1970 (i.e., the Unix Epoch).

−no_check_time
: This option suppresses checking the validity period of certificates and CRLs against the current time. If option **−attime** is used to specify a verification time, the check is not suppressed.

−x509_strict
: This disables non-compliant workarounds for broken certificates. Thus errors are thrown on certificates not compliant with RFC 5280.

  When this option is set, among others, the following certificate well-formedness conditions are checked:

  - The basicConstraints of CA certificates must be marked critical.
  - CA certificates must explicitly include the keyUsage extension.
  - If a pathlenConstraint is given the key usage keyCertSign must be allowed.
  - The pathlenConstraint must not be given for non-CA certificates.
  - The issuer name of any certificate must not be empty.
  - The subject name of CA certs, certs with keyUsage crlSign, and certs without subjectAlternativeName must not be empty.
  - If a subjectAlternativeName extension is given it must not be empty.

- The signatureAlgorithm field and the cert signature must be consistent.
- Any given authorityKeyIdentifier and any given subjectKeyIdentifier must not be marked critical.
- The authorityKeyIdentifier must be given for X.509v3 certs unless they are self-signed.
- The subjectKeyIdentifier must be given for all X.509v3 CA certs.

−ignore_critical
Normally if an unhandled critical extension is present that is not supported by OpenSSL the certificate is rejected (as required by RFC5280). If this option is set critical extensions are ignored.

−issuer_checks
Ignored.

−crl_check
Checks end entity certificate validity by attempting to look up a valid CRL. If a valid CRL cannot be found an error occurs.

−crl_check_all
Checks the validity of **all** certificates in the chain by attempting to look up valid CRLs.

−use_deltas
Enable support for delta CRLs.

−extended_crl
Enable extended CRL features such as indirect CRLs and alternate CRL signing keys.

−suiteB_128_only, **−suiteB_128**, **−suiteB_192**
Enable the Suite B mode operation at 128 bit Level of Security, 128 bit or 192 bit, or only 192 bit Level of Security respectively. See RFC6460 for details. In particular the supported signature algorithms are reduced to support only ECDSA and SHA256 or SHA384 and only the elliptic curves P−256 and P−384.

−auth_level *level*
Set the certificate chain authentication security level to *level*. The authentication security level determines the acceptable signature and public key strength when verifying certificate chains. For a certificate chain to validate, the public keys of all the certificates must meet the specified security *level*. The signature algorithm security level is enforced for all the certificates in the chain except for the chain's *trust anchor*, which is either directly trusted or validated by means other than its signature. See **SSL_CTX_set_security_level** (3) for the definitions of the available levels. The default security level is −1, or ''not set''. At security level 0 or lower all algorithms are acceptable. Security level 1 requires at least 80−bit−equivalent security and is broadly interoperable, though it will, for example, reject MD5 signatures or RSA keys shorter than 1024 bits.

−partial_chain
Allow verification to succeed if an incomplete chain can be built. That is, a chain ending in a certificate that normally would not be trusted (because it has no matching positive trust attributes and is not self-signed) but is an element of the trust store. This certificate may be self-issued or belong to an intermediate CA.

−check_ss_sig
Verify the signature of the last certificate in a chain if the certificate is supposedly self-signed. This is prohibited and will result in an error if it is a non-conforming CA certificate with key usage restrictions not including the keyCertSign bit. This verification is disabled by default because it doesn't add any security.

−allow_proxy_certs
Allow the verification of proxy certificates.

−trusted_first
As of OpenSSL 1.1.0 this option is on by default and cannot be disabled.

When constructing the certificate chain, the trusted certificates specified via **−CAfile**, **−CApath**,

−CAstore or **−trusted** are always used before any certificates specified via **−untrusted**.

−no_alt_chains
As of OpenSSL 1.1.0, since **−trusted_first** always on, this option has no effect.

−trusted *file*
Parse *file* as a set of one or more certificates. Each of them qualifies as trusted if has a suitable positive trust attribute or it is self-signed or the **−partial_chain** option is specified. This option implies the **−no−CAfile**, **−no−CApath**, and **−no−CAstore** options and it cannot be used with the **−CAfile**, **−CApath** or **−CAstore** options, so only certificates specified using the **−trusted** option are trust anchors. This option may be used multiple times.

−untrusted *file*
Parse *file* as a set of one or more certificates. All certificates (typically of intermediate CAs) are considered untrusted and may be used to construct a certificate chain from the target certificate to a trust anchor. This option may be used multiple times.

−policy *arg*
Enable policy processing and add *arg* to the user-initial-policy-set (see RFC5280). The policy *arg* can be an object name an OID in numeric form. This argument can appear more than once.

−explicit_policy
Set policy variable require-explicit-policy (see RFC5280).

−policy_check
Enables certificate policy processing.

−policy_print
Print out diagnostics related to policy processing.

−inhibit_any
Set policy variable inhibit-any-policy (see RFC5280).

−inhibit_map
Set policy variable inhibit-policy-mapping (see RFC5280).

−purpose *purpose*
The intended use for the certificate. Currently defined purposes are `sslclient`, `sslserver`, `nssslserver`, `smimesign`, `smimeencrypt`, `crlsign`, `ocsphelper`, `timestampsign`, and `any`. If peer certificate verification is enabled, by default the TLS implementation as well as the commands **s_client** and **s_server** check for consistency with TLS server or TLS client use, respectively.

While IETF RFC 5280 says that **id-kp-serverAuth** and **id-kp-clientAuth** are only for WWW use, in practice they are used for all kinds of TLS clients and servers, and this is what OpenSSL assumes as well.

−verify_depth *num*
Limit the certificate chain to *num* intermediate CA certificates. A maximal depth chain can have up to *num*+2 certificates, since neither the end-entity certificate nor the trust-anchor certificate count against the **−verify_depth** limit.

−verify_email *email*
Verify if *email* matches the email address in Subject Alternative Name or the email in the subject Distinguished Name.

−verify_hostname *hostname*
Verify if *hostname* matches DNS name in Subject Alternative Name or Common Name in the subject certificate.

−verify_ip *ip*
Verify if *ip* matches the IP address in Subject Alternative Name of the subject certificate.

−verify_name *name*
: Use default verification policies like trust model and required certificate policies identified by *name*. The trust model determines which auxiliary trust or reject OIDs are applicable to verifying the given certificate chain. They can be given using the **−addtrust** and **−addreject** options for **openssl−x509** (1). Supported policy names include: **default**, **pkcs7**, **smime_sign**, **ssl_client**, **ssl_server**. These mimics the combinations of purpose and trust settings used in SSL, CMS and S/MIME. As of OpenSSL 1.1.0, the trust model is inferred from the purpose when not specified, so the **−verify_name** options are functionally equivalent to the corresponding **−purpose** settings.

Extended Verification Options

Sometimes there may be more than one certificate chain leading to an end-entity certificate. This usually happens when a root or intermediate CA signs a certificate for another a CA in other organization. Another reason is when a CA might have intermediates that use two different signature formats, such as a SHA−1 and a SHA−256 digest.

The following options can be used to provide data that will allow the OpenSSL command to generate an alternative chain.

−xkey *infile*, **−xcert** *infile*, **−xchain**
: Specify an extra certificate, private key and certificate chain. These behave in the same manner as the **−cert**, **−key** and **−cert_chain** options. When specified, the callback returning the first valid chain will be in use by the client.

−xchain_build
: Specify whether the application should build the certificate chain to be provided to the server for the extra certificates via the **−xkey**, **−xcert**, and **−xchain** options.

−xcertform DER|**PEM**|**P12**
: The input format for the extra certificate. This option has no effect and is retained for backward compatibility only.

−xkeyform DER|**PEM**|**P12**
: The input format for the extra key. This option has no effect and is retained for backward compatibility only.

Certificate Extensions

Options like **−purpose** lead to checking the certificate extensions, which determine what the target certificate and intermediate CA certificates can be used for.

Basic Constraints

The basicConstraints extension CA flag is used to determine whether the certificate can be used as a CA. If the CA flag is true then it is a CA, if the CA flag is false then it is not a CA. **All** CAs should have the CA flag set to true.

If the basicConstraints extension is absent, which includes the case that it is an X.509v1 certificate, then the certificate is considered to be a ‘‘possible CA” and other extensions are checked according to the intended use of the certificate. The treatment of certificates without basicConstraints as a CA is presently supported, but this could change in the future.

Key Usage

If the keyUsage extension is present then additional restraints are made on the uses of the certificate. A CA certificate **must** have the keyCertSign bit set if the keyUsage extension is present.

Extended Key Usage

The extKeyUsage (EKU) extension places additional restrictions on the certificate uses. If this extension is present (whether critical or not) the key can only be used for the purposes specified.

A complete description of each check is given below. The comments about basicConstraints and keyUsage and X.509v1 certificates above apply to **all** CA certificates.

SSL Client
The extended key usage extension must be absent or include the ''web client authentication'' OID. The keyUsage extension must be absent or it must have the digitalSignature bit set. The Netscape certificate type must be absent or it must have the SSL client bit set.

SSL Client CA
The extended key usage extension must be absent or include the ''web client authentication'' OID. The Netscape certificate type must be absent or it must have the SSL CA bit set. This is used as a work around if the basicConstraints extension is absent.

SSL Server
The extended key usage extension must be absent or include the ''web server authentication'' and/or one of the SGC OIDs. The keyUsage extension must be absent or it must have the digitalSignature, the keyEncipherment set or both bits set. The Netscape certificate type must be absent or have the SSL server bit set.

SSL Server CA
The extended key usage extension must be absent or include the ''web server authentication'' and/or one of the SGC OIDs. The Netscape certificate type must be absent or the SSL CA bit must be set. This is used as a work around if the basicConstraints extension is absent.

Netscape SSL Server
For Netscape SSL clients to connect to an SSL server it must have the keyEncipherment bit set if the keyUsage extension is present. This isn't always valid because some cipher suites use the key for digital signing. Otherwise it is the same as a normal SSL server.

Common S/MIME Client Tests
The extended key usage extension must be absent or include the ''email protection'' OID. The Netscape certificate type must be absent or should have the S/MIME bit set. If the S/MIME bit is not set in the Netscape certificate type then the SSL client bit is tolerated as an alternative but a warning is shown. This is because some Verisign certificates don't set the S/MIME bit.

S/MIME Signing
In addition to the common S/MIME client tests the digitalSignature bit or the nonRepudiation bit must be set if the keyUsage extension is present.

S/MIME Encryption
In addition to the common S/MIME tests the keyEncipherment bit must be set if the keyUsage extension is present.

S/MIME CA
The extended key usage extension must be absent or include the ''email protection'' OID. The Netscape certificate type must be absent or must have the S/MIME CA bit set. This is used as a work around if the basicConstraints extension is absent.

CRL Signing
The keyUsage extension must be absent or it must have the CRL signing bit set.

CRL Signing CA
The normal CA tests apply. Except in this case the basicConstraints extension must be present.

BUGS

The issuer checks still suffer from limitations in the underlying X509_LOOKUP API. One consequence of this is that trusted certificates with matching subject name must appear in a file (as specified by the **−CAfile** option), a directory (as specified by **−CApath**), or a store (as specified by **−CAstore**). If there are multiple such matches, possibly in multiple locations, only the first one (in the mentioned order of locations) is recognised.

SEE ALSO

X509_verify_cert (3), **openssl−verify** (1), **openssl−ocsp** (1), **openssl−ts** (1), **openssl−s_client** (1), **openssl−s_server** (1), **openssl−smime** (1), **openssl−cmp** (1), **openssl−cms** (1)

HISTORY

The checks enabled by **−x509_strict** have been extended in OpenSSL 3.0.

COPYRIGHT