Red Hat Enterprise Linux Release 9.2 Manual Pages on 'openssl.cnf.5' command

$ man openssl.cnf.5

CONFIG(5ossl) OpenSSL CONFIG(5ossl)

NAME

config - OpenSSL CONF library configuration files

DESCRIPTION

This page documents the syntax of OpenSSL configuration files, as parsed by NCONF_load(3) and related functions. This format is used by many of the OpenSSL commands, and to initialize the libraries when used by any application.

The first part describes the general syntax of the configuration files, and subsequent sections describe the semantics of individual modules. Other modules are described in fips_config(5) and x509v3_config(5). The syntax for defining ASN.1 values is described in ASN1_generate_nconf(3).

SYNTAX

A configuration file is a series of lines. Blank lines, and whitespace between the elements of a line, have no significance. A comment starts with a # character; the rest of the line is ignored. If the # is the first non-space character in a line, the entire line is ignored.

Directives

Two directives can be used to control the parsing of configuration files: .include and .pragma.

For compatibility with older versions of OpenSSL, an equal sign after the directive will be ignored. Older versions will treat it as an assignment, so care should be taken if the difference in semantics is

important.

A file can include other files using the include syntax:

```
  .include [=] pathname
```

If pathname is a simple filename, that file is included directly at

that point. Included files can have .include statements that specify

other files. If pathname is a directory, all files within that

directory that have a ".cnf" or ".conf" extension will be included.

(This is only available on systems with POSIX IO support.) Any sub-

directories found inside the pathname are ignored. Similarly, if a

file is opened while scanning a directory, and that file has an

.include directive that specifies a directory, that is also ignored.

As a general rule, the pathname should be an absolute path; this can be

enforced with the abspath and includedir pragmas, described below. The

environment variable OPENSSL_CONF_INCLUDE, if it exists, is prepended

to all relative pathnames. If the pathname is still relative, it is

interpreted based on the current working directory.

To require all file inclusions to name absolute paths, use the

following directive:

```
 .pragma [=] abspath:value
```

The default behavior, where the value is false or off, is to allow

relative paths. To require all .include pathnames to be absolute paths,

use a value of true or on.

In these files, the dollar sign, $, is used to reference a variable, as

described below. On some platforms, however, it is common to treat $

as a regular character in symbol names. Supporting this behavior can

be done with the following directive:

```
 .pragma [=] dollarid:value
```

The default behavior, where the value is false or off, is to treat the

dollarsign as indicating a variable name; "foo$bar" is interpreted as

"foo" followed by the expansion of the variable "bar". If value is true

or on, then "foo$bar" is a single seven-character name nad variable

expansions must be specified using braces or parentheses.

```
 .pragma [=] includedir:value
```

If a relative pathname is specified in the .include directive, and the OPENSSL_CONF_INCLUDE environment variable doesn't exist, then the value of the includedir pragma, if it exists, is prepended to the pathname.

Settings

A configuration file is divided into a number of sections. A section begins with the section name in square brackets, and ends when a new section starts, or at the end of the file. The section name can consist of alphanumeric characters and underscores. Whitespace between the name and the brackets is removed.

The first section of a configuration file is special and is referred to as the default section. This section is usually unnamed and spans from the start of file until the first named section. When a name is being looked up, it is first looked up in the current or named section, and then the default section if necessary.

The environment is mapped onto a section called ENV.

Within a section are a series of name/value assignments, described in more detail below. As a reminder, the square brackets shown in this example are required, not optional:

```
 [ section ]
 name1 = This is value1
 name2 = Another value
 ...
 [ newsection ]
 name1 = New value1
 name3 = Value 3
```

The name can contain any alphanumeric characters as well as a few punctuation symbols such as . , ; and _. Whitespace after the name and before the equal sign is ignored.

If a name is repeated in the same section, then all but the last value are ignored. In certain circumstances, such as with Certificate DNs, the same field may occur multiple times. In order to support this, commands like openssl-req(1) ignore any leading text that is preceded with a period. For example:

```
 1.OU = First OU
 2.OU = Second OU
```

The value consists of the string following the = character until end of line with any leading and trailing whitespace removed.

The value string undergoes variable expansion. The text $var or "${var}" inserts the value of the named variable from the current section. To use a value from another section use $section::name or "${section::name}". By using $ENV::name, the value of the specified environment variable will be substituted.

Variables must be defined before their value is referenced, otherwise an error is flagged and the file will not load. This can be worked around by specifying a default value in the default section before the variable is used.

Any name/value settings in an ENV section are available to the configuration file, but are not propagated to the environment.

It is an error if the value ends up longer than 64k.

It is possible to escape certain characters by using a single ' or double " quote around the value, or using a backslash \ before the character, By making the last character of a line a \ a value string can be spread across multiple lines. In addition the sequences \n, \r, \b and \t are recognized.

The expansion and escape rules as described above that apply to value also apply to the pathname of the .include directive.

OPENSSL LIBRARY CONFIGURATION

The sections below use the informal term module to refer to a part of the OpenSSL functionality. This is not the same as the formal term FIPS module, for example.

The OpenSSL configuration looks up the value of openssl_conf in the default section and takes that as the name of a section that specifies how to configure any modules in the library. It is not an error to leave any module in its default configuration. An application can specify a different name by calling CONF_modules_load_file(), for example, directly.

OpenSSL also looks up the value of config_diagnostics. If this exists and has a nonzero numeric value, any error suppressing flags passed to CONF_modules_load() will be ignored. This is useful for diagnosing misconfigurations but its use in production requires additional consideration. With this option enabled, a configuration error will completely prevent access to a service. Without this option and in the presence of a configuration error, access will be allowed but the desired configuration will not be used.

```
 # These must be in the default section
 config_diagnostics = 1
 openssl_conf = openssl_init
 [openssl_init]
 oid_section = oids
 providers = providers
 alg_section = evp_properties
 ssl_conf = ssl_configuration
 engines = engines
 random = random
 [oids]
 ... new oids here ...
 [providers]
 ... provider stuff here ...
 [evp_properties]
 ... EVP properties here ...
 [ssl_configuration]
 ... SSL/TLS configuration properties here ...
 [engines]
 ... engine properties here ...
 [random]
 ... random properties here ...
```

The semantics of each module are described below. The phrase "in the initialization section" refers to the section identified by the openssl_conf or other name (given as openssl_init in the example above). The examples below assume the configuration above is used to specify the individual sections.

ASN.1 Object Identifier Configuration

The name oid_section in the initialization section names the section containing name/value pairs of OID's. The name is the short name; the value is an optional long name followed by a comma, and the numeric value. While some OpenSSL commands have their own section for specifying OID's, this section makes them available to all commands and applications.

```
 [oids]
 shortName = a very long OID name, 1.2.3.4
 newoid1 = 1.2.3.4.1
 some_other_oid = 1.2.3.5
```

If a full configuration with the above fragment is in the file example.cnf, then the following command line:

```
 OPENSSL_CONF=example.cnf openssl asn1parse -genstr OID:1.2.3.4.1
```

will output:

```
 0:d=0  hl=2 l=   4 prim: OBJECT            :newoid1
```

showing that the OID "newoid1" has been added as "1.2.3.4.1".

Provider Configuration

The name providers in the initialization section names the section containing cryptographic provider configuration. The name/value assignments in this section each name a provider, and point to the configuration section for that provider. The provider-specific section is used to specify how to load the module, activate it, and set other parameters.

Within a provider section, the following names have meaning:

identity

This is used to specify an alternate name, overriding the default name specified in the list of providers. For example:

```
 [providers]
 foo = foo_provider
```

```
         [foo_provider]

         identity = my_fips_module
```

module

Specifies the pathname of the module (typically a shared library) to load.

activate

If present, the module is activated. The value assigned to this name is not significant.

All parameters in the section as well as sub-sections are made available to the provider.

Loading the legacy provider

Uncomment the sections that start with ## in openssl.cnf to enable the legacy provider. Note: In general it is not recommended to use the above mentioned algorithms for security critical operations, as they are cryptographically weak or vulnerable to side-channel attacks and as such have been deprecated.

Default provider and its activation

If no providers are activated explicitly, the default one is activated implicitly. See OSSL_PROVIDER-default(7) for more details.

If you add a section explicitly activating any other provider(s), you most probably need to explicitly activate the default provider, otherwise it becomes unavailable in openssl. It may make the system remotely unavailable.

EVP Configuration

The name alg_section in the initialization section names the section containing algorithmic properties when using the EVP API.

Within the algorithm properties section, the following names have meaning:

default_properties

The value may be anything that is acceptable as a property query string for EVP_set_default_properties().

rh-allow-sha1-signatures

The value is a boolean that can be yes or no. If the value is not set, it behaves as if it was set to no.

When set to no, any attempt to create or verify a signature with a SHA1 digest will fail. For compatibility with older versions of OpenSSL, set this option to yes. This setting also affects TLS, where signature algorithms that use SHA1 as digest will no longer be supported if this option is set to no. Note that enabling rh-allow-sha1-signatures will allow TLS signature algorithms that use SHA1 in security level 2, despite the definition of security level 2 of 112 bits of security, which SHA1 does not meet. Because TLS 1.1 or lower use MD5-SHA1 as pseudorandom function (PRF) to derive key material, disabling rh-allow-sha1-signatures requires the use of TLS 1.2 or newer.

fips_mode (deprecated)

The value is a boolean that can be yes or no. If the value is yes, this is exactly equivalent to:

```
 default_properties = fips=yes
```

If the value is no, nothing happens. Using this name is deprecated, and if used, it must be the only name in the section.

SSL Configuration

The name ssl_conf in the initialization section names the section containing the list of SSL/TLS configurations. As with the providers, each name in this section identifies a section with the configuration for that name. For example:

```
 [ssl_configuration]
 server = server_tls_config
 client = client_tls_config
 system_default = tls_system_default
 [server_tls_config]
 ... configuration for SSL/TLS servers ...
 [client_tls_config]
 ... configuration for SSL/TLS clients ...
```

The configuration name system_default has a special meaning. If it exists, it is applied whenever an SSL_CTX object is created. For example, to impose system-wide minimum TLS and DTLS protocol versions:

```
 [tls_system_default]
 MinProtocol = TLSv1.2
 MinProtocol = DTLSv1.2
```

The minimum TLS protocol is applied to SSL_CTX objects that are TLS-based, and the minimum DTLS protocol to those are DTLS-based. The same applies also to maximum versions set with MaxProtocol.

Each configuration section consists of name/value pairs that are parsed by SSL_CONF_cmd(3), which will be called by SSL_CTX_config() or SSL_config(), appropriately. Note that any characters before an initial dot in the configuration section are ignored, so that the same command can be used multiple times. This probably is most useful for loading different key types, as shown here:

```
 [server_tls_config]
 RSA.Certificate = server-rsa.pem
 ECDSA.Certificate = server-ecdsa.pem
```

Engine Configuration

The name engines in the initialization section names the section containing the list of ENGINE configurations. As with the providers, each name in this section identifies an engine with the configuration for that engine. The engine-specific section is used to specify how to load the engine, activate it, and set other parameters.

Within an engine section, the following names have meaning:

engine_id

This is used to specify an alternate name, overriding the default name specified in the list of engines. If present, it must be first. For example:

```
 [engines]
 foo = foo_engine
 [foo_engine]
 engine_id = myfoo
```

dynamic_path

This loads and adds an ENGINE from the given path. It is equivalent to sending the ctrls SO_PATH with the path argument followed by LIST_ADD with value 2 and LOAD to the dynamic ENGINE. If this is not the required behaviour then alternative ctrls can be sent directly to the dynamic ENGINE using ctrl commands.

init

This specifies whether to initialize the ENGINE. If the value is 0 the ENGINE will not be initialized, if the value is 1 an attempt is made to initialize the ENGINE immediately. If the init command is not present then an attempt will be made to initialize the ENGINE after all commands in its section have been processed.

default_algorithms

This sets the default algorithms an ENGINE will supply using the function ENGINE_set_default_string().

All other names are taken to be the name of a ctrl command that is sent to the ENGINE, and the value is the argument passed with the command. The special value EMPTY means no value is sent with the command. For example:

```
 [engines]
 foo = foo_engine
 [foo_engine]
 dynamic_path = /some/path/fooengine.so
 some_ctrl = some_value
 default_algorithms = ALL
 other_ctrl = EMPTY
```

Random Configuration

The name random in the initialization section names the section containing the random number generater settings.

Within the random section, the following names have meaning:

random

This is used to specify the random bit generator. For example:

```
 [random]
 random = CTR-DRBG
```

The available random bit generators are:

CTR-DRBG

HASH-DRBG

HMAC-DRBG

cipher

This specifies what cipher a CTR-DRBG random bit generator will use. Other random bit generators ignore this name. The default value is AES-256-CTR.

digest

This specifies what digest the HASH-DRBG or HMAC-DRBG random bit generators will use. Other random bit generators ignore this name.

properties

This sets the property query used when fetching the random bit generator and any underlying algorithms.

seed

This sets the randomness source that should be used. By default SEED-SRC will be used outside of the FIPS provider. The FIPS provider uses call backs to access the same randomness sources from outside the validated boundary.

seed_properties

This sets the property query used when fetching the randomness source.

EXAMPLES

This example shows how to use quoting and escaping.

```
 # This is the default section.
 HOME = /temp
 configdir = $ENV::HOME/config
 [ section_one ]
 # Quotes permit leading and trailing whitespace
 any = " any variable name "
 other = A string that can \
 cover several lines \
 by including \\ characters
 message = Hello World\n
 [ section_two ]
 greeting = $section_one::message
```

This example shows how to expand environment variables safely. In this example, the variable tempfile is intended to refer to a temporary file, and the environment variable TEMP or TMP, if present, specify the directory where the file should be put. Since the default section is checked if a variable does not exist, it is possible to set TMP to default to /tmp, and TEMP to default to TMP.

```
 # These two lines must be in the default section.
 TMP = /tmp
 TEMP = $ENV::TMP
 # This can be used anywhere
 tmpfile = ${ENV::TEMP}/tmp.filename
```

This example shows how to enforce FIPS mode for the application sample.

```
 sample = fips_config
 [fips_config]
 alg_section = evp_properties
 [evp_properties]
 default_properties = "fips=yes"
```

ENVIRONMENT

OPENSSL_CONF

The path to the config file, or the empty string for none. Ignored in set-user-ID and set-group-ID programs.

OPENSSL_ENGINES

The path to the engines directory. Ignored in set-user-ID and set-group-ID programs.

OPENSSL_MODULES

The path to the directory with OpenSSL modules, such as providers. Ignored in set-user-ID and set-group-ID programs.

OPENSSL_CONF_INCLUDE

The optional path to prepend to all .include paths.

BUGS

There is no way to include characters using the octal \nnn form.

Strings are all null terminated so nulls cannot form part of the value.

The escaping isn't quite right: if you want to use sequences like \n

you can't use any quote escaping on the same line.

The limit that only one directory can be opened and read at a time can

be considered a bug and should be fixed.

HISTORY

An undocumented API, NCONF_WIN32(), used a slightly different set of

parsing rules there were intended to be tailored to the Microsoft

Windows platform. Specifically, the backslash character was not an

escape character and could be used in pathnames, only the double-quote

character was recognized, and comments began with a semi-colon. This

function was deprecated in OpenSSL 3.0; applications with configuration

files using that syntax will have to be modified.

SEE ALSO

openssl-x509(1), openssl-req(1), openssl-ca(1), ASN1_generate_nconf(3),

EVP_set_default_properties(3), CONF_modules_load(3),

CONF_modules_load_file(3), fips_config(5), and x509v3_config(5).

COPYRIGHT