OpenSSL

Warning: Collaborated research into OpenSSL protocol usage, published in May 2015, showed further significant risks for SSL connections; named "Logjam" attack. See **https://weakdh.org/** for results and **https://weakdh.org/sysadmin.html** for suggested server-side configuration changes.

OpenSSL (http://www.openssl.org) is an open-source implementation of the SSL and TLS protocols, dual-licensed under the OpenSSL (Apache License 1.0) and the SSLeay (4-clause BSD) licenses. It is supported on a variety of platforms, including BSD, Linux, OpenVMS, Solaris and Windows. It is designed to be as flexible as possible, and is free to use for both personal and commercial uses. It is based on the earlier SSLeay library. Version 1.0.0 of OpenSSL was released on March 29, 2010.

Contents

SSL introduction

In order to focus on setting up a SSL/TLS solution, rather than explaining the bare basics regarding the subject, the approach used throughout the article to explain SSL concepts is by and large file-oriented.

Consult both **Wikipedia:Certificate authority** and **Wikipedia:Public key infrastructure** for more information.

Certificate authority (CA)

Certificate authorities return certificates from end-user requests. In order to do this, the returned end-user certificate is signed with the CA private key and CA certificate, which in turn contains the CA public key. CA also distribute certificate revocation lists (CRL) which tell the end-user what certificates are no longer valid, and when the next CRL is due.

CA private key

The CA private key is the crucial part of the trifecta. Exposing it would defeat the purpose of designating a central authority that validates and revokes permissions, and at the same time, it is the signed counter part to the CA public key used to certify against the CA certificate. An exposed CA private key could allow an attacker to replicate the CA certificate since the CA private key signature is embedded in the CA certificate itself.

CA certificate and public key

These are distributed in a single file to all end-users. They are used to certify other end-user certificates that claimed to be signed by the matching CA, such as mail servers or websites.

End-users

End-users submit certificate requests to the CA which contain a distinguished name (DN). Normally, CA do not allow more than one valid certificate with the same DN without revoking the previous one. End-user certificates may be revoked if they are not renewed when due, among other reasons.

End-user generated key

End-users generate keys in order to sign certificate requests that are submitted to the CA. As with the CA private key, an exposed user-key could facilitate impersonating the user to the point where an attacker could submit a request under the user's name, resulting in the CA revoking the former, legitimate, user certificate.

Certificate requests

These contain the user's DN and public key. As their name implies, they fully represent the initial part of the process of acquiring certification from a CA.

End-user certificate

The main distinction between an end-user certificate and CA certificate is that end-user ones cannot sign certificates themselves; they merely provide means of identification in exchanges of information.

Certificate revocation list (CRL)

CRLs are also signed with the CA key, but they only dictate information regarding end-user certificates. Usually, a 30 day span is given between new CRL submissions.

Configuring

The OpenSSL configuration file, conventionally placed in `/etc/ssl/openssl.cnf`, may appear complicated at first. This is not remedied by the fact that there is no *include* directive to split configuration into a modal setup. Nevertheless, this section covers the essential settings.

Remember that variables may be expanded in assignments, much like how shell scripts work. For a more thorough explanation of the configuration file format, see **config(5ssl) (https://jlk.fjfi.cvut.cz/arch/manpages/man/config.5ssl)**. In some operating systems, this **man page** is named `config(5)` or `openssl-config(5)`. Sometimes, it may not even be available through the man hierarchy at all, for example, it may be placed in the following location `/usr/share/openssl`.

Global variables

These settings are relevant in all sections. For that to happen, they can not be specified under a section header:

```
DIR=            .                       # Useful macro for populating real vars.
RANDFILE=       ${DIR}/private/.rnd     # Entropy source.
default_md=     sha1                    # Default message digest.
```

ca section

These settings are used when signing CRLs, and signing and revoking certificates. Users that only want to generate requests can safely skip to the **#req section**.

```
[ ca ]
default_ca=     dft_ca  # Configuration files may have more than one CA
                        # section for different scenarios.

[ dft_ca ]
certificate=    ${DIR}/cacert.pem       # The CA certificate.
database=       ${DIR}/index.txt        # Keeps tracks of valid/revoked certs.
new_certs_dir=  ${DIR}/newcerts         # Copies of signed certificates, for
                                        # administrative purposes.
private_key=    ${DIR}/private/cakey.pem # The CA key.
serial=         ${DIR}/serial           # Should be populated with the next
                                        # cert hex serial.

# These govern the way certificates are displayed while confirming
# the signing process.
name_opt=       ca_default
cert_opt=       ca_default

default_days=   365     # How long to sign certificates for.
default_crl_days=30     # The same, but for CRL.

policy=         dft_policy      # The default policy should be lenient.
```

```
x509_extensions=cert_v3         # For v3 certificates.

[ dft_policy ]
# A value of 'supplied' means the field must be present in the certificate,
# whereas 'match' means the field must be populated with the same contents as
# the CA certificate. 'optional' dictates that the field is entirely optional.

C=      supplied        # Country
ST=     supplied        # State or province
L=      optional        # Locality
O=      supplied        # Organization
OU=     optional        # Organizational unit
CN=     supplied        # Common name

[ cert_v3 ]
# With the exception of 'CA:FALSE', there are PKIX recommendations for end-user
# certificates that should not be able to sign other certificates.
# 'CA:FALSE' is explicitely set because some software will malfunction without.

subjectKeyIdentifier=   hash
basicConstraints=       CA:FALSE
keyUsage=               nonRepudiation, digitalSignature, keyEncipherment

nsCertType=             client, email
nsComment=              "OpenSSL Generated Certificate"

authorityKeyIdentifier=keyid:always,issuer:always
```

req section

Settings related to generating keys, requests and self-signed certificates.

The req section is responsible for the DN prompts. A general misconception is the *Common Name* (CN) prompt, which suggests that it should have the user's proper name as a value. End-user certificates need to have the **machine hostname** as CN, whereas CA should *not* have a valid TLD, so that there is no chance that, between the possible combinations of

certified end-users' CN and the CA certificate's, there is a match that could be misinterpreted by some software as meaning that the end-user certificate is self-signed. Some CA certificates do not even have a CN, such as **Equifax (http://www.equifax.com)**:

```
$ openssl x509 -subject -noout < /etc/ssl/certs/Equifax_Secure_CA.pem
```

```
subject= /C=US/O=Equifax/OU=Equifax Secure Certificate Authority
```

Even though splitting the files is not strictly necessary to normal functioning, it is very confusing to handle request generation and CA administration from the same configuration file, so it is advised to follow the convention of clearly separating the settings into two `cnf` files and into two containing directories.

Here are the settings that are common to both tasks:

```
[ req ]
# Default bit encryption and out file for generated keys.
default_bits=   2048
default_keyfile=private/cakey.pem

string_mask=    utf8only        # Only allow utf8 strings in request/ca fields.
prompt=         no              # Do not prompt for field value confirmation.
```

CA req settings

The settings should produce a standard CA capable of only singing other certificates:

```
distinguished_name=ca_dn        # Distinguished name contents.
x509_extensions=ca_v3           # For generating ca certificates.

[ ca_dn ]
# CN is not needed for CA certificates
C=      US
ST=     New Jersey
O=      localdomain

[ ca_v3 ]
subjectKeyIdentifier=   hash

# PKIX says this should also contain the 'crucial' value, yet some programs
# have trouble handling it.
basicConstraints=       CA:TRUE

keyUsage=       cRLSign, keyCertSign

nsCertType=     sslCA
nsComment=      "OpenSSL Generated CA Certificate"

authorityKeyIdentifier=keyid:always,issuer:always
```

End-user req settings

Makes a v3 request suitable for most circumstances:

```
distinguished_name=ca_dn        # Distinguished name contents.
req_extensions=req_v3           # For generating ca certificates.

[ ca_dn ]
C=      US
ST=     New Jersey
O=      localdomain
CN=     localhost

[ req_v3 ]
basicConstraints=       CA:FALSE
keyUsage=               nonRepudiation, digitalSignature, keyEncipherment
```

GOST engine support

First, be sure that libgost.so exist on your system

```
$ pacman -Ql openssl | grep libgost
```

In case everything is fine, add the following lines to the config:

```
openssl_conf = openssl_def # this must be a top-level declaration
```

Put the following lines in the end of the document:

```
[ openssl_def ]
engines = engine_section

[ engine_section ]
gost = gost_section

[ gost_section ]
engine_id = gost
soft_load = 1
dynamic_path = /usr/lib/engines/libgost.so
default_algorithms = ALL
CRYPT_PARAMS = id-Gost28147-89-CryptoPro-A-ParamSet
```

The official **README.gost (http://ftp.netbsd.org/pub/NetBSD/NetBSD-current/src/crypto/external/bsd/openssl/dist/engines/ccgost/README.gost)** should contain more examples on this.

Generating keys

Before generating the key, make a secure directory to host it:

```
$ mkdir -m0700 private
```

Followed by preemptively assigning secure permissions for the key itself:

```
$ touch private/key.pem
$ chmod 0600 private/key.pem
```

Alternatively set **umask** to restrict permissions of newly created files and directories:

```
$ umask 077
```

An example `genpkey` key generation:

```
$ openssl genpkey -algorithm RSA -out private/key.pem -pkeyopt rsa_keygen_bits:4096
```

If an encrypted key is desired, use the following command. Password will be prompted for:

```
$ openssl genpkey -aes-256-cbc -algorithm RSA -out private/key.pem -pkeyopt rsa_keygen_bits:4096
```

Making requests

To obtain a certificate from a CA, whether a public one such as **CAcert.org (http://www.cacert.org)** or a locally managed solution, a request file must be submitted which is known as a **Certificate Signing Request** or CSR.

Make a new request and sign it with a previously **generated key**:

```
$ openssl req -new -sha256 -key private/key.pem -out req.csr
```

Signing certificates

Covers the process of local CA signing: directly self-signed certificates or through a local CA.

Self-signed certificate

A significant amount of programs will not work with self-signed certificates, and maintaining more than one system with self-signed certificates is more trouble than investing the initial effort in setting up a **certificate authority**.

If a key was already generated as **explained before**, use this command to sign the new certificate with the aforementioned key:

```
$ openssl req -key private/key.pem -x509 -new -days 3650 -out cacert.pem
```

Certificate authority

OpenSSL Certificate Authority (https://jamielinux.com/docs/openssl-certificate-authority/) is a detailed guide on using OpenSSL to act as a CA.

The method shown in this section is mostly meant to show how signing works; it is not suited for large deployments that need to automate signing a large number of certificates. Consider installing an SSL server for that purpose.

Before using the Makefile, make a configuration file according to **#Configuring**. Be sure to follow instructions relevant to CA administration; not request generation.

Makefile

Saving the file as `Makefile` and issuing `make` in the containing directory will generate the initial CRL along with its prerequisites:

```
OPENSSL=        openssl
CNF=            openssl.cnf
CA=             ${OPENSSL} ca -config ${CNF}
```

```
REQ=            ${OPENSSL} req -config ${CNF}

KEY=            private/cakey.pem
KEYMODE=        RSA

CACERT=         cacert.pem
CADAYS=         3650

CRL=            crl.pem
INDEX=          index.txt
SERIAL=         serial

CADEPS=         ${CNF} ${KEY} ${CACERT}

all:    ${CRL}

${CRL}: ${CADEPS}
        ${CA} -gencrl -out ${CRL}

${CACERT}: ${CNF} ${KEY}
        ${REQ} -key ${KEY} -x509 -new -days ${CADAYS} -out ${CACERT}
        rm -f ${INDEX}
        touch ${INDEX}
        echo 100001 > ${SERIAL}

${KEY}: ${CNF}
        mkdir -m0700 -p $(dir ${KEY})
        touch ${KEY}
        chmod 0600 ${KEY}
        ${OPENSSL} genpkey -algorithm ${KEYMODE} -out ${KEY}

revoke: ${CADEPS} ${item}
        @test -n $${item:?'usage: ${MAKE} revoke item=cert.pem'}
        ${CA} -revoke ${item}
        ${MAKE} ${CRL}

sign:   ${CADEPS} ${item}
        @test -n $${item:?'usage: ${MAKE} sign item=request.csr'}
        mkdir -p newcerts
        ${CA} -in ${item} -out ${item:.csr=.crt}
```

To sign certificates:

```
$ make sign item=req.csr
```

To revoke certificates:

```
$ make revoke item=cert.pem
```

Troubleshooting

"bad decrypt" while decrypting

OpenSSL 1.1.0 changed the default digest algorithm for the dgst and enc commands from MD5 to SHA256. **[1] (https://www.openssl.org/news/changelog.html#x6)**

Therefore if a file has been encrypted using OpenSSL 1.0.2 or older, trying to decrypt it with an up to date version may result in an error like:

```
error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:crypto/evp/evp_enc.c:540
```

Supplying the `-md md5` option should solve the issue:

```
$ openssl enc -d -md md5 -in encrypted -out decrypted
```

See also

- **Wikipedia page** on OpenSSL, with background information.
- **OpenSSL (http://www.openssl.org)** project page.
- **FreeBSD Handbook (http://www.freebsd.org/doc/en/books/handbook/openssl.html)**
- **Step-by-step guide to create a signed SSL certificate (http://www.akadia.com/services/ssh_test_certificate.html)**
- **OpenSSL Certificate Authority (https://jamielinux.com/docs/openssl-certificate-authority/)**
- **Bulletproof SSL and TLS (https://www.feistyduck.com/books/bulletproof-ssl-and-tls/bulletproof-ssl-and-tls-introduction.pdf)** by Ivan Ristić, a more formal introduction to SSL/TLS