cubeSQL

SSL/TLS support

A guide by Tino Gruse (Roth Soft AG)

Introduction

Starting from version 4.3 cubeSQL fully support the SSL protocol. SSL is dynamically loaded if OpenSSL is installed on your system. OpenSSL is freely available from: http://www.openssl.org.

SSL support in cubeSQL version 5 has been greatly improved with changes both on server side and client side. This document will briefly summarize all of them. When not differently specified, validations will use default OpenSSL algorithms. Validations means protocol validation, certificates validation and peer verifications. SSL is not safe per default; it's all about the right configuration.

This article leads you through the whole process to get a secure SSL/TLS connection with CubeSQL on Debian/Ubuntu Linux respecting the current Best Practices of SSL Labs https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf

Server Side

- By default cubeSQL uses the strongest available protocol with the following order: TLSv1.2, TSLv1.1, TSLv1 and SSLv3 (or the strongest one available in the OpenSSL library).

- If a root certificate is provided then peer verification is performed.

- User can decide which protocol to disable. The following configurations can be used:
  Disable SSLv3
  Disable SSLv3 and TSLv1
  Disable SSLv3, TSLv1 and TSLv1.1
  Disable SSLv3, TSLv1 and TSLv1.1 and TSLv1.2 (to support just a future TSLv1.3 protocol)

- User can specify DH key bits and that key will be generated on the fly by the server.

- User can specify a list of allowed SSL ciphers to accept. Un-allowed ciphers will be refused before a connection will be established.

- By default SSLv2 is disabled.

- By default when performing renegotiation as a server, always start a new session (i.e., session resumption requests are only accepted in the initial handshake). SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION flag.

- By default always create a new key when using temporary/ephemeral DH parameters. This option must be used to prevent small subgroup attacks, when the DH parameters were not generated using ``strong'' primes. SSL_OP_SINGLE_DH_USE flag.

- By default apply all available patches and workaround to OpenSSL bugs. SSL_OP_ALL flag.

Client Side

- Added support for loading SSL shared libraries in order to have the possibility to always stay up to date with the latest OpenSSL versions.

- By default client will use the strongest available protocol with the following order: TLSv1.2, TSLv1.1, TSLv1 and SSLv3.

- If a root certificate is provided then peer verification is used.

- User can specify a list of SSL ciphers to use.

Certificate Creation with OpenSSL

Please create the productive certificates and especially the private keys only on a machine that is not connected to the internet.

Install or Update OpenSSL

Install OpenSSL if it is not already.

```
$ sudo apt-get update
$ sudo apt-get install openssl
```

Be sure to always update to the newest stable version available.

```
$ sudo apt-get upgrade
$ openssl version -a
```

```
OpenSSL 1.0.1 14 Mar 2012
built on: Fri May 29 00:53:01 UTC 2015
```

Configure OpenSSL

Edit the default OpenSSL configuration to fit your needs.

```
$ sudo nano /etc/ssl/openssl.cnf
```

```
[ CA_default ]
dir                                 =       ./myCompanyCA
default_md                          =       sha256
[ req_distinguished_name ]
stateOrProvinceName_default         =       Zuerich
localityName_default                =       Zuerich
0.organizationName_default          =       My Company
organizationalUnitName_default  =       My Company Unit

[ srv_cert ]
nsCertType                          =       server
basicConstraints                    =       CA:FALSE
authorityKeyIdentifier              =       keyid,issuer
subjectKeyIdentifier                =       hash
subjectAltName                      =       “DNS:www.mycompany.com”
extendedKeyUsage                    =       serverAuth
keyUsage                            =       nonRepudiation, digitalSignature,
keyEncipherment, dataEncipherment, keyAgreement

[ v3_ca ]
nsCertType                          =       sslCA
basicConstraints                    =       CA:true,pathlen:1
authorityKeyIdentifier              =       keyid:always,issuer:always
subjectKeyIdentifier                =       hash
extendedKeyUsage                    =       serverAuth, clientAuth, codeSigning,
OCSPSigning
keyUsage                            =       cRLSign, keyCertSign, digitalSignature,
nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement
```

Create Root CA

Before creating the root CA you need to set up your certification environment.

```
$ mkdir -p ./certs/myCompanyCA/newcerts
$ cd ./certs && touch myCompanyCA/index.txt
```

Now generate an AES-encrypted 4096 bit private key (rootca.key) for the root CA.

```
$ openssl genrsa -aes256 -out rootca.key 4096
```

With this private key you may create a certificate signing request (CSR).

```
$ openssl req -new -sha256 -key rootca.key -out rootca.req
```

You will be asked for some information about your root CA. So give your root CA a meaningful name like "My Company CA" in "Common Name" and set a valid E-Mail.
Next you need to generate a serial number for your root CA. I prefer a random 16 bytes hexadecimal number.

```
$ openssl rand 16 -hex > ./myCompanyCA/serial
```

Since this certificate should be a root CA you need to self-sign your CSR with your own private key. You may want to adjust the validation time of 3650 days from today on. Please note that all certificates in the certificate chain have to be valid.

```
$ openssl ca -out rootca.crt -days 3650 -md sha256 -selfsign -extensions v3_ca -keyfile
rootca.key -infiles rootca.req
```

Now you have your root CA certificate in rootca.crt.
Please save the private key of the rootca only in a very secure location and do not copy it to other locations.

Create Server Certificate

With the root CA you are able to generate a valid server certificate for your environment. First you need to generate another private key for the server certificate. For the server 2048 bit are a good compromise of security and performance nowadays.

```
$ openssl genrsa -aes256 -out rootca.key 2048
```

Next you have to create the CSR with this private key.

```
$ openssl req -new –sha256 -key server.key -out server.req
```

When you will be asked for the "Common Name" you need to type in the correct hostname of your CubeSQL server. Please note that you are only able to type in one hostname. The CubeSQL client does actually not support alternative names in the certificate. So if you need to support multiple subdomains you have to use wildcards (like *.mycompany.com).

Next you can generate another random serial number to identify the server certificate.

```
$ openssl rand 16 -hex > ./myCompanyCA/serial
```

Now you may sign this certification request with your root CA. In this example a validation time of two years (~730 days) is used. Ideally this interval fits in the validation time of the root CA.

```
$ openssl ca -days 730 -in server.req -md sha256 -extensions srv_cert -cert rootca.crt -
keyfile rootca.key -out server.crt
```

Finally you may validate the server certificate against the root CA.

```
$ openssl verify -verbose -purpose sslserver -CAfile rootca.crt server.crt
```

CubeSQL Configuration

Certification Setup

To use your server certificate with CubeSQL server you need one file containing the server certificate and the unencrypted private key in PEM encoding (ASCII Base64).

Save unencrypted private key in server.tmp.

```
$ openssl rsa -in server.key -out server.tmp
```

Merge server certificate and unencrypted private key.

```
$ cat server.crt server.tmp > cubesql.tmp
```

Remove unnecessary information and save to cubesql.pem.

```
$ sed -ne '/-BEGIN CERTIFICATE-/,/-END PRIVATE-/p' cubesql.tmp > cubesql.pem
```

Remove temporary files.

```
$ rm *.tmp
```

Please copy the cubesql.pem only to a secured folder on the CubeSQL server and remove it locally.

To validate the server certificate on the client side you can save the necessary information of the root CA as new file.

```
$ sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' rootca.crt > MyCompany.crt
```

The file "MyCompany.crt" contains no sensitive information and can be published with your software.

Configure CubeSQL Server

So you have a secure certificate but you need to use it in the right way to gain security in real terms. Please be sure to use the latest version of CubeSQL server published by SQLabs (http://www.sqlabs.com/cubesql_download.php).

Copy the server certificate (cubesql.pem) to a secured location on your CubeSQL server. I prefer a directory "cubesql/etc" which is only accessible by the cubesql user and contains the cubesql.settings and the cubesql.pem file.

Use following startup parameters for cubesql binary:

Full-Path to certificate file:	-m /full/path/to/cubesql.pem
Allow only SSL connections:	-q
Allow only TLS1.0 and above:	-k 1
Define 2048 bit for DH parameters:	-j 2048

Please note that the certificates for Diffie-Hellman parameters are random certificates generated on startup of CubeSQL. So this may take some time depending on the bit length and the CPU performance.

Via CubeSQLAdmin you may set the allowed ciphers as preference.
SET PREFERENCE ALLOWED_CIPHERS TO ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-CBC-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA

These ciphers will allow users with older MacBooks to connect to your server via TLS1.0 with Forward Secrecy. Newer clients will use better ciphers automatically. If you have bundled the necessary SSL and crypto libraries into your software you may want to use stricter ciphers and/or TLS versions (-k 2 ↦ TLS1.1 and above; -k 3 ↦ TLS1.2 only).

Please note that you need to restart your CubeSQL server to apply these changes.

Configure CubeSQL Client with RealBasic/Xojo

Now that the server is secure you have to configure the client to use the new possibilities in the right way. You need to set up the client to use SSL and to check the identity of the server.

Firstly set SSL and crypto libraries in the CubeSQL plugin.
CubeSQLPlugin.SSLLibrary = <FolderItem libssl>
CubeSQLPlugin.CryptoLibrary = <FolderItem libeay/libcrypto>

Keep also the OpenSSL version of the client up to date.
(openssl version -a / CubeSQLPlugin.OpenSSLVersion)

Next set the encryption of the client to SSL and also define the allowed ciphers on client side.
CubeSQLServer.Encryption = CubeSQLPlugin.kSSL
CubeSQLServer.SSLCipherList = "ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-CBC-SHA:DHE-RSA-AES256-SHA:@STRENGTH"
(Get a list of supported ciphers with openssl ciphers.)

Lastly set the root certificate to check the identity of the server you want to connect.
CubeSQLServer.RootCertificate = <FolderItem MyCompany.crt>

You really need to check that the file exists and is readable. If it is not, the client does not check the identity of the server so you won't try to connect and return an error.
CubeSQL client always checks for valid hostname and CA. At the moment it is not possible to check only CA and not the hostname.

Security

Secure Storage Location

Copy the server certificate (cubesql.pem) to a secured location on your CubeSQL server where only the process which runs the CubeSQL server has access.
Never copy the private key of your root CA to a server connected to the internet. If this key is compromised another user is able to generate valid server certificates and your security is completely lost regardless if your still using SSL/TLS.

Check Security via SSL Labs

After your successful configuration you may temporary forward port 443 to CubeSQL port 4430 and check your server security level via SSL Labs.

```
$ iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 -j REDIRECT --to-port 4430
```

Visit https://www.ssllabs.com/ssltest/ and type in your domain name. SSL Labs cannot verify your own created certificate of course, but you can ignore the rating T "not trusted" issue and have a look at the bars with "Protocol Support", "Key Exchange" and "Cipher Strength" and the details listed below.

Check Security via SSLyze

If you want to check your security offline and with your own CA, SSLyze is the way to go.
"SSLyze is a Python tool that can analyze the SSL configuration of a server by connecting to it. It is designed to be fast and comprehensive, and should help organizations and testers identify misconfigurations affecting their SSL servers." (Source: http://tools.kali.org/information-gathering/sslyze)

```
$ sslyze --regular www.mycompany.com
```

Source Code: https://github.com/iSECPartners/sslyze
Quick Start Guide: https://code.google.com/p/sslyze/wiki/QuickStart