MQTTS : Comment utiliser MQTT avec TLS ?

Julien Grossholtz(https://openest.io/author/openest/)
février 6, 2019(https://openest.io/2019/02/06/)

Table des matières

Protection de donnés avec chiffrement

Dans **un article précédent (https://openest.io/2018/05/22/mqtt-un-protocole-de-communication-pour-vos-objets-connectes/)** nous avons présenté le fonctionnement du protocole MQTT. Ici nous utilisont sa variante sécurisée : MQTTS. C'est une bonne pratique de l'utiliser, en particulier pour des **systèmes embarqués (https://openest.io/2019/12/17/developpement-linux-embarque-5-etapes-pour-commencer/)**.

L'objectif est d'établir une communication chiffrée entre un broker et des clients MQTTS présents sur la même machine. Les options utilisées pour OpenSSL sont une suggestion, à vous de déterminer lesquelles correspondent à vos besoins exactement lorsque vous aurez réussi à établir la communication.

Dans un premier temps assurez-vous d'avoir installé mosquitto (broker et clients) :

```
# Sous Debian/Ubuntu
$ apt-get install mosquitto mosquitto-clients

# Sous Fedora
$ dnf install mosquitto
```

Logiquement **OpenSSL (https://www.openssl.org/)** est également déjà installé sur votre machine fonctionnant sous Linux.

Génération des clés et certificats avec OpenSSL

Autorité de certification

On crée un dossier de travail puis on génère la clé et le certificat de notre propre autorité de certification :

Attention : le Common Name (CN) ne dois pas être le même que pour les clients.

```
$ mkdir certs
$ cd certs
$ mkdir ca
$ cd ca/
$ openssl req -new -x509 -days 365 -extensions v3_ca -keyout ca.key -out ca.crt
Generating a RSA private key
.....+++++
..............................+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:France
Locality Name (eg, city) [Default City]:Strasbourg
Organization Name (eg, company) [Default Company Ltd]:opeNest
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:openest.io
Email Address []:contact@openest.io
$ ls
ca.crt  ca.key
$ cd ..
```

Certificats pour le broker

A présent on dispose d'un certificat. Maintenant on peut créer les clés et certificats du broker :

On génère une clé privée (sans mot de passe):

```
$  mkdir broker
$ cd broker
$  openssl genrsa -out broker.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................
.....................................................+++++
.....................................................................
........+++++
e is 65537 (0x010001)
$ ls
broker.key
```

On crée un fichier de requête de signature à partir de cette clé :

```
$ openssl req -out broker.csr -key broker.key -new
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:France
Locality Name (eg, city) [Default City]:Strasbourg
Organization Name (eg, company) [Default Company Ltd]:opeNest
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:contact@openest.io

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
$ ls
broker.csr  broker.key
```

Maintenant on peut donne le fichier. csr (Certificate Signing Request) de requête de signature du certificat à notre authorité de validation :

```
$ openssl x509 -req -in broker.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -
CAcreateserial -out broker.crt -days 100
Signature ok
subject=C = FR, ST = France, L = Strasbourg, O = opeNest, CN = localhost,
emailAddress = contact@openest.io
Getting CA Private Key
Enter pass phrase for ../ca/ca.key:
$ ls
broker.crt  broker.csr  broker.key
$ rm broker.csr
```

Les certificats du client MQTTS

On recommence pratiquement la même chose, cette fois-ci pour certifier un client :

```
[julien@alderan certs]$ mkdir client
[julien@alderan certs]$ cd client
[julien@alderan client]$ openssl genrsa -out client.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................+++++
.........+++++
e is 65537 (0x010001)
[julien@alderan client]$ openssl req -out client.csr -key client.key -new
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:FR
State or Province Name (full name) []:France
Locality Name (eg, city) [Default City]:Strasbourg
Organization Name (eg, company) [Default Company Ltd]:opeNest
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:contact@openest.io

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
$ ls
$ openssl x509 -req -in client.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -
CAcreateserial -out client.crt -days 100
Signature ok
subject=C = FR, ST = France, L = Strasbourg, O = opeNest, CN = localhost,
emailAddress = contact@openest.io
Getting CA Private Key
Enter pass phrase for ../ca/ca.key:
$ ls
client.crt  client.csr  client.key
$ cd ..
```

Configuration du broker MQTTS

A présent vous devriez avoir tous ces fichiers :

```
$ tree .
.
├── broker
│   ├── broker.crt
│   └── broker.key
├── ca
│   ├── ca.crt
│   ├── ca.key
│   └── ca.srl
└── client
    ├── client.crt
    └── client.key
```

A présent nous pouvons configurer mosquitto : le broker. Je vous présente ici une version simplifiée mais fonctionnelle de son fichier de configuration :

```
$ sudo vim /etc/mosquitto/mosquitto.conf
```

```
port 8883

cafile /home/openest/certs/ca/ca.crt
#capath /home/openest/certs/ca

# Path to the PEM encoded server certificate.
certfile /home/openest/certs/broker/broker.crt

# Path to the PEM encoded keyfile.
keyfile /home/openest/certs/broker/broker.key
require_certificate true
```

Le port 8883 est le port standard des connexions MQTT chiffrés (pour les connexions non chiffrés c'est 1883). Mais vous pouvez utiliser n'importe quel port du moment que vous utilisez le même pour les clients.

Assurez vous qu'il n'est pas déjà en fonctionnement puis démarrez votre broker afin qu'il utilise ce fichier de configuration :

```
$ sudo mosquitto -v -c /etc/mosquitto/mosquitto.conf
[sudo] password for openest:
1548955097: mosquitto version 1.5.4 starting
1548955097: Config loaded from /etc/mosquitto/mosquitto.conf.
1548955097: Opening ipv4 listen socket on port 8883.
1548955097: Opening ipv6 listen socket on port 8883.
```

Configuration d'un client MQTTS

Dans un second terminal vous pouvez maintenant publier en utilisant mosquitto_pub :

```
$ cd client
$  mosquitto_pub -p 8883 --cafile ../ca/ca.crt --cert client.crt --key
client.key -h localhost -m hello -t /world
```

Dans le terminal où vous avez lance le broker vous verrez que le message est bien arrivé:

```
1549402034: New connection from ::1 on port 8883.
1549402034: New client connected from ::1 as mosqpub|32006-alderan (c1, k60).
1549402034: No will message specified.
1549402034: Sending CONNACK to mosqpub|32006-alderan (0, 0)
1549402034: Received PUBLISH from mosqpub|32006-alderan (d0, q0, r0, m0,
'/world', ... (5 bytes))
1549402034: Received DISCONNECT from mosqpub|32006-alderan
1549402034: Client mosqpub|32006-alderan disconnected.
```

Maintenant vous pouvez créer un jeu de clés supplémentaire pour un second client et utiliser mosquitto_sub pour vous abonner, en utilisant bien les même options (*-p 8883 –cafile ../ca/ca.crt –cert client.crt –key client.key -h localhost*).

Erreurs courantes lors de la mise en place de MQTTS :

Voici quelques problèmes que vous pourriez rencontrer en cours de route avec quelques suggestion pour vous aider à les comprendre et les corriger .

Error: A TLS error occurred.

```
$ mosquitto_pub --cafile ../ca_authority/ca.crt --cert client.crt --key
client.key -h localhost  -m toto -t /toto
Error: A TLS error occurred.



# Du coté du broker:
1548955428: New connection from ::1 on port 8883.
1548955428: OpenSSL Error: error:14094438:SSL routines:ssl3_read_bytes:tlsv1
alert internal error
1548955428: Socket error on client <unknown>, disconnecting.
```

La cause probable est que le hostname de votre broker ne corresponde pas au Common Name (CN) que vous avez défini lors de l'étape de génération du fichier .csr de « requête ».

On rencontre également cette erreur lorsque le Common Name de l'autorité de certification est le même que celui du client. Il est necessaire que le CN de l'autorité de certification soit différent des clients.

peer did not return a certificate

```
# Du coté du broker vous pouvez avoir cette erreur:
OpenSSL Error: error:1417C0C7:SSL routines:tls_process_client_certificate:peer
did not return a certificate
```

Cela signifie que l'option require_certificate de mosquitto.conf est activée, mais que le client n'envoie pas son certificat. Assurez vous que mosquitto_pub ou mosquitto_sub utilisent bien ces options : *–cert client.crt –key client.key*. Sinon vous pouvez essayer de désactiver temporairement l'option *require_certificate true* du broker.

Vérifier le contenu d'un certificat**A TLS error occured:**

Pour debugger votre communication il est souvent utile de vérifier les informations contenues dans vos certificats. En particulier les CN. Si vous rencontrez des problèmes vous pouvez les vérifier facilement avec OpenSSL:

```
openssl x509 -in broker.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            ab:2e:86:94:c4:18:1d:a0
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = FR, ST = France, L = Strasbourg, O = openest, CN = rootca
        Validity
            Not Before: Feb  1 10:23:19 2019 GMT
            Not After : May 12 10:23:19 2019 GMT
        Subject: C = FR, ST = Some-State, O = Internet Widgits Pty Ltd, CN =
localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:be:3a:7a:45:ad:f4:d5:33:c8:94:da:9e:38:5b:
                    d1:2b:13:26:c6:77:45:bc:a0:c3:f0:7a:56:8a:2d:
                    5b:7c:6c:75:54:a4:2b:1f:e3:ab:2a:c8:4f:74:5f:
                    47:7b:90:d2:39:1e:20:54:2c:55:61:41:81:42:72:
                    9d:ee:d1:ab:3e:3f:0f:fc:73:01:df:e0:3d:de:ae:
                    9c:d0:51:8c:dc:34:92:ff:53:97:c8:3c:86:c6:4d:
                    ae:7a:17:5e:65:38:21:f3:dc:1d:cc:28:a9:f9:ac:
                    3c:24:9a:01:f3:f8:75:41:81:81:23:7b:17:e0:55:
                    f4:1f:0f:b1:d0:f2:5e:de:d4:7e:72:bf:a9:8d:8f:
                    ab:4f:5f:38:3b:8c:49:cf:cf:2c:22:ba:0c:7e:7a:
                    01:65:08:1a:e9:1a:b6:d7:9c:93:bb:66:b4:83:65:
                    91:ae:89:e1:e3:11:21:33:f6:b2:4c:f3:c6:3d:b1:
                    7b:3a:28:6d:e7:36:b1:df:17:41:e1:e0:f3:d1:0c:
                    62:3a:e1:af:c2:62:47:30:6f:9b:28:4e:e4:aa:cb:
                    35:f0:f1:fd:17:f5:b5:cc:0d:99:b6:25:38:57:17:
                    1d:a6:f5:47:9c:76:8c:6b:32:aa:75:2e:f5:ff:b1:
                    25:77:31:fa:b9:5a:33:d4:18:8d:86:bf:9a:b0:c8:
                    1d:69
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         22:08:e1:5a:5f:d9:33:7e:ca:f3:98:88:ba:ab:5d:34:b7:0e:
         58:b9:15:f8:57:e5:eb:17:ce:ed:80:0e:f6:90:7c:45:fc:1a:
         c3:1b:7a:29:8d:6b:e2:79:4a:23:ba:6b:74:6f:c5:1e:93:bd:
         ce:b2:7a:60:74:ba:c5:49:9a:48:67:92:c9:80:02:a0:43:c4:
         3a:90:b0:ae:aa:7d:49:ab:80:b8:64:e7:c7:0a:b2:84:90:01:
         78:79:cb:95:d3:88:4f:ae:57:6c:bd:d3:88:40:e1:3e:f7:b4:
         79:7f:a4:d8:01:37:9a:03:78:f5:ec:81:5e:1a:cb:bc:8f:da:
         eb:1d:2e:07:72:9d:d9:0e:9f:28:00:6e:4d:07:d8:5a:5c:ce:
         e7:bc:7d:30:44:77:fd:c0:c3:a8:4f:b8:e1:6f:62:32:ac:a8:
         0c:06:50:19:5a:02:1a:10:a0:55:bb:00:86:0f:d4:22:49:0c:
         14:58:65:96:00:e2:01:05:d2:fe:d3:49:c2:1f:e1:21:25:5b:
         82:c6:bd:01:ac:5a:e4:65:b8:4c:5c:e1:a8:ff:41:e8:74:ad:
         80:8e:da:37:de:8f:30:0b:b7:e0:b2:d2:e2:4b:db:f5:5b:37:
         10:57:ac:f7:fe:db:bd:44:01:5c:40:f1:80:a1:6e:9e:05:08:
         fc:0e:91:85
```

Et avec paho-c-mqtt ?

Comment nous l'avons expliqué dans **un article précédent (https://openest.io/2018/05/22/mqtt-un-protocole-de-communication-pour-vos-objets-connectes/)**, nous avons choisi d'utiliser paho.mqtt.c pour un projet de communication m2m. Son API est bien documentée, voici un exemple d'utilisation qui reprend les fichiers que nous avons généré au début de cet article:

```
MQTTAsync_create( &paho_handler, "ssl://localhost:8883", "paho_test_client",
MQTTCLIENT_PERSISTENCE_NONE, NULL);
```

Et la partie plus spécifique qui permet de configurer les certificats :

```
        MQTTAsync_connectOptions conn_opts =
MQTTAsync_connectOptions_initializer;
         MQTTAsync_SSLOptions ssl_opts = MQTTAsync_SSLOptions_initializer;

         conn_opts.keepAliveInterval = 10;
         conn_opts.cleansession = 1;
         conn_opts.onSuccess = _on_connect;
         conn_opts.onFailure = _on_connect_failure;
         conn_opts.context = g_mqtt_client.paho_handler;

         if(g_mqtt_client.certs_path != NULL ) {

                 conn_opts.ssl = &ssl_opts;
                 conn_opts.ssl->trustStore = "/home/openest/certs/ca/ca.crt";
                 conn_opts.ssl->privateKey =
"/home/openest/certs/client/client.key";
                 conn_opts.ssl->keyStore =
"/home/openest/certs/clientclient.crt";
                 conn_opts.ssl->ssl_error_cb = ssl_error_cb;
                 conn_opts.ssl->enableServerCertAuth = 1;
                 conn_opts.ssl->verify = 1;
         }

        ret = MQTTAsync_connect( &paho_handler, &conn_opts);
```

Références

Voici quelques documents qui nous ont été utiles pour chiffrer la communication MQTT de nos devices:

- la man page de **mosquitto (https://mosquitto.org/man/mosquitto-8.html)**
- la man page de **mosquitto-tls (https://mosquitto.org/man/mosquitto-tls-7.html)**
- la **documentation de la structure MQTTClient_SSLOptions (https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/struct_m_q_t_t_client___s_s_l_op** de paho.mqtt.c

Pour la rédaction de cet article avons utilisé la verison 1.5.4 de mosquitto (broker et clients) et la version 1.3.0 de paho.c.mqtt.

Partager cet article