Travail Pratique : MQTT - Qualité de Service (QoS) et Chiffrement TLS

Contents

1 Introduction

1.1 Visualisation des logs du broker MQTT

Un accès SSH a été mis en place afin de permettre aux utilisateurs de consulter en temps réel les logs du broker MQTT **Mosquitto**, qui est déployé en conteneur avec **Eclipse Mosquitto**. Le broker écoute sur le port **1883** pour les connexions non sécurisées et sur le port **8883** pour les connexions chiffrées via **TLS**. Les logs du conteneur sont accessibles en se connectant au serveur via **SSH**.

1.1.1 Connexion au serveur

Pour accéder aux logs, connectez-vous au serveur via SSH en utilisant les identifiants suivants :

- **Utilisateur** : `mqttlogs`
- **Mot de passe** : `mqtt2025`
- **Adresse du serveur** : `hugo-serveur.cloudns.eu`

Utilisez la commande suivante pour vous connecter :

```
ssh mqttlogs@hugo-serveur.cloudns.eu
```

Une fois connecté, les logs du broker **Mosquitto** s'afficheront automatiquement en temps réel. Vous pourrez ainsi observer les connexions des clients et les messages échangés. Grâce à ces logs, il est possible de suivre en détail les échanges de données, y compris les différents paquets MQTT tels que **CONNECT**, **CONNACK**, **PUBLISH**, **SUBSCRIBE**, **PINGREQ** et **PINGRESP**, permettant ainsi d'analyser le fonctionnement du broker et le comportement des clients.

1.1.2 Explication

Sur le serveur, un nouvel utilisateur nommé **mqttlogs** a été créé. Un script a ensuite été mis en place pour exécuter la commande suivante et afficher les logs en temps réel :

```
sudo docker logs -f mosquitto
```

La configuration SSH de cet utilisateur a ensuite été modifiée afin qu'il exécute automatiquement ce script dès sa connexion, lui interdisant ainsi l'accès à toute autre partie du système et limitant son interaction au seul affichage des logs.

1.2 Fichiers mis a disposition

Plusieurs fichiers sont mis à disposition pour ce travail pratique :

- `ClientMQTT_Ping.py` : Mettre en évidence les échanges de messages `PINGREQ` et `PINGRESP`.
- `ClientMQTT_QoS.py` : Tester les niveaux de qualité de service (QoS) 0, 1 et 2.
- `ClientMQTT_TLS.py` : Connexion sécurisée avec TLS.
- `mosquitto.crt` : Certificat public pour la connexion TLS.
- `ClientMQTT_Pub.py` : Simple script Python pour publier un message sur un topic.
- `ClientMQTT_Sub.py` : Simple script Python pour s'abonner à un topic.

1.3 Commandes utiles

Lors de ce TP, vous utiliserez Wireshark pour analyser les échanges MQTT et TLS. Voici quelques filtres utiles pour faciliter l'observation des paquets pertinents.

1.3.1 Filtres généraux

- `mqtt` : Affiche tous les paquets du protocole MQTT.
- `tcp.port == 1883` : Affiche uniquement les paquets circulant sur le port MQTT non sécurisé.
- `tcp.port == 8883` : Affiche uniquement les paquets circulant sur le port MQTT sécurisé (TLS).
- `tcp.port == 1883 || tcp.port == 8883` : Affiche les paquets circulant sur l'un des deux ports MQTT.

1.3.2 Filtres spécifiques aux messages MQTT

Value	MQTT message	Direction of flow	Description
1	**CONNECT**	Client to server (i.e Broker)	**Client requests to connect to server**
2	**CONNACK**	Server to client	**Connect acknowledgment**
3	**PUBLISH**	Client to server OR server to client	**Publish message**
4	**PUBACK**	Client to server OR server to client	**Publish acknowledgment**
5	**PUBREC**	Client to server OR server to client	**Publish received**
6	**PUBREL**	Client to server OR server to client	**Publish released**
7	**PUBCOMP**	Client to server OR server to client	**Publish complete**
8	**SUBSCRIBE**	Client to server	**Client subscribe request**
9	**SUBACK**	Server to client	**Subscribe acknowledgment**
10	**UNSUBSCRIBE**	Client to server	**Unsubscribe request**
11	**UNSUBACK**	Server to client	**Unsubscribe acknowledgment**
12	**PINGREQ**	Client to server	**Ping request**
13	**PINGRESP**	Server to client	**Ping response**
14	**DISCONNECT**	Client to server	**Client is disconnecting**

Figure 1: Types de messages MQTT

Les messages MQTT possèdent différents types identifiés par un numéro spécifique. On peut filtrer ces messages dans Wireshark avec la syntaxe suivante :

`mqtt.msgtype == X`

où `X` est la valeur correspondant au type de message (voir la figure **??**).

1.3.3 Filtres pour l'analyse TLS

- `tls` : Affiche tous les paquets du protocole TLS.
- `tls.handshake.type == 1` : Filtre les paquets contenant un message **Client Hello**.
- `tls.handshake.type == 2` : Filtre les paquets contenant un message **Server Hello**.
- `tls.handshake.type == 11` : Filtre les paquets contenant un **Certificat** envoyé par le serveur.
- `tls.record.content_type == 23` : Affiche les paquets contenant des données applicatives chiffrées (**Application Data**).

L'ensemble de ces filtres peuvent être combinés avec des opérateurs logiques comme `AND` et `OR` pour affiner les résultats de la capture.

2 Mise en œuvre des niveaux de qualité de service (QoS) avec MQTT

2.1 Objectifs

Ce travail pratique a pour but de comprendre le fonctionnement des protocoles de qualité de service (QoS) dans le cadre de la communication MQTT. Vous utiliserez un script Python pour vous connecter à un broker MQTT et observer les différents comportements en fonction des valeurs de QoS (0, 1 et 2).

2.2 Pré-requis

- Accès au logs du serveur MQTT via SSH (voir section précédente).
- Connaissance des concepts de base de MQTT et des niveaux de QoS.
- Script Python mis à disposition pour simulation de la communication avec le broker.

2.3 Connexion au Broker et Analyse des Logs

Lancez le script Python `ClientMQTT_QoS.py`, en choisissant un QoS de votre choix (0, 1, ou 2). Le script publiera un message et s'abonnera à un topic.

Examinez les logs du serveur MQTT pour identifier les éléments suivants :

- Le message de connexion `CONNACK` et les informations associées (p. ex. le code de retour).
- Les échanges de messages lors de l'abonnement (avec `SUBSCRIBE` et `SUBACK`).
- Les échanges de messages pour chaque niveau de QoS (publiés avec `PUBLISH`, `PUBREC`, `PUBREL`, `PUBCOMP`).

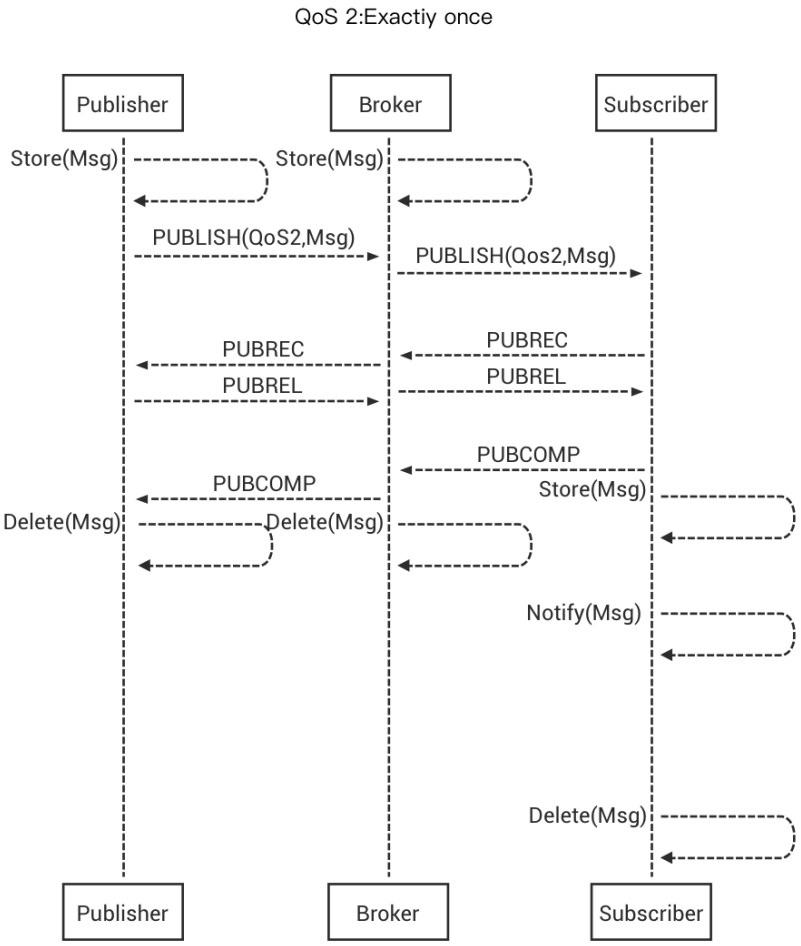


Figure 2: Exemple d'échanges de messages avec QoS 2

2.4 Identification des comportements selon le QoS

En fonction du QoS sélectionné (0, 1 ou 2), analysez les différences dans les échanges et dans les messages échangés entre le client et le broker.

Répondez aux questions suivantes :

- Que se passe-t-il lorsque vous utilisez QoS 0 ? Identifiez les messages échangés entre le client et le broker dans les logs.
- Que se passe-t-il avec QoS 1 ? Observez l'échange des messages et notez les différences par rapport au QoS 0.
- Que se passe-t-il avec QoS 2 ? Décrivez en détail les étapes supplémentaires de la communication (par exemple, `PUBREC`, `PUBREL`, `PUBCOMP`).

2.5 Discussion et Analyse

Une fois l'analyse terminée, répondez aux questions suivantes :

- Quelle est la différence principale entre QoS 0, 1 et 2 en termes de fiabilité et de performance ?
- Comment le QoS influe-t-il sur la latence et le nombre de messages échangés entre le client et le broker ?
- Pourquoi le QoS 2 est-il utilisé dans des cas où la fiabilité maximale est nécessaire ?

3 Mise en œuvre des messages PINGREQ et PINGRESP avec MQTT

Utilisez le script Python `ClientMQTT_Ping.py`, que signifient les messages `PINGREQ` et `PINGRESP` dans le contexte de MQTT ?

Que signifient les paramètres suivants dans le contexte de MQTT ?

- `p0`, `p1`, `p2`
- `c0`, `c1`
- `kx`

Observez les logs du serveur MQTT pour identifier les échanges de messages `PINGREQ` et `PINGRESP`. Comment ces messages sont-ils utilisés pour maintenir la connexion entre le client et le broker ?

4 Mise en évidence des parametres lors de la publication de messages avec MQTT

Lancez le script Python `ClientMQTT_Pub.py` pour publier un message sur un topic de votre choix. Analysez les logs du serveur MQTT pour identifier les paramètres des messages échangés.

Que signifient les paramètres suivants dans le contexte de MQTT ?

- `dx`
- `q0`, `q1`, `q2`
- `r0`, `r1`
- `mx`

Envoyez plusieurs messages à la suite sur le même topic et observez les valeurs de ces paramètres. Comment évoluent-ils à chaque message publié ?

5 Mise en œuvre du chiffrement TLS avec MQTT

5.1 Introduction : Le chiffrement TLS

Le protocole TLS (Transport Layer Security) sécurise les communications réseau en garantissant la confidentialité, l'intégrité et l'authenticité des échanges. Il repose sur une paire de clés : une clé privée (`mosquitto.key`), conservée sur le serveur, et un certificat public (`mosquitto.crt`), distribué aux clients.

Lorsqu'un client se connecte, il envoie un message **Client Hello** (TLSv1.3). Le serveur répond avec un **Server Hello** accompagné de son certificat. Si celui-ci est valide, une clé de session est générée pour chiffrer toutes les communications MQTT.

Pour les plus curieux, voici un super site qui illustre les échanges TLS 1.3: `https://tls13.xargs.org/`

5.2 Mise en place et analyse

Le broker Mosquitto est configuré pour écouter sur le port **8883**, avec les certificats TLS déjà générés et en place.

Connexion d'un client Python sécurisé
Le script client doit utiliser la fonction `tls_set()` pour spécifier le certificat `mosquitto.crt`, qui est auto-signé (généré par vous-même et non par une autorité reconnue comme Let's Encrypt). Ce certificat doit être accessible au client pour établir une connexion sécurisée via TLS, en vérifiant le serveur.

Analyse du chiffrement avec Wireshark
En capturant le trafic sur le port **8883**, il est possible d'observer les échanges TLS : identification des messages **Client Hello** et **Server Hello**, ainsi que la transformation des messages MQTT en **Application Data** chiffrée.

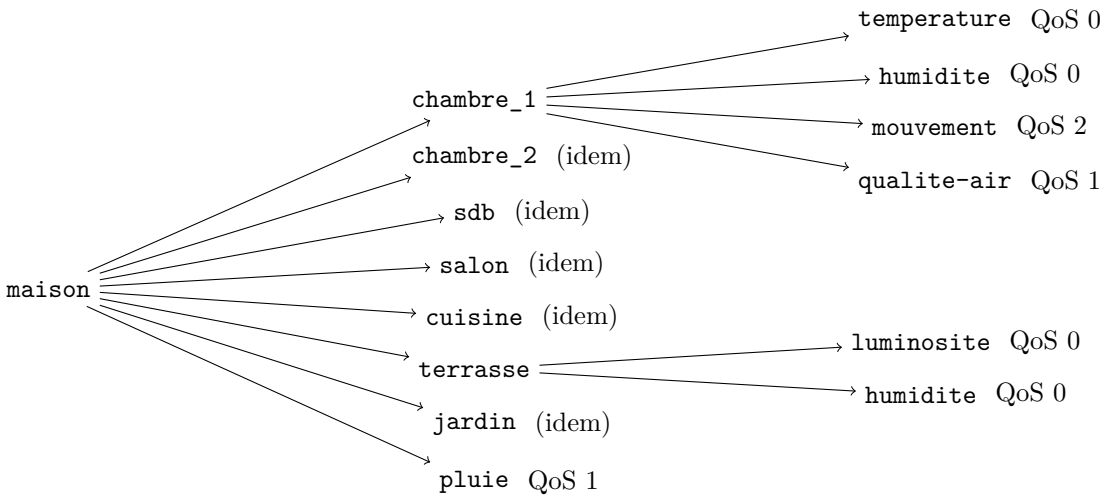
5.3 Analyse et compréhension

Quelques questions permettent de valider la compréhension du chiffrement TLS :

- Quels sont ses objectifs ?
- Que contient le message **Client Hello** ?
- Comment s'assurer que la communication MQTT est bien chiffrée ?
- Pourquoi Wireshark ne peut-il plus lire le contenu des messages MQTT ?

6 Mise en oeuvre d'un environnement complet avec MQTT

Nous allons simuler un environnement d'une maison connectée avec plusieurs capteurs. La hiérarchie des topics est représentée dans l'arbre suivant :



Ces topic représentent les différents capteurs de la maison connectée. Chaque capteur publie des informations sur son topic respectif. Par exemple :
Si on veut souscrire à la température de la chambre 1, on souscrit au topic `maison/chambre_1/temperature`.
Si on veut toute les informations du salon, on souscrit au topic `maison/salon/#`.
Si on veut les informations de tous les capteurs de la maison, on souscrit au topic `maison/#`.
Si on veut les informations de toutes les temperature de la maison, on souscrit au topic `maison/+/temperature`.

6.1 Question

Mettre en oeuvre quelques commandes pour comprendre le fonctionnement des # et des + dans les topics MQTT.