Exercise Sheet 5

Deadline: July 09, 2024 – 04:00 am CEST

Project "'Room Manager IoT Solution"'

Manually updating the room manager database, particularly the information about the current occupancy and the reserved status can be cumbersome. In many cases it would be preferable to update this information automatically (via sensors in the rooms) and get push notifications upon changes to these properties. The MQTT protocol is well suited for such a IoT-like scenarios, implementing the *pub-sub* communication pattern. Hence, in this example you should implement two small MQTT applications: one that updates the reservations status and occupancy and one that subscribes to these *topics* and prints the update notifications.

Getting to know MQTT

Before you start implementing, you should first inform yourself about MQTT. The MQTT library of the Eclipse *Paho* project will be used for the implementation of the client application

MQTT is a *publish-subscribe* protocol that allows the transmission of data in the form of messages. A central *broker* establishes the connection to even resource-poor nodes. The messages are published under *topics*.

- The blog posts at `https://www.hivemq.com/mqtt-essentials/` present a good introduction into the topic.
- For concise documentation you can also refer to the manpage from the *mosquitto* project: `https://mosquitto.org/man/mqtt-7.html`
- Familiarize yourself with the API documentation of the Paho-C API. `https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/`
- There are examples listed in the documentation. Try these out.
- For the encryption and authentication aspects, look at the API documentation for the `MQTTClient_SSLOptions` structure and how it is used in the test programs of the Paho project (`https://github.com/eclipse/paho.mqtt.c/`).

Test with the mosquitto clients

For Linux there is the package *mosquitto* which provides a MQTT broker as well as ready-to-use command line tools for manual 'publish' and 'subscribe'.

Here we will first test the function of the publish-subscribe protocol using the `mosquitto` tools.

There is a MQTT broker already configured and running at `mqtt.roomman.dahahm.de`. However, you can also locally create your own `mosquitto` broker [1]. The configuration of the broker which you can also use for your own broker, can be found in the `configs` folder in your repository. (*note:* Attention if necessary paths have to be adapted!)

The mosquitto package provides generic publish and subscribe applications. With `mosquitto_sub` you can subscribe to arbitrary topics and output the messages to the console, and with `mosquitto_pub` to send messages to topics.

The MQTT broker is configured so that only certain topics can be created and used. can be used. However, for initial testing, there is the topic "'`test`"'.

Subscribe to the topic with
```
mosquitto_sub -u <USER> -P <PASSWORD> -t "test" -h
mqtt.roomman.dahahm.de
```

In a second console, you can then send messages to the Topic with:
```
mosquitto_pub -u <USER> -P <PASSWORD> -t "test" -h
mqtt.roomman.dahahm.de -m "MESSAGE"
```

Next, start setting up your own small C program, that subscribes to the test topic.

Send again with `mosquitto_pub` to the test topic and check their reception with your own subscriber program.

Exercise: Implementing the MQTT clients

In your repository you find the broker configuration and a template for implementing the clients. The two files `src/roommanager_client.c` and `src/roommanager_iot.c` are already complete. What is missing is the implementation of the functions in `src/roomman_mqtt.c`.

You will first need to connect to the broker using the correct credentials. The client application (`roommanager_client`) can subscribe to the necessary topics using `client` as username **and** password, the IoT application (`roommanager_iot`) can write these properties with `iot` as username **and** password. (Additionally there is

[1] projectpage: `https://mosquitto.org`, documentation at `https://mosquitto.org/man/mosquitto-8.html`

an admin user (`admin` as username **and** password) that can read and write to the properties.)

The following topics are available via the broker:

- **Topic:** `roomman/room/{ROOMID}/reserved`
  - **Description:** This topic represents the current reservation status of a room.
    The `ROOMID` and the payload for this topic shall be integer values.[2] Zero in the payload shall be interpreted as the room being currently free, other values indicate the room currently being reserved.
  - **ACL:** client-r, iot-w, admin-rw
  - **QoS:** Confirmed message, exactly once.
- **Topic:** `roomman/room/{ROOMID}/persons`
  - **Description:** This topic represents the current occupancy of a room.
    The `ROOMID` and the payload for this topic shall be integer values.
  - **ACL:** client-r, iot-w, admin-rw
  - **QoS:** Confirmed message, at least once.

Confidentiality: Encrypted Communication via TLS

Besides the unencrypted connection to the MQTT broker (default port 1883) there is also the possibility to connect encrypted via TLS (default port 8883). To ensure that the client knows that it is dealing with the correct broker, a PKI (public key infrastructure) based on X.509 certificates is used. A CA (Certificate Authority) has signed the certificate signed by the broker. If you trust this CA, then you can determine whether the broker is also trustworthy.

Extend your program in such a way that an encrypted connection to the broker is established. Use the provided certificate of the CA for the encrypted connection.

Use appropriate means to check the data communication. Are the messages transmitted encrypted?

Generate your own CA certificate and use it. Is it recognized that the server cannot be trusted?

[2] The server will accept any room ID.

Authenticity: Authentication via X.509

Using X.509 certification, the client application can also authenticate itself to the server. For this purpose, our MQTT broker on port 8884 offers another access.

Use OpenSSL to create an RSA key pair for the IoT application with a minimum length of 2048 bits and a corresponding X.509 certificate. Fill in the necessary details in a meaningful way. Setting a password is optional.

Then generate (also with OpenSSL) a CSR (Certificate Signing Request) for your certificate and send it to `oliver.hahm@fb2.fra-uas.de`. You will receive a certificate signed with the private CA key.

Document their actions: Put their generated certificate files in the `certs` folder and add them to the git repo.

Extend your application to use your certificate for authentication to the broker. This should be done using the program parameter flag `-V` to enable this.

Required packages

If you want to develop this project on your private computer you will need to install a few packages. On Debian based systems (Ubuntu, Mint ...) the following packages are required:

- libmosquitto-dev
- mosquitto
- mosquitto-clients
- libpaho-mqtt-dev