IoT Home Challenge 3

Bernardo Camajori Tedeschini, Luca Ferraro, Fabio Losavio

15 - May - 2020

1 What's the difference between the message with MID: 3978 and the one with MID: 22636?

We use as filters:

- `coap.mid == 3978`
- `coap.mid == 22636`

Message with MID 3978:
It's a confirmable message (packet number 9701) since Type: **Confirmable (0)** and it is present an ACK (packet number 6702).

Message with MID 22636:
It's a non-confirmable message (packet number 6943) since Type: **Non-Confirmable (1)**; since it is non-confirmable, the message is not ACKed.

Both messages use DELETE method as it can be seen in Code: **DELETE (4)**. The ACK of the confirmable message indicates the method DELETE is not allowed.

2 Does the client receive the response of message No. 6949?

From the fields of **COAP** protocol we can see that it is a Confirmable message and that it uses a **GET method**. To see if it has received the Response, we search for the same Token, that is *6f:b6:3c:18*.
To do this we use the filter:

- `coap.token == 6f:b6:3c:18`

Message 6953 is the Response: it's an ACK message.

3 How many replies of type connectable and result code "Content" are received by the server "localhost"?

We need to use as filters:

- `Destination address = localhost:  ip.dst_host == 127.0.0.1`
- `Result code must be = \Content":  coap.code == 69`

We don't care about the fact that they must be confirmable since content messages piggybacks ACKs.
Using the filter `(ip.dst_host == 127.0.0.1) && (caop.type == 69)` we can see 8 packets.

To do a further check, we've also tried to use the filter:
`(ip.dst_host == 127.0.0.1 && coap.code == 69) || (ip.src_host == 127.0.0.1 && coap.code == 69 && coap.type == 0)`
This filter would find eventual "content" packets sent with delay because the resource was not available when requested. In this case, the server would have sent an ACK with no content and a confirmable message with the content as soon as the resource would have become available.
The result is the same of the previous filter.

4 How many messages containing the topic "factory/department*/+" are published by a client with user name: *"jane"*?

Where * replaces the dep. number, e.g. factory/department1/+, factory/department2/+ and so on. (btw, * is NOT an MQTT wildcard)

- `Message must be a publish message:  mqtt.msgtype == 3`
- `Publisher must be Jane:  mqtt.username == jane`

So we apply as filters: `(mqtt.msgtype == 3) && (mqtt.username == jane)`.
If we do this, we see jane has published nothing of any topic.
If we apply as filter only mqtt.username == jane, we can see 3 CONNECT messages with MQIsdp (the old name of MQTT) that have clientID:

- 0sMEkncjoYI47sofTNzUON
- 25cGIdhv5Oi9qC7l7Ykaju
- 5a9OJR9zWKLU8E8lqrAjvw

There is also a 4th packet, that is a CONNECT message containing a Last Will Message:

- Will topic: $factory/department1/section3/hydraulic_valve$

- Will message: “error: cyisksbh”

Anyway, each time Jane connects to the broker, it opens a socket with the following source ports: `42821, 40989, 40005, 50985`.
For that reason, we must look for all the publish messages sent by any of these port and to do it we use the following filter:

```
(tcp.srcport == 42821 || tcp.srcport == 40989 || tcp.srcport == 4005
|| tcp.srcport == 50985) && (mqtt.msgtype == 3)
```

In this case, we obtain 7 messages, but only 6 published for there requested topic.

In case we also want the messages Jane has sent to subscribe to the requested topic, we need to use this filter:

```
(tcp.srcport == 42821 || tcp.srcport == 40989 || tcp.srcport == 40005
|| tcp.srcport == 50985) && (mqtt.msgtype == 3 || mqtt.msgtype == 8)
```

In this case, we obtain 13 packets, but only 12 corresponding to the requested topic.

5 How many clients connected to the broker “hivemq” have specified a will message?

- Filter for messages with a will message specified: `mqtt.conflag.willflag == 1`
- Filter for seeing only clients connected to the broker “hivemq” (using IP address resolution): `ip.dst == 18.185.199.22`

To find the correct answer we must use this filter:

```
(mqtt.conflag.willflag == 1) && (ip.dst == 18.185.199.22)
```

Using this filter we can see 7 messages, all having the same source IP address, meaning that only one client with a last will message is connected to broker.hivemq.org

6 How many publishes with QoS 1 don’t receive the ACK?

- Filtering for publish messages: `mqtt.msgtype == 3`.
- Filtering for QoS: `mqtt.qos == 1`

So combining them: `(mqtt.msgtype == 3) && (mqtt.qos == 1)`.
This way we find 124 messages published with QoS 1.

If we look for the **PUBACK** messages (mqtt.msgtype == 4) we find 74 messages.
If we do $124-74$ we find the messages published with QoS 1 that doesn't receive an ACK are 50.

7 How many last will messages with QoS set to 0 are actually delivered?

Using **Question7.py** we get 4 delivered last will message.
The program looks for the set last will message (looked for connect messages with will flag == 1) and then looks for the published message having as data a last will message found previously.

8 Are all the messages with $QoS > 0$ published by the client "4m3DWYzWr40pce6OaBQAfk" correctly delivered to the subscribers?

Using filter `mqtt.clientid == 4m3DWYzWr40pce6OaBQAfk` we get a connect message and then the socket of the connection.

We use the filter `tcp.srcport == 58313 && mqtt.msgtype == 3` for looking for the messages sent by that client ID: there are two messages, but both have QoS == 0. This means no message will be published with $QoS > 0$.

9 What is the average message length of a connect msg using mqttv5 protocol? Why messages have different size?

Using **Question9.py** we get as average length: 30.22.
The program looks for connect messages using MQTTv5 protocol and puts in a list the lengths of those packets and takes the average.
Different lengths are due to the presence of a last will message.

10 Why there aren't any REQ/RESP pings in the pcap?

Because publisher publish messages before the expiration of any keep alive timeout.