HOME CHALLENGE #3: SNIFFING

The goal of this challenge is to analyze internet traffic saved in a .pcapng file. We achieved this by using WireShark.
Code sections represent WireShark filters used, bolded parts are key elements for the answers.
Team member names and contacts are listed at the end of the file.

Questions

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

Filtering by MID:

```
coap.mid == 3978 || coap.mid == 22636
```

As we see from the capture, the message with MID 3978 is **confirmable**, which results in an ACK being received, while the one with MID 22636 is **non confirmable** and no ACK is sent back to the client.

No.	Time	Source	Destination	Protocol	Length	Info
6701	130.92440…	10.0.2.15	104.196.15.150	CoAP	70	CON, MID:3978, DELETE, TKN:8d 68 40 a0, End of …
6702	131.03994…	104.196.15.150	10.0.2.15	CoAP	62	ACK, MID:3978, 4.05 Method Not Allowed, TKN:8d …
6943	139.40341…	10.0.2.15	104.196.15.150	CoAP	70	NON, MID:22636, DELETE, TKN:55 20 eb 62, End of…

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

Either by manual search or by filtering with the `frame.number` display filter, we see that it's another message from the CoAP protocol, now with MID 28357. We can therefore use a similar filter as question #1:

```
frame.number == 6949 || coap.mid == 28357
```

From the capture we see that **the ACK message is sent at frame No. 6953**.

No.	Time	Source	Destination	Protocol	Length	Info
6949	139.40599…	10.0.2.15	104.196.15.150	CoAP	70	CON, MID:28357, GET, TKN:6f b6 3c 18, End of Bl…
6953	139.51914…	104.196.15.150	10.0.2.15	CoAP	62	ACK, MID:28357, 4.05 Method Not Allowed, TKN:6f…

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

First, we check how many requests of type confirmable are sent by localhost:

```
ip.addr == 127.0.0.1 && coap.type == 0
```

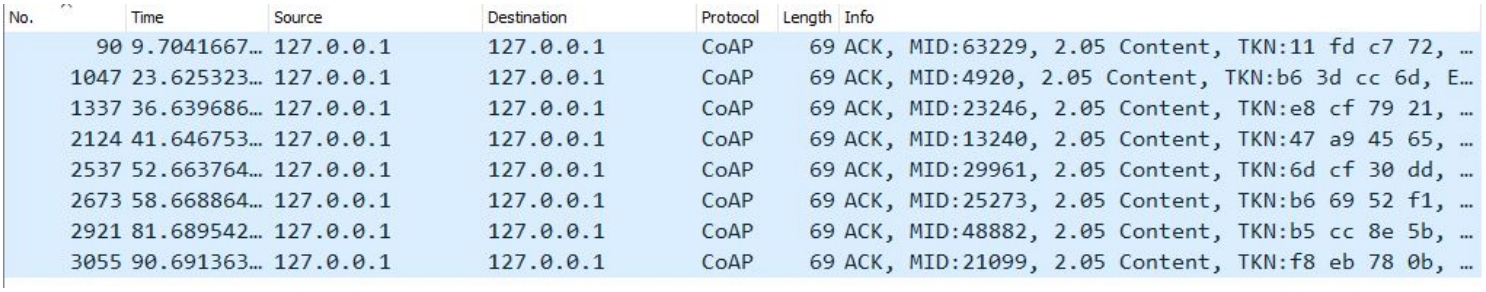
WireShark shows a total of 14 messages. To see the ACK replies, we just need to change the type property:

```
ip.addr == 127.0.0.1 && coap.type == 2
```

We see that all 14 ACKs are received, but some of those result in a "Not-Found" (132) result code. The final filter is used to capture only those with "Content" (69) code:

```
ip.addr == 127.0.0.1 && coap.type == 2 && coap.code == 69
```

The resulting number is 8. The final capture can be seen in the screenshot below.



No.	Time	Source	Destination	Protocol	Length	Info
90	9.7041667…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:63229, 2.05 Content, TKN:11 fd c7 72, …
1047	23.625323…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:4920, 2.05 Content, TKN:b6 3d cc 6d, E…
1337	36.639686…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:23246, 2.05 Content, TKN:e8 cf 79 21, …
2124	41.646753…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:13240, 2.05 Content, TKN:47 a9 45 65, …
2537	52.663764…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:29961, 2.05 Content, TKN:6d cf 30 dd, …
2673	58.668864…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:25273, 2.05 Content, TKN:b6 69 52 f1, …
2921	81.689542…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:48882, 2.05 Content, TKN:b5 cc 8e 5b, …
3055	90.691363…	127.0.0.1	127.0.0.1	CoAP	69	ACK, MID:21099, 2.05 Content, TKN:f8 eb 78 0b, …

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

We first filter the connect messages on the given username: all of them are on localhost, thus we need to associate each user to the port it's using to connect with the broker to distinguish between them. The first filter is:

```
mqtt.username == jane
```

This returns a list of 4 connections, where the broker is always working on the same port (1883) and the clients are the following:

- [No. 1554] username = "jane", password = "password", port = "42821"
- [No. 1623] username = "jane", password = "admin", port = "40989"
- [No. 3708] username = "jane", password = "idles", port = "40005"
- [No. 5816] username = "jane", password = "mybyrthday", port = "50985"

The next step is filtering the packets with the specified ports, intersecting the result with a regular expression match on the given topic and msgtype "Publish" (3).

```
(tcp.port == 42821 || tcp.port == 40989 || tcp.port == 40005 ||
tcp.port == 50985) && mqtt.msgtype == 3 &&  mqtt.topic matches
"factory\/department[0-9]\/"
```

This results in a total of **12 messages** shown below. Worth noting that some messages have an ID associated to them, while others do not: this depends whether the publish request is associated with a QoS of *at least once* (ACKed) or *at most once* (fire and forget, no ACK).

No.	Time	Source	Destination	Protocol	Length	Info
1580	39.396334183	127.0.0.1	127.0.0.1	MQTT	197	Publish Message [factory/department1/section1/plc]
1584	39.396671378	127.0.0.1	127.0.0.1	MQTT	200	Publish Message (id=4) [factory/department2/section4/
1585	39.396677011	127.0.0.1	127.0.0.1	MQTT	209	Subscribe Ack (id=3), Publish Message [factory/depart
1629	39.401658935	127.0.0.1	127.0.0.1	MQTT	210	Publish Message (id=1) [factory/department2/section1/
2540	53.418595851	127.0.0.1	127.0.0.1	MQTT	213	Publish Message (id=2) [factory/department2/section1/
2863	78.402142281	127.0.0.1	127.0.0.1	MQTT	194	Publish Message (id=5) [factory/department2/section4/
3132	98.453437832	127.0.0.1	127.0.0.1	MQTT	207	Publish Message (id=3) [factory/department2/section1/
3732	106.694026292	127.0.0.1	127.0.0.1	MQTT	192	Publish Message [factory/department2/section1/plc]
3942	106.736670623	127.0.0.1	127.0.0.1	MQTT	209	Publish Message [factory/department2/section4/hydrau
4164	108.285719336	127.0.0.1	127.0.0.1	MQTT	123	Publish Message [factory/department2/section2/deposit
5830	122.397452291	127.0.0.1	127.0.0.1	MQTT	209	Publish Message (id=2) [factory/department1/section3/
5831	122.397464197	127.0.0.1	127.0.0.1	MQTT	201	Publish Message [factory/department3/section1/deposit

5. <u>How many clients connected to the broker "hivemq" have specified a will message?</u>
First, we need to know what is the address of the *broker.hivemq.com* address (as seen in this link). This can be seen in the DNS requests with the following filter:

```
dns.qry.name == "broker.hivemq.com"
```

This gives us the following two addresses: 3.120.68.56 and 18.185.199.22. We can finally filter out the connect messages that only contain a will with:

```
(ip.dst_host == 3.120.68.56 || ip.dst_host == 18.185.199.22) &&
mqtt.willmsg
```

This returns a total of **16 connections**, as shown below.

No.	Time	Source	Destination	Protocol	Length	Info
544	21.167980418	10.0.2.15	3.120.68.56	MQTT	133	Connect Command
770	21.211984200	10.0.2.15	3.120.68.56	MQTT	127	Connect Command
1540	39.391816705	10.0.2.15	18.185.199.22	MQTT	149	Connect Command
2002	39.468809058	10.0.2.15	18.185.199.22	MQTT	123	Connect Command
2266	44.443554102	10.0.2.15	3.120.68.56	MQTT	127	Connect Command
2304	44.459288018	10.0.2.15	3.120.68.56	MQTT	125	Connect Command
2481	49.447970967	10.0.2.15	18.185.199.22	MQTT	127	Connect Command
4681	109.334623694	10.0.2.15	18.185.199.22	MQTT	121	Connect Command
4774	109.360199401	10.0.2.15	3.120.68.56	MQTT	157	Connect Command
4803	109.406567313	10.0.2.15	3.120.68.56	MQTT	124	Connect Command
5373	119.355984565	10.0.2.15	18.185.199.22	MQTT	125	Connect Command
6042	122.434076572	10.0.2.15	18.185.199.22	MQTT	125	Connect Command
6083	122.443709707	10.0.2.15	3.120.68.56	MQTT	123	Connect Command
6464	127.440668823	10.0.2.15	3.120.68.56	MQTT	135	Connect Command
6485	127.446603554	10.0.2.15	18.185.199.22	MQTT	121	Connect Command
7408	160.486785885	10.0.2.15	3.120.68.56	MQTT	124	Connect Command

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

- Pub messages with QoS == 1: `mqtt.msgtype == 3 && mqtt.qos == 1` [124]
- Puback messages: `mqtt.msgtype == 4` [74]

We can calculate the messages without ACK with the difference **124 - 74 = 50.**

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

We can see that both `mqtt.willmsg` and `mqtt.willmsg matches "error:.*"` find 56 packets: this means that all last will messages that are eventually sent will contain "error" as text. Now, we can use the following filter to match those with QoS == 0 only:

```
mqtt.msg matches "error:.*" && mqtt.qos == 0
```

This returns a single packet at No. 4164, which matches the error code in the last will message at packet No. 3619, where the message is "*error: tvmgvjml*". Now all that is left to do is actually check that this is a last will message and not a regular one. If we remove the filters, we notice that between messages 4097 and 4214 there are a bunch of TCP disconnects and 3 publishes, all starting with "error" (last will publishes, where one is with QoS == 0 and two with QoS == 1). This is precisely the situation where will messages are fired.

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

The client with the given ID connects from IP 10.0.2.15:58313 to the broker at IP 5.196.95.208:1883 (test.mosquito.org, only address) at frame No. 964. This filter

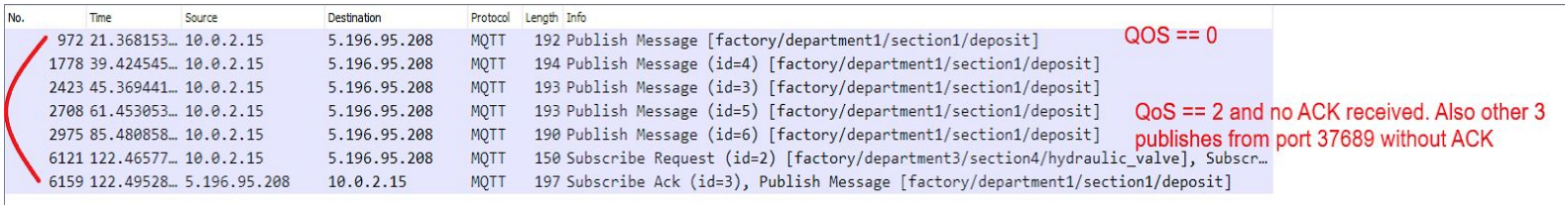
```
ip.addr == 5.196.95.208 && mqtt && tcp.srcport == 58313
```

shows that the user publishes two messages (972 and 2473) on the topic *factory/department1/section1/deposit*, the first being QoS == 0 and the latter with QoS == 2. Luckily for us, the activity on this topic is quite low so the following filter is sufficient to provide us the answer:

```
ip.addr == 5.196.95.208 && mqtt.topic ==
"factory/department1/section1/deposit"
```

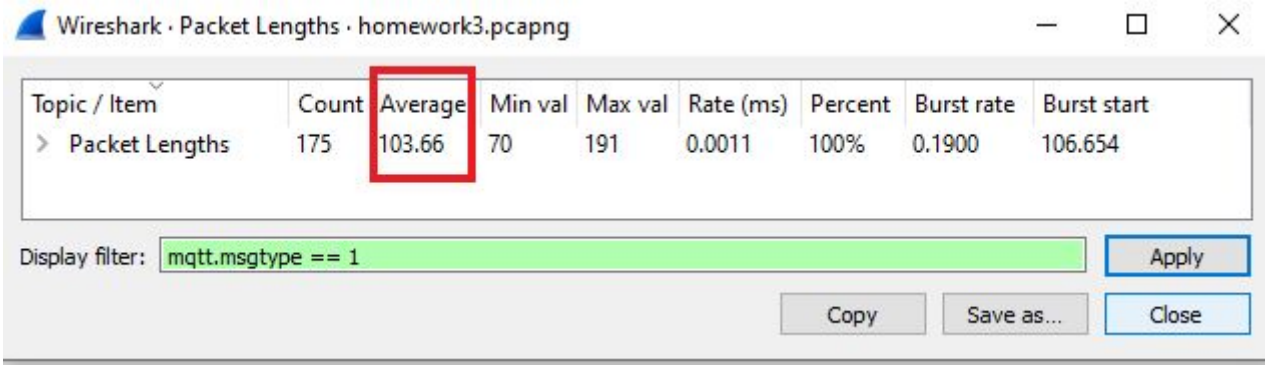
There is a single subscription at 6121, which is ACKed at 6159: by comparing the content of the message, we can see that it matches the one published at 972, which had QoS == 0, **meaning that the publish message with QoS == 2 at frame 2473**

was never delivered to the subscriber. This is not a surprise, since by inspecting the traffic we see that no ACK was received by the publishing client from the broker.



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

WireShark can average the packet sizes with the tool provided in Statistics > Packet Lengths. The average is 103.66 bytes among all the 175 connection messages sent.



Messages have different sizes because the connection message contains optional values (such as the last will message), which increase the packet size.

10. <u>Why there aren't any REQ/RESP pings in the pcap?</u>

This is because all the clients keep having connections with the broker before the keepalive timer expires, thus there is no need for the client to ping it.

Repository link and contacts

The project repository can be found at the following link:
https://github.com/NonSvizzero/IoT2020.
Team members:

- Giuseppe Maria Fiorentino (10590418, giuseppemaria.fiorentino@mail.polimi.it)
- Riccardo Novic (10496965, riccardo.novic@mail.polimi.it).
- Raffaele Zenga (10611699, raffaele.zenga@mail.polimi.it)