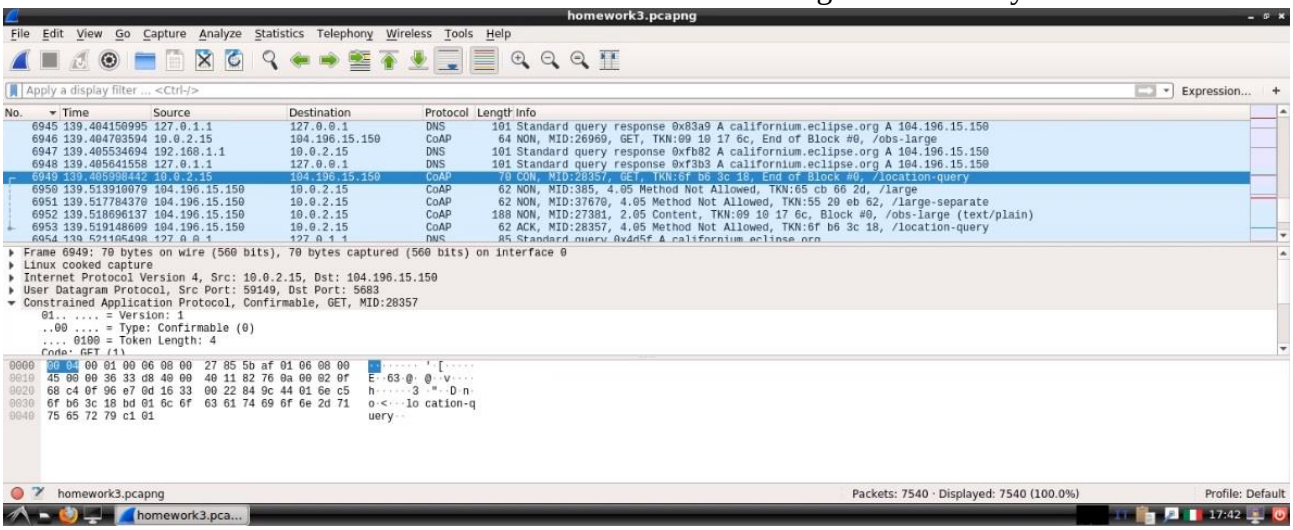
Home Challenge #3 Internet of Things 2019/20

Armenante Valerio 10691549

Di Salvo Dario 10687968

1) We used the query “coap.mid == 3978 || coap.mid == 22636” to find the two messages. The message with MID = 3978 is a confirmable type, indeed the ACK is performed. The message with the MID = 22638 is a non confirmable type. We noticed that the destination port is the same while the source port is different.

2) Yes, we have manually searched the message No.6949 and as it is possible to see from the image below, since it is a confirmable type message then there is the ACK message No.6953 with the same MID = 28357 to confirm that the message was correctly received.



3) There are no replies of confirmable type and result code content that are received by the server localhost, indeed we used the query on pcap “coap.code==69 && coap.type==0” and there were no results displayed.

4) We used the query “mqtt.username contains jane” to find the associated TCP ports. Then we submitted the “mqtt.topic contains factory/department* && tcp.sourceport==NUMBEROFTCPPORT && mqtt.msgtype == 3” , where:

- *mqtt.topic contains factory/department**, the * is not a wildcard but is the changing number to refer to a different department (department1, department2,…). We didn’t used the factory/department*/+ cause Wireshark filters is not able to recognized the + as wildcard, but as regex.
- *tcp.sourceport == NUMBEROFTCPPORT,* the NUMBEROFTCPPORT is a set of TCP ports that we found associated to client with username jane that changing.
- *mqtt.msgtype == 3* , the message type is publish.

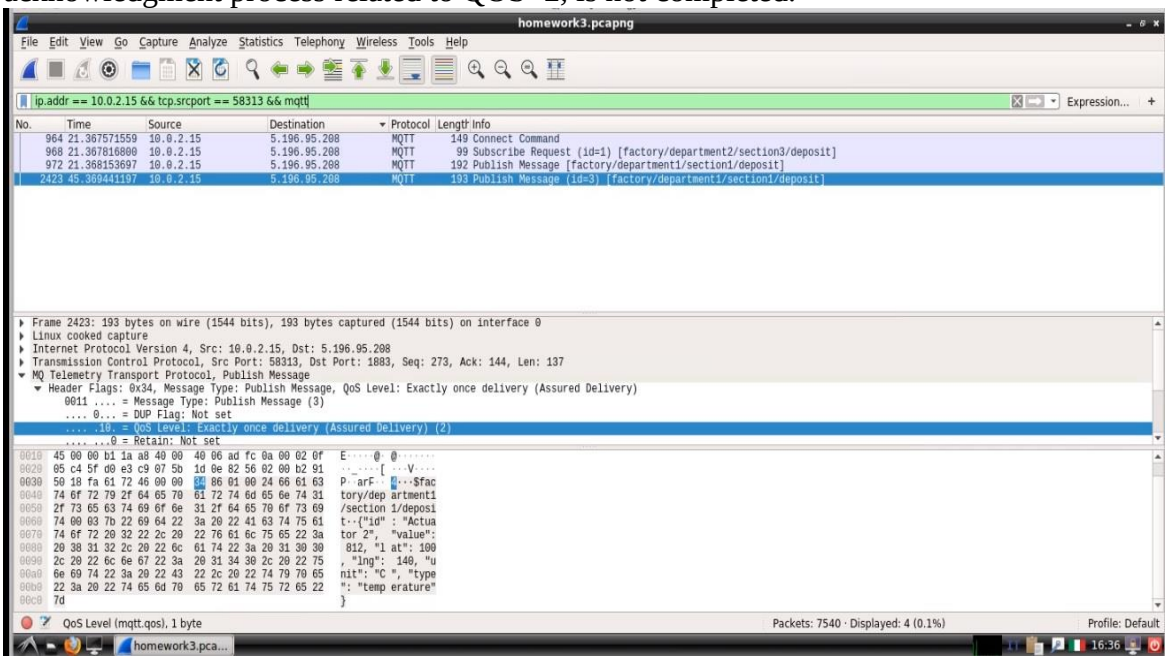
And we found one message in department1 with “tcp.srcport == 50985”, in department2 we found with “tcp.srcport == 4281” two messages and three messages with the “tcp.srcport == 40989”. In conclusion we have found then six messages associated to client with username jane.

5) First of all we have seen that the broker "hivemq" is associated to two addresses that are, 3.120.68.56 and 18.185.199.22 . After that we used the query *"mqtt.msgtype ==1 && (ip.addr == 3.120.68.56 || ip.addr == 18.185.199.22) && mqtt.willmsg"* to find the presence of a will message for each clients. We discovered 9 different client IDs among which we found empty client ID values. The meaning of empty client ID value is that in MQTT 3.1.1 can be sent an empty Client ID, if a state of client is not needed to be held by the broker.

6) There are 2 publish packets with QOS == 1, that don't receive ACK. We found them using the query "mqtt.qos == 1 && mqtt.msgtype == 3 && mqtt.dupflag == 1". We used the DUP flag cause it is indicates the messages that is a duplicate and was resent because the intended recipient (client or broker) did not acknowledge the original message. This is only relevant for QoS greater than zero.

7) We found only one delivered message with QOS == 0 through query "mqtt.qos ==0"(at most one delivered fire and forget) cause will message contains error and we used a "trick" to find in a easiest way the packet using "mqtt.msg contains error".

8) We have seen that client "4m3DWYzWr40pce6OaBQAfk" is associated to the address 10.0.2.15 and TCP port 58313. This client sends two publish messages, only one with QOS>0 that is 2 using the query "ip.addr == 10.0.2.15 && tcp.srcport == 58313 && mqtt.msgtype == 3 && mqtt.qos !=0", and with MID = 3. We have seen that the client received the related PUBREC, verified through the query "ip.addr == 10.0.2.15 && tcp.dstport == 58313 && mqtt.msgtype ==5" referred to this MID = 3 from the broker. After that we checked for the PUBREL and as is shown in the image below, the client didn't send any PUBREL message to the broker. As consequence of that, no PUBCOMP message is sent by broker to client. In conclusion, the packet is correctly delivered to the broker, but we notice that no subscribers are related to this client so it is improper to say that the packets are correctly delivered to subscribers. Moreover, in the second part of the acknowledgment process related to QOS=2, is not completed.



9) We used the query “mqtt.ver == 5 && mqtt.msgtype == 1” to find the relative connect messages using mqttv5 protocol that are 63. So we computed the average message length that is 30.2 . The size are different cause the *connect flags* field can be set in a different ways, and there is also the *payload* field.

MQTT Connect Message Structure

Connect Clean Session True Client ID =PYTON1

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Meaning	Header	Remaining Length	Length of prtocol name		Protocol Name +Version					Connect Flags	Keep Alive		Length								
Hex	0x10	0x13	0x0	0x4	0x4d	0x51	0x54	0x54	0x4	0x2	0x0	0x3c	0x0	0x7	0x70	0x79	0x74	0x68	0x6F	6E	0x31
Ascii		19		4	M	Q	T	T	4			60		7	P	Y	T	H	O	N	1

Connect Flags

User name flag =	bit 7
Password Flag =	bit 6
Will Retain =	bit 5
Will QOS =	bit 5
Will QOS =	bit 4
Will Flag =	bit 2
Clean Session =	bit 1
Reserved =	bit 0

10) Since no client waits until the *keep-alive* timer is over, then no PINGREQ and consequently PINGRESP are sent. The clients that let pass too much time will be disconnected directly through disconnected type messages.