Internet of Things - Second Challenge

Francesco Pastore 10629332

April 7, 2024

To answer the questions I used the Python library Scapy together with Wireshark.

I have reported the answers to all the questions with the code used and important screenshots of the Wireshark filters.

My assumption was that each client was assigned a unique unregistered port, whereas the servers were running only on default ports.

The Jupyter notebook is also available at the following link:

https://gist.github.com/pastore-francesco/fc709ebeb633ccab6b3d500551c418e6

[1]:
```
import datetime

from scapy.all import rdpcap, bind_bottom_up, bind_layers
from scapy.contrib.coap import CoAP
from scapy.layers.inet import IP, UDP, TCP
from scapy.layers.dns import DNSRR
from scapy.contrib.mqtt import MQTT, MQTTSubscribe
from scapy.contrib.mqtt import MQTTTopicQOS, MQTTConnect, MQTTPublish
from scapy.contrib.mqttsn import MQTTSN, MQTTSNPublish
```

[2]:
```
# Overwrite ports configuration for CoAP
bind_layers(UDP, CoAP, sport=5683)
bind_layers(UDP, CoAP, dport=5683)
```

[3]:
```
# Overwrite ports configuration for MQTT
bind_layers(TCP, MQTT, sport=1883)
bind_layers(TCP, MQTT, dport=1883)
```

[4]:
```
# Overwrite ports configuration for MQTT-SN
bind_bottom_up(UDP, MQTTSN, sport=1885)
bind_bottom_up(UDP, MQTTSN, dport=1885)
bind_layers(UDP, MQTTSN, dport=1885, sport=1885)
```

[5]:
```
packets = rdpcap("./challenge2.pcapng")
```

```
WARNING: Invalid Option Delta or Length
WARNING: Invalid Option Delta or Length
WARNING: Invalid Option Length or Delta 15
```

```
WARNING: Invalid Option Length or Delta 15
WARNING: more Invalid Option Delta or Length
WARNING: more Invalid Option Length or Delta 15
```

1 Questions

1.1 Question 1a

How many different CoAP clients sent a GET request to a temperature resource (…/temperature)?

Answer: 8

The answer is given by filtering the packets that have the GET method and the URI path ending in "/temperature".

After filtering, I counted the number of unique ports that sent the packets.

[6]:
```python
# List of clients needed to avoid duplicates
clients_1 = []
count = 0
for p in packets:
    if (
        p.haslayer(CoAP) # Only CoAP Packets
        and p[CoAP].code == 1 # Only GET requests
    ):
        # Check if the Uri-Path contains 'temperature'
        found = False
        for option in p[CoAP].options:
            if option[0] != 'Uri-Path':
                continue

            # Uri-Path is split into segments
            # Check if at least one segment is 'temperature'
            # There is no temperature/... but only .../temperature
            # So we don't need to check the whole string
            if option[1] == b'temperature':
                found = True
                break

        if not found:
            continue

        # Each client has a unique port
        client = p[UDP].sport
        if client in clients_1:
            continue

        clients_1.append(client)
```

```
        count += 1

count
```

[6]: 8

1.2 Question 1b

For each of the clients found in 1a, write the MID of the longest CoAP response (any response) received by the client

Answer: 25, 26, 29, 63, 7365, 10589, 12426, 47747

The answer is given by filtering the packets, looking only for CoAP responses related to the previous clients.

I then checked the length of each CoAP response and stored its MID.

I considered the CoAP length and not the packet length because it was asked for "the longest CoAP response" not the longest packet.

[7]:
```
# Prepare the list of responses for each client
# We need to store the length and the MID to find the largest response
responses = {}
for client in clients_1:
    responses[client] = {'length': 0, 'MID': 0}

for p in packets:
    if (
        p.haslayer(CoAP) # Only CoAP Packets
        and p[CoAP].code >= 64 # Only responses
        and p.haslayer(UDP) # Only UDP Packets
    ):
        # Check if the response is for one of the previous clients
        client = p[UDP].dport
        if client not in clients_1:
            continue

        # Check if the length is greater than the previous one
        length = p[UDP].len
        if length > responses[client]['length']:
            responses[client]['length'] = length
            responses[client]['MID'] = p[CoAP].msg_id

mids = []
for _, v in responses.items():
    mids.append(v['MID'])

mids.sort()
```

```
mids
```

[7]: [25, 26, 29, 63, 7365, 10589, 12426, 47747]

1.3 Question 2a

How many CoAP POST requests directed to the "coap.me" server did NOT produce a successful result?

Answer: 18

The answer is given by three scans of the packets:

1. Check the DNS packet responses for the coap.me server to find all possible IPs.
2. Find all POST requests to the coap.me server and store their mids and tokens using previous IPs as destination.
3. Find all the responses from the coap.me server with the previous tokens or mids and check their status code.

Based on what I found, CoAP requests and responses should be matched through the token.

https://en.wikipedia.org/wiki/Constrained_Application_Protocol#Token

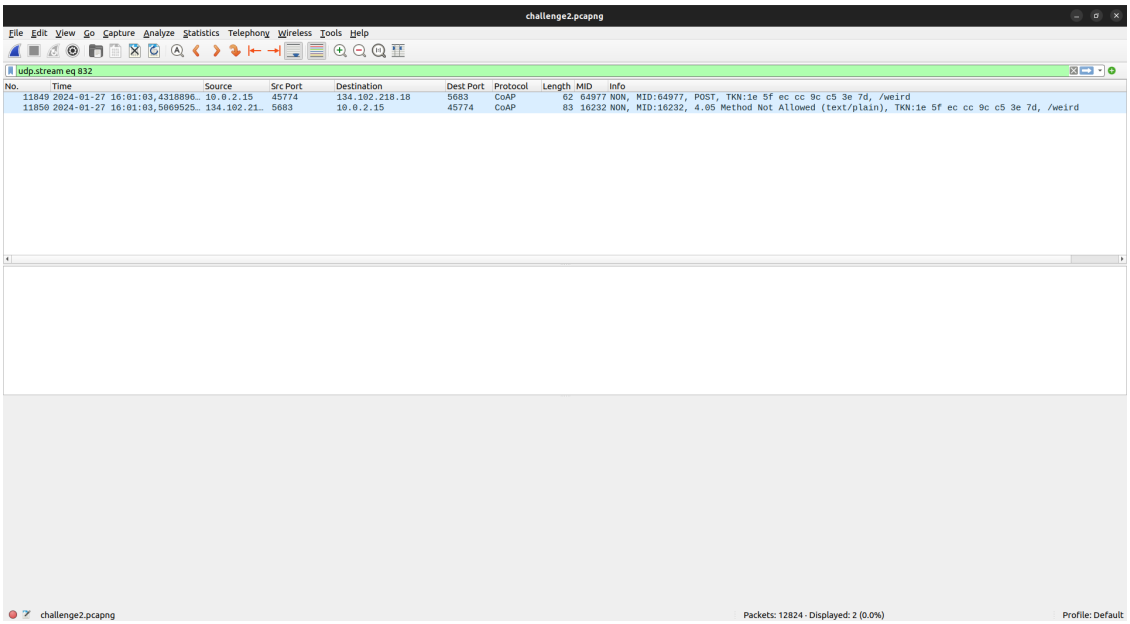
However, looking with Wireshark I found out that multipart requests have empty tokens and instead the MID is constant with the request.

For this reason to match requests and responses I used both the tokens and the mids.

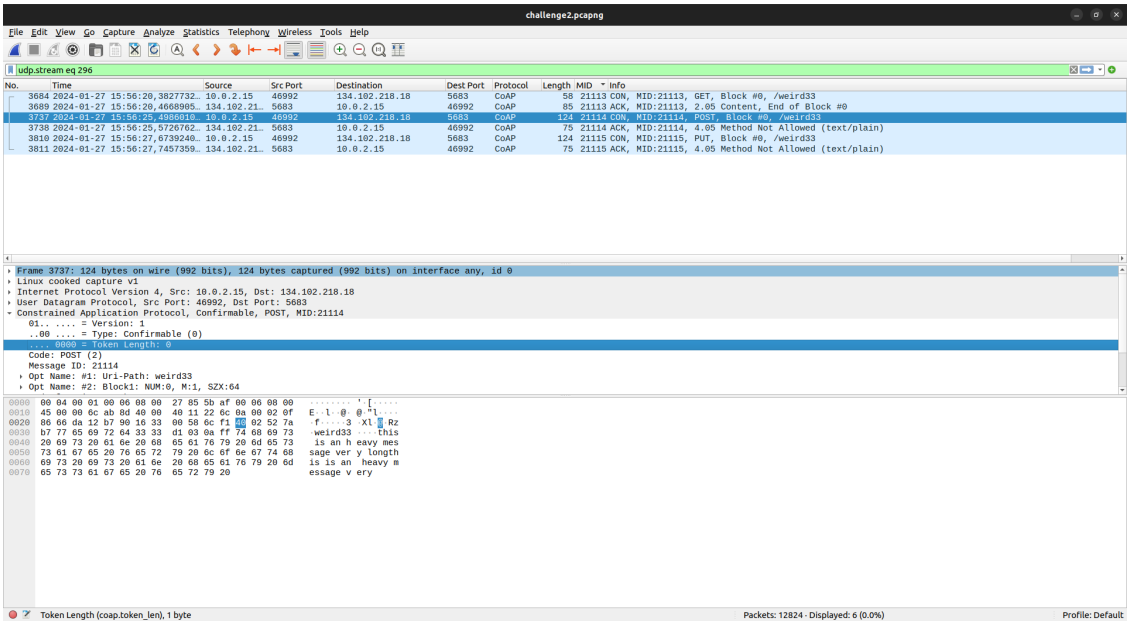
Also, there is one request with no response and because it was asked for a not successful result and not for a wrong result, I count it too.

This request was sent to the path /hello.

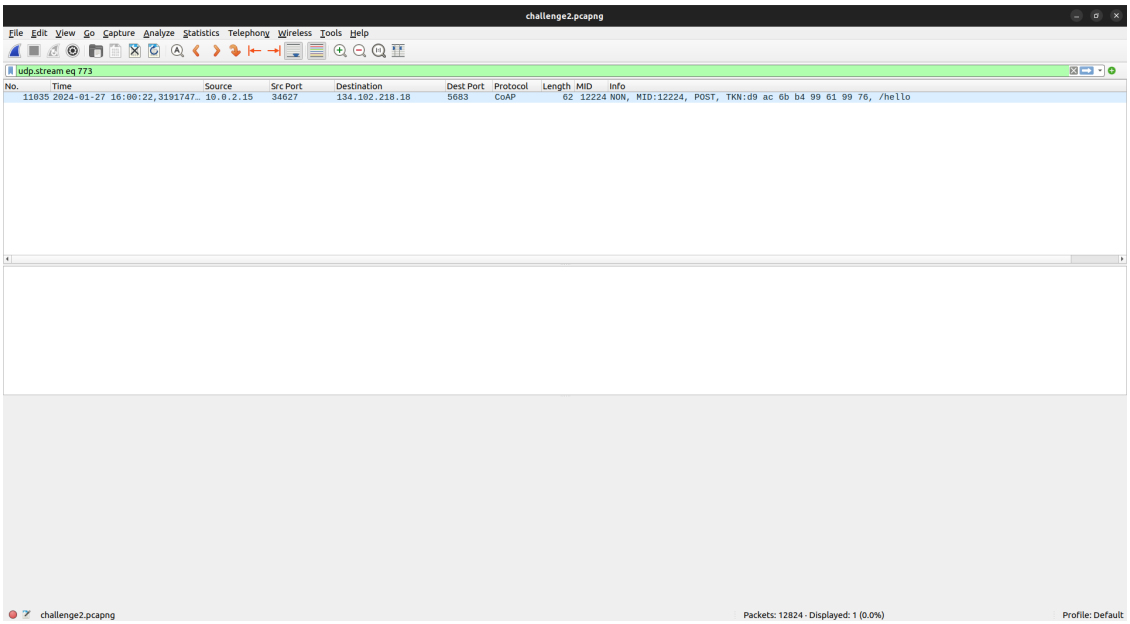
In a standard POST request with a bad response we can see that the token remains the same while the MID changes (next image).



In the POST request inside a multipart request we can see that the token is empty but the MID is the same (next image).



In the POST request to /hello we can see that no answer is found by matching the token, the MID, or the UDP stream (next image).



[8]:
```python
# Find IPs of the coap.me server by looking to the DNS requests
coapme = []
for p in packets:
    if (p.haslayer(DNSRR) # Only DNS Resource Record packets
        and p[DNSRR].type == 1 # Only records of type A
        # Only records for the coap.me server
        and p[DNSRR].rrname == b'coap.me.'
    ):
        # Save the IP of the coap.me server
        coapme.append(p[DNSRR].rdata)

# Only unique IPs
coapme = set(coapme)

print("coap.me IPs:", coapme)
```

```
coap.me IPs: {'134.102.218.18'}
```

[9]:
```python
# Find all CoAP requests to coap.me server saving the tokens
# that will be used to find the responses
requests_tokens_2 = []
requests_mids_2 = []
for p in packets:
    if (
        p.haslayer(CoAP) # Only CoAP Packets
        and p[CoAP].code == 2 # Only POST requests
        and p[IP].dst in coapme # Only requests to the coap.me server
    ):
        # Block requests have empty tokens
        if p[CoAP].token != b'':
```

```
            requests_tokens_2.append(p[CoAP].token)

        requests_mids_2.append(p[CoAP].msg_id)

print("Found %d tokens and %d mids of CoAP requests"
    % (len(requests_tokens_2), len(requests_mids_2)))
```

```
Found 27 tokens and 34 mids of CoAP requests
```

[10]:
```
# Find all CoAP responses to the previous requests
# searching only for the ones with a bad status code
responses_count_2 = 0
bad_responses_tokens_2 = []
bad_responses_mids_2 = []
for p in packets:
    if (
        p.haslayer(CoAP)  # Only CoAP Packets
        and p.haslayer(UDP)  # Only UDP Packets
        and p[CoAP].code > 64  # Only responses
        and (  # Only responses to the previous requests
            p[CoAP].token in requests_tokens_2  # Check token
            or p[CoAP].msg_id in requests_mids_2  # Check mid
        )
    ):
        # Increment the total responses count
        responses_count_2 += 1

        # Save bad responses
        if p[CoAP].code > 127:
            bad_responses_tokens_2.append(p[CoAP].token)
            bad_responses_mids_2.append(p[CoAP].msg_id)

print("Found %d responses of which %d are bad"
    % (responses_count_2, len(bad_responses_mids_2)))
```

```
Found 33 responses of which 17 are bad
```

1.4 Question 2b

How many requests from 2a are directed to a “weird” resource? (resources like /weirdXX)?

Answer: 8

The answer is given by filtering the packets, looking only for CoAP requests related to the previous clients.

I then checked the uri_path of each CoAP request and stored the number of requests with the string “weird” in the uri_path.

[11]:
```
count = 0
for p in packets:
    if (
        p.haslayer(CoAP) # Only CoAP Packets
        and (  # Only previous requests
            p[CoAP].token in requests_tokens_2  # Check token
            or p[CoAP].msg_id in requests_mids_2  # Check mid
        )
    ):
        # Check if the request has a 'weird' segment in the Uri-Path
        found = False
        for option in p[CoAP].options:
            if option[0] != 'Uri-Path':
                continue

            if b'weird' in option[1]:
                found = True
                break

        # If no 'weird' segment is found, continue
        if not found:
            continue

        count += 1

count
```

[11]: 8

1.5 Question 3a

How many MQTT Publish messages with qos=2 are RECEIVED by the clients running in the machine capturing the traffic?

Answer: 2

The answer is given by filtering the packets, looking only for MQTT Publish (type 3) messages with QoS set to 2.

The destination port must be different then the default one (1883) to find received messages.

[12]:
```
# Save clients and topics for next questions
clients_3 = []
topics_3 = []

count = 0
for p in packets:
    if (
        p.haslayer(MQTT) # Only MQTT packets
```

```
        and p[MQTT].type == 3 # Only PUBLISH packets
        and p[MQTT].QOS == 2 # Only QoS 2 packets
        and p[TCP].dport != 1883 # Only packets received by clients
    ):
        count += 1
        clients_3.append(p[TCP].dport)
        topics_3.append(p[MQTT].topic)

count
```

[12]: 2

1.6 Question 3b

How many clients are involved in the messages found in 3a?

Answer: 1

The answer is given by counting the number of unique clients (ports) that were found in the previous question.

[13]:
```
clients_3 = set(clients_3)
len(clients_3)
```

[13]: 1

1.7 Question 3c

What are the MQTT Message identifiers (ID) of the subscribe requests that let the client receive the messages found in 3a?

Answer: 15

The answer is given by filtering the packets, looking only for MQTT Subscribe (type 8) messages that are related to the previous clients.

Each subscribe message then must be checked to see if it is related to the previous topics.

In particular the topics of the previous publish messages are:

- hospital/facility2/section0
- hospital/facility2/room4/temperature

Because there is only one client and there are many combinations to check I used Wireshark to find the answer.

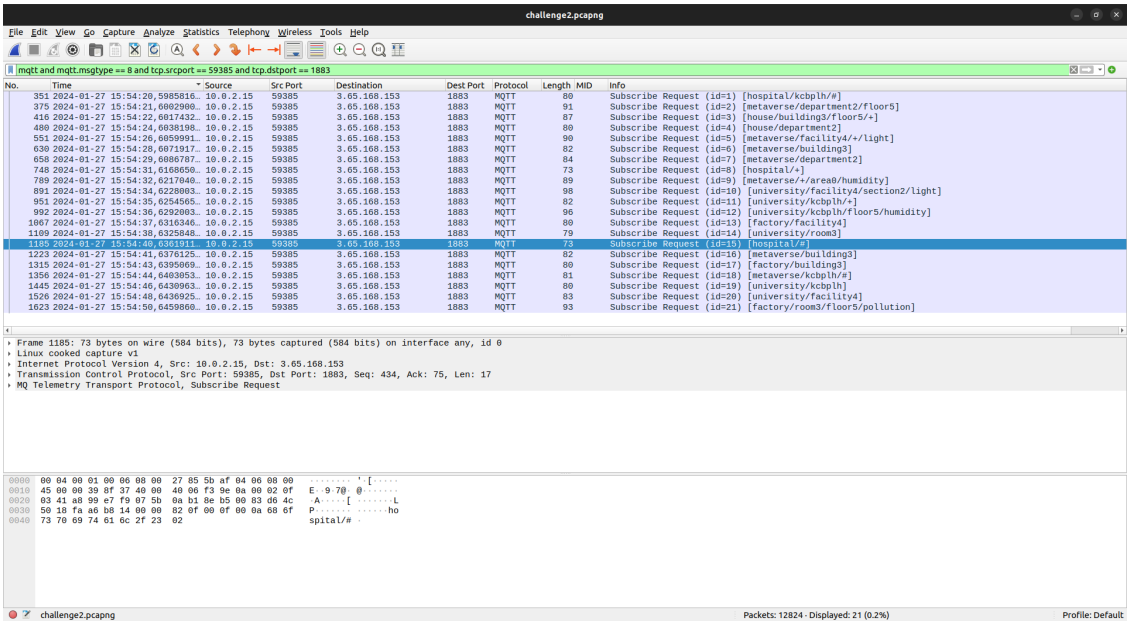
`mqtt and mqtt.msgtype == 8 and tcp.srcport == 59385 and tcp.dstport == 1883`

Possible valid combinations:

- hospital/#
- hospital/facility2/#
- hospital/facility2/room4/+

- hospital/+/room4/temperature
- hospital/facility2/section0
- hospital/+/section0
- ...

In the end, only the subscribe message with the topic hospital/# and ID 15 was found.



[14]:
```
topics_3
```

[14]: `[b'hospital/facility2/section0', b'hospital/facility2/room4/temperature']`

[15]:
```
clients_3
```

[15]: `{59385}`

1.8 Question 4a

How many MQTT clients sent a subscribe message to a public broker using at least one wildcard?

Answer: 4

The answer is given by filtering the packets, looking only for MQTT Subscribe (type 8) messages that are related to a public broker.

This is done by checking the destination IP of the packets to be different from the local IP.

Then each subscribe message must be checked to see if it contains a wildcard '+' or '#'.

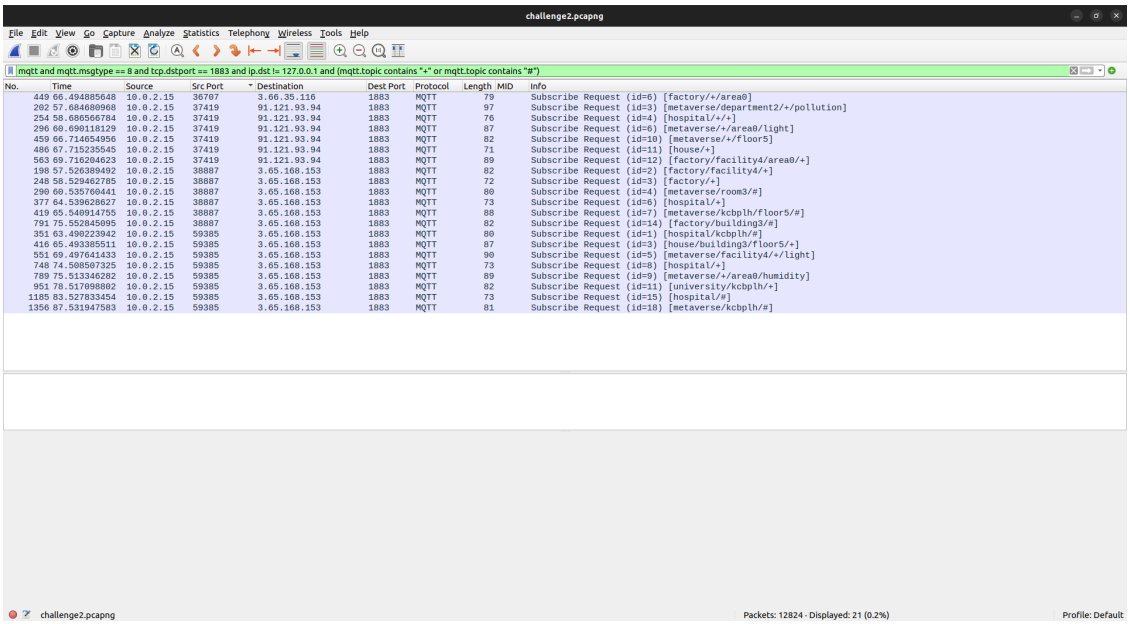
[31]:
```
# Save topics for next questions
topics_4 = []
clients_4 = []
for p in packets:
    if (
```

```
        p.haslayer(MQTT) # Only MQTT packets
        and p[MQTT].type == 8 # Only SUBSCRIBE packets
        and p[TCP].dport == 1883 # Only packets to the MQTT Broker
        and p[IP].dst != "127.0.0.1" # Only packets not to localhost
        and p.haslayer(MQTTTopicQOS) # Only packets with a list of topics
    ):
        # Check if the topic has at least one wildcard
        found = False
        for topic in p[MQTTSubscribe].topics:
            t = topic[MQTTTopicQOS].topic
            if b'+' in t or b'#' in t:
                # In some topics the length of the topics
                # and the topic itself are different
                # Probably there is something wrong with the parsing
                # This is an issue for the next questions
                # l = topic[MQTTTopicQOS].length
                # print(t, l, len(t) == l)
                topics_4.append(t)
                found = True
                break

        if not found:
            continue

        clients_4.append(p[TCP].sport)

# Only unique clients
clients_4 = set(clients_4)

len(clients_4)
```

[31]: 4

If we use Wireshark we could find the same result by applying the following filter:

```
mqtt and mqtt.msgtype == 8 and tcp.dstport == 1883 and ip.dst != 127.0.0.1
    and (mqtt.topic contains "+" or mqtt.topic contains "#")
```



1.9 Question 4b

Considering clients found in 4a, how many of them WOULD receive a publish message directed to the topic: “metaverse/facility4/area0/light”?

Answer: 2

Only the subscribe requests with topic `metaverse/facility4/+/light` and `metaverse/+/area0/light` from two different clients will receive a publish message about `metaverse/facility4/area0/light`

There is a problem in Scapy for parsing the topics, so the answer was found by using Wireshark.

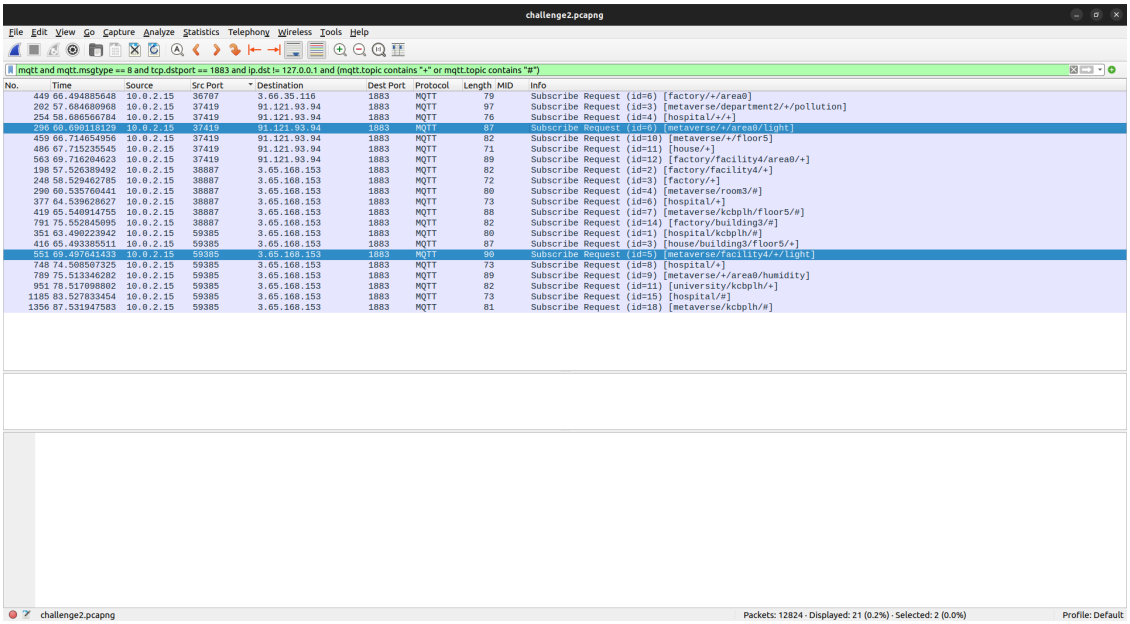
[17]:
```
# List of topics from previous question
# Some of them are malformed with not printable characters
for t in topics_4:
    print(t)
```

```
b'factory/facility4/+'
b'taverse/department2/+/pollution\x01'
b'factory/+'
b'spital/+/+\x00'
b'metaverse/room3/#'
b'taverse/+/area0/light\x01'
b'hospital/kcbplh/#'
b'hospital/+'
b'house/building3/floor5/+'
b'metaverse/kcbplh/floor5/#'
b'ctory/+/area0\x00'
b'taverse/+/floor5\x01'
b'use/+\x02'
```

```
b'metaverse/facility4/+/light'
b'ctory/facility4/area0/+\x02'
b'hospital/+'
b'metaverse/+/area0/humidity'
b'factory/building3/#'
b'university/kcbplh/+'
b'hospital/#'
b'metaverse/kcbplh/#'
```

Using the same filter of the previous question, we can see that there are only two possible topics that satisfy the request:

- metaverse/facility4/+/light
- metaverse/+/area0/light



1.10 Question 5

How many MQTT ACK messages in total are received by clients who connected to brokers specifying a client identifier shorter than 15 bytes and using MQTT version 3.1.1?

Answer: 233

The answer is given by first finding all the clients that have sent a CONNECT message with a client id shorter than 15 bytes and MQTT version 3.1.1

After that we have to directly count all the ACK messages that are received by these clients.

I didn't consider if the clients have done multiple connections with different ids.

Instead, I interpreted it as if at least one connection was done as required than count all the ACK messages for these clients.

For the ACK messages I considered the following types:

msg	type
CONNACK	2
PUBACK	4
PUBREC	5
PUBREL	6
PUBCOMP	7
SUBACK	9
UNSUBACK	11

[18]:
```
# Save clients between first and second filter
clients_5 = []

# We need to find first all the clients that have connected to a MQTT broker
# with a client id that is shorter than 15 bytes and using MQTT version 3.1.1.
for p in packets:
    if (
        p.haslayer(MQTT) # Only MQTT packets
        and p.haslayer(MQTTConnect) # Only connect packets
        and p[MQTTConnect].protolevel == 4 # Check MQTT version to be 3.1.1 (4)
        and len(p[MQTTConnect].clientId) < 15
    ):
        clients_5.append(p[TCP].sport)

# Only unique clients
clients_5 = set(clients_5)

print("Found %d clients" % len(clients_5))
```

```
Found 2 clients
```

[19]:
```
# Once we have the right clients, we can simply count the number
# of ACK packets that are sent by a MQTT Broker to these clients
count = 0
for p in packets:
    if (
        p.haslayer(MQTT) # Only MQTT packets
        and p[MQTT].type in [2, 4, 5, 6, 7, 9, 11] # Only ACK packets
        and p[TCP].dport in clients_5 # Only packets to the clients
    ):
        count += 1

count
```

[19]: 233

1.11 Question 6a

How many MQTT subscribe requests with message ID=1 are directed to the HiveMQ broker?

Answer: 3

The answer is given by first finding all the IPs of the HiveMQ broker.

In order to find them, we have to check all DNS resource records for "broker.hivemq.com" and saving the related IPs.

After that, we have to find all the MQTT subscribe requests with message ID=1 and directed to one of the IPs found before.

[20]:
```
# Find IPs of the HiveMQ server by looking to the DNS requests
hivemq = []
for p in packets:
    if (
        p.haslayer(DNSRR) # Only DNS Resource Record packets
        and p[DNSRR].type == 1 # Only records of type A
        # Only records for the HiveMQ server
        and p[DNSRR].rrname == b'broker.hivemq.com.'
    ):
        # Save the IP of the coap.me server
        hivemq.append(p[DNSRR].rdata)

# Only unique IPs
hivemq = set(hivemq)

print("HiveMQ IPs:", hivemq)
```

```
HiveMQ IPs: {'3.65.168.153', '3.66.35.116'}
```

[21]:
```
# Save topics and clients for next questions
topics_6 = []
clients_6 = []

# Search for subscribe messages to the HiveMQ server
# with message id 1 using previous IPs
count = 0
for p in packets:
    if (
        p.haslayer(MQTTSubscribe) # Only MQTT SUBSCRIBE packets
        and p[MQTTSubscribe].msgid == 1 # Only message id 1
        and p[IP].dst in hivemq # Only packets to the HiveMQ server
    ):
        count += 1
        topics_6.append(p[MQTTSubscribe].topics)
        clients_6.append(p[TCP].sport)
```

```
count
```

[21]: 3

1.12 Question 6b

How many publish messages are received by the clients thanks to the subscribe requests found in 6a?

Answer: 0

The answer is given by finding all the MQTT publish messages with the same topic of the subscribe requests found in 6a.

The subscribe requests have the following topics:

- hospital/department5/room4
- hospital/kcbplh/#

The publish requests have the following topics:

- hospital/department5/room4
- factory/room2
- factory/department5
- factory/building5
- factory/department5
- factory/room2
- hospital/facility2/section0
- hospital/facility2/room4/temperature
- hospital/room2/floor1
- factory/facility2

There is no match between the topics, so no publish message is received by the clients due to the subscription requests found in 6a.

[22]:
```
# Subscribe messages from previous question
for t in topics_6:
    print(t[0][MQTTTopicQOS].topic.decode())
```

```
university/department2/floor5

hospital/kcbplh/#
```

[23]:
```
# Print topics of all publish messages to the previous clients
for packet in packets:
    if (
        packet.haslayer(MQTTPublish)  # Only MQTT Publish packets
        and packet[TCP].dport in clients_6  # Only previous clients
    ):
        print(packet[MQTTPublish].topic.decode())
```

```
hospital/department5/room4
factory/room2
factory/department5
factory/building5
factory/department5
factory/room2
hospital/facility2/section0
hospital/facility2/room4/temperature
hospital/room2/floor1
factory/facility2
```

1.13 Question 7a

How many MQTT-SN (on port 1885) publish messages sent after the hour 3.59PM (Milan Time) are directed to topic 6?

Answer: 3

The answer is given by filtering the packets, looking only for MQTT-SN Publish (type 3) messages that are related to the topic 6 and sent after the hour 3.59PM (Milan Time).

[24]:
```
count = 0
for packet in packets:
    packet_time = datetime.datetime.fromtimestamp(int(packet.time)).time()
    if (packet.haslayer(UDP) # Only UDP packets
        and packet.haslayer(MQTTSNPublish) # Only MQTT-SN Publish packets
        and packet_time > datetime.time(15, 59) # Only packets after 15:59
        and packet[MQTTSNPublish].tid == 6 # Only packets with tid 6
    ):
        count += 1

count
```

[24]:
```
3
```

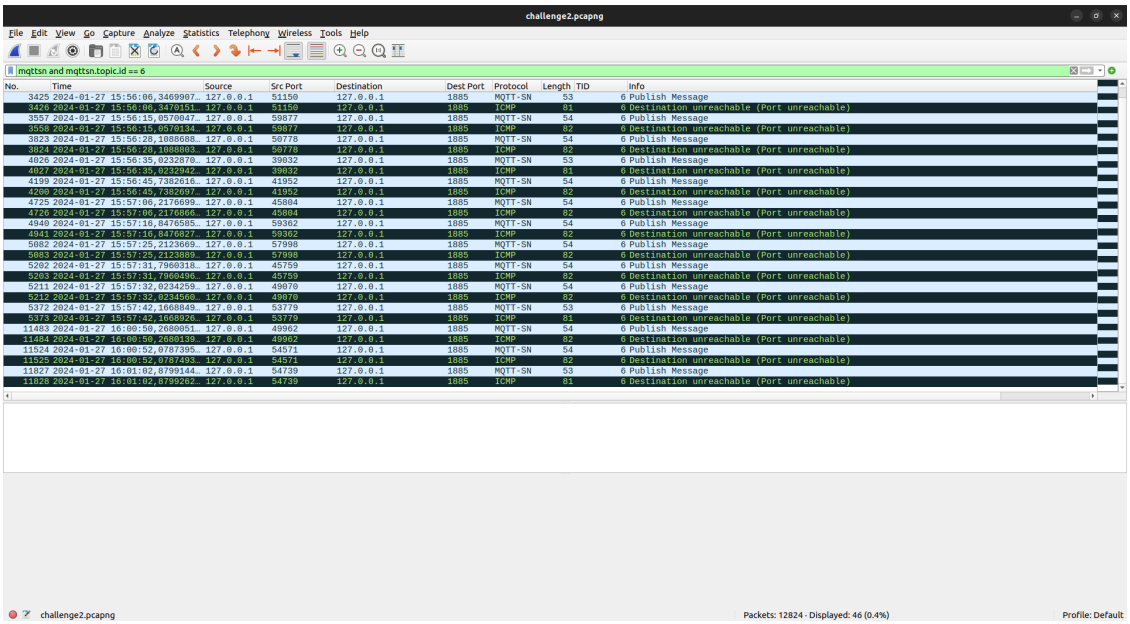
1.14 Question 7b

Explain possible reasons why messages in 7a are not handled by the server

Answer

Each message in 7a receive a ICMP Destination Unreachable response indicating possible issues beyond MQTT-SN itself.

A firewall might be blocking the port necessary for communication, the server may not be running, or it could be accessible via a different port.


challenge2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
mqttsn and mqttsn.topic.id == 6
No. Time Source Src Port Destination Dest Port Protocol Length TID Info
3425 2024-01-27 15:56:06,3469907… 127.0.0.1 51150 127.0.0.1 1885 MQTT-SN 53 6 Publish Message
3426 2024-01-27 15:56:06,3470131… 127.0.0.1 51150 127.0.0.1 1885 ICMP 81 6 Destination unreachable (Port unreachable)
3557 2024-01-27 15:56:15,0570047… 127.0.0.1 59077 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
3558 2024-01-27 15:56:15,0570134… 127.0.0.1 59077 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
3823 2024-01-27 15:56:28,1088688… 127.0.0.1 50778 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
3824 2024-01-27 15:56:28,1088803… 127.0.0.1 50778 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
4026 2024-01-27 15:56:35,0232870… 127.0.0.1 39032 127.0.0.1 1885 MQTT-SN 53 6 Publish Message
4027 2024-01-27 15:56:35,0232942… 127.0.0.1 39032 127.0.0.1 1885 ICMP 81 6 Destination unreachable (Port unreachable)
4199 2024-01-27 15:56:45,7382616… 127.0.0.1 41952 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
4200 2024-01-27 15:56:45,7382697… 127.0.0.1 41952 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
4725 2024-01-27 15:57:06,2176699… 127.0.0.1 45804 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
4726 2024-01-27 15:57:06,2176866… 127.0.0.1 45804 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
4940 2024-01-27 15:57:16,0476585… 127.0.0.1 59362 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
4941 2024-01-27 15:57:16,0476827… 127.0.0.1 59362 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
5082 2024-01-27 15:57:25,2123669… 127.0.0.1 57998 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
5083 2024-01-27 15:57:25,2123889… 127.0.0.1 57998 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
5202 2024-01-27 15:57:31,7960318… 127.0.0.1 45759 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
5203 2024-01-27 15:57:31,7960496… 127.0.0.1 45759 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
5211 2024-01-27 15:57:32,0234259… 127.0.0.1 49070 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
5212 2024-01-27 15:57:32,0234560… 127.0.0.1 49070 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
5372 2024-01-27 15:57:42,1668849… 127.0.0.1 53779 127.0.0.1 1885 MQTT-SN 53 6 Publish Message
5373 2024-01-27 15:57:42,1668926… 127.0.0.1 53779 127.0.0.1 1885 ICMP 81 6 Destination unreachable (Port unreachable)
11483 2024-01-27 16:00:50,2000951… 127.0.0.1 49962 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
11484 2024-01-27 16:00:50,2680139… 127.0.0.1 49962 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
11524 2024-01-27 16:00:52,0787395… 127.0.0.1 54571 127.0.0.1 1885 MQTT-SN 54 6 Publish Message
11525 2024-01-27 16:00:52,0787493… 127.0.0.1 54571 127.0.0.1 1885 ICMP 82 6 Destination unreachable (Port unreachable)
11827 2024-01-27 16:01:02,8799144… 127.0.0.1 54739 127.0.0.1 1885 MQTT-SN 53 6 Publish Message
11828 2024-01-27 16:01:02,8799262… 127.0.0.1 54739 127.0.0.1 1885 ICMP 81 6 Destination unreachable (Port unreachable)
challenge2.pcapng
Packets: 12824 · Displayed: 46 (0.4%)
Profile: Default