

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/fc709eb6b633ccab6b3d500551c418e6>

```
[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 MQTTPublish, MQTTConnect, MQTTTopicQOS
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]: # 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).

The image shows a Wireshark capture of a CoAP message. The packet list shows two packets: a CoAP POST request (No. 11849) and a CoAP 4.05 Method Not Allowed response (No. 11850). The selected packet (No. 11849) is expanded to show the CoAP message details. The message is a POST to /weird with a token of 5f ec cc 9c c5 3e 7d. The packet details pane shows the CoAP message structure: Code: POST (2), Message ID: 21113, URI-Path: /weird3, and Options: Block1: NUM=0, M=1, SZX=04.

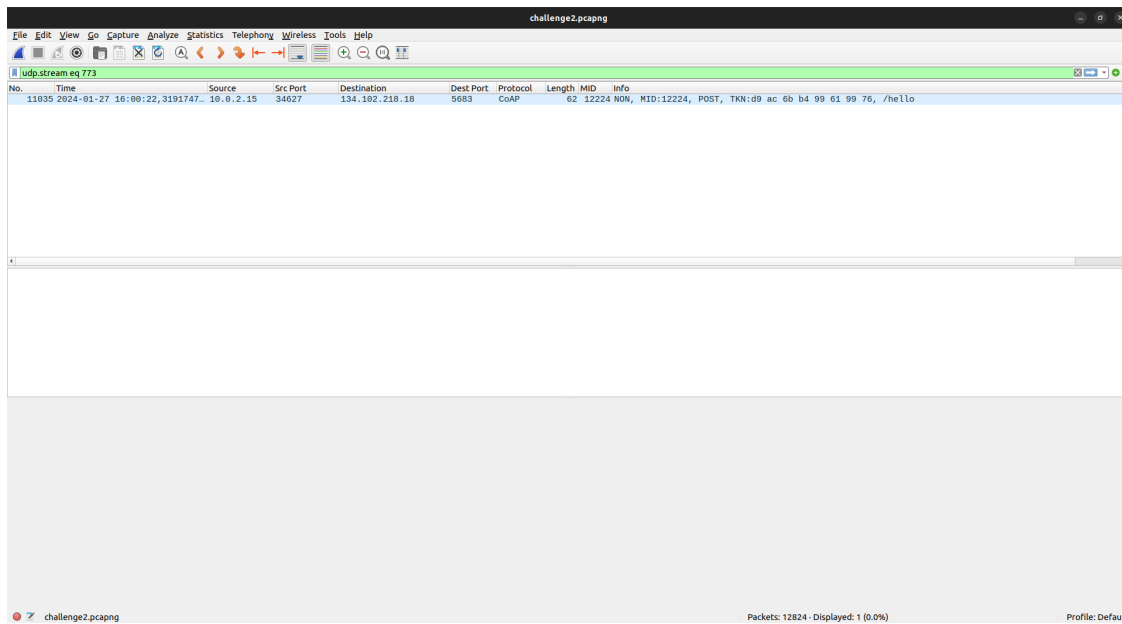
No.	Time	Source	Src Port	Destination	Dest Port	Protocol	Length	MID	Info
11849	2024-01-27 16:01:03.4318896	10.0.2.15	45774	134.102.218.18	5683	CoAP	62	64977	NON, MID:64977, POST, TKN:1e 5f ec cc 9c c5 3e 7d, /weird
11850	2024-01-27 16:01:03.5069525	134.102.21.5683	5683	10.0.2.15	45774	CoAP	83	10232	NON, MID:10232, 4.05 Method Not Allowed (text/plain), TKN:1e 5f ec cc 9c c5 3e 7d, /weird

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

The image shows a Wireshark capture of a CoAP message. The packet list shows several packets, including a CoAP POST request (No. 3684) and a CoAP 4.05 Method Not Allowed response (No. 3738). The selected packet (No. 3684) is expanded to show the CoAP message details. The message is a POST to /weird3 with a token of 5f ec cc 9c c5 3e 7d. The packet details pane shows the CoAP message structure: Code: POST (2), Message ID: 21114, URI-Path: /weird3, and Options: Block1: NUM=0, M=1, SZX=04.

No.	Time	Source	Src Port	Destination	Dest Port	Protocol	Length	MID	Info
3684	2024-01-27 15:56:20.3827732	10.0.2.15	46992	134.102.218.18	5683	CoAP	58	21113	CON, MID:21113, GET, Block #0, /weird33
3688	2024-01-27 15:56:20.4668905	134.102.21.5683	5683	10.0.2.15	46992	CoAP	85	21113	ACK, MID:21113, 2.05 Content, End of Block #0
3738	2024-01-27 15:56:25.5726762	134.102.21.5683	5683	10.0.2.15	46992	CoAP	75	21114	ACK, MID:21114, 4.05 Method Not Allowed (text/plain)

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]: # 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]: # 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.

No.	Time	Source	Src Port	Destination	Dest Port	Protocol	Length	MID	Info
351	2024-01-27 15:54:20,5985816	10.0.2.15	59385	3.65.168.153	1883	MQTT	80	80	Subscribe Request (id=1) [hospital/kcblh/w]
375	2024-01-27 15:54:21,6892986	10.0.2.15	59385	3.65.168.153	1883	MQTT	91	91	Subscribe Request (id=2) [metaverse/department2/floor5]
410	2024-01-27 15:54:22,6917432	10.0.2.15	59385	3.65.168.153	1883	MQTT	87	87	Subscribe Request (id=3) [house/building2/floor5/+]
480	2024-01-27 15:54:24,6838198	10.0.2.15	59385	3.65.168.153	1883	MQTT	80	80	Subscribe Request (id=4) [house/department2]
551	2024-01-27 15:54:26,6895991	10.0.2.15	59385	3.65.168.153	1883	MQTT	90	90	Subscribe Request (id=5) [metaverse/facility/+ /light]
630	2024-01-27 15:54:29,6971917	10.0.2.15	59385	3.65.168.153	1883	MQTT	82	82	Subscribe Request (id=6) [metaverse/buildings]
658	2024-01-27 15:54:29,6886787	10.0.2.15	59385	3.65.168.153	1883	MQTT	84	84	Subscribe Request (id=7) [metaverse/department2]
748	2024-01-27 15:54:31,6106596	10.0.2.15	59385	3.65.168.153	1883	MQTT	73	73	Subscribe Request (id=8) [hospital/+]
789	2024-01-27 15:54:32,6217046	10.0.2.15	59385	3.65.168.153	1883	MQTT	89	89	Subscribe Request (id=9) [metaverse/+ /read/humidity]
891	2024-01-27 15:54:34,6228093	10.0.2.15	59385	3.65.168.153	1883	MQTT	98	98	Subscribe Request (id=10) [university/facility4/section2/light]
951	2024-01-27 15:54:35,6254565	10.0.2.15	59385	3.65.168.153	1883	MQTT	82	82	Subscribe Request (id=11) [university/kcblh/+]
992	2024-01-27 15:54:36,6292093	10.0.2.15	59385	3.65.168.153	1883	MQTT	96	96	Subscribe Request (id=12) [university/kcblh/floor5/humidity]
1067	2024-01-27 15:54:37,6316346	10.0.2.15	59385	3.65.168.153	1883	MQTT	80	80	Subscribe Request (id=13) [factory/facility4]
1109	2024-01-27 15:54:38,6325848	10.0.2.15	59385	3.65.168.153	1883	MQTT	79	79	Subscribe Request (id=14) [university/room3]
1185	2024-01-27 15:54:40,6361911	10.0.2.15	59385	3.65.168.153	1883	MQTT	73	73	Subscribe Request (id=15) [hospital/#]
1223	2024-01-27 15:54:41,6376125	10.0.2.15	59385	3.65.168.153	1883	MQTT	82	82	Subscribe Request (id=16) [metaverse/building3]
1313	2024-01-27 15:54:43,6395869	10.0.2.15	59385	3.65.168.153	1883	MQTT	80	80	Subscribe Request (id=17) [factory/building3]
1356	2024-01-27 15:54:44,6403953	10.0.2.15	59385	3.65.168.153	1883	MQTT	81	81	Subscribe Request (id=18) [metaverse/kcblh/+]
1445	2024-01-27 15:54:46,6438963	10.0.2.15	59385	3.65.168.153	1883	MQTT	80	80	Subscribe Request (id=19) [university/kcblh]
1526	2024-01-27 15:54:48,6438925	10.0.2.15	59385	3.65.168.153	1883	MQTT	83	83	Subscribe Request (id=20) [university/facility4]
1623	2024-01-27 15:54:50,6459866	10.0.2.15	59385	3.65.168.153	1883	MQTT	93	93	Subscribe Request (id=21) [factory/room3/floor5/pollution]

[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 "#")

```

The image shows a Wireshark capture of MQTT traffic. The filter is set to `[mqtt and mqtt.msghtype == 8 and tcp.dstport == 1883 and ip.dst != 127.0.0.1 and (mqtt.topic contains "+" or mqtt.topic contains "#")]`. The table below lists the captured messages.

No.	Time	Source	Src Port	Destination	Dest Port	Protocol	Length	MID	Info
449	66.434885648	10.0.2.15	36707	3.65.35.116	1883	MQTT	79		Subscribe Request (id=6) [factory/+/area0]
282	57.684680968	10.0.2.15	37419	91.121.93.94	1883	MQTT	97		Subscribe Request (id=3) [metaverse/department2/+/pollution]
254	56.606566784	10.0.2.15	37419	91.121.93.94	1883	MQTT	76		Subscribe Request (id=4) [hospital/+/+]
296	60.696118129	10.0.2.15	37419	91.121.93.94	1883	MQTT	87		Subscribe Request (id=6) [metaverse/+/area0/light]
459	66.714654956	10.0.2.15	37419	91.121.93.94	1883	MQTT	82		Subscribe Request (id=10) [metaverse/+/floor5]
496	67.715235545	10.0.2.15	37419	91.121.93.94	1883	MQTT	71		Subscribe Request (id=11) [house/+]
563	69.716284623	10.0.2.15	37419	91.121.93.94	1883	MQTT	89		Subscribe Request (id=12) [factory/facility4/area0/+]
198	57.526389492	10.0.2.15	38887	3.65.168.153	1883	MQTT	82		Subscribe Request (id=2) [factory/facility4/+]
248	56.529462785	10.0.2.15	38887	3.65.168.153	1883	MQTT	72		Subscribe Request (id=3) [factory/+]
290	60.535766441	10.0.2.15	38887	3.65.168.153	1883	MQTT	80		Subscribe Request (id=4) [metaverse/room3/#]
377	64.539628627	10.0.2.15	38887	3.65.168.153	1883	MQTT	73		Subscribe Request (id=6) [hospital/+]
419	65.548914755	10.0.2.15	38887	3.65.168.153	1883	MQTT	88		Subscribe Request (id=7) [metaverse/kcbplh/floor5/+]
791	75.552845095	10.0.2.15	38887	3.65.168.153	1883	MQTT	82		Subscribe Request (id=14) [factory/building3/#]
351	63.490223942	10.0.2.15	59385	3.65.168.153	1883	MQTT	80		Subscribe Request (id=1) [hospital/kcbplh/#]
416	65.493385511	10.0.2.15	59385	3.65.168.153	1883	MQTT	87		Subscribe Request (id=3) [house/building3/floor5/+]
551	69.497641433	10.0.2.15	59385	3.65.168.153	1883	MQTT	90		Subscribe Request (id=5) [metaverse/facility4/+/light]
748	74.508567325	10.0.2.15	59385	3.65.168.153	1883	MQTT	73		Subscribe Request (id=8) [hospital/+]
789	75.513462822	10.0.2.15	59385	3.65.168.153	1883	MQTT	89		Subscribe Request (id=9) [metaverse/+/area0/humidity]
951	78.517098802	10.0.2.15	59385	3.65.168.153	1883	MQTT	82		Subscribe Request (id=11) [university/kcbplh/+]
1185	83.527833454	10.0.2.15	59385	3.65.168.153	1883	MQTT	73		Subscribe Request (id=15) [hospital/#]
1356	87.531947583	10.0.2.15	59385	3.65.168.153	1883	MQTT	81		Subscribe Request (id=16) [metaverse/kcbplh/s]

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

No.	Time	Source	Src Port	Destination	Dest Port	Protocol	Length	MID	Info
449	66.494885648	10.0.2.15	36707	3.65.35.116	1883	MQTT	79		Subscribe Request (id=6) [factory/+\\area0]
292	57.684680968	10.0.2.15	37419	91.121.93.94	1883	MQTT	97		Subscribe Request (id=3) [metaverse/departement2/+\\pollution]
254	58.686566784	10.0.2.15	37419	91.121.93.94	1883	MQTT	76		Subscribe Request (id=4) [hospital/+\\]
216	59.550111152	10.0.2.15	37419	91.121.93.94	1883	MQTT	57		Subscribe Request (id=4) [metaverse/+\\area0/light]
459	65.714545956	10.0.2.15	37419	91.121.93.94	1883	MQTT	82		Subscribe Request (id=10) [metaverse/+\\floor5]
486	67.715235545	10.0.2.15	37419	91.121.93.94	1883	MQTT	71		Subscribe Request (id=11) [house/+\\]
563	69.716204623	10.0.2.15	37419	91.121.93.94	1883	MQTT	89		Subscribe Request (id=12) [factory/facility4/area0/+\\]
198	57.526389492	10.0.2.15	38887	3.65.168.153	1883	MQTT	82		Subscribe Request (id=2) [factory/facility4/+\\]
248	58.529462785	10.0.2.15	38887	3.65.168.153	1883	MQTT	72		Subscribe Request (id=3) [factory/+\\]
290	60.535708441	10.0.2.15	38887	3.65.168.153	1883	MQTT	80		Subscribe Request (id=4) [metaverse/room3/#]
377	64.539628027	10.0.2.15	38887	3.65.168.153	1883	MQTT	73		Subscribe Request (id=6) [hospital/+\\]
419	65.540914755	10.0.2.15	38887	3.65.168.153	1883	MQTT	88		Subscribe Request (id=7) [metaverse/kcbplh/floor5/#]
791	75.552845995	10.0.2.15	38887	3.65.168.153	1883	MQTT	82		Subscribe Request (id=14) [factory/building3/#]
351	63.490223942	10.0.2.15	59305	3.65.168.153	1883	MQTT	80		Subscribe Request (id=1) [hospital/kcbplh/+\\]
416	65.493385511	10.0.2.15	59305	3.65.168.153	1883	MQTT	87		Subscribe Request (id=3) [house/building3/floor5/+\\]
553	69.547912113	10.0.2.15	59305	3.65.168.153	1883	MQTT	57		Subscribe Request (id=5) [metaverse/facility4/+\\light]
748	74.509071325	10.0.2.15	59305	3.65.168.153	1883	MQTT	73		Subscribe Request (id=8) [hospital/+\\]
789	75.513346282	10.0.2.15	59305	3.65.168.153	1883	MQTT	89		Subscribe Request (id=9) [metaverse/+\\area0/humidity]
951	78.517098802	10.0.2.15	59305	3.65.168.153	1883	MQTT	82		Subscribe Request (id=11) [university/kcbplh/+\\]
1185	83.527833454	10.0.2.15	59305	3.65.168.153	1883	MQTT	73		Subscribe Request (id=15) [hospital/+\\]
1356	87.531947583	10.0.2.15	59305	3.65.168.153	1883	MQTT	81		Subscribe Request (id=18) [metaverse/kcbplh/#]

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

mqttan and mqttan.topicId == 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 15:56:06,3470151	127.0.0.1	51150	127.0.0.1	1885	ICMP	81	6	Destination unreachable (Port unreachable)
3557	2024-01-27 15:56:15,6570047	127.0.0.1	50877	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
3558	2024-01-27 15:56:15,6570134	127.0.0.1	50877	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
3823	2024-01-27 15:56:28,1888688	127.0.0.1	50778	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
3824	2024-01-27 15:56:28,1888688	127.0.0.1	50778	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
4026	2024-01-27 15:56:35,6232876	127.0.0.1	50852	127.0.0.1	1885	MQTT-SN	53	6	Publish Message
4027	2024-01-27 15:56:35,6232942	127.0.0.1	50852	127.0.0.1	1885	ICMP	81	6	Destination unreachable (Port unreachable)
4189	2024-01-27 15:56:45,7382616	127.0.0.1	41552	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
4190	2024-01-27 15:56:45,7382697	127.0.0.1	41552	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 15:57:06,2176699	127.0.0.1	45804	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
4940	2024-01-27 15:57:16,8476585	127.0.0.1	59362	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
4941	2024-01-27 15:57:16,8476585	127.0.0.1	59362	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
5082	2024-01-27 15:57:25,2123009	127.0.0.1	97998	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
5083	2024-01-27 15:57:25,2123089	127.0.0.1	97998	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 15:57:31,7960405	127.0.0.1	45759	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
5211	2024-01-27 15:57:32,6234255	127.0.0.1	49070	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
5212	2024-01-27 15:57:32,6234506	127.0.0.1	49070	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
5312	2024-01-27 15:57:42,1668045	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 16:00:56,2688951	127.0.0.1	49962	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
11484	2024-01-27 16:00:56,2688951	127.0.0.1	49962	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
11524	2024-01-27 16:00:52,8787395	127.0.0.1	54571	127.0.0.1	1885	MQTT-SN	54	6	Publish Message
11525	2024-01-27 16:00:52,8787493	127.0.0.1	54571	127.0.0.1	1885	ICMP	82	6	Destination unreachable (Port unreachable)
11927	2024-01-27 16:01:02,8799144	127.0.0.1	54739	127.0.0.1	1885	MQTT-SN	53	6	Publish Message
11928	2024-01-27 16:01:02,8799202	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