POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

IoT Challenge #2, Packet Sniffing

Internet of Things

Authors: **Kevin Ziroldi - 10764177**

Matteo Volpari - 10773593

Professors: Alessandro Redondi, Fabio Palmese
Academic Year: 2024-2025
Version: 1.0
Release date: 6-4-2025

Contents

1 | Packet sniffing PCAP file

1.1. CQ1

Question

How many different Confirmable PUT requests obtained an unsuccessful response from the local CoAP server?

Answer

The number of Confirmable PUT request that obtained an unsuccessful response from the local CoAP server is 22.

Explanation

We filter the Confirmable PUT requests to a local CoAP server with a filter: Confirmable requests have type 0, PUT requests have code 3.

```
coap && coap.type == 0 && coap.code == 3 && ip.src == ip.dst
```

We get 26 filtered messages.

We filter the unsuccessful responses from a local CoAP server based on the code, which needs to be bigger or equal than 128.

```
coap && (coap.code >= 128) && ip.src == ip.dst
```

We get 141 filtered messages.
In order to answer QC1, we should follow the UDP stream for all 26 requests and check the code associated to the response, checking that all requests are different, i.e. have different token.
To avoid doing this manually, we used a Python script written using PyShark library.

```
import pyshark
import sys
import re
```

```
def answer_cq1(capture):
    requests = {}
    responses = {}
    for pkt in capture:
        try:
            # check local server
            if 'IP' not in pkt or pkt.ip.src != pkt.ip.dst:
                continue
            # check CoAP layer
            if 'COAP' in pkt:
                coap = pkt.coap
                # get fields
                coap_type = int(coap.get_field('type')) if hasattr(coap,
                                                     'type') else None
                coap_code = int(coap.get_field('code')) if hasattr(coap,
                                                     'code') else None
                token = coap.get_field('token') if hasattr(coap, 'token'
                                                   ) else None
                if coap_type is None or coap_code is None or token is
                                                   None:
                    continue
                if coap_type == 0 and coap_code == 3:
                    # Confirmable PUT request
                    requests[token] = pkt
                elif coap_code >= 128:
                    # unsuccessful response
                    responses[token] = pkt
        except Exception:
            continue
    count = sum(1 for token in requests if token in responses)
    return count
def main():
    if len(sys.argv) != 2:
        print("pcap file not provided")
        sys.exit(1)
    pcap_file = sys.argv[1]
    capture = pyshark.FileCapture(pcap_file, keep_packets=False)
    print(answer_cq1(capture))
    capture.close()
if __name__ == "__main__":
    main()
```

Running this script, we get a result of 22 requests.

1.2. CQ2

Question

How many CoAP resources in the coap.me public server received the same number of unique Confirmable and Non Confirmable GET requests?
Assuming a resource receives X different CONFIRMABLE requests and Y different NON-CONFIRMABLE GET requests, how many resources have X=Y, with X>0?

Answer

The number of CoAP resources in the coap.me public server that received the same number of unique Confirmable and Non Confirmable GET requests is 3.

Explanation

We can find the IP address of coap.me server with a Wireshark filter.

`dns.qry.name == "coap.me"`

We can see that the IP is 134.102.218.18.

We can find the Confirmable GET requests issued to coap.me with the following filter:

`coap.type == 0 && coap.code == 1 && ip.dst==134.102.218.18`

While Non Confirmable GET requests to coap.me can be found with this filter:

`coap.type == 1 && coap.code == 1 && ip.dst==134.102.218.18`

We find 39 Confirmable request and 31 Non Confirmable ones and we would need to count the number of requests for each resource.
Instead, we can use a PyShark algorithm to do the same.

```python
import pyshark
import re
import sys

def answer_cq2(capture):
    # IP found with Wireshark filter
    coap_me_ip = "134.102.218.18"
    resource_stats = {}
    # find target requests and responses
    for pkt in capture:
```

```
        try:
            # check destination is coap.me
            if 'IP' not in pkt or pkt.ip.dst != coap_me_ip:
                continue
            # check CoAP layer
            if 'COAP' in pkt:
                coap = pkt.coap
                # get fields
                coap_type = int(coap.get_field('type')) if hasattr(coap,
                                                        'type') else None
                coap_code = int(coap.get_field('code')) if hasattr(coap,
                                                        'code') else None
                token = coap.get_field('token') if hasattr(coap, 'token'
                                                        ) else None
                resource = coap.get_field('opt_uri_path') if hasattr(
                                                        coap, 'opt_uri_path'
                                                        ) else None
                if coap_type is None or coap_code is None or token is
                                                        None or resource is
                                                        None:
                    continue
                # check GET request
                if coap_code != 1:
                    continue
                if resource not in resource_stats:
                    resource_stats[resource] = {'conf': set(), 'nonconf'
                                                        : set()}
                if coap_type == 0:
                    # Confirmable
                    resource_stats[resource]['conf'].add(token)
                elif coap_type == 1:
                    # Non Confirmable
                    resource_stats[resource]['nonconf'].add(token)
        except Exception:
            continue
    # count target resources
    count = 0
    for stats in resource_stats.values():
        if len(stats['conf']) == len(stats['nonconf']) and len(stats['
                                                conf']) > 0:
            count += 1
    return count

def main():
```

```python
    if len(sys.argv) != 2:
        print("pcap file not provided")
        sys.exit(1)
    pcap_file = sys.argv[1]
    capture = pyshark.FileCapture(pcap_file, keep_packets=False)
    print(answer_cq2(capture))
    capture.close()

if __name__ == "__main__":
    main()
```

Running the algorithm, we find out that 3 resources have the same number of Confirmable and Non Confirmable GET requests.

1.3. CQ3

Question

How many different MQTT clients subscribe to the public broker HiveMQ using multi-level wildcards?

Answer

The number of clients who subscribe to the public broker HiveMQ using multi-level wildcards is 4.

Explanation

In order to find the IP address of the HiveMQ broker, we filter the response of the DNS server using the following Wireshark filter:

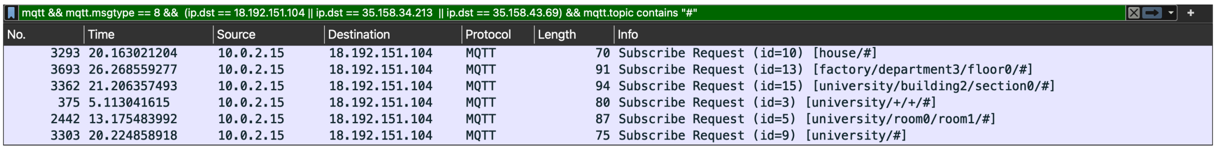
```
dns.qry.name == "broker.hivemq.com"
```

All DNS responses return 3 addresses: 18.192.151.104, 35.158.34.213 and 35.158.43.69.

We use a second filter to find SUBSCRIBE messages, with message type 8, sent to HiveMQ broker, to one of the IP addresses found above, with a multi-level wildcard, ending with "#":

```
mqtt && mqtt.msgtype == 8 &&
(ip.dst == 18.192.151.104 || ip.dst == 35.158.34.213
|| ip.dst == 35.158.43.69) && mqtt.topic contains "#"
```

We find out that HiveMQ broker receives 6 messages of this type, all at the IP address 18.192.151.104.



mqtt && mqtt.msgtype == 8 && (ip.dst == 18.192.151.104 || ip.dst == 35.158.34.213 || ip.dst == 35.158.43.69) && mqtt.topic contains "#"

No.	Time	Source	Destination	Protocol	Length	Info
3293	20.163021204	10.0.2.15	18.192.151.104	MQTT	70	Subscribe Request (id=10) [house/#]
3693	26.268559277	10.0.2.15	18.192.151.104	MQTT	91	Subscribe Request (id=13) [factory/department3/floor0/#]
3362	21.206357493	10.0.2.15	18.192.151.104	MQTT	94	Subscribe Request (id=15) [university/building2/section0/#]
375	5.113041615	10.0.2.15	18.192.151.104	MQTT	80	Subscribe Request (id=3) [university/+/+/#]
2442	13.175483992	10.0.2.15	18.192.151.104	MQTT	87	Subscribe Request (id=5) [university/room0/room1/#]
3303	20.224858918	10.0.2.15	18.192.151.104	MQTT	75	Subscribe Request (id=9) [university/#]

Figure 1.1: SUBSCRIBE messages to HiveMQ broker with "#"

Since the question asks for the number of MQTT clients who subscribe, we need to identify the clients who sent these messages. For each message, we select the TCP stream, which identifies the client.

Message number	TCP stream
375	8
2442	15
3293	20
3303	15
3362	3
3693	15

Table 1.1: TCP streams

Since there are 4 TCP streams, the 6 messages have been sent by 4 different client.
We can also find the Client ID of these clients by finding the CONNECT message, of type 1, they sent to the broker. For the TCP stream 8, we can use the following filter:

```
mqtt && mqtt.msgtype == 1 && tcp.stream == 8
```

The same filter with different TCP stream can be used for other clients.

TCP stream	Client ID
3	cpoepjzkhibxgjiu
8	dzcxnwdqef
15	tukvxesuhe
20	fcthvjikxjul

Table 1.2: Client IDs table

1.4. CQ4

Question

How many different MQTT clients specify a Last Will Message to be directed to a topic having as first level "university"?

Answer

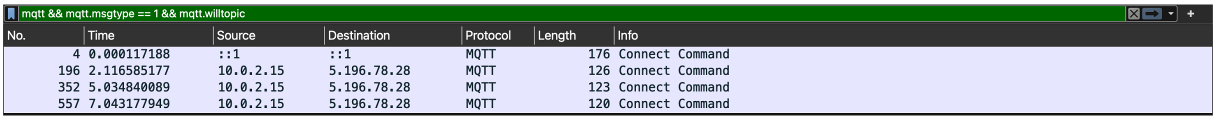
The number of clients who specify a Last Will Message to be directed to a topic having as first level "university" is 1.

Explanation

MQTT clients can specify a Last Will Message in the CONNECT message. In order to find the described messages, we filter CONNECT messages, of type 1, with a Last Will Topic:

```
mqtt && mqtt.msgtype == 1 && mqtt.willtopic
```

We find four messages, but only one of them has a Last Will Topic having as first level "university".


mqtt && mqtt.msgtype == 1 && mqtt.willtopic

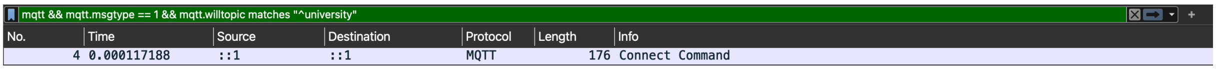
No.	Time	Source	Destination	Protocol	Length	Info
4	0.000117188	::1	::1	MQTT	176	Connect Command
196	2.116585177	10.0.2.15	5.196.78.28	MQTT	126	Connect Command
352	5.034840089	10.0.2.15	5.196.78.28	MQTT	123	Connect Command
557	7.043177949	10.0.2.15	5.196.78.28	MQTT	120	Connect Command

Figure 1.2: CONNECT messages specifying a Last Will Topic

We can find the result by enriching the filter and avoiding manually checking the topics, using the following filter:

```
mqtt && mqtt.msgtype == 1 && mqtt.willtopic matches "^university"
```

Using this filter, we directly get the only message asked by CQ4.



mqtt && mqtt.msgtype == 1 && mqtt.willtopic matches "^university"

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000117188	::1	::1	MQTT	176	Connect Command

Figure 1.3: CONNECT messages specifying a Last Will Topic starting with "university"

1.5. CQ5

Question

How many MQTT subscribers receive a last will message derived from a subscription without a wildcard?

Answer

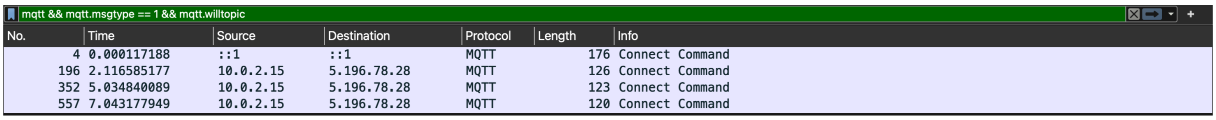
The number of subscribers who receive a Last Will Message derived from a subscription without a wildcard is 3.

Explanation

We start by identifying the possible Last Will Messages (LWM). To do so, we find all the CONNECT messages, with message type 1, that specify a LWM.

```
mqtt && mqtt.msgtype== 1 && mqtt.willmsg
```

We find four messages.



mqtt && mqtt.msgtype == 1 && mqtt.willtopic

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000117188	::1	::1	MQTT	176	Connect Command
196	2.116585177	10.0.2.15	5.196.78.28	MQTT	126	Connect Command
352	5.034840089	10.0.2.15	5.196.78.28	MQTT	123	Connect Command
557	7.043177949	10.0.2.15	5.196.78.28	MQTT	120	Connect Command

Figure 1.4: CONNECT messages specifying a LWM

These messages specify the Last Will Messages, which we don't report in the table, and Last Will Topics.

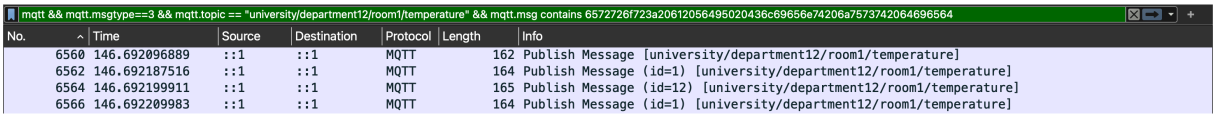
Message number	Destination	Last Will Topic
4	::1	university/department12/room1/temperature
196	5.196.78.28	metaverse/room2/floor4
352	5.196.78.28	hospital/facility3/area3
557	5.196.78.28	metaverse/room2/room2

Table 1.3: Last Will Topics

Starting from the first Last Will Topic (LWT), we filter all PUBLISH messages with that topic and with same message as the LWM, found in the CONNECT message.

```
mqtt && mqtt.msgtype==3 &&
mqtt.topic == "university/department12/room1/temperature" &&
mqtt.msg contains 6572726f723a20612056495020436c69656e74206a7573742064696564
```

We find four messages:



mqtt && mqtt.msgtype==3 && mqtt.topic == "university/department12/room1/temperature" && mqtt.msg contains 6572726f723a20612056495020436c69656e74206a7573742064696564

No.	Time	Source	Destination	Protocol	Length	Info
6560	146.692096889	::1	::1	MQTT	162	Publish Message [university/department12/room1/temperature]
6562	146.692187516	::1	::1	MQTT	164	Publish Message (id=1) [university/department12/room1/temperature]
6564	146.692199911	::1	::1	MQTT	165	Publish Message (id=12) [university/department12/room1/temperature]
6566	146.692209983	::1	::1	MQTT	164	Publish Message (id=1) [university/department12/room1/temperature]

Figure 1.5: LWM with topic "university/department12/room1/temperature"

By looking at the messages, we can see a TCP Reset message, representing an hard disconnection, followed by four Last Will Messages sent by the broker, on port 1883, to different client, on different ports and using different TCP streams.

6559	146.691800286	::1	::1	TCP	88	38083 → 1883 [RST, ACK] Seq=9316 Ack=85 Win=65536 Len=0 TSval=2654936537 TSecr=265493493
6560	146.692096889	::1	::1	MQTT	162	Publish Message [university/department12/room1/temperature]
6561	146.692172650	::1	::1	TCP	88	39551 → 1883 [ACK] Seq=77 Ack=86 Win=65536 Len=0 TSval=2654936537 TSecr=2654936537
6562	146.692187516	::1	::1	MQTT	164	Publish Message (id=1) [university/department12/room1/temperature]
6563	146.692189976	::1	::1	TCP	88	53557 → 1883 [ACK] Seq=94 Ack=86 Win=65536 Len=0 TSval=2654936537 TSecr=2654936537
6564	146.692199911	::1	::1	MQTT	165	Publish Message (id=12) [university/department12/room1/temperature]
6565	146.692202117	::1	::1	TCP	88	51743 → 1883 [ACK] Seq=573 Ack=14152 Win=64896 Len=0 TSval=2654936537 TSecr=2654936537
6566	146.692209983	::1	::1	MQTT	164	Publish Message (id=1) [university/department12/room1/temperature]

Figure 1.6: TCP Reset and Last Will Messages

The clients who receive the LWM are grouped in the following table.

Message number	Subscriber port	TCP stream
6560	39551	2
6562	53557	6
6564	51743	10
6566	41789	14

Table 1.4: Last Will Topics

In order to answer to QC5, we need to find which of these clients subscribed to the Last Will Topic without a wildcard. We can do so by filtering the SUBSCRIBE messages, with message type 8, with the correct TCP stream. For each of them we also find the CONNECT message and Client ID.

```
mqtt && mqtt.msgtype == 8 && mqtt.topic matches "^university" &&
(tcp.stream == 2 || tcp.stream == 6 || tcp.stream == 10 || tcp.stream == 14)
```

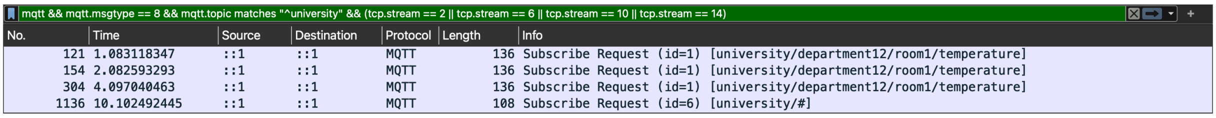


Figure 1.7: TCP Reset and Last Will Messages

We can see that only three out of four clients subscribed to the Last Will Topic without a wildcard.

TCP stream	Client ID	Specified topic
2	auyvhrhdudnm	university/department12/room1/temperature
6	ntpiopsqc	university/department12/room1/temperature
10	zmjnxudohrkaegmh	university/#
14	mjdocmjxt	university/department12/room1/temperature

Table 1.5: Specified topics

For what concerns the other three Last Will Topics, we filter all PUBLISH messages, with message type 3, from the broker, with IP 5.196.78.28.

```
mqtt && ip.src == 5.196.78.28 && mqtt.msgtype == 3
```

We don't find any result, which means that the broker doesn't publish any message, nor LWM.
We conclude that three clients receive a LWM from a subscription without a wildcard, all of them from the first topic.

1.6. CQ6

Question

How many MQTT publish messages directed to the public broker mosquitto are sent with the retain option and use QoS âĂIJAt most onceâĂİ?

Answer

The number of publish messages directed to the public broker mosquitto are sent with the retain option and use QoS âĂIJAt most onceâĂİ is 208.

Explanation

In order to find the IP address of the public broker mosquitto, we filter the response of the DNS server using the following Wireshark filter:

```
dns.qry.name == "test.mosquitto.org"
```

All DNS responses return the same IP address: 5.196.78.28.
Now we need a second filter in order to obtain all the publish messages that satisfy the question:

```
mqtt.msgtype == 3 and mqtt.qos == 0 and mqtt.retain == 1
and ip.dst == 5.196.78.28
```

We find 208 results, so we can conclude that the number of packets that satisfy all the constraints is 208.

1.7. CQ7

Question

How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine?

Answer

The number of MQTT-SN messages on port 1885 sent by the clients to a broker in the local machine is 0.

Explanation

Natively MQTT-SN is not recognised by Wireshark and therefore a preliminary operation is required. By going into the settings we can set 1885 as the port of MQTT-SN. Since MQTT-SN is a UDP protocol, in order to filter all the messages using MQTT-SN we can use this Wireshark filter:

```
udp.dstport == 1885
```

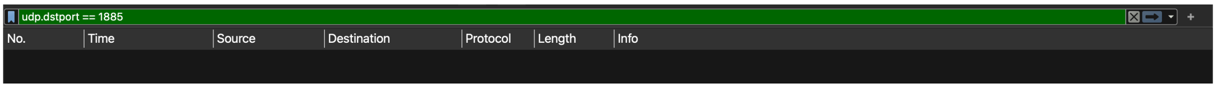


Figure 1.8: UDP destination port 1885

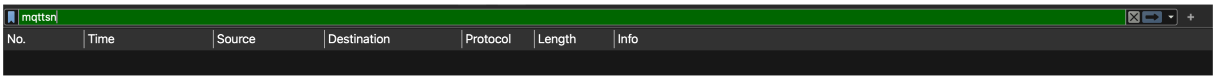


Figure 1.9: MQTT-SN messages

However, both this filter and the standard âĂŸmqttsnâĂŹ filter do not return a single packet, so to be sure that the result is 0, it is possible to carry out a further check knowing that MQTT-SN always has topic length equal to 2:

```
mqtt.topic_len == 2
```

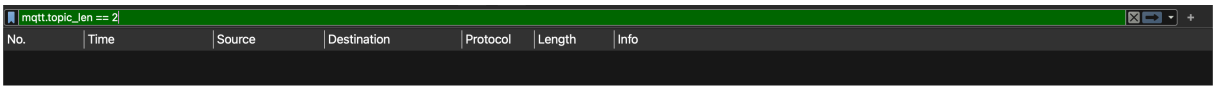


Figure 1.10: MQTT messages with topic length 2

But this filter also gives no results, so we can conclude that there are no MQTT-SN messages sent on port 1885 by the clients to a broker in the local machine.

1.8. References

In this section we provide a set of references we used.

- https://datatracker.ietf.org/doc/html/rfc7641
- https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html