

# Internet Of Things - Second Challenge

Authors:

Kevin Abate: 10812892

Lorenzo Pigato: 10766953 [Team Leader]

## Project topic

The project consists in **analyzing network traffic** produced by **MQTT** and **CoAP** clients and brokers. Packets have been analyzed using Wireshark, mainly through *thsark* terminal interface in order to exploit advanced filtering and sorting

## Questions

### CQ1

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

**Answer: 22**

The answer is obtained by filtering all **confirmable PUT requests** and saving their **message id** in order to check if any **error response** has been sent by the server:

#### Filters

```
coap.type == 0 && coap.code == 3          // CONFirmable PUT requests
coap.code >= 128 && ip.src == 127.0.0.1  // Error rsponse sent from the server
```

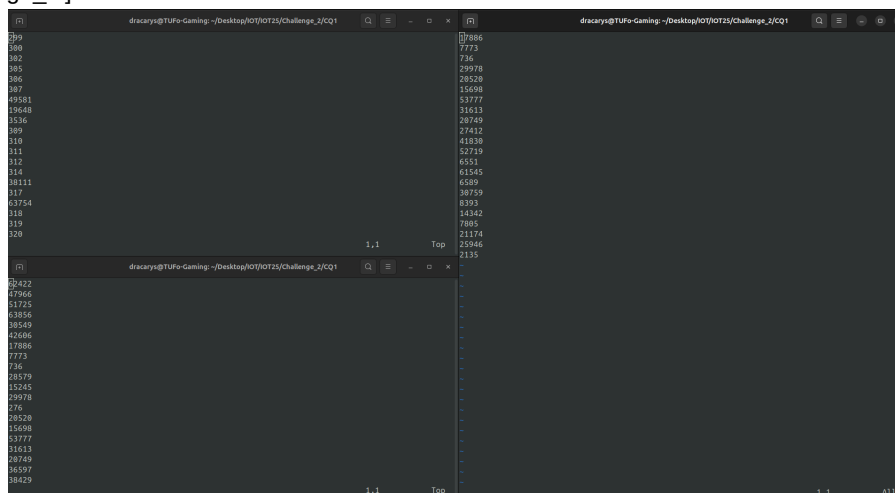
#### tshark commands

```
tshark -r challenge2.pcapng -Y "coap.type == 0 && coap.code == 3" -T fields -e coap.mid >
put_requests.txt
```

```
tshark -r challenge2.pcapng -Y "coap.code >= 128 && ip.src == 127.0.0.1" -T
fields -e coap.mid > error_responses.txt
```

```
grep -F -f put_requests.txt error_responses.txt | wc -l
```

#### Final output [message\_id]



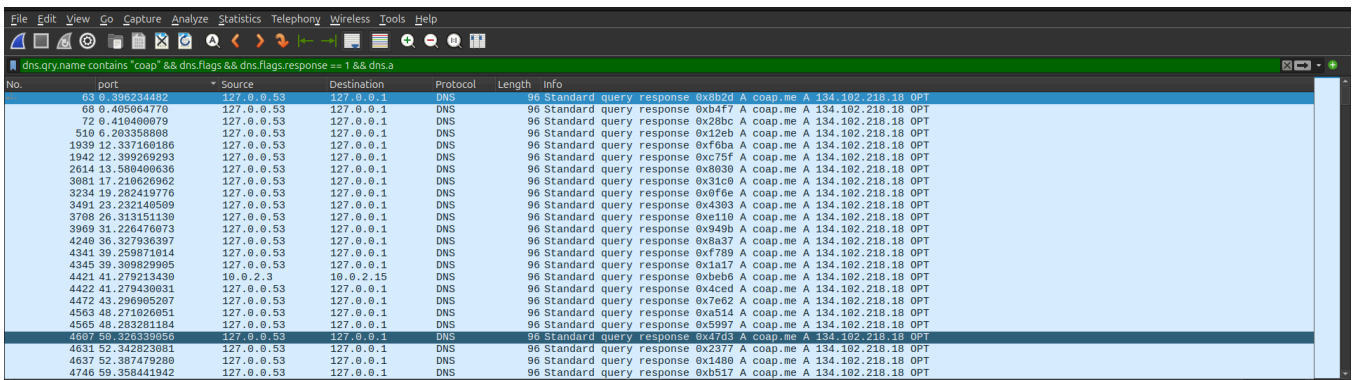
## CQ2

How many CoAP resources in the coap.me public server received the same number of unique Confirmable and Non Confirmable GET requests?

**Answer: 3 [large, secret, validate]**

In order to solve the problem, it was necessary to find out the **address** of the **coap.me** server first. This can be done by looking up DNS requests:

```
dns.qry.name contains "coap" && dns.flags && dns.flags.response == 1 && dns.a
```



No.	port	Source	Destination	Protocol	Length	Info
65	356234482	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xb2d A coap.me A 134.102.218.18 OPT
66	0.405064770	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xb4f7 A coap.me A 134.102.218.18 OPT
72	0.410400079	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x28bc A coap.me A 134.102.218.18 OPT
510	6.203358008	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x12eb A coap.me A 134.102.218.18 OPT
1839	12.337160186	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xf6ba A coap.me A 134.102.218.18 OPT
1942	12.399269293	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xc75f A coap.me A 134.102.218.18 OPT
2014	13.580400636	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x8030 A coap.me A 134.102.218.18 OPT
3081	17.218626962	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x31c0 A coap.me A 134.102.218.18 OPT
3234	19.282419776	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x8f6e A coap.me A 134.102.218.18 OPT
3491	23.232140509	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x4303 A coap.me A 134.102.218.18 OPT
3708	26.313511330	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xe110 A coap.me A 134.102.218.18 OPT
3969	31.226476073	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x949b A coap.me A 134.102.218.18 OPT
4240	36.327936397	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x8a37 A coap.me A 134.102.218.18 OPT
4341	39.259871014	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xf789 A coap.me A 134.102.218.18 OPT
4345	39.399829905	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x1a17 A coap.me A 134.102.218.18 OPT
4421	41.279213430	10.0.2.3	10.0.2.15	DNS	96	Standard query response 0x9beb A coap.me A 134.102.218.18 OPT
4422	41.279439831	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x4ced A coap.me A 134.102.218.18 OPT
4472	43.296905207	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x7e62 A coap.me A 134.102.218.18 OPT
4503	46.271026051	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xa514 A coap.me A 134.102.218.18 OPT
4565	48.203201104	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x5997 A coap.me A 134.102.218.18 OPT
4607	50.320383056	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x4703 A coap.me A 134.102.218.18 OPT
4631	52.342823081	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x2377 A coap.me A 134.102.218.18 OPT
4637	52.387479280	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0x1489 A coap.me A 134.102.218.18 OPT
4746	59.358441942	127.0.0.1	127.0.0.1	DNS	96	Standard query response 0xb517 A coap.me A 134.102.218.18 OPT

from these DNS requests it is possible to retrieve the **coap.me ip address**: 134.102.218.18

Now two different filters have to be used: the first one will return the **NON-confirmable GET requests** directed to the **coap.me** server while the second one retrieves the **CONFIRMABLE** ones. Finally, the outputs of the two previously are compared and matching rows are returned.

### Filters

```
coap.type == 0 && coap.code == 1 && ip.dst == 134.102.218.18 // CON GET
coap.type == 1 && coap.code == 1 && ip.dst == 134.102.218.18 // NON GET
```

### tshark commands

```
tshark -r challenge2.pcapng -Y "coap.type == 1 && coap.code == 1 && ip.dst == 134.102.218.18" -T
fields -e coap.opt.uri_path -e coap.token | sort | uniq | cut -f1 | sort | uniq -c >
con_get_resources_count.txt

tshark -r challenge2.pcapng -Y "coap.type == 0 && coap.code == 1 && ip.dst == 134.102.218.18" -T
fields -e coap.opt.uri_path -e coap.token | sort | uniq | cut -f1 | sort | uniq -c >
non_get_resources_count.txt

# Find common columns between the two sorted files
comm -12 <(sort con_get_resources_count.txt) <(sort non_get_resources_count.txt)
```

Each *tshark* command produces a list of sets [resource\_path, token] which is then piped through a *sort* and *uniq* command in order to find unique couples of resources and tokens. This new list is then cut to keep only the resource and piped again through *sort* and counted using *uniq -c* to find how many requests are done for each resource

### Final output [requests, topic]

```
1 large
1 secret
1 validate
```

## CQ3

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

**Answer: 4 [38619, 38641, 54449, 57863]**

First, a Wireshark filter has been made in order to find out the **address** of the HiveMQ broker from DNS requests:

```
dns.qry.name == broker.hivemq.com && dns.flags && dns.flags.response == 1 && dns.a
```

No.	port	Source	Destination	Protocol	Length	Info
12	0.014094009	10.0.2.3	10.0.2.15	DNS	138	Standard query response 0x897a A broker.hivemq.com A 35.158.43.69 A 35.158.34.213 A 18.192.151.104 OPT
13	0.014250732	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0x60cb A broker.hivemq.com A 35.158.43.69 A 35.158.34.213 A 18.192.151.104 OPT
25	0.015270994	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0x4092 A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
111	1.001807121	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0x857a A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
310	4.100927463	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0x5418 A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
482	6.100326763	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0xaa5b A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
545	7.010606547	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0xc338 A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
613	8.016619966	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0x45d7 A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
1723	11.035386563	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0x928c A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT
1728	11.037562667	127.0.0.53	127.0.0.1	DNS	138	Standard query response 0xa285 A broker.hivemq.com A 18.192.151.104 A 35.158.34.213 A 35.158.43.69 OPT

**3 IP addresses** are provided by the DNS when the resolution of `broker.hivemq.com` is requested: `35.158.43.69` , `35.158.34.213` , `18.192.151.104`

Then, MQTT subscribe messages which were sent to these IPs were analyzed to identify topics that contain the **multi-level wildcard**:

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

No.	port	Source	Destination	Protocol	Length	Info
375	5.113041015	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/#]
3293	20.163021204	10.0.2.15	18.192.151.104	MQTT	70	Subscribe Request (id=10) [house/#]
3303	20.224859918	10.0.2.15	18.192.151.104	MQTT	75	Subscribe Request (id=9) [university/#]
3362	21.206357493	10.0.2.15	18.192.151.104	MQTT	94	Subscribe Request (id=15) [university/building2/section0/#]
3693	26.268559277	10.0.2.15	18.192.151.104	MQTT	91	Subscribe Request (id=13) [factory/departement3/floor0/#]

The clients **differ** from each other by the **TCP port** they use to make requests

**tshark commands**

```
tshark -r challenge2.pcapng -Y "(ip.addr == 35.158.43.69 || ip.addr == 35.158.34.213 || ip.addr == 18.192.151.104) && mqtt && mqtt.msgtype == 8 && mqtt.topic contains \"#\" \" -T fields -e ip.src -e tcp.srcport -e mqtt.topic > wildcard_subs.txt

cut -f2 wilcard_subs.txt | sort | uniq -c | wc -l
```

**Final Output** [ip\_src, src\_port, topic]

wildcard_subs.txt		
~/Desktop/IOT/IOT25/Challenge_2/CQ3		
10.0.2.15	38641	university/+/#
10.0.2.15	38619	university/room0/room1/#
10.0.2.15	54449	house/#
10.0.2.15	38619	university/#
10.0.2.15	57863	university/building2/section0/#
10.0.2.15	38619	factory/departement3/floor0/#

## CQ4

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

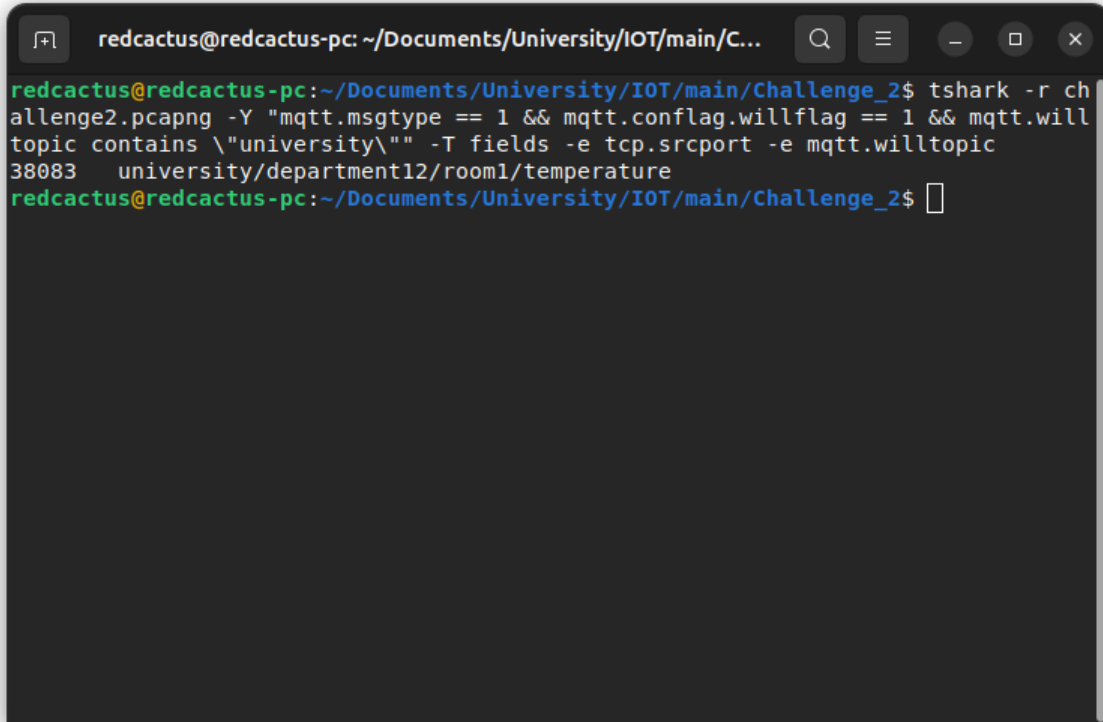
**Answer: 1 [38083 - university/department12/room1/temperature]**

A set of filter rules can be used to find all the **MQTT connect** requests, with the **willflag** and **CONNACK** flags set to 1 and a **Last Will Topic** that contains the string "university"

**Filter**

```
mqtt.msgtype == 1 && mqtt.conflag.willflag == 1 && mqtt.willtopic contains "university"
```

**Output** [src\_port, topic]



```
redcactus@redcactus-pc: ~/Documents/University/IOT/main/C...
redcactus@redcactus-pc:~/Documents/University/IOT/main/Challenge_2$ tshark -r challenge2.pcapng -Y "mqtt.msgtype == 1 && mqtt.conflag.willflag == 1 && mqtt.willtopic contains \"university\"" -T fields -e tcp.srcport -e mqtt.willtopic
38083    university/department12/room1/temperature
redcactus@redcactus-pc:~/Documents/University/IOT/main/Challenge_2$
```

## CQ5

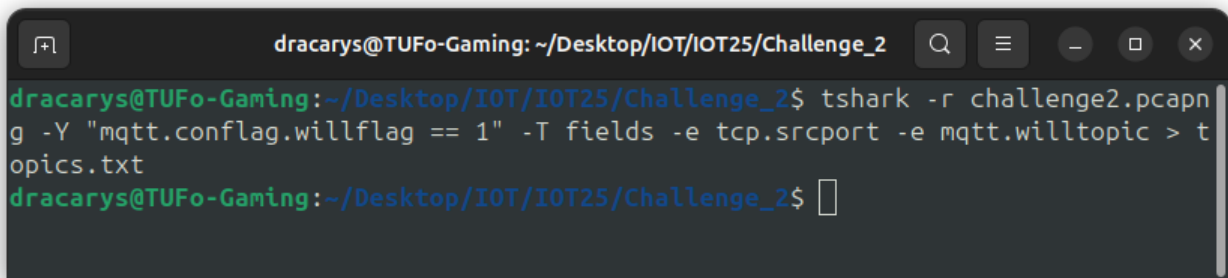
How many MQTT subscribers receive a last will message derived from a subscription without a wildcard?

**Answer: 3 [39551, 53557, 41789]**

To answer this question, five steps are required:

1 - **Distinct topics** that have a last will message flag set and the clients which set it are collected:

```
mqtt.conflag.willflag == 1
```



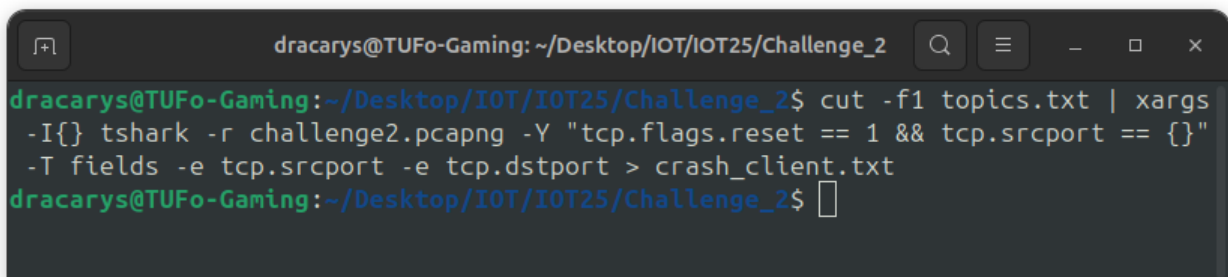
```
dracarys@TUfo-Gaming: ~/Desktop/IOT/IOT25/Challenge_2
dracarys@TUfo-Gaming:~/Desktop/IOT/IOT25/Challenge_2$ tshark -r challenge2.pcapng -Y "mqtt.conflag.willflag == 1" -T fields -e tcp.srcport -e mqtt.willtopic > topics.txt
dracarys@TUfo-Gaming:~/Desktop/IOT/IOT25/Challenge_2$
```

**Output:** [src\_port, topic]

```
38083  university/department12/room1/temperature
56285  metaverse/room2/floor4
53485  hospital/facility3/area3
42665  metaverse/room2/room2
```

2 - **Filter connection reset packets** to determine if any **connection reset** has been sent by a client that has configured a **Last Will Topic** message:

```
tcp.flags.reset == 1 && tcp.srcport == {topics.txt}
```



```
dracarys@TUfo-Gaming: ~/Desktop/IOT/IOT25/Challenge_2
dracarys@TUfo-Gaming:~/Desktop/IOT/IOT25/Challenge_2$ cut -f1 topics.txt | xargs -I{} tshark -r challenge2.pcapng -Y "tcp.flags.reset == 1 && tcp.srcport == {}" -T fields -e tcp.srcport -e tcp.dstport > crash_client.txt
dracarys@TUfo-Gaming:~/Desktop/IOT/IOT25/Challenge_2$
```

**Note:** `xargs` is used in order to provide filter inputs from the `topics.txt` file

**Output:**[src\_port, dst\_port]

```
38083  1883
```

#### 4 - Match Last Will Topic with failing clients:

```
dracarys@TUFO-Gaming: ~/Desktop/IOT/IOT25/Challenge_2
dracarys@TUFO-Gaming:~/Desktop/IOT/IOT25/Challenge_2$ grep "^$(cut -f1 ./CQ5/crash_client.txt)" ./CQ5/topics.txt | cut -f2 > topics_with_failures.txt
dracarys@TUFO-Gaming:~/Desktop/IOT/IOT25/Challenge_2$
```

Output: [topic]

```
university/department12/room1/temperature
```

5 - **Retrieve broker's messages destination:** Now that the single interesting topic is known, messages sent by the broker on it can be retrieved, with their destination being the interesting part:

```
tcp.srcport == 1883 && mqtt.msgtype == 3 && mqtt.topic == university/department12/room1/temperature
```

```
dracarys@TUFO-Gaming: ~/Desktop/IOT/IOT25/Challenge_2
dracarys@TUFO-Gaming:~/Desktop/IOT/IOT25/Challenge_2$ tshark -r challenge2.pcapng -Y "tcp.srcport == 1883 && mqtt.msgtype == 3 && mqtt.topic == university/department12/room1/temperature" -T fields -e tcp.dstport > sent_messages_to_clients.txt
dracarys@TUFO-Gaming:~/Desktop/IOT/IOT25/Challenge_2$
```

Output: [dst\_port]

```
39551
53557
51743
41789
```

6 - **Check subscriptions without wildcards:** With clients' port in hand, subscriptions messages from them to the topic without wildcards can be easily found

```
mqtt.msgtype == 8 && mqtt.topic == university/department12/room1/temperature
```

```
dracarys@TUFO-Gaming: ~/Desktop/IOT/IOT25/Challenge_2
dracarys@TUFO-Gaming:~/Desktop/IOT/IOT25/Challenge_2$ tshark -r challenge2.pcapng -Y "mqtt.msgtype == 8 && mqtt.topic == university/department12/room1/temperature" -T fields -e tcp.srcport | grep -F -f ./sent_messages_to_clients.txt > subscribed_clients.txt
dracarys@TUFO-Gaming:~/Desktop/IOT/IOT25/Challenge_2$
```

**Note:** `grep` is used in order to find common elements between the output found at point 5 and the output of this last filter

**Final answer:**

39551

53557

41789

---

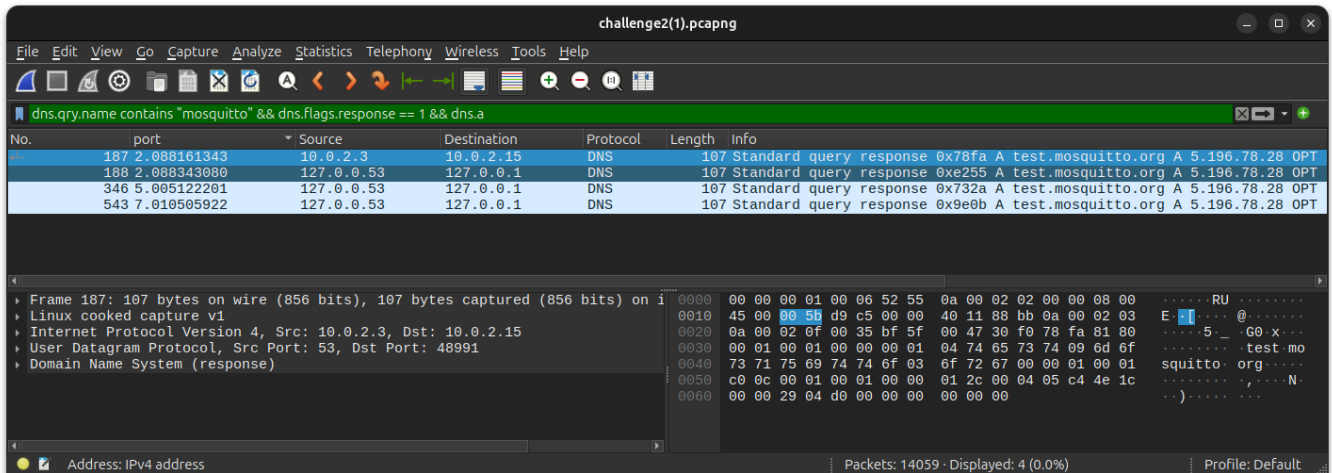
## CQ6

How many MQTT publish messages directed to the public broker Mosquitto are sent with the retain option and use QoS "At most once"?

**Answer: 208**

To find the address of the Mosquitto broker, DNS requests are filtered

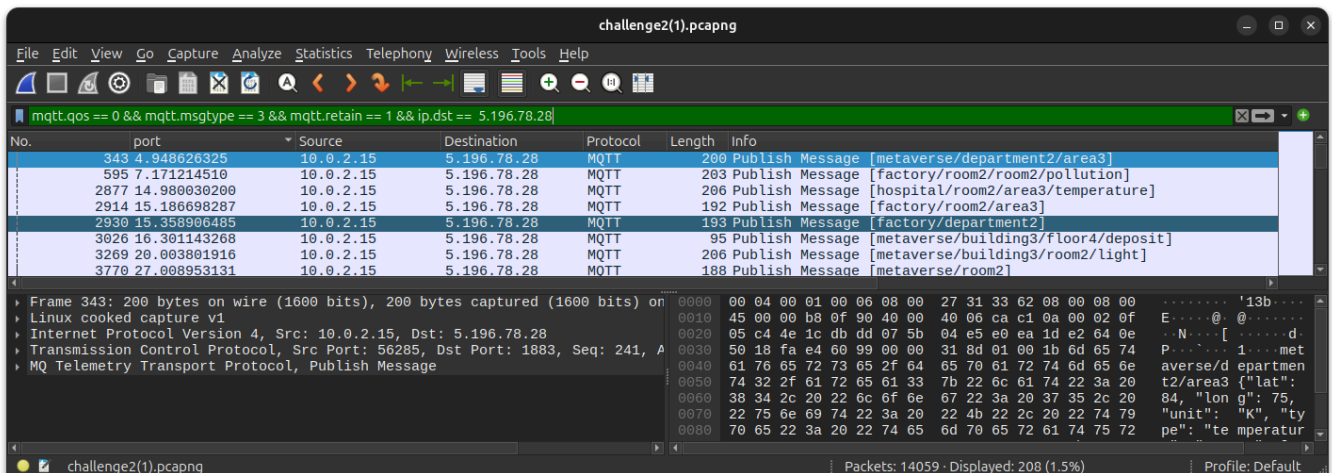
```
dns.qry.name contains "mosquitto" && dns.flags.response == 1 && dns.a
```



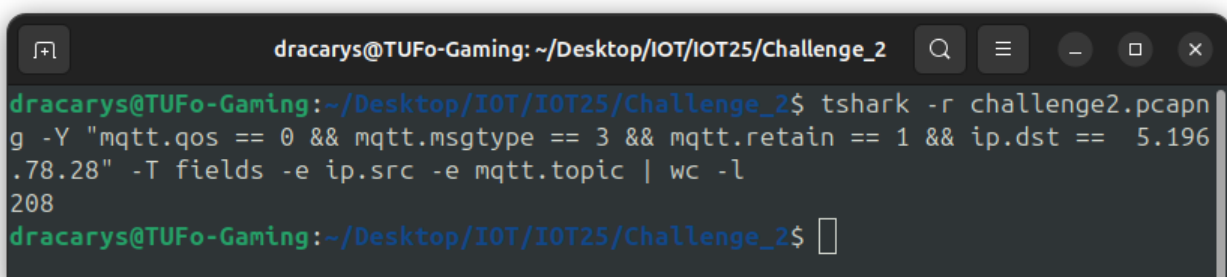
Mosquitto broker IP is: 5.196.78.28

It's now possible to analyze the traffic with QoS equals to zero (at most once), sent by the broker with the retain option activated

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



Packets are then counted using `wc -l`





## CQ7

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

### Answer: 0

After changing the port of the MQTT-SN to 1885, filters to find *mqttsn* packets were applied. No packet is shown, hence there is no *mqttsn* traffic

```
mqttsn && tcp.dstport == 1885
```

