



POLITECNICO
MILANO 1863

Internet of things

IoT Challenge #2
CoAP and MQTT protocols
Challenge PDF

Authors:

Daniel Shala - 10710181

Jurij Diego Scandola - 10709931

Academic Year:

2024 - 2025

CQ1 - How many different Confirmable PUT requests obtained an unsuccessful response from the local CoAP server?

We were able to find how many different confirmable PUT requests obtained an unsuccessful response from the local CoAP by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filters "coap.type == 0" and "coap.type == 3", respectively filtering CON and RST messages, excluding them from the set of interest ("capture" in the provided code);
- Then the code processes CoAP packets from the set, extracts message types and IDs, stores PUT request IDs in another set ("put_request"), and then reopens the capture set to filter error responses (CoAP codes ≤ 128);
- Finally, the code iterates through CoAP packets in the set, extracts the resource path and message type, and removes message IDs from put_requests if they match;

Related code:

```
1 def cq1(self):
2     print("\nCQ1: Find the number of unique confirmable PUT requests
3         which obtained an unsuccessful response")
4
5     pcap_file = self.pcap_file
6     # Dictionary to store unique message IDs for PUT requests
7     put_requests = set()
8     # Dictionary to store unique message IDs for unsuccessful
9     responses
10    unsuccessful_responses = defaultdict(set)
11    # read pcap file and filter for PUT requests
12    capture = pyshark.FileCapture(pcap_file,
13                                  display_filter="coap.type == 0 and
14                                                  coap.code == 3",
15                                  keep_packets=True
16                                  )
17    # Analyze packets to find unique message IDs for PUT requests and
18    unsuccessful responses
19    for packet in capture:
20        try:
21            if hasattr(packet, 'coap'):
22                # Check message type
23                msg_type = int(packet.coap.type)
24                mid = packet.coap.mid
25
26                put_requests.add(mid)
27
28        except Exception as e:
29            print(f"Error processing packet: {e}")
30    capture.close()
31
32    capture = pyshark.FileCapture(pcap_file,
33                                  display_filter="coap.code >= 128",
34                                  keep_packets=True
35                                  )
```

```

32     for packet in capture:
33         try:
34             if hasattr(packet, 'coap'):
35                 # Resource path
36                 resource = packet.coap.get('coap.opt.uri_path', '')
37
38                 # Check message type
39                 msg_type = int(packet.coap.type)
40                 mid = packet.coap.mid
41
42                 # remove the message id from the set if present
43                 if mid in put_requests:
44                     put_requests.remove(mid)
45
46             except Exception as e:
47                 print(f"Error processing packet: {e}")
48         capture.close()
49
50     print(f"Total number of unique confirmable PUT requests which
51           obtained an unsuccessful response:", {
52           len(put_requests)})
53     return len(put_requests)

```

Total number of unique confirmable PUT requests which obtained an unsuccessful response: 8

CQ2 - How many CoAP resources in the coap.me public server received the same number of unique Confirmable and Non Confirmable GET requests?

We were able to answer the question by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filters "coap.code == 1" and "ip.dst == 134.102.218.18", respectively filtering GET requests and selecting only packets sent to the destination IP, meaning we are analyzing GET requests directed specifically to that server;
- By iterating through the filtered packets, we checked whether each packet contained a CoAP layer and extracted the resource path ("coap.opt.uri_path"), message type ("coap.type"), and message ID ("coap.mid");
- We classified requests based on their message type: *Confirmable* (CON, type 0) and *Non-Confirmable* (NON, type 1), storing unique message IDs for each resource in two separate dictionaries ("con_requests" and "non_con_requests");
- After processing all packets, we identified CoAP resources where the number of unique Confirmable and Non-Confirmable GET requests was the same, ensuring that only resources with at least one of each request type were considered;
- Finally, we printed the list of such resources along with their unique request counts and displayed the total number of resources that met the condition.

Related code:

```
1 def cq2(self):
2     print("\nCQ2: Find resources with equal unique Confirmable and
3         Non-Confirmable GET requests")
4     pcap_file = self.pcap_file
5     #COAP IP FROM DNS: 134.102.218.18
6     # Dictionaries to store unique message IDs for resources
7     con_requests = defaultdict(set)
8     non_con_requests = defaultdict(set)
9
10    # Read the capture file
11    capture = pyshark.FileCapture(pcap_file,
12                                display_filter='coap.code == 1 and
13                                                ip.dst == 134.102.218.18', #
14                                                GET requests
15                                                keep_packets=True
16                                )
17
18    # Analyze packets
19    for packet in capture:
20        try:
21            # Check if it's a CoAP packet
22            if hasattr(packet, 'coap'):
```

```

23         # Check message type
24         msg_type = int(packet.coap.type)
25         mid = packet.coap.mid
26
27         # Track unique message IDs
28         if msg_type == 0: # Confirmable
29             con_requests[resource].add(mid)
30         elif msg_type == 1: # Non-Confirmable
31             non_con_requests[resource].add(mid)
32
33     except Exception as e:
34         print(f"Error processing packet: {e}")
35     capture.close()
36
37     # Find resources with equal number of unique Confirmable and Non-
38     # Confirmable requests
39     equal_resources = []
40     for resource in set(list(con_requests.keys()) + list(
41         non_con_requests.keys())):
42         con_count = len(con_requests.get(resource, set()))
43         non_con_count = len(non_con_requests.get(resource, set()))
44
45         if con_count == non_con_count and con_count > 0:
46             equal_resources.append((resource, con_count))
47
48     print(
49         "CoAP resources with equal unique Confirmable and Non-
50         Confirmable GET requests:")
51     for resource, count in equal_resources:
52         print(f"Resource: {resource}, Unique Request Count: {count}")
53     print(f"\nTotal number of such resources: {len(equal_resources)}")
54
55     return equal_resources

```

CoAP resources with equal unique Confirmable and Non-Confirmable GET requests:

Resource: secret, Unique Request Count: 1
Resource: validate, Unique Request Count: 1
Resource: large, Unique Request Count: 14
Total number of such resources: 3

CQ3 - How many different MQTT clients subscribe to the public broker HiveMQ using multi-level wildcards?

We were able to answer the question by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filters "mqtt.msgtype == 8 and mqtt.topic contains #" or mqtt.msgtype == 1" and "ip.dst == 18.192.151.104 or ip.dst == 35.158.43.69 or ip.dst == 35.158.34.213", respectively filtering MQTT SUBSCRIBE messages containing the wildcard topic and MQTT CONNECT messages, while selecting only packets sent to the specified destination IPs, meaning we are analyzing subscription and connection attempts to those MQTT brokers;
- We iterated through the filtered packets and identified MQTT-related packets;
- For CONNECT packets "mqtt.msgtype == 1", we extracted the "clientid" and mapped it to the corresponding TCP stream number in the "mqtt_clients" dictionary, allowing us to track unique MQTT clients;
- For SUBSCRIBE packets "mqtt.msgtype == 8", we checked whether the topic contained a multi-level wildcard and updated the corresponding client entry in "mqtt_clients", marking it as a client that subscribed to a wildcard topic;
- After processing all packets, we filtered the "mqtt_clients" dictionary to count only those clients that subscribed using the multi-level wildcard;
- Finally, we printed the list of unique MQTT clients subscribing with a wildcard and displayed the total number of such clients.

Related code:

```
1 def cq3(self):
2     # Challenge 2 CQ3: How many different MQTT clients subscribe to
3     # the public broker hivemq using multi level wildcards?
4     print("\nCQ3: How many different MQTT clients subscribe to the
5     # public broker hivemq using multi level wildcards?")
6     # hivemq ip add from dns: 35.158.43.69, 35.158.34.213,
7     # 18.192.151.104
8     pcap_file = self.pcap_file
9
10    mqtt_clients: dict = {}
11
12    try:
13        capture = pyshark.FileCapture(
14            pcap_file,
15            display_filter='(mqtt.msgtype == 8 and mqtt.topic
16            contains "#") or mqtt.msgtype == 1) and (ip.dst ==
17            18.192.151.104 or ip.dst == 35.158.43.69 or ip.dst ==
18            35.158.34.213)',
19            only_summaries=False
20        )
```

```

16         for packet in capture:
17             try:
18                 if hasattr(packet, 'mqtt'):
19                     # For CONNECT packets (msgtype 1), store the
                       clientid with IP address and possibly source
                       port
20                     if packet.mqtt.msgtype == '1': # CONNECT packet
21                         client_id = packet.mqtt.clientid
22                         tcp_stream = packet.tcp.stream # Get the TCP
                       stream number
23                         # Map (IP, source port, clientid) to track
                       unique clients
24                         mqtt_clients[tcp_stream] = (client_id, False)
25
26                     # For SUBSCRIBE packets (msgtype 8), check if the
                       topic contains a multi-level wildcard
27                     elif packet.mqtt.msgtype == '8': # SUBSCRIBE
                       packet
28                         tcp_stream = packet.tcp.stream # Get the TCP
                       stream number
29                         # Now check for all clients from this IP and
                       port that are subscribing
30                         # Mark the client as subscribing to a multi-
                       level wildcard
31                         clientid, _ = mqtt_clients[tcp_stream]
32                         mqtt_clients[tcp_stream] = (
33                             clientid, True)
34
35             except Exception as e:
36                 print(f"Error processing packet: {e}")
37
38             # Close the capture to free resources
39             capture.close()
40
41         except Exception as capture_error:
42             print(f"Error reading capture file: {capture_error}")
43             return set()
44
45         # Count the number of unique clients subscribing to multi-level
           wildcard_subscriptions
46         ret: set = [client_id for (client_id, is_sub)
47                     in mqtt_clients.values() if is_sub is True]
48         print(ret)
49         print(
50             f"Total number of different MQTT clients subscribing to the
               public broker hivemq using multi level wildcards: {len(
               ret)}")
51
52         return len(ret)

```

Total number of different MQTT clients subscribing to the public broker hivemq using multi-level wildcards: 4

CQ4 - How many different MQTT clients specify a last Will Message to be directed to a topic having as first level "university"?

We were able to answer the question by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filter "mqtt.msgtype == 1 and mqtt.willtopic", respectively filtering MQTT CONNECT messages that specify a Last Will Message, meaning we are analyzing clients that define a Last Will topic;
- We iterated through the filtered packets and identified MQTT packets containing a Last Will topic;
- For each MQTT CONNECT packet, we checked if the "willtopic" field was present and if the topic started with the keyword "university";
- If the condition was met, we extracted the "clientid" and stored it in the "mqtt_clients" dictionary, mapping each unique client to its Last Will topic;
- After processing all packets, we counted the number of unique MQTT clients that specified a Last Will topic starting with "university";
- Finally, we printed and returned the total number of such clients.

Related code:

```
1 def cq4(self):
2     # CQ4: How many different MQTT clients specify a last Will
3     #     Message to be directed to a topic having as first level "
4     #     university"?
5     print("\nCQ4: How many different MQTT clients specify a last Will
6     #     Message to be directed to a topic having as first level '
7     #     university'?")
8     capture = pyshark.FileCapture(
9         self.pcap_file,
10        display_filter='mqtt.msgtype == 1 and mqtt.willtopic',
11        only_summaries=False
12    )
13
14    mqtt_clients: dict = {}
15    for packet in capture:
16        try:
17            if hasattr(packet, 'mqtt'):
18                if packet.mqtt.willtopic and packet.mqtt.willtopic.
19                    startswith("university"):
20                    client_id = packet.mqtt.clientid
21                    will_topic = packet.mqtt.willtopic
22                    mqtt_clients[client_id] = will_topic
23
24        except Exception as e:
25            print(f"Error processing packet: {e}")
26
```



```
22     capture.close()
23     print(
24         f"Total number of different MQTT clients specifying a last
           Will Message to be directed to a topic having as first
           level 'university': {len(mqtt_clients)}")
25     return len(mqtt_clients)
```

Total number of different MQTT clients specifying a last Will Message to be directed to a topic having as first level 'university': 1

CQ5 - How many MQTT subscribers receive a last will message derived from a subscription without a wildcard?

We were able to answer the question by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filter "mqtt and mqtt.msgtype == 1 and mqtt.willtopic", meaning we are analyzing all MQTT-related packets in the capture and filtering on CONNECT packets with willtopic;
- For each MQTT CONNECT message ("msgtype == 1"), we stored the topic and last will message for later comparison;
- For each MQTT SUBSCRIBE message ("msgtype == 8"), we applied a filter on the pcap again for each last will topic without wildcards previously found and stored the client as a tuple (ip, port);
- In the end we checked which client got the last will message and stored them in a set;
- After processing all packets, we printed and returned the total number of MQTT subscribers that would receive a Last Will message on a strictly defined (non-wildcard) topic.

Related code:

```
1 def cq5(self):
2     # CQ5: How many MQTT subscribers receive a last will message derived
3     #     from a subscription without a wildcard?
4     print("\nCQ5: How many MQTT subscribers receive a last will message
5         derived from a subscription without a wildcard?")
6
7     # Open capture file
8     cap = pyshark.FileCapture(
9         self.pcap_file, display_filter="mqtt and mqtt.msgtype == 1 and
10        mqtt.willtopic")
11
12    lwt_topic_msg = {}
13    client_ids = {}
14
15    # First pass: Process CONNECT and SUBSCRIBE packets
16    for pkt in cap:
17        try:
18            # Track will message and topic if present
19            will_topic = pkt.mqtt.willtopic
20            will_message = pkt.mqtt.willmsg
21
22            if will_message:
23                lwt_topic_msg[will_topic] = will_message
24
25        except Exception as e:
26            # Handle malformed packets
27            continue
28
29    # Reset the capture to read it again from the beginning
30    cap.close()
```

```

28 print(lwt_topic_msg)
29 for topic in lwt_topic_msg.keys():
30     cap = pyshark.FileCapture(
31         self.pcap_file, display_filter=f"mqtt and mqtt.msgtype == 8
           and mqtt.topic == \"{topic}\"")
32     for pkt in cap:
33         try:
34             topic = pkt.mqtt.topic
35             if client_ids.get(topic) is None:
36                 client_ids[topic] = set()
37
38             if hasattr(pkt, 'ipv6'):
39                 client = (pkt.ipv6.src, pkt.tcp.srcport)
40             else:
41                 client = (pkt.ip.src, pkt.tcp.srcport)
42             client_ids[topic].add(
43                 client)
44         except Exception as e:
45             # Handle malformed packets
46             print(e)
47             continue
48     cap.close()
49
50 print(f"client ids: {client_ids}")
51
52 clients_receiving_lwt = set()
53 for will_topic, will_message in lwt_topic_msg.items():
54     capture = pyshark.FileCapture(
55         self.pcap_file, display_filter=f"mqtt and mqtt.msgtype == 3
           and mqtt.msg == {will_message} and mqtt.topic == {
           will_topic}")
56
57     for pkt in capture:
58         try:
59             if hasattr(pkt, 'ipv6'):
60                 client = (pkt.ipv6.dst, pkt.tcp.dstport)
61             else:
62                 client = (pkt.ip.dst, pkt.tcp.dstport)
63
64             if client not in client_ids[will_topic]:
65                 continue
66
67             clients_receiving_lwt.add((client, will_topic))
68         except Exception as e:
69             # Handle malformed packets
70             print(e)
71             continue
72
73     capture.close()
74
75 subscribers_count = len(clients_receiving_lwt)
76 print(f"Total number of MQTT subscribers receiving a last will
       message derived from a subscription without a wildcard: {
       subscribers_count}")
77 return subscribers_count
78

```

LWT Topic: university/department12/room1/temperature

LWT Topic: metaverse/room2/floor4

LWT Topic: hospital/facility3/area3

LWT Topic: metaverse/room2/room2

Total number of MQTT subscribers receiving a last will message derived from a subscription without a wildcard: 3

CQ6 - How many MQTT publish messages directed to the public broker mosquitto are sent with the retain option and use QoS "At most once"?

We were able to answer the question by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filter "mqtt.msgtype == 3 and mqtt.qos == 0 and mqtt.retain == 1 and ip.dst == 5.196.78.28", respectively filtering MQTT PUBLISH messages ("msgtype == 3"), selecting only those that use QoS level 0 ("At most once"), have the retain flag enabled, and are directed to the public Mosquitto broker ("5.196.78.28"), meaning we are analyzing retained messages sent with the lowest QoS level to this broker;
- We iterated through the filtered packets and counted the number of MQTT PUBLISH messages that matched the criteria;
- After processing all packets, we printed and returned the total number of MQTT PUBLISH messages sent to the Mosquitto broker with the retain flag enabled and using QoS "At most once" (QoS 0).

Related code:

```
1 def cq6(self):
2     # CQ6: How many MQTT publish messages directed to the public
3     #     broker mosquitto are sent with the retain option and use QoS
4     #     At most once ?
5     print("\nCQ6: How many MQTT publish messages directed to the
6     #     public broker mosquitto are sent with the retain option and
7     #     use QoS 'At most once'?")
8     # Mosquitto broker ip from dns: 5.196.78.28
9     capture = pyshark.FileCapture(
10         self.pcap_file,
11         display_filter=f'mqtt.msgtype == 3 and mqtt.qos == 0 and mqtt
12         .retain == 1 and ip.dst == 5.196.78.28',
13         only_summaries=False
14     )
15
16     count = 0
17     for packet in capture:
18         count += 1
19     print(
20         f"Total number of MQTT publish messages directed to the
21         public broker mosquitto sent with the retain option and
22         use QoS 'At most once': {count}")
23     return count
```

Total number of MQTT publish messages directed to the public broker mosquitto sent with the retain option and use QoS 'At most once': 208

CQ7 - How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine?

We were able to answer the question by creating a function that:

- By utilizing the FileCapture command from the pyshark library, we scanned the "challenge2.pcap" file, applying the filter "mqtt and tcp.port == 1885", meaning we are analyzing **MQTT messages transmitted over TCP on port 1885**, which is assumed to be used for local MQTT brokers;
- We loaded all the captured packets into memory using "capture.load_packets()", ensuring that the dataset is fully available before processing;
- We counted the total number of packets matching the filter criteria by evaluating "len(capture)", representing the number of MQTT messages sent by clients to a broker on the local machine via port 1885;
- Finally, we printed and returned the total number of MQTT messages matching these conditions.

Related code:

```
1 def cq7(self):
2     # CQ7: How many MQTT-SN messages on port 1885 are sent by the clients
3     # to a broker in the local machine?
4     print("\nCQ7: How many MQTT-SN messages on port 1885 are sent by the
5     # clients to a broker in the local machine?")
6
7     # Use 'udp.port == 1885' since MQTT-SN typically uses UDP
8     # We'll filter specifically in the code to be thorough
9     capture = pyshark.FileCapture(
10         self.pcap_file, display_filter="udp.port == 1885")
11
12     count = 0
13     # Add any other local IPs if needed
14     local_ips = ['127.0.0.1', 'localhost']
15
16     for pkt in capture:
17         # Check if it's a UDP packet with the correct port
18         if 'UDP' in pkt and int(pkt.udp.dstport) == 1885:
19             # Check if the destination is local
20             if hasattr(pkt, 'ip') and pkt.ip.dst in local_ips:
21                 # For MQTT-SN, we need to look at the payload/data
22                 # This is simplified - actual implementation would need
23                 # to parse MQTT-SN format
24                 # from the raw packet data since pyshark might not have a
25                 # built-in dissector for MQTT-SN
26                 if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and
27                     hasattr(pkt.udp, 'payload'):
28                     count += 1
29             # For IPv6
30             elif hasattr(pkt, 'ipv6') and (pkt.ipv6.dst == ':::1' or pkt.
31                 ipv6.dst in local_ips):
```

```

26         if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and
27             hasattr(pkt.udp, 'payload'):
28             count += 1
29
30     print(
31         f"Total number of MQTT-SN messages on port 1885 sent by the
           clients to a broker in the local machine: {count}")
    return count

```

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

General code

```
1 import multiprocessing
2 import pyshark
3 from collections import defaultdict
4
5
6 class Questions:
7     def __init__(self, pcap_file):
8         self.pcap_file = pcap_file
9
10    def cq1(self):
11        print("\nCQ1: Find the number of unique confirmable PUT requests
12              which obtained an unsuccessful response")
13
14        pcap_file = self.pcap_file
15        # Dictionary to store unique message IDs for PUT requests
16        put_requests = set()
17        # Dictionary to store unique message IDs for unsuccessful
18        # responses
19        unsuccessful_responses = defaultdict(set)
20        # read pcap file and filter for PUT requests
21        capture = pyshark.FileCapture(pcap_file,
22                                     display_filter="coap.type == 0 and
23                                     coap.code == 3",
24                                     keep_packets=True
25                                     )
26
27        # Analyze packets to find unique message IDs for PUT requests and
28        # unsuccessful responses
29        for packet in capture:
30            try:
31                if hasattr(packet, 'coap'):
32                    # Check message type
33                    msg_type = int(packet.coap.type)
34                    mid = packet.coap.mid
35
36                    put_requests.add(mid)
37
38            except Exception as e:
39                print(f"Error processing packet: {e}")
40        capture.close()
41
42        capture = pyshark.FileCapture(pcap_file,
43                                     display_filter="coap.code >= 128",
44                                     keep_packets=True
45                                     )
46
47        for packet in capture:
48            try:
49                if hasattr(packet, 'coap'):
50                    # Resource path
51                    resource = packet.coap.get('coap.opt.uri_path', '')
52
53                    # Check message type
54                    msg_type = int(packet.coap.type)
55                    mid = packet.coap.mid
56
57                    # remove the message id from the set if present
```



```

52         if mid in put_requests:
53             put_requests.remove(mid)
54
55     except Exception as e:
56         print(f"Error processing packet: {e}")
57     capture.close()
58
59     print(f"Total number of unique confirmable PUT requests which
60           obtained an unsuccessful response:", {
61               len(put_requests)})
62     return len(put_requests)
63
64 def cq2(self):
65     print("\nCQ2: Find resources with equal unique Confirmable and
66           Non-Confirmable GET requests")
67     pcap_file = self.pcap_file
68     #COAP IP FROM DNS: 134.102.218.18
69     # Dictionaries to store unique message IDs for resources
70     con_requests = defaultdict(set)
71     non_con_requests = defaultdict(set)
72
73     # Read the capture file
74     capture = pyshark.FileCapture(pcap_file,
75                                   display_filter='coap.code == 1 and
76                                   ip.dst == 134.102.218.18', #
77                                   GET requests
78                                   keep_packets=True
79                                   )
80
81     # Analyze packets
82     for packet in capture:
83         try:
84             # Check if it's a CoAP packet
85             if hasattr(packet, 'coap'):
86                 # Resource path
87                 resource = packet.coap.get('coap.opt.uri_path', '')
88
89                 # Check message type
90                 msg_type = int(packet.coap.type)
91                 mid = packet.coap.mid
92
93                 # Track unique message IDs
94                 if msg_type == 0: # Confirmable
95                     con_requests[resource].add(mid)
96                 elif msg_type == 1: # Non-Confirmable
97                     non_con_requests[resource].add(mid)
98
99             except Exception as e:
100                 print(f"Error processing packet: {e}")
101         capture.close()
102
103     # Find resources with equal number of unique Confirmable and Non-
104     Confirmable requests
105     equal_resources = []
106     for resource in set(list(con_requests.keys()) + list(
107         non_con_requests.keys())):
108         con_count = len(con_requests.get(resource, set()))
109         non_con_count = len(non_con_requests.get(resource, set()))

```

```

104         if con_count == non_con_count and con_count > 0:
105             equal_resources.append((resource, con_count))
106
107     print(
108         "CoAP resources with equal unique Confirmable and Non-
109         Confirmable GET requests:")
110     for resource, count in equal_resources:
111         print(f"Resource: {resource}, Unique Request Count: {count}")
112
113     print(f"\nTotal number of such resources: {len(equal_resources)}"
114           )
115
116     return equal_resources
117
118 def cq3(self):
119     # Challenge 2 CQ3: How many different MQTT clients subscribe to
120     # the public broker hivemq using multi level wildcards?
121     print("\nCQ3: How many different MQTT clients subscribe to the
122     public broker hivemq using multi level wildcards?")
123     # hivemq ip add from dns: 35.158.43.69, 35.158.34.213,
124     # 18.192.151.104
125     pcap_file = self.pcap_file
126
127     mqtt_clients: dict = {}
128
129     try:
130         capture = pyshark.FileCapture(
131             pcap_file,
132             display_filter='((mqtt.msgtype == 8 and mqtt.topic
133             contains "#") or mqtt.msgtype == 1) and (ip.dst ==
134             18.192.151.104 or ip.dst == 35.158.43.69 or ip.dst ==
135             35.158.34.213)',
136             only_summaries=False
137         )
138
139     for packet in capture:
140         try:
141             if hasattr(packet, 'mqtt'):
142                 # For CONNECT packets (msgtype 1), store the
143                 # clientid with IP address and possibly source
144                 # port
145                 if packet.mqtt.msgtype == '1': # CONNECT packet
146                     client_id = packet.mqtt.clientid
147                     tcp_stream = packet.tcp.stream # Get the TCP
148                     # stream number
149                     # Map (IP, source port, clientid) to track
150                     # unique clients
151                     mqtt_clients[tcp_stream] = (client_id, False)
152
153                 # For SUBSCRIBE packets (msgtype 8), check if the
154                 # topic contains a multi-level wildcard
155                 elif packet.mqtt.msgtype == '8': # SUBSCRIBE
156                     packet
157                     tcp_stream = packet.tcp.stream # Get the TCP
158                     # stream number
159                     # Now check for all clients from this IP and
160                     # port that are subscribing

```

```

146         # Mark the client as subscribing to a multi-
147         level wildcard
148         clientid, _ = mqtt_clients[tcp_stream]
149         mqtt_clients[tcp_stream] = (
150             clientid, True)
151
152     except Exception as e:
153         print(f"Error processing packet: {e}")
154
155     # Close the capture to free resources
156     capture.close()
157
158 except Exception as capture_error:
159     print(f"Error reading capture file: {capture_error}")
160     return set()
161
162 # Count the number of unique clients subscribing to multi-level
163 wildcard_subscriptions
164 ret: set = [client_id for (client_id, is_sub)
165              in mqtt_clients.values() if is_sub is True]
166 print(ret)
167 print(
168     f"Total number of different MQTT clients subscribing to the
169     public broker hivemq using multi level wildcards: {len(
170         ret)}")
171
172 return len(ret)
173
174 def cq4(self):
175     # CQ4: How many different MQTT clients specify a last Will
176     Message to be directed to a topic having as first level "
177     university"?
178     print("\nCQ4: How many different MQTT clients specify a last Will
179     Message to be directed to a topic having as first level '
180     university'?")
181     capture = pyshark.FileCapture(
182         self.pcap_file,
183         display_filter='mqtt.msgtype == 1 and mqtt.willtopic',
184         only_summaries=False
185     )
186
187     mqtt_clients: dict = {}
188     for packet in capture:
189         try:
190             if hasattr(packet, 'mqtt'):
191                 if hasattr(packet.mqtt, 'willtopic'):
192                     will_topic = packet.mqtt.willtopic
193                     if will_topic == "university" or will_topic.
194                         startswith("university/"):
195                         client_id = packet.mqtt.clientid
196                         will_topic = packet.mqtt.willtopic
197                         mqtt_clients[client_id] = will_topic
198
199         except Exception as e:
200             print(f"Error processing packet: {e}")
201
202     capture.close()
203     print(

```

```

195         f"Total number of different MQTT clients specifying a last
196           Will Message to be directed to a topic having as first
197           level 'university': {len(mqtt_clients)}")
198     return len(mqtt_clients)
199
200     def cq5(self):
201         # CQ5: How many MQTT subscribers receive a last will message
202           derived from a subscription without a wildcard?
203         print("\nCQ5: How many MQTT subscribers receive a last will
204           message derived from a subscription without a wildcard?")
205
206         # Open capture file
207         cap = pyshark.FileCapture(
208             self.pcap_file, display_filter="mqtt and mqtt.msgtype == 1
209             and mqtt.willtopic")
210
211         lwt_topic_msg = {}
212         client_ids = {}
213
214         # First pass: Process CONNECT and SUBSCRIBE packets
215         for pkt in cap:
216             try:
217                 # Track will message and topic if present
218                 will_topic = pkt.mqtt.willtopic
219                 will_message = pkt.mqtt.willmsg
220
221                 if will_message:
222                     lwt_topic_msg[will_topic] = will_message
223
224             except Exception as e:
225                 # Handle malformed packets
226                 continue
227
228         # Reset the capture to read it again from the beginning
229         cap.close()
230         print(lwt_topic_msg)
231         for topic in lwt_topic_msg.keys():
232             cap = pyshark.FileCapture(
233                 self.pcap_file, display_filter=f"mqtt and mqtt.msgtype ==
234                 8 and mqtt.topic == \"{topic}\"")
235             for pkt in cap:
236                 try:
237                     topic = pkt.mqtt.topic
238                     if client_ids.get(topic) is None:
239                         client_ids[topic] = set()
240
241                     if hasattr(pkt, 'ipv6'):
242                         client = (pkt.ipv6.src, pkt.tcp.srcport)
243                     else:
244                         client = (pkt.ip.src, pkt.tcp.srcport)
245                     client_ids[topic].add(
246                         client)
247                 except Exception as e:
248                     # Handle malformed packets
249                     print(e)
250                     continue
251             cap.close()

```

```

247     print(f"client ids: {client_ids}")
248
249     clients_receiving_lwt = set()
250     for will_topic, will_message in lwt_topic_msg.items():
251         capture = pyshark.FileCapture(
252             self.pcap_file, display_filter=f"mqtt and mqtt.msgtype ==
                3 and mqtt.msg == {will_message} and mqtt.topic == {
                will_topic}")
253
254         for pkt in capture:
255             try:
256                 if hasattr(pkt, 'ipv6'):
257                     client = (pkt.ipv6.dst, pkt.tcp.dstport)
258                 else:
259                     client = (pkt.ip.dst, pkt.tcp.dstport)
260
261                 if client not in client_ids[will_topic]:
262                     continue
263
264                 clients_receiving_lwt.add((client, will_topic))
265             except Exception as e:
266                 # Handle malformed packets
267                 print(e)
268                 continue
269
270         capture.close()
271
272     subscribers_count = len(clients_receiving_lwt)
273     print(f"Total number of MQTT subscribers receiving a last will
        message derived from a subscription without a wildcard: {
        subscribers_count}")
274     return subscribers_count
275
276
277 def cq6(self):
278     # CQ6: How many MQTT publish messages directed to the public
279         broker mosquitto are sent with the retain option and use QoS
        At most once ?
280     print("\nCQ6: How many MQTT publish messages directed to the
        public broker mosquitto are sent with the retain option and
        use QoS 'At most once'?")
281     # Mosquitto broker ip from dns: 5.196.78.28
282     capture = pyshark.FileCapture(
283         self.pcap_file,
284         display_filter=f'mqtt.msgtype == 3 and mqtt.qos == 0 and mqtt
            .retain == 1 and ip.dst == 5.196.78.28',
285         only_summaries=False
286     )
287
288     count = 0
289     for packet in capture:
290         count += 1
291     print(
292         f"Total number of MQTT publish messages directed to the
        public broker mosquitto sent with the retain option and
        use QoS 'At most once': {count}")
293     return count
294

```

```

295 def cq7(self):
296     # CQ7: How many MQTT-SN messages on port 1885 are sent by the
        clients to a broker in the local machine?
297     print("\nCQ7: How many MQTT-SN messages on port 1885 are sent by
        the clients to a broker in the local machine?")
298
299     # Use 'udp.port == 1885' since MQTT-SN typically uses UDP
300     # We'll filter specifically in the code to be thorough
301     capture = pyshark.FileCapture(
302         self.pcap_file, display_filter="udp.port == 1885")
303
304     count = 0
305     # Add any other local IPs if needed
306     local_ips = ['127.0.0.1', 'localhost']
307
308     for pkt in capture:
309         # Check if it's a UDP packet with the correct port
310         if 'UDP' in pkt and int(pkt.udp.dstport) == 1885:
311             # Check if the destination is local
312             if hasattr(pkt, 'ip') and pkt.ip.dst in local_ips:
313                 # For MQTT-SN, we need to look at the payload/data
314                 # This is simplified - actual implementation would
315                 # need to parse MQTT-SN format
316                 # from the raw packet data since pyshark might not
317                 # have a built-in dissector for MQTT-SN
318                 if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and
319                     hasattr(pkt.udp, 'payload'):
320                     count += 1
321                 # For IPv6
322                 elif hasattr(pkt, 'ipv6') and (pkt.ipv6.dst == '::1' or
323                     pkt.ipv6.dst in local_ips):
324                     if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and
325                         hasattr(pkt.udp, 'payload'):
326                         count += 1
327
328     print(
329         f"Total number of MQTT-SN messages on port 1885 sent by the
330         clients to a broker in the local machine: {count}")
331     return count
332
333 def main():
334     pcap_file = 'challenge2.pcapng'
335
336     # Get resources with equal Confirmable and Non-Confirmable requests
337     q = Questions(pcap_file)
338     _ = q.cq1()
339     _ = q.cq2()
340     _ = q.cq3()
341     _ = q.cq4()
342     _ = q.cq5()
343     _ = q.cq6()
344     _ = q.cq7()
345
346 if __name__ == "__main__":
347     main()

```