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:

```
def cq1(self):
        print("\nCQ1: Find the number of unique confirmable PUT requests which obtained an unsuccessful response")

        pcap_file = self.pcap_file
        # Dictionary to store unique message IDs for PUT requests
        put_requests = set()
        # Dictionary to store unique message IDs for unsuccessful responses
        unsuccessful_responses = defaultdict(set)
        # read pcap file and filter for PUT requests
        capture = pyshark.FileCapture(pcap_file,
                                      display_filter="coap.type == 0 and coap.code == 3",
                                      keep_packets=True
                                      )
        # Analyze packets to find unique message IDs for PUT requests and unsuccessful responses
        for packet in capture:
            try:
                if hasattr(packet, 'coap'):
                    # Check message type
                    msg_type = int(packet.coap.type)
                    mid = packet.coap.mid

                    put_requests.add(mid)

            except Exception as e:
                print(f"Error processing packet: {e}")
        capture.close()

        capture = pyshark.FileCapture(pcap_file,
                                      display_filter="coap.code >= 128",
                                      keep_packets=True
                                      )
```

```
        for packet in capture:
            try:
                if hasattr(packet, 'coap'):
                    # Resource path
                    resource = packet.coap.get('coap.opt.uri_path', '')

                    # Check message type
                    msg_type = int(packet.coap.type)
                    mid = packet.coap.mid

                    # remove the message id from the set if present
                    if mid in put_requests:
                        put_requests.remove(mid)

            except Exception as e:
                print(f"Error processing packet: {e}")
        capture.close()

        print(f"Total number of unique confirmable PUT requests which
            obtained an unsuccessful response:", {
              len(put_requests)})
        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:

```
def cq2(self):
        print("\nCQ2: Find resources with equal unique Confirmable and
            Non-Confirmable GET requests")
        pcap_file = self.pcap_file
        #COAP IP FROM DNS: 134.102.218.18
        # Dictionaries to store unique message IDs for resources
        con_requests = defaultdict(set)
        non_con_requests = defaultdict(set)

        # Read the capture file
        capture = pyshark.FileCapture(pcap_file,
                                      display_filter='coap.code == 1 and
                                          ip.dst == 134.102.218.18',  #
                                          GET requests
                                      keep_packets=True
                                      )

        # Analyze packets
        for packet in capture:
            try:
                # Check if it's a CoAP packet
                if hasattr(packet, 'coap'):
                    # Resource path
                    resource = packet.coap.get('coap.opt.uri_path', '')

```

```
                    # Check message type
                    msg_type = int(packet.coap.type)
                    mid = packet.coap.mid

                    # Track unique message IDs
                    if msg_type == 0:  # Confirmable
                        con_requests[resource].add(mid)
                    elif msg_type == 1:  # Non-Confirmable
                        non_con_requests[resource].add(mid)

            except Exception as e:
                print(f"Error processing packet: {e}")
        capture.close()

        # Find resources with equal number of unique Confirmable and Non-
            Confirmable requests
        equal_resources = []
        for resource in set(list(con_requests.keys()) + list(
            non_con_requests.keys())):
            con_count = len(con_requests.get(resource, set()))
            non_con_count = len(non_con_requests.get(resource, set()))

            if con_count == non_con_count and con_count > 0:
                equal_resources.append((resource, con_count))

        print(
            "CoAP resources with equal unique Confirmable and Non-
                Confirmable GET requests:")
        for resource, count in equal_resources:
            print(f"Resource: {resource}, Unique Request Count: {count}")

        print(f"\nTotal number of such resources: {len(equal_resources)}"
            )

        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:

```
def cq3(self):
        # Challenge 2 CQ3: How many different MQTT clients subscribe to
            the public broker hivemq using multi level wildcards?
        print("\nCQ3: How many different MQTT clients subscribe to the
            public broker hivemq using multi level wildcards?")
        # hivemq ip add from dns: 35.158.43.69, 35.158.34.213,
            18.192.151.104
        pcap_file = self.pcap_file

        mqtt_clients: dict = {}

        try:
            capture = pyshark.FileCapture(
                pcap_file,
                display_filter='((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)',
                only_summaries=False
            )

```

```
            for packet in capture:
                try:
                    if hasattr(packet, 'mqtt'):
                        # For CONNECT packets (msgtype 1), store the clientid with IP address and possibly source port
                        if packet.mqtt.msgtype == '1':  # CONNECT packet
                            client_id = packet.mqtt.clientid
                            tcp_stream = packet.tcp.stream  # Get the TCP stream number
                            # Map (IP, source port, clientid) to track unique clients
                            mqtt_clients[tcp_stream] = (client_id, False)

                        # For SUBSCRIBE packets (msgtype 8), check if the topic contains a multi-level wildcard
                        elif packet.mqtt.msgtype == '8':  # SUBSCRIBE packet
                            tcp_stream = packet.tcp.stream  # Get the TCP stream number
                            # Now check for all clients from this IP and port that are subscribing
                            # Mark the client as subscribing to a multi-level wildcard
                            clientid, _ = mqtt_clients[tcp_stream]
                            mqtt_clients[tcp_stream] = (
                                clientid, True)

                except Exception as e:
                    print(f"Error processing packet: {e}")

            # Close the capture to free resources
            capture.close()

        except Exception as capture_error:
            print(f"Error reading capture file: {capture_error}")
            return set()

        # Count the number of unique clients subscribing to multi-level wildcard_subscriptions
        ret: set = [client_id for (client_id, is_sub)
                    in mqtt_clients.values() if is_sub is True]
        print(ret)
        print(
            f"Total number of different MQTT clients subscribing to the public broker hivemq using multi level wildcards: {len(ret)}")

        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:

```
def cq4(self):
        # CQ4: How many different MQTT clients specify a last Will
            Message to be directed to a topic having as first level "
            university"?
        print("\nCQ4: How many different MQTT clients specify a last Will
             Message to be directed to a topic having as first level '
            university'?")
        capture = pyshark.FileCapture(
            self.pcap_file,
            display_filter='mqtt.msgtype == 1 and mqtt.willtopic',
            only_summaries=False
        )

        mqtt_clients: dict = {}
        for packet in capture:
            try:
                if hasattr(packet, 'mqtt'):
                    if packet.mqtt.willtopic and packet.mqtt.willtopic.
                        startswith("university"):
                        client_id = packet.mqtt.clientid
                        will_topic = packet.mqtt.willtopic
                        mqtt_clients[client_id] = will_topic

            except Exception as e:
                print(f"Error processing packet: {e}")

```

```
        capture.close()
        print(
            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)}")
        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 cliant 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:

```
def cq5(self):
    # CQ5: How many MQTT subscribers receive a last will message derived from a subscription without a wildcard?
    print("\nCQ5: How many MQTT subscribers receive a last will message derived from a subscription without a wildcard?")

    # Open capture file
    cap = pyshark.FileCapture(
        self.pcap_file, display_filter="mqtt and mqtt.msgtype == 1 and mqtt.willtopic")

    lwt_topic_msg = {}
    client_ids = {}

    # First pass: Process CONNECT and SUBSCRIBE packets
    for pkt in cap:
        try:
            # Track will message and topic if present
            will_topic = pkt.mqtt.willtopic
            will_message = pkt.mqtt.willmsg

            if will_message:
                lwt_topic_msg[will_topic] = will_message

        except Exception as e:
            # Handle malformed packets
            continue

    # Reset the capture to read it again from the beginning
    cap.close()
```

```
    print(lwt_topic_msg)
    for topic in lwt_topic_msg.keys():
        cap = pyshark.FileCapture(
            self.pcap_file, display_filter=f"mqtt and mqtt.msgtype == 8
                and mqtt.topic == \"{topic}\"")
        for pkt in cap:
            try:
                topic = pkt.mqtt.topic
                if client_ids.get(topic) is None:
                    client_ids[topic] = set()

                if hasattr(pkt, 'ipv6'):
                    client = (pkt.ipv6.src, pkt.tcp.srcport)
                else:
                    client = (pkt.ip.src, pkt.tcp.srcport)
                client_ids[topic].add(
                    client)
            except Exception as e:
                # Handle malformed packets
                print(e)
                continue
        cap.close()

    print(f"client ids: {client_ids}")

    clients_receiving_lwt = set()
    for will_topic, will_message in lwt_topic_msg.items():
        capture = pyshark.FileCapture(
            self.pcap_file, display_filter=f"mqtt and mqtt.msgtype == 3
                and mqtt.msg == {will_message} and mqtt.topic == {
                will_topic}")

        for pkt in capture:
            try:
                if hasattr(pkt, 'ipv6'):
                    client = (pkt.ipv6.dst, pkt.tcp.dstport)
                else:
                    client = (pkt.ip.dst, pkt.tcp.dstport)

                if client not in client_ids[will_topic]:
                    continue

                clients_receiving_lwt.add((client, will_topic))
            except Exception as e:
                # Handle malformed packets
                print(e)
                continue

        capture.close()

    subscribers_count = len(clients_receiving_lwt)
    print(f"Total number of MQTT subscribers receiving a last will
        message derived from a subscription without a wildcard: {
          subscribers_count}")
    return subscribers_count
```

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:

```
def cq6(self):
        # CQ6: How many MQTT publish messages directed to the public
            broker mosquitto are sent with the retain option and use QoS
              At most once ?
        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'?")
        # Mosquitto broker ip from dns: 5.196.78.28
        capture = pyshark.FileCapture(
            self.pcap_file,
            display_filter=f'mqtt.msgtype == 3 and mqtt.qos == 0 and mqtt
                .retain == 1 and ip.dst == 5.196.78.28',
            only_summaries=False
        )

        count = 0
        for packet in capture:
            count += 1
        print(
            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}")
        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:

```
def cq7(self):
    # CQ7: How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine?
    print("\nCQ7: How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine?")

    # Use 'udp.port == 1885' since MQTT-SN typically uses UDP
    # We'll filter specifically in the code to be thorough
    capture = pyshark.FileCapture(
        self.pcap_file, display_filter="udp.port == 1885")

    count = 0
    # Add any other local IPs if needed
    local_ips = ['127.0.0.1', 'localhost']

    for pkt in capture:
        # Check if it's a UDP packet with the correct port
        if 'UDP' in pkt and int(pkt.udp.dstport) == 1885:
            # Check if the destination is local
            if hasattr(pkt, 'ip') and pkt.ip.dst in local_ips:
                # For MQTT-SN, we need to look at the payload/data
                # This is simplified - actual implementation would need to parse MQTT-SN format
                # from the raw packet data since pyshark might not have a built-in dissector for MQTT-SN
                if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and hasattr(pkt.udp, 'payload'):
                    count += 1
            # For IPv6
            elif hasattr(pkt, 'ipv6') and (pkt.ipv6.dst == '::1' or pkt.ipv6.dst in local_ips):
```

```
                if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and
                    hasattr(pkt.udp, 'payload'):
                    count += 1

    print(
        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

```
import multiprocessing
import pyshark
from collections import defaultdict


class Questions:
    def __init__(self, pcap_file):
        self.pcap_file = pcap_file

    def cq1(self):
        print("\nCQ1: Find the number of unique confirmable PUT requests which obtained an unsuccessful response")

        pcap_file = self.pcap_file
        # Dictionary to store unique message IDs for PUT requests
        put_requests = set()
        # Dictionary to store unique message IDs for unsuccessful responses
        unsuccessful_responses = defaultdict(set)
        # read pcap file and filter for PUT requests
        capture = pyshark.FileCapture(pcap_file,
                                      display_filter="coap.type == 0 and coap.code == 3",
                                      keep_packets=True
                                      )
        # Analyze packets to find unique message IDs for PUT requests and unsuccessful responses
        for packet in capture:
            try:
                if hasattr(packet, 'coap'):
                    # Check message type
                    msg_type = int(packet.coap.type)
                    mid = packet.coap.mid

                    put_requests.add(mid)

            except Exception as e:
                print(f"Error processing packet: {e}")
        capture.close()

        capture = pyshark.FileCapture(pcap_file,
                                      display_filter="coap.code >= 128",
                                      keep_packets=True
                                      )
        for packet in capture:
            try:
                if hasattr(packet, 'coap'):
                    # Resource path
                    resource = packet.coap.get('coap.opt.uri_path', '')

                    # Check message type
                    msg_type = int(packet.coap.type)
                    mid = packet.coap.mid

                    # remove the message id from the set if present
```

```
                    if mid in put_requests:
                        put_requests.remove(mid)

            except Exception as e:
                print(f"Error processing packet: {e}")
        capture.close()

        print(f"Total number of unique confirmable PUT requests which obtained an unsuccessful response:", {len(put_requests)})
        return len(put_requests)

    def cq2(self):
        print("\nCQ2: Find resources with equal unique Confirmable and Non-Confirmable GET requests")
        pcap_file = self.pcap_file
        #COAP IP FROM DNS: 134.102.218.18
        # Dictionaries to store unique message IDs for resources
        con_requests = defaultdict(set)
        non_con_requests = defaultdict(set)

        # Read the capture file
        capture = pyshark.FileCapture(pcap_file,
                                      display_filter='coap.code == 1 and ip.dst == 134.102.218.18',  # GET requests
                                      keep_packets=True
                                      )

        # Analyze packets
        for packet in capture:
            try:
                # Check if it's a CoAP packet
                if hasattr(packet, 'coap'):
                    # Resource path
                    resource = packet.coap.get('coap.opt.uri_path', '')

                    # Check message type
                    msg_type = int(packet.coap.type)
                    mid = packet.coap.mid

                    # Track unique message IDs
                    if msg_type == 0:  # Confirmable
                        con_requests[resource].add(mid)
                    elif msg_type == 1:  # Non-Confirmable
                        non_con_requests[resource].add(mid)

            except Exception as e:
                print(f"Error processing packet: {e}")
        capture.close()

        # Find resources with equal number of unique Confirmable and Non-Confirmable requests
        equal_resources = []
        for resource in set(list(con_requests.keys()) + list(non_con_requests.keys())):
            con_count = len(con_requests.get(resource, set()))
            non_con_count = len(non_con_requests.get(resource, set()))
```

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

```
                            # Mark the client as subscribing to a multi-level wildcard
                            clientid, _ = mqtt_clients[tcp_stream]
                            mqtt_clients[tcp_stream] = (clientid, True)

                except Exception as e:
                    print(f"Error processing packet: {e}")

            # Close the capture to free resources
            capture.close()

        except Exception as capture_error:
            print(f"Error reading capture file: {capture_error}")
            return set()

        # Count the number of unique clients subscribing to multi-level wildcard_subscriptions
        ret: set = [client_id for (client_id, is_sub) in mqtt_clients.values() if is_sub is True]
        print(ret)
        print(
            f"Total number of different MQTT clients subscribing to the public broker hivemq using multi level wildcards: {len(ret)}")

        return len(ret)

    def cq4(self):
        # CQ4: How many different MQTT clients specify a last Will Message to be directed to a topic having as first level "university"?
        print("\nCQ4: How many different MQTT clients specify a last Will Message to be directed to a topic having as first level 'university'?")
        capture = pyshark.FileCapture(
            self.pcap_file,
            display_filter='mqtt.msgtype == 1 and mqtt.willtopic',
            only_summaries=False
        )

        mqtt_clients: dict = {}
        for packet in capture:
            try:
                if hasattr(packet, 'mqtt'):
                    if hasattr(packet.mqtt, 'willtopic'):
                        will_topic = packet.mqtt.willtopic
                        if will_topic == "university" or will_topic.startswith("university/"):
                            client_id = packet.mqtt.clientid
                            will_topic = packet.mqtt.willtopic
                            mqtt_clients[client_id] = will_topic

            except Exception as e:
                print(f"Error processing packet: {e}")

        capture.close()
        print(
```

```
            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)}")
        return len(mqtt_clients)

        def cq5(self):
        # CQ5: How many MQTT subscribers receive a last will message
            derived from a subscription without a wildcard?
        print("\nCQ5: How many MQTT subscribers receive a last will
            message derived from a subscription without a wildcard?")

        # Open capture file
        cap = pyshark.FileCapture(
            self.pcap_file, display_filter="mqtt and mqtt.msgtype == 1
                and mqtt.willtopic")

        lwt_topic_msg = {}
        client_ids = {}

        # First pass: Process CONNECT and SUBSCRIBE packets
        for pkt in cap:
            try:
                # Track will message and topic if present
                will_topic = pkt.mqtt.willtopic
                will_message = pkt.mqtt.willmsg

                if will_message:
                    lwt_topic_msg[will_topic] = will_message

            except Exception as e:
                # Handle malformed packets
                continue

        # Reset the capture to read it again from the beginning
        cap.close()
        print(lwt_topic_msg)
        for topic in lwt_topic_msg.keys():
            cap = pyshark.FileCapture(
                self.pcap_file, display_filter=f"mqtt and mqtt.msgtype ==
                     8 and mqtt.topic == \"{topic}\"")
            for pkt in cap:
                try:
                    topic = pkt.mqtt.topic
                    if client_ids.get(topic) is None:
                        client_ids[topic] = set()

                    if hasattr(pkt, 'ipv6'):
                        client = (pkt.ipv6.src, pkt.tcp.srcport)
                    else:
                        client = (pkt.ip.src, pkt.tcp.srcport)
                    client_ids[topic].add(
                        client)
                except Exception as e:
                    # Handle malformed packets
                    print(e)
                    continue
            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: {
274               subscribers_count}")
275         return subscribers_count
276
277
278     def cq6(self):
279         # CQ6: How many MQTT publish messages directed to the public
                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
```

```
    def cq7(self):
        # CQ7: How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine?
        print("\nCQ7: How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine?")

        # Use 'udp.port == 1885' since MQTT-SN typically uses UDP
        # We'll filter specifically in the code to be thorough
        capture = pyshark.FileCapture(
            self.pcap_file, display_filter="udp.port == 1885")

        count = 0
        # Add any other local IPs if needed
        local_ips = ['127.0.0.1', 'localhost']

        for pkt in capture:
            # Check if it's a UDP packet with the correct port
            if 'UDP' in pkt and int(pkt.udp.dstport) == 1885:
                # Check if the destination is local
                if hasattr(pkt, 'ip') and pkt.ip.dst in local_ips:
                    # For MQTT-SN, we need to look at the payload/data
                    # This is simplified - actual implementation would need to parse MQTT-SN format
                    # from the raw packet data since pyshark might not have a built-in dissector for MQTT-SN
                    if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and hasattr(pkt.udp, 'payload'):
                        count += 1
                # For IPv6
                elif hasattr(pkt, 'ipv6') and (pkt.ipv6.dst == '::1' or pkt.ipv6.dst in local_ips):
                    if hasattr(pkt, 'data') or hasattr(pkt, 'udp') and hasattr(pkt.udp, 'payload'):
                        count += 1

        print(
            f"Total number of MQTT-SN messages on port 1885 sent by the clients to a broker in the local machine: {count}")
        return count


def main():
    pcap_file = 'challenge2.pcapng'

    # Get resources with equal Confirmable and Non-Confirmable requests
    q = Questions(pcap_file)
    _ = q.cq1()
    _ = q.cq2()
    _ = q.cq3()
    _ = q.cq4()
    _ = q.cq5()
    _ = q.cq6()
    _ = q.cq7()


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