

Challenge 2

Chiara Thien Thao Nguyen Ba 10727985

April 4, 2025

1 Challenge Questions

The answers are made by using both Wireshark filters and scripts made with Python by using the Scapy library.

```
1 # Scapy import
2 from scapy.all import rdpcap, bind_bottom_up, bind_layers,
   Raw
3 from scapy.contrib.coap import CoAP
4 from scapy.layers.inet import IP, UDP, TCP
5 from scapy.layers.dns import DNSRR
6 from scapy.contrib.mqtt import MQTT, MQTTSubscribe,
   MQTTConnect, MQTTTopicQOS, MQTTPublish
7 from scapy.contrib.mqttsn import MQTTSN, MQTTSNPublish
8
9 # Overwrite ports configuration for CoAP
10 bind_layers(UDP, CoAP, sport=5683)
11 bind_layers(UDP, CoAP, dport=5683)
12
13 # Overwrite ports configuration for MQTT
14 bind_layers(TCP, MQTT, sport=1883)
15 bind_layers(TCP, MQTT, dport=1883)
16
17 # Overwrite ports configuration for MQTT-SN
18 bind_bottom_up(UDP, MQTTSN, sport=1885)
19 bind_bottom_up(UDP, MQTTSN, dport=1885)
20 bind_layers(UDP, MQTTSN, dport=1885, sport=1885)
21
22 # Read file
23 packets = rdpcap("./challenge2.pcapng")
```

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

1. I retrieve all the tokens of Confirmable PUT Requests having destination the local CoAP server using these filters:

```

1 # Set to track unique tokens of Confirmable PUT requests
2 confirmable_put_tokens = set()
3 for packet in packets:
4     if (
5         packet.haslayer(CoAP) # Check if packet has CoAP
6         layer
7         and packet[CoAP].code == 3 # PUT request
8         and packet[CoAP].type == 0 # Confirmable
9         and packet[IP].dst == "127.0.0.1" # Destination
10        IP local server
11        and packet[CoAP].token # Ensure the token
12        exists
13    )

```

The total number of unique tokens of Confirmable PUT Requests is 26.

2. I retrieve all the responses (ACKs) from the server that match the previous requests, and look for the ones that have an unsuccessful response by applying the following filters:

```

1 for packet in packets:
2     if (
3         packet.haslayer(CoAP) # Check if packet has CoAP
4         layer
5         and packet[CoAP].type == 2 # Acknowledgment
6         response
7         and packet[IP].src == local_coap_server_ip #
8         Response from server
9         and packet[CoAP].token in confirmable_put_tokens
10        # Matches previous requests
11        and packet[CoAP].code >= 128 # Error response
12        (4.xx or 5.xx codes)
13    )

```

3. By counting the number of the different matches I have got from the previous point, I retrieve a total of 22 different Confirmable PUT requests obtained an unsuccessful response from the local CoAP server. In particular, 7 ACKs are 4.04 Not Found responses and the remaining 15 are 4.05 Method Not Allowed responses.

Result: 22.

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

1. First, I retrieve the IP addresses of coap.me public server by looking at the DNS requests:

```

1 coapme = []
2 for p in packets:
3     if (p.haslayer(DNSRR) # Only DNS Resource Record
        packets
4         and p[DNSRR].type == 1 # Only records of type A
5         # Only records for the coap.me server
6         and p[DNSRR].rrname == b'coap.me.'
7     ):
8         coapme.append(p[DNSRR].rdata)
9 coapme = set(coapme)

```

The result gives only one IP address of coap.me, which is 134.102.218.18.

2. I look for CoAP GET Requests directed to the IP address of coap.me by applying these filters:

```

1 for p in packets:
2     if (
3         p.haslayer(CoAP) # Only CoAP Packets
4         and p[CoAP].code == 1 # Only GET requests
5         and p[IP].dst in coapme # Requests directed to
            coap.me
6     )

```

3. I extract the option in CoAP packet in order to retrieve the resource name and each time I update the number of occurrences for the current resource:

```

1 # Create a dictionary to store the count of each option
2 option_counts = {}
3 # Iterate over the options in the CoAP packet
4 for option in p[CoAP].options:
5     option_value = option[1]
6     # Update the count of the option in the dictionary
7     if option_value in option_counts:
8         option_counts[option_value] += 1
9     else:
10        option_counts[option_value] = 1

```

4. I save the current resource with the updated number of occurrences based on the type of CoAP message, Confirmable or Non Confirmable:

```

1 # Store the resource in the appropriate list based on
    message type
2 if p[CoAP].type == 0: # Confirmable
3     for opt, count in option_counts.items():
4         if opt in con_requests:
5             con_requests[opt] += count
6         else:
7             con_requests[opt] = count

```

```

8 elif p[CoAP].type == 1: # Non-Confirmable
9     for opt, count in option_counts.items():
10         if opt in non_con_requests:
11             non_con_requests[opt] += count
12         else:
13             non_con_requests[opt] = count

```

5. I count the number of CoAP resources where the unique Confirmable and Non-Confirmable GET requests are equal. This is done by comparing the request counts for each resource in the two separate dictionaries and incrementing the match count whenever the counts are the same:

```

1 matches = 0
2 for opt in con_requests:
3     if opt in non_con_requests
4         and con_requests[opt] == non_con_requests[opt]:
5         matches += 1

```

The output shows three results:

```

Option: b'large', Confirmable Count: 14, Non-Confirmable Count: 14
Option: b'validate', Confirmable Count: 1, Non-Confirmable Count: 1
Option: b'secret', Confirmable Count: 1, Non-Confirmable Count: 1

```

Result: 3.

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

1. First, I retrieve the IP addresses of the public broker HiveMQ by looking at the DNS requests:

```

1 hivemq = []
2 for p in packets:
3     if (
4         p.haslayer(DNSRR) # Only DNS Resource Record
5         packets
6         and p[DNSRR].type == 1 # Only records of type A
7         # Only records for the HiveMQ server
8         and p[DNSRR].rrname == b'broker.hivemq.com.'
9     ):
10         hivemq.append(p[DNSRR].rdata)
11 hivemq = set(hivemq)

```

The result gives two IP addresses for HiveMQ: 35.158.43.69 and 18.192.151.104.

2. I filter MQTT Subscribe packets that are directed to the public broker HiveMQ:

```

1 for p in packets:
2     if (
3         p.haslayer(MQTT) # Only MQTT packets
4         and p[MQTT].type == 8 # Only Subscribe packets
5         and p.haslayer(IP) # Only IP packets
6         and p[IP].dst in hivemq # Requests directed to
           HiveMQ
7     )

```

3. I extract the subscribed topic from the previously filtered packets using one of two methods:

- (a) If the packet contains a structured topic list, I decode the first topic found.
- (b) If no structured topic list is available but the packet contains raw data, I attempt to extract and decode the topic from the raw payload.

After extracting the topic, I filter only those that contain a multi-level wildcard.

```

1 found_topic = None
2 # Check if the packet has structured topic list
3 if hasattr(p[MQTTSubscribe], "topics") and isinstance(p[
   MQTTSubscribe].topics, list):
4     for topic_entry in p[MQTTSubscribe].topics:
5         if hasattr(topic_entry, "topic") and topic_entry
           .topic:
6             found_topic = topic_entry.topic.decode() if
               isinstance(topic_entry.topic, bytes)
7             else topic_entry.topic
8             break
9
10 # Check if the packet has raw data
11 if not found_topic and p.haslayer(Raw):
12     raw_data = p[Raw].load
13     found_topic = raw_data[:-1].decode()
14
15 # Check if the topic contains multi-level wildcard
16 if found_topic and "#" in found_topic:
17     unique_src_ports.add(p[IP].sport)

```

The retrieved topics that contain a multi-level wildcard are:

```

university/+/#
university/room0/room1/#
house/#
university/#
university/building2/section0/#
factory/department3/floor0/#

```

Thus, by counting the number of unique source ports, I obtain the number of MQTT clients subscribe to the public broker HiveMQ using multi-level wildcard that is 4. The port numbers are 38641, 38619, 54449 and 57863. In particular, I retrieve the following results for each multi-level topic:

Topic: university/+/#, Count: 1, Source Port: 38641
 Topic: university/room0/room1/#, Count: 1, Source Port: 38619
 Topic: house/#, Count: 1, Source Port: 54449
 Topic: university/#, Count: 1, Source Port: 38619
 Topic: university/building2/section0/#, Count: 1, Source Port: 57863
 Topic: factory/department3/floor0/#, Count: 1, Source Port: 38619

Also, by using the following filters in Wireshark:

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

I obtain the same result as shown in Figure 1.

No.	Time	Source	Destination	Protocol	Length	Info
375	5.113041615	10.0.2.15	18.192.151.104	MQTT	80	Subscribe Request (id=3) [university/+/#]
2442	13.175483992	10.0.2.15	18.192.151.104	MQTT	87	Subscribe Request (id=5) [university/room0/room1/#]
3293	20.163021204	10.0.2.15	18.192.151.104	MQTT	70	Subscribe Request (id=10) [house/#]
3363	20.224858918	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/department3/floor0/#]

Figure 1: The result of CQ3

Result: 4.

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

1. I filter MQTT packets with a Last Will Message using the following filter in Wireshark: `mqtt.willtopic`

By applying the filter, I retrieve 4 packets, but only one packet the number 4 (Source Port: 38083) has last Will Message directed to a topic having as first level "university". The other 3 packets have different topics. In particular, the retrieved last will topics are:

metaverse/room2/room2,
 university/department12/room1/temperature,
 metaverse/room2/floor4,
 hospital/facility3/area3

Result: 1.

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

Figure 2: The result of CQ4 and CQ5

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

1. As in the previous question CQ4, I retrieve the MQTT packets with a last will message using the following filter: `mqtt.willtopic`

The result shows 4 packets with the following last will topics:

```
metaverse/room2/room2,
university/department12/room1/temperature,
metaverse/room2/floor4,
hospital/facility3/area3
```

As we can notice, all the retrieved last will topics do not contain a wildcard.

2. I filter MQTT Subscribe packets where the topic matches any previously identified last will topic, using the following filters:

```
mqtt.msgtype==8 and mqtt.topic==<last_will_topic>
```

By applying these filters for each last will topic, I retrieve three packets for `university/department12/room1/temperature`. While for the other three last will topics I do not find any MQTT Subscribe packet.

3. Since the question asks for how many MQTT subscribers receive a last will message derived from a subscription without a wildcard, I count the number of different source ports in the packets. Thus, I obtain the following unique source ports: 39551, 53557 and 41789.

mqtt.msgtype == 8 && mqtt.topic == "university/department12/room1/temperature"						
No.	Time	Source	Destination	Protocol	Length	Info
121	1.083118347	:::1	:::1	MQTT	136	Subscribe Request (id=1) [university/department12/room1/temperature]
154	2.082593293	:::1	:::1	MQTT	136	Subscribe Request (id=1) [university/department12/room1/temperature]
304	4.097040463	:::1	:::1	MQTT	136	Subscribe Request (id=1) [university/department12/room1/temperature]

Figure 3: The result of CQ5

Result: 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”?

1. First, I retrieve the IP addresses of the public broker mosquitto by looking at the DNS requests:

```
1 mosquitto = []
2 for p in packets:
3     if (
4         p.haslayer(DNSRR) # Only DNS Resource Record
5         packets
6         and p[DNSRR].type == 1 # Only records of type A
7         # Only records for the mosquitto server
8         and p[DNSRR].rrname == b'test.mosquitto.org.'
9     ):
10        mosquitto.append(p[DNSRR].rdata)
11 mosquitto = set(mosquitto)
```

The result gives only one IP address for the public broker mosquitto, which is 5.196.78.28.

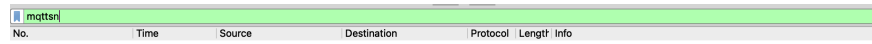
2. I filter the packets by looking for MQTT Publish packets with IP address destination of mosquitto, Quality of Service “At most once” and Retain flag set to 1.

```
1 for p in packets:
2     if (
3         p.haslayer(IP) # IP layer exists
4         and p.haslayer(MQTT) # Only MQTT packets
5         and p[IP].dst in mosquitto # IP address in
6         mosquitto
7         and p[MQTT].QOS == 0 # QoS is 0 (At most once)
8         and p[MQTT].RETAIN == 1 # Retain flag is set to
9         1
10        and p[MQTT].type == 3 # Message type is 3 (
11        PUBLISH)
12    )
```

By counting the number of packets, I retrieve a total of 208 packets with these specific conditions.

Result: 208.

CQ7) How many MQTT-SN messages on port 1885 are sent by the clients to a broker in the local machine? By applying the following filter on Wireshark, I retrieve all the MQTT-SN packets: `mqttsn`. The result in Wireshark shows 0 results, meaning that there are no MQTT-SN packets exchanged.



The image shows a Wireshark window with a filter bar at the top containing the text 'mqttsn'. Below the filter bar is a table with the following columns: No., Time, Source, Destination, Protocol, Length, and Info. The table is currently empty, indicating that no packets matching the filter were found.

No.	Time	Source	Destination	Protocol	Length	Info
-----	------	--------	-------------	----------	--------	------

Figure 4: The result of CQ7

Result: 0.