POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

IOT Challenge 2

Author: **Matteo Leonardo Luraghi**

Person Code: 10772886
Academic Year: 2024-25

Contents

1 | Introduction

To answer the following questions, I parsed the `pcapng` file using the `pyshark` library. This allowed me to filter and analyze network traffic to extract relevant information, such as MQTT and CoAP messages, along with specific attributes like topics, request types, and responses. The parsed data was then processed using Python to identify patterns, count occurrences, and match conditions required for each question.

2 | CQ 1

Question

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

To answer this question, the process is divided into two parts: processing the requests and processing the responses.

1. Processing the Confirmable PUT Requests:

I filtered the messages to identify the confirmable PUT requests sent to the local CoAP server using the following criteria:

- `coap_layer.type == 0` to filter confirmable requests
- `coap_layer.code == 3` to filter PUT requests
- `pkt.ip.dst == 127.0.0.1` to filter requests directed to the local server

For each matching request, I saved its token into a Python `set`.

2. Processing the Responses:

To process the responses, I filtered the packets to identify error responses:

- `coap_layer.code >= 128 && coap_layer.code <= 165` to filter error responses
- `token in requests` to ensure the response corresponds to a previous valid request
- `pkt.ip.src == 127.0.0.1` to confirm the response is from the local server

If a response matched all of these criteria, I added it to a Python `list`.

The result is the length of the final list which is **22 different requests**.

```
import pyshark

def answer_q1():
    print("QUESTION 1\n")
```

```
# load the file and apply the filter
packets = pyshark.FileCapture("./docs/challenge2.pcapng",
    display_filter="coap")

requests = set()
failed_responses = []

########################### Process requests
for pkt in packets:
    # skip non coap packets
    if not hasattr(pkt, "coap"):
        continue

    coap_layer = pkt.coap

    # check if the packet has a token
    if not hasattr(coap_layer, "token"):
        continue

    token = coap_layer.token

    if (
        # check if it's a confirmable request
        int(coap_layer.type) == 0
        # check if it's a PUT request
        and int(coap_layer.code) == 3
        # check if it's a request to the local server
        and pkt.ip.dst == "127.0.0.1"
    ):
        # store the request by its coap token
        requests.add(token)

########################### Process responses
for pkt in packets:
    # skip non coap packets
    if not hasattr(pkt, "coap"):
        continue

    coap_layer = pkt.coap

    # check if the packet has a Token
    if not hasattr(coap_layer, "token"):
        continue
```

```
        token = coap_layer.token

        if (
            # check if it's an error response
            int(coap_layer.code) >= 128
            and int(coap_layer.code) <= 165
            # check if the token is present in the requests
            and token in requests
            # check if the response is from the local server
            and pkt.ip.src == "127.0.0.1"
        ):
            failed_responses.append(pkt)

    print(len(failed_responses))

if __name__ == "__main__":
    answer_q1()
```

3 | CQ 2

Question

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

To answer this question, I applied a filter to capture all GET requests sent to the public coap.me server:

- `coap.code==1` to select only `GET` requests
- `ip.dst==134.102.218.18` to isolate requests directed to the `coap.me` server

Next, I used a Python dictionary to keep track of each CoAP resource, recording the number of confirmable and non-confirmable GET requests it received.

Finally, I counted how many resources had equal confirmable and non-confirmable GET requests. The result was **3 resources**.

```
import pyshark

def answer_q2():
    print("QUESTION 2\n")

    # load the file and apply the filter
    # coap.code filters only GET requests
    # ip.dst filters requests to the public coap.me server
    packets = pyshark.FileCapture("./docs/challenge2.pcapng",
        display_filter="coap and coap.code==1 and ip.dst==134.102.218.18
           ")

    resources = {}

    for pkt in packets:
        # skip non coap packets
        if not hasattr(pkt, "coap"):
            continue
```

```
        coap_layer = pkt.coap

        # check if it has the resource path
        if not hasattr(coap_layer, "opt_uri_path"):
            continue

        # get the resource and if needed initialize the data structure
        # saving the number of confirmable and non-confirmable requests
        # per resource
        resource = coap_layer.opt_uri_path
        if resource not in resources:
            resources[resource] = {"confirmable": 0, "non-confirmable":
                0}

        # check if it's a confirmable request and update the counter
        if int(coap_layer.type) == 0:
            resources[resource]["confirmable"] += 1
        # check if it's a non confirmable request and update the counter
        elif int(coap_layer.type) == 1:
            resources[resource]["non-confirmable"] += 1

    # remove resources where at least one of the number of confirmable
        or
    # non confirmable resources is zero
    filtered_resources = {}
    for resource in resources:
        if resources[resource]["confirmable"] != 0 and resources[
            resource]["non-confirmable"] != 0:
            filtered_resources[resource] = resources[resource]

    print(filtered_resources)

    # count the resources that satisfy the conditions
    count = 0
    for resource in filtered_resources:
        if filtered_resources[resource]["confirmable"] ==
            filtered_resources[resource]["non-confirmable"]:
            print(resource)
            count += 1
    print(count)

if __name__ == "__main__":
    answer_q2()
```

4 | CQ 3

Question

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

To answer this question, I applied a filter to capture all SUBSCRIBE messages directed to the public HiveMQ broker that include the multi-level wildcard (#) in the topic:

- `mqtt.msgtype==8` to select only `SUBSCRIBE` messages
- `topic contains '#'` to identify subscriptions using the multi-level wildcard
- `ip.dst==18.192.151.104` to isolate messages sent to the public HiveMQ broker

To determine how many different clients performed such subscriptions, I tracked unique combinations of client IP address and TCP port using a Python `set`.

The result was **4 distinct clients**.

```
import pyshark

def answer_q3():
    print("QUESTION 3\n")

    # load the file and apply the filter
    # mqtt.msgtype filters SUBSCRIBE messages
    # topic contains '#' filters subscriptions to topics using multi-
       level wildcards
    # ip.dst filters requests to the public HiveMQ broker
    packets = pyshark.FileCapture(
        "./docs/challenge2.pcapng",
        display_filter="mqtt and mqtt.msgtype == 8 and mqtt.topic
           contains '#' and ip.dst==18.192.151.104",
    )

    clients = set()
```

```
    for pkt in packets:
        mqtt_layer = pkt.mqtt

        print(mqtt_layer.topic)
        print(pkt.ip.src)
        print(pkt.tcp.srcport)

        # save different clients based on the IP address and the TCP
            connection port
        clients.add((pkt.ip.src, pkt.tcp.srcport))

    print(clients, len(clients))

if __name__ == "__main__":
    answer_q3()
```

5 | CQ 4

Question

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

To answer this question, I applied a filter to identify MQTT CONNECT messages that specify a last will message:

- `mqtt.msgtype==1` filters for CONNECT messages
- `mqtt.willtopic_len!=0` selects clients that define a last will topic

Then, I verified that the specified Last Will topic starts with `"university"` as the first-level topic. Using a Python `set`, I tracked the unique client identifiers that met both conditions.

As a result, I found that **only 1 distinct MQTT client** matches the conditions.

```
import pyshark

def answer_q4():
    print("QUESTION 4\n")

    # load the file and apply the filter
    # mqtt.msgtype filters CONNECT messages
    # mqtt.willtopic_len filters the clients that specify a last will
       message for a topic
    packets = pyshark.FileCapture("./docs/challenge2.pcapng",
        display_filter="mqtt and mqtt.msgtype==1 and mqtt.willtopic_len
           != 0")

    clients = set()

    for pkt in packets:
        # skip non mqtt packets
        if not hasattr(pkt, "mqtt"):
```

```
            continue

        mqtt_layer = pkt.mqtt

        # check that the last will message is for a topic
        # that has the first level "university"
        if mqtt_layer.willtopic.split("/")[0] == "university":
            # save different clients based on their client id
            clients.add(mqtt_layer.clientid)

    print(clients, len(clients))

if __name__ == "__main__":
    answer_q4()
```

6 | CQ 5

Question

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

To answer this question, I followed a the following process:

1. Iterating Over MQTT Packets:

I began by iterating over all MQTT packets to find clients' specified last will messages and their associated topics. I saved this information in a Python `dictionary`, where the keys were the topics and the values were the last will messages.

2. Counting Subscribers per Topic:

For each of these topics, I counted the number of subscribers by checking all the SUBSCRIBE messages that didn't contain any wildcard (+, #) in the subscribing topic, and stored this information in another dictionary, indexed by the topic name.

3. Matching PUBLISH Messages to Last Will Messages:

I then searched for PUBLISH messages where the topic matched one of the topics that had a last will message set. I compared the content of the PUBLISH message with the last will message defined during connection. If they matched, I added the number of subscribers for that topic to a counter. After processing, I reset the subscriber count for the topic, as the client providing the topic had failed.

The result was **3 subscribers**.

```
import pyshark

def answer_q5():
    print("QUESTION 5\n")

    # load the file and apply the filter
```

```
packets = pyshark.FileCapture("./docs/challenge2.pcapng",
    display_filter="mqtt")

subscribers = {}
last_wills = {}

############################ Find last will messages and topics
for pkt in packets:
    # skip non mqtt packets
    if not hasattr(pkt, "mqtt"):
        continue

    mqtt_layer = pkt.mqtt

    # save the last will messages in a dictionary where the keys are
        the topics
    # the messages refer to
    if hasattr(mqtt_layer, "willtopic_len") and int(mqtt_layer.
        willtopic_len) != 0:
        last_wills[mqtt_layer.willtopic] = mqtt_layer.willmsg

print(last_wills)

############################ Find subscribers without wildcard per
    topic
for pkt in packets:
    # skip non mqtt packets
    if not hasattr(pkt, "mqtt"):
        continue

    mqtt_layer = pkt.mqtt

    # filter subscriptions not containing wildcards
    if (
        # check if it's a SUBSCRIBE message
        int(mqtt_layer.msgtype) == 8
        # check that the topic doesn't contain wildcards
        and "#" not in mqtt_layer.topic
        and "+" not in mqtt_layer.topic
    ):
        if mqtt_layer.topic in last_wills:
            # add new pair (topic, number of subscribers)
            if mqtt_layer.topic not in subscribers:
                subscribers[mqtt_layer.topic] = 0
```

```
                # update the counter of subscribers per topic
                subscribers[mqtt_layer.topic] += 1

    print(subscribers)

    ########################## Find how many subscribers received a
        last will message

    receivers_of_last_will_message = 0
    for pkt in packets:
        # skip non mqtt packets
        if not hasattr(pkt, "mqtt"):
            continue

        mqtt_layer = pkt.mqtt

        if (
            # check if it's a PUBLISH message
            int(mqtt_layer.msgtype) == 3
            # check that the message is the last will associated to a
                topic
            and mqtt_layer.topic in last_wills
            and last_wills[mqtt_layer.topic] == mqtt_layer.msg
        ):
            # update the counter with the number of subscribers to the
                topic sending the last will message
            receivers_of_last_will_message += subscribers[mqtt_layer.
                topic]
            # the publisher has crashed, no need to count its
                subscribers anymore
            subscribers[mqtt_layer.topic] = 0

    print(receivers_of_last_will_message)

if __name__ == "__main__":
    answer_q5()
```

7 | CQ 6

Question

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

To answer this question, I applied the following filter to count the relevant MQTT messages:

- `mqtt.msgtype==3` to select only PUBLISH messages
- `mqtt.retain==True` to include only messages with the retain flag set
- `mqtt.qos==0` to filter messages using the "At most once" QoS level (QoS 0)
- `ip.dst==5.196.78.28` to target messages sent to the public Mosquitto broker

I found that **208 MQTT publish messages** match all these criteria.

```
import pyshark

def answer_q6():
    print("QUESTION 6\n")

    # load the file and apply the filter
    # mqtt.msgtype filters PUBLISH messages
    # mqtt.retain filters messages where the retain option is set
    # mqtt.qos filters messages with use the "at most once" QoS
    # ip.dst filters messages directed to the public mosquitto broker
    packets = pyshark.FileCapture("./docs/challenge2.pcapng",
        display_filter="mqtt.msgtype==3 and mqtt.retain==True and mqtt.
           qos == 0 and ip.dst==5.196.78.28")

    # count the messages
    count = 0
    for _ in packets:
        count += 1
    print(count)
```

```
if __name__ == "__main__":
    answer_q6()
```

8 | CQ 7

Question

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

To answer this question, I applied a filter for UDP traffic on port 1885, as MQTT-SN uses UDP and the question explicitly mentions this port. The filter returned zero matching packets, indicating that no MQTT-SN messages were sent by clients to the broker on the local machine using port 1885. Therefore, the answer is **0**.

```
import pyshark

def answer_q7():
    print("QUESTION 7\n")

    # load the file and apply the filter
    # udp.dstport filters messages sent on the port 1885
    packets = pyshark.FileCapture("./docs/challenge2.pcapng",
        display_filter="udp and udp.dstport==1885")

    count = 0
    for _ in packets:
        count += 1
    print(count)

if __name__ == "__main__":
    answer_q7()
```