TNE20003 – Internet & Cybersecurity Engineering for Applications - Project – Semester 2 July 2023

Janaka Pruthuvi Muthunayake
104315180
Swinburne University of Technology, Hawthorn, Australia.

Abstract:

This report offers a thorough analysis of the security features related to the MQTT (Mosquitto) Message Broker that is implemented in our company's Internet of Things infrastructure. For different Internet of Things (IoT) devices and clients to communicate and share data, the MQTT broker is an essential component.

I. Introduction

The topic of Internet of Things (IoT) is expanding quickly and involves connecting devices and sensors to facilitate data exchange and communication. Message Queuing Telemetry Transport, or MQTT, is a messaging protocol that is popular on the Internet of Things due to its extensive libraries, ease of use, and versatility. It is extensively used in the IoT, mobile internet, IoV, and power industries because it is particularly advantageous for devices with constrained resources and low-bandwidth networks. This report will go over how MQTT was set up for a project and how it was implemented in an Internet of Things application.

II. MQTT Setup

Devices were created as Python programs to set up MQTT, which were then integrated into the final solution. The apps received and processed requests from the broker to take an action, as well as periodically generated fictitious data to post to the MQTT broker. On the screen, requests to execute an action were displayed. To serve as the user interface for the finished product, a Python program was created to track system status and provide commands and updates to the system.

The client set up an account and deployed the MQTT broker on rule28.i4t.swin.edu.au for the project. The student ID (104315180) served as the account's username, and the password was the same. A message broker service is defined by the MQTT Protocol. After clients connect to the server and are granted authorization, they can post messages to various topics and subscribe to topics. The client will receive all messages submitted to a topic to which they have subscribed. There is an infinite subject depth supported by MQTT, and topics are hierarchical.

There are limitations on the topics that can be posted or subscribed to on the account, and the MQTT infrastructure is used by several projects. Only posts and subscriptions to the top-level topic <username>, where <username> is the

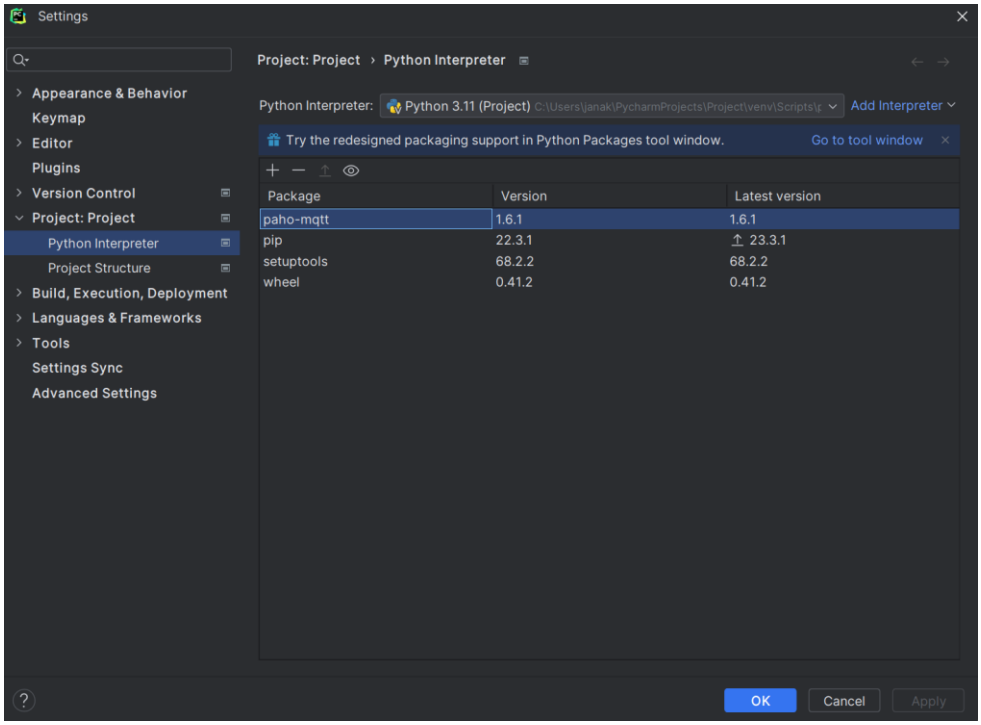

fig. 1- Installing paho-mqtt library on Python platform.

username used to log onto the server, are permitted by the user. Under this top-level topic, the user can establish as many subtopics as they wish. The user is free to create as many sub-topics beneath this top-level topic as they wish, and all projects are able to post or subscribe to it publicly.

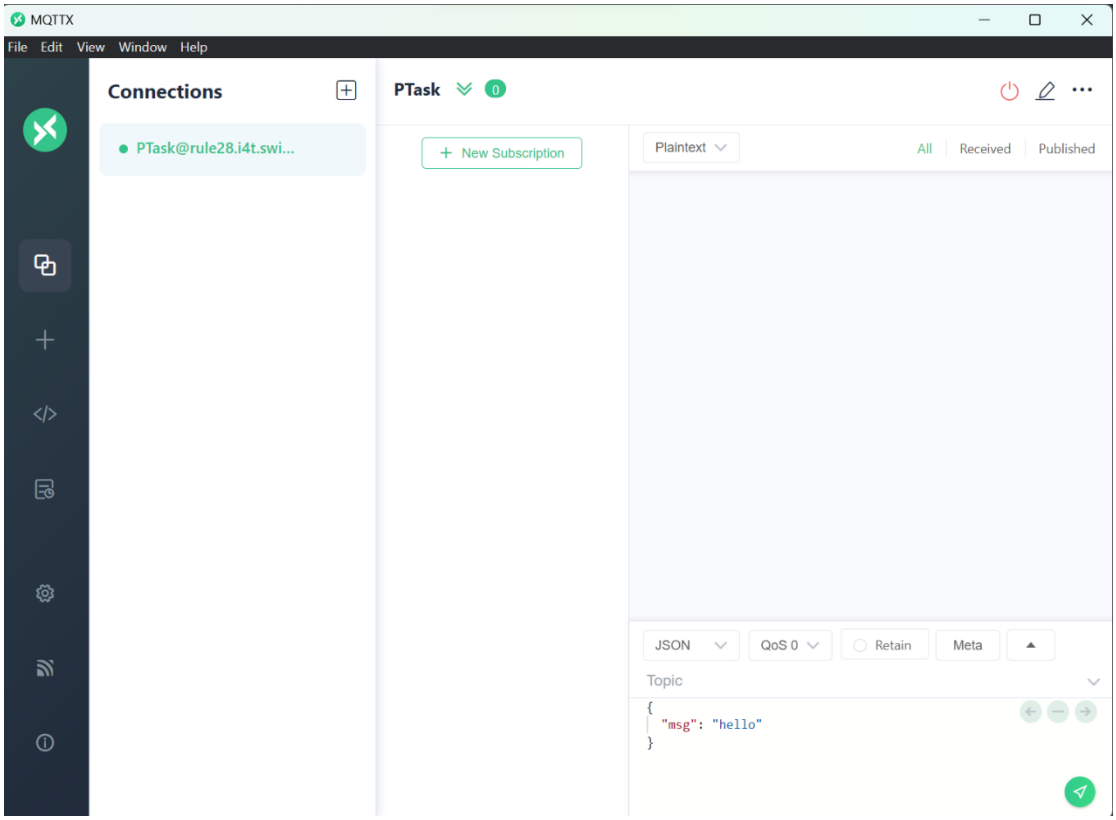

Fig.2 – MQTT Graphical Client (MQTTX)

On Graphical Client, a graphical MQTT client was downloaded in order to establish a connection to the MQTT server. To submit messages, subscribe to

topics, and establish a connection with the MQTT server, the client was utilized. To learn how the MQTT subscription process works, experiments were conducted with subscribing to wildcard topics.

III. Assessment (Pass Task)

- Publisher Client:

```
import paho.mqtt.client as mqtt
```

The Paho MQTT client library, which is needed to implement the MQTT protocol in Python.

```
import time
```

The time library, which is used to introduce a delay between the messages being published.

```
broker_address = "rule28.i4t.swin.edu.au"
```

The MQTT broker URL that the client will connect.

```
username = "104315180"
password = "104315180"
```

The username and password for the MQTT broker account that the client will use to establish a connection with the broker.

```
client = mqtt.Client("Device1")
```

By using this line, a new MQTT client object called "Device1" is created.

```
client.username_pw_set(username, password)
```

The client object has its username and password configured. This is required in order to verify the client's identity to the MQTT broker.

```
client.connect(broker_address)
```

At the designated broker_address, the client establishes a connection with the MQTT broker. The relationship between the client and the broker is established.

```
private_topic = f"{username}/private/device1"
```

The subject to which the client will publish messages is established in this line. The topic consists of the device name, the username, and the phrase "private".

```
for i in range(1, 6):
    message = f"Data from Device 1, Message {i}"
    client.publish(private_topic, message)
    time.sleep(1)
```

A loop is configured to send one to five messages at a time. Using the loop variable as the starting point, a message with a special identification is constructed. In basically, the client sends data to a designated topic by publishing the message to the private topic. It is possible to think of this as message broadcasting. There is an extra one-second pause in between every message. This controls how quickly messages are sent.

```
client.loop_start()
```

The MQTT client's networking loop begins by this line. In the background, it tells the client to start managing network communication. Maintaining the client's connection and response to incoming messages is crucial.

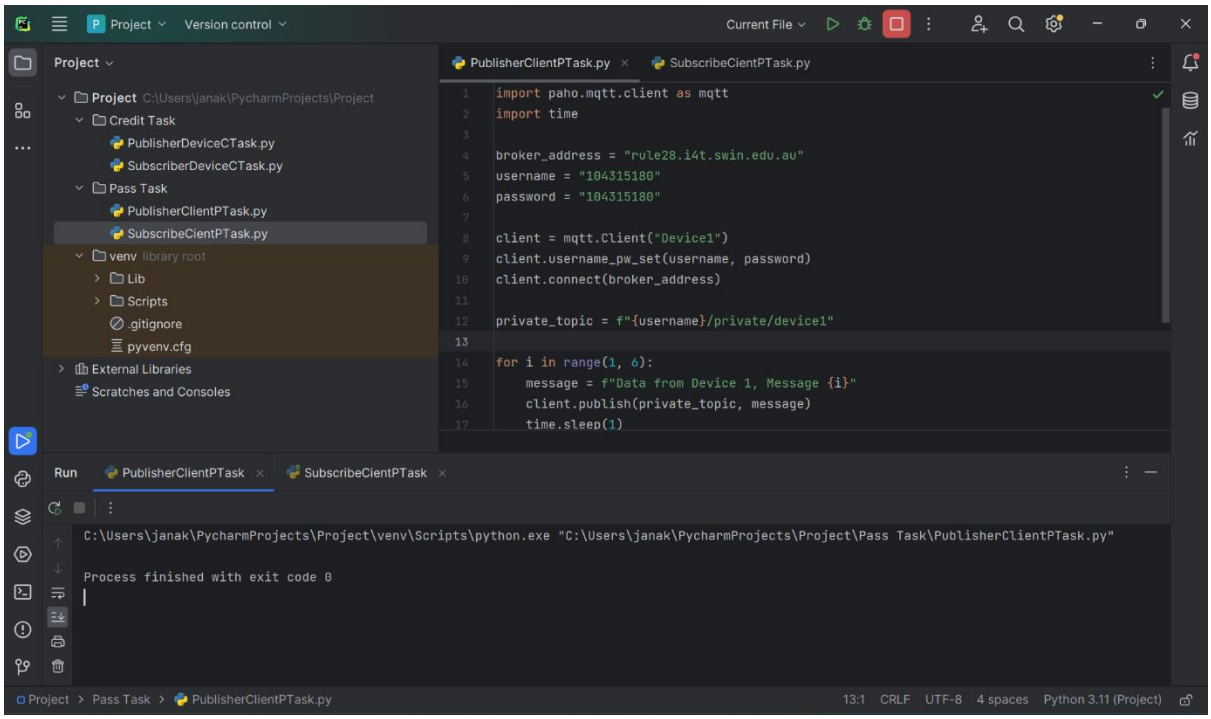

Fig.3 – Publisher client (P Task)

- Subscriber Client:

```
import paho.mqtt.client as mqtt
```

The Paho MQTT client library, which is needed to implement the MQTT protocol in Python.

```
broker_address = "rule28.i4t.swin.edu.au"
username = "104315180"
password = "104315180"
```

As in the publisher client, these lines set the authentication credentials and the address of the MQTT broker.

```
client = mqtt.Client("Device2")
```

"Device2" is the name of the constructed MQTT client. When logging in to the broker, this client's name is used to identify them.

```
client.username_pw_set(username, password)
```

For the client's authentication, a username and password have been established.

```
private_topic = f"{username}/private/device1"
public_topic = "public"
```

There are two defined topics: public_topic, which is a public topic to which other clients may subscribe, and private_topic, which is the private subject utilized by the publisher client.

```
def on_message(client, userdata, message):
    print(f"Received message on topic '{message.topic}':
{message.payload.decode()}")
```

There is a defined function called on_message. Whenever a new message is received, this function will be triggered. It prints the message that was received and the topic to which it was published.

```
client.on_message = on_message
```

The client's on_message event handler is given the on_message function. This guarantees that the function is invoked upon the arrival of fresh messages.

```
client.connect(broker_address)
```

Broker_address is provided by the client to establish a connection with the MQTT broker.

```
client.subscribe([(private_topic, 0), (public_topic, 0)])
```

The two topics that the client subscribes to are private_topic and public_topic. It will therefore receive communications pertaining to these subjects.

```
client.loop_forever()
```

The MQTT loop is started by the client to continuously handle incoming messages. While it waits for messages on subscribed topics, the client will continue to operate.

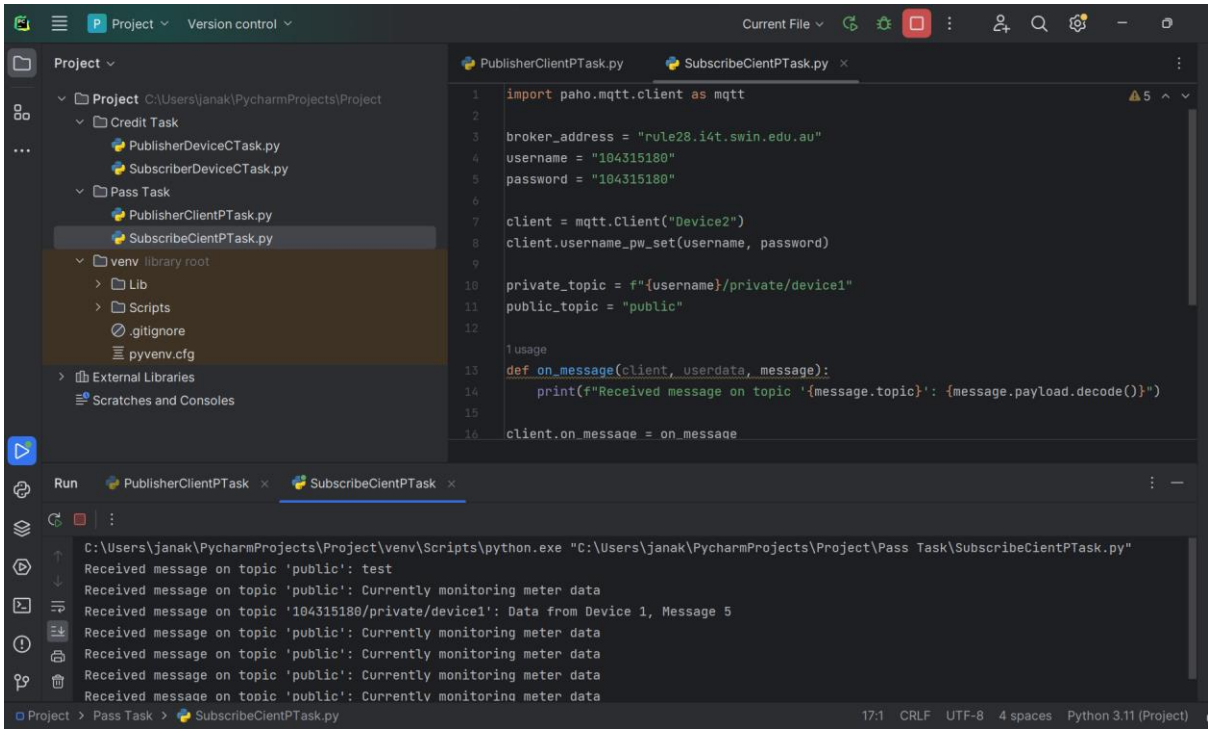

Fig.4 – Subscriber client (P Task)

IV. Assessment (Credit task)

- Publisher Device:

```
import paho.mqtt.client as mqtt
import time
```

The modules paho.mqtt.client for MQTT communication and time for time-related actions are being imported here.

```
broker_address = "rule28.i4t.swin.edu.au"
port = 1883
username = "104315180"
password = "104315180"
```

These lines configure the port, username, and password for your authentication, as well as the MQTT broker address.

```
client = mqtt.Client("PublisherDevice")
client.username_pw_set(username, password)
```

Using the username_pw_set method, you construct a MQTT client instance called "PublisherDevice" and configure its credentials.

```
client.connect(broker_address, port=port, keepalive=60)
```

Using the connect method and the broker's address, port, and keep-alive time, you connect to the MQTT broker.

```
private_topic = f"{username}/private"
public_topic = "public"
```

The private_topic and public_topic topics that you will publish to are specified on these lines.

```
for i in range(5):
    message = f"Message {i}"
    client.publish(private_topic, message)
    client.publish(public_topic, message)
    time.sleep(1)
```

Messages are published to both private_topic and public_topic using this loop. After posting a message in the pattern "Message {i}" and waiting one second for using time, it runs five times.Prior to releasing the following message, sleep(1).

```
client.disconnect()
```

Disconnect the MQTT Client.

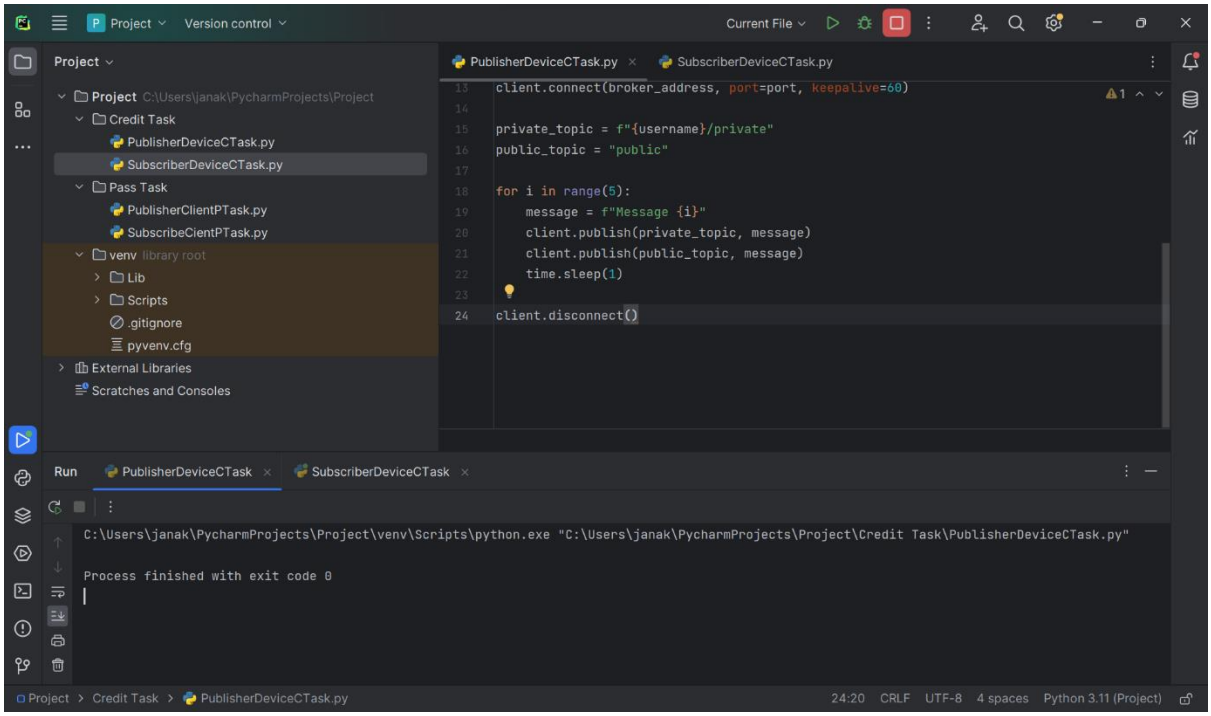

Fig.5 – Publisher Device (C Task)

- Subscriber Device:

```
import paho.mqtt.client as mqtt
```

Import the MQTT client library.

```
broker_address = "rule28.i4t.swin.edu.au"
port = 1883
username = "104315180"
password = "104315180"
```

These lines configure the port, username, and password for your authentication, as well as the MQTT broker address.

```
def on_message(client, userdata, message):
    print(f"Received message on topic {message.topic}:
{message.payload.decode()}")
```

An on_message callback function is defined by you. When a message is received, this function will be triggered. It prints the subject and the message that was received.

```
client = mqtt.Client("SubscriberDevice")
client.username_pw_set(username, password)
```

You establish a "SubscriberDevice" MQTT client instance and configure its credentials.

```
client.connect(broker_address, port=port, keepalive=60)
```

You establish a connection with the MQTT broker by providing the port, address, and keep-alive time.

```
private_topic = f"{username}/private"
```

```
public_topic = "public"
```

You specify the private and public topics that you wish to subscribe to.

```
client.subscribe(private_topic)
client.subscribe(public_topic)
```

Using the subscribe technique, you can subscribe to the MQTT topics. You are subscribing to both the public_topic and the private_topic in this instance.

```
client.on_message = on_message
```

To handle received messages, you designate the on_message function, which was defined earlier as the callback. It means that the on_message function will be triggered to handle messages received on any of the subscribed topics.

```
client.loop_forever()
```

With this command, the main loop of the MQTT client begins to listen for messages endlessly. The on_message callback method is triggered upon receiving a message, processing and printing the message along with its subject.

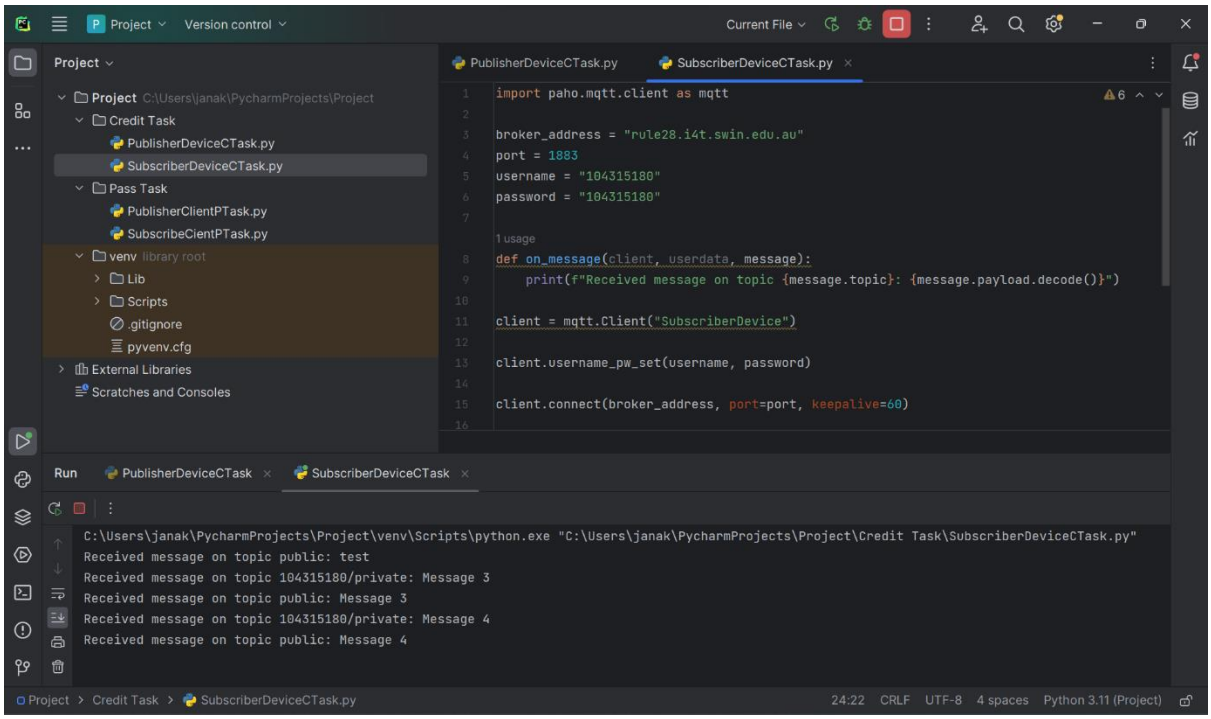

Fig.6 – Subscriber Device (C Task)

V. Security Issues

One popular communication protocol for IoT and IIoT implementations is MQTT. But it is not impervious to security problems.

1. **Lack of Transport Encryption**: MQTT is a publish-subscribe protocol that allows data to be transmitted in clear text, making it readable by anyone able to intercept it. This is a serious security vulnerability since it makes private information available to possible attackers. TLS encryption is supported by MQTT, but many MQTT brokers are deployed without it because it is not enabled by default.

2. **Weak Authentication**: Before allowing a connection, MQTT brokers may ask a client to provide a working login and password. Nevertheless, without transport encryption, the login and password combination is sent in clear text and is not secure. This implies that the login and password can be accessed by anyone who intercepts the data. Furthermore, a lot of MQTT brokers utilize weak passwords that are simple to decipher or brute-force.
3. **Lack of Access Control**: MQTT brokers frequently do not provide access control features that limit access to specific clients or topics. This could jeopardize the system's integrity and allow unwanted access to private information.
4. **Insecure Broker Configuration**: Access to specific topics can be granted or denied by MQTT brokers. Unfortunately, a lot of MQTT brokers are set up insecurely, making it possible for anybody to publish content or subscribe to any topic. This could jeopardize the system's integrity and allow unwanted access to private information.
5. **Vulnerabilities in Design and Implementation**: MQTT has flaws in its design and implementation that could give hackers access to undermine the apps and systems that use it. For instance, a design flaw could allow a malicious client to provide false information, overloading clients with repeatedly the same message.

VI. Conclusion

In conclusion, The MQTT (Mosquitto) Message Broker is utilized by the project to facilitate communication between components by utilizing the pre-existing IoT infrastructure. The paper outlines the security flaws in the MQTT broker's present implementation and offers recommended techniques for protecting MQTT connections. MQTT connections may be safeguarded and a high degree of security can be applied to the project by adhering to these best practices.

VII. References

1. https://aws.amazon.com/what-is/mqtt/
2. https://developer.electricimp.com/api/mqtt/mqttclient/connect
3. https://help.ivanti.com/wl/help/en_US/VScript/1.2/MQTTClient_publish.htm
4. https://www.trendmicro.com/vinfo/es/security/news/internet-of-things/mqtt-and-coap-security-and-privacy-issues-in-iot-and-iiot-communication-protocols
5. http://www.steves-internet-guide.com/mqtt-security-mechanisms/

*Python codes has been attached to the submission document.