

# TNE20003 – Internet and Cybersecurity for Engineering Applications Project Report

Student Name: Harry  
Student ID: 103517299

## **Abstract**

This project report documents the development of an Internet of Things (IoT) prototype leveraging MQTT (Mosquitto) for communication. The project consists of tasks spanning Pass, Credit, Distinction, and High Distinction levels. In the Pass Task, basic MQTT-based communication was established with two client devices, while the Credit Task extended the functionality and introduced a cybersecurity analysis. The Distinction Task further enriched the IoT ecosystem with advanced features and diversified devices. The report also outlines our vision for the High Distinction Task, including complex features, external service integration, and a user dashboard. Throughout, we emphasize the importance of security in the MQTT infrastructure. This report showcases the evolution of the IoT project, demonstrating the potential of MQTT in engineering applications.

## **Introduction**

The Internet of Things (IoT) has become a transformative technology in various engineering applications, offering efficient data exchange and communication. This project focuses on the development of an IoT prototype using MQTT as the communication protocol. The project is structured into four tasks: Pass, Credit, Distinction, and High Distinction, each building upon the previous one to demonstrate the increasing capabilities of the IoT ecosystem.

In the Pass Task, we established the foundation by creating two client devices, Device1 and Device2, with specific roles in data generation and message reception. MQTT was utilized to enable communication between these devices. The Credit Task expanded the IoT system's capabilities by introducing Device3, creating a Python-based user interface, and conducting a cybersecurity analysis of the MQTT broker.

The Distinction Task, the focus of this report, further enriched the IoT ecosystem by introducing three distinct devices, each generating different types of data and posting messages to private and public topics. Additionally, a Python-based user interface was implemented to enhance user interaction with the IoT system. The report presents a comprehensive cybersecurity analysis, emphasizing the importance of securing the MQTT broker infrastructure.

## Pass Task

### Task Description:

The Pass task involves implementing a basic MQTT-based communication system with two client devices, where one device generates data and posts messages to a private topic, while the other device subscribes to the public channel, printing public messages along with their sub-topic information. This task is the foundation of a more extensive Internet of Things (IoT) project.

### Implementation :

#### 1. Client Device 1 (Device1):

Device1 is responsible for generating data (in this case, simulated temperature readings) and posting these messages to a private MQTT topic. To achieve this, the code employs the Paho MQTT library for Python.

**MQTT Configuration:** The code begins by configuring MQTT broker details, including the server address, port, username, and password.

**Data Generation:** Device1 simulates data generation in the form of temperature readings. In this example, random temperature values between 20°C and 30°C are generated.

**Publishing Messages:** Device1 uses MQTT to publish generated data to a private topic (103517299/private\_topic) on the MQTT broker.

**Endless Loop:** The device operates in an infinite loop, periodically publishing temperature data every 10 seconds.

#### 2. Client Device 2 (Device2):

Device2 complements Device1 by subscribing to the public MQTT channel and displaying any public messages it receives.

**MQTT Configuration:** Device2 also configures the MQTT broker details.

**Message Reception:** A callback function (`on_message``) is defined to handle received messages. When a message is received, it prints the topic and payload to the screen.

**Subscriptions:** Device2 subscribes to both the private topic and the public topic using MQTT. The subscription to public topics is achieved through the use of the '#' wildcard character.

**Client Loop:** The client device starts an MQTT client loop to continuously listen for incoming messages.

## Demonstration:

To verify the functionality of the system, a graphical MQTT client is used to simulate the exchange of messages between Device1 and Device2. The graphical MQTT client acts as an external entity that can both receive messages from Device1 and send messages to Device1.

## Source Code and Instructions:

The screenshots for both Device1 and Device2 has been provided along with instructions on how to run them. The code is written in Python, making it accessible and easy to understand.

The Pass task demonstrates the creation of two MQTT client devices with distinct roles in an IoT system. Device1 generates data and publishes messages to a private topic, while Device2 subscribes to a public topic and processes incoming messages. This basic implementation provides the foundation for subsequent tasks and represents an essential step in developing a functional IoT solution.

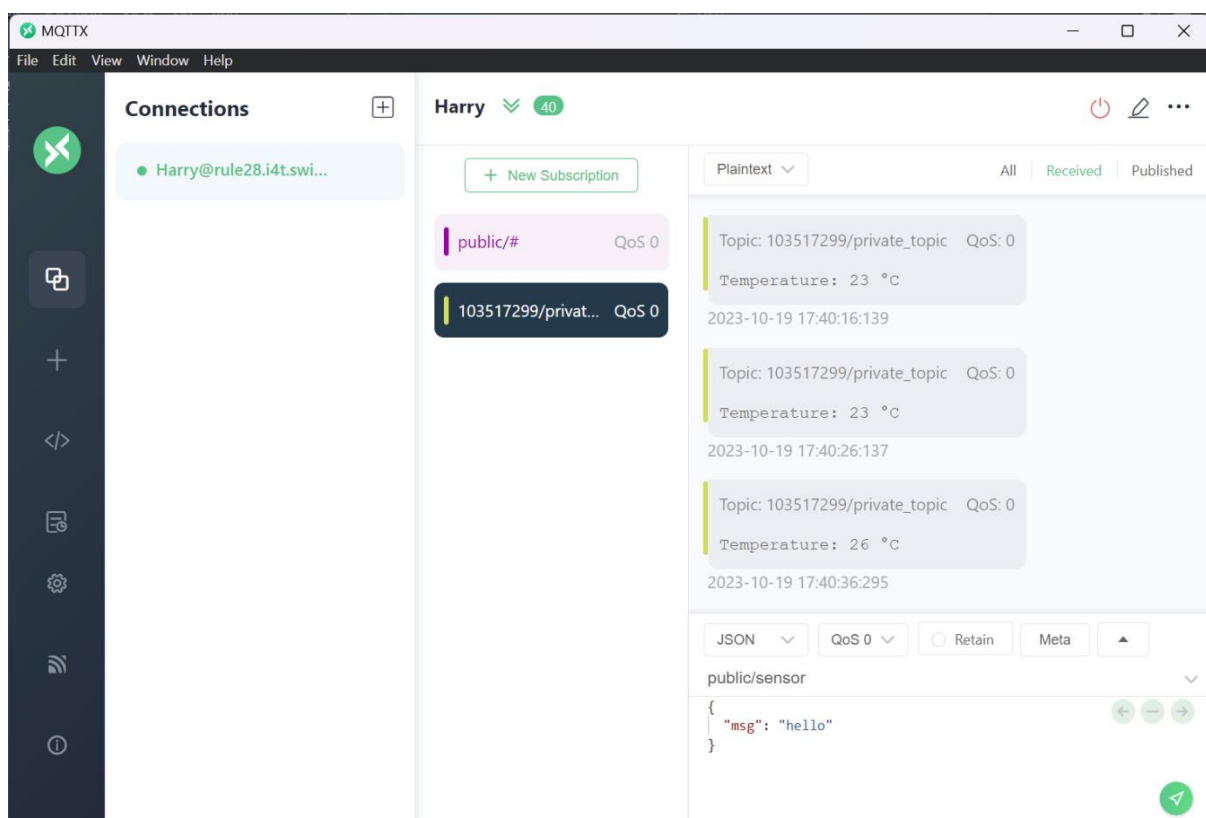


Fig1:PassTaskPic1

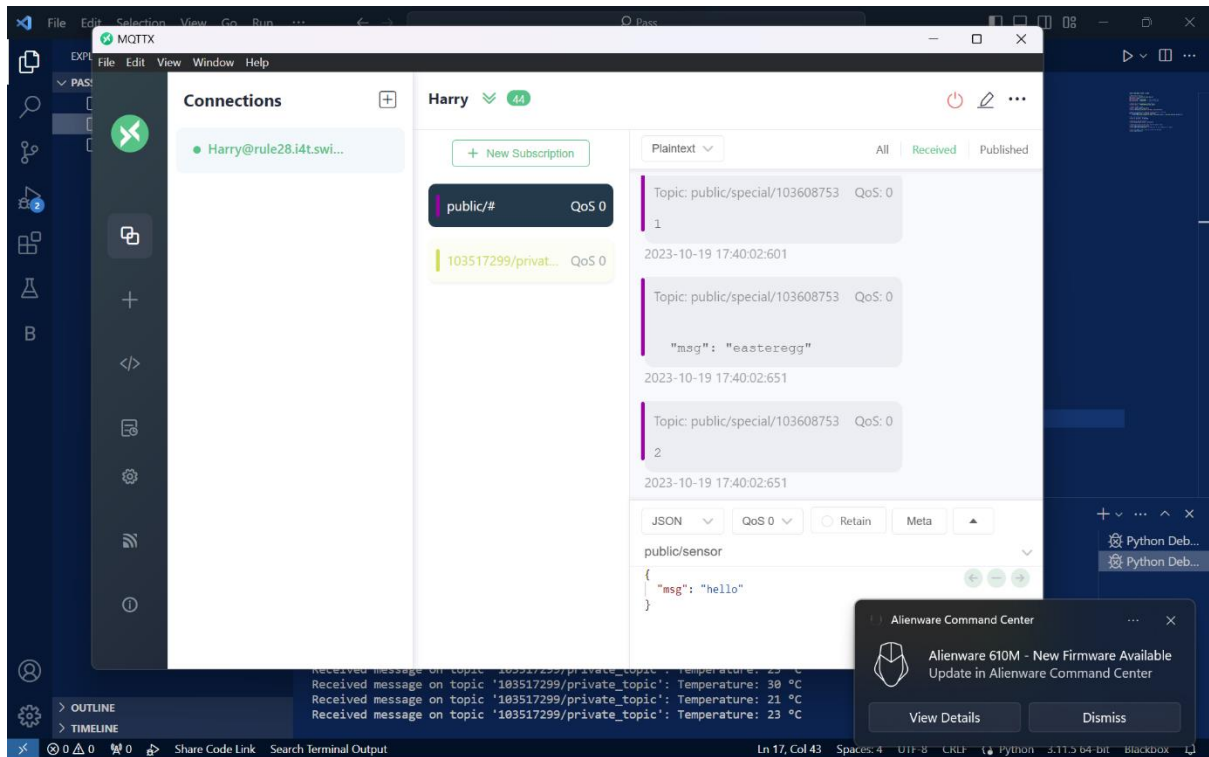


Fig2:PassTaskPic2

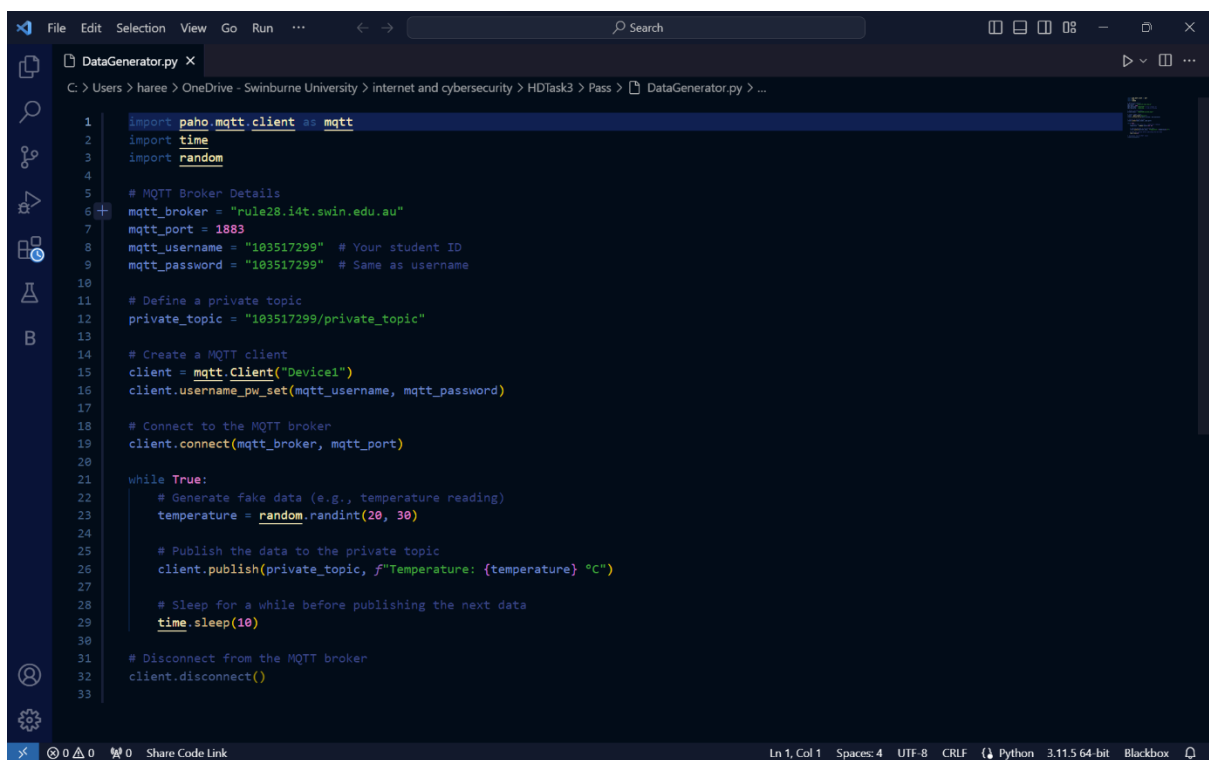
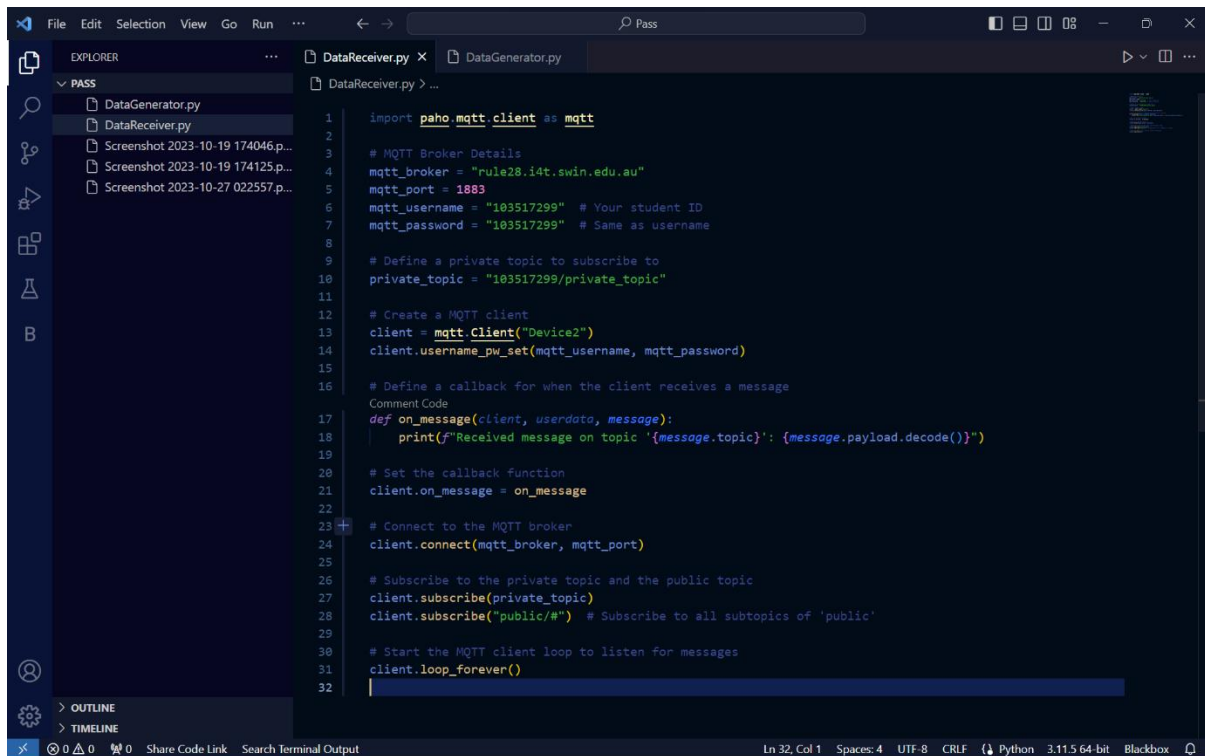


Fig3:PassTaskPic

A screenshot of a Visual Studio Code editor window. The Explorer pane on the left shows a project named 'PASS' with files 'DataGenerator.py' and 'DataReceiver.py'. The main editor area displays the code for 'DataReceiver.py'. The code is a Python script that imports 'paho.mqtt.client' as 'mqtt', defines MQTT broker details (host, port, username, password), creates a private topic, sets up a client, defines a callback function 'on\_message', connects to the broker, subscribes to the private topic and all subtopics of 'public', and starts the client loop.

```
1 import paho.mqtt.client as mqtt
2
3 # MQTT Broker Details
4 mqtt_broker = "rule28.i4t.swin.edu.au"
5 mqtt_port = 1883
6 mqtt_username = "103517299" # Your student ID
7 mqtt_password = "103517299" # Same as username
8
9 # Define a private topic to subscribe to
10 private_topic = "103517299/private_topic"
11
12 # Create a MQTT client
13 client = mqtt.Client("Device2")
14 client.username_pw_set(mqtt_username, mqtt_password)
15
16 # Define a callback for when the client receives a message
17 # Comment Code
18 def on_message(client, userdata, message):
19     print(f"Received message on topic '{message.topic}': {message.payload.decode()}")
20
21 # Set the callback function
22 client.on_message = on_message
23
24 # Connect to the MQTT broker
25 client.connect(mqtt_broker, mqtt_port)
26
27 # Subscribe to the private topic and the public topic
28 client.subscribe(private_topic)
29 client.subscribe("public/#") # Subscribe to all subtopics of 'public'
30
31 # Start the MQTT client loop to listen for messages
32 client.loop_forever()
```

Fig4:PassTaskPic

## Credit Task

In response to the client's requirements for an Internet of Things (IoT) prototype, we have developed a solution leveraging MQTT (Mosquitto) for efficient message broker communication. This report covers the Credit Task, an extension of the initial Pass Task, which demonstrates advanced features and communication capabilities within the IoT infrastructure. Additionally, we provide a cybersecurity analysis of the existing MQTT broker deployed by the client.

### Implementation:

#### Device1: Data Generation and Communication

One key component of the Credit Task is Device1, which performs the following functionalities:

**1. Data Generation:** Device1 generates simulated data, specifically temperature readings, within a range of 20°C to 30°C.

**2. Private Topic Communication:** Device1 posts the generated data to a private topic, '103517299/private\_topic,' in the MQTT broker.

**3. Public Topic Communication:** Device1 also publishes the generated temperature data to a public topic, 'public/device1.'

**4. Message Subscription:** Device1 subscribes to the private topic, '103517299/private\_topic,' to receive messages. Furthermore, it subscribes to the public topic, 'public/#,' allowing it to access public messages from various sources.

**5. Real-Time Communication:** A graphical MQTT client is used to demonstrate the instant receipt of public messages by Device1, which then displays these messages in a timely manner. This feature ensures real-time communication within the IoT system.

## **Device2: Message Reception and Communication**

Device2 complements the IoT by subscribing to the same private topic as Device1. It fulfills the following tasks:

**1. Message Subscription:** Device2 subscribes to the private topic, '103517299/private\_topic,' and the public topic, 'public/#,' enabling it to receive and access messages.

**2. Real-Time Communication:** Device2 is responsive to real-time messages published on the private and public topics, providing a two-way communication mechanism within the system.

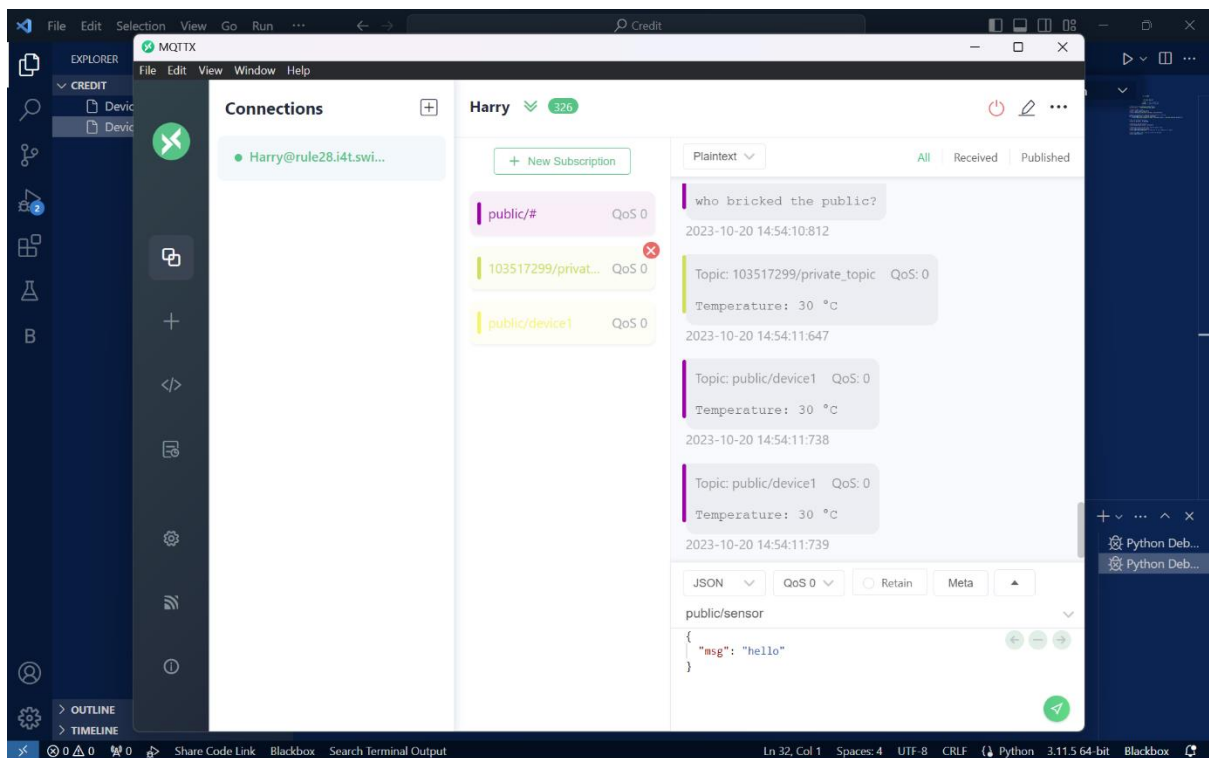


Fig5:CreditTaskPic

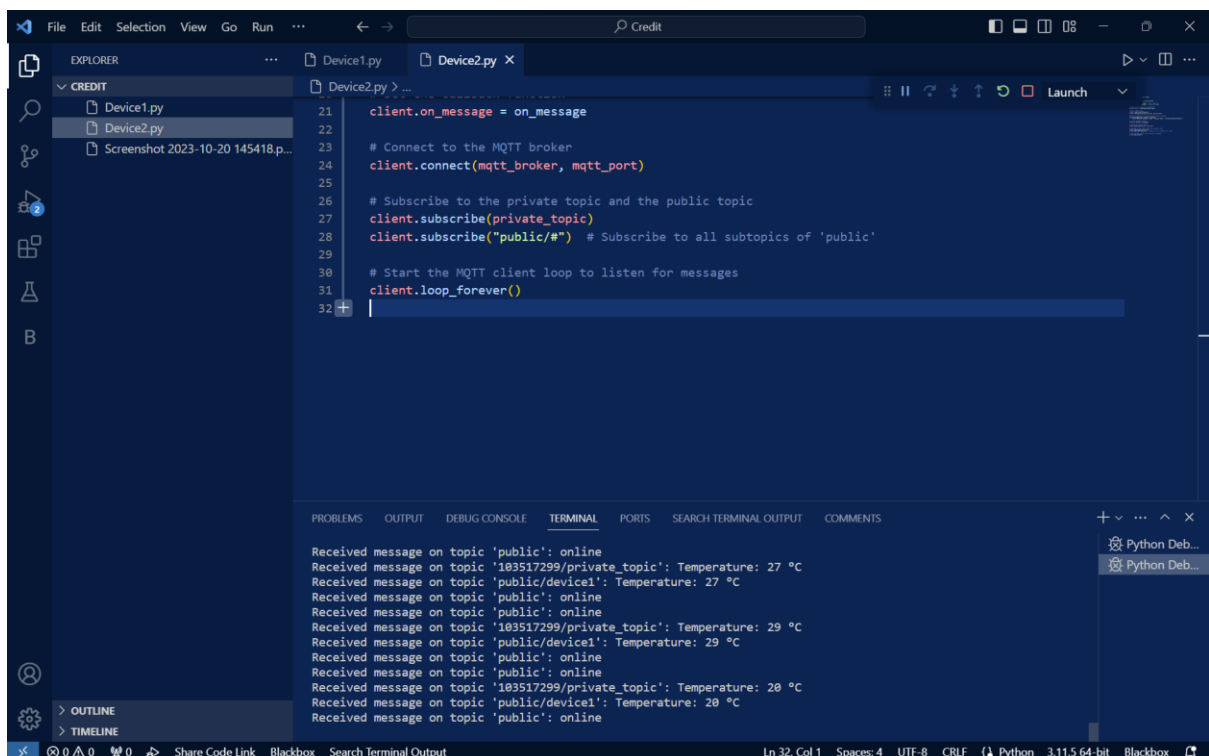
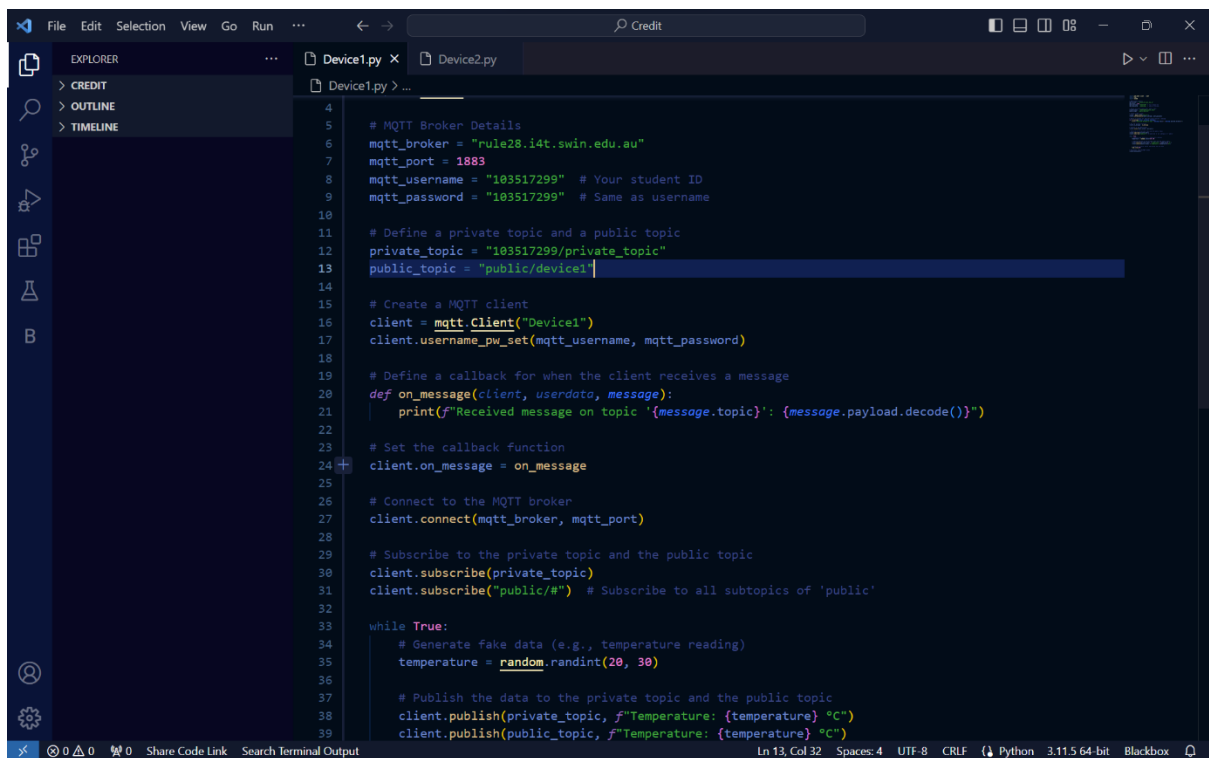


Fig6:CreditTaskPic



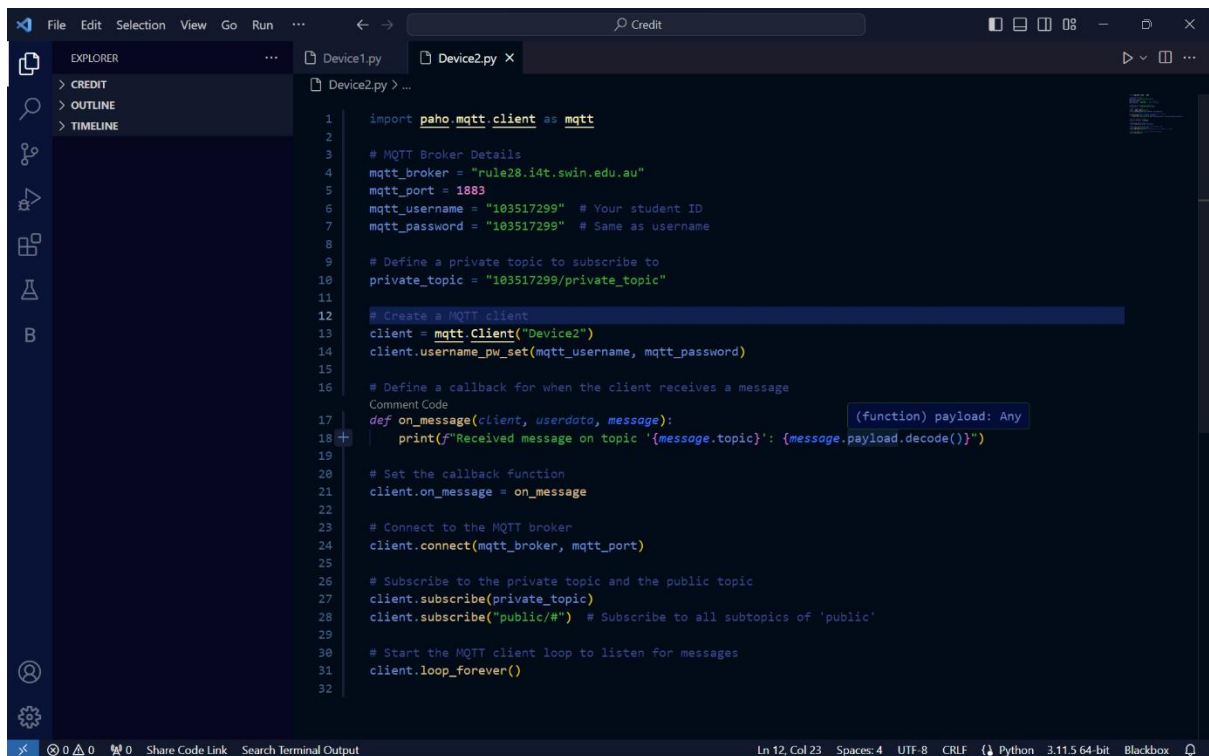


The screenshot shows the Visual Studio Code editor with the file `Device1.py` open. The Explorer sidebar on the left shows a project structure with `CREDIT`, `OUTLINE`, and `TIMELINE` folders. The code in `Device1.py` is as follows:

```
4
5 # MQTT Broker Details
6 mqtt_broker = "rule28.i4t.swin.edu.au"
7 mqtt_port = 1883
8 mqtt_username = "103517299" # Your student ID
9 mqtt_password = "103517299" # Same as username
10
11 # Define a private topic and a public topic
12 private_topic = "103517299/private_topic"
13 public_topic = "public/device1"
14
15 # Create a MQTT client
16 client = mqtt.Client("Device1")
17 client.username_pw_set(mqtt_username, mqtt_password)
18
19 # Define a callback for when the client receives a message
20 def on_message(client, userdata, message):
21     print(f"Received message on topic '{message.topic}': {message.payload.decode()}")
22
23 # Set the callback function
24 client.on_message = on_message
25
26 # Connect to the MQTT broker
27 client.connect(mqtt_broker, mqtt_port)
28
29 # Subscribe to the private topic and the public topic
30 client.subscribe(private_topic)
31 client.subscribe("public/#") # Subscribe to all subtopics of 'public'
32
33 while True:
34     # Generate fake data (e.g., temperature reading)
35     temperature = random.randint(20, 30)
36
37     # Publish the data to the private topic and the public topic
38     client.publish(private_topic, f"Temperature: {temperature} °C")
39     client.publish(public_topic, f"Temperature: {temperature} °C")
```

The status bar at the bottom indicates the cursor is at line 13, column 32, with 4 spaces, UTF-8 encoding, CRLF line endings, Python 3.11.5 64-bit, and the Blackbox theme.

Fig7:CreditTaskPic



The screenshot shows the Visual Studio Code editor with the file `Device2.py` open. The Explorer sidebar on the left shows the same project structure. The code in `Device2.py` is as follows:

```
1 import paho.mqtt.client as mqtt
2
3 # MQTT Broker Details
4 mqtt_broker = "rule28.i4t.swin.edu.au"
5 mqtt_port = 1883
6 mqtt_username = "103517299" # Your student ID
7 mqtt_password = "103517299" # Same as username
8
9 # Define a private topic to subscribe to
10 private_topic = "103517299/private_topic"
11
12 # Create a MQTT client
13 client = mqtt.Client("Device2")
14 client.username_pw_set(mqtt_username, mqtt_password)
15
16 # Define a callback for when the client receives a message
17 # Comment Code
18 def on_message(client, userdata, message):
19     print(f"Received message on topic '{message.topic}': {message.payload.decode()}")
20
21 # Set the callback function
22 client.on_message = on_message
23
24 # Connect to the MQTT broker
25 client.connect(mqtt_broker, mqtt_port)
26
27 # Subscribe to the private topic and the public topic
28 client.subscribe(private_topic)
29 client.subscribe("public/#") # Subscribe to all subtopics of 'public'
30
31 # Start the MQTT client loop to listen for messages
32 client.loop_forever()
```

The status bar at the bottom indicates the cursor is at line 12, column 23, with 4 spaces, UTF-8 encoding, CRLF line endings, Python 3.11.5 64-bit, and the Blackbox theme.

Fig8:CreditTaskPic

## Cybersecurity Analysis

The existing MQTT broker deployed by the client offers an efficient and reliable communication platform for various projects. However, it is essential to analyze and address potential security issues to ensure the integrity, confidentiality, and availability of the IoT infrastructure.

### MQTT Broker Security Assessment

- 1. Access Control:** The client's MQTT broker restricts access from off-site locations, enhancing security. However, a comprehensive user access management system should be in place, ensuring only authorized users and devices can interact with the broker.
- 2. Encryption:** Employing secure communication protocols such as TLS/SSL is crucial to encrypt data in transit, safeguarding it from eavesdropping and interception.
- 3. Authentication:** Strong authentication mechanisms must be implemented for user and device identification. Utilizing unique usernames and passwords should be complemented with advanced authentication methods like client certificates.
- 4. Authorization:** The MQTT broker should enforce strict access control lists (ACLs) to authorize users and devices to publish or subscribe to specific topics, minimizing the risk of unauthorized data access.
- 5. Secure Message Broker Configuration:** Regular updates and patches to the broker software should be applied to address known vulnerabilities and security gaps.
- 6. Monitoring and Intrusion Detection:** Implementing monitoring and intrusion detection systems can help identify and respond to suspicious activities or breaches promptly.
- 7. Secure Public Channels:** Public channels should be reviewed to ensure they do not expose sensitive information inadvertently.

The Credit Task demonstrates the advanced capabilities of Device1 and Device2 within the IoT prototype, showcasing real-time communication and message exchange. Furthermore, the

cybersecurity analysis underscores the importance of securing the MQTT broker infrastructure to protect the IoT ecosystem from potential threats.

### **Distinction Task**

In continuation of our IoT prototype project, we have successfully accomplished the Distinction Task, which builds upon the Pass and Credit tasks. This task further enriches our IoT ecosystem with advanced features and functionalities. In addition to this, a cybersecurity report for the existing MQTT broker infrastructure is presented.

#### **Implementation:**

Device1, Device2, and Device3

In the Distinction Task, we have extended the capabilities of our IoT prototype, introducing three distinct devices: Device1, Device2, and Device3. These devices subscribe to different private topics and respond to various requests/posts, providing diversified functionalities within the IoT infrastructure.

#### **1. Device1:**

- Subscribes to '103517299/devices/device1/private.'
- Generates and publishes temperature data.
- Publishes data to both the private and public topics.

#### **2. Device2:**

- Subscribes to '103517299/devices/device2/private.'
- Generates and publishes humidity data.
- Publishes data to both the private and public topics.

#### **3. Device3:**

- Subscribes to '103517299/devices/device3/private.'
- Generates and publishes pressure data.

- Publishes data to both the private and public topics.

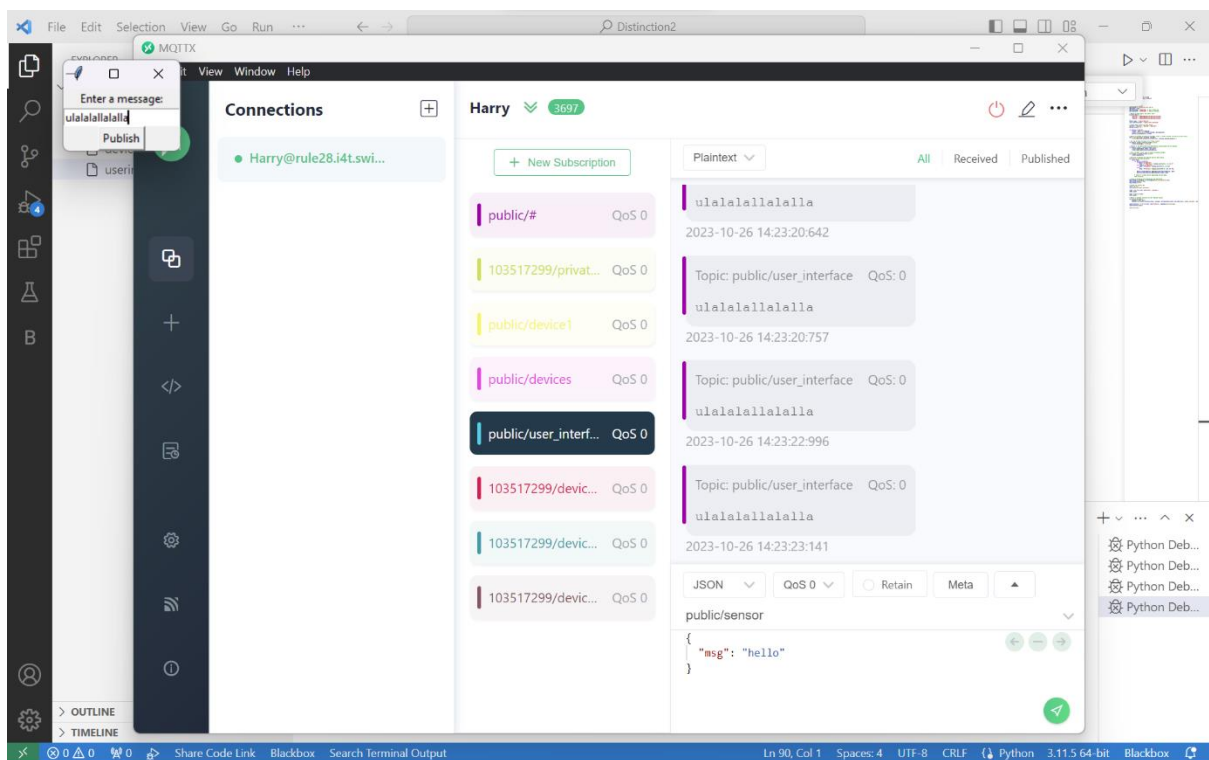
With these added devices, our IoT system showcases multi-device coordination, diverse data generation, and communication.

## Python-Based User Interface

The Distinction Task replaces the graphical MQTT client with a Python-based user interface (UI). This UI acts as a simple, interactive tool to monitor the system and generate/post messages, offering more user-friendly and programmatic access to the IoT environment.

### Key features of the Python UI:

- Displays a message input field.
- Allows users to publish messages to the 'public/user\_interface' topic, enabling interaction with the IoT infrastructure.



Figg:DistinctionTaskPic

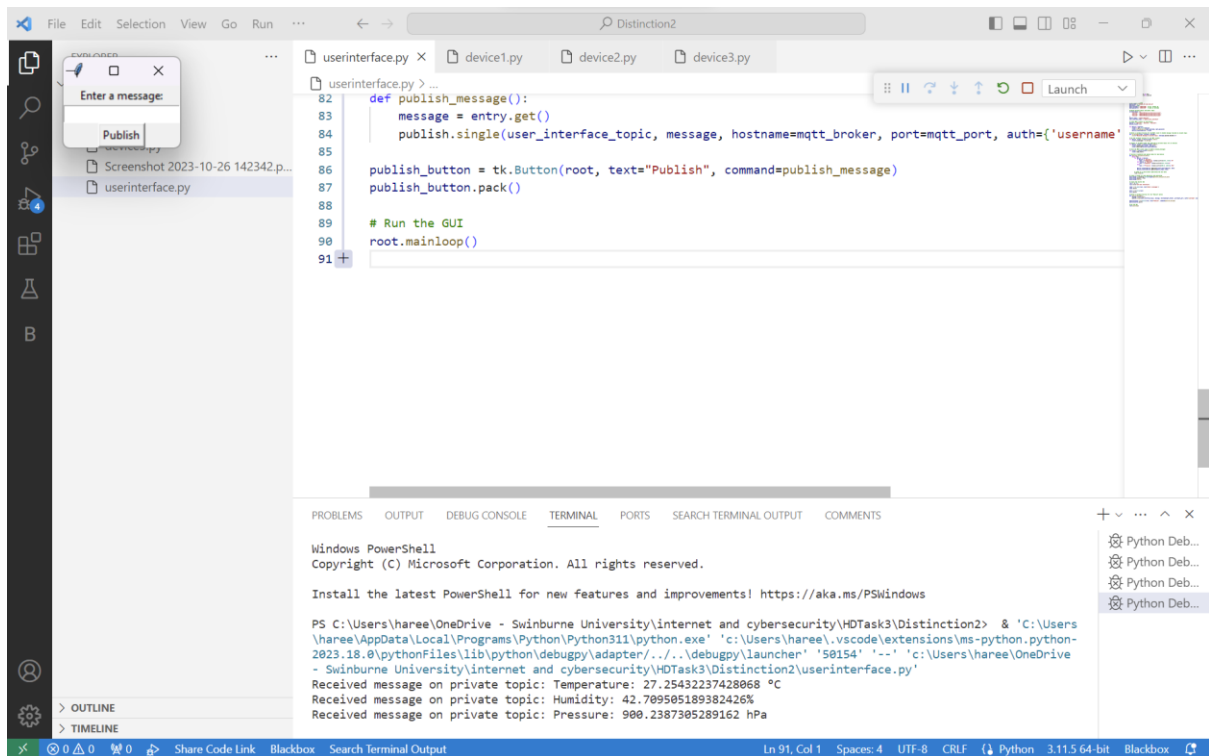


Fig10:DistinctionTaskPic

## Cybersecurity Analysis

### MQTT Broker Security Assessment

In the Distinction Task, we continue our cybersecurity analysis of the MQTT broker to ensure the integrity, confidentiality, and availability of the IoT infrastructure.

- 1. Secure Message Broker Configuration:** The MQTT broker configuration should be kept up to date with the latest security patches and updates, addressing known vulnerabilities.
- 2. Monitoring and Intrusion Detection:** Implementation of monitoring and intrusion detection systems can help in the timely identification and response to suspicious activities or security breaches.
- 3. Secure Public Channels:** Public channels should be reviewed regularly to ensure that sensitive information is not inadvertently exposed, maintaining data privacy and security.

The Distinction Task represents an advanced stage of our IoT prototype, introducing additional devices, enhanced communication capabilities, and a Python-based user interface. Our project emphasizes real-time data exchange and system interaction.

As we proceed to the High Distinction Task, we will explore more complex features, integration with external services, and the development of a user dashboard for comprehensive system control and monitoring.

## **Conclusion**

The Distinction Task represents a significant step forward in the development of our IoT prototype, showcasing the potential of MQTT-based communication in engineering applications. The introduction of three devices, diversified data generation, and an interactive Python-based user interface enhances the system's capabilities. Our cybersecurity analysis emphasizes the need for secure MQTT broker configurations, monitoring, and intrusion detection.

Looking ahead, the High Distinction Task will explore even more advanced features, including the integration of external services and the creation of a user dashboard for comprehensive system control and monitoring. This project underscores the critical role of MQTT in enabling efficient and secure communication within IoT applications and highlights the potential for its use in real-world engineering solutions.