SWINBURNE UNIVERSITY OF TECHNOLOGY

School of Science, Computing, and Engineering Technologies

Internet and Cybersecurity for Engineering Applications
COS30049

Cybersecurity Report
Security in Developing Soil Moisture Monitoring for Smart Garden

STUDENTS

Tran Anh Thu Pham - 10381840

HAWTHORN – OCTOBER 2024

Table of Contents

I. Executive Summary

This project develops a Smart Garden system that employs the MQTT protocol for communication and control of sensor data. It can monitor and manage soil moisture levels to optimize plant growth and reduce water waste. The project emphasizes security aspects, integrating security measures to ensure the data's CIA (Confidentiality, Integrity and Availability). Through simulated experiments, these security measures highlight their role in enhancing the functionality of the Smart Garden.

II. Introduction

The Internet of Things (IoT) offers numerous applications but faces significant cybersecurity challenges. MQTT (Message Queuing Telemetry Transport) is a widely used lightweight protocol for IoT communications. However, it lacks built-in security features, making it vulnerable to various cyber threats.
This project focuses on designing a secure MQTT system for a Smart Garden application, which ensures efficient communication between sensors and a central management system while addressing security issues like data confidentiality, authentication, and message integrity. This report demonstrates all features that meet requirements for HD grade level according to project instructions.

III. Background Information

1. Overview of MQTT in IoT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for Internet of Things (IoT) devices, particularly in environments with low bandwidth and high latency. It operates on a publish-subscribe model, facilitating efficient communication between devices while minimising resource use.

2. Key Concepts

- **Publish-Subscribe Model:** publishers (clients) send messages to a broker, distributing them to subscribers. There are no direct connections between clients, allowing flexible communication.
- **MQTT Clients and Broker:** The broker serves as a central server, managing message distribution and subscriber sessions. It filters messages based on topics and sends them to subscribed clients.
- **Working Mechanism:** Clients can act as both publishers and subscribers. If connectivity is lost, the broker buffers messages and sends

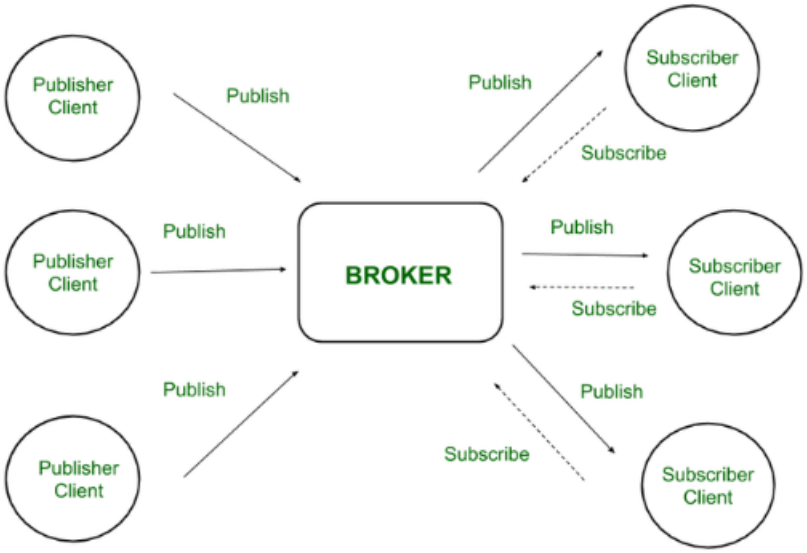

Fig 1. Publisher and Subscriber Model

3. Security Vulnerabilities in the MQTT protocol

MQTT has several security vulnerabilities, including:

- Confidentiality and Data Integrity: Messages in MQTT are often transmitted in plain text, making them vulnerable to eavesdropping and modification through man-in-the-middle attacks.
- Weak Authentication and Authorization: Using the same shared credentials (username and password) without strong authentication methods makes it easy for attackers to guess or gain unauthorized access.
- Broker Vulnerability: A compromised broker can jeopardize the entire system by dropping, altering, or injecting malicious messages.
- Low Quality of Service: Ineffective QoS levels can result in message loss during network congestion or high traffic.

This project is aimed at analysing and enhancing these security features by some specific security technical implementations.

IV. System Architecture

1. System Overview:

- In this project, the Smart Garden is a system that simulates the real-life soil moisture sensor. To be more specific, whenever the watering system is turned on, the soil moisture level will increase over time. The monitoring system will record the increase or decrease of the soil moisture and announce this information to the client by sending messages.

- If a natural disaster occurs such as flooding, the central system (broker) will be disconnected from all clients suddenly to reduce the damage.
- To utilize this system's functionalities, users can use either the MQTTX GUI provided by the MQTTX website or the custom GUI developed by myself. All functions and options of this custom interface will be demonstrated in the next section below.

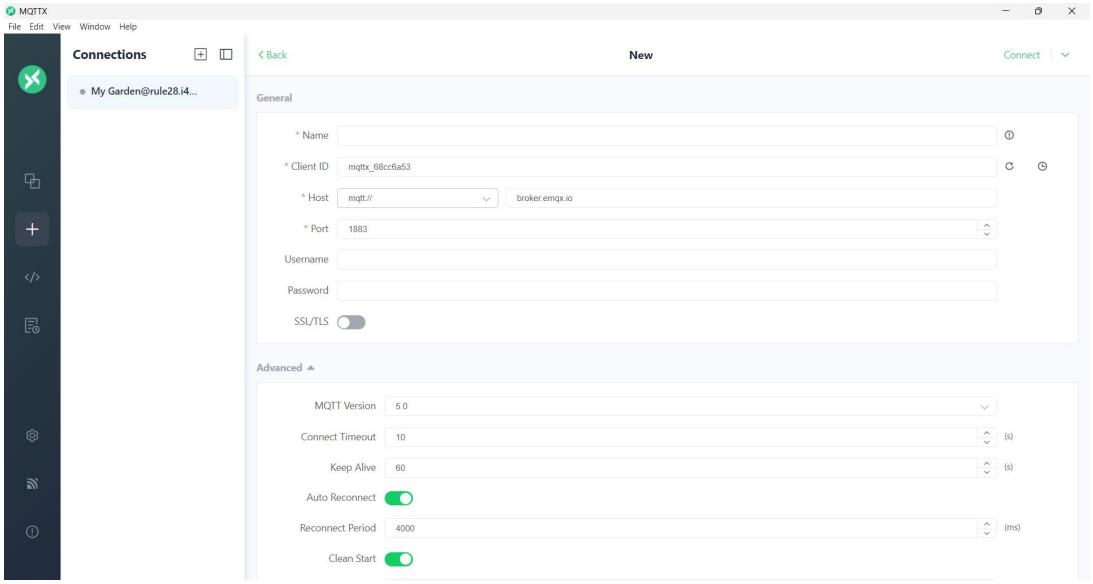

Fig 2. MQTTX's conventional graphical user interface

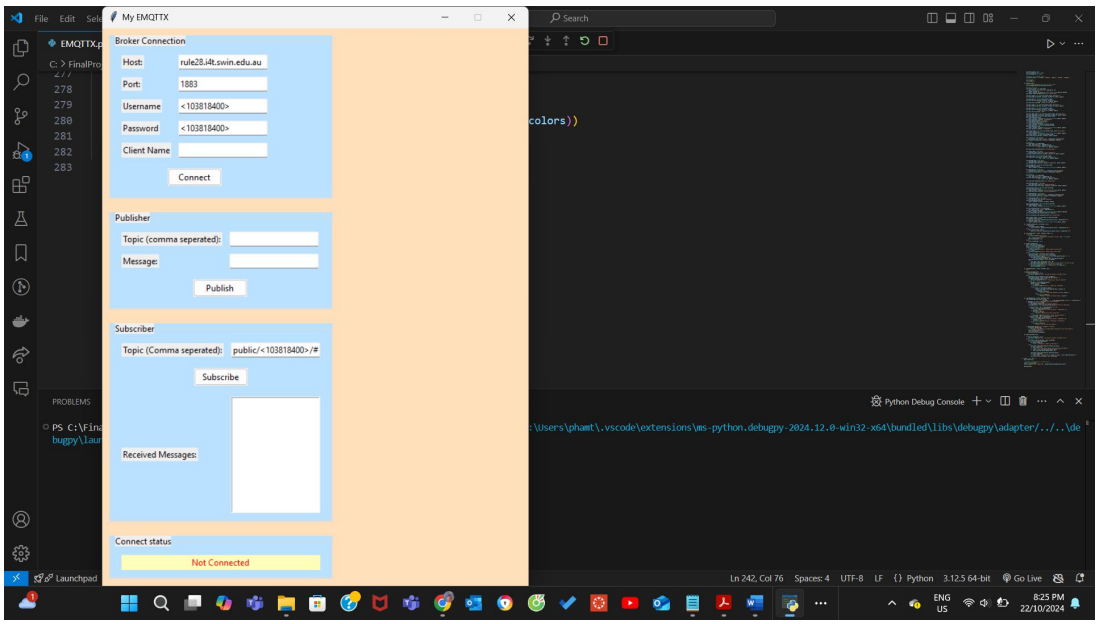

Fig 3. Custom graphical user interface

2. **Custom GUI:**
   Apart from the MQTTX GUI, I also create my custom user interface. The figure below will illustrate my GUI. To get this GUI interface, please run the script file "MyMQTTX.py". This interface contains the following functionalities allowing users to interact with the system:
   - **Broker Connection:** Allow clients to connect to the broker by typing the hostname, port number, username, password, and client name. After hitting the "Connect" button, the message box will appear to inform the successful connection.
   - **Publisher and Subscriber:** Allow clients to publish messages to a specific topic and subscribe to any topic they want. After hitting the "Publish" button or "Subscribe" button, the message box will pop up on the screen to show the successful actions. After that, clients can view their messages in the "Received Messages" field.
   - **Connect Status:** Show the status of the connection

3. **Parameter:** soil moisture, flood.

4. **Functionalities:**
   The system architecture of the Smart Garden project consists of the following key components corresponding to their functionalities:
   a. Show watering status: Every 2 seconds, the central system (broker) will send a message to the client followed by the moisture level value. There are two statuses: watering and not watering. The watering status will be contained in the message sent to the clients. The moisture level might be increased or decreased depending on the status of the system.
   b. Watering control:
      - Turn on: If the users send the message with the content: "on", the watering system will be turned on and at this stage, the soil moisture level will be incremented by one.
      - Turn off: when users send "off" messages, the watering system will change the status to be turned off suddenly and the moisture level will be decremented by one.

V. Environment Setup

This section outlines the necessary components and configurations to establish the environment for developing my project:

1. **Environment for Developing Scripts:**
   This project is implemented using Python. To ensure effective development and usage, the following installations are recommended:

- **Python**: Install the latest version of Python.
- **Python Package Installer (pip)**: This package manager allows for easy installation of additional libraries needed for the project.
- **MQTT Client Library**: Install the paho-mqtt library, which provides a simple interface for MQTT communication. In the CLI, type the following command:

  pip install "paho-mqtt<2.0.0"

  Make sure that, the pip package is installed before installing the MQTT client libraries.

2. **Graphical User Interface (GUI) for Testing:**

   To facilitate communication between devices, an MQTT broker needs to be set up. The following steps outline the process:

   - Install an MQTTX broker by downloading from the MQTTX website.
   - Create 1 connection for the client which is running the MQTT (Mosquitto) Message Broker on: **rule28.i4t.swin.edu.au.**
   - The port number should be set to 1883 by default.
   - Enter the following credentials for successful client authentication:
     - Username: <103818400>
     - Password: <103818400>
   - The SSL/TLS option should not be turned on due to the limitation of the broker site.

   At this stage, your connection should be successful, then create 2 subscriptions corresponding to 2 topics:

   - public/<103818400>/soil moisture/control: Control the system's watering status.
   - public/<103818400>/soil moisture/status: Monitoring the watering status of the system.

   After applying all GUI configurations, the MQTTX GUI should be similar to the below figure:

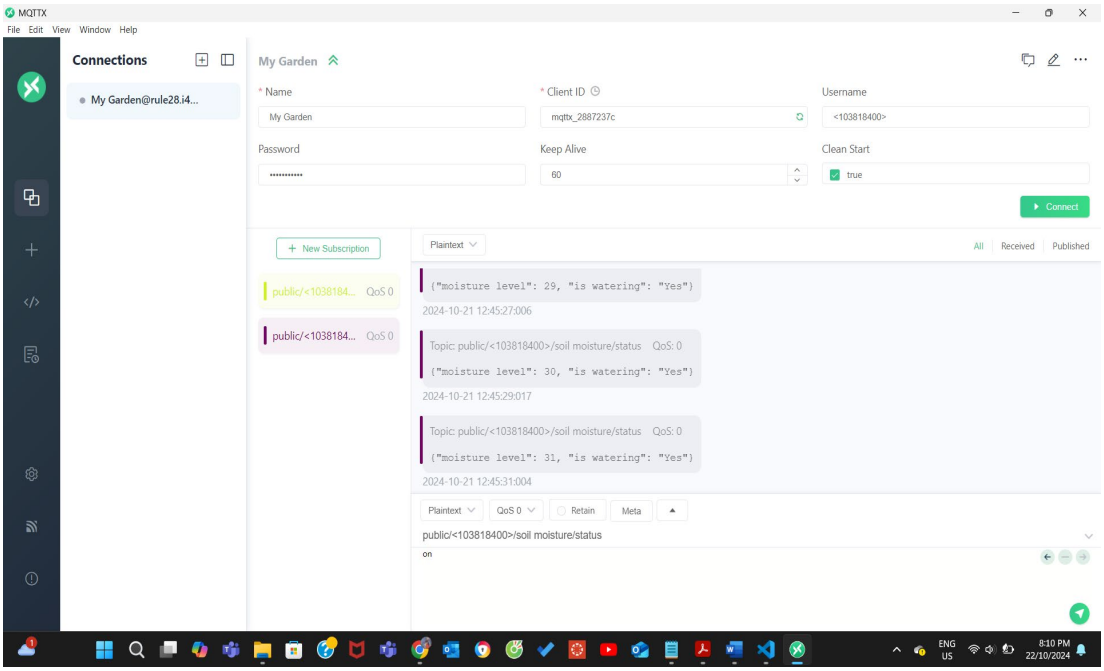

Fig 4. MQTTX configurations

VI. Technical Implementation

1. Necessary Libraries Importation

The implementation of the MQTT client application leverages several Python libraries, each providing essential functionalities:

- **tkinter:** Creates the graphical user interface (GUI) with windows, labels, buttons, and entry fields for user interaction.
- **paho.mqtt.client:** Manages MQTT communication, allowing the app to connect to the broker, publish, and subscribe to messages.
- **random:** Generates random background colours for a dynamic and visually appealing interface.
- **json:** Handles JSON-encoded payloads for processing incoming MQTT messages, such as soil moisture data for system control.

These libraries enable a responsive, connected MQTT client with a user-friendly interface.

2. MQTT client implementations

The Python script “ClientSignal.py” is an MQTT client that connects to a broker and simulates monitoring and controlling soil moisture in a garden.

a. MQTT Broker Setup

The code implemented in the script defines the broker address and user credentials for connecting. It also specifies the topics to subscribe to and publish. For instance, the client can subscribe to the control command such as “on” or “off” to the “Control” topic.

```python
# Define the broker address and credentials for connecting to the MQTT broker
broker = 'rule28.i4t.swin.edu.au'
user_information = {
    "client_identifier": "My Garden (Publisher)",
    "username": "<103818400>",
    "password": "<103818400>"
}
watering = False

topic = [("public/#", 0), ("public/<103818400>/soil moisture/control", 0)]
publish_topic = "public/<103818400>/soil moisture/status"
```

Fig 5. MQTT broker setup

By default, I set the watering status off to protect the system operation from wasting water.

b. Functions:

- **connect_mqtt():** Establishes the connection to the MQTT broker with authentication, and sets the callback for a successful connection.
- **__showLog():** Logs MQTT activity for debugging (e.g., connections, disconnections, message publications).
- **subscribe():** Subscribes to the specified topics and processes incoming messages, toggling the watering system on or off based on received commands.

```python
# Subscribing to Topics
def subscribe(client: mqtt.Client) -> None:
    def on_message(client, userdata, msg):
        global watering

        print("\n\n**********SUBSCRIBE**********")
        __showLog(client)
        print(f"Topic: {msg.topic}")
        print(f"Payload: {msg.payload.decode('utf-8')}")
        print(f"QoS: {msg.qos}")
        print(f"Retain: {msg.retain}")
        print("**********SUBSCRIBE**********\n\n")
        if msg.payload.decode("utf-8").lower() == "on":
            watering = True
        if msg.payload.decode("utf8").lower() == "off":
            watering = False

    client.on_message = on_message
    client.subscribe(topic)
```

Fig 6. Subscribe function

- **publish():** Publishes soil moisture status to the topic every 2 seconds, increasing or decreasing moisture level depending on whether the watering system is on or off.
- **__disconnect():** Handles and logs the disconnection from the broker.

c. Core Functionalities:

- **Connection to MQTT Broker:** Authenticates and establishes a connection to a central broker.
- **Publishing Soil Moisture Data**: Simulates soil moisture readings and publishes them in JSON format at regular intervals.
- **Dynamic Watering Control**: Adjusts soil moisture readings based on received MQTT commands ("on" or "off").
- Besides, it handles user interruptions (Ctrl-C) by safely disconnecting from the broker.

3. **Custom GUI implementation**
   a. **Overall Interface Design**

   The GUI is designed using the Tkinter library, providing a user-friendly interface for interacting with the MQTT broker. The main components of the interface include:
   - **Window Creation:** The main window is created using Tk() and titled appropriately.
   - **Layout:** The layout utilizes frames to separate different functionalities. The subscription and publishing areas are placed in separate frames for clarity.
   - **Status Display:** A Text widget is included to display incoming messages and connection status.

   b. **Subscribe Topic Functionality**

   The subscription functionality allows users to subscribe to MQTT topics and receive messages.
   - **Input for Subscription:** An entry widget is created for the user to input the topic they wish to subscribe to
   - **Subscribe Button:** A button triggers the subscription to the entered topic
   - **Subscription Logic:** The subscribe method retrieves the topic from the entry and calls the MQTT client's subscribe function.
   - **Message Handling:** Incoming messages are handled by the on_message callback, which appends the message to the Text widget for display.

   View the comments in my code files for more details.

   c. **Publish Message Functionality**
   - **Input for Publishing:** The GUI includes two entry fields—one for the topic and another for the message content
   - **Publish Button:** A button allows users to publish messages
   - **Publishing Logic:** The publish method retrieves the topic and message from the entries and publishes them using the MQTT client

   View the comments in my code files for more details.

4. **Logistics of the System Operation:**

   a. **Moisture Level Measure Operation**

   The watering control system uses moisture level thresholds to manage irrigation effectively. If the moisture_level exceeds 30 and the watering_status is not "off," a message is published via MQTT to turn off watering, prompting a success or failure notification based on the publish result. Conversely, if the moisture_level drops below 20 and the watering_status is "off," it sends a command to turn on watering, also providing feedback on the operation's success or failure. This automated control helps maintain optimal soil moisture for plant health.

```
if moisture_level > 30 and watering_status != "off":
    result = self.mqtt_client.publish(
        "public/<103818400>/soil moisture/control", "WATER OFF", 0)
    if result[0] == 0:
        messagebox.showinfo(
            "Success", "Watering has been turned off")
    else:
        messagebox.showerror("Failed", "Could not send request")
if moisture_level < 20 and watering_status == "off":
    result = self.mqtt_client.publish(
        "public/<103818400>/soil moisture/control", "WATER ON", 0)
    if result[0] == 0:
        messagebox.showinfo("Success", "Watering is turned on")
    else:
        messagebox.showerror(
            "Failed", "Can not send the request")
```

Fig 7. Moisture Level Measure Operation

   b. **Watering Control System Operation**

   The watering control system simulates soil moisture levels based on the watering status. When watering is active, the moisture_level increases by 1, reflecting added moisture, while an information dictionary updates to indicate the watering status. Conversely, when watering is off, the moisture_level decreases by 1, simulating natural moisture loss. To improve the system, additional features could include moisture level thresholds to prevent extremes, real-time sensor feedback for automated irrigation control, and user notifications for optimal plant health management.

```
# If will_turn_on is True, the soil moisture level increases, simulating the watering being on
if watering:
    moisture_level += 1
    info = {
        "moisture level": moisture_level,
        "is watering": "Yes"
    }

# If will_turn_on is True, the soil moisture level decreases, simulating the watering being off
if not watering:
    moisture_level -= 1
    info = {
        "moisture level": moisture_level,
        "is watering": "No"
    }
```

Fig 8. Watering control system operation

The figure below shows that the watering status changes to “Yes” after typing “on”.

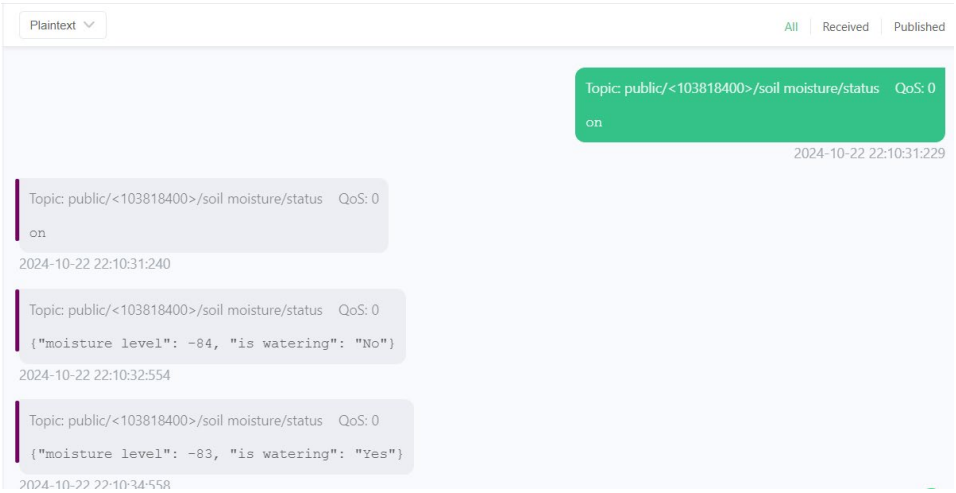

Fig 9. Sending a message to change the watering status

c. Disconnect from the broker if “flooding”

In the watering control system, if a message with the payload "flood" is received, it signifies an alert for a natural disaster. This triggers a warning to the user, indicating that the system needs to take immediate action due to the impending threat. The system updates the connection status to false, marking it as disconnected, and the MQTT client is subsequently disconnected to halt all watering activities. This precautionary measure ensures that the system does not operate during a flood, thereby protecting the infrastructure and preventing further complications associated with the disaster.

```
if msg.payload.decode("utf-8").lower() == "flood":
    messagebox.showwarning(
        "Warning", "The broker is having problem, disconnect all of the clients!")
    self._connect_status(False)
    self.is_connected = False
    self.mqtt_client.disconnect()
```

Fig 10. Disconnect from the broker if there is flood

VII. Security Enhancement

1. **Confidentiality and Data Integrity:** MQTT messages are frequently transmitted in plain text, making them susceptible to eavesdropping and manipulation through man-in-the-middle (MitM) attacks. This vulnerability can lead to significant consequences, such as unauthorized interception of sensitive data, resulting in data breaches and privacy violations. Additionally, attackers may alter messages before they reach their intended recipients, causing incorrect or harmful commands to be executed in IoT systems, which undermines user trust in these technologies. To mitigate these risks, implementing Transport Layer Security (TLS) is crucial, as it encrypts data transmitted between clients and brokers, ensuring that intercepted messages remain unreadable. Furthermore, incorporating message integrity checks, such as checksums or hash functions, can help validate that messages have not been altered during transmission, enhancing overall security.

2. **Weak Authentication and Authorization**: The use of shared credentials (username and password) without robust authentication methods poses a significant security risk in MQTT environments. Weak authentication enables attackers to easily guess or brute-force credentials, leading to unauthorized access to IoT devices and systems. This unauthorized access can facilitate data tampering, where malicious actors alter or delete messages, thereby compromising data integrity and potentially causing physical damage to connected devices. To address this vulnerability, organizations should adopt strong authentication methods, such as multi-factor authentication (MFA) or OAuth, which provide an additional layer of verification to enhance security. Implementing Role-Based Access Control (RBAC) is also essential, as it restricts access based on user roles, ensuring that individuals only have access to the specific topics and operations necessary for their responsibilities.

3. **Broker Vulnerability:** A compromised MQTT broker can jeopardize the entire system, as it has the potential to drop, alter, or inject malicious messages. The impact of such a compromise can be severe, leading to widespread system failures that disrupt communication across all connected devices. This disruption may result in critical application failures in sectors like healthcare or transportation. Furthermore, malicious payload injection can occur, enabling attackers to insert harmful commands or data, thereby breaching data integrity

and privacy. To combat this threat, organizations should ensure regular updates and patching of their MQTT brokers to mitigate known vulnerabilities. Additionally, employing firewalls and intrusion detection systems can help monitor traffic to and from the broker, effectively detecting unauthorized access attempts and enhancing the overall security posture.

4. **Low Quality of Service (QoS):** Ineffective Quality of Service (QoS) levels in MQTT can lead to message loss during network congestion or high traffic, significantly impacting the reliability of communications. This issue can result in incomplete data collection, leading to gaps that hinder accurate analysis and poor decision-making. In time-sensitive applications, such as autonomous vehicles or healthcare systems, delays in response due to lost messages can pose serious safety risks. To improve QoS, organizations should carefully select the appropriate QoS levels (0, 1, or 2) based on the importance of their messages. For critical messages, using QoS level 2, which guarantees delivery exactly once, is recommended, while QoS level 1 may suffice for less critical messages. Additionally, optimizing network infrastructure to handle expected traffic loads, including redundancy measures to reduce congestion, can further enhance the reliability of message delivery in MQTT systems.

VIII. Conclusion

This report details the development of a Smart Garden system using MQTT for efficient soil moisture management. It identifies key security vulnerabilities in MQTT, such as insufficient encryption and weak authentication. By implementing robust security measures, the system enhances data confidentiality, integrity, and availability. Simulated experiments confirm the effectiveness of these security measures against potential cyber threats. The Smart Garden not only optimizes plant growth and conserves water but also serves as a model for secure IoT applications, highlighting the importance of addressing security in IoT protocols for future developments.

IX. References

1. Tran, T.D., Nguyen, H.S. and Vo, T.A., 2020. Design and Implementation of a Smart Garden System Using IoT Technology. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 9(2), pp.489-494. Available at: https://www.ijitee.org/wp-content/uploads/papers/v9i2/B2647129219.pdf [Accessed 17 October 2024].

2. Rao, B.P.K., 2018. MQTT Protocol: An Overview. *International Journal of Advanced Research in Computer Science*, 9(2), pp.1-4. Available at: https://www.ijarcsse.com/docs/papers/Volume_9/2_February2019/V9I2-0258.pdf [Accessed 15 October 2024].

3. Hashim, H.K.A., Abdalla, R.M. and Mahmoud, A.A., 2021. Security Approaches for MQTT Protocol: A Comprehensive Survey. *Sensors*, 21(3), p.685. Available at: https://doi.org/10.3390/s21030685 [Accessed 20 October 2024].

4. Freire, J.A.A.S., Ribeiro, C.A.J.S.D. and de Lima, L.M.S., 2020. Security for MQTT Protocol in Internet of Things. *2020 15th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pp.135-140. Available at: https://ieeexplore.ieee.org/document/9240680 [Accessed 18 October 2024].

5. Ali, M.A., Alzahrani, A.J. and Alshahrani, A.Z., 2020. Security Challenges in IoT: A Survey. *Journal of King Saud University - Computer and Information Sciences*, 32(4), pp.457-472. Available at: https://doi.org/10.1016/j.jksuci.2018.06.004 [Accessed 18 October 2024].

X. Appendices

This project contains 2 source code files which were compressed in a zip file:

- MyMQTTX.py
- ClientSignal.py