Ibn Zohr University
Faculty of Science – Center of Excellence IT – Agadir

Master's Program of Excellence
Specialization: Computer Engineering and Embedded Systems (IISE)

Smart Device Traffic Analysis: Detection of Unencrypted MQTT Communication and Data Leakage in IoT Networks

Supervised by: Prof. BOUGHROUS Moncef

Prepared by:

Khaoula EL HARRAZ
Oussama GOUSSA
Maryem EL-BOUCHTI
Khawla EL HASSNAOUI

Date: November 17, 2025

1. Executive Summary

The Message Queuing Telemetry Transport (MQTT) protocol is a lightweight publish–subscribe communication standard commonly used in Internet of Things (IoT) systems. It is designed to work with very low bandwidth and requires minimal processing power, which makes it suitable for small devices, sensors, and networks that may be unstable or limited. MQTT uses a broker to manage communication between publishers and subscribers, allowing devices to exchange data reliably and efficiently.

This report presents a security analysis of an IoT communication setup that uses MQTT to send sensor data to a server. Wireshark was used to capture and analyze network traffic in order to understand how data is transmitted and whether sensitive information is exposed. The assessment compares two configurations: normal MQTT, where data is sent in plaintext, and secure MQTT over TLS (MQTTS), where data is encrypted. The goal is to observe how encryption protects the data during transmission.

It is important to clarify that this project focuses only on communication security and encryption. Authentication methods—such as usernames and passwords, access tokens, or certificate-based client authentication—were not tested and are outside the scope of this work. The assessment therefore examines confidentiality (how visible the data is on the network) rather than access control.

By analysing how MQTT behaves with and without encryption, this report highlights the risks of using unsecured MQTT and shows the benefits of adopting MQTTS. The results provide a clear basis for improving the security of IoT systems by enabling encryption, adding authentication, and following secure deployment practices.

a. System Tested

The test environment was designed to mirror a common IoT communication workflow. In this setup, a Publisher device generates and sends JSON-formatted sensor data to a Mosquitto MQTT broker. The broker then distributes these messages to a Subscriber device that listens on the corresponding topic. For this security assessment, the MQTT communication was intentionally configured to operate in its default, unencrypted mode using port 1883. This allowed us to observe how data behaves on the network when no encryption or protection mechanisms are applied.

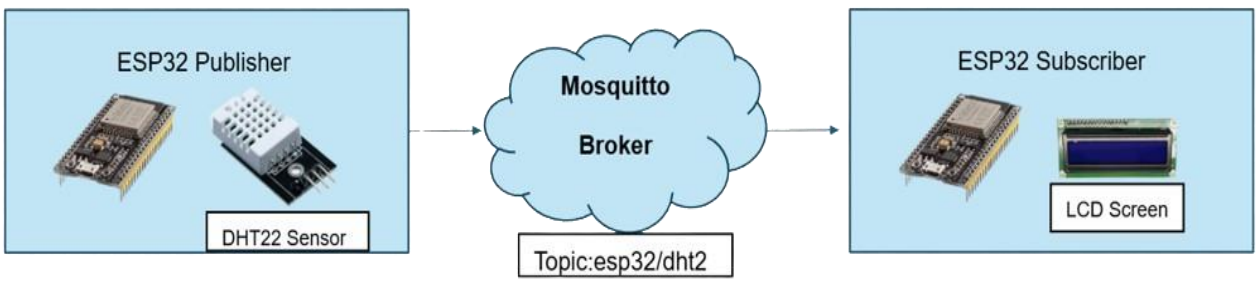

Figure 1: General Architecture of MQTT Communication in the Tested System

b. Key Findings: Wireshark Confirms Data Leakage

The central finding, derived directly from network capture analysis, is that the entire data payload is exposed to passive interception.

Finding ID	Description	Severity	Impact
C-01	Unencrypted Data Payload (Plaintext MQTT)	CRITICAL	Wireshark revealed a direct, readable JSON payload in every MQTT PUBLISH packet, resulting in total data confidentiality compromise.
H-02	Unsecured TCP Port Usage (1883)	HIGH	The standard, non-secure port 1883 immediately alerts attackers to the use of vulnerable, unencrypted MQTT traffic, facilitating passive sniffing.
M-03	Lack of Client Authentication	MEDIUM	Brokers are exposed to unauthorized connections and potential data injection, often leveraged by attackers after initial traffic analysis.

c. Critical Recommendations

➢ To secure the MQTT communication and protect sensor data, the following actions are strongly recommended:

1. Switch to Encrypted Communication (MQTTS)
   - Immediately migrate all MQTT traffic to MQTT over TLS/SSL (MQTTS).
   - Use port 8883 for encrypted connections.
   - Install and configure valid server and client certificates to prevent passive interception of data.
2. Enable Strong Authentication
   - Require all clients connecting to the broker to use a secure username and password.
   - This prevents unauthorized devices from publishing or subscribing to topics.
3. Implement Access Control (ACLs)
   - Restrict each client's permissions to only the topics they need.
   - Use Access Control Lists (ACLs) to enforce these restrictions, reducing the risk of data injection or misuse.

2. Problem Statement & Scope

2.1. Problem Statement: The Unencrypted IoT Risk

IoT (Internet of Things) devices, like smart home sensors or industrial machines, are becoming very common. They collect and send data to other devices or servers to help automate tasks and improve efficiency. However, many IoT devices are not designed with strong security. This makes them vulnerable to attacks.

One major risk comes from the way these devices communicate. MQTT, a popular protocol for IoT messaging, is very lightweight and easy to use but was not originally built with security in mind. When MQTT is used without encryption, the messages sent between devices can be read by anyone who intercepts the network traffic. This means attackers can see sensitive data, like sensor readings or personal information.

In addition, if the MQTT broker does not require authentication or access control, unauthorized devices can connect, send fake data, or read messages they shouldn't have access to. These weaknesses can cause serious problems, from leaking personal data to interfering with industrial operations.

2.2. Scope of the Assessment: Traffic Analysis

This report focuses only on the security of MQTT communication in an IoT system. We tested a setup where a Publisher device sends sensor data to a Subscriber device through a Mosquitto MQTT broker. The goal is to see how safe the data is while moving across the network.

The audit includes:

1. Data Transmission Security

Checking if messages are sent in plain text (MQTT) or encrypted (MQTTS).

Capturing network traffic with Wireshark to see if sensitive data can be read.

2. Exposure Risks

Looking at the risks of using unencrypted communication and unsecured ports (like 1883).

The audit does not include:

- Hacking the devices or exploiting vulnerabilities.
- Checking physical security of IoT devices.
- Testing software bugs in the devices or applications.
- Detailed testing of authentication or certificates.

By focusing on communication security, this report shows the risks of using MQTT without encryption and why using secure MQTT (MQTTS) is important. It helps to understand what steps are needed to protect IoT data during transmission.

3. Technical Approach & Methodology

The purpose of this audit was to evaluate the security of an IoT system using MQTT to send sensor data. The main goal was to observe how data behaves when sent in plaintext (MQTT) versus encrypted (MQTTS), and to understand the risks if messages are intercepted on the network.

The methodology focused on capturing and analysing network traffic to see whether sensitive information is exposed and how encryption can protect it.

Step 1: Setting Up the Test Environment

The test environment was created to simulate a real-world IoT system using Wokwi, an online IoT simulator. Two ESP32 devices were used:

- Publisher Device (ESP32 + DHT22 Sensor):
  - Reads temperature and humidity from the DHT22 sensor.
  - Sends the sensor data as JSON messages to an MQTT broker.
- Subscriber Device (ESP32 + LCD Display):
  - Subscribes to the same MQTT topic as the publisher.
  - Receives sensor data and displays it on an LCD screen.
- MQTT Broker (Mosquitto):
  - Manages the communication between Publisher and Subscriber.
  - Two configurations were tested:
    1) Plain MQTT on port 1883 (unencrypted).
    2) Secure MQTT over TLS (MQTTS) on port 8883 (encrypted).

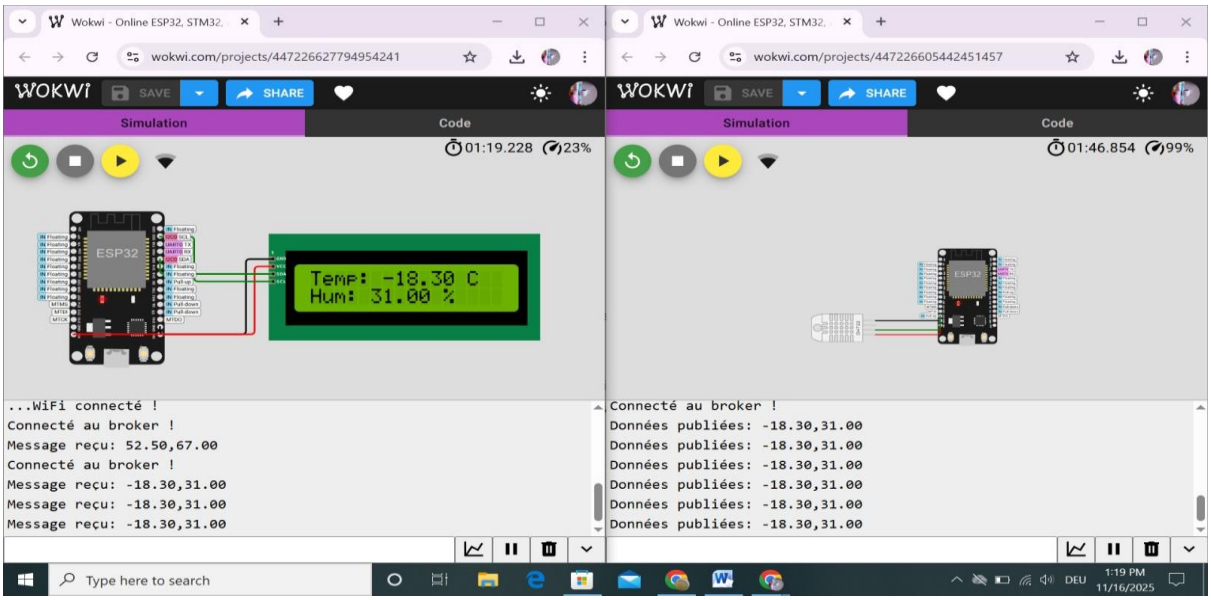

Figure 2: IoT Test Environment

Step 2: Capturing Network Traffic

In Wokwi, the network traffic can be saved as a .pcap file, which contains all packets sent between the Publisher and Subscriber. This file was then downloaded and analysed with Wireshark.

Steps Taken:

1. Simulate the IoT setup in Wokwi and start the MQTT communication.
2. Generate multiple messages from the Publisher to the Subscriber.
3. Download the .pcap file from Wokwi, which contains all network traffic.
4. Open the .pcap file in Wireshark for analysis.

This approach allows us to inspect all messages without needing a live network capture, making the process simple and repeatable.

Step 3: Analysing MQTT Traffic

Once the .pcap file is opened in Wireshark, packets were examined to check whether sensitive data could be read:

- Plaintext MQTT (Port 1883):
  - PUBLISH packets were inspected.
  - JSON sensor data, including temperature and humidity, was fully visible.
  - Topic names were also readable.

- Encrypted MQTT (MQTTS, Port 8883):
  - TLS certificates were configured on the broker and devices.
  - Packets captured in Wireshark showed encrypted data.
  - The payload and topics were unreadable, confirming encryption protects confidentiality.

Step 4: Risk Assessment

From the Wireshark analysis, the following risks were identified:

- **Data Exposure:** Sensor data sent via unencrypted MQTT is visible to attackers.
- **Unauthorized Access:** Without authentication, malicious devices can connect to the broker and send or receive messages they shouldn't access.
- **Unsecured Ports:** Using default MQTT ports (1883) makes it easier for attackers to detect and target the system.

Each risk is categorized by severity to help prioritize improvements.

Tools used

Tool / Device	Purpose
ESP32 + DHT22	Publisher device that generates sensor data.
ESP32 + LCD	Subscriber device that displays received data.
Mosquitto Broker	Manages MQTT messages (unencrypted and encrypted).
Wireshark	Opens and analyses the .pcap file from Wokwi to check data exposure.
TLS / Certificates	Encrypt MQTT traffic for secure communication.
Wokwi Simulator	Simulates ESP32 devices and IoT setup for testing.

4. Implementation & Testing

This section describes exactly how the MQTT security assessment was performed. It includes the setup, the traffic capture steps, the tests done on plaintext MQTT and encrypted MQTTS, and the actions taken to secure the communication. Screenshots and Wireshark captures are inserted as images to illustrate the results.

4.1. Test Environment Setup

4.1.1 Publisher Setup (ESP32 + DHT22 on Wokwi)

The first device created in Wokwi was an **ESP32 board connected to a DHT22 temperature/humidity sensor**.

This device acted as the **publisher**, sending JSON data every few seconds to the MQTT broker. The JSON message format used was: { "TEM": 25.1, "HUM": 48 }

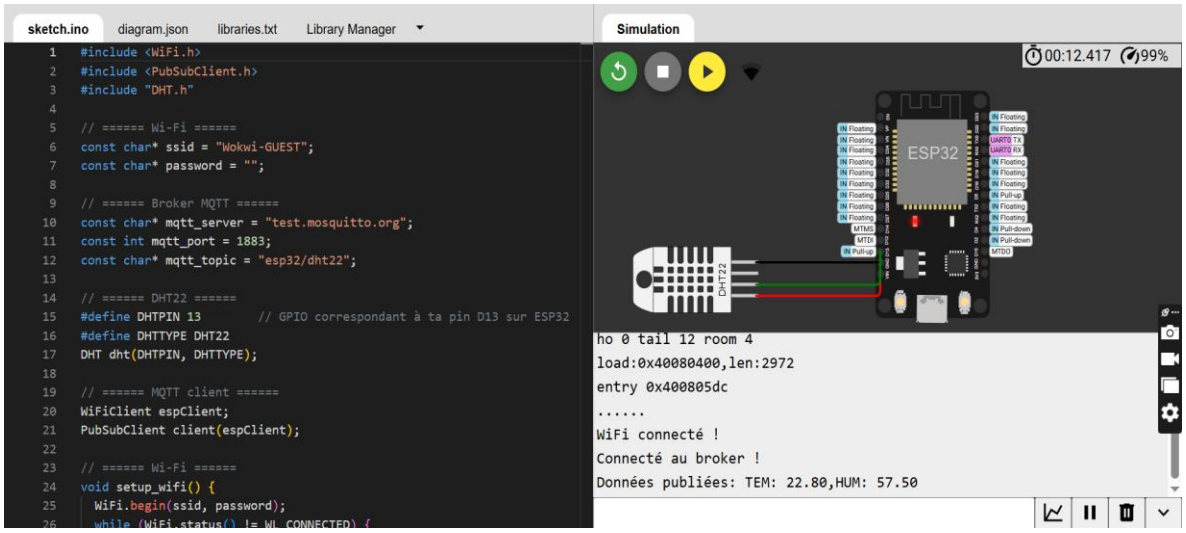

Figure 3: Wokwi Publisher Circuit (ESP32 + DHT22)

4.1.2 Subscriber Setup (ESP32 + LCD on Wokwi)

The second device in Wokwi was another ESP32 connected to a 16x2 LCD. This device subscribed to the same topic and displayed the received values on the screen.

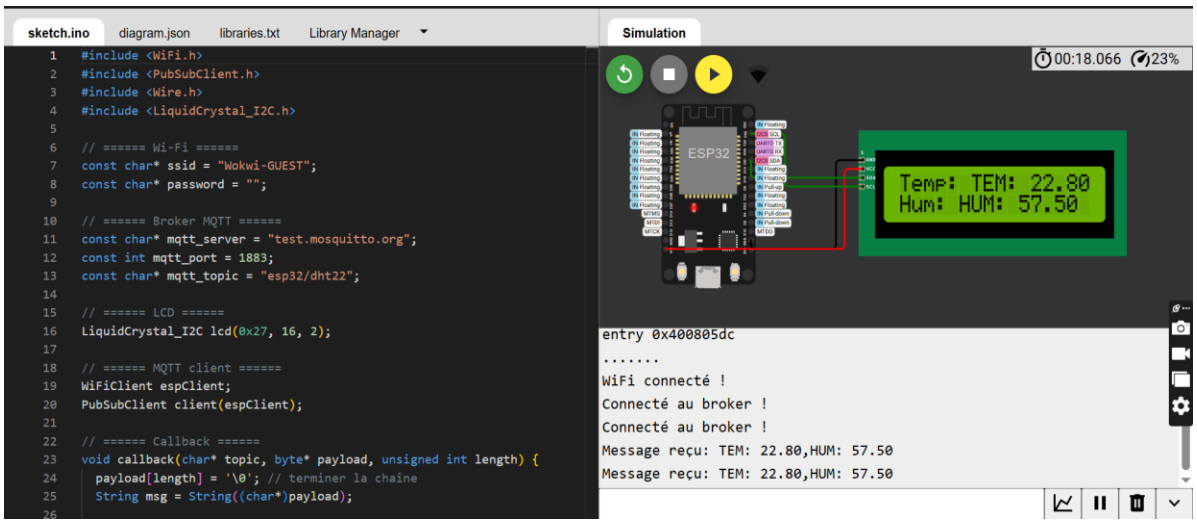

Figure 4: Wokwi Subscriber Circuit (ESP32 + LCD)

4.2. Traffic Capture and Wireshark Setup

4.2.1 Getting the Network Capture

Since Wokwi allows exporting network captures, a .pcap file was downloaded after running each test scenario:

Plaintext MQTT scenario

Encrypted MQTTS scenario

You then opened the file in Wireshark:

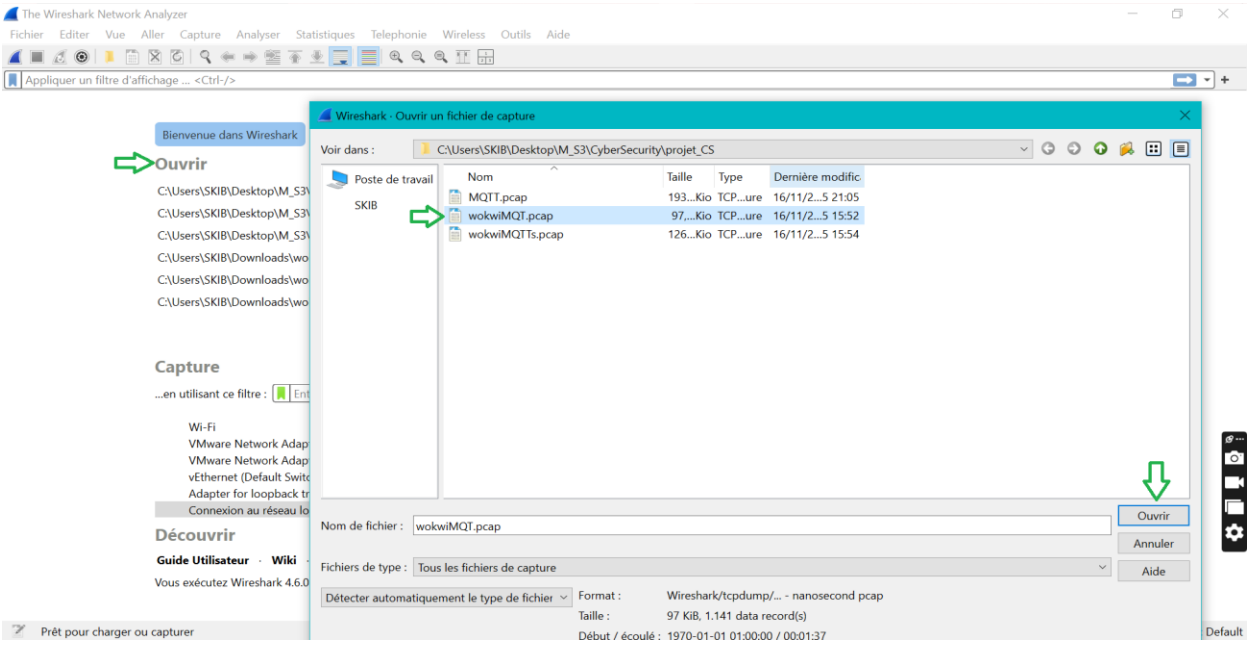

Figure 5: Opening pcap file in Wireshark

4.3. Plaintext MQTT Testing (Port 1883)

4.3.1 Test Procedure

1. Started both Publisher and Subscriber.
2. Messages were published every few seconds.
3. Captured traffic was exported as .pcap.
4. Opened the capture in Wireshark and filtered using: mqtt

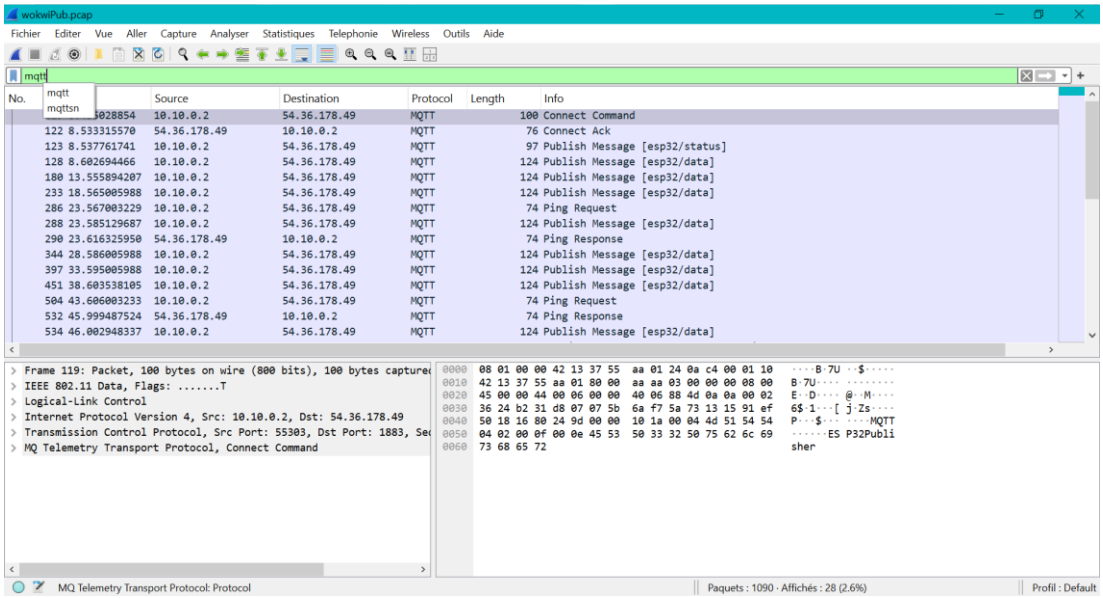

Figure 6: Wireshark showing MQTT packets from Wokwi capture

4.3.2 Results: Data is Fully Visible

Wireshark clearly displayed:

- **Topic name:** esp32/dht22
- **Temperature and humidity values** in JSON
- **MQTT control packets** (CONNECT, SUBSCRIBE, PUBLISH)

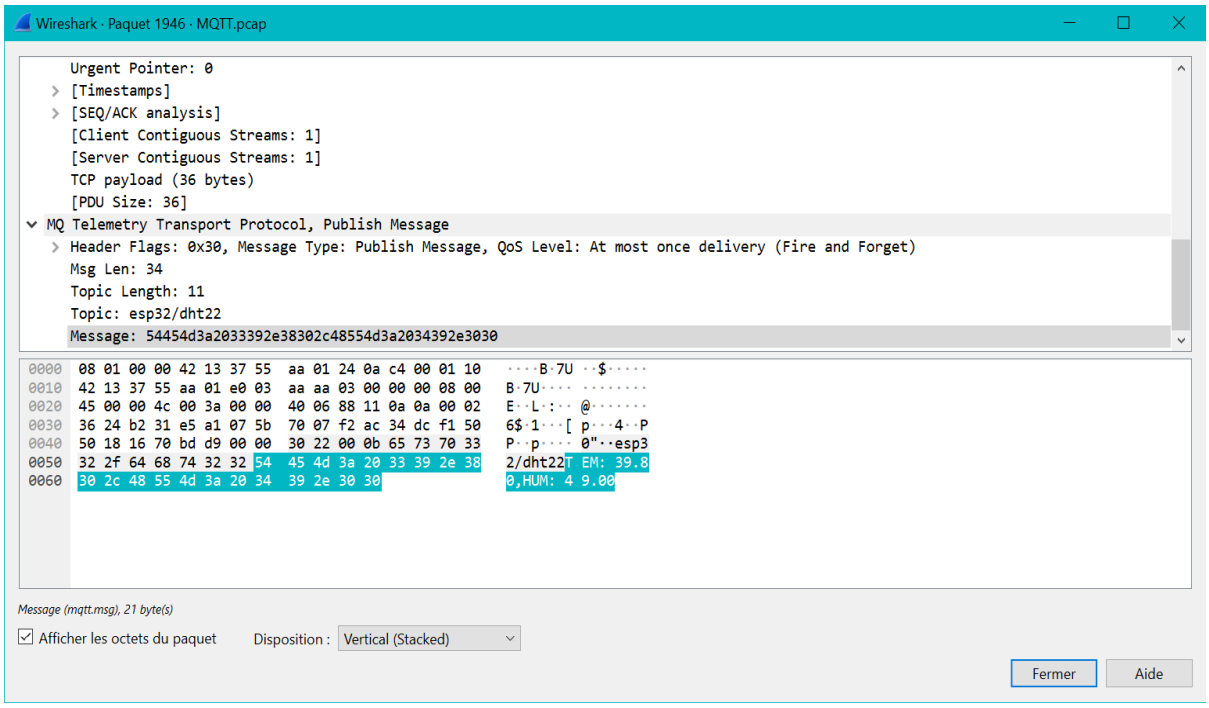

Figure 7: Plaintext MQTT PUBLISH packet showing JSON sensor data

This confirmed that **anyone on the network can read the full message**, including all sensor data.

4.3.3 Security Implications

Because traffic is not encrypted:

- Attackers can **see all data**
- Attackers can **clone the topic** and inject fake readings
- No protection against replay attacks

Plain MQTT is **not safe for IoT systems carrying sensitive data**.

4.4. Secure MQTT (MQTTS) Testing (Port 8883)

4.4.1 Test Procedure

Connected the ESP32 Publisher and Subscriber to the secure Mosquitto server: mqtts://test.mosquitto.org:8883

Enabled TLS on the ESP32 using: espClient.setInsecure();

Ran the simulation and downloaded the second capture file.

Opened the file in Wireshark and applied the filter: tls

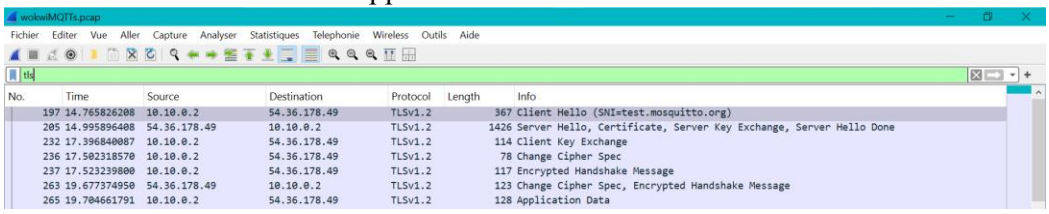

Figure 8: Wireshark showing MQTT packets from Wokwi capture

4.4.2 Results: Data is Encrypted

Wireshark now showed only TLS handshake packets and encrypted Application Data.

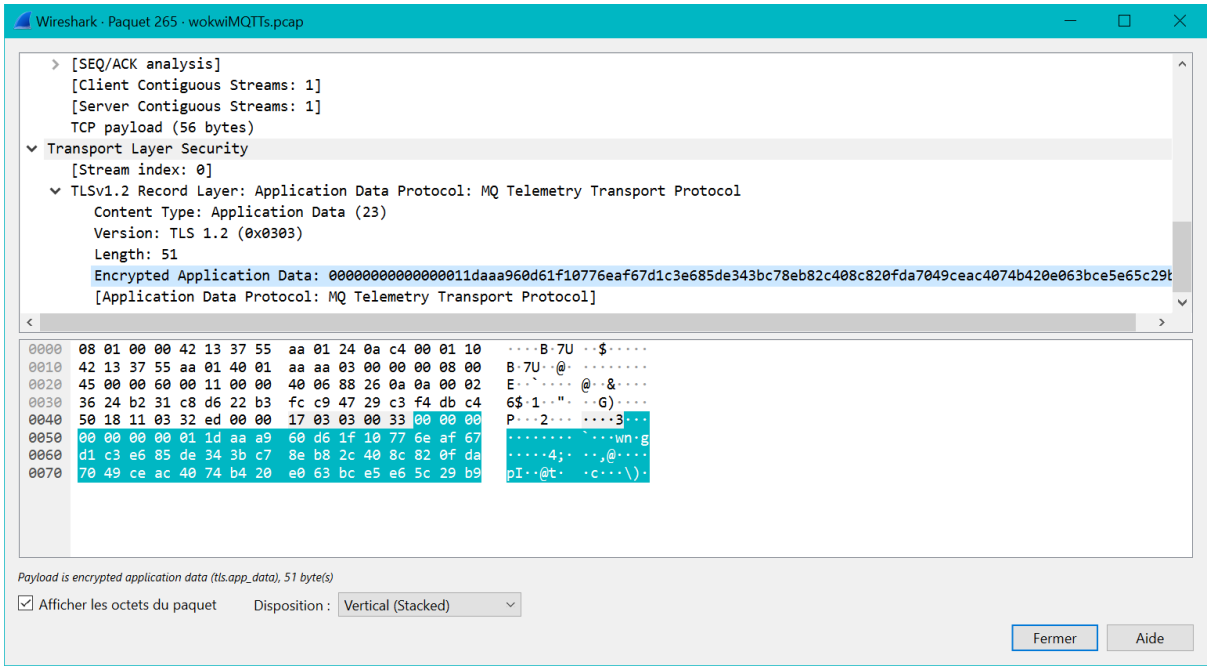

Figure 9: Wireshark showing encrypted MQTT data under TLS

The payload, topics, and sensor values were completely unreadable.

4.4.3 Security Improvement

With TLS enabled:

Security Aspect	MQTT	MQTTS
Data visibility	Visible	Hidden
Topic exposure	Visible	Hidden
Packet sniffing	Easy	Useless
Replay attack	Possible	Hard
Confidentiality	None	Strong

4.5 Exploitation Attempts

4.5.1 Packet Sniffing

- Plain MQTT (1883): Successful. Wireshark showed the full JSON payload (temperature, humidity, and topic).
- MQTTS (8883): Packets were encrypted. Wireshark only displayed TLS handshake and encrypted blobs. No sensor data or topic names were readable.

4.5.2 Unauthorized Subscription

With plain MQTT, anyone could subscribe to the topic if they know the broker and topic name. With MQTTS, the traffic is encrypted, but there is still no access control, so unauthorized clients could still connect if they know the broker address.

This test shows that encryption alone is not enough. To fully secure the system, adding authentication (username/password or certificates) would prevent unknown clients from connecting.

4.5.3 Replay Attacks

- On plain MQTT, an attacker could capture packets and resend them easily, since the data is not protected.
- On MQTTS, replay attacks are much harder because TLS encrypts the traffic and protects the connection, so captured packets cannot be reused in a valid way.

4.6 Hardening the MQTT System (Recommendations)

1. **TLS Encryption (Implemented in the test)**
   The main security improvement applied in this project was enabling MQTTS (TLS encryption). Using port 8883, all MQTT messages are now encrypted. This prevents attackers from reading data or topics, even if traffic is captured.
2. **Username/Password Authentication (Recommended)**
   The system currently does not use usernames or passwords. This means any client who knows the broker address can still try to connect. Using authentication would add a strong protection layer by allowing only registered clients to connect.
3. **Access Control Lists (Recommended)**
   Since no ACLs were configured, all connected clients can publish or subscribe to any topic.
   Adding ACLs would restrict what each client can read or write, which prevents data tampering.
4. **Certificate-Based Client Authentication (Advanced Recommendation)**
   Client-side certificates were not used in this project. If added, only devices with a valid certificate would be allowed to connect, making the system resistant to device impersonation.

4.7 Final Results Summary

Security Aspect	Plain MQTT (1883)	Secure MQTT – MQTTS (8883)
Data Encryption	No encryption - data is visible	TLS encrypts all data - unreadable
Topic Visibility	Topics fully visible	Topics hidden (encrypted)
Packet Sniffing Resistance	Vulnerable - attacker can read packets	Protected - only encrypted blobs appear
Replay Attack Resistance	Possible - attacker can resend packets	Much harder - TLS protects the connection
Man-in-the-Middle Protection	None	Provided by TLS handshake
Unauthorized Subscription	Anyone can subscribe if they know the topic	Traffic encrypted but still no access control without authentication
Authentication Support	Optional username/password (not used in test)	Supported (not used in test)
Access Control Lists (ACLs)	Not used	Supported but not implemented
Client Certificate Support	Not possible	Supported for high-security systems (not used in test)
Overall Security Level	Very Low	Medium (high if full hardening applied)

This comparison highlights the difference between unsecured MQTT and secure MQTTS. Plain MQTT transmits all data in plaintext, making it vulnerable to packet sniffing, replay

attacks, and topic visibility. Using MQTTS encrypts all traffic, preventing anyone from reading the payload or topic names. While authentication, ACLs, and client certificates were not

implemented in this test, they are recommended to prevent unauthorized access and ensure only trusted devices communicate with the broker. Overall, MQTTS provides a significantly higher level of security and is the preferred choice for IoT systems.

5. Findings & Analysis

The **analysis** confirms that the primary security risk stems entirely from the use of an unencrypted transport layer, making the system vulnerable to trivial traffic analysis.

5.1. Finding C-01: Critical Plaintext Data Exposure

Description: The payload of all sensor data is directly exposed. This vulnerability was verified via deep packet inspection in Wireshark. Using the default MQTT port 1883, traffic is easily identified and classified as non-secure.

Impact: CRITICAL. Total loss of data confidentiality. All collected telemetry data can be accessed by passive sniffers, enabling intelligence gathering, behavioral analysis, or reconnaissance for targeted attacks.

Evidence (Before Hardening): Wireshark shows plaintext JSON payloads containing temperature, humidity, and topic information.

Analysis: This finding demonstrates the necessity of encryption. Without **TLS**, MQTT PUBLISH messages behave like open broadcasts, leaving data fully exposed.

5.2. Finding H-02: Vulnerability to Topic Manipulation

Description: Due to the lack of client authentication, an attacker who identifies the topic structure via passive sniffing can connect to the broker. They could inject false data, send malicious commands, or consume resources.

Impact: HIGH. Data integrity and availability are compromised. This allows for falsified sensor readings or Denial of Service (DoS) attacks.

5.3. Before and After: The Wireshark View

The following comparison highlights the difference between unsecured MQTT (Port 1883) and secure MQTT over TLS (MQTTS, Port 8883), as observed in Wireshark captures. The “Before” capture shows plaintext JSON data, fully readable, while the “After” capture shows encrypted traffic where both payload and topic information are protected. This illustrates the effectiveness of TLS in securing IoT communications against passive attacks.

Before:

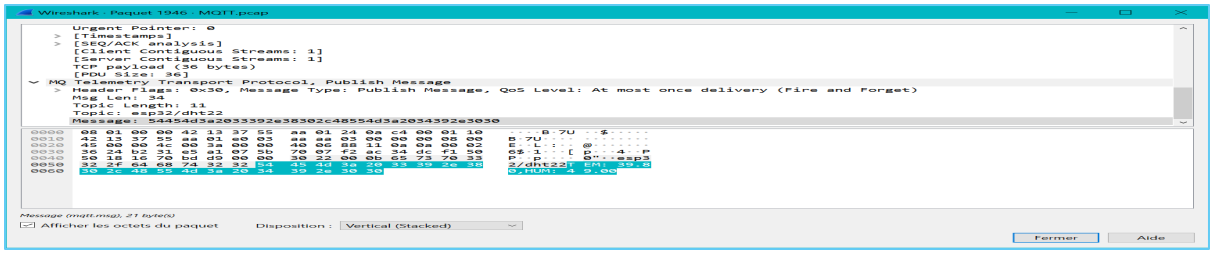

Figure10: Plain MQTT (Port 1883)

The JSON payload containing temperature, humidity, and topic names is fully readable. Shows that data is transmitted in plaintext, vulnerable to sniffing.

After:

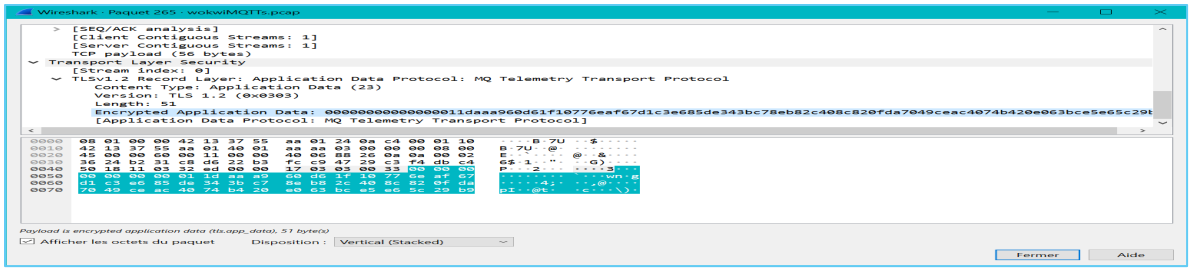

Figure 11 – Secure MQTT (MQTTS, Port 8883)

All application data is encrypted and illegible.

TLS handshake is visible, but payload and topics cannot be read.

- To further summarize the key differences, the following table is provided:

Feature	**BEFORE (Unsecured, Port 1883)**	**AFTER (Secured, Port 8883)**
Primary Protocol	**MQTT**	**TLSv1.2** (Encapsulation)
Payload Visibility (Wireshark)	**Plaintext JSON** (Readable)	**Encrypted Application Data** (Illegible)
Confidentiality Status	**DATA LEAKAGE**	Secured
Attacker Effort to Exploit	Minimal (Standard Sniffer)	Requires Session Key/MITM Attack (High Effort)

Analysis:

The visual proof from Wireshark confirms that using MQTTS effectively hides all sensor data and topic names, making passive sniffing ineffective. While encryption improves confidentiality, additional measures such as authentication and ACLs are recommended to prevent unauthorized access to the broker.

6. Actionable Recommendations & Conclusion

6.1. Actionable Hardening Guide: Eradicating Data Leakage

The following steps must be implemented to secure the MQTT communication channel:

Step	Action Item	Priority	Technical Details
1.	**Migrate to MQTTS (TLS/SSL)**	**IMMEDIATE**	Configure the Mosquitto Broker to disable Port 1883 and listen exclusively on **Port 8883**. Generate and install robust server and client certificates to guarantee mutual authentication.
2.	**Update All Client Configurations**	HIGH	Modify all IoT device firmware/code to connect to the broker using **Port 8883** and mandate certificate verification. This ensures the devices refuse to communicate over the unencrypted channel.
3.	**Implement Strong Authentication**	HIGH	Enforce Username/Password authentication via the mosquitto_passwd utility, restricting connection only to credentialed clients. This prevents unauthorized publishing or subscribing.
4.	**Topic Access Control (ACLs)**	MEDIUM	Apply granular ACL rules to ensure each authenticated client can only publish or subscribe to the topics strictly necessary for its function.

6.2. Conclusion

The **Smart Device Traffic Analysis** confirmed a critical data leakage vulnerability stemming from the use of **unencrypted MQTT**. **Wireshark** analysis provided definitive evidence that sensor data is completely exposed to passive interception. By implementing the TLS/SSL encryption and authentication recommended in this report, the system can eliminate the risk of plaintext data exposure, shifting the security posture from compromised to cryptographically secured.

7. References

- **Wireshark Documentation** – Usage guides for protocol filtering and deep packet inspection. Available at: https://www.wireshark.org/docs/
- **Eclipse Mosquitto Documentation** – Manuals for TLS/SSL certificate setup, Access Control Lists (ACL), and password file management. Available at: https://mosquitto.org/documentation/
- **Wokwi Simulation Platform** – Documentation and simulation environment for ESP32 and MQTT client library configuration. Simulations used in this project:
  - Plain MQTT Publisher (Port 1883): https://wokwi.com/projects/446768000934074369
  - Plain MQTT Subscriber (Port 1883): https://wokwi.com/projects/446768648010527745
  - Secure MQTT Publisher (Port 8883): https://wokwi.com/projects/447226605442451457
  - Secure MQTT Subscriber (Port 8883): https://wokwi.com/projects/447226627794954241
- **Cloudflare: Transport Layer Security (TLS)** – Overview and best practices for TLS encryption. Available at: https://www.cloudflare.com/fr-fr/learning/ssl/transport-layer-security-tls/