

# UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE - SCIENCE AND ENGINEERING

MASTER'S DEGREE COURSE IN COMPUTER SCIENCE

FINAL PROJECT REPORT FOR CYBERSECURITY COURSE

---

---

## MQTT-RED TEAM

---

---

MQTT Vulnerability Analysis: Attacks and Mitigations on ESP32

Emanuele Grasso  
Simone Rinaldi

Enrollment number: 0001141478  
Enrollment number: 0001140193

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Reconnaissance and information gathering</b>	<b>2</b>
1.1 Identifying the Network and Devices . . . . .	2
1.2 MQTT Traffic Analysis . . . . .	2
<b>2 First Breach</b>	<b>4</b>
2.1 Men-in-the-Middle (MitM) Attack . . . . .	4
<b>3 Escalation</b>	<b>5</b>
3.1 Session Hijacking . . . . .	5
3.2 Replay Attack . . . . .	5
<b>4 Connectivity Attacks</b>	<b>7</b>
4.1 Deauthentication Attack . . . . .	7
4.2 Denial of Service (DoS) Attack . . . . .	8
<b>5 Security Improvements</b>	<b>9</b>
5.1 Enhancements on the ESP32 Devices . . . . .	9
5.2 Enhancements on the MQTT Broker . . . . .	9
5.3 Results and Effectiveness of Mitigations . . . . .	9
<b>Conclusion and Future Improvements</b>	<b>10</b>

# Introduction

This study explores the security vulnerabilities of an IoT environment utilizing ESP32 devices and the MQTT protocol, focusing on the risks associated with the absence of encryption and authentication.

By simulating real-world attack scenarios, we can gain valuable insight into the threats that IoT networks face and explore effective defense mechanisms and, to maintain a controlled and ethical testing scope, we deliberately assumed that the Wi-Fi network to which the ESP32 devices were connected was open and without security measures. This decision allowed us to concentrate exclusively on MQTT-related vulnerabilities without extending our assessment to the network infrastructure itself.

Our methodology involved reconnaissance to identify devices and analyze MQTT traffic, followed by a series of targeted attacks, including **Man-in-the-Middle (MitM)**, **Session Hijacking**, **Replay Attacks**, **Deauthentication Attack**, and **Denial of Service (DoS)**.

The objective was to evaluate how an unsecured MQTT environment could be exploited and to understand the impact of these vulnerabilities on system confidentiality, integrity, and availability. By capturing and analyzing MQTT messages, we assessed how attackers could intercept, manipulate, or disrupt communication between IoT devices.

The study highlights the significant risks associated with plaintext MQTT communication and the lack of security mechanisms within an IoT ecosystem. The findings demonstrate how attackers can exploit unsecured brokers to perform MitM attacks, inject malicious commands, or even disrupt service availability.

Through controlled attack simulations, we identified weaknesses that could be leveraged to compromise IoT devices, emphasizing the urgent need for security enhancements. This research serves as a foundation for developing improved security strategies to mitigate the threats posed by unprotected MQTT implementations.

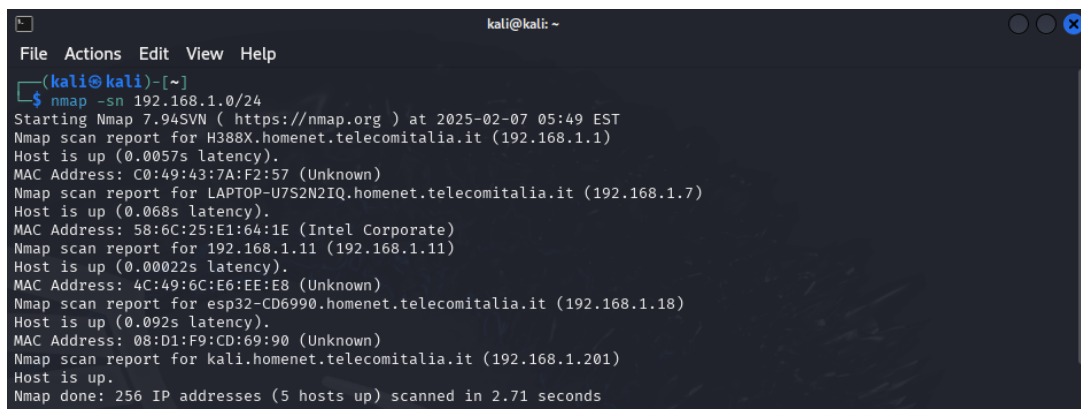
# Reconnaissance and information gathering

## 1.1 Identifying the Network and Devices

Before launching any attack, we begin with a reconnaissance to gather as much information as possible about the target network and connected devices.

We performed an active scan using **Nmap** to identify the ESP32 devices and the MQTT broker and, analyzing the network topology, we discovered essential details such as IP addresses and open ports. The scan results provided insights into the network's structure, including the communication channels used by the ESP32 devices.

To further refine our reconnaissance, we employed ARP scanning to detect active devices and their interactions with the broker. This approach allowed us to map the communication flow and determine the level of encryption applied to MQTT traffic. By identifying unsecured brokers, we established potential points of entry for our simulated attacks.



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ nmap -sn 192.168.1.0/24  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-07 05:49 EST  
Nmap scan report for H388X.homenet.telecomitalia.it (192.168.1.1)  
Host is up (0.0057s latency).  
MAC Address: C0:49:43:7A:F2:57 (Unknown)  
Nmap scan report for LAPTOP-U7S2N2IQ.homenet.telecomitalia.it (192.168.1.7)  
Host is up (0.068s latency).  
MAC Address: 58:6C:25:E1:64:1E (Intel Corporate)  
Nmap scan report for 192.168.1.11 (192.168.1.11)  
Host is up (0.00022s latency).  
MAC Address: 4C:49:6C:E6:EE:E8 (Unknown)  
Nmap scan report for esp32-CD6990.homenet.telecomitalia.it (192.168.1.18)  
Host is up (0.092s latency).  
MAC Address: 08:D1:F9:CD:69:90 (Unknown)  
Nmap scan report for kali.homenet.telecomitalia.it (192.168.1.201)  
Host is up.  
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.71 seconds
```

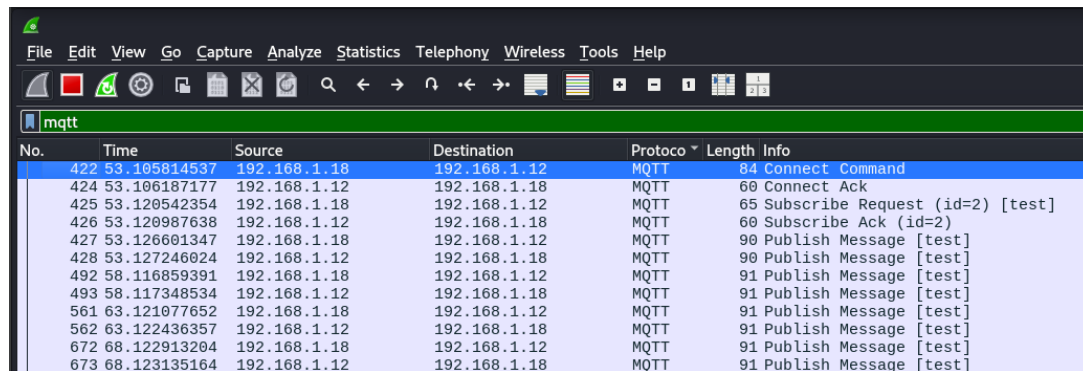
Figure 1.1: Active network scan with Nmap

## 1.2 MQTT Traffic Analysis

Using **Wireshark**, we captured and analyzed MQTT packets exchanged between the ESP32 devices and the broker. When operating with a plaintext MQTT broker, we observed that messages, including topic subscriptions and payload data, were transmitted in an unencrypted format. This exposure makes the system highly vulnerable to eavesdropping and data manipulation attacks.

To further analyze the security differences, we tested an MQTT broker configured with TLS encryption and authentication mechanisms. Unlike the plaintext broker, the TLS-protected broker ensured

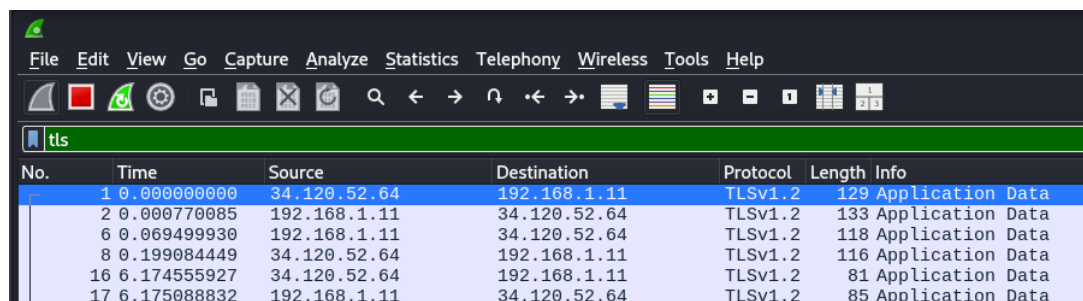
that all data was securely transmitted, preventing unauthorized interception.



The image shows a Wireshark packet capture for the MQTT protocol. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help), a toolbar with various icons, and a packet list table. The table has columns for No., Time, Source, Destination, Protocol, Length, and Info. The selected packet is number 422, a Connect Command from 192.168.1.18 to 192.168.1.12.

No.	Time	Source	Destination	Protocol	Length	Info
422	53.105814537	192.168.1.18	192.168.1.12	MQTT	84	Connect Command
424	53.106187177	192.168.1.12	192.168.1.18	MQTT	60	Connect Ack
425	53.120542354	192.168.1.18	192.168.1.12	MQTT	65	Subscribe Request (id=2) [test]
426	53.120987638	192.168.1.12	192.168.1.18	MQTT	60	Subscribe Ack (id=2)
427	53.126601347	192.168.1.18	192.168.1.12	MQTT	90	Publish Message [test]
428	53.127246024	192.168.1.12	192.168.1.18	MQTT	90	Publish Message [test]
492	58.116859391	192.168.1.18	192.168.1.12	MQTT	91	Publish Message [test]
493	58.117348534	192.168.1.12	192.168.1.18	MQTT	91	Publish Message [test]
561	63.121077652	192.168.1.18	192.168.1.12	MQTT	91	Publish Message [test]
562	63.122436357	192.168.1.12	192.168.1.18	MQTT	91	Publish Message [test]
672	68.122913204	192.168.1.18	192.168.1.12	MQTT	91	Publish Message [test]
673	68.123135164	192.168.1.12	192.168.1.18	MQTT	91	Publish Message [test]

Figure 1.2: Packets exchanged seen with Wireshark



The image shows a Wireshark packet capture for the TLS protocol. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help), a toolbar with various icons, and a packet list table. The table has columns for No., Time, Source, Destination, Protocol, Length, and Info. The selected packet is number 1, an Application Data packet from 34.120.52.64 to 192.168.1.11.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	34.120.52.64	192.168.1.11	TLSv1.2	129	Application Data
2	0.000770085	192.168.1.11	34.120.52.64	TLSv1.2	133	Application Data
6	0.069499930	192.168.1.11	34.120.52.64	TLSv1.2	118	Application Data
8	0.199084449	34.120.52.64	192.168.1.11	TLSv1.2	116	Application Data
16	6.174555927	34.120.52.64	192.168.1.11	TLSv1.2	81	Application Data
17	6.175088832	192.168.1.11	34.120.52.64	TLSv1.2	85	Application Data

Figure 1.3: Packets exchanged in TLS seen with Wireshark

Through our reconnaissance and traffic analysis, we identified several weaknesses in MQTT communication, particularly in unsecured environments. These findings served as the foundation for subsequent attack simulations, enabling us to exploit vulnerabilities effectively and assess their impact on the integrity of the system.

# First Breach

## 2.1 Men-in-the-Middle (MitM) Attack

In this scenario, we positioning ourselves between the two communicating ESP32 devices and the MQTT broker, intercepting the transmitted messages.

To achieve this, we used ARP Spoofing, a technique that manipulates the **Address Resolution Protocol** to associate the attacker's MAC address with the broker's IP address. This deception allows the attacker to redirect MQTT traffic through their system, enabling message interception. We executed ARP Spoofing using a Python script with the library *Scapy*, and confirmed that the traffic was successfully rerouted.

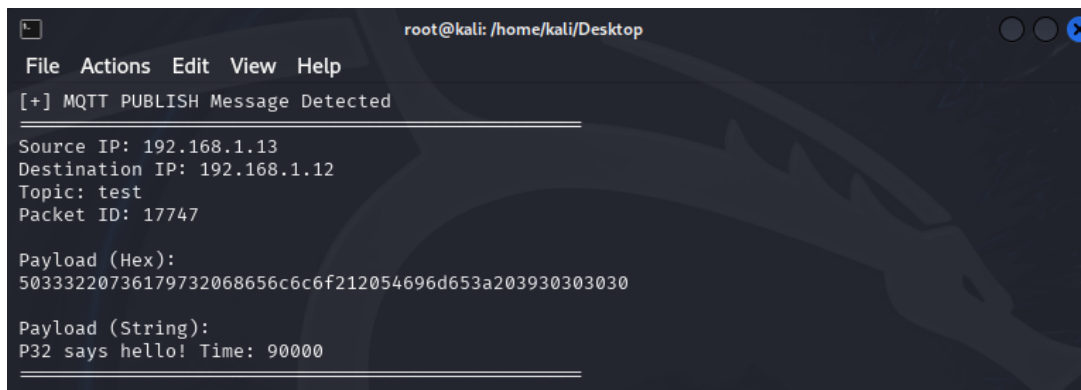


Figure 2.1: Results of the script for Man in the Middle Attack

Once in position, we monitored MQTT packets exchanged between the ESP32 devices. Since the broker was operating in plaintext mode, we were able to capture unencrypted MQTT topics and payloads. This demonstrated a significant security risk, as sensitive information could easily be extracted and misused.

In the next chapters, we introduce a more advanced MitM scenario with different types of attacks. By intercepting a control command from one ESP32 device and modifying its payload, we could be able to alter the device's behavior. This experiment highlighted how attackers could manipulate IoT systems for malicious purposes, such as controlling smart home devices or disrupting industrial automation.

However, when we attempted the same attack on the TLS-secured MQTT broker, all intercepted packets appeared encrypted, making it impossible to extract meaningful data or inject malicious content. This confirmed that enabling TLS encryption effectively mitigates MitM attacks by preventing unauthorized interception and modification of MQTT messages.

# Escalation

## 3.1 Session Hijacking

In our experiment, in the textplain broker, we found that MQTT does not maintain persistent session states for clients, meaning that traditional session hijacking was not feasible. Instead, we exploited the lack of authentication and encryption to impersonate an ESP32 device. To achieve this, we first launched a [Wi-Fi Deauthentication Attack](#) (described in the next section) to forcefully disconnect one of the ESP32 devices from the network.

Once the target ESP32 was disconnected, we executed ARP poisoning to intercept and manipulate network traffic. By responding to ARP requests with a forged MAC address, we made the MQTT broker believe that our attacking device was the legitimate ESP32 client. This allowed us to establish an unauthorized MQTT connection using the disconnected ESP32's topics.

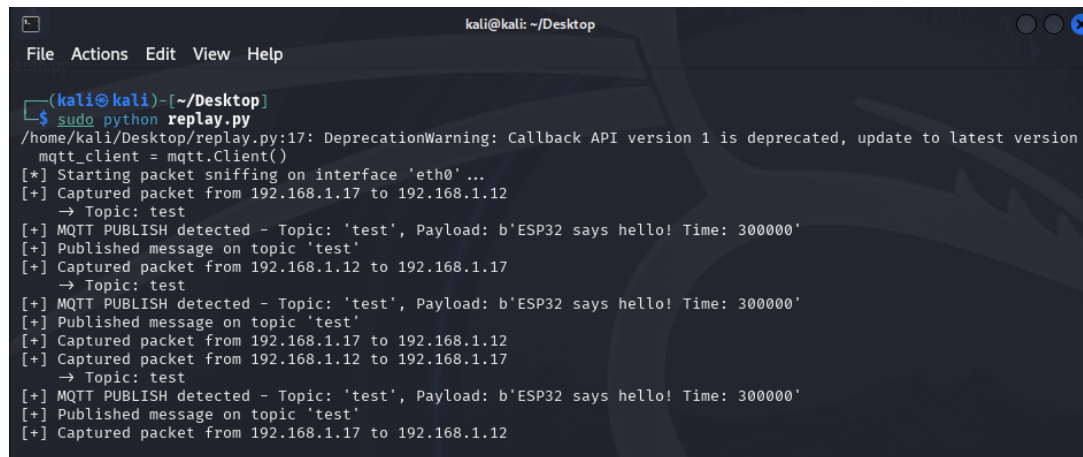
By impersonating the device, we successfully subscribed to sensitive topics, injected custom messages, and disrupted the intended communication flow. This attack highlighted the inherent vulnerability in open MQTT networks where there is no mechanism to validate the legitimacy of a connecting client. Implementing TLS encryption and enforcing authentication at the broker level would significantly mitigate the risk of this attack.

## 3.2 Replay Attack

Replay attacks leverage the absence of integrity checks in unsecured MQTT setups by capturing and resending legitimate messages. In our experiment, we used [Wireshark](#) and [Nmap](#) to first identify an active ESP32 device communicating with the MQTT broker. Once we captured relevant packets, we deployed a Python-based replay attack using the **Scapy** library, which allowed us to intercept MQTT control messages and resend them to the broker.

Our approach consisted of two phases: first, we captured operational commands directed at an ESP32 sensor, storing the packets for later use. These commands included basic MQTT publish messages that control the state of the device. In the second phase, we retransmitted these captured packets, effectively forcing the ESP32 to execute repeated or outdated commands. Since the broker lacked mechanisms such as timestamps or nonce verification, it processed the replayed packets without distinguishing them from legitimate ones, leading to unexpected device behavior.

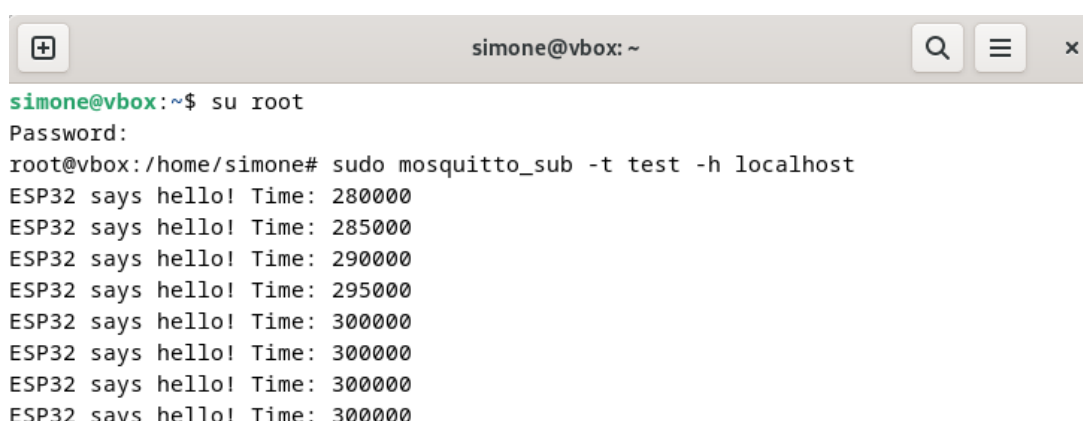
The results demonstrated the severity of replay attacks in unsecured IoT environments, highlighting the need for additional security mechanisms. To mitigate such risks, we recommend implementing message timestamping, nonce-based authentication, or challenge-response verification at the application level. Moreover, enabling TLS encryption prevents attackers from passively capturing MQTT packets for later retransmission.



```
kali@kali: ~/Desktop
File Actions Edit View Help

(kali@kali)-[~/Desktop]
$ sudo python replay.py
/home/kali/Desktop/replay.py:17: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
mqtt_client = mqtt.Client()
[*] Starting packet sniffing on interface 'eth0'...
[+] Captured packet from 192.168.1.17 to 192.168.1.12
    → Topic: test
[+] MQTT PUBLISH detected - Topic: 'test', Payload: b'ESP32 says hello! Time: 300000'
[+] Published message on topic 'test'
[+] Captured packet from 192.168.1.12 to 192.168.1.17
    → Topic: test
[+] MQTT PUBLISH detected - Topic: 'test', Payload: b'ESP32 says hello! Time: 300000'
[+] Published message on topic 'test'
[+] Captured packet from 192.168.1.17 to 192.168.1.12
[+] Captured packet from 192.168.1.12 to 192.168.1.17
    → Topic: test
[+] MQTT PUBLISH detected - Topic: 'test', Payload: b'ESP32 says hello! Time: 300000'
[+] Published message on topic 'test'
[+] Captured packet from 192.168.1.17 to 192.168.1.12
```

Figure 3.1: Result of the script for Replay Attack seen by the attacker



```
simone@vbox: ~
+  simone@vbox: ~
simone@vbox:~$ su root
Password:
root@vbox:/home/simone# sudo mosquitto_sub -t test -h localhost
ESP32 says hello! Time: 280000
ESP32 says hello! Time: 285000
ESP32 says hello! Time: 290000
ESP32 says hello! Time: 295000
ESP32 says hello! Time: 300000
ESP32 says hello! Time: 300000
ESP32 says hello! Time: 300000
ESP32 says hello! Time: 300000
```

Figure 3.2: Result of the script for Replay Attack seen by the victim

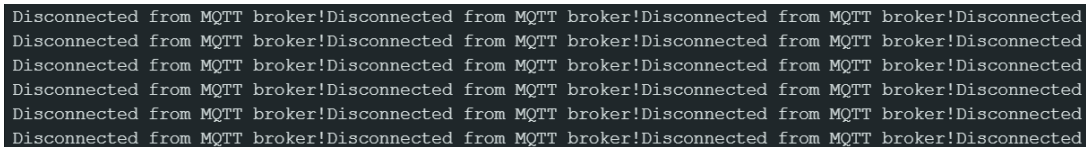




## 4.2 Denial of Service (DoS) Attack

Our DoS attack targeted the ESP32 devices rather than the broker, aiming to overwhelm their processing capacity by flooding them with MQTT messages. Using [Nmap](#), we identified the broker's IP and active ESP32 clients. We then employed a Python script utilizing the **Paho-MQTT** library to publish a massive volume of messages to multiple topics, consuming the ESP32's limited resources.

By rapidly sending large-sized messages at high frequencies, the ESP32 devices experienced severe performance degradation, delayed processing, and crashes. The lack of rate-limiting or resource management at the device level made them particularly vulnerable to this type of attack.



```
Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!
Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!
Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!
Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!
Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!Disconnected from MQTT broker!
```

Figure 4.3: Result of the script for DoS Attack seen by the victim

To mitigate these risks, it is essential to implement QoS (Quality of Service) policies, set message rate limits, and introduce device-level memory protections to prevent resource exhaustion. Additionally, utilizing authentication and authorization mechanisms at the broker level can help restrict unauthorized high-frequency message publication from untrusted clients.

# Security Improvements

After identifying the various vulnerabilities in our MQTT-based IoT setup, we implemented a series of improvements at both the ESP32 device level and the broker configuration to mitigate these attacks.

## 5.1 Enhancements on the ESP32 Devices

First of all each ESP32 device was configured with unique authentication credentials to ensure that only legitimate clients could connect to the broker and we enabled TLS encryption on the ESP32 MQTT clients to prevent unauthorized interception of MQTT messages.

A nonce-based challenge-response mechanism was introduced to protect against replay attacks by ensuring that each message had a unique timestamp and, to defend against DoS attacks, we implemented message rate limiting and memory management mechanisms to prevent resource exhaustion from excessive message flooding.

## 5.2 Enhancements on the MQTT Broker

The broker was configured to require TLS encryption and user authentication to prevent unauthorized access and mitigate MitM attacks and ACLs were implemented to restrict topic subscriptions and publications, ensuring that only authorized clients could access specific MQTT topics.

The broker was set up to enforce session timeouts and use token-based authentication to prevent unauthorized reconnections after session hijacking and the broker was configured to limit the number of connection requests per client and introduced Quality of Service (QoS) policies to prevent message flooding from overloading the system.

## 5.3 Results and Effectiveness of Mitigations

After applying these security improvements, we conducted testing to assess their effectiveness:

- The use of TLS encryption and authentication prevented unauthorized interception of MQTT messages, avoiding the Man-in-the-Middle (MitM) and Session Hijacking Attacks.
- The challenge-response mechanism with unique timestamps successfully blocked replayed messages from being processed, avoiding the Replay Attack.
- By implementing rate limiting and broker-side QoS policies, the ESP32 devices remained operational even under high message traffic conditions, avoiding DoS Attack.

By implementing these security measures, we significantly improved the robustness of our MQTT-based IoT environment, reducing its susceptibility to common cyber threats. Future improvements could include further enhancing anomaly detection mechanisms and integrating hardware-based security modules to reinforce device authentication and encryption.

# Conclusion and Future Improvements

After evaluating the vulnerabilities in an unsecured MQTT ecosystem, we implemented key security improvements, including **TLS encryption**, **mandatory authentication**, **access control lists (ACLs)**, **rate limiting**, and **anti-replay mechanisms**. These measures significantly mitigated the risks of MitM, replay, and DoS attacks, improving the overall resilience of the system. However, as cybersecurity is a constantly evolving field, further enhancements are necessary to ensure long-term protection against emerging threats.

One promising avenue is the integration of AI-driven anomaly detection systems to monitor MQTT traffic in real time, identifying suspicious patterns and blocking malicious activity before an attack is executed. Advanced authentication mechanisms, such as device identity verification through digital certificates or secure hardware modules like TPM (Trusted Platform Module), could further restrict access to only authorized devices.

Additionally, enhancing session management by enforcing mutual TLS authentication, persistent sessions, and automatic token revocation upon detecting anomalies could prevent unauthorized access and session hijacking attempts. Beyond securing MQTT itself, improvements at the network level could further protect IoT devices. Although we assumed an open Wi-Fi environment in our study, real-world deployments should adopt WPA3 with enterprise authentication to mitigate deauthentication attacks.

Network segmentation and software-defined networking (SDN) could be used to dynamically regulate device access, reducing the risk of lateral movement by attackers. Strengthening application-layer security by enforcing message integrity checks, signed access tokens, and timestamp-based validation could further protect against injection and replay attacks.

Lastly, advanced logging and forensic analysis should be incorporated to enhance threat detection and incident response. Integrating an SIEM (Security Information and Event Management) system to continuously monitor MQTT sessions could help detect unauthorized access attempts and ongoing attacks in real time.

By combining AI-based anomaly detection, hardware-based authentication, and enhanced network security policies, future IoT environments can achieve a higher level of resilience against increasingly sophisticated cyber threats.