Developing an Open Source Tool for Man-in-the-Middle (MitM) Attacks on the MQTT Protocol

Daniel Feito-Pin and Jose Losada

Faculty of Computer Science, Universidade da Coruña, 15071 A Coruña, Spain
Correspondence: daniel.feito.pin@udc.es

DOI: *https://doi.org/10.17979/spu.23.c24*

Abstract: IoT has facilitated the connection of millions of devices, creating an interconnected ecosystem of machines that collect, process, and transmit large amounts of data. This ecosystem is becoming a target of interest for the cybercriminals because many of the devices within it lack robust security protocols and measures. The resulting vulnerabilities, while already posing risks to users of these technologies, can have devastating impacts at the industrial level. The objective of this work is to develop a new tool that allows the alteration of MQTT traffic, a popular IoT protocol, in conjunction with other known technologies and techniques to carry out Man-in-the-Middle attacks, providing a new option to understand this kind of attack and to test the resistance of the communications.

1 Introduction

The number of devices connected to the Internet is rapidly increasing as new technologies are developed and deployed in the industry domain. At the same time, in the home environment, the popularity of IoT devices continues to expand among both tech enthusiasts and users who lack technical knowledge. Nevertheless, this interconnected scenario poses significant security challenges, as many of these devices were designed to be as efficient as possible and often lack proper security measures. This is the case of MQ Telemetry Transport (MQTT), a popular protocol widely used in IoT that was designed to be extremely lightweight.

While MQTT allows the use of usernames and passwords, these credentials are susceptible to interception, as the protocol's packets travel through the network in plaintext. Furthermore, this condition allows for tampering with any field in the MQTT messages using techniques or attacks such as Man-in-the-Middle (MitM). Additional security can be achieved by implementing Transport Layer Security (TLS) or encrypting the communications. However, devices with low computing power might not be able to implement these measures.

In an era in which crime tends towards digitization, the cybercrimes related to interference with data and systems grow year after year, as shown in Figure 1. With cyberattacks becoming increasingly frequent and sophisticated, systems lacking protection are completely vulnerable to cybercriminals. In light of this situation, it is crucial to understand how the used technologies work, as well as their weaknesses and the options available to users to verify their correct operation. For this reason, this work aims to illustrate how an attacker could alter the messages of a known protocol, using MQTT as a case study, and to provide an open-source tool that allows users to verify the robustness of communications using this protocol in its default configuration without additional security measures.

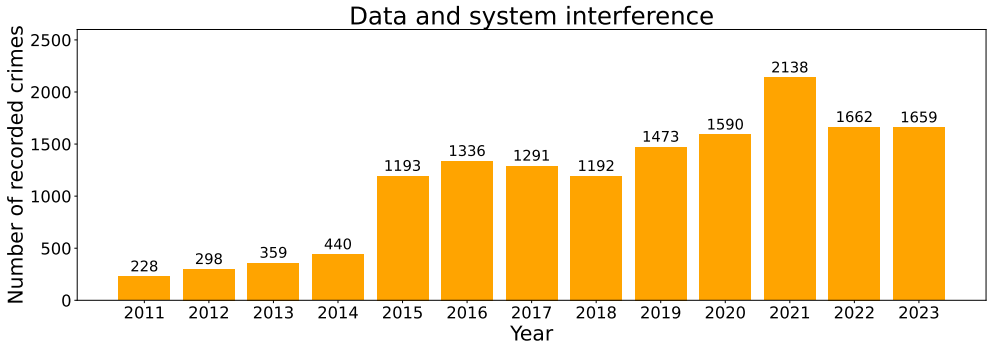

Figure 1: Number of recorded crimes by year inside the "data and system interference" criminal group from the Crime Statistics Portal of the Ministry of the Interior, Government of Spain (2025).

2 Objectives

The primary objective of this work is to develop a tool for executing Man-in-the-Middle attacks on networks using the MQTT protocol, enabling the interception, modification, and re-injection of messages in communication between MQTT clients and brokers to assess vulnerabilities and demonstrate potential attacks.

The specific objectives are as follows:

- Develop a tool that allows data to be altered before it reaches its target.
- Contribute to the repertoire of tools for analyzing and testing IoT technologies.
- Produce an open-source tool, facilitating its use and allowing for its improvement and continuity in the future.

3 Background

3.1 Attacks

ARP Poisoning This type of attack exploits the lack of authentication in the Address Resolution Protocol (ARP) by sending fraudulent ARP packets to poison the victims' ARP caches, causing them to transmit packets to the attackers instead of the intended recipient, and making the modification of these packets possible before they resend them to the intended destination. These ARP reply packets contain the attacker device's physical address (MAC address) so that the targeted system registers it as the destination of the spoofed one. Specifically, this *poisoning* of the ARP cache refers to the tampered pairing of the legitimate network address (IP address) with the attacker's physical address. Additionally, as illustrated in Figure 2, the attackers can send *gratuitous ARP reply* packets that do not correspond to any ARP request, allowing them to perform this attack at any time without the need to wait for the receipt of an ARP request packet.

Man-in-the-Middle In this class of attack, an attacker positions a controlled device among the communications of the targeted systems to then intercept or modify the data shared by them. The most notable feature of this type of attack is that it remains transparent to the victim, allowing sensitive data to be analyzed or altered in real-time, thereby going unnoticed by the attacked users or system that does not implement proper security measures. Moreover, this technique can be performed unidirectionally between two systems, as shown in Figure 3, but also in bidirectional communications or with multiple participants.

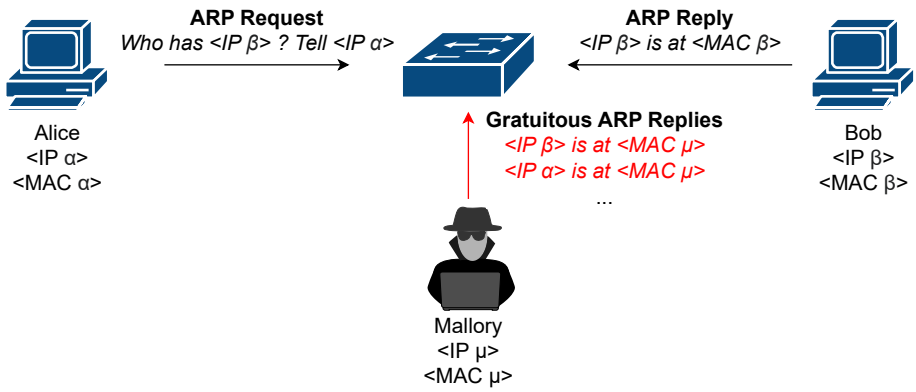

Figure 2: Representation of an ARP Poisoning attack.

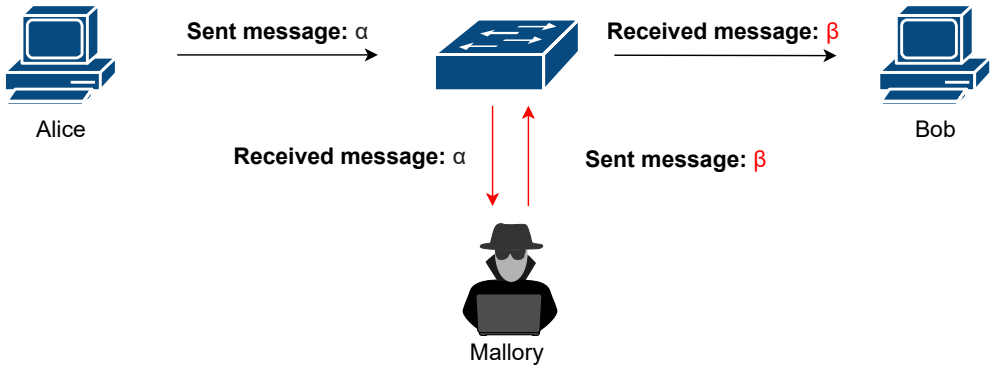

Figure 3: Representation of a Man-in-the-Middle attack.

3.2 The MQTT protocol

MQTT is a standard messaging protocol developed by the Organization for the Advancement of Structured Information Standards (OASIS) that uses a publish/subscribe communication model.

This publish/subscribe model utilizes memory spaces denominated topics managed by a central server, known as a broker, which is responsible for handling connections from the clients that request to receive the messages sent to specific topics (subscribers), and the clients that post them (publishers). Consequently, as shown in Figure 4, there is no need for direct communication between the publishers and the subscribers.

As previously stated, one of this protocol's most distinctive features is its extremely lightweight design, which makes it an excellent option for devices belonging to constrained environments in need of minimal code footprint or low network bandwidth (Banks et al., 2019). However, this design decision came with a trade-off in terms of safety, as MQTT's principal security measure relies on using Transport Layer Security (TLS) to encrypt the MQTT messages, which would otherwise travel across the network in plaintext.

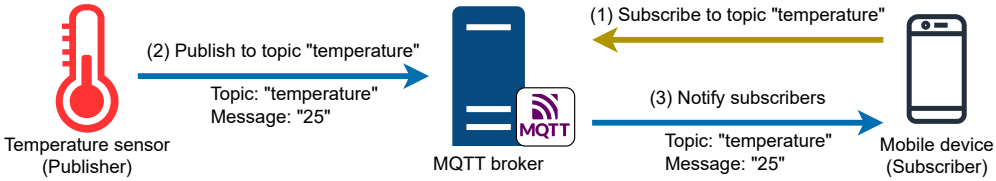

Figure 4: Representation of MQTT's publish-subscribe model.
One device subscribes to the desired topics (1), then another device publishes data to the topic (2), and finally the broker receives this message and forwards it to all subscribed clients (3).

Lastly, it is worth mentioning that the structure of an MQTT message is composed of three main components: a fixed header, a variable header that depends on the type of packet, and the payload. As shown in Figure 5, the first 4 bits of the header determine the packet type, allowing 16 possible values, of which 14 are used (values 1 to 14) and 2 are reserved (values 0 and 15). Of these types of MQTT messages, the values 1 and 3, designated as `CONNECT` and `PUBLISH`, respectively, are crucial to this work, as the former packets might contain a client's credentials and the latter include the messages sent to the topics.

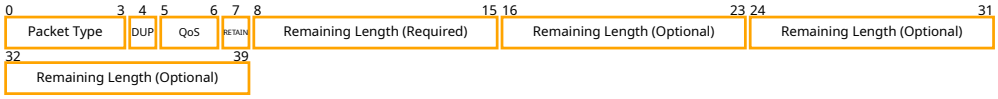

Figure 5: MQTT fixed header.

4 State of the art

There is a wide range of tools designed to exploit the vulnerabilities of the MQTT protocol, with their principal approaches including credential brute-forcing tools to attempt to access MQTT brokers, fuzzing, code or command injections, denial-of-service attacks to overwhelm the broker or clients, and topic enumeration. However, there are not many of these that follow the Man-in-the-Middle approach.

While it is possible to tamper with MQTT messages using tools designed to alter the network traffic, in the manner of packet stream editors such as `Netsed` (de Galbert, n.d.), or MitM tools like `ettercap` (Ettercap Project, n.d.), their use is limited when trying to modify the original content with another of a different length, as illustrated in Table 1. This happens because these kinds of tools manipulate the incoming network traffic by applying substitution rules in a way that merely replaces the data contents without correcting the fields of the underlying layers of protocols that may need to be fixed.

Table 1: Results of using `Netsed` to alter MQTT messages.

Substitution Rule	Sent Value	Received Value	Expected Value	Result
s/TestX/TestY	TestX	TestY	TestY	✓
s/Test123/Test1234	Test123	Test123	Test1234	✗
s/Test987/Test98	Test987	Test98 ?	Test98	✗

For their part, several of the solutions designed to perform MitM attacks that support MQTT

follow approaches that are too specific in scope, like the BERT-based adversarial model presented by Wong and Luo (2020), or the method of use requires too much human intervention to be a practical tool, in the style of IOXY's MQTT interception proxy (Rigas, 2020).

In contrast to the above solutions, the developed tool takes into account these limits and provides a rule-based approach to process the received packets and tamper with them by using a set of predefined methods to replace their topics, payloads, or both, regardless of the lengths of the original and modified data.

5 Materials

The created tool, named `mqtwister`, was entirely developed in `Python`, using the `Scapy` library to work with network packets (Scapy community, 2025), the `psutil` module to implement auxiliary utilities (Rodola, 2025), and the `pytest` framework for testing (Krekel and pytest-dev team, 2015). Furthermore, for testing purposes, a development environment was created, composed of four virtual machines: an attacker device, a broker, a publisher, and a subscriber. This attacker machine consisted of a Kali Linux in which the tool was executed. For the broker and clients, the Eclipse Mosquitto broker and clients were chosen due to their open source nature, popularity, and versatility (Eclipse Mosquitto, 2018). Moreover, a `Vagrantfile`, a minimal `Mosquitto` configuration, and provision scripts for the broker and clients were configured, aiming for a quick, automatic, and deterministic deployment of the MQTT scenario with `Vagrant`.

6 Architecture

The tool was designed utilizing a modular approach that covered the following four well-distinguished tasks:

Language management A module was dedicated to controlling the language of the tool (currently available in English, Spanish, and Galician) by keeping various dictionaries with all the strings used to write to both the standard output and the error output used by the logger.

User interface Another module was used to display the interactive interface, manage tool configuration, receive user input, and start the functions corresponding to the selected options.

MQTT traffic processing The principal pillar of this work. This module sets the sniffer, manages user-defined rules, and processes the received MQTT traffic. In turn, it was divided into two submodules:

- Submodule for rule managing: A system of rules was implemented to compare against the topic and payload from the processed packets. This rules follow the syntax `item="value" item.action(args)`, where `item` can either be `topic` or `payload`, `value` contains a string to be used as a regular expression to conduct the matching, and `action` corresponds to one of the predefined functions' name which would be applied to the specified `item` and any arguments it can accept (`args`) which must be valid `Python` literals.
- Submodule for message tampering: Using `Scapy`'s asynchronous sniffer, upon arrival of MQTT messages, the tool determines whether the MQTT packet type is `CONNECT` or `PUBLISH`. In the case of `CONNECT` packets, the fields for `Client ID`, `User Name`, and `Password` are registered for further consultation. On the other hand, when a `PUBLISH` packet is received, its topic and value are retrieved and subsequently checked against the rules list to be processed accordingly. This submodule also corrects the underlying protocols' fields, such as lengths or checksums, and reverses the MAC Spoofing caused

by the ARP Poisoning by restoring the MAC addresses to the destination's correct value before forwarding the processed packets into the network.

Auxiliary utilities This module contains complementary functionalities such as configuring the tool's log system and network functions.

7 Usage

The process of using the developed tool is detailed below:

- First, the user configures the `config.py` to modify any preferences, such as the tool's default values for language (including English and Spanish), network interface, and logging level.
- Secondly, the tool is executed as a Python package with the command `python -m mqtwister`. Once started, the user can use an interactive menu to set the configuration to use in the attack. This includes a list of user-defined rules to compare against the received MQTT messages, which contain the match criteria and the actions to apply to the packets' topic or payload.
- Once configured, the user can start the tool's sniffer to begin processing the received packets. To be able to do this, the attacker device needs to be interposed between the targets' communications. Tools like `ettercap` can be utilized for this purpose. The code snippet shown below contains a filter that logs and drops the received MQTT traffic (assuming the default port, 1883). With this filter, `ettercap` won't forward the MQTT packets, leaving its processing to `mqtwister`, and keeping the original messages from reaching their destination without applying changes to the device's operating system or kernel. This filter can be compiled using the command `etterfilter mqtt_filter.ecf -o mqtt_filter.ef` and then used by `ettercap` to launch an ARP Poisoning to stablish the attacking machine between the targets communication right before starting the Man-in-the-Middle attack. It's worth mentioning that one of the targets should be the broker, since there's no direct communication between clients. If the other target is a subscriber, the messages will be modified for that client. On the contrary, if it is a publisher, the modified messages will be received by all clients subscribed to the topic to which they're sent.

```
# Filename: mqtt_filter.ecf
if (ip.proto == TCP && tcp.src == 1883) {
    msg("\nReceived packet with src port 1883.\n");
    drop();
}
if (ip.proto == TCP && tcp.dst == 1883) {
    msg("\nReceived packet with dst port 1883.\n");
    drop();
}
```

After this process, the tool is ready to collect credentials and modify the MQTT traffic based on the user-defined substitution rules.

8 Results & discussion

This work resulted in a new open-source tool capable of processing MQTT traffic for credential harvesting and automatic packet modification in real time using user-defined substitution rules. This solution adds another option to the repertoire of tools that can be used to verify

the integrity and security of IoT systems, overcoming the limitations of other tools, such as the substitution of data strings with others of different lengths.

As an open-source tool, the developed code is published under a license that grants permission to inspect, use, modify, and distribute it. The initial intention was to license this project under the GNU General Public License version 3 (GPL-3.0) due to its enhanced legal protections, ethical considerations, and long-term sustainability. However, to ensure compliance and compatibility with all dependencies, mqtwister was licensed under the GNU General Public License version 2 (GPL-2.0) to comply with the licenses of its dependencies: `Scapy` (GPL-2.0), `psutil` (BSD-3-Clause), and `pytest` (MIT). The developed code, as well as the configurations used for the automatic deployment of the testing environment with `Vagrant`, were published in the following GitHub repository (Feito-Pin, 2025):

https://github.com/danielfeitopin/mqtwister

Bibliography

A. Banks, E. Briggs, K. Borgendale, and R. Gupta. MQTT Version 5.0. *https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html*, 2019. [Online; accessed 26-September-2025].

J. V. de Galbert. Netsed — Silicone's web. *http://silicone.homelinux.org/projects/netsed/*, n.d. [Online; accessed 24-September-2025].

Eclipse Mosquitto. Eclipse Mosquitto. *https://mosquitto.org/*, 2018. [Online; accessed 26-September-2025].

Ettercap Project. Ettercap Home Page. *https://www.ettercap-project.org/index.html*, n.d. [Online; accessed 24-September-2025].

D. Feito-Pin. MQTwister. *https://github.com/danielfeitopin/mqtwister*, 2025. [Online; accessed 28-September-2025].

H. Krekel and pytest-dev team. pytest documentation. *https://docs.pytest.org/en/stable/*, 2015. [Online; accessed 28-September-2025].

Ministry of the Interior, Government of Spain. Crime Statistics Portal. *https://estadisticasdecriminalidad.ses.mir.es/publico/portalestadistico/datos?type=jaxi&title=Cibercriminalidad&path=/Datos5/*, 2025. [Online; accessed 21-September-2025].

T. Rigas. Introducing IOXY: an open-source MQTT intercepting proxy – NVISO Labs. *https://en.wikipedia.org/w/index.php?title=LaTeX&oldid=413720397*, 2020. [Online; accessed 26-September-2025].

G. Rodola. giampaolo/psutil: Cross-platform lib for process and system monitoring in python. *https://github.com/giampaolo/psutil*, 2025. [Online; accessed 28-September-2025].

Scapy community. Scapy. *https://scapy.net/*, 2025. [Online; accessed 28-September-2025].

H. Wong and T. Luo. Man-in-the-middle attacks on mqtt-based iot using bert based adversarial message generation. In *KDD 2020 AIoT Workshop*, pages 130–145, 2020.