DOI: 10.1109/CSNet64211.2024.10851733

Pirates of the MQTT: Raiding IIoT Systems with a Rogue Client

Wael Alsabbagh[1,2], Samuel Amogbonjaye[1], Chaerin Kim[1,2] and Peter Langendörfer[1,2]
[1]*Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany*
[2]*IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany*
E-mail: (Wael.alsabbagh, Chaerin.kim, amogbolu, peter.Langendoerfer)@b-tu.de

***Abstract*—The integration of MQTT (Message Queuing Telemetry Transport) in Industrial Internet of Things (IIoT) systems enhances operational efficiency but introduces significant security vulnerabilities, particularly through rogue MQTT clients. These clients exploit MQTT weaknesses to disrupt industrial processes and compromise data integrity. This paper examines the impact of rogue MQTT client attacks, demonstrated through a detailed case study using the Fischertechnik Lernfabrik 4.0. We highlight how attackers can exploit MQTT's inherent vulnerabilities, including anonymous connections and message retention, to inject false data and interfere with operations.**

To address these risks, we propose several mitigation strategies: implementing fine-grained authorization for topic access control, enhancing replay attack protection with Message Authentication Code (MAC), employing mutual TLS (mTLS) for secure client authentication, and incorporating real-time client activity monitoring and anomaly detection. These measures aim to bolster IIoT system security, mitigate potential disruptions, and maintain operational integrity. Our findings and recommendations contribute to advancing security practices in MQTT-based IIoT environments. All attack codes and a proof-of-concept are publicly available.

***Index Terms*—Rogue Client; MQTT Protocol; Cyberattacks; Cybersecurity.**

I. Introduction

The integration of Industrial Internet of Things (IIoT) systems has revolutionized modern industrial operations, delivering unprecedented levels of efficiency and adaptability [1], [2]. Central to this transformation is the MQTT (Message Queuing Telemetry Transport) protocol [5], a lightweight, publish-subscribe messaging protocol designed for real-time, low-bandwidth communication across a myriad of IoT devices. Its minimal overhead and effective data handling make MQTT essential for the seamless connectivity of various industrial components, including smart sensors, Programmable Logic Controllers (PLCs), Human-Machine Interfaces (HMIs), and end-users, thus enhancing the operational coherence of IIoT systems.

However, this widespread adoption of MQTT in IIoT environments introduces significant security vulnerabilities [11], [12]. The very connectivity that enhances operational efficiency also exposes these systems to sophisticated threats. A particularly concerning issue is the exploitation of MQTT's inherent weaknesses by rogue clients. These malicious entities can disrupt industrial processes and manipulate system behaviors by exploiting the protocol's vulnerabilities, leading to compromised data integrity, erroneous operational decisions, and severe security breaches.

In this paper, we delve into the impact of rogue MQTT clients on MQTT-based IIoT systems, highlighting how these attacks can exploit protocol vulnerabilities and disrupt industrial operations. We present a comprehensive attack scenario using the Fischertechnik Lernfabrik 4.0 [1] as a testbed, simulating various attack vectors to underscore the profound risks associated with these threats.

Our attack approach involves several key steps:

- **Compromising the MQTT Broker:** We first exploit weaknesses in the MQTT broker's security, including directory fuzzing and static analysis of control software, to gain unauthorized access and look out the current broker settings.
- **Offline Pre-Attack Phase:** In this phase, we capture and record MQTT packets to prepare for the main attack. We focus on exploiting the MQTT protocol's wildcard topic vulnerability to sniff all messages and store critical data for later use.
- **Denial of Service (DoS) Attack:** We then launch a DoS attack to disconnect legitimate MQTT clients, ensuring that our attack can proceed unimpeded. This involves publishing malicious payloads to interrupt client communication and using MQTT's retain flag to persist the disruption.
- **Online Attack Phase:** Finally, with legitimate clients disconnected, we introduce a rogue MQTT client to execute the main attack. This phase includes replaying pre-recorded MQTT messages to simulate normal operations and performing a file upload attack via Secure Shell/Secure Copy Protocol (SSH/SCP) to manipulate factory processes and induce system instability.

To address these vulnerabilities, we propose a set of security recommendations aimed at bolstering IIoT systems against such threats. These include enhancing protocol security, implementing fine-grained authorization mechanisms, and improving system monitoring for anomaly detection. By providing

[1]https://www.fischertechnikwebshop.com/en-gb/trainingfactory-4-0-en-gb

978-1-7281-9023-5/21/$31.00 ©2021 IEEE

insights into these attack methods and their impacts, our work aims to advance the security of IIoT ecosystems. Additionally, our attack codes and proof-of-concept demonstrations are publicly available [16] [17], encouraging further research and collaborative efforts to strengthen defenses against emerging threats in IIoT security.

The rest of the paper is structured as follows. Section II discusses related works, while Section III presents the experimental setup we used in this paper. In Section IV, we provide an overview of both MQTT protocol, while Section V presents our attack approach. Afterwards, we discuss our findings in Section VI, and suggest some security recommendations and mitigation solutions in Section VII, while Section VIII concludes the paper.

II. Related Work

Several studies have examined the vulnerabilities in MQTT-based IIoT systems, particularly regarding rogue clients. In [6], the authors demonstrated a man-in-the-middle (MitM) attack where an adversary redirects IoT devices to a rogue MQTT broker. While this attack is similar to ours in targeting communication between devices and the broker, their method fails when MQTT runs over TLS, whereas our approach remains effective regardless of whether TLS is used. In [7], the researchers presented an attack where a rogue IoT device acts as an authorized device by bypassing MQTT broker authentication. The rogue device subscribes to the wildcard topic ('#') to intercept all communications. Although this aligns with our use of rogue devices to exploit broker authentication, their approach primarily focuses on data interception, while our attack goes further, allowing us to disconnect legitimate clients and execute more sophisticated attacks, such as reconfiguring factory controllers. The authors of [8] explored the use of rogue devices to carry out a distributed denial-of-service (DDoS) attack. By impersonating a legitimate device, the rogue device can flood the network with traffic, overloading the system. This is similar to our work in that we also use a rogue device to disrupt the network, but our approach combines a DoS attack with a replay attack, maintaining control over the compromised system after legitimate clients are disconnected. Similarly, [9] demonstrated how a rogue device can intercept and manipulate communications in an IoT network. The rogue device poses as a trusted component, allowing it to alter or inject malicious data into the network. While both approaches involve communication manipulation, our attack focuses on exploiting the MQTT protocol itself to execute a replay attack and compromise the system configuration. Finally, Rostami et al. [10] showed how a rogue device can exploit memory vulnerabilities in IoT systems to trigger memory corruption and gain control. Although their approach also involves taking control of devices, their focus is on memory-level exploits, while our attack exploits weak authentication and MQTT protocol flaws to take over system functions without requiring memory-level attacks.

In contrast to these works, our approach integrates several phases, including authentication bypass, message interception, DoS, and replay attacks, to compromise IIoT systems more comprehensively. This combination of attacks allows us to not only intercept communications but also control factory processes and disrupt operations, offering a more versatile threat model against MQTT-based IIoT systems.

III. Experimental Setup

To verify the attack method described in this paper, we utilized a Fischertechnik training factory, also known as Lernfabrik 4.0 9V. This system is designed for developing Industry 4.0 applications, as shown in Figure 1. It allows for simulating, understanding, and applying industrial automation processes on a smaller scale before they are implemented in larger settings. The factory is composed of four key industrial workstations [15]:

- Vacuum Gripper Robot (VGR): A robot featuring a vacuum gripper for handling objects.
- High-Bay Warehouse (HBW): An automated rack system with 3x3 storage slots.
- Multi-Processing Station (MPO): Represents an industrial oven and a machining workbench.
- Sorting Line and Color Detection (SLD): Includes a conveyor belt, color detection unit, and part sorting mechanism.

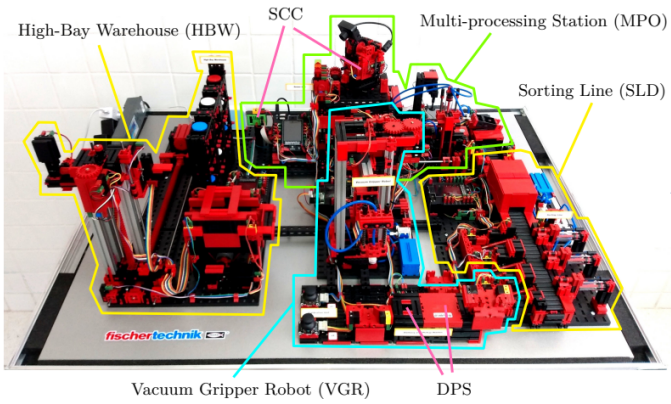

Fig. 1: Fischertechnik Lernfabrik 4.0 9V, adapted from [15].

In addition to these workstations, the factory includes various additional modules, such as a Delivery and Pickup System (DPS), a Near-Field Communication (NFC) reader/writer, an RGB camera, and a Sensor Station with a camera (SSC) for collecting further data. The production environment simulates moving small workpieces of various colors (white, red, and blue), with each workpiece equipped with a unique NFC tag. This tag stores data such as the color and production history, including timestamps. The factory setup consists of six Fischertechnik TXT 4.0 controllers, as depicted in fig. 2.

The TXT controllers communicate bidirectionally with the central controller (TXT-0), which acts as an MQTT broker, connecting the controllers via a Wi-Fi network provided by a TP-Link Router. TXT-0, configured as a cloud client, links to the Fischertechnik cloud using a remote MQTT bridge.

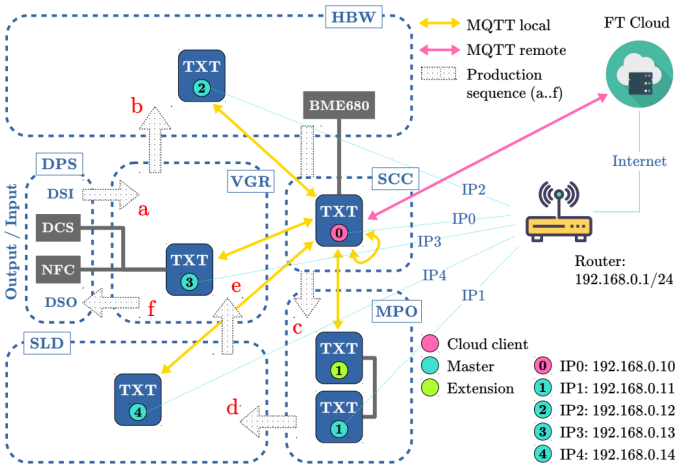

Fig. 2: Block Diagram of the Lernfabrik 4.0, adapted from [15].

Users can monitor and interact with the factory's production processes through a web-based cloud dashboard. As the cloud master, TXT-0 is equipped with a USB camera and environmental sensors to monitor air pressure, humidity, temperature, and gas levels. TXT-1 to TXT-4 controllers also serve as cloud masters, with TXT-1 having an extension. TXT-2 controls the warehouse, TXT-3 manages the DPS, and TXT-4 operates the SLD station. The block diagram illustrates the typical manufacturing flow from DSI to DSO, with each TXT controller running compiled C++ code for its designated tasks.

IV. MQTT Protocol

MQTT is a lightweight messaging protocol specifically designed for scenarios with low bandwidth, high latency, or unreliable network conditions [14]. Figure 3 illustrates the structure of an MQTT packet, which comprises a fixed header, and in some cases, a variable header and payload, depending on the packet type.

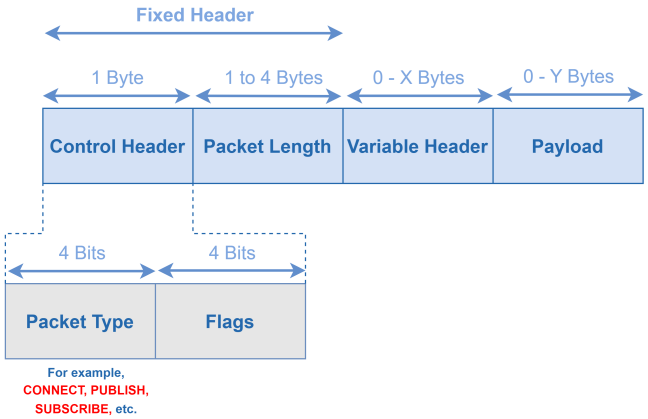

Fig. 3: MQTT Packet Structure, adopted from [3].

The packet format is determined by its type, which can vary based on the MQTT operation being performed. The following sections describe the components of an MQTT packet:

*1) **Fixed Header**:* Present in all MQTT packets, the fixed header includes key information:

- **Packet Type** (4 bits): Specifies the type of MQTT packet, such as *CONNECT*, *PUBLISH*, or *SUBSCRIBE*.
- **Flags** (4 bits): Contains various control flags, which vary depending on the packet type (e.g., Quality of Service (QoS) level, Duplicate (DUP) flag, Retain flag).
- **Packet Length** (variable length): Represents the length of the variable header and payload, using a variable-length encoding scheme where each byte reserves 7 bits for the length.

*2) **Variable Header**:* The variable header is optional and its structure is dependent on the packet type. It contains additional details specific to the operation being performed.

*3) **Payload**:* The payload carries the actual data being transmitted. Its structure varies depending on the type of packet:

- For a *CONNECT* packet, the payload includes the client identifier, will topic, will message, username, and password.
- For a *PUBLISH* packet, the payload contains the message topic and the message itself.
- For a *SUBSCRIBE* packet, the payload lists topic filters and the requested QoS levels.

The MQTT protocol follows a publish-subscribe model, which provides an efficient communication framework. The core components of the MQTT architecture include:

- **Publisher:** Sends messages to a specific topic managed by the broker.
- **Subscriber:** Subscribes to one or more topics and receives messages from the broker.
- **Broker:** Facilitates the communication between publishers and subscribers.
- **Topics:** Act as communication channels to which messages are published and from which subscribers retrieve messages.

IoT devices communicate with the MQTT broker over the Transmission Control Protocol/Internet Protocol (TCP/IP). These devices can function as publishers, subscribers, or both, within the MQTT framework. The interaction is managed through MQTT topic names, which are designated by the publisher and subscribed to by the subscriber. Adhering to the correct convention when defining MQTT topic names is essential for seamless communication between IoT devices (e.g., Programmable Logic Controllers, PLCs) and cloud servers.

Notably, publishers and subscribers are not required to know the IP addresses of other devices to communicate effectively, as the broker manages message routing. For example, a PLC may publish a message with the topic "sensor/temperature," where the payload contains the current temperature reading. This well-structured approach ensures efficient, targeted communication between IoT devices and cloud infrastructure, as detailed in [13]. Each MQTT message sent to the broker carries a distinct topic and payload, encapsulating the core data being transmitted.

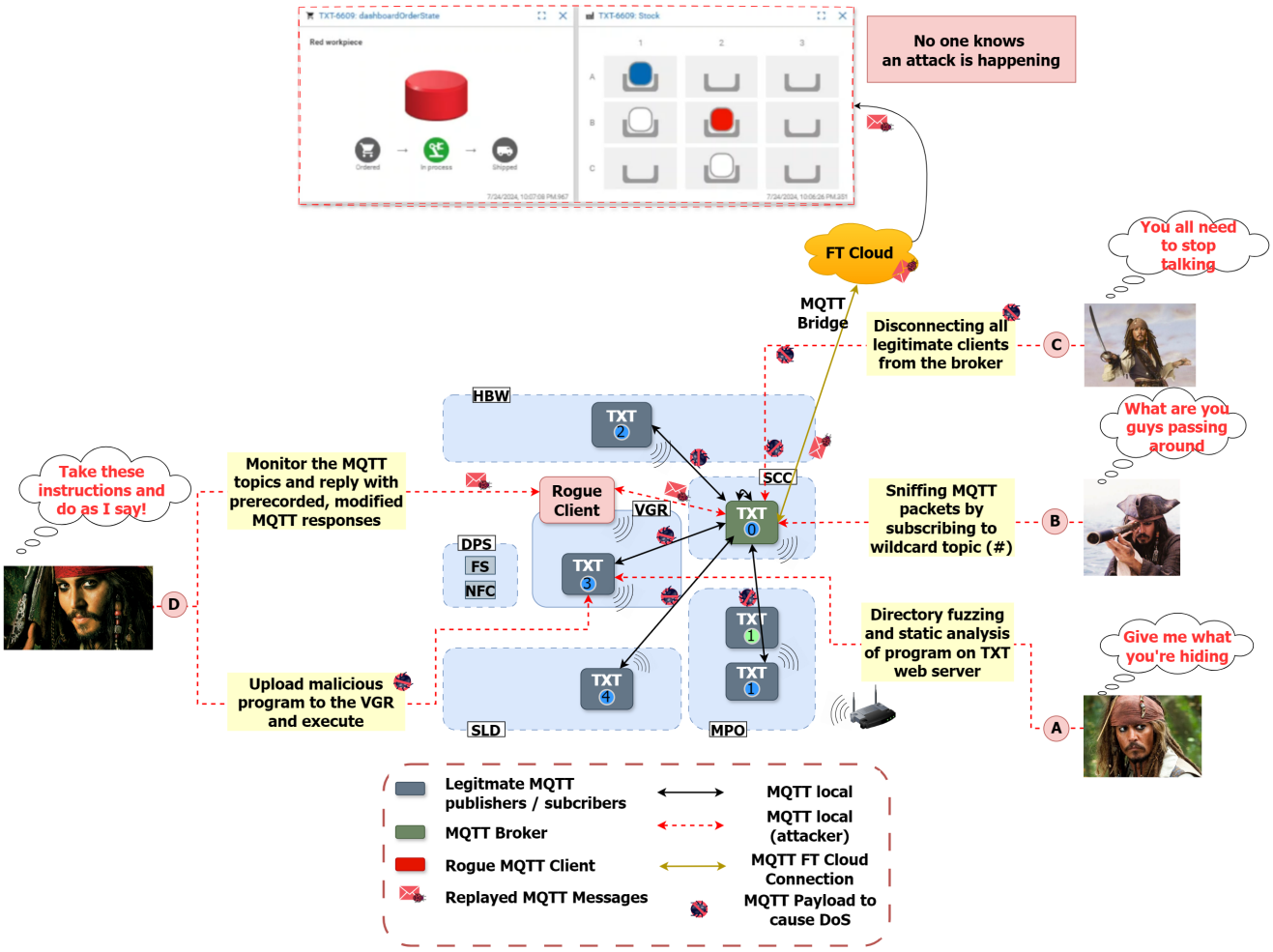

Fig. 4: Overview of MQTT Rogue Client Attack Approach

V. Attack Approach

Our attack approach is illustrated in Figure 4, which outlines the steps taken to conduct rogue MQTT client attack on our Fischertechnik system, specifically targeting the TXT-3 controller (part of the VGR module). While this work primarily focuses on the TXT-3, the approach is applicable to other controllers, depending on the specific processes each automates. The attack is conducted in five phases:

1) Breaking the authentication of the TP-Link router.
2) Compromising the security of the MQTT Broker.
3) Offline Pre-attack Phase.
4) Disconnecting legitimate clients via a DoS attack.
5) Executing the Online Attack.

In this work, we omit the detailed depiction of the first phase, which has been extensively covered in our previous paper [2]. Our emphasis in this paper is solely on phases 2, 3, 4, and 5 as elucidated in the following subsections.

A. Compromising the Security of the MQTT Broker

The MQTT broker plays a central role in managing communication between clients in an MQTT-based IIoT system. Compromising the broker's security enables unauthorized access to all communications between connected devices. Our approach introduces a novel method for bypassing the broker's authentication, differing from previous techniques discussed in [2]–[4]. This phase consists of two main steps:

1) Directory Fuzzing on an MQTT Client's Webserver: Directory fuzzing involves probing the MQTT client's webserver (in our setup, the TXT-3) by sending a series of HTTP requests to identify existing URL paths. By using predefined wordlists, such as the *SecLists wordlist*[2], and tools like *ffuf*[3] or *gobuster*[4], we automated the process of detecting accessible directories. HTTP status codes (e.g., 200 for success, 404 for non-existent paths) were used to determine valid paths. This fuzzing technique revealed the critical directory */txt*, which contained control programs and configuration files, such as the *TxtParkPosVGR* C++ program, essential for managing the TXT-3 controller's logic (see Figure 5).

2) Static Analysis of Control Software: Following the discovery of the *TxtParkPosVGR* C++ program, we performed

[2]https://github.com/danielmiessler/SecLists
[3]https://github.com/ffuf/ffuf
[4]https://github.com/OJ/gobuster

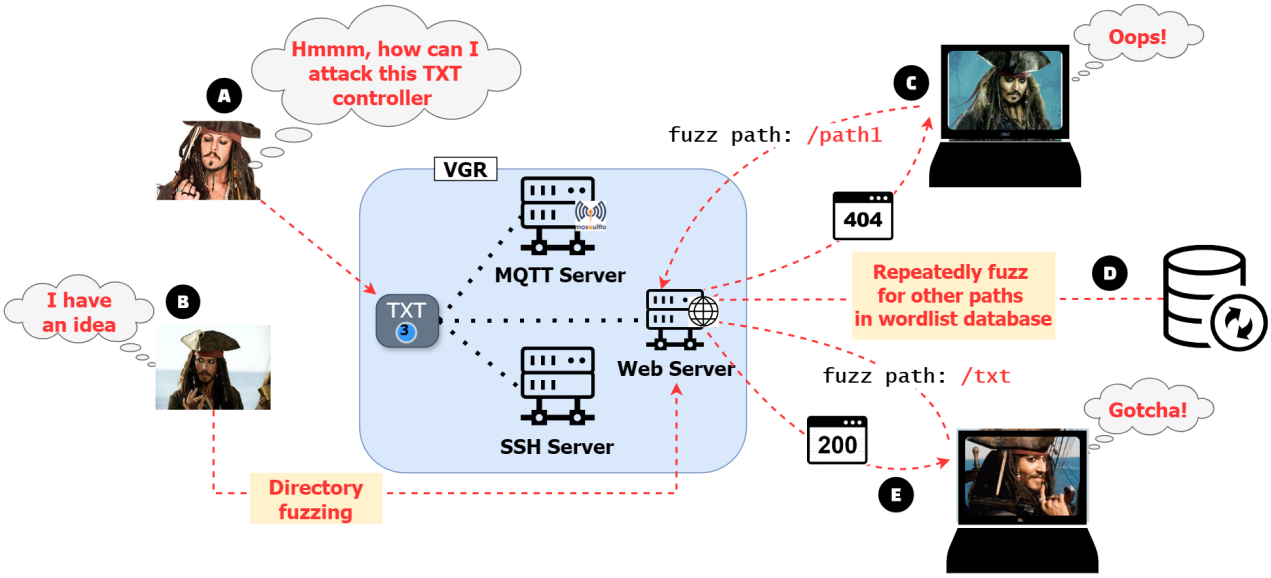

Fig. 5: Overview of Directory Fuzzing Attack

static analysis to examine the program without executing it. Using the Ghidra decompiler[5], we reverse-engineered the program into a more human-readable form, allowing us to identify key functions and critical information, such as the IP address, port, username, and password used to establish MQTT connections (see Figure 6).

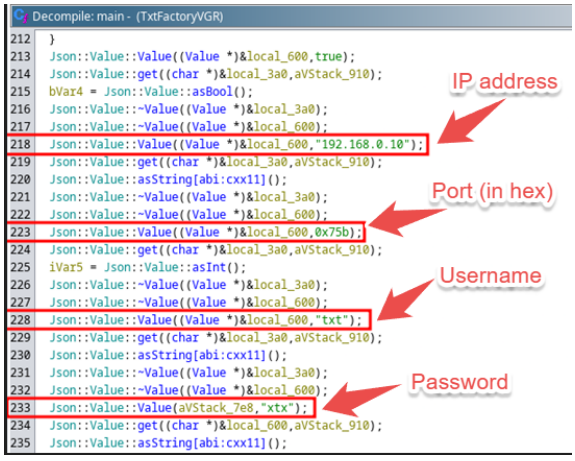

Fig. 6: Static Analysis of TxtFactoryVGR Program

This analysis provided the details necessary to further compromise the MQTT broker and prepare for the next phase of the attack.

B. Offline Pre-Attack Phase

With access to the MQTT broker established, the next phase involves an offline attack. Figure 7 provides an overview of this phase. Here, we exploit the MQTT protocol's wildcard topic vulnerability, which allows clients to subscribe to the "#" topic, effectively giving them access to all messages on the broker.

During this phase, we perform packet sniffing to record MQTT packets, focusing on MQTT *PUBLISH* messages. These packets are processed, and we filter out Quality of Service (QoS) 0 messages, which are primarily used for streaming video payloads that are irrelevant to our attack. Instead, we store QoS 1 messages, which are crucial for controlling the factory processes, in a database, mapping each message to a timestamp for our subsequent replay attack.

C. Denial of Service (DoS) Attack

Following the collection of critical MQTT messages, the next step is to disconnect legitimate clients from the broker by executing a DoS attack. The attack is directed at the shared topic "*f/o/state/ack*", which is responsible for error acknowledgment messages between TXT controllers.

By publishing a malicious payload to this topic, we cause the legitimate controllers to disconnect, preventing them from interfering with the attack. To ensure the disconnection is persistent, we exploit MQTT's "*Retain*" flag, which stores the malicious payload on the broker. Any attempt by a client to reconnect results in the immediate re-transmission of the retained payload, keeping legitimate devices disconnected throughout the attack.

D. Online Attack Phase

With legitimate clients disconnected, we introduce a rogue MQTT client to carry out the online phase of the attack, see figure 8. This phase consists of two main steps:

[5]https://ghidra-sre.org/

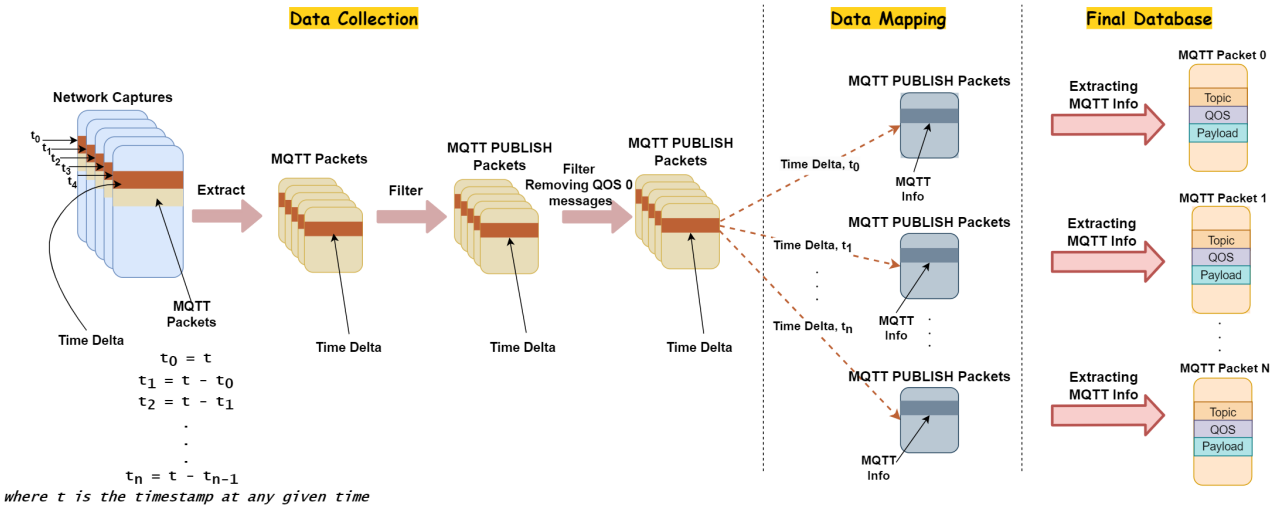

Fig. 7: Offline Attack Overview: Packet Recording and Database Creation

1) Replay of Pre-Recorded MQTT Messages:

To avoid detection by system administrators or factory operators, we perform a replay attack using the MQTT messages recorded during the offline phase. These messages are replayed in real time to simulate normal factory operations. By ensuring that the timestamps and other relevant integrity checks are adjusted to match real-time conditions, the attack remains hidden from the Fischertechnik Cloud Dashboard.

2) File Upload Attack via SSH/SCP:

While the factory operations are being mimicked through the replay attack, we also exploit weak SSH credentials (default: *RoboPro:RoboPro*) to gain direct access to the TXT-3 controller. Using SCP protocol, we upload a malicious configuration file (*Config.ParkPos.json*) that controls the VGR arm's movements within the factory.

Once the malicious file is in place, we restart the controller via SSH, causing the factory system to adopt the new, altered configuration. As a result, the VGR arm behaves erratically, disrupting factory operations, leading to potential damage or complete system failure, see our proof-of-concept [17].

VI. Findings & Discussion

During our research, we discovered several vulnerabilities in the web interface of the IIoT device that we tested. Upon navigating to the root URL path ("/"), a login page was presented, which was not easily guessable. This indicated an effort by the web server to restrict unauthorized access. However, while the login requirement prevented unauthorized writes to the system, it did not completely secure the device. Through directory fuzzing, we uncovered hidden URL paths that were accessible with only read permissions. Even with just read access, these hidden paths exposed sensitive information and created potential attack vectors that could be exploited to launch further attacks.

Our attack methodology proved effective regardless of whether plain MQTT (on port 1883) or secure MQTT (on port 8883) was being used. In the case of secure MQTT which encrypts traffic using TLS, an attacker can bypass the encryption by waiting for the MQTT handler or library at the application layer to decode the traffic. Once the traffic is processed, the attacker can extract critical MQTT information, such as topics, QoS levels, and payloads. The delay in extracting this information, compared to monitoring direct network traffic, is only a few milliseconds, making this approach highly feasible. In our test scenario, we monitored unencrypted traffic for simplicity, but the attack remains equally effective in encrypted environments.

Moreover, our attack specifically targets the MQTT protocol at the application layer, bypassing the need to exploit lower layers of the TCP/IP stack. Unlike typical Man-in-the-Middle (MiTM) attacks that rely on Address Resolution Protocol (ARP) spoofing or Internet Control Message Protocol (ICMP) redirects to intercept traffic, our approach leverages a fundamental feature of MQTT itself. By subscribing to the wildcard topic "#", an attacker can effortlessly sniff all MQTT traffic without needing to perform complex network-layer attacks. This is feasible because the MQTT configuration did not provide the detailed authorization for subscribing to topics that may leak sensitive information.

These findings demonstrate the inherent vulnerabilities within MQTT-based IIoT systems, where even limited access or knowledge of the system can lead to significant exploitation opportunities.

VII. Mitigation Solutions

To effectively mitigate our rogue client attack, a combination of technical measures and best practices is required. Below, we recommend key mitigation strategies that, when applied, would prevent the presented attack from being successful.

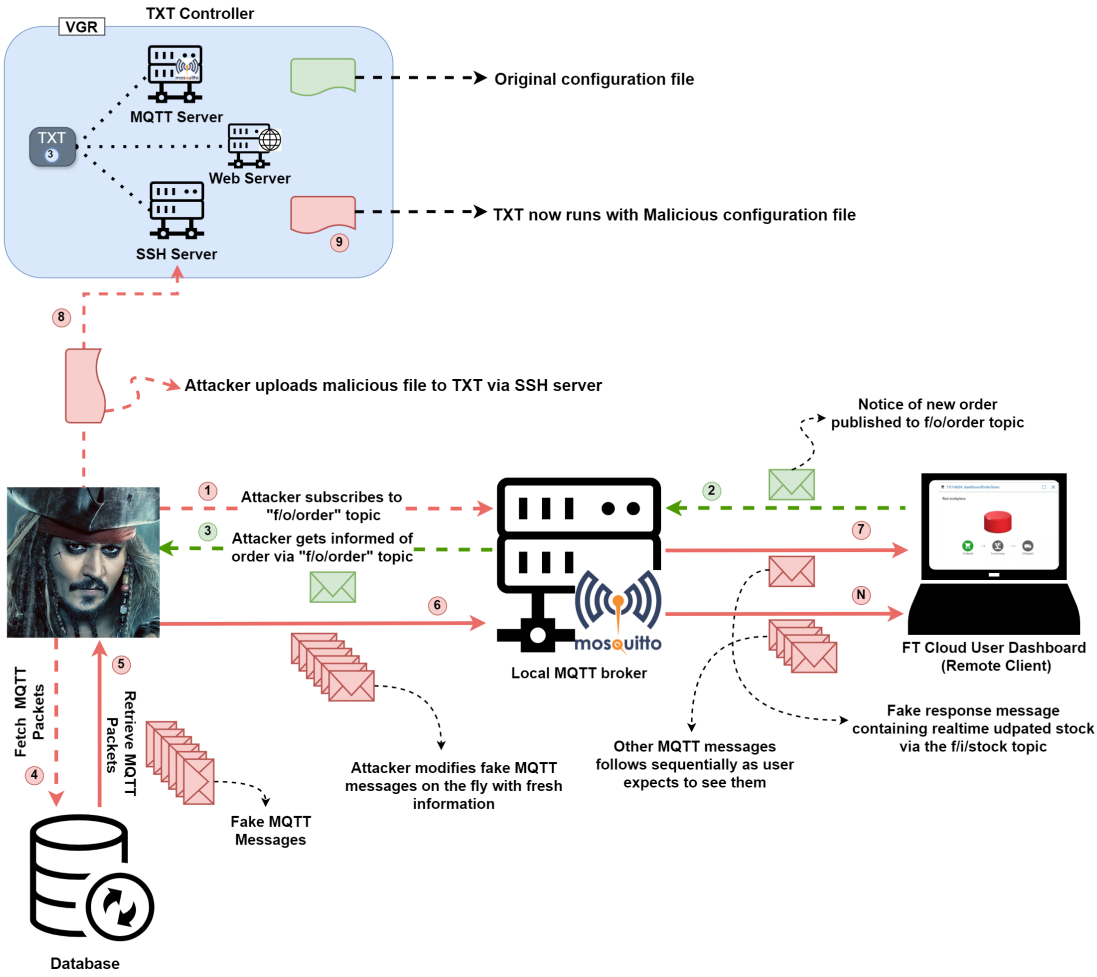

Fig. 8: Online Attack Phase

A. *Fine-Grained Authorization (FGA)*

Our findings indicate that many MQTT-based IIoT systems, including the Fischertechnik Lernfabrik 4.0, suffer from a lack of fine-grained authorization controls for topics. This deficiency is not unique to the Fischertechnik system but is a common issue in many MQTT implementations. The MQTT protocol, by default, does not include robust mechanisms for restricting which publishers can publish to specific topics.

Without FGA, any client with valid MQTT authentication credentials—or even an anonymous connection—can publish messages to any topic within the broker's domain. This lack of control allows potential attackers to exploit the system by injecting unauthorized messages or disrupting critical communication channels.

In IIoT environments, this vulnerability can lead to severe operational disruptions. For example, an attacker could interfere with production processes, manipulate sensor data, or send erroneous commands to control systems. To address this issue, MQTT implementations should incorporate fine-grained authorization mechanisms. These would allow for detailed access controls, where permissions to publish or subscribe to specific topics are granted based on the client's role and credentials. Implementing hierarchical topic restrictions and role-based access controls are essential for mitigating these risks and securing IIoT systems.

B. *Replay Attack Protection*

Replay attack protection is another critical area where many MQTT-based IIoT systems show vulnerabilities. Our experiments revealed that while some systems implement time stamping to ensure message freshness, such as the Fischertechnik system, this approach alone is insufficient for robust protection against replay attacks.

In general, time stamping helps prevent the reprocessing of old messages by discarding messages with outdated timestamps. However, if the timestamp validation process is not secure, attackers can easily manipulate or forge timestamps, circumventing these defenses. This was observed in our case study and reflects a broader issue in MQTT-based systems.

To provide effective protection against replay attacks, IIoT systems should incorporate more robust security measures. Implementing Message Authentication Code (MAC) or similar cryptographic techniques would significantly enhance the integrity and authenticity of messages. MAC ensures that

messages have not been tampered with and verifies their source, making it much harder for attackers to successfully execute replay attacks.

C. Client Authentication Using Mutual TLS (mTLS)

One of the most effective ways to prevent rogue MQTT clients from connecting to a broker is by employing mutual Transport Layer Security (mTLS). In traditional TLS, only the client verifies the broker's identity, but in mTLS, both the client and the broker authenticate each other using certificates. This provides an additional layer of security, ensuring that only authenticated devices can connect and interact with the MQTT broker.

By implementing mTLS, IIoT systems can ensure that rogue clients lacking valid certificates are unable to establish a connection with the broker. Furthermore, the use of strong encryption within TLS ensures that communications between legitimate clients and the broker remain secure, mitigating risks such as data interception or tampering during transmission. Additionally, rotating certificates periodically can further reduce the risk of key compromise and ensure security.

D. Client Activity Monitoring and Anomaly Detection

Another important mitigation strategy is continuous monitoring of MQTT client activities and implementing anomaly detection mechanisms. By setting up real-time logging and tracking of client behavior, brokers can detect abnormal activities such as rapid message publication, unauthorized topic subscriptions, or unusual connection attempts that could indicate rogue client activity. Anomaly detection can be powered by machine learning models or rule-based systems to identify patterns that deviate from expected behavior.

If an anomalous behavior is detected, such as a client attempting to publish to unauthorized topics or sending a high volume of messages in a short time, the broker can trigger automated actions like client disconnection, throttling, or alerting administrators. This proactive monitoring helps identify rogue clients in real-time, providing an additional line of defense against attacks.

VIII. Conclusion

This paper explores MQTT rogue client attacks in IIoT systems, using the Fischertechnik Lernfabrik 4.0 as a case study. We demonstrated how vulnerabilities in the MQTT protocol, such as weak authentication and authorization, can be exploited to disrupt industrial operations, posing significant security risks. In response, we proposed mitigation strategies, including fine-grained authorization, mutual TLS for authentication, replay attack protection, and real-time monitoring. These measures can strengthen MQTT-based IIoT systems against rogue client attacks.

Our research highlights the importance of ongoing efforts to improve IIoT security and encourages collaboration within the research community to protect industrial systems from evolving threats.

References

[1] W. Alsabbagh and P. Langendörfer, "Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow," IEEE Open Journal of the Industrial Electronics Society, vol. 4, pp. 659-693, 2023, doi: 10.1109/OJIES.2023.3335976.

[2] W. Alsabbagh, C. Kim and P. Langendörfer, "Silent Sabotage: A Stealthy Control Logic Injection in IIoT Systems," 2024 Silicon Valley Cybersecurity Conference (SVCC), Seoul, Korea, Republic of, 2024, pp. 1-8, doi: 10.1109/SVCC61185.2024.10637363.

[3] W. Alsabbagh, C. Kim and P. Langendörfer, "A Payload of Lies: False Data Injection Attacks on MQTT-based IIoT Systems," Accepted at the 2024 Annual Conference of the IEEE Industrial Electronics Society (IECON), Chicago, USA, 2024, doi: 10.13140/RG.2.2.14002.58565.

[4] W. Alsabbagh, C. Kim, N. S. Patil and P. Langendörfer, "Beyond the Lens: False Data Injection Attacks on IIoT-Cameras through MQTT Manipulation," Accepted at the 2024 7th Conference on Cloud and Internet of Things, Montreal, Canada, 2024, doi: 10.13140/RG.2.2.25837.81129.

[5] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S—A publish/subscribe protocol for wireless sensor networks," in Proc. IEEE 3rd Int. Conf. Commun. Syst. Softw. Middleware Workshops (COMSWARE), 2008, pp. 791–798.

[6] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul and A. Panya, "Authorization mechanism for MQTT-based Internet of Things," 2016 IEEE International Conference on Communications Workshops (ICC), Kuala Lumpur, Malaysia, 2016, pp. 290-295, doi: 10.1109/ICCW.2016.7503802.

[7] V. Tewolde, 'Comparison of authentication options forMQTT communication in an IoT basedsmart grid solution', Dissertation, 2019.

[8] F. Chen, Y. Huo, J. Zhu, and D. Fan, "A review on the study on mqtt security challenge," in 2020 IEEE International Conference on Smart Cloud (SmartCloud). IEEE, 2020, pp. 128–133.

[9] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Internet of Things (IoT): Taxonomy of security attacks," in 2016 3rd International Conference on Electronic Design (ICED), Aug. 2016, pp. 321–326. doi: 10.1109/ICED.2016.7804660.

[10] A. Rostami, M. Vigren, S. Raza, and B. Brown, "Being hacked: Understanding victims' experiences of IoT hacking," in Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022), 2022, pp. 613–631.

[11] A. C. Panchal, V. M. Khadse and P. N. Mahalle, "Security Issues in IIoT: A Comprehensive Survey of Attacks on IIoT and Its Countermeasures," 2018 IEEE Global Conference on Wireless Computing and Networking (GCWCN), Lonavala, India, 2018, pp. 124-130, doi: 10.1109/GCWCN.2018.8668630.

[12] D. Serpanos, "Industrial Internet of Things: Trends and Challenges," in Computer, vol. 57, no. 1, pp. 124-128, Jan. 2024, doi: 10.1109/MC.2023.3331552.

[13] H. Xu, W. Yu, X. Liu, D. Griffith and N. Golmie, "On Data Integrity Attacks against Industrial Internet of Things," 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 2020, pp. 21-28, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00020.

[14] A. H. Eyeleko and T. Feng, "A Critical Overview of Industrial Internet of Things Security and Privacy Issues Using a Layer-Based Hacking Scenario," IEEE Internet of Things Journal, vol. 10, no. 24, pp. 21917-21941, 15 Dec.15, 2023, doi: 10.1109/JIOT.2023.3308195.

[15] A. L. de Sousa and A. S. de Oliveira, "Order-Controlled Production Employing Multi-Agent and Flexible Job-Shop Scheduling on a Physical Simulation Platform," 2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE), São Bernardo do Campo, Brazil, 2022, pp. 229-234, doi: 10.1109/LARS/SBR/WRE56824.2022.9995868.

[16] [Online]. Available: https://github.com/the-pythonist/pirates_of_the_mqtt.

[17] [Online]. Available: https://youtu.be/alKavJ9x6VY.