International Journal of Safety and Security Engineering
Vol. 16, No. 10, October, 2026, pp. 2351-2361
Journal homepage: http://iieta.org/journals/ijsse

An Integrated Snort and Fail2Ban Approach for Detecting and Mitigating Brute-Force Attacks on MQTT Protocol

Fahrurrozi Lubis[1*], Romi Fadillah Rahmat[1], Poltak Sihombing[2], Suherman[3], Pauzi Ibrahim Nainggolan[2], Seniman[1], Fleming Rabbidores Saragih[1]

[1] Department of Information Technology, Universitas Sumatera Utara, Medan 20155, Indonesia
[2] Department of Computer Science, Universitas Sumatera Utara, Medan 20155, Indonesia
[3] Electrical Engineering Department, Universitas Sumatera Utara, Medan 20155, Indonesia

Corresponding Author Email: fahrurrozi.lubis@usu.ac.id

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/ijsse.161013

Received: 16 August 2026
Revised: 6 October 2026
Accepted: 10 October 2026
Available online: 31 October 2026

Keywords:
Message Queuing Telemetry Transport, Mosquitto, authentication brute-force attack, topic brute-force attack, Quality of Service abuse, Snort, Fail2Ban, firewall

ABSTRACT

Message Queuing Telemetry Transport (MQTT) is widely used within Internet of Things (IoT) environments because of its lightweight publish–subscribe communication model. However, inadequate authentication and security configurations may expose MQTT brokers to authentication brute-force attacks, topic brute-force attacks, and Quality of Service (QoS) abuse, potentially resulting in unauthorized access, malicious message publication, resource exhaustion, and service disruption. This study developed an automated detection and mitigation mechanism for a Mosquitto MQTT broker by integrating Snort as an Intrusion Detection System (IDS) with Fail2Ban as a log-based response mechanism. Snort analyzes network traffic using predefined rules and records alerts when the configured packet characteristics and traffic thresholds are satisfied. Fail2Ban subsequently processes the Snort alerts and temporarily blocks the corresponding source IP addresses by updating the iptables firewall rules. The mechanism was evaluated in controlled scenarios involving authentication brute-force attacks, topic brute-force attacks, QoS abuse, and TCP/TLS connection-rate anomalies. Each scenario was executed for 120 seconds and repeated ten times. Under the evaluated configurations, Snort generated alerts for authentication brute-force activities on port 1883 in approximately 6.0–6.5 seconds, TCP/TLS connection-rate anomalies on port 8883 in approximately 6.3–6.5 seconds, topic brute-force activities in approximately 5.6 seconds, and QoS abuse in less than 0.5 seconds. Following alert generation, Fail2Ban blocked the associated source addresses in approximately one second. The blocking action limited further traffic from those addresses and reduced sustained CPU utilization on the broker host. These findings demonstrate the operation of the proposed mechanism under the tested traffic patterns and threshold configurations; however, they should not be generalized to unseen, slow, distributed sub-threshold, or rule-evasion attacks. For encrypted MQTT traffic on port 8883, the evaluated rule detected repeated TCP connection initiations rather than MQTT authentication failures.

1. INTRODUCTION

The Internet of Things is a concept in which physical devices such as sensors, cameras, vehicles, and other objects are connected to each other via the internet to collect and exchange data [1]. IoT has seen a lot of use across various sectors such as healthcare, industry, transportation, and smart homes. To ensure communication between devices in the IoT ecosystem, various communication protocols are used, such as HTTP, CoAP, AMQP, and MQTT [2]. Among these protocols, Message Queuing Telemetry Transport (MQTT) is one of the most popular. This protocol is optimized for network environments with low bandwidth and high latency [3]. The MQTT protocol works using a publisher-subscriber model with a broker as the data traffic manager. Publishers send messages to specific topics, while subscribers receive messages from the topics they have subscribed to. One of the popular MQTT brokers, Mosquitto, is responsible for distributing messages between publishers and subscribers [4].

Although MQTT excels in terms of communication efficiency, this protocol has limitations in terms of security. One of its vulnerabilities is the lack of built-in security mechanisms such as strong authentication and adequate encryption. This makes it vulnerable to attacks that can intercept and modify communication between IoT devices, which could result in the compromise of device integrity or theft of sensitive data [5-7].

Mosquitto broker can be deployed on both local servers and cloud platforms and is frequently utilized within Internet of Things (IoT) ecosystems, such as smart homes or industrial IoT [8]. However, due to a lack of robust authentication mechanisms and proper security configurations, the broker can

become a vulnerability susceptible to exploitation. Many MQTT brokers do not enable authentication by default; consequently, any individual who possesses the server address can connect as a client and subscribe to all available topics, potentially leading to the leakage of sensitive internal data.

This condition makes MQTT brokers vulnerable to various types of attacks, such as Brute-Force Attacks, which target broker authentication to gain illegal access [9-11]. The threat is not only limited to attempts to gain access through authentication, but also to attacks that target topics and Quality of Service (QoS) in MQTT communications [8]. These attacks are designed to take over and exhaust broker resources, ultimately crippling MQTT services. Figure 1 shows how the MQTT protocol mechanism works.

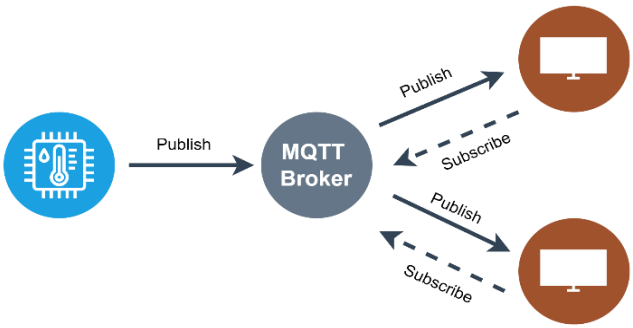


Figure 1. MQTT protocol works

Currently, Intrusion Detection System (IDS) tools such as Snort are widely used to monitor network traffic and identify attack patterns using a set of rules [12]. However, most existing implementations still focus on passive attack detection, without automatic prevention mechanisms. MQTT-specific denial-of-service mitigation has also been investigated at the IoT edge layer to preserve QoS and node energy [13].

This limitation shows the need for an integrated security system between an IDS and firewall-based prevention mechanisms to improve MQTT broker security. Integrating Snort as an IDS and Fail2Ban as an Intrusion Prevention System (IPS) creates an effective defense mechanism capable of:

•Capture and analyze network packets traversing the Mosquitto broker.

•Identify patterns of brute-force attacks targeting authentication and topics, as well as QoS abuse attacks on the MQTT protocol through Snort rule design.

•Log and generate alerts for suspicious activities.

•Establish an automatic prevention mechanism to block attackers in real-time at the firewall level.

•Provide a lightweight and practical security architecture, able to be applied in the resource constrained environment such as IoT environments.

The rest of this paper is organized as follows: Section 2 discusses related studies. Section 3 explains the research methodology, including the system architecture design and details of the Snort rules implementation and Fail2Ban configuration. Section 4 presents the test results and performance analysis of the proposed system in detecting and preventing attacks. Finally, Section 5 summarizes the main findings from the analysis and testing results.

2. RELATED WORKS

The lightweight publish-subscribe architecture of MQTT supports efficient communication among resource-constrained Internet of Things (IoT) devices. However, MQTT deployments remain susceptible to unauthorized access, brute-force authentication attempts, topic enumeration, message flooding, and denial-of-service attacks when authentication, authorization, and transport security are inadequately configured [1, 3, 7, 8]. Large-scale assessments of publicly accessible MQTT brokers have further demonstrated that many deployments retain insecure default configurations, including anonymous access and insufficient encryption, thereby exposing broker resources and transmitted data to potential exploitation [3].

Several studies have investigated intrusion detection techniques for identifying malicious MQTT traffic. Signature-based intrusion detection systems offer a lightweight approach because suspicious traffic can be identified through protocol fields, packet characteristics, and predefined thresholds. MQTT-specific protocol parsing integrated with Suricata has been shown to detect malformed packets and denial-of-service activities while maintaining broker stability and reducing CPU utilization during attacks [1]. Snort-based detection has also been applied to MQTT-enabled smart devices, where custom rules were used to identify denial-of-service traffic, although the reported precision indicates that rule design and threshold selection remain important determinants of detection performance [14]. In addition, MQTT attack detection has been investigated using machine learning. LightGBM-based detection achieved a high true-positive rate and a low false-positive rate for denial-of-service traffic, whereas Random Forest has been employed to classify brute-force attacks on MQTT communication [11, 15]. Nevertheless, machine-learning approaches generally require representative training data, feature extraction, and computational resources that may not be suitable for every constrained IoT deployment.

Attack prevention has also been studied through the integration of intrusion detection systems, firewalls, and automated IP-blocking mechanisms. Architectures combining host-based and network-based intrusion detection with firewall controls have demonstrated that malicious traffic can be intercepted before it causes prolonged disruption to protected services [16]. Fail2Ban provides a comparatively lightweight response mechanism by monitoring service or intrusion-detection logs and automatically creating firewall rules against addresses that exceed predefined thresholds [17-20]. Previous implementations have demonstrated its ability to mitigate brute-force and denial-of-service activities on SSH and web services. The combination of Snort and Fail2Ban has also been examined for SSH protection, showing that Snort can generate attack alerts while Fail2Ban applies automated blocking actions [18]. However, the effectiveness of such integration depends on the accuracy of the detection signatures, the selected thresholds, and the ability of the generated logs to reliably identify the source of malicious traffic.

As summarized in Table 1, existing approaches differ in terms of their target environments, attack categories, and response mechanisms. MQTT-oriented studies have mainly emphasized denial-of-service detection, vulnerability assessment, and machine-learning classification, whereas automated blocking using Fail2Ban has been more frequently evaluated on SSH and general server environments. The present study extends these directions by examining an

integrated rule-based detection and automated prevention mechanism for authentication attacks, topic-access attempts, and QoS abuse within an MQTT broker environment.

Existing MQTT security studies have predominantly focused on vulnerability assessment, denial-of-service detection, machine-learning classification, or secure broker configuration. Meanwhile, previous Snort-Fail2Ban integrations have primarily been evaluated on SSH or general server traffic rather than on MQTT-specific authentication, topic-access, and Quality of Service (QoS) behaviors. Consequently, limited evidence is available regarding how a lightweight rule-based detection and automated firewall response mechanism performs across multiple MQTT attack categories within the same experimental environment.

To address this gap, this study integrates Snort and Fail2Ban to detect and mitigate three categories of malicious MQTT activity: repeated authentication attempts, unauthorized topic-access attempts, and high-frequency message transmission across QoS levels. Snort analyzes network traffic using MQTT-oriented rules and generates structured alerts, while Fail2Ban processes these alerts and temporarily blocks the identified source addresses through the host firewall. The evaluation compares broker behavior under three conditions: without a security mechanism, with Snort-based detection, and with the integrated Snort-Fail2Ban mechanism. The study measures attack-detection time, blocking time, and broker resource utilization to assess the operational effectiveness and limitations of the proposed approach.

Table 1. Comparison of existing MQTT attack detection and prevention approaches

References	Environment	IDS/IPS	Attacks	Result
Husnain et al. [1]	MQTT	Suricata	DoS, DDoS	Blocked malicious packets, reduced CPU load during attacks.
Farhannullah and Hardjianto [17]	SSH	Fail2Ban	Brute-Force	Blocked 2,498 attacks within one week through automated IP banning.
da Silva et al. [15]	MQTT	Machine Learning	DoS	Achieved 97% True Positive rate and 0.16% False Positive rate with low memory consumption.
Nadeem et al. [16]	Cloud	Firewall	Brute Force, DDoS	Firewalls blocked traffic before reaching the server, HIDS was effective for pattern matching.
Palmieri et al. [3]	MQTT	MQTTSA	Brute Force, DoS, Data Tampering	Brokers like Mosquitto, were insecure due to default configurations lacking authentication.
Gunawan et al. [14]	MQTT	Snort	DoS	proposed detection rules achieved a precision value of 0.8.
Abdullah et al. [18]	SSH	Snort, Fail2Ban	Brute Force	Optimal topology (Snort in front of the server) detected 100% of attacks with rapid response times.

3. METHODOLOGY

3.1 System architecture

The openness of MQTT broker communities such as Mosquitto to communication from various IoT devices opens up loopholes to various cyber attacks, particularly Brute-Force Attacks. Brute-Force Attacks targeting MQTT authentication and topics can exploit default configurations or weak credential validation, leading to data leaks or IoT device access takeover [3]. The vulnerability of the MQTT protocol to QoS abuse attacks can also cause resource exhaustion on the server and disrupt MQTT services.

The system proposed in this study integrates Snort with Fail2Ban to detect and prevent brute-force activities and QoS level abuse with a rule-based detection approach and automatic prevention response through a firewall. Snort and Fail2Ban were chosen for this study due to several key considerations. Snort is a lightweight, stable open-source IDS with relatively low CPU and RAM consumption, making it suitable for use in limited IoT ecosystems [12, 18]. Meanwhile, Fail2Ban was chosen because it is capable of preventing attacks by monitoring log files from various services to detect suspicious access patterns [19, 20]. The combination of these two components is expected to improve communication security on the MQTT protocol, minimize the risk of unauthorized access, reduce message flooding on brokers, and strengthen the resilience of IoT systems against Brute-Force Attacks.

In this study, the proposed topology is a simple scheme, as illustrated in Figure 2. All network devices, such as attackers, clients, sensors (IoT devices), and Mosquitto brokers, are connected to a single Local Area Network (LAN) Interface as a distribution center. In this architecture, Snort and Fail2Ban are placed on the Mosquitto broker (host), which is tasked with detecting and preventing attacks on the MQTT protocol [4].

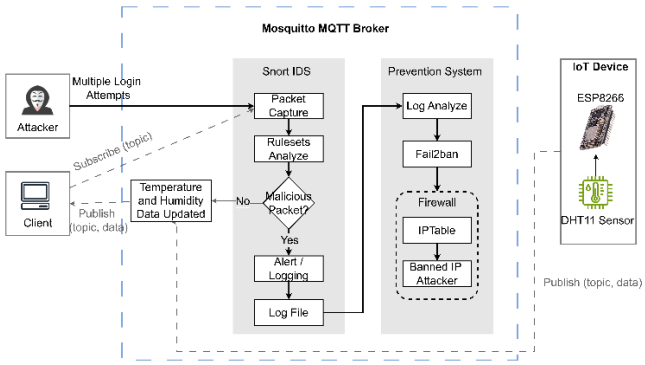


Figure 2. General architecture

3.2 Attack scenarios

3.2.1 Authentication brute-force and TCP/TLS connection-rate scenarios

The authentication brute-force scenario on port 1883 was designed to evaluate repeated MQTT login attempts against the broker [11]. The attack was performed by transmitting MQTT CONNECT requests using combinations of usernames and passwords. In the single-source scenario, one attacker generated the authentication attempts. In the multi-source scenario, ten Docker containers generated authentication attempts concurrently.

The ten attacker containers were connected to an IPvlan network and assigned distinct source IP addresses ranging from 192.168.18.71 to 192.168.18.80, while the Mosquitto broker used the address 192.168.18.200. All containers

executed the same attack procedure and were started concurrently using Docker Compose. For port 1883, each container used a 100-entry password wordlist. The Snort rule tracked matching events by source address and generated an alert after more than four matching attempts occurred within 30 seconds. The per-source detection records indicate that all ten source addresses independently generated alerts under the evaluated configuration.

The exact inter-attempt interval, per-source request rate, aggregate attack rate, and container start-time differences were not retained in the original experimental records. Therefore, the multi-source scenario on port 1883 represents concurrent attacks in which each source independently exceeded the configured Snort threshold. It does not represent a distributed sub-threshold attack in which the aggregate attack volume is high while each source remains below the detection threshold.

For encrypted MQTT communication on port 8883, the experiment used repeated TCP/TLS connection attempts involving dummy client certificate and key pairs. Because the MQTT authentication exchange was encrypted, the evaluated Snort rule could not inspect usernames, passwords, certificate-validation results, MQTT CONNECT fields, or broker authentication responses. Consequently, the port 8883 scenario was evaluated as a TCP/TLS connection-rate anomaly rather than as direct MQTT authentication brute-force detection.

In the multi-source port 8883 scenario, ten containers concurrently generated TCP/TLS connection attempts. However, because the Snort rule tracked SYN packets by destination address, connections from different sources directed to the same broker could be counted collectively. Therefore, the results cannot establish whether each source independently exceeded the configured threshold. Figure 3 illustrates the authentication brute-force scenario on port 1883 and the TCP/TLS connection-rate scenario on port 8883.

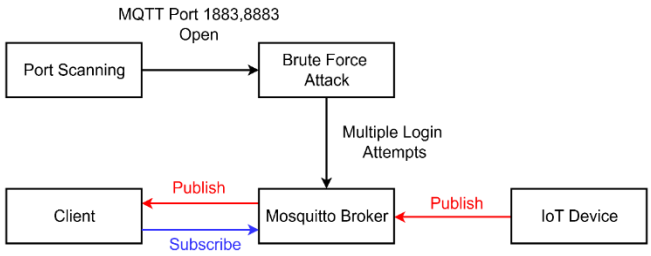


Figure 3. Authentication brute-force and TCP/TLS connection-rate test scenarios

3.2.2 Topic brute-force attacks

The topic brute-force scenario was designed to generate repeated publish or subscribe attempts involving valid and invalid MQTT topic names. As shown in Figure 4, the scenario evaluated whether the configured Snort rules could detect repeated topic-access activities that might be used to obtain sensitive data or disrupt communication between legitimate clients.

3.2.3 QoS abuse

QoS in the MQTT protocol is a parameter inherent to each message that defines the level of delivery assurance between a publisher and subscriber [21]. The MQTT protocol supports three distinct QoS levels 0, 1, and 2, enabling users to select an optimal balance between delivery reliability and network overhead.

Consequently, the utilization of high-level QoS messages within the MQTT protocol can increase network congestion and escalate the memory consumption of the broker [21]. Figure 5 illustrates the operational differences among the three MQTT QoS levels.

QoS abuse attacks are attack schemes that exploit the QoS parameters in the MQTT protocol [8]. These attacks exploit the message delivery mechanism in the MQTT protocol by flooding the broker with a large number of messages at rapid intervals [22]. as shown in Figure 6. Testing was conducted by alternating between three QoS levels (0, 1, and 2). At high QoS levels (especially QoS 2), the complex handshake mechanism is forced to work excessively, which aims to exhaust computational resources (resource exhaustion) on the broker [8].

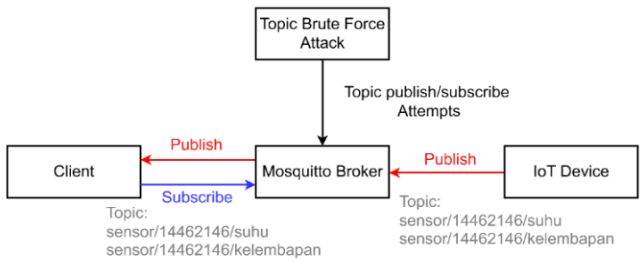


Figure 4. Topic brute-force attack scheme

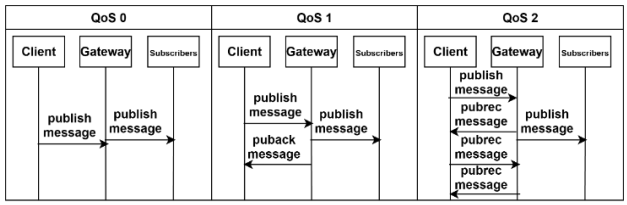


Figure 5. Operational differences among MQTT QoS levels

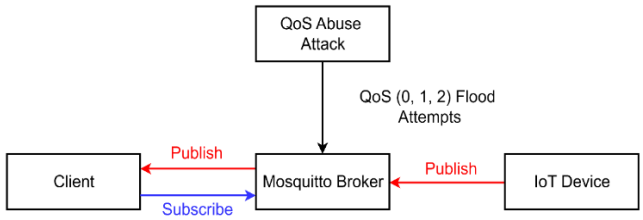


Figure 6. QoS abuse attack scheme

3.3 Snort detection mechanism

Snort is used in this study as an IDS that will be implemented in the Mosquitto broker to monitor suspicious packets from the network interface. All packets will be analyzed based on a set of rules with the main configuration including:

•Snort.conf, which serves as the main Snort configuration file to define network variables and rule paths.

•Rules, which represent a collection of detection signatures utilized to identify suspicious network traffic patterns or potential cyberattacks. Each rule comprises specific instructions designed to detect particular traffic types based on protocols, IP addresses, ports, and payload patterns.

In this section, the researcher configured several rules based on the analysis of the MQTT protocol packet header as shown

in Figure 7, to conduct tests on predetermined attack scenarios.

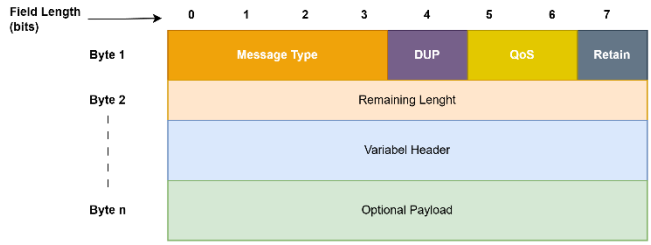


Figure 7. MQTT protocol packet structure

To detect authentication brute-force activities on the unencrypted MQTT service, a custom Snort rule was configured to monitor TCP traffic directed to port 1883. The rule identified packets containing the "MQTT" pattern and applied source-based tracking (track by_src) to count repeated matching events from each source IP address. An alert was generated when the configured threshold was exceeded within a 30-second observation window. The rule used in the experiment was:

alert tcp any any -> $HOME_NET 1883 (msg:"MQTT Brute-Force Attempt"; content:"MQTT"; detection_filter:track by_src, count 4, seconds 30; sid:1000002; rev:1;)

A separate rule was used to monitor repeated TCP connection initiations directed to the TLS-enabled MQTT service on port 8883. Because the MQTT authentication exchange was encrypted, the rule could not inspect usernames, passwords, certificate-validation results, MQTT CONNECT fields, or broker authentication responses. Instead, the rule matched TCP SYN packets using the flags:S condition and applied destination-based tracking (track by_dst). The rule used in the experiment was:

alert tcp any any -> $HOME_NET 8883 (msg:"MQTT Brute-Force Attempt"; flags:S; detection_filter:track by_dst, count 4, seconds 30; sid:1000003; rev:1;)

Although the message field of the original port 8883 rule labels the event as an "MQTT Brute-Force Attempt," its matching conditions do not directly identify MQTT authentication failures. Therefore, alerts generated by this rule are interpreted as TCP/TLS connection-rate anomaly alerts. Because the rule uses destination-based tracking, TCP SYN packets from multiple sources directed to the same broker may be counted collectively.

To detect repeated topic-access attempts, custom rules were configured to distinguish traffic from trusted clients from publish or subscribe activities that matched the predefined detection conditions. The author designed custom rules to allow legitimate traffic from trusted networks (TRUSTED_NETS) to the MQTT broker on port 1883. These rules are used to reduce false positives by ensuring that clients authorized to access certain topics are not detected as suspicious activity. "pass tcp $TRUSTED_NETS any -> $HOME_NET 1883 (msg:"MQTT Authorized Topic sensor/ -PASS"; content:"sensor/"; sid:1000202; rev:1;)". Meanwhile, another rule is enforced to detect unauthorized publish or subscribe activity to specific topics. Detection is performed by examining the MQTT packet structure using a byte test method to identify the type of message being sent. If there is repeated activity from the same source, it will be considered an indication of an attack. "alert tcp any any -> $HOME_NET 1883 (msg:"MQTT Unauthorized Topic Publish Attempt"; byte_test:1,&,0xF0,4;detection_filter:track by_src, count 4, seconds 30; sid:1000203; rev:3;)" and "alert tcp any any -> $HOME_NET 1883 (msg:"MQTT Unauthorized Topic Subscribe Attempt"; content:"|82|"; offset:0; depth:1; detection_filter:track by_src, count 4, seconds 30; sid:1000204; rev:1;)".

Then, to detect QoS abuse on the MQTT protocol, rules are designed to identify specific MQTT packet patterns and monitor message delivery frequency at each QoS level from the same source. When the number of matching packets exceeds the configured threshold, Snort generates an alert. The following are rules designed to detect attacks on three QoS levels. "alert tcp any any -> $HOME_NET 1883 (msg:"MQTT QoS0 Abuse"; content:"|30|"; offset:0; depth:1; detection_filter:track by_src, count 20, seconds 10; sid:1000101; rev:3;)", "alert tcp any any -> $HOME_NET 1883 (msg:"MQTT QoS1 Abuse"; content:"|32|"; offset:0; depth:1; detection_filter:track by_src, count 20, seconds 10; sid:1000102; rev:3;)", and "alert tcp any any -> $HOME_NET 1883 (msg:"MQTT QoS2 Abuse"; content:"|34|"; offset:0; depth:1; detection_filter:track by_src, count 20, seconds 10; sid:1000103; rev:3;)".

3.4 Fail2Ban prevention mechanism

Fail2Ban was used as an automated response mechanism to process the attack alerts generated by Snort. Fail2Ban monitored the Snort alert log, extracted the source IP addresses associated with matching alerts, and applied temporary blocking rules through iptables. Thus, Snort evaluated the packet characteristics and traffic thresholds, whereas Fail2Ban performed the subsequent blocking action based on the resulting alert records [23]. A firewall acts as a gatekeeper by inspecting every packet and determining whether it should be forwarded or blocked based on predefined rules, ensuring that only secure and legitimate traffic is allowed to access the network [24]. The prevention mechanism utilizes several main Fail2Ban configuration files, including:

In the port 1883 multi-source scenario, Fail2Ban processed the source addresses recorded in the individual Snort alerts. Therefore, blocking was applied to each of the ten source addresses that independently triggered the source-based Snort rule. For port 8883, the Snort alert was generated using destination-based tracking; consequently, the experimental results do not establish that each individual source independently exceeded the detection threshold.

•jail.conf, which defines the monitored log source, the matching filter, the blocking action, and the duration of the applied ban. In the evaluated mechanism, the packet-frequency thresholds were defined in the Snort rules, while Fail2Ban responded to the alert records produced by Snort.

•filter.d, contains filter files in the form of regular expressions to recognize log patterns that indicate attack activity. In this study, the filters were adjusted to read logs generated by Snort.

Preventive measures can be taken by applying blocking rules on the system firewall using iptables [24, 25]. This blocking is automatic and is intended to stop network traffic from the attacker's IP address to the MQTT broker, specifically on the MQTT communication port.

3.5 Statistical analysis

The detection and blocking times were summarized using

descriptive statistics calculated from ten experimental repetitions for each scenario. The reported statistics include the arithmetic mean, sample standard deviation, two-sided 95% confidence interval for the mean, and minimum–maximum values. The 95% confidence interval was calculated using the *t*-distribution with nine degrees of freedom. No inferential comparison among the evaluated scenarios was performed. Accordingly, numerical differences among the reported means were interpreted descriptively and were not considered evidence of statistically significant differences.

4. RESULT AND DISCUSSION

4.1 System environment

The experiments were conducted in a controlled virtualized environment. The host computer used an AMD Ryzen 5 5625U processor operating at 2.3 GHz and 8 GB of RAM. The Mosquitto broker was deployed in an Oracle VM VirtualBox virtual machine running Ubuntu 22.04.5, with 1 vCPU, 2,048 MB of allocated memory, and 30.29 GB of virtual storage. The experimental software consisted of Mosquitto 2.0.11, Snort 2.9.20, Fail2Ban 0.11.2, and Docker 26.1.5. The available network bandwidth was not recorded during the original experiment.

Table 2. Experimental system configuration and workload

Category	Parameter	Configuration
Host hardware	Processor	AMD Ryzen 5 5625U, 2.3 GHz
Host hardware	Installed memory	8 GB RAM
Virtualization	Platform	Oracle VM VirtualBox
Broker VM	Allocated processor	1 vCPU
Broker VM	Allocated memory	2,048 MB
Broker VM	Allocated storage	30.29 GB
Broker VM	Operating system	Ubuntu 22.04.5
Network	Available bandwidth	Not recorded
Software	Mosquitto	Version 2.0.11
Software	Snort	Version 2.9.20
Software	Fail2Ban	Version 0.11.2
Software	Docker	Version 26.1.5
Normal workload	MQTT clients	2 clients
Normal workload	Aggregate message rate	2 messages every 5 s (0.4 messages/s)
Normal workload	Message payload size	Not recorded
Attack workload	Attacker containers	1–10 containers
Attack workload	Brute-force rate per source	Not recorded
Attack workload	QoS publication rate per source	Not recorded
Attack workload	Inter-message interval	Not recorded
Experimental protocol	Duration	120 s per test
Experimental protocol	Repetitions	10 repetitions per scenario

The normal workload consisted of two MQTT clients generating an aggregate of two messages every five seconds, equivalent to 0.4 messages per second. Attack scenarios used between one and ten attacker containers. Each scenario was executed for 120 seconds and repeated ten times. The original experimental records did not retain the MQTT payload size, the brute-force attempt rate per source address, the QoS publication rate per source address, or the precise inter-message interval. Accordingly, these parameters are reported as not recorded rather than reconstructed retrospectively. Table 2 summarizes the available hardware, virtualization, software, workload, and testing configurations.

The reported configuration improves the reproducibility of the experimental environment; however, exact replication remains limited by parameters that were not retained in the original records, including network bandwidth, payload size, per-source attack-request frequency, and the precise inter-message interval. These factors may influence broker resource utilization and the time required for traffic to reach the configured Snort thresholds. The reported CPU usage and detection times should therefore be interpreted within the documented environment and tested workload rather than as hardware-independent performance values.

4.2 System result

4.2.1 Scenario without Snort and Fail2Ban

In this scenario, the MQTT broker was run without any attack detection or prevention mechanisms. Testing was conducted by simulating Brute-Force authentication attacks, topic Brute-Force Attacks, and QoS abuse against the broker.

While the attack testing mechanism was running, researchers also observed the Mosquitto broker's response to authentication attempts and topic exploration through log files. Figures 8–10 clearly show the Mosquitto logs that record all connection requests that return a hexadecimal code or error in the event of authentication failure or publish- subscribe attempts to random topics.

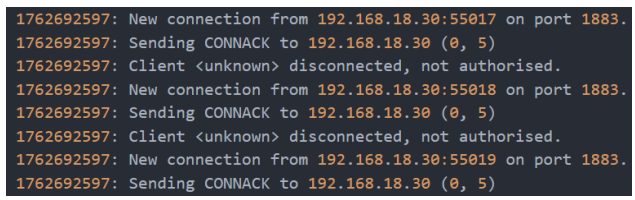


Figure 8. Mosquitto log when login failed

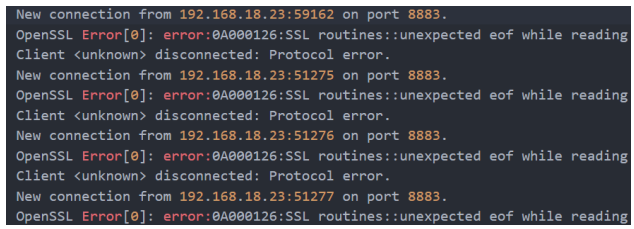


Figure 9. Mosquitto log when failed to log in with SSL

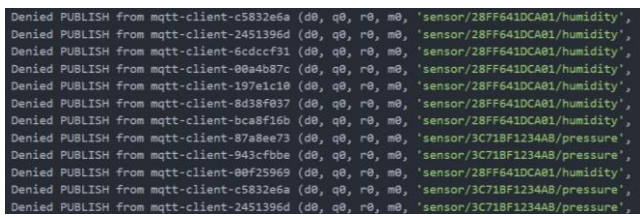


Figure 10. Log when failed to publish to random topics

Results of the test conducted over 120 seconds show that the MQTT broker continues to accept all connections and requests from the attacker without any alerts. Brute-Force attempts,

topic exploration, and high-frequency message delivery did not result in any warnings or blockages. All attack activities were recorded as normal traffic on the broker, so the system was unable to distinguish between legitimate and malicious traffic. This caused the broker's resources, as shown in Table 3 to tend to increase during attacks, especially during QoS abuse attacks that pushed CPU usage higher.

Table 3. Broker performance measurements

Type of Scenario	Port	Avg.Broker Performance	
		CPU (%)	Memory (%)
Normal	1883	1.34	32.20
Normal	8883	1.37	32.20
Brute-Force	1883	2.74	31.20
Distributed Brute-Force	1883	5.24	33.70
Brute-Force	8883	2.63	32.50
Distributed Brute-Force	8883	7.22	32.70
Brute-Force Topic	1883	2.12	31.20
QoS 0 Flood	1883	46.56	32.10
QoS 1 Flood	1883	48.34	32.10
QoS 2 Flood	1883	63.28	32.17

4.2.2 Snort detection scenario

In this scheme, Snort is activated as an IDS and configured to monitor traffic to the Mosquitto broker. The purpose of this scenario is to observe whether Snort can recognize attack patterns and generate logs and alerts according to the established rules. The same attack scenario is performed to see how the system detects attacks.

The detection and blocking times obtained from the ten repetitions of each scenario were summarized using the mean, sample standard deviation, 95% confidence interval, and minimum–maximum values. These descriptive statistics characterize the central tendency and run-to-run variability of the recorded measurements. Because no inferential comparison was performed, differences among the mean values should not be interpreted as statistically significant.

Table 4 presents the descriptive statistics of Snort detection time calculated from ten repetitions of each evaluated scenario.

The results presented in Table 4 show that Snort is capable of detecting attack activity and generating alerts when the number of connection attempts or packet transmissions exceeds the limits set in accordance with the configured rules. Information such as the source IP address of the attack, the time of the incident, and the type of attack is successfully recorded in the Snort log. During a QoS abuse attack, Snort exhibits a faster detection rate as the attack transmits packets rapidly, causing the thresholds within the Snort rules to be met more expeditiously.

Table 4. Descriptive statistics of snort detection time

Evaluated Activity	Port	n	Mean (s)	SD (s)	95% CI (s)	Min-Max (s)
Brute-force authentication	1883	10	6.010	0.837	5.411–6.609	4.000–6.900
Multi-source TCP/TLS connection-rate anomaly	8883	10	6.390	0.292	6.181–6.599	6.100–6.900
Distributed Brute-Force	1883	10	6.170	0.570	5.762–6.578	4.600-6.500
Distributed TCP/TLS anomaly	8883	10	6.463	0.301	6.247–6.679	6.030-6.900
Topic brute-force attack	1883	10	5.550	0.217	5.395–5.705	5.100-5.900
QoS 0 abuse	1883	10	0.426	0.045	0.394–0.458	0.370-0.500
QoS 1 abuse	1883	10	0.416	0.047	0.382–0.450	0.370-0.490
QoS 2 abuse	1883	10	0.402	0.031	0.380–0.424	0.360-0.450

Although the attack was successfully detected, as shown in Figure 11. No preventive or blocking measures were taken on the source IP, so as long as the attack continues, Snort will continue to generate alerts. This causes an increase in resources on the broker because the attack continues while Snort remains active in detecting attacks. Figures 12–14 show how resource statistics are affected during an attack.

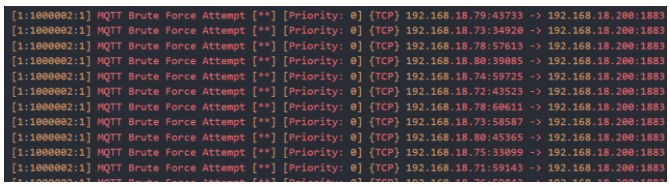


Figure 11. Snort alert when detecting an attack

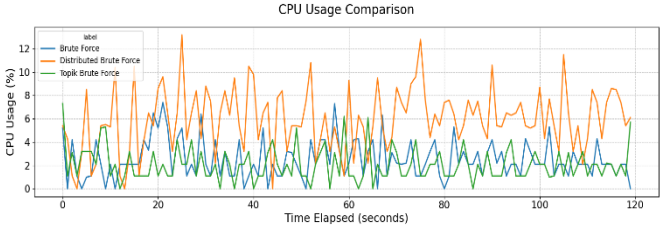


Figure 12. IDS CPU – Brute-Force, D. Brute-Force, Topic

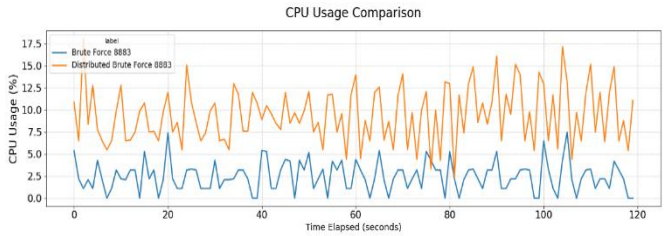


Figure 13. IDS CPU – Brute Force, D. Brute-Force Port 8883

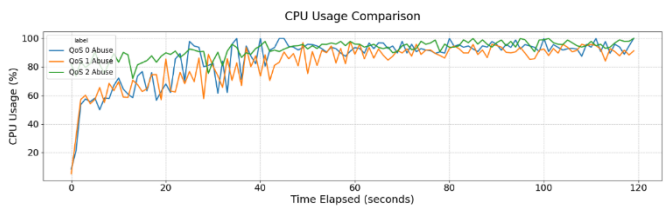


Figure 14. IDS CPU – QoS (0,1,2) Abuse

4.2.3 Integrated Snort and Fail2Ban scenario results

In this section, the entire system has been equipped with detection and prevention mechanisms. In this test, Snort will detect attack activity and generate logs, while Fail2Ban automatically reads these logs and blocks the attacker's IP address via iptables.

Figure 15 shows that the system successfully blocked the IP

address that carried out the attack. Fail2Ban works by reading the log path from Snort to recognize the type of attack and the source IP address of the attacker [18].

Tables 5 and 6 present the descriptive statistics obtained under the integrated Snort–Fail2Ban configuration. Table 5 reports the Snort detection time, whereas Table 6 reports the time required for Fail2Ban to apply the blocking action after the corresponding Snort alert was generated. Table 7 summarizes the mean detection and blocking times for all evaluated attack scenarios.

Test results show that when an attack is detected by Snort, Fail2Ban automatically blocks the source IP address of the attack through the firewall mechanism. After the block is implemented, further connections from that IP address will not be able to access the MQTT broker for a duration of 600 seconds. Thus, attacks can be stopped automatically without manual intervention. Figure 16 shows that the attack was successfully detected with a banned status.

```
Status for the jail: snort-bruteforce
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     16
|  `- File list:        /var/log/snort/alert
`- Actions
   |- Currently banned: 1
   |- Total banned:     11
   `- Banned IP list:   192.168.18.71
```

Figure 15. Fail2Ban log when successfully blocking IP

Table 5. Descriptive statistics of detection time in the integrated Snort–Fail2Ban configuration

Evaluated Activity	Port	n	Mean (s)	SD (s)	95% CI (s)	Min-Max (s)
Brute-force authentication	1883	10	6.450	0.510	6.085–6.815	5.200–6.900
Multi-source TCP/TLS connection-rate anomaly	8883	10	6.341	0.210	6.191–6.491	6.010–6.600
Distributed Brute-Force	1883	10	6.120	0.561	5.718–6.522	4.900–6.600
Distributed TCP/TLS anomaly	8883	10	6.420	0.278	6.221–6.619	6.100–6.900
Topic brute-force attack	1883	10	5.650	0.222	5.491–5.809	5.300–5.900
QoS 0 abuse	1883	10	0.409	0.044	0.377–0.441	0.350–0.490
QoS 1 abuse	1883	10	0.383	0.047	0.349–0.417	0.340–0.450
QoS 2 abuse	1883	10	0.428	0.033	0.404–0.452	0.390–0.490

Table 6. Descriptive statistics of Fail2Ban blocking time after Snort alert generation

Evaluated Activity	Port	n	Mean (s)	SD (s)	95% CI (s)	Min-Max (s)
Brute-force authentication	1883	10	0.830	0.206	0.683–0.977	0.600–1.200
Multi-source TCP/TLS connection-rate anomaly	8883	10	0.955	0.261	0.768–1.142	0.500–1.400
Distributed Brute-Force	1883	10	0.980	0.322	0.749–1.211	0.500–1.400
Distributed TCP/TLS anomaly	8883	10	0.880	0.338	0.638–1.122	0.300–1.300
Topic brute-force attack	1883	10	1.070	0.279	0.870–1.270	0.400–1.300
QoS 0 abuse	1883	10	0.872	0.305	0.654–1.090	0.400–1.360
QoS 1 abuse	1883	10	0.657	0.362	0.398–0.916	0.200–1.070
QoS 2 abuse	1883	10	0.620	0.091	0.555–0.685	0.510–0.770

Table 7. Detection and blocking times of the integrated Snort–Fail2Ban system

Evaluated Activity	Duration (s)	Port	Mean Detection Time (s)	Mean Blocking Time (s)
Authentication brute-force attack	120	1883	6.45	0.83
Distributed Authentication Brute-Force Attack	120	1883	6.12	0.98
TCP/TLS connection-rate anomaly	120	8883	6.34	0.95
Multi-source TCP/TLS connection-rate anomaly	120	8883	6.42	0.90
Topic Brute-Force Attack	120	1883	5.65	1.07
QoS 0 abuse	120	1883	0.409	0.87
QoS 1 abuse	120	1883	0.38	0.65
QoS 2 abuse	120	1883	0.42	0.62

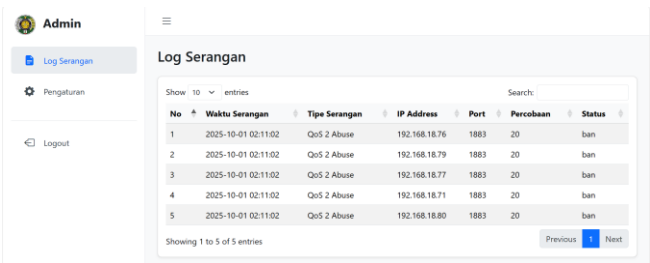


Figure 16. Attack log on web dashboard

The combination of these two systems, Snort and Fail2Ban, also successfully suppressed CPU spikes, especially during QoS abuse attacks that consumed a lot of broker resources. This allowed MQTT communication to continue and remain stable in the face of malicious activity. Figures 17–19 show how this system can suppress CPU spikes during an attack.

When the configured Snort threshold was satisfied, Snort generated an alert containing the source address and attack category. Fail2Ban subsequently processed the alert and inserted an iptables rule that blocked further traffic from the identified address. Thus, the packets required to satisfy the detection threshold had already reached the host before the blocking action was applied. The mechanism limited subsequent traffic during the configured ban period rather than preventing every attack packet from reaching the broker host.

The reduction in CPU utilization after the initial increase is associated with the firewall dropping subsequent packets from the blocked source addresses. The result demonstrates post-detection mitigation under the evaluated traffic conditions. It does not demonstrate pre-connection filtering or prevention of

all malicious packets before they reached the broker host.

The integrated mechanism detected the configured traffic patterns and blocked the associated source addresses after the detection thresholds were reached. Following the blocking action, subsequent connections from those addresses were rejected for the configured ban duration. This response reduced the continued processing of malicious traffic and was associated with lower broker CPU utilization in the evaluated scenarios.

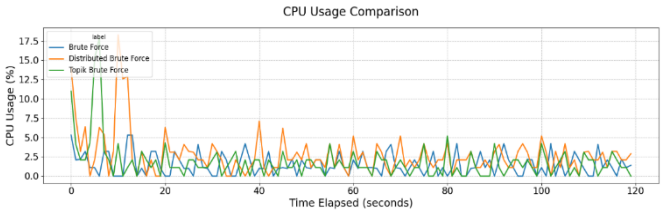


Figure 17. IPS CPU – Brute-Force, D.Brute-Force, Topic

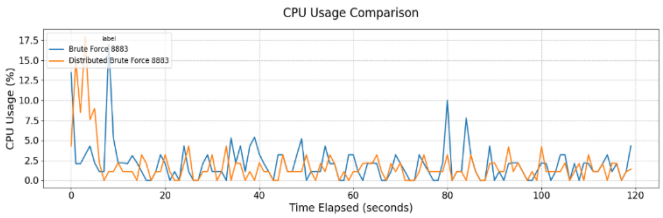


Figure 18. IPS CPU – Brute-Force, D. Brute-Force Port 8883

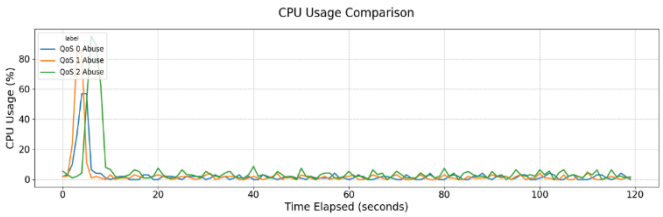


Figure 19. IPS CPU – QoS (0,1,2) Abuse

4.3 Limitation

The attack scenarios evaluated in this study were constructed according to the packet characteristics and threshold conditions defined in the proposed Snort rules. Therefore, the reported results demonstrate the operation of the integrated Snort–Fail2Ban mechanism only under the specific traffic patterns, rates, and configurations examined in the experiment. The evaluation did not include attack patterns developed independently from the detection rules, previously unseen attack patterns, slow brute-force attempts, distributed sources operating below the per-source threshold, malformed MQTT packets, altered traffic rates, or deliberate rule-evasion techniques. Consequently, the findings should not be generalized to all variants of MQTT attacks.

Because the proposed detection mechanism relies on predefined signatures and traffic thresholds, malicious activities that remain below the configured threshold may produce false-negative results. Conversely, legitimate clients that generate connection or publication bursts above the threshold may be classified as suspicious. The present study did not quantitatively measure false-positive and false-negative rates under these conditions.

The evaluation of encrypted MQTT traffic on port 8883 was limited to TCP/TLS connection-rate monitoring. Because the MQTT authentication exchange was encrypted, the SYN-based Snort rule could not inspect usernames, passwords, certificate-validation results, MQTT CONNECT fields, or broker authentication responses. Accordingly, alerts generated for port 8883 indicate that the configured TCP connection threshold was exceeded rather than confirming an MQTT authentication brute-force attack. In addition, destination-based tracking may aggregate concurrent connection attempts from multiple sources and could classify legitimate TLS connection bursts as suspicious traffic. The present study did not separately evaluate this potential false-positive condition using multiple legitimate TLS clients.

Although the distributed scenario employed up to ten attacker containers, the precise per-source attempt rate, aggregate attack rate, inter-attempt interval, and container start-time differences were not retained in the original experimental records. For port 1883, the per-source detection records indicate that each source triggered an alert under the tested configuration. However, the experiment does not represent a distributed low-rate attack in which each source remains below the configured threshold. For port 8883, destination-based tracking prevents the results from establishing whether every source independently exceeded the detection threshold.

Future work should evaluate the proposed mechanism using attack patterns developed independently from the detection rules. The evaluation should include slow brute-force attempts, distributed sub-threshold sources, variable transmission rates, malformed MQTT packets, deliberate rule-evasion techniques, and representative legitimate high-rate workloads. Future experiments should also report false-positive and false-negative rates and correlate Snort alerts with Mosquitto authentication logs, particularly for encrypted MQTT traffic.

5. CONCLUSIONS

The experimental results show that the Mosquitto broker configuration evaluated without Snort and Fail2Ban did not include a network-level mechanism capable of distinguishing the tested malicious traffic from legitimate MQTT communication. Under this baseline configuration, authentication brute-force attempts, topic exploration, and high-frequency QoS message publication continued to reach the broker without generating network-level detection alerts or automated blocking actions. These findings indicate that the evaluated broker configuration required an additional monitoring and response mechanism to detect and mitigate the specific attack patterns examined in this study.

The implementation of Snort as an intrusion detection system improved the visibility of suspicious network activities that matched the predefined rules and threshold conditions. Snort generated alerts and logs containing the source IP address, detection time, and corresponding rule information. However, Snort operated only as a detection mechanism in this configuration. Consequently, malicious traffic continued to reach the broker after detection when no automated response mechanism was enabled. The continued processing of attack traffic provides a plausible explanation for the observed broker resource utilization; however, the experiment did not separately isolate the computational overhead introduced by Snort.

The integration of Snort and Fail2Ban provided automated post-detection mitigation under the evaluated traffic patterns

and threshold configurations. Under the integrated configuration, Snort detected authentication brute-force activities on port 1883 in approximately 6.12–6.45 seconds, TCP/TLS connection-rate anomalies on port 8883 in approximately 6.34–6.42 seconds, topic brute-force activities in approximately 5.65 seconds, and QoS abuse in approximately 0.38–0.42 seconds. Following alert generation, Fail2Ban processed the Snort alerts and blocked the associated source addresses through the host firewall in approximately one second. The initial packets used to satisfy the configured Snort thresholds reached the broker host before blocking occurred; therefore, the mechanism should be interpreted as post-detection mitigation rather than as prevention of all malicious traffic before it reached the broker.

The findings demonstrate the operation of the integrated Snort–Fail2Ban mechanism only under the attack patterns, traffic rates, and threshold configurations evaluated in this study. They do not establish protection against all MQTT attack variants. In particular, unseen, slow, distributed sub-threshold, and deliberately evasive attack patterns were not evaluated. Furthermore, the results for encrypted MQTT communication on port 8883 represent TCP/TLS connection-rate anomaly detection rather than direct detection of MQTT authentication failures. Future research should evaluate the mechanism using attack patterns developed independently from the detection rules, variable traffic rates, representative legitimate workloads, and quantitative false-positive and false-negative measurements.

REFERENCES

[1] Husnain, M., Hayat, K., Cambiaso, E., et al. (2022). Preventing MQTT vulnerabilities using IoT-enabled intrusion detection system. Sensors, 22(2): 567. https://doi.org/10.3390/s22020567

[2] Mansour, M., Gamal, A., Ahmed, A.I., et al. (2023). Internet of things: A comprehensive overview on protocols, architectures, technologies, simulation tools, and future directions. Energies, 16(8): 3465. https://doi.org/10.3390/en16083465

[3] Palmieri, A., Prem, P., Ranise, S., Morelli, U., Ahmad, T. (2019). MQTTSA: A tool for automatically assisting the secure deployments of MQTT brokers. In 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, pp. 47-53. https://doi.org/10.1109/SERVICES.2019.00023

[4] Light, R.A. (2017). Mosquitto: Server and client implementation of the MQTT protocol. Journal of Open Source Software, 2(13): 265. https://doi.org/10.21105/joss.00265

[5] Chen, F., Huo, Y., Zhu, J., Fan, D. (2020). A review on the study on MQTT security challenge. In 2020 IEEE International Conference on Smart Cloud (SmartCloud), Washington, DC, USA, pp. 128-133. https://doi.org/10.1109/SmartCloud49737.2020.00032

[6] Kant, D., Johannsen, A., Creutzburg, R. (2021). Analysis of IoT security risks based on the exposure of the MQTT protocol. Electronic Imaging, 33: 1-8.

[7] Stiawan, D., Idris, M.Y., Malik, R.F., Nurmaini, S., Alsharif, N., Budiarto, R. (2019). Investigating brute force attack patterns in IoT network. Journal of Electrical and Computer Engineering, 2019(1): 4568368. https://doi.org/10.1155/2019/4568368

[8] Laghari, S.U.A., Li, W., Manickam, S., Nanda, P., Al-Ani, A.K., Karuppayah, S. (2024). Securing MQTT ecosystem: Exploring vulnerabilities, mitigations, and future trajectories. IEEE Access, 12: 139273-139289. https://doi.org/10.1109/ACCESS.2024.3412030

[9] Swathi, K. (2022). Brute force attack on real world passwords. International Journal of Research Publication and Reviews, 3(11): 552-558.

[10] Hamza, A.A., Al Janabi, R.J.S. (2024). Detecting brute-force attacks using machine learning. BIO Web of Conferences, 97: 00045. https://doi.org/10.1051/bioconf/20249700045

[11] Akbar, G.M.I., Hariyadi, M.A., Hanani, A. (2023). Detection of Bruteforce attacks on the MQTT protocol using random forest algorithm. Internet of Things and Artificial Intelligence Journal, 3(3): 250-272. https://doi.org/10.31763/iota.v3i3.630

[12] Boukebous, A.A.E., Fettache, M.I., Bendiab, G., Shiaeles, S. (2023). A comparative analysis of Snort 3 and Suricata. In 2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET), London, United Kingdom, pp. 1-6. https://doi.org/10.1109/GlobConET56651.2023.10150141

[13] De Rango, F., Tropea, M., Fazio, P. (2020). Mitigating DoS attacks in IoT EDGE Layer to preserve QoS topics and nodes' energy. In IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, pp. 842-847. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162902

[14] Gunawan, G.B., Sukarno, P., Putrada, A.G. (2018). Pendeteksian Serangan Denial of Service (DoS) pada perangkat smartlock berbasis Wifi menggunakan SNORT IDS. e-Proceeding of Engineering, 5(3): 7875-7884.

[15] da Silva, L.M., Menezes, H.B.D.B., Luccas, M.D.S., et al. (2022). Development of an efficiency platform based on MQTT for UAV controlling and DoS attack detection. Sensors, 22(17): 6567. https://doi.org/10.3390/s22176567

[16] Nadeem, M., Arshad, A., Riaz, S., Band, S.S., Mosavi, A. (2021). Intercept the cloud network from brute force and DDoS attacks via intrusion detection and prevention system. IEEE Access, 9: 152300-152309. https://doi.org/10.1109/ACCESS.2021.3126535

[17] Farhannullah, Hardjianto, M. (2022). Sistem monitoring serangan SSH dengan metode intrusion prevention system (IPS) Fail2ban menggunakan Python pada sistem operasi Linux. Jurnal TICOM: Technology of Information and Communication, 11(1): 33-38. https://doi.org/10.70309/ticom.v11i1.68

[18] Abdullah, R.K., Fudhail, M.T., Mujahidin, S. (2024). Penggunaan snort dan Fail2Ban sebagai IDS untuk mengatasi brute force attack dengan notifikasi telegram: Studi kasus pada institusi XYZ. Jurnal Sistem dan Teknologi Informasi, 12(3): 530-542. https://doi.org/10.26418/justin.v12i3.79617

[19] Dawamsyach, F., Ruslianto, I., Ristian, U. (2023). Implementation of IPS (Intrusion Prevention System) Fail2ban on Server for DDoS and Brute Force Attacks. CESS (Journal of Computer Engineering, System and Science), 8(1): 149-161.

https://doi.org/10.24114/cess.v8i1.40259
[20] Muakhori, I., Sunardi, Fadlil, A. (2020). Security of dynamic domain name system servers against DDoS attacks using IPTABLE and FAIL2BA. Jurnal Mantik, 4(1): 41-49.
[21] Mishra, B., Mishra, B., Kertesz, A. (2021). Stress-testing MQTT brokers: A comparative analysis of performance measurements. Energies, 14(18): 5817. https://doi.org/10.3390/en14185817
[22] Anwer, H., Azam, F., Anwar, M.W., Rashid, M. (2020). A model-driven approach for load-balanced MQTT protocol in Internet of Things (IoT). In Complex, Intelligent, and Software Intensive Systems. CISIS 2019. Springer, Cham, pp. 368-378. https://doi.org/10.1007/978-3-030-22354-0_33
[23] Mukkamala, P.P., Rajendran, S. (2020). A survey on the different firewall technologies. International Journal of Engineering Applied Sciences and Technology, 5(1): 363-365. https://doi.org/10.33564/IJEAST.2020.V05I01.059
[24] Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F., Yusupov, J. (2023). Automated firewall configuration in virtual networks. IEEE Transactions on Dependable and Secure Computing, 20(2): 1559-1576. https://doi.org/10.1109/TDSC.2022.3160293
[25] Efe, A., Abacı, İ.N. (2022). Comparison of the host based intrusion detection systems and network based intrusion detection systems. Celal Bayar University Journal of Science, 18(1): 23-32. https://doi.org/10.18466/cbayarfbe.832533