Real Time Detection of Man-in-the-Middle Attacks on Message Queing Telemetary Transport Based Networks Using Machine Learning

Mubarak Haruna Bako
Physics Department
Bayero University
Kano, Nigeria
mubarakmubeey@gmail.com

M.I.Bello
Physics Department
Bayero University
Kano, Nigeria
mibello.elt@buk.edu.ng

S.M.Gana
Physics Department
Bayero University
Kano, Nigeria
smgana.phy@buk.edu.ng

N. M Shehu
Physics Department
Bayero University
Kano, Nigeria
nmshehu.phy@buk.edu.ng

***Abstract*— The Message Queuing Telemetry Transport (MQTT) protocol is increasingly used in IoT systems due to its low communication overhead. However, its lightweight nature and lack of built-in encryption make it susceptible to various network-layer attacks, particularly Man-in-the-Middle (MitM) attacks. This paper presents a machine learning-based Intrusion Detection System (IDS) designed to detect adversarial modifications in MQTT traffic within a realistic IoT environment. A physical testbed was developed using ESP32 devices and multiple sensors communicating through an MQTT broker. Several attack scenarios were emulated, including message tampering, data injection, and selective packet dropping. Network traffic was collected and processed using a combination of flow-based and MQTT-specific feature extraction techniques. A Random Forest classifier was trained on the resulting dataset after applying standard preprocessing and balancing techniques. The system was further extended with a lightweight real-time detection module operating on live MQTT flows. Results indicate that the proposed approach can effectively distinguish malicious activity from legitimate traffic with promising detection performance.**

Keywords*— *Internet of Things, MQTT Protocol, Intrusion Detection System, Man-in-the-Middle Attack, Machine Learning.

I. INTRODUCTION

The Internet of Things (IoT) has become a foundational element in modern intelligent systems, enabling real-time monitoring, automation, and data-driven decision-making across domains such as healthcare, agriculture, smart homes, and industrial control systems. As these systems grow in scale and complexity, the communication protocols used to connect devices become critical to both functionality and security. One of the most widely adopted protocols in IoT environments is the Message Queuing Telemetry Transport (MQTT), a lightweight publish-subscribe protocol designed for low-bandwidth, high-latency networks [1].

While MQTT's efficiency and simplicity make it suitable for constrained environments, its lack of native encryption, authentication, or integrity verification introduces significant security vulnerabilities. In particular, Man-in-the-Middle (MitM) attacks are a known threat vector, wherein an attacker silently intercepts and manipulates MQTT messages between publishers and subscribers. This can result in the injection of false sensor data, suppression of critical alerts, or unauthorized actuation in safety-critical systems [2], [3].

Conventional intrusion detection systems (IDS) are often ineffective in MQTT-based networks, especially when they rely on transport-layer inspection or static rule sets. Such systems typically fail to capture the nuances of application-layer protocols like MQTT, particularly when the attacks involve subtle payload manipulations or topic-specific tampering [4]. This limitation has led to growing interest in the use of machine learning (ML) for anomaly detection in IoT networks, where data-driven models are trained to distinguish between benign and malicious behavior patterns [5], [6].

This study presents the design and implementation of a supervised ML-based IDS for detecting MitM attacks targeting MQTT traffic in a realistic IoT testbed. A hardware-based system was constructed using ESP32 nodes connected through a local MQTT broker, with multiple attack scenarios simulated via ARP spoofing and payload manipulation. Features were extracted from both flow-level and MQTT-layer characteristics, including entropy, retransmission patterns, and topic frequency. A Random Forest classifier was trained on the resulting dataset, and a real-time detection module was deployed for live monitoring of MQTT flows.

The remainder of this paper is organized as follows: Section II presents an overview of related work in the field, highlighting key studies and methodologies. Section III provides a detailed description of the system architecture, along with the proposed attack model and its relevance. Section IV outlines the dataset used, including the data collection process and necessary preprocessing steps to prepare it for analysis. Section V delves into the feature extraction techniques and classification methods applied to detect anomalies or intrusions. Section VI discusses the results and deployment of the real-time Intrusion IDS,

including its integration and operational considerations. Finally, Section VII concludes the paper by summarizing the findings.

II. RELATED WORK

The increasing deployment of MQTT in IoT environments has led to a wide range of research efforts aimed at securing communication against various forms of cyberattacks, including MitM scenarios. Several intrusion detection approaches have been proposed using classical machine learning, deep learning, ensemble models, and hybrid feature engineering. However, limitations persist in terms of realism, deployment feasibility, and detection granularity.

Khan et al. [7] presented a deep learning-based IDS using MQTT traffic features extracted from public datasets. Their system achieved high classification accuracy using DNNs, GRUs, and LSTMs, but the attacks were simulated and did not involve live payload manipulation. Ullah et al. [8] introduced TNN-IDS, a transformer-based IDS for MQTT-enabled networks. While it demonstrated strong performance on benchmark datasets, the study lacked real-world deployment or adversarial data injection.

Al Hanif and Ilyas [9] proposed an ensemble learning model combining XGBoost and LightGBM to detect brute-force and DoS attacks in MQTT environments. Although their system achieved high accuracy, it focused on volumetric attacks and omitted application-layer payload changes. Similarly, Fortino et al. [10] used convolutional neural networks for MQTT attack detection but relied entirely on offline, synthetic datasets.

Feature selection and optimization were explored by S. C. Prajisha and Vasudevan [11], who applied an enhanced chaotic salp swarm algorithm to reduce irrelevant attributes. Their classifier achieved good results but did not consider real-time constraints or topic-specific tampering. Discover IoT's 2025 article [12] used Random Forests and LSTMs to detect MitM in IoT, though the attack implementations remained abstract and lacked message-level manipulations.

Further studies such as those by Alshamrani et al. [13] and Mujahid et al. [14] presented multi-class detection techniques using datasets like MQTT-IoT-IDS2020. These works primarily simulate attack traffic without deploying the scenarios on physical hardware. While Nkemelu et al. [15] introduced a packet representation-based detection system for efficiency, it lacks sensitivity to application-level anomalies like altered MQTT topics or entropy shifts.

Specification-level detection approaches have also emerged. Shukla et al. [16] developed an IDS focused on malformed MQTT messages, integrating their parser into an existing network IDS. This work is valuable for protocol compliance but is less applicable to subtle adversarial content manipulation. IDS-MA [17] investigated centralized and federated learning methods for MQTT-based IDSs but did not explore application-layer features or router-based deployment.

Finally, Mishra et al. [18] proposed MediHunt, a forensic framework for medical IoT traffic that includes MQTT detection. Though aligned with real-time scenarios, the work primarily monitors communication patterns rather than detecting fine-grained attacks like selective packet dropping or message injection. Despite the diversity of approaches, a common limitation across these works is the lack of hardware-based, router-level traffic collection involving active MitM attacks that manipulate MQTT payloads in-flight. Moreover, application-specific features such as topic frequency, payload entropy, ARP anomalies, and retransmission counts are seldom incorporated. Real-time detection modules capable of operating in constrained environments are also rarely demonstrated.

In contrast, the present study addresses these limitations by implementing live adversarial attack scenarios in a physical testbed, extracting both flow-level and MQTT-specific features, and deploying a lightweight real-time IDS using trained machine learning models

III. IOT TESTBED ARCHITECTURE AND THREAT MODEL

A. Testbed Overview

To simulate realistic MQTT-based IoT environments under adversarial conditions, a physical testbed was constructed using commercially available microcontrollers and sensors. The system consists of two ESP32 development boards ESP32A and ESP32B which act as smart devices communicating over an MQTT broker deployed within a local area network (LAN) managed by an OpenWRT router. Each ESP32 node is equipped with various sensors and actuators to emulate a multi-device IoT ecosystem as shown in Table I.

TABLE I. TESTBED ARCHITECTURE

Node	Sensors/ Actuators	Publish Topics	Subscribe Topics
ESP32A	DHT22, PIR, LDR, Buzzers.	esp32a/temp, esp32a/humidity, esp32a/motion, esp32a/ldr, esp32a/keypad	esp32b/rain, esp32b/flame
ESP32B	Rain sensor, Flame sensor, RFID reader, PWM-controlled LED, Servo motor.	esp32b/rain, esp32b/flame, esp32b/rfid	esp32a/temp, esp32a/motion

All messages in this system are exchanged using the MQTT protocol, with a Quality of Service (QoS) level set to 0, which ensures that messages are delivered at most once and without acknowledgment. This means there is no guarantee of delivery, but the overhead is minimized, making it suitable for applications where speed is critical and occasional message loss is acceptable. These messages are transmitted over port 1883, the default port for MQTT communication, which is typically used for unencrypted data transmission.

The broker that facilitates the communication between the various devices in the network is hosted on a machine that is directly accessible within the LAN. This means the MQTT broker is locally hosted rather than using a cloud-based service, offering faster response times and greater control over the data flow within the network.

The OpenWRT router, which is installed with a customizable, open-source firmware, plays a crucial role in ensuring full visibility over the network traffic. By being positioned at the gateway between the devices and the wider network, the router can monitor all data packets being transmitted and received. This setup provides a vantage point for network analysis and troubleshooting, as it allows the capture of network packets for inspection.

Additionally, the OpenWRT router's capabilities can be leveraged for more advanced network security measures, such as the execution of a man-in-the-middle (MitM) attack. In a MitM scenario, the router intercepts and potentially alters the communication between the MQTT broker and the client devices, without either party being aware. This could be used for debugging, security testing, or even exploiting vulnerabilities in the communication protocol, depending on the situation and the security measures in place.

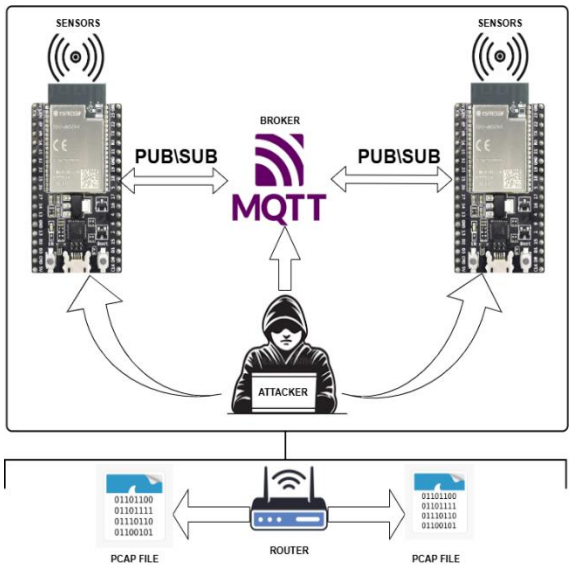

Figure 1: *Testbed architecture*

B. Communication and Control Flow

In normal operation, each ESP32 node publishes sensor data periodically (2 seconds) and responds to subscribed topics with actuation events. For instance:

- ESP32B triggers a servo when a valid RFID tag is scanned.
- ESP32A activates a buzzer when a flame or rain alert is received.
- Esp32B varies the led light intensity using Esp32A ldr analog value.

MQTT topic and payload structures are string-based and unencrypted, exposing the system to content-level manipulation.

C. Threat Model

The adversary is positioned within the same LAN as the IoT devices and broker, having access to intercept and manipulate packets via ARP spoofing and redirection. The attacker does not compromise the broker or devices directly but relies on network-level interception to manipulate live MQTT traffic.

IV. Data Acquisition and Preprocessing

A. Data Capture Setup

Network traffic was captured directly from the LAN using a customized OpenWRT router flashed on a TP-Link MR3420 device. All MQTT traffic passed through this router, allowing full visibility of both benign and adversarial communication. The capture tool used was tcpdump, configured with appropriate port filters. Each scenario was captured for five minutes, producing 15 normal and 15 attack PCAP files. All ESP32 nodes, the broker, and the attacker device operated simultaneously during traffic generation to reflect realistic interaction.

B. Scenario Design and Dataset Structure

To promote generalizability, both normal and attack traffic were recorded under three distinct behavioral scenarios. This improves feature diversity and robustness of ML models. Each scenario was executed five times (5 minutes each), resulting in 30 total captures:

1) NSC1–Baseline operation

a) Idle, regular sensor publishes

b) No active events

2) NSC2–Active sensor triggers

a) Motion, RFID, rain, flame, keypad triggered manually

3) NSC3–High traffic stress test

a) Publish rates increased

b) All sensors triggered repeatedly

4) ASC1–Payload Modification (Silent Alert Suppression):

a) Alters esp32b/rain payloads from "1" to "0" in transit.

b) Prevents ESP32A from triggering rain alert buzzers.

c) Detection relies on entropy and topic frequency mismatch.

5) ASC2 – Fake Data Injection:

a) Periodically injects fake esp32a/temp messages with value "999".

b) Appears to originate from ESP32A but bypasses the device entirely.

c) Mimics abnormal sensor readings in the data stream.

6) ASC3 – Selective Packet Drop (Stealth Tampering):

a) Drops a subset (50%) of esp32b/flame messages.

b) Leads to intermittent suppression of fire alerts on ESP32A.

7) Requires behavioral anomaly detection (e.g., topic silence during expected activity).

Each attack is executed using a custom Python script acting as an MQTT-aware TCP-level proxy. Messages are intercepted and altered without breaking the protocol flow, preserving CONNECT, SUBSCRIBE, and PUBLISH sequences.

C. Data Cleaning and Filtering

Raw PCAPs included noise due to network fluctuations, retransmissions, and excessive ARP traffic during MitM attacks. To reduce this:

- TCP retransmissions and duplicate ACKs were filtered using tshark.
- ARP filtering retained only malicious ARP replies.
- Cleaned MQTT and filtered ARP streams were merged using Wireshark's mergecap utility.

This preprocessing reduced total packet counts by up to 50% in some attack PCAPs, removing unnecessary noise while preserving key attack indicators.

D. Feature Extraction

All cleaned PCAPs were processed through CICFlowMeter v4.0, producing enriched flow-level features such as:

1) Flow Duration, Packet Length Mean
2) Forward/Backward IAT (Inter-Arrival Time)
3) PSH Flag Count, Fwd/Bwd Header Len
4) Fwd Act Data Packets, Bwd Pkts/s

In total, 83 bidirectional features were extracted per flow and custom application-layer features were merged using a Python script. All features were aligned by timestamp and Flow ID where applicable. A unified dataset of over 15,000 network flows across 30 scenarios was created, with a balanced 50/50 split between normal (NSC1–3) and attack (ASC1–3) traffic, each flow represented by approximately 90 features and clearly labeled as 0 for normal or 1 for attack.

V. Machine Learning Model Design and Evaluation

A. Feature Selection and Dataset Preparation

The cleaned dataset, containing both CICFlowMeter features and MQTT-specific enhancements, was loaded into a Pandas DataFrame. To ensure meaningful training and reduce redundancy, a multi-step feature selection approach was applied:

1) Information Gain (IG) Filtering

Each feature's mutual information was computed with the target label and used permutation-based information gain analysis to identify and discard potentially leaky or trivial features specifically those showing high information gain on both real and randomly shuffled labels to reduce the risk of overfitting, as defined in (1)

$$IG(Y,X) = H(Y) - H(Y \mid X) \quad (1)$$

Where $IG(Y,X)$ is the information gain of feature X with respect to label Y, $H(Y)$ is the entropy of the target variable, $H(Y|X)$ is the conditional entropy of the label given the feature.

2) Manual Feature Pruning

Network-identifying fields such as Flow ID, Src IP, Dst IP, and Timestamp were removed. Protocol-level categorical fields were also excluded due to limited variability in the controlled environment.

Table II shows the features selected and their descriptions. These features collectively capture both network-layer anomalies and application-layer irregularities.

TABLE II. Selected Features

NO.	Feature	Description
1	Fwd IAT Tot	Total time for forward packets.
2	Bwd IAT Tot	Total time for backward packets.
3	Fwd Header Len	Length of forward packet headers.
4	Bwd Header Len	Length of backward packet headers.
5	PSH Flag Cnt	Count of PSH flag packets.
6	Fwd Act Data Pkts	Data packets sent forward.
7	ARP_Count	Count of ARP packets.
8	Retrans_Count	Count of retransmitted packets.
9	Payload_Entropy_Mean	Average randomness of payload data.
10	MQTT_Topic_Freq_Rain	MQTT topic frequency/variability.

3) Data Scaling and Balancing

To ensure uniform feature scaling, a Standard Scaler was fitted on the training set and applied to all inputs. Class imbalance common in intrusion detection tasks was addressed using SMOTE, which synthetically balances minority class samples through nearest-neighbor interpolation.

B. Classifier Selection and Training

After experimenting with several classifiers, including Logistic Regression and XGBoost, a Random Forest (RF) model was selected for its interpretability, resistance to overfitting, and strong performance on small-to-medium feature spaces. RF constructs an ensemble of K decision trees $h\_1(x), h\_2(x), \ldots, h\_K(x)$, each trained on a random bootstrap sample and a random subset of features. The final prediction is made via majority voting as describe in (2)

$$\hat{y} = mode\{\, h_k(x),\ k = 1, 2, \ldots, K \,\} \quad (2)$$

Where $\hat{y}$ is the final predicted class label, $h_k(x)$ is the prediction from the k decision tree in the ensemble, k is the total number of trees in the Random Forest.

Each individual decision tree selects the best split at each node based on the Gini impurity, a measure of class heterogeneity. For a node containing samples from C different classes with relative class frequencies $\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_\mathrm{C}$, the Gini impurity is computed as:

$$G = 1 - \sum_{\{i=1\}}^{\{C\}} p_i^2 \quad (3)$$

Where G is the Gini impurity at a given node in a tree, C is the number of unique classes (in this case, $C = 2$: normal or attack), p_i is the proportion of samples belonging to class i at the node.

C. Model Deployment

For real-time use, the final model pipeline (classifier, scaler and selected feature list) was exported using joblib and

integrated into a lightweight Scapy-based live detection script deployed on a monitoring system connected to the LAN. The system operates passively, sniffing MQTT and ARP traffic on the interface using scapy.sniff(), without interfering with the packet forwarding logic of the network.

As packets are captured, the system aggregates them into temporary flow structures based on source/destination IPs and ports. Once a minimum window size (typically 10 packets) is reached, it computes the selected features such as Payload_Entropy_Mean,MQTT_Topic_Freq_Rain,ARP_Count, and Retrans_Count. These features are normalized using the previously exported StandardScaler and passed into the trained Random Forest model to predict whether the flow is anomalous. Upon detecting an anomaly, the system immediately logs the event to the console.

VI. RESULT AND DISCUSSION

A. Dataset Generation

Our evaluation is based on a novel dataset generated from a physical IoT testbed, comprising 30 packet captures across three benign and three adversarial scenarios. Capturing traffic in a real-world environment ensures the dataset authentically includes network behaviors like packet retransmissions, ARP floods, MQTT topic manipulations, and timing shifts. To create a robust feature set, standard network flow data from CICFlowMeter was augmented with MQTT-specific metrics, such as payload entropy and topic frequency, resulting in a more nuanced collection for analysis than previously available.

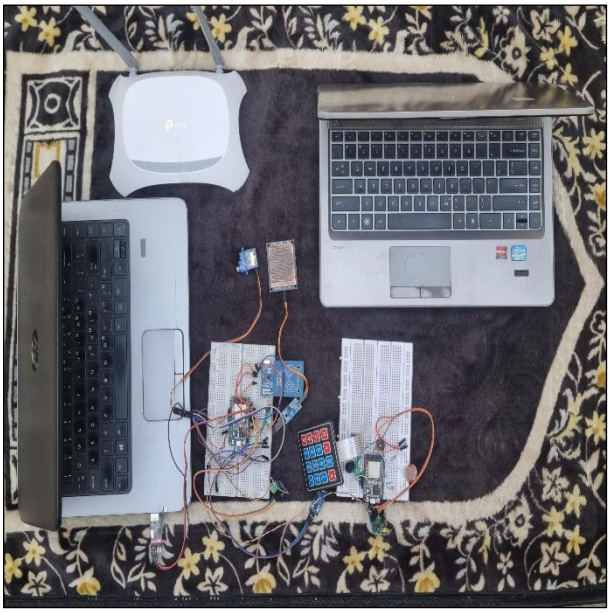

Figure 2: *IoT Testbed*

B. ML Model Performance

The Random Forest model was trained using the final preprocessed dataset consisting of hybrid features extracted from both network and application layers. During 5-fold cross-validation, the model achieved consistently high performance across multiple metrics with accuracy 99%, precision 98%, recall 99%, and F1-score 98%. These results demonstrate strong generalization capability and low false alarm rates.

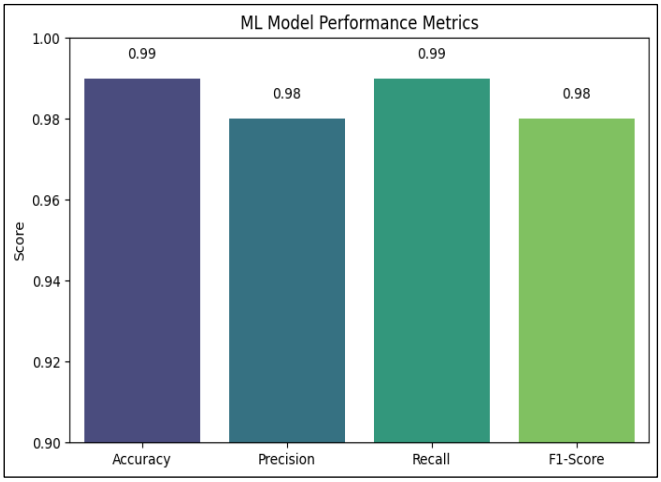

Figure 3: *Evaluation metrics*

To further understand what contributed to the classifier's predictive power, a feature importance analysis was conducted. The most influential indicators included:

1) ARP_Count: reflecting spoofing activity in the LAN
2) Retrans_Count: suggesting congestion or redirection
3) Payload_Entropy_Mean: capturing static tampering of MQTT payloads
4) MQTT_Topic_Freq_Rain: highlighting repeated topic injections or suppressions

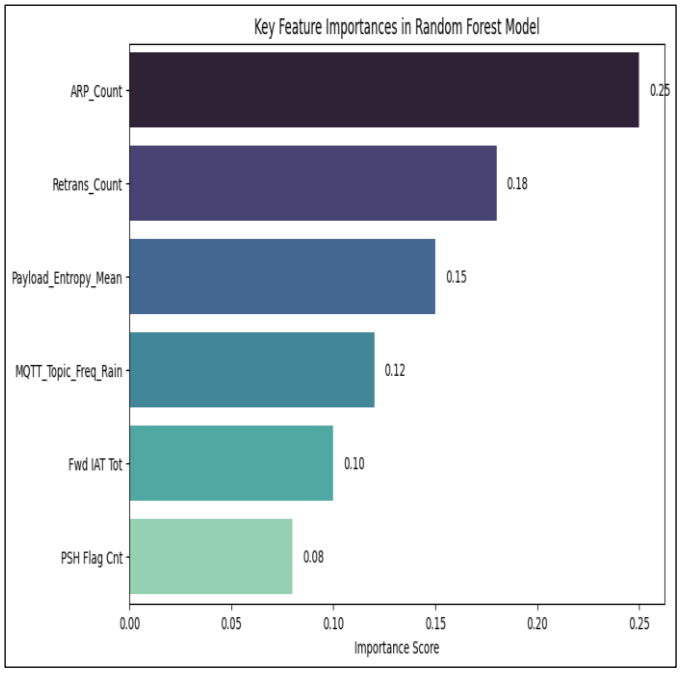

Figure 3: Features ranking

These results validate the design choice of including both flow-level metrics like retransmissions and MQTT-specific application-layer features like entropy and topic frequency, which collectively offer a robust attack signal.

C. Deployment

To ensure real-world applicability, the trained model and its preprocessing pipeline were integrated into a lightweight Python-based detection script. The engine passively monitors MQTT traffic over the LAN, extracts flow-based and MQTT-aware features in real-time, and performs inference using the preloaded model. Resource usage was tracked during a 10-cycle simulation. The system maintained:

1) *CPU usage under 6%*
2) *Memory usage under 11 MB*
3) *Flow inference latency stable around 200 ms*

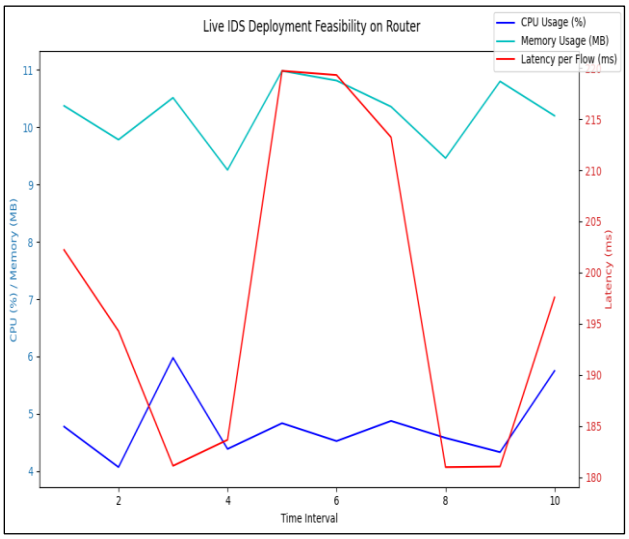

Figure 4: Resource usage

This confirms that the detection engine is deployable on low-resource hardware such as OpenWRT routers or Raspberry Pi-class devices, without interfering with normal network operation.

VII. Conclusion

This study presented a lightweight, real-time IDS designed to detect adversarial MitM attacks in MQTT-based IoT environments. A complete experimental pipeline was implemented from the development of a realistic testbed using ESP32 microcontrollers to the capture of real-world attack traffic and deployment of a trained model on a LAN monitoring node. Unlike many existing approaches that rely solely on flow-level statistics or simulation-based datasets, this work introduced a hybrid feature extraction method that combines conventional network features with application-layer indicators. This enabled the detection of subtle attack scenarios such as alert suppression and payload injection, which often evade rule-based or stateless systems.

The Random Forest classifier trained on this dataset achieved high performance across cross-validation folds (F1-score: 98%), while permutation testing confirmed the statistical validity of these results. Feature importance analysis further highlighted the relevance of MQTT-specific indicators in detecting protocol-aware adversarial behaviors.

Beyond detection accuracy, the system was successfully deployed as a real-time monitor using a Python-based engine integrated with Scapy and joblib, running on low-resource devices such as OpenWRT routers or Raspberry Pi boards. During operation, the IDS maintained a low memory and CPU footprint, with an average inference latency below 200 milliseconds confirming its suitability for edge-based intrusion detection in constrained IoT environments.

Overall, this research demonstrates that effective MitM attack detection in MQTT networks requires not only robust machine learning algorithms but also visibility into application-layer behaviors. The proposed system offers a deployable, interpretable, and efficient solution for protecting MQTT-based IoT deployments against increasingly stealthy adversarial threats.

References

[1] A. Banks and R. Gupta, "MQTT Version 3.1.1," OASIS Standard, 2014. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[2] A. Mahmud, A. Iqbal, and F. Aadil, "Security Analysis of MQTT Protocol in IoT Applications," 2019 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), pp. 1–6, IEEE, 2019.

[3] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A Survey on the Security of IoT Frameworks," Journal of Information Security and Applications, vol. 38, pp. 8–27, 2018.

[4] M. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations," IEEE Communications Surveys & Tutorials, vol. 21, no. 3, pp. 2702–2733, 2019.

[5] M. R. Palattella et al., "Internet of Things in the 5G Era: Enablers, Architecture, and Business Models," IEEE Journal on Selected Areas in Communications, vol. 34, no. 3, pp. 510–527, 2016.

[6] A. Ferrag, L. Maglaras, A. Derhab, and H. Janicke, "Edge Intrusion Detection Systems for IoT-Based Smart Environments: A Systematic Literature Review," Electronics, vol. 9, no. 9, p. 1379, 2020.

[7] M. A. Khan et al., "A Deep Learning-Based Intrusion Detection System for MQTT Enabled IoT," Sensors, vol. 21, no. 21, p. 7016, 2021.

[8] S. Ullah et al., "TNN-IDS: Transformer Neural Network-Based Intrusion Detection System for MQTT-Enabled IoT Networks," Computer Networks, vol. 237, p. 110072, 2023.

[9] A. E. Al Hanif and M. Ilyas, "Enhancing MQTT Attack Detection through Ensemble Learning and Feature Engineering," arXiv preprint arXiv:2408.00480, 2024.

[10] G. Fortino et al., "An MQTT IoT Intrusion Detection System Using Deep Learning," in Proc. ICCCN, Springer, 2023.

[11] S. C. Prajisha and A. R. Vasudevan, "An Efficient Intrusion Detection System for MQTT-IoT Using Enhanced Chaotic Salp Swarm Algorithm and LightGBM," Int. J. Inf. Secur., vol. 21, pp. 1263–1282, 2022.

[12] M. A. Ali et al., "Intrusion Detection in IoT Networks Using ML and DL Approaches for MitM Attack Mitigation," Discover IoT, vol. 5, art. 48, 2025.

[13] A. Alshamrani et al., “Multiclass Classification for Detecting Attacks on MQTT-IoT Protocol,” arXiv preprint arXiv:2402.03270, 2024.

[14] S. Mujahid et al., “Characterization of Threats in IoT from an MQTT Protocol-Oriented Dataset,” Complex & Intelligent Systems, vol. 9, pp. 1501–1514, 2023.

[15] C. Nkemelu et al., “IoT-PRIDS: Packet Representation for Intrusion Detection in IoT,” Computers & Security, vol. 133, p. 103456, 2024.

[16] A. Shukla et al., “Preventing MQTT Vulnerabilities Using an IDS in IoT Environments,” Sensors, vol. 22, no. 2, p. 567, 2022.

[17] Y. Zhao et al., “IDS-MA: Centralized and Federated Learning-Based IDS for MQTT Attacks in IoT,” in Proc. IEEE COMPSAC, 2023.

[18] A. Mishra et al., “MediHunt: A Network Forensics Framework for Medical IoT Devices,” arXiv preprint arXiv:2312.04096, 2023.