Defending IoT Networks: Analyzing and Mitigating Attacks with pfSense

---

1. Introduction

With the increasing adoption of Internet of Things (IoT) devices across smart homes, industries, and critical infrastructure, securing the communication between these devices has become paramount. This project investigates how open-source network security tools — particularly pfSense with Snort/Suricata — can be used to analyze, detect, and mitigate protocol-level threats in IoT environments. The focus is placed on MQTT, a lightweight messaging protocol commonly used in constrained devices, to explore how attackers can exploit its simplicity, and how defenders can respond.

---

2. Technologies Used

2.1 pfSense

An open-source firewall and router platform, pfSense served as the core network gateway in the virtual lab. It was configured to:

- Segment the network into subnets for attacker and target VMs,
- Implement firewall rules, and
- Host IDS/IPS tools (Snort and Suricata) to monitor protocol-level activity.

2.2 Kali Linux

Kali was used to simulate adversarial behavior, including message flooding on the MQTT port (1883) and other unauthorized publish/subscribe attempts.

2.3 Ubuntu

Ubuntu hosted the **Mosquitto MQTT broker**, acting as a typical IoT endpoint. It allowed for the replay of benign and malicious traffic and supported traffic analysis in real time.

2.4 VirtualBox

Oracle VirtualBox enabled the creation of a fully isolated, reproducible virtual testbed consisting of pfSense, Kali, and Ubuntu VMs.

---

3. IoT Protocol Vulnerabilities and Focus on MQTT

IoT devices often suffer from weak authentication, limited firmware updates, and insecure default protocols. MQTT, while efficient and lightweight, lacks built-in security and can be exploited via:

- **Flooding**: Overwhelming the broker with publish requests,
- **Spoofing**: Faking client IDs to impersonate trusted devices,
- **Unauthorized access**: Publishing or subscribing to topics without proper control.

MQTT was chosen due to its real-world usage in IoT devices and its vulnerability profile, making it ideal for demonstration of threat simulation and defense testing in pfSense.

What is MQTT?

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol widely used in **Internet of Things (IoT)** applications. It was originally designed by IBM in 1999 for communication between devices in constrained networks with low bandwidth, high latency, or unreliable connections.

How MQTT Works

MQTT follows a **client-broker architecture**, where:

- **Clients** (e.g., sensors, smart devices, or mobile apps) publish messages or subscribe to topics.
- The **broker** (e.g., Mosquitto) is a central server that receives messages from publishers and distributes them to the appropriate subscribers.

✅ **Key Concepts:**

- **Topics**: Hierarchical strings like home/temperature used to categorize messages.
- **Publish**: Clients send data to a topic.
- **Subscribe**: Clients listen for messages from one or more topics.
- **QoS (Quality of Service)**:
    - 0: At most once (fire and forget)
    - 1: At least once (may duplicate)
    - 2: Exactly once (most reliable, most overhead)
- **Keep Alive**: A heartbeat signal to confirm a client is still connected.

Why MQTT is Popular in IoT

- **Lightweight header** (2 bytes) makes it ideal for constrained devices (e.g., ESP32, Raspberry Pi).
- **Asynchronous** communication fits event-driven IoT systems.
- **Low power use** suits battery-operated devices.
- Supports **TCP/IP** and, more recently, **MQTT over WebSockets**.

Security Challenges with MQTT

Despite its benefits, MQTT lacks built-in security, making it vulnerable unless layered with additional protections. Common vulnerabilities include:

❌ **No Built-in Authentication:**

- MQTT 3.1.1 supports **username/password**, but it's optional and often unused or hardcoded.

❌ **No Encryption by Default:**

- Messages are sent in **plaintext** unless wrapped in **TLS**.
- Attackers can easily **sniff sensitive data** or session credentials.

❌ **Topic Hijacking:**

- Topics are not access-controlled by default.
- Unauthorized clients can publish fake commands (e.g., unlock doors) or flood traffic.

❌ **DoS via Flooding:**

- Attackers can overwhelm a broker by **spamming thousands of publishes per second**.
- MQTT brokers like Mosquitto can crash or lag under attack.

❌ **Session Hijacking:**

- MQTT uses persistent sessions with **client IDs** — if IDs are reused, attackers can impersonate devices.

Why I Chose MQTT for This Project

MQTT represents a **realistic and widespread attack surface in IoT deployments**, particularly in:

- Smart home systems
- Industrial IoT (SCADA, sensors)
- Health monitors and wearables

Its **simplicity and widespread use** make it an ideal protocol to simulate:

- Attacks such as **DoS floods**
- Detection via **custom IDS rules**
- Real-time alerting and mitigation strategies

This aligns with the learning objectives of analyzing protocol behavior, understanding IoT vulnerabilities, and applying security monitoring techniques.

---

4. Experimental Design and Implementation

4.1 Network Topology

The virtual environment included:

- **pfSense** with three interfaces:
    - WAN (em0): for external internet simulation
    - LAN (em1): connected to the Ubuntu MQTT broker (192.168.1.0/24)
    - OPT1 (em2): connected to Kali (192.168.30.0/24)

This forced traffic between Kali and Ubuntu to route through pfSense, enabling traffic inspection.

✅ Setting Up Mosquitto MQTT Broker on Ubuntu

🔹 1. Install Mosquitto Broker and Client Tools

On your Ubuntu VM (MQTT server):

bash

CopyEdit

sudo apt update

sudo apt install mosquitto mosquitto-clients -y

This installs:

- mosquitto: the MQTT broker (server)
- mosquitto_pub and mosquitto_sub: MQTT command-line client tools

---

🔹 2. Enable and Start the Mosquitto Service

To make the broker start on boot and begin serving clients:

sudo systemctl enable mosquitto

sudo systemctl start mosquitto

To check the status:

sudo systemctl status mosquitto

You should see that it is **active (running)** on port **1883**, which is the default for unencrypted MQTT traffic.

✅ Testing MQTT Functionality on the Broker Itself (Ubuntu)

To verify that the broker is accepting and handling messages correctly, you can simulate **one terminal as a subscriber** and **another as a publisher**.

🔹 **3. Open Two Terminals on Ubuntu**

Terminal 1 (Subscriber):

Subscribe to the topic test:

mosquitto_sub -h localhost -t test

This command:

- Connects to the broker at localhost (127.0.0.1)
- Subscribes to the test topic
- Waits to print any messages received

Terminal 2 (Publisher):

Send a message to the topic test:

mosquitto_pub -h localhost -t test -m "hello from broker"

You should see "hello from broker" appear in Terminal 1 (subscriber window).

✅ **Testing MQTT from Kali (Simulated Attacker)**

From the Kali Linux VM, you can simulate external MQTT clients publishing or flooding the broker.

🔹 **4. Basic Publish Test from Kali**

mosquitto_pub -h 192.168.1.102 -t test -m "message from kali"

- -h: IP address of the Ubuntu MQTT broker
- -t: topic to publish to (e.g. test)
- -m: message payload

Ensure the subscriber is running on the broker side to see the message appear.

🔹 **5. Simulating MQTT Flood Attack**

To simulate a **DoS-style flood**:

for i in {1..100}; do mosquitto_pub -h 192.168.1.102 -t test -m "flood $i"; done

This sends 100 rapid messages to the test topic on the broker at 192.168.1.102.

You can increase the number or reduce delays to simulate heavier loads.

4.2 Attack Simulation

From Kali, various MQTT publish floods were launched against the broker using:

for i in {1..100}; do mosquitto_pub -h 192.168.1.102 -t test -m "flood $i"; done

4.3 IDS/IPS Configuration

Suricata was installed on pfSense and configured to monitor LAN traffic. A custom rule was added:

snort

alert tcp any any -> any 1883 (msg:"MQTT Flood Detected"; flow:to_server,established; detection_filter:track by_src, count 10, seconds 1; classtype:attempted-dos; sid:900001; rev:1;)

This rule triggered an alert when more than 10 publish packets from the same source were detected within 1 second.

---

5. Results and Observations

- **Traffic Verification**: Packet captures via pfSense confirmed MQTT traffic was correctly routed from Kali to Ubuntu.
- **Suricata Alerts**: Custom rules successfully triggered alerts in response to flood attempts, proving the IDS's ability to monitor MQTT traffic.
- **Logging and Rule Effectiveness**: Observed alerts matched the attack patterns, and no false positives were generated during benign MQTT traffic tests.

---

6. Reflection and Alignment with Unit Learning Outcomes

ULO1: The project demonstrated layered defense using pfSense and detailed analysis of MQTT protocol vulnerabilities — fulfilling network architecture and defense learning outcomes.

ULO2: Using Wireshark and Suricata logs, I differentiated between legitimate and malicious publish/subscribe activity, enabling accurate protocol behavior assessment.

ULO3: Log analysis, alert verification, and firewall rule reflection supported critical evaluation of IDS effectiveness, aligning with cybersecurity monitoring and review capabilities.

---

7. Future Work

To extend this project beyond the current scope, I plan to:

- **Export Suricata alerts** for correlation in external SIEMs
- **Create a complete Snort rule set** for MQTT, CoAP, and WebSockets
- **Integrate with Wazuh and Kibana** to build a real-time security dashboard with visual threat insights
- **Expand protocol simulation** to include malformed packets and authentication bypass scenarios

---

8. Conclusion

This project demonstrates how pfSense, coupled with Suricata, can effectively detect and monitor IoT-specific threats, particularly against MQTT. By simulating realistic attacks in a controlled lab and customizing IDS rules, defenders can build an affordable and adaptable security stack for IoT environments. This hands-on exploration also highlighted the strengths and limitations of signature-based detection in home and small business settings.