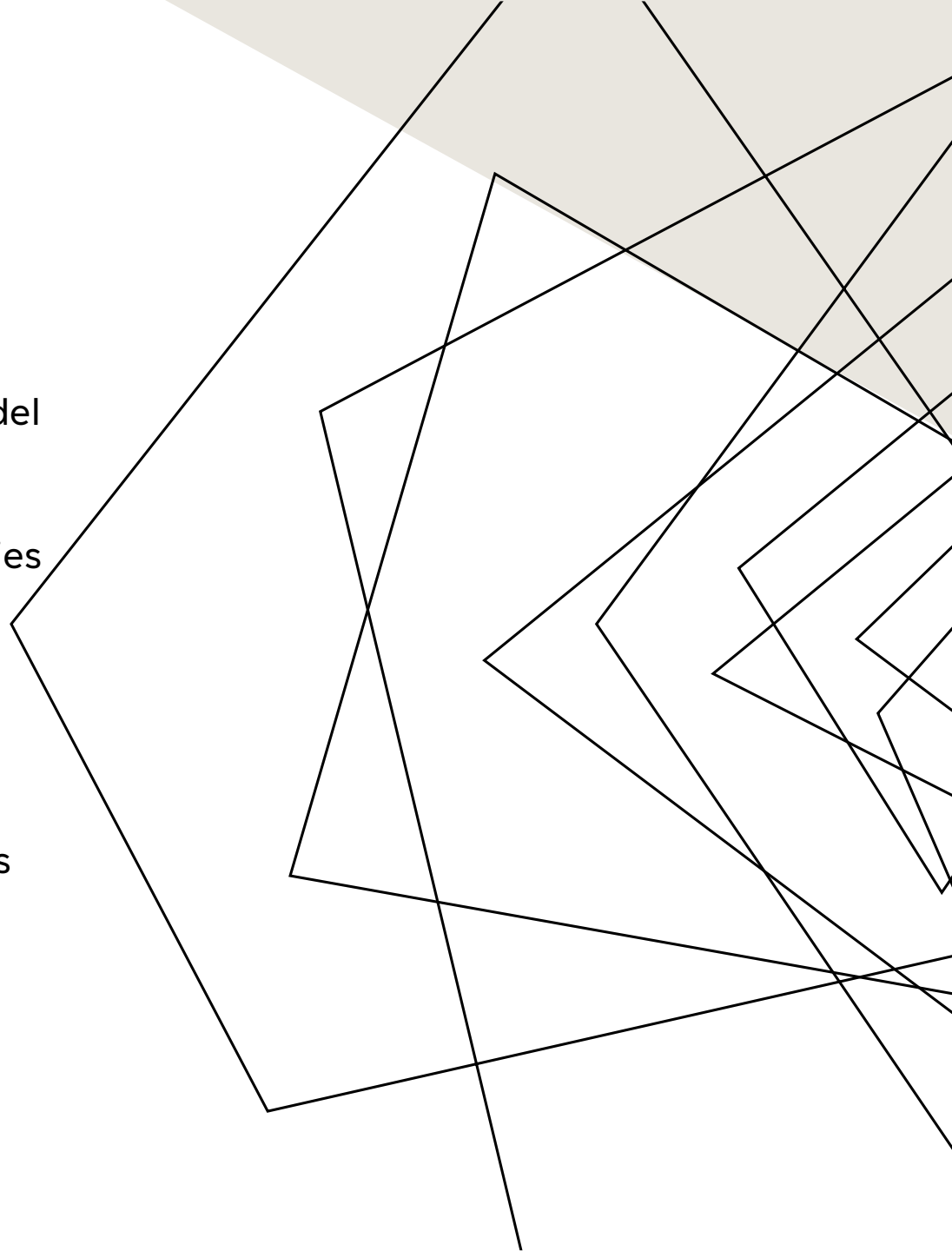
MQTT MAYHEM; PUBLISH & PWN

- Sanjay NS

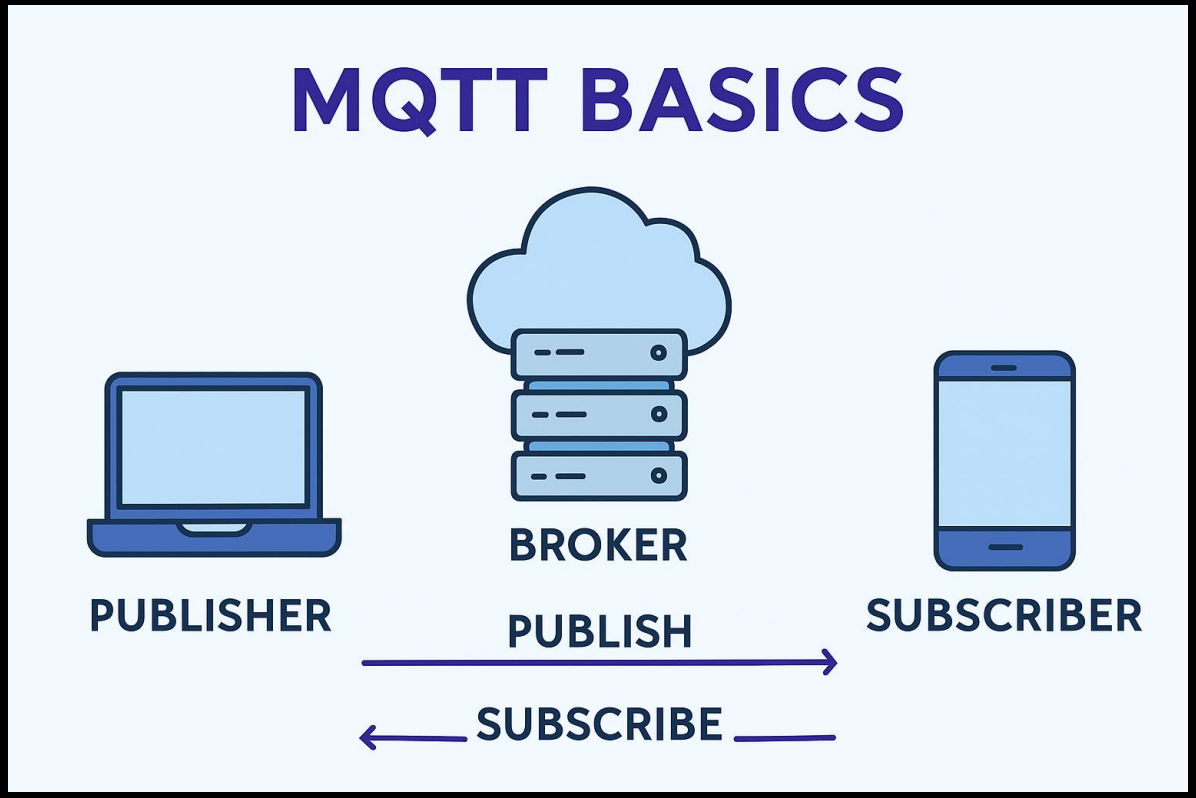
Researcher – IoTSRG.org

AGENDA

- Part 1 (MQTT Basics)
  - Understand MQTT protocol fundamentals and architecture
  - Explain the publish-subscribe communication model
  - Protocol Features & Benefits
  - Design effective topic hierarchies and QoS strategies
- Part 2 (MQTT Security)
  - Identify MQTT security challenges and solutions
  - Implement basic MQTT security measures
  - Apply best practices for secure MQTT deployments
- Q&A and Wrap Up



MQTT BASICS


MQTT BASICS
BROKER
PUBLISHER
SUBSCRIBER
PUBLISH
SUBSCRIBE

WHAT IS MQTT?

- **MQTT (Message Queuing Telemetry Transport)**
- **Lightweight messaging protocol** designed for IoT devices
- **Publish-Subscribe** communication pattern
- Created by **IBM in 1999** for satellite communication
- Now an **OASIS standard** (2014) and **ISO standard** (2016)
- Optimized for **low bandwidth, high latency networks**
- Perfect for **resource-constrained devices**
- *"The messaging protocol for the Internet of Things"*

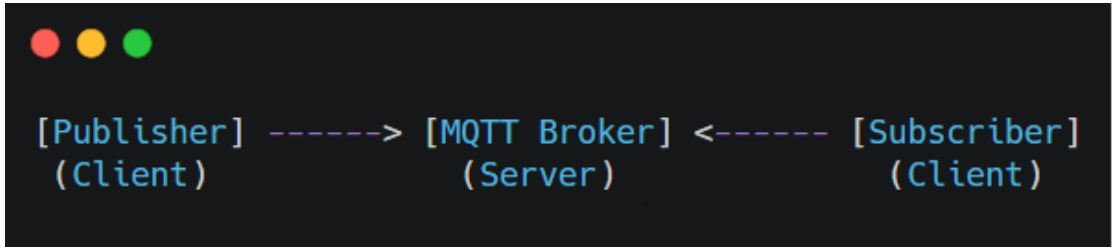
WHY MQTT?

- **Key Advantages**
- ✅ **Lightweight**: Minimal packet overhead (2-byte header minimum)
- ✅ **Efficient**: Low power consumption, ideal for battery devices
- ✅ **Reliable**: Quality of Service (QoS) levels ensure message delivery
- ✅ **Scalable**: Handles thousands of concurrent connections
- ✅ **Simple**: Easy to implement and understand
- ✅ **Flexible**: Works over TCP/IP, supports various network types

- **Perfect for**: IoT sensors, mobile apps, home automation, industrial monitoring

MQTT ARCHITECTURE OVERVIEW

Publish-Subscribe Model:



- **Key Components:**
- **MQTT Broker:** Central server that routes messages
- **MQTT Clients:** Publishers and/or Subscribers
- **Topics:** Message channels/categories
- **Messages:** Data payload being transmitted

- ***Decoupled communication - publishers don't know subscribers directly***

MQTT BROKER - THE HEART OF MQTT

MQTT Broker Responsibilities

- 🔄 **Message Routing**: Receives and distributes messages to subscribers
- 📝 **Client Management**: Handles client connections and sessions
- 🏷️ **Topic Management**: Organizes messages by topic hierarchy
- 💾 **Message Persistence**: Stores messages based on QoS levels
- 🔐 **Authentication**: Validates client credentials (optional)

Popular Brokers:

- Mosquitto (Eclipse)
- HiveMQ
- EMQ X
- AWS IoT Core
- Azure IoT Hub

MQTT TOPICS - MESSAGE ORGANIZATION

Topic Structure

Topics use hierarchical structure with / as separator:

home/livingroom/temperature

home/livingroom/humidity

home/bedroom/temperature

factory/machine1/status

factory/machine1/vibration

vehicle/truck001/location

Topic Rules:

- Case sensitive
- UTF-8 strings
- Can't start with $ (reserved for system topics)
- Maximum 65,535 characters

TOPIC WILDCARDS

Flexible Subscription Patterns

a) Single Level Wildcard (+)

```
home/+/temperature
```

Example Use Cases:
Monitor temperature in all rooms: home/+/temperature

Matches:

home/livingroom/temperature and home/bedroom/temperature

b) Multi Level Wildcard (#)

```
home/#
```

Example Use Cases:
Monitor all sensors in a building: building1/#

Matches:

home/livingroom/temperature, home/bedroom/humidity, home/kitchen/lights/status

QUALITY OF SERVICE (QOS) LEVELS

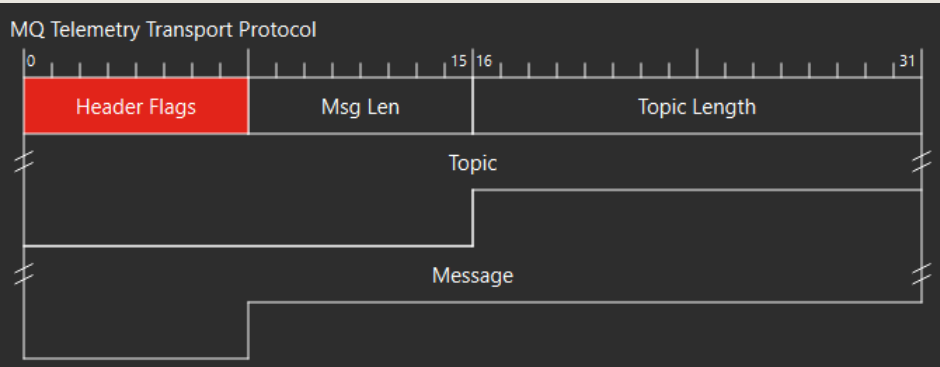
QoS Level	Name	Guarantee	Use Case
0	At most once	Fire and Forget	Sensor readings, frequent updates
1	At least once	Guaranteed delivery (duplicates possible)	Important notifications
2	Exactly once	Guaranteed single delivery	Critical commands, financial data

- **Trade-offs:**
- QoS 0: Fastest, lowest overhead
- QoS 1: Good balance of reliability and performance
- QoS 2: Highest reliability, most overhead

MQTT MESSAGE STRUCTURE

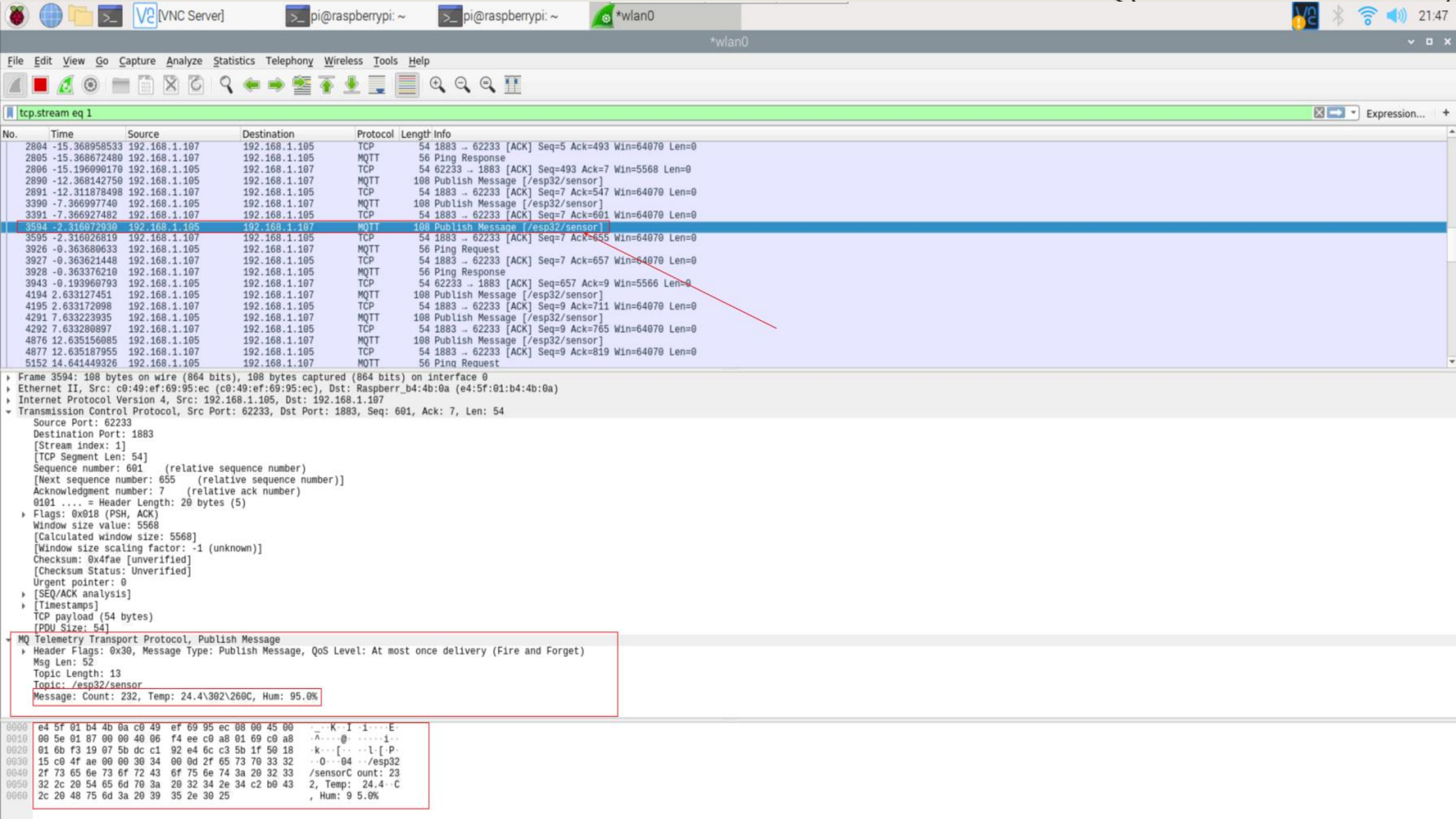
Message Components:

```
MQ Telemetry Transport Protocol, Publish Message
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    0011 .... = Message Type: Publish Message (3)
    .... 0... = DUP Flag: Not set
    .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
    .... ...0 = Retain: Not set
  Msg Len: 23
  Topic Length: 10
  Topic: test/topic
  Message: 68656c6c6f20776f726c64
```



Example Message:

- **Topic:** sensor/temperature
- **Payload:** {"value": 23.5, "unit": "Celsius", "timestamp": "2024-08-04T10:30:00Z"}
- **QoS:** 1
- **Retain:** false


[VNC Server]
pi@raspberrypi: ~
pi@raspberrypi: ~
*wlan0
21:47
*wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.stream eq 1
Expression...
No. Time Source Destination Protocol Length Info
2804 -15.368958533 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=5 Ack=493 Win=64070 Len=0
2805 -15.368672480 192.168.1.107 192.168.1.105 MQTT 56 Ping Response
2806 -15.196090170 192.168.1.105 192.168.1.107 TCP 54 62233 → 1883 [ACK] Seq=493 Ack=7 Win=5568 Len=0
2890 -12.368142750 192.168.1.105 192.168.1.107 MQTT 108 Publish Message [/esp32/sensor]
2891 -12.311878498 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=7 Ack=547 Win=64070 Len=0
3390 -7.366997740 192.168.1.105 192.168.1.107 MQTT 108 Publish Message [/esp32/sensor]
3391 -7.366927482 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=7 Ack=601 Win=64070 Len=0
3594 -2.316072930 192.168.1.105 192.168.1.107 MQTT 108 Publish Message [/esp32/sensor]
3595 -2.316026819 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=7 Ack=655 Win=64070 Len=0
3926 -0.363680633 192.168.1.105 192.168.1.107 MQTT 56 Ping Request
3927 -0.363621448 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=7 Ack=657 Win=64070 Len=0
3928 -0.363376210 192.168.1.107 192.168.1.105 MQTT 56 Ping Response
3943 -0.193960793 192.168.1.105 192.168.1.107 TCP 54 62233 → 1883 [ACK] Seq=657 Ack=9 Win=5566 Len=0
4194 2.633127451 192.168.1.105 192.168.1.107 MQTT 108 Publish Message [/esp32/sensor]
4195 2.633172098 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=9 Ack=711 Win=64070 Len=0
4291 7.633223935 192.168.1.105 192.168.1.107 MQTT 108 Publish Message [/esp32/sensor]
4292 7.633280897 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=9 Ack=765 Win=64070 Len=0
4876 12.635156085 192.168.1.105 192.168.1.107 MQTT 108 Publish Message [/esp32/sensor]
4877 12.635187955 192.168.1.107 192.168.1.105 TCP 54 1883 → 62233 [ACK] Seq=9 Ack=819 Win=64070 Len=0
5152 14.641449326 192.168.1.105 192.168.1.107 MQTT 56 Ping Request
Frame 3594: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface 0
Ethernet II, Src: c0:49:ef:69:95:ec (c0:49:ef:69:95:ec), Dst: Raspberr_b4:4b:0a (e4:5f:01:b4:4b:0a)
Internet Protocol Version 4, Src: 192.168.1.105, Dst: 192.168.1.107
Transmission Control Protocol, Src Port: 62233, Dst Port: 1883, Seq: 601, Ack: 7, Len: 54
Source Port: 62233
Destination Port: 1883
[Stream index: 1]
[TCP Segment Len: 54]
Sequence number: 601 (relative sequence number)
[Next sequence number: 655 (relative sequence number)]
Acknowledgment number: 7 (relative ack number)
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window size value: 5568
[Calculated window size: 5568]
[Window size scaling factor: -1 (unknown)]
Checksum: 0x4fae [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
[SEQ/ACK analysis]
[Timestamps]
TCP payload (54 bytes)
[PDU Size: 54]
MQ Telemetry Transport Protocol, Publish Message
Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
Msg Len: 52
Topic Length: 13
Topic: /esp32/sensor
Message: Count: 232, Temp: 24.4\302\260C, Hum: 95.0%
0000 e4 5f 01 b4 4b 0a c0 49 ef 69 95 ec 08 00 45 00 ·_··K··I ·i····E·
0010 00 5e 01 87 00 00 40 06 f4 ee c0 a8 01 69 c0 a8 ·^····@· ·····i··
0020 01 6b f3 19 07 5b dc c1 92 e4 6c c3 5b 1f 50 18 ·k···[·· ··l·[·P·
0030 15 c0 4f ae 00 00 30 34 00 0d 2f 65 73 70 33 32 ··O···04 ··/esp32
0040 2f 73 65 6e 73 6f 72 43 6f 75 6e 74 3a 20 32 33 /sensorC ount: 23
0050 32 2c 20 54 65 6d 70 3a 20 32 34 2e 34 c2 b0 43 2, Temp: 24.4··C
0060 2c 20 48 75 6d 3a 20 39 35 2e 30 25 , Hum: 9 5.0%

RETAINED MESSAGES

Persistent Topic State

What are Retained Messages?

- Messages stored by broker on specific topics
- New subscribers immediately receive the last retained message
- Only **one retained message per topic**

Example Scenario:

```
1. Device publishes: topic="device/status", payload="online", retain=true
2. Broker stores this message
3. New client subscribes to "device/status"
4. Client immediately receives "online" status
```

Use Cases: Device status, configuration settings, last known values

LAST WILL AND TESTAMENT (LWT)

Handling Unexpected Disconnections

LWT Message Setup:

- Configured during client connection
- Automatically published if client disconnects unexpectedly
- Helps detect device failures

Example Scenario:

Benefits: Automatic failure detection, system monitoring, graceful degradation

MQTT COMMUNICATION FLOW

Step-by-Step Process

1. **Connection:**

   Client ——CONNECT——▶ Broker

   Client ◀—CONNACK——— Broker

2. **Subscription:**

   Client ——SUBSCRIBE——▶ Broker (topic: "sensors/+")

   Client ◀—SUBACK———— Broker

MQTT COMMUNICATION FLOW

Step-by-Step Process

3. **Publishing:**

   Publisher ——PUBLISH——▶ Broker (topic: "sensors/temp", payload: "25°C")

   Broker ————PUBLISH——▶ Subscriber

4. **Disconnection:**

   Client ——DISCONNECT——▶ Broker

REAL-WORLD EXAMPLE - SMART HOME

MQTT in Home Automation

- **Scenario:** Smart thermostat system
- **Topics Structure:**

```
home/thermostat/temperature/set          « Target temperature
home/thermostat/temperature/actual       « Current temperature
home/thermostat/mode                     « heating/cooling/off
home/thermostat/status                   « online/offline
```

Flow:

- Mobile app publishes to *home/thermostat/temperature/set* → 22°C
- Thermostat subscribes to */set* topic, receives command
- Thermostat publishes current temp to */actual* topic
- Dashboard subscribes to all *home/thermostat/#* topics

REAL-WORLD EXAMPLE - INDUSTRIAL IOT

MQTT in Manufacturing

- **Scenario:** Factory Machine Monitoring
- **Architecture:**

[Machine Sensors] ⟶ [Edge Gateway] ⟶ [MQTT Broker] ⟶ [Control Room Dashboard]
|⟶ [Mobile Alerts]
|⟶ [Database Logger]

Topics:

```
factory/line1/machine001/temperature
factory/line1/machine001/vibration
factory/line1/machine001/status
factory/line1/machine001/alerts
```

Benefits: - Real-time monitoring
- predictive maintenance
- remote control

MQTT VS OTHER PROTOCOLS

Feature	MQTT	HTTP	WebSocket	CoAP
Model	Pub/Sub	Request/Response	Bidirectional	Request/Response
Overhead	Very Low	High	Medium	Very Low
Real-time	Excellent	Poor	Good	Good
Reliability	QoS Levels	TCP-based	TCP-based	UDP-based
IoT Suitability	Excellent	Poor	Good	Excellent

MQTT Sweet Spot: IoT, real-time messaging, resource-constrained devices

MQTT VERSIONS

Protocol Evolution

a) MQTT 3.1.1 (2014)

- OASIS standard
- Most widely supported
- Core features: QoS, retained messages, LWT

b) MQTT 5.0 (2019)

- Enhanced features
- Better error handling
- Message expiry
- User properties
- Shared subscriptions

Recommendation:
Use 3.1.1 for maximum compatibility,
5.0 for advanced features

GETTING STARTED - TOOLS & TESTING

Development Resources

MQTT Brokers (Free/Testing):

- test.mosquitto.org (public test broker)
- Eclipse Mosquitto (local installation)
- HiveMQ public broker

Client Tools:

- MQTT Explorer (GUI client)
- mosquitto_pub/sub (command line)
- MQTT.fx (testing tool)
- Mobile apps: MQTT Dashboard, IoT MQTT Panel

Quick Test:

```bash
# Subscribe
mosquitto_sub -h test.mosquitto.org -t "test/topic"

# Publish

mosquitto_pub -h test.mosquitto.org -t "test/topic" -m "Hello MQTT!"
```

MQTT CLIENT LIBRARIES

Implementation Support

Popular Libraries:

- Python: paho-mqtt
- JavaScript: mqtt.js
- Java: Eclipse Paho
- C/C++: libmosquitto
- Arduino: PubSubClient
- Android/iOS: Native MQTT SDKs

```python
import paho.mqtt.client as mqtt

def on_message(client, userdata, message) :
print(f"Received: {message.payload. decode()}")

client = mqtt. Client()
client.on_message = on_message
client.connect("test.mosquitto.org", 1883)
client.subscribe("sensors/temperature")
client. loop_forever()
```

MQTT USE CASES

Where MQTT Shines?

- 🏠 **Smart Home & Buildings**
  Lighting control, HVAC, security systems
- 🏭 **Industrial IoT**
  Machine monitoring, predictive maintenance
- 🚗 **Connected Vehicles**
  Telematics, fleet management
- 🌾 **Agriculture**
  Soil monitoring, irrigation control
- 📱 **Mobile Applications**
  Push notifications, real-time chat
- ⚡ **Energy Management**
  Smart grid, renewable energy monitoring

BEST PRACTICES

MQTT Implementation Tips:

✅ **Topic Design:**
- Use consistent naming conventions
- Keep hierarchies logical and shallow
- Avoid special characters

✅ **Message Design:**
- Keep payloads small and efficient
- Use JSON for structured data
- Include timestamps when needed

✅ **QoS Selection:**
- Use QoS 0 for frequent, non-critical data
- Use QoS 1 for important notifications
- Reserve QoS 2 for critical operations

✅ **Connection Management:**
- Implement proper reconnection logic
- Use appropriate keep-alive intervals
- Handle network interruptions gracefully

SUMMARY & NEXT STEPS

Key Takeaways

- 🎯 **MQTT is perfect for IoT** - lightweight, reliable, scalable
- 🔄 **Pub/Sub model** enables flexible, decoupled communication
- 🏷️ **Topics and wildcards** provide powerful message organization
- ⚖️ **QoS levels** balance reliability vs. performance
- 🛠️ **Rich ecosystem** of brokers, tools, and libraries

- **Coming Up Next:**

  🔒 **MQTT Security** - Authentication, encryption, authorization, security best practices

UP-NEXT:

MQTT SECURITY

Guess what, Demos included !!!

DEMOS

Lab Setup

- Attack Scenario 1 -> (Authentication)
- Attack Scenario 2 -> (Authorization)
- Attack Scenario 3 -> (Retained Message Abuse)
- Attack Scenario 4 -> (Wildcards Abuse)
- Attack Scenario 5 -> (Topic/Message/Client Flood)
- Case Study: CVE-2021-34432
  - Broker Crash Demo using FUME Tool (https://github.com/PBearson/FUME-Fuzzing-MQTT-Brokers)

MQTT SECURITY BEST PRACTICES

1. Authentication
    - Require username + strong password for all clients.
    - Avoid anonymous connections (allow_anonymous false in Mosquitto).
    - Use unique credentials per client/device (not shared accounts).
    - Store secrets securely on devices (not hardcoded in plain text if possible).
2. Authorization
    - Use Access Control Lists (ACLs) to restrict clients to specific topics.
        - Example: /esp32/sensor only for ESP32.
        - Example: /admin/cmd only for admin clients.
    - Enforce principle of least privilege (no # subscriptions for regular clients).
    - Separate publish and subscribe permissions.

MQTT SECURITY BEST PRACTICES

3. Confidentiality & Integrity (TLS/SSL)

- Always enable TLS (port 8883) to encrypt data in transit.
- Use trusted CA-signed or well-distributed self-signed certificates.
- Pin broker’s CA certificate in IoT devices (ESP32).
- Enforce certificate validation to prevent MITM attacks.
- Optionally require mutual TLS (client certificates) for high-security use cases.

4. Client Identity & Session Management

- Use unique client IDs for each device.
- Disable clean_session for critical devices (so they don’t lose subscriptions).
- Limit maximum connections per user to prevent abuse.
- Set appropriate keepalive intervals to detect dropped clients quickly.

MQTT SECURITY BEST PRACTICES

5. QoS & Reliability

- Choose QoS 1 or 2 only when necessary (to avoid flooding or replay).
- Don’t let untrusted users abuse high QoS levels.
- Limit retained messages to avoid attackers leaving malicious payloads behind.

6. Broker Hardening

- Disable unused listeners (only expose required ports: 8883 instead of 1883).
- Bind broker to internal network if devices don’t need internet exposure.
- Limit maximum message size to prevent memory exhaustion attacks.
- Limit max_inflight_messages and max_queued_messages to resist flooding.
- Enable logging and monitor unusual connection patterns.

MQTT SECURITY BEST PRACTICES

7. Protection Against DoS / Flooding

- Rate-limit publish frequency per client.
- Restrict wildcard subscriptions (# or +) for normal users.
- Use firewalls or reverse proxies (like NGINX or EMQX Gateway) to filter traffic.

8. Device Security

- Secure ESP32 firmware (avoid hardcoding secrets in plain).
- Use secure boot and flash encryption (ESP32 supports both).
- Regularly rotate credentials/certificates.
- Update firmware to patch vulnerabilities.

MQTT SECURITY BEST PRACTICES

9. Monitoring & Logging

- Enable Mosquitto’s detailed logs (log_type all).
- Monitor for failed logins, repeated publish attempts, topic floods.
- Integrate logs into a SIEM for anomaly detection.

10. Deployment Practices

- Isolate MQTT broker on a separate VLAN/segment (esp. in OT/ICS networks).
- Don’t expose MQTT directly to the internet — use a VPN or secure gateway.
- If remote access is needed, put MQTT behind an authenticated reverse proxy.
- Run broker as a non-root user to limit impact of compromise.

QUICK REMEDIATION MAPPING (FROM LABS/DEMOS)

- Anonymous access → Enforce auth + ACLs.
- Unauthorized publish/subscribe → ACLs.
- Topic flooding → Rate-limit + resource caps.
- Sniffing MITM → TLS/SSL with certs.
- Wildcards abuse → Restrict # usage.
- DoS risk → Limit inflight, queued messages, use monitoring.

THANK YOU

- Sanjay NS

Special Thanks to IoTSRG.org and Mr.IoT

SANJAY NS

Chapter Lead @NullBangalore |
Product Security Engineer @Honeywell | ...