Internet of Things Security

Lecture 4: Application Protocols and Security Challenges

Instructor: Mehmoona Jabeen
Email: mehmoona.jabeen@au.edu.pk
Department of Cyber Security, Air University
Course: CYS, Spring 2025

Lecture Outlines

- Introduction to Application Protocols
- COAP
- Security Challenges in COAP
- MQTT
- COAP vs MQTT and HTTP

Need of IoT Application Protocols

- To enable web-based services in constrained wireless networks in:
  - 8-bit micro-controllers
  - Limited memory
  - Low-power networks
- **Problem:**
  - Web solutions like HTTP are hardly applicable
- **Solution:**
  - Re-design web-based services for constrained networks
  - Use:
    - Request-Response Model
    - Publish-Subscribe Model

Application Protocols

1. Constrained Application Protocol (CoAP)

- Designed for low-power devices
- Paired with UDP for high efficiency

2. Message Queuing Telemetry Transport (MQTT)

- Ideal for remote environments or applications with limited bandwidth
- Uses a connection-oriented **publish/subscribe** architecture
- MQTT apps can **publish (transmit)** or **subscribe (receive)** topics

3. Advanced Message Queuing Protocol (AMQP)

- Open-source protocol for Message-Oriented Middleware (MOM)
- Supports communications between systems/devices/apps from multiple vendors
- Offers more routing options than MQTT
- More complex and has additional protocol overhead

4. Extensive Messaging and Presence Protocol (XMPP)

- Built on XML
- Initially designed for instant messaging (IM)
- Not optimized for memory-constrained devices
- Includes overhead for presence information exchange

Constrained Application Protocol (CoAP)

- IETF standard (RFC 7252)
- Suited for nodes with:
  - Simple microcontrollers
  - Limited ROM and RAM
- Works at the application layer
- Uses **UDP** as transport protocol

Features

- Simple discovery mechanism
- Easy integration with the Web
- Asynchronous message exchange
- Uses URIs to define resources/services
- REST-like request/response model

CoAP Features

- Web protocol for M2M in constrained environments
- UDP binding with optional reliability
- Supports unicast and multicast requests
- Asynchronous exchanges
- Low header overhead
- URI and Content-type support
- Simple proxy and caching capabilities

CoAP Messages

Four types of messages:

1. **Confirmable Message (CON):**
   Requires acknowledgment from receiver
2. **Non-Confirmable Message (NON):**
   No acknowledgment required
3. **Acknowledgment Message (ACK):**
   Acknowledges a Confirmable message
4. **Reset Message (RST):**
   Sent when message cannot be processed

CoAP Message Format

- Simple binary format
- Fixed-size 4-byte header
- Variable-length token (0 to 8 bytes)

CoAP Model

- Requests use Four request methods:
  - **GET, PUT, POST, DELETE**
- Responses use binary response codes:
  - 2.xx – Success
  - 4.xx – Client error
  - 5.xx – Server error
- 0–8 byte **Tokens** used to map requests/responses
- Responses to CON messages can be:
  - Piggy-backed in ACK
  - Sent as separate CON/NON

CoAP Reliable Messaging

- Mark message as **Confirmable (CON)**
- Receiver must:
  - Acknowledge with **ACK**, or
  - Reject with **RST**
- Sender retransmits at **increasing intervals** until ACK or reset received

---

CoAP Unreliable Transmission

- Mark message as **Non-confirmable (NON)**
- Always carries request or response (never Empty)
- Recipient does **not acknowledge**
- Can send **RST** if rejected
- Sender cannot detect if message was received
- NON messages still use a **Message ID**

Packet Loss Examples

Confirmable Transmission:

Client: CON [0x43A1] GET /light
Server: ACK [0x43A1] 2.05 /light 400lx

Non-Confirmable Transmission:

Client: NON [0x63A1] GET /light
Server: NON [0x63A1] 2.05 /light 400lx

CoAP Semantics

- Response matched using **client-generated token**
- Code field identifies **Response Code**
- Codes indicate:
  - Success
  - Client Error
  - Server Error
- Code numbers maintained in **CoAP Response Code Registry**

CoAP Response Code Classes

- **Success:** Request successfully received, understood, accepted
- **Client Error:** Bad syntax or cannot be fulfilled
- **Server Error:** Server failed to fulfill a valid request

Proxy and Caching

(CoAP supports proxy and caching mechanisms)

CoAP Observation

Problem with REST:

- REST is pull-based
- IoT often uses periodic or event-driven updates

Solution:

- **Observation extension (RFC 7641)**
- Client registers for state changes
- Server **pushes** updates without requests

CoAP vs HTTP

Feature	HTTP	CoAP
Type	Content-oriented	Network-oriented
Transport Protocol	TCP	UDP
REST Methods	Yes	Yes (GET, POST, PUT, DELETE)
Multicast	Not supported	Supported
Retransmission	Not defined	Defined mechanism
Port	80	5683 (default), 5684 (secure)

Headers

(Comparison of CoAP and HTTP headers—no content listed in original slide)

Security Challenges in CoAP

- CoAP lacks built-in **authentication** and **authorization**
- Security can be provided by:
    - **IPsec** or
    - **DTLS** or
    - **Object security**

Threats:

- **Path Traversal:**
    - Ignoring ".." in URI can lead to directory traversal
- **Cross-Protocol Attacks:**
    - CoAP's similarity with HTTP may allow similar attacks
- **Malicious Input Attacks:**
    - Fuzzing malformed requests can cause DoS
- **Unauthorized Access:**
    - Gaining read/write access to sensitive resources

Mitigation Strategies

- Use **DTLS** for secure transmission
- Avoid custom authentication/encryption
- Use all **8-byte tokens** for randomness
- **Filter ".." and "."** in URI-path
- Secure **keying material/certificates**
- **Input filtering**:
    - On server (device side)
    - On cloud (response payload)
- Implement proper **access control** and **auth mechanisms**
- Log all activity
- Alert user/cloud on suspicious requests
- Don't hardcode credentials in firmware

Transport Layer Security (TLS)

Services Provided by SSL Record Protocol

- **Client and Server Authentication:** Uses Public Key Infrastructure (PKI).
- **Confidentiality:** Handshake Protocol defines a shared secret key for encrypting SSL payloads.
- **Message Integrity:** Handshake Protocol also defines a shared secret key for Message Authentication Code (MAC).
- **Change Cipher Spec Protocol:** Updates the cipher suite by copying the pending state into the current state.
- **Alert Protocol:** Communicates SSL-related alerts to the peer entity.

TLS Architecture

Transport Layer (TCP/UDP)
↓
Network Layer (IP)
↓
TLS Record Protocol
↓
Handshake Protocol
↓
Change Cipher Spec Protocol
↓
Alert Protocol

Alert Messages (memorize any 2 for examples)

Alert Code	Alert Message	Description
0	close_notify	Sender will not send more messages.
10	unexpected_message	Fatal. Inappropriate message received.
20	bad_record_mac	Fatal. Record has incorrect MAC.
21	decryption_failed	Fatal. Decryption failed.
22	record_overflow	Fatal. Record exceeded allowed size.
30	decompression_failure	Fatal. Invalid input to decompression.
40	handshake_failure	Fatal. Incompatible security parameters.
42	bad_certificate	Problem with certificate integrity.
43	unsupported_certificate	Certificate type unsupported.
44	certificate_revoked	Received a revoked certificate.
45	certificate_expired	Received expired/invalid certificate.
46	certificate_unknown	Unspecified certificate processing error.

Handshake Protocol

Structure

- **Type (1 byte):** Identifies message type.
- **Length (3 bytes):** Message size.
- **Payload:** Parameters of the message.

Phases

1. **Establish Security Capabilities:** Cipher suite, compression, random numbers.
2. **Server Authentication and Key Exchange.**
3. **Client Authentication and Key Exchange.**
4. **Change Cipher Suite and Finish.**

Client Hello Message

- Highest SSL version supported.
- Client random: 32-bit timestamp + 28-byte pseudo-random.
- Session ID (empty for new sessions).
- Cipher suites list:
    - Examples: {0,0} to {0,10} (e.g., SSL_RSA_WITH_RC4_128_SHA)
- Compression methods supported.

Server Hello Message

- Server version number (compatible with client).
- Server random.
- Session ID.
- Cipher suite (from client's list).
- Compression method.

TLS Session State Parameters

- **Session Identifier:** Byte sequence for session resumption.
- **Peer Certificate:** X.509v3 certificate.
- **Compression Method**
- **Cipher Spec:** Encryption + MAC algorithm.
- **Master Secret:** 48-byte shared secret.
- **Resumable:** Indicates reusability of session.
- **Server/Client Random**
- **MAC Secrets:** For both server and client.
- **Write Keys:** Encryption keys for each direction.
- **Initialization Vectors:** For CBC mode.
- **Sequence Numbers:** Separate for sending and receiving.

Datagram Transport Layer Security (DTLS)

Why TLS Doesn’t Work Over UDP

- Packet loss and reordering in UDP.
- TLS lacks mechanisms to handle unreliability.
- DTLS introduces minimal changes to fix this.

Problems Solved

1. **TLS Record Dependence:** DTLS adds explicit sequence numbers and bans stream ciphers.
2. **Handshake Reliability:** DTLS adds retransmission timers and fragmentation.

Providing Reliability in DTLS Handshake

Handling Packet Loss

- Retransmission timers used by both client and server.
- No retransmission for HelloVerifyRequest.
- Alert messages are not retransmitted.

Handling Reordering

- Handshake messages have sequence numbers.
- Queued and processed in order upon receipt.

Handling Large Messages

- Handshake messages fragmented to fit datagrams.
- Each fragment includes offset and length.

Replay Detection

- Bitmap window tracks received records.
- Duplicates and too-old records are discarded.

DTLS Differences from TLS

1. The DTLS record layer is extremely similar to that of TLS 1.2. The only change is the inclusion of an explicit sequence number in the record. Main Changes:
2. **Stateless Cookie Exchange:**
    - Server sends HelloVerifyRequest with cookie.
    - Client resends ClientHello with cookie.
3. **Handshake Header Modifications:** Support for reordering and fragmentation.
4. **Retransmission Timers:** Handle handshake message loss.

DTLS Handshake Flights

- Messages grouped into "flights" for retransmission.

  Flight 1: Client_Hello

  Flight 2: HelloVerifyRequest

  Flight 3: Client_Hello (with cookie)

  Flight 4: Server messages (Server_Hello, Key_Exchange, etc.)

  Flight 5: Client messages (Certificate, Key_Exchange, etc.)

  Flight 6: Final messages (Change_Cipher_Spec, Finished)

MQTT (Message Queuing Telemetry Transport)

Overview

- Lightweight publish/subscribe protocol for M2M communication.
- ISO standard (ISO/IEC PRF 20922).
- Designed for low bandwidth, remote locations.

Use Cases

- Facebook Messenger uses MQTT to conserve battery.

MQTT Architecture

- **Broker:** Central server for managing topics.
- **Publisher:** Sends data to topic.
- **Subscriber:** Receives data from topic.
- **Topics:** Virtual channels for communication.

Client Actions

- Can publish and subscribe simultaneously.
- Open-source brokers: Mosquitto, RSMB, Micro broker.

MQTT Protocol and Header Format

Message Format

Fixed Header (always present) [2 bytes]
+ Variable Header (optional)
+ Payload (optional)

- **Fixed Header:** Includes Control field + Length.
- **Examples:**
    - CONNACK → Fixed Header
    - PUBACK → Fixed + Variable Header
    - CONNECT → All 3 parts

Packet Length

- Minimum: 1 byte (<127 bytes).
- Larger packets use 2–4 bytes (up to 256MB).
- 7-bit encoding + continuation bit.

MQTT Flags and Fields

Bit Position	Field	Description
3	DUP	Duplicate delivery
2-1	QoS	Quality of Service
0	RETAIN	Broker retains last message

MQTT Message Types

Mnemonic	Code	Description
CONNECT	1	Client request to connect
CONNACK	2	Connection acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish acknowledgment
PUBREC	5	Assured delivery part 1
PUBREL	6	Assured delivery part 2
PUBCOMP	7	Assured delivery part 3
SUBSCRIBE	8	Client subscribes
SUBACK	9	Acknowledgment for subscription
UNSUBSCRIBE	10	Client unsubscribes
UNSUBACK	11	Acknowledgment for unsubscription
PINGREQ	12	PING request
PINGRESP	13	PING response
DISCONNECT	14	Client is disconnecting

Iss mein se jo main lag ra wo yad krne bs 3-4
