[[1]](#footnote-2)

A brief study on MQTT protocol and its application as in maintenance of cereal storage atmosphere

**Akshita Mohite, *PVG’s COET*, Aniket Datar *PVG’s COET*, and Nishad Khare, PVG’s COET**

*Abstract*—Today the world is at the feet of IoT. Automation has spanned the whole digital and cyber market. Wherever there is internet, there arrives IoT. Though it is opulent for usage, its security is equally important. The devices working on the basic principle of IoT use numerous protocols as per their usage and scope. Given that IoT devices are complex, there is an ascertained probability of the device having an issue. Such is the MQTT protocol, which is used in most of the automotive IoT devices. Some devices which work on MQTT are sensitive, such as automatic guns. Here security is the concern. Thus Focusing on consanguine issues, we have tried to cover every context prerequisite/abstract apropos of Cyber Security in Internet of Things (IoT) with a short chronicle of vulnerabilities in the widely used MQTT protocol with its conjoint case study as in PACs (Programmable Automated Controller).

*Index Terms*—IoT, Security, Lua, MQTT, Broker, Client, Publish, Packet, Subscriber, etc.

# INTRODUCTION

The MQTT

Protocol is used widely in its own sector, IoT. There is a common misconception about the abbreviation of MQTT. The fact is that MQTT is not an acronym, but is MQ Telemetry Transport, where MQ stands for the family of message-oriented middleware products that IBM launched in 1993. Two employees from the same firm in 1999 with the help of MQ invented the MQTT protocol.

MQTT protocol works on the basic principle of the involvement of third-party services in the transmission of data between two clients. Subsequently the concern regarding data security arrives unless the protocol works in Secure Shell (SSH) or has a functioning over TLS. Today general automated devices use the AMQ Protocol, which is more advanced in data security and encrypted message transmission. We will dive deeper into securing MQTT further in this paper.

## Abbreviations and Acronyms

There are various abbreviations and acronyms used in our papers, some of them are TLS (Transmission Layer Security), PING (Packet Internet Groper), UDP (User Datagram Protocol), TCP (Transmission Control Protocol), PLC, etc.

## Other Recommendations

As far as recommendations are concerned in MQTT, there are various brokers (term will be introduced later) which provide service to various IoT devices. Some of them have their own documentations thereby clarifying the precise usage and control of MQTT in their modems.

Also, there are numerous books written on the same protocol management systems, which can be referred to for clarity about the mechanism and working of the components involved in the whole process.

## Scope

MQTT protocol was used in almost all IoT devices until the introduction of AMQP. Nevertheless, MQTT is still the most used protocol in IoT devices such as switches, routers, etc. This protocol is also used where data transfer in logical terms is required, for example a PLC is commanded to reduce the frame ratio on SCADA screen operations, it sends the log files to the central server using MQTT protocol.

(1)

It is an obvious fact that the larger the scope, the more are the chances of finding vulnerabilities. Same applies to MQTT. This protocol has numerous vulnerabilities in its structure, mechanism, algorithmic processing, etc. However, it is still used because these vulnerabilities are difficult to find, and even more difficult to assess. Also, these days layered securities make intrusions difficult. U

## Purpose

The basic purpose of selecting MQTT as our study topic is that the protocol is said to be secure by many, but it is not. As the functioning of the device depends solely on the algorithm used, these algorithms can be easily accessed and manipulated by just capturing the corresponding packets or JSON files or log files sent by the broker to the client/publisher/subscriber

Hence the need to investigate this from security perspective. As far as advancements take place in this field, numerous vulnerabilities will emerge, and more security analysis will be needed.

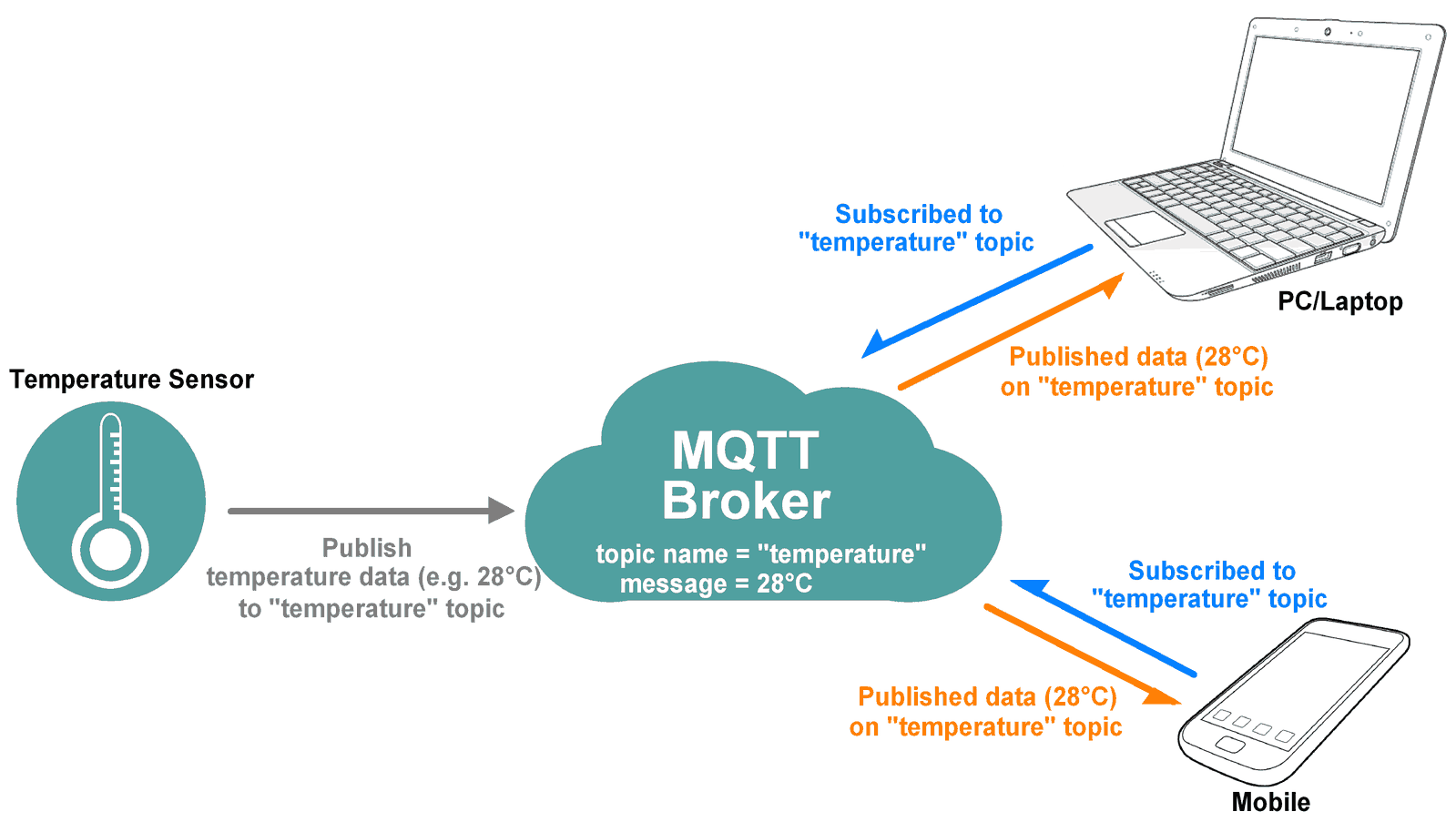


Fig. 1. MQTT fundamental mechanism. Here the broker collects subscription requests from one client and verifies it with the published data from the publisher client via CONNACK and CONNECT messages.

# Mechanism

## Getting connected and transmission

MQTT works on the basic principle of publish and subscribe.

The publish/subscribe pattern (also known as pub/sub) acts as an option to traditional client-server architecture. In the client-server model, a client communicates directly with the other side. The pub/sub model **decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers)**. MQTT broker has several preset assigned characters to define the exact module of communication in the whole mechanism.

The mechanism is similar to the familiar three-way handshake. The client sends a CONNECT message to the broker to initiate connection. The broker acknowledges it by replying with a CONNACK message. In this way the first stage of the authentication is complete. Everything happens over TCP/IP (5-6 layer) or a single back-to-back UDP interface. In many common use cases, the MQTT client is situated behind a router that uses network address translation (NAT) to translate from a private network address (like 192.168.x.x, 10.0.x.x) to a public facing address. As we already mentioned, the MQTT client initiates the connection by sending a CONNECT message to the broker. Because the broker has a public address and keeps the connection open to allow bidirectional sending and receiving of messages (after the initial CONNECT), there is no problem at all with clients that are located behind a NAT.

Here a noticeable fact is that the broker keeps its ports open while transmitting and receiving packets or PINGs from /to subscriber and publishers. The intruder can easily send fake packets to the broker with the same hash keys and can access the data easily. The broker can be deceived easily by any ways.

Most important is the network security in which the strength of the used SSH/TLS is analyzed. Local PINGs can be throttled to the central server to which the MQTT broker is connected. Here a basic contemplation is considered that until and unless the broker receives the SUBACK authentication message from the client, it stores the data. This verifies that the client side is secure. Also, the SUBACK or UNSUBACK message comes with a unique hashed value which is identical to that of the publisher’s packet.

In MQTT, the word topic means an UTF-8 string that the broker uses to separate messages for each connected client. The topic comprises of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

Example of topic:

myhome/groundfloor/livingroom/temperature

USA/California/San Francisco/Silicon Valley

5ff4a2ce-e485-40f4-826c-b1a5d81be9b6/status

Germany/Bavaria/car/2382340923453/latitude

TABLE I

Ports used by mqtt while connecting/

|  |  |  |
| --- | --- | --- |
| Port | Port significance | Security layer |
| 1883 | Used mostly for initiating MQTT via TCP | TCP/IP, over TLS /SSH |
| 8883 | Used only when request-response module is not needed | TLS disabled/ UDP oto enabled. |

## Vulnerability analysis

Following are the general and known methods which can be used to scan any MQTT broker for vulnerabilities and working towards it exploitation.

### *Browsing for open brokers:*

### Shodan is such website where huge amount of broker information is available. We can just search for any type of broker with any identification and its server shows up list of all MQTT brokers without passwords.

### *Using tools like MQTT.fx*

### There are various tools on the web which help to directly intrude the broker and retrieve the source-code over which the respective mechanism is based.

### *MQTTLens*

### It is a GUI/CLI software with which we can edit the source code obtained from the broker. It also acts as a listener.

### Lua Lua/Lisp is a prototyping platform where we can edit any type of source code and insert it again into the file. Used for JSON.

## Exploitation:

Exploitation of the source code is carried out accordingly:

.

A close up of text on a black background

Description automatically generated

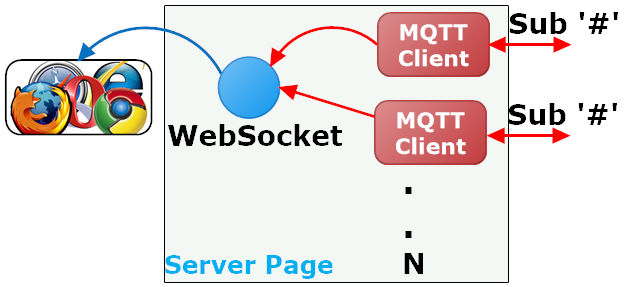
*Figure 1.3: There are various components of a single packet which can be seen. Any one of these can help intruders for getting the network addresses and information about the client/subscriber/broker*

# process

1. *LUA exploit*

The Mako Server is a web application server with integrated support for Lua. The server provides a mixed development environment where web developers can create traditional server pages and a socket API that enables developers to create advanced network applications. The server also comes with integrated support for MQTT and WebSocket.

The online Mako Server tutorials include several small Lua programs that you can execute and run on the server. However, the online server is not limited to running the Lua programs provided by the tutorials. Any program can run on the server. The online Lua web server is also open to abuse. Hence continuous development is the key feature here.



The number of MQTT clients created (depicted as N above) depends on how many broker IP addresses Shodan returns. Each MQTT client instance attempts to connect to the IP address provided. Connections to brokers that require authentication fail and are ignored.

| Return Code | Return Code Response |
| --- | --- |
| 0 | Success - Maximum QoS 0 |
| 1 | Success - Maximum QoS 1 |
| 2 | Success - Maximum QoS 2 |
| 128 | Failure |

.

*Figure: Quality of service (QoS) shows the guarantee of delivery for a particular message. 0 for at most once, 1 for at least once and 2 for exactly once.*

A service can host the MQTT exploit by executing the code below in the Lua editor.

**The crucial payload:**

local function startMQTT(ip, info) -- Create and connect one MQTT client

ba.socket.event(function()

local mqtt,err=require"mqttc".connect(ip, function(topic,msg)

onpub(string.format("%s: %s: %s",info,ip,topic), msg) end)

if mqtt then

table.insert(mqttT, mqtt)

mqtt:subscribe("#") -- Muahahaha

mqtt:run()

end

end)

end

if request:header"Sec-WebSocket-Key" then -- If a WebSocket request

ws = ba.socket.req2sock(request) -- Upgrade to a WebSocket connection

if ws then

-- Create an HTTP object and send an MQTT query to Shodan

local http = require"httpm".create{shark=mako.sharkclient()}

http:timeout(60\*1000) -- Shodan can be slow

local rsp,err = http:json(url, {key=key,query="mqtt"})

if rsp and rsp.matches then -- If JSON response OK

file = \_G.io.open(string.format("/tmp/mqtt%d.txt",ba.rnd()),"w")

ws:event(function() while ws:read() do end end, "s")

for k,v in ipairs(rsp.matches) do

startMQTT(v.ip\_str,v.org)

end

return -- OK

end

ws:write("Shodan response err: "..(err or "unknown"))

end

response:setheader("x-xss-protection","1; mode=block")

response:setheader("content-security-policy",

"default-src 'self'; connect-src http: https: ws: wss:; script-src 'self' 'unsafe-inline'")

The LSP page initially returns a web page to the browser with embedded JavaScript code. The returned web page implements a basic web console that is used for dumping all data received from the connected MQTT brokers. The JavaScript code is initiated as soon as the page loads and the code starts by establishing a WebSocket connection to the server. A Shodan MQTT broker search is initiated when the LSP page, executing at the server, receives the WebSocket upgrade request from the browser. Shodan returns a list of brokers in JSON format. The server code then iterates this list and creates an MQTT client for each MQTT broker in this list. The code logic then waits for MQTT data and sends all received MQTT data over the WebSocket connection to the browser.

Above source code is just an example of the core part of the payload used to exploit MQTT broker-client authenticating convention.

# LISTENER

| *Table 1. Available listener types* | |
| --- | --- |
| **Listener** | **Description** |
| [TCP Listener](https://www.hivemq.com/docs/hivemq/4.2/user-guide/listeners.html" \l "tcp-listener) | A listener for MQTT which uses TCP |
| [Secure TCP Listener](https://www.hivemq.com/docs/hivemq/4.2/user-guide/listeners.html" \l "tcp-tls-listener) | A secure listener for MQTT which uses TLS |
| [WebSocket Listener](https://www.hivemq.com/docs/hivemq/4.2/user-guide/listeners.html" \l "ws-listener) | A listener for MQTT over WebSockets |
| [Secure WebSocket Listener](https://www.hivemq.com/docs/hivemq/4.2/user-guide/listeners.html" \l "wss-listener) | A secure listener for MQTT over secure WebSockets (TLS) |

Many MQ brokers can be configured to run with multiple listeners. It’s e.g. possible to handle standard TCP connections on one port and secure connections on another port. This is completely transparent, and all clients can communicate among themselves via Publish/Subscribe regardless how they are connected to the broker. Clients can decide if they want to use secure or a "standard" TCP, non-TLS connection. This is extremely useful when some clients only have a unreliable network connectivity and/or very limited available bandwidth where every additional byte overhead matters and bandwidth efficiency is more important than a secure connection. In this case, these clients can connect to a MQTT broker via the unsecured port while other clients can use a secure connection via the TLS enabled port.

An example can be given as:

<brokername xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<listeners>

*<!-- Open to the rest-->*

<tcp-listener>

<port>1883</port>

<bind-address>0.0.0.0</bind-address>

<name>open-world-listener</name>

</tcp-listener>

*<!-- Only reachable for clients on the same machine -->*

<tcp-listener>

<port>1884</port>

<bind-address>127.0.0.1</bind-address>

<name>same-machine-listener</name>

</tcp-listener>

*<!-- Secure connection -->*

<tls-tcp-listener>

<port>8883</port>

<bind-address>0.0.0.0</bind-address>

<name>secure-listener</name>

<tls>

<keystore>

<path>/path/to/the/key/store.jks</path>

<password>password-keystore</password>

<private-key-password>password-key</private-key-password>

</keystore>

</tls>

</tls-tcp-listener>

</listeners>

...

</brokername>

*TABLE 3: TLS element options*

|  |  |  |  |
| --- | --- | --- | --- |
| protocols | All protocols enabled by Netty | no | *The enabled protocols* |
| cipher-suites | All cipher suites enabled by Netty | no | *The enabled cipher-suites* |
| client-authentication-mode | "NONE" | no | *The client authentication mode, possibilities are NONE, OPTIONAL (client certificate is used if presented), REQUIRED (client certificate is required)* |
| handshake-timeout | 10000 | no | *The SSL handshake timeout in milliseconds* |
| keystore.path | none | yes | *The path to the key store where your certificate and private key are included* |
| keystore.password | none | yes | *The password to open the key store* |
| keystore.private-key-password | none | no | *The password for the private key (if any)* |
| truststore.path | none | no | *The path for the trust store which includes trusted client certificates* |

# **CASE STUDY:**

***MQTT connection to remote PLCs.***

MQTT is the key to the system implementation because :

• MQTT is much faster and lower cost to setup than the older polled connection methods.

• The MQTT engine and transmission modules integrate very well with industrial controls eliminating the need to program or maintain highly specialized IT systems.

• Published tags from MQTT allow rapid commissioning of remote sites. The tags are automatically added to the SCADA system and then quickly linked to templates on the graphic displays.

• The low bandwidth of MQTT publishing data is critical due to it being cost effective for sites that may rely on cellular connections when compared to polling.

• Remote sites do not need static IP addresses making easier to implement and more secure with client-initiated communications.

Mechanism:

Ignition is the database-centric, web-based HMI/SCADA software package from Inductive Automation. Inductive Automation has created major buzz within the Industrial Industry over last decade changing the model of HMI/SCADA and leading innovation. As being a change agent, Inductive Automation saw the value in creating strategic alliance with Cirrus Link Solutions for its IIOT solutions ideally suited for Ignition platform enabling it to connect to MQTT architectures.

APQ Engineering identified the requirements for Smart Grain Solutions. We need a flexible, easy to integrate HMI/SCADA solution to acquire large amounts of I/O over cellular technologies to have rapid responsivity with security and control integrated into Ignition. To solve this, we can use the advanced MQTT Modules to enable an MQTT architecture. The solution is depicted below where the Cirrus Link MQTT Transmission Module and Ignition OPC Driver is installed in a Beckhoff PLC at the grain elevator or storage facility. It connects to a Cirrus Link Chariot MQTT Server over cellular or wired network where the Ignition platform subscribes to the PLC data as a client with the Cirrus Link MQTT Engine module installed.

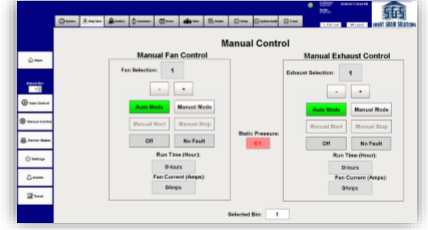


Figure 6: Humidity/moisture sensor initiating connection to Chariot MQ broker.

conclusion

To conclude, we have focused on the main MQTT analysis and its vulnerability scanning and exploitation and its respective application

We aim to create a system which resolves the common vulnerability issues in the devices based on MQTT/AMQP mechanism.

Our case study guarantees secure grain management solutions with constantly updating the moisture/temperature data from a grain storage to the MQTT broker, which in turn would help reduce common production losses as well as manufacturing mistakes.

References

1. Shodan web host (https://www.shodan.io)
2. IBM MQ (https://www.ibm.com/in-en/products/mq)
3. Lua/lisp platform (https://www.lua.org/docs.html)
4. Eclipse Mosq (https://mosquitto.org/man/mosquitto-8.html)

AUTHORS

1. AKSHITA MOHITE

F.E. COMPUTER SCIENCE,

PVG’S COET, PUNE, MAHARASHTRA

1. ANIKET DATAR

F.E. COMPUTER SCIENCE,

PVG’S COET, PUNE, MAHARASHTRA

1. NISHAD KHARE

F.E. COMPUTER SCIENCE,

PVG’S COET, PUNE, MAHARASHTRA

1.  [↑](#footnote-ref-2)