ESTUDAM Bilişim Derg, 2023; 4(2); 16-27.

J ESTUDAM Information, 2023; 4(2); 16-27.

DOI:10.53608/estudambilisim.1297052

(ReceivedDate: 14.05.2023, AcceptedDate: 05.06.2023)

(**Research Article**)

Attacks on Availability of IoT Middleware Protocols: A Case Study on MQTT

Muhammed Mustafa SIMSEK[1], Emrah ATILGAN[*2]

[1]Eskisehir Osmangazi University, Engineering and Architecture Faculty, Department of Computer Engineering, 26040, Eskisehir, ORCID No: http://orcid.org/0000-0002-2533-4934

[2]Eskisehir Osmangazi University, Engineering and Architecture Faculty, Department of Computer Engineering, 26040, Eskisehir, ORCID No: http://orcid.org/0000-0002-0395-9976

Keywords:
Internet of Things,
MQTT,
ESP8266,
MitM Attacks,
DoS Attacks,
Brute Force Attacks

Abstract: The Internet of Things (IoT) encompasses a technological ecosystem that improves the daily lives of individuals by increasing productivity, safety, comfort, health and sustainability. In addition, the IoT brings a variety of benefits to many industries, including increased efficiency, productivity and cost savings. However, the proliferation of IoT technologies has revealed many security vulnerabilities, especially in the middleware layer. In this article, we presented possible attacks on availability of middleware layer messaging protocols. In the research, a comprehensive case study was carried out, especially focusing on the MQTT (Message Queuing Telemetry Transport) protocol. We performed Man-in-the-Middle (MitM), Denial of Service (DoS) and Brute-Force attacks in our experimental environment. The effects and results of the attacks made in cases where the connection to the MQTT protocol is made with a user name and password, and when the user name and password are not used are examined. The results of the attacks that emerged in the different scenarios created were evaluated and the precautions to be taken to protect against the attacks were discussed.

(**Araştırma Makalesi**)

Nesnelerin İnternetinde Ara Yazılım Protokollerinin Hazır Bulunurluğuna Yapılan Saldırılar: MQTT Üzerine Bir Vaka Çalışması

Anahtar Kelimeler:
Nesnelerin İnterneti,
MQTT,
ESP8266,
Ortadaki Adam Saldırıları,
Hizmet Reddi Saldırıları,
Kaba Kuvvet Saldırıları

Özet: Nesnelerin İnterneti, üretkenliği, güvenliği, konforu, sağlığı ve sürdürülebilirliği artırarak bireylerin günlük yaşamlarını iyileştiren teknolojik bir ekosistemi kapsar. Buna ek olarak, Nesnelerin İnterneti birçok sektöre artan verimlilik, üretkenlik ve maliyet tasarrufu dahil olmak üzere çeşitli faydalar sağlar. Ancak Nesnelerin İnterneti teknolojilerinin yaygınlaşması, özellikle arayazılım katmanında birçok güvenlik açığını ortaya çıkarmıştır. Bu makalede, ara yazılım katmanı mesajlaşma protokollerinin kullanılabilirliğine yönelik olası saldırıları inceledik. Araştırmada özellikle MQTT protokolüne odaklanılmış ve kapsamlı bir vaka çalışması yapılmıştır. Deney ortamımızda Ortadaki Adam, Hizmet Reddi ve Kaba Kuvvet saldırılarını gerçekleştirdik. MQTT protokolüne bağlantının kullanıcı adı ve şifre ile yapıldığı durumlar ile kullanıcı adı ve şifre kullanılmadığı durumlarda yapılan saldırıların etkileri ve sonuçları incelenmiştir. Oluşturulan farklı senaryolarda ortaya çıkan saldırıların sonuçları değerlendirilerek, saldırılara karşı korunmak için alınması gereken önlemler tartışılmıştır.

1. INTRODUCTION

The term "Internet of Things" was first coined by Kevin Ashton, Executive Director of MIT's Auto-ID Labs, in 1999 while giving a presentation for Procter & Gamble [1]. The system that displays the coffee machine and shares these images over the Internet by academics at Cambridge University in 1991 is shown as the first application example of the Internet of Things technology [2]. The installed system transmitted the image of the coffee machine over the Internet three times a minute. Internet of Things applications have developed continuously since that day and started to be used in

*Corresponding Author: emrah.atılgan@ogu.edu.tr

different areas. Although the Internet of Things has brought many conveniences to people's lives, it has also revealed various security threats. The fact that these systems do not have traditional security mechanisms has caused attackers to target these systems.

There are many studies on the security of IoT ecosystems and messaging protocols in the literature. Hintaw et al. [3] examined security solutions for the MQTT protocol and presented the most ideal security measures that can be applied in addition to these solutions. In the study, it was suggested to use TLS for MQTT security, but it was emphasized that in some cases, IOT devices would be insufficient as hardware for TLS. Chen et al.[4] touched upon the most recent security issues faced by the MQTT protocol and what needs to be done in the future to deal with these issues. In this study, it is recommended to use complex machine learning algorithms for IOT security. Upadhyay et al. [5] conducted a study examining how IOT systems used in smart homes can make communication more secure with MQTT. In the study, the advantages of the MQTT protocol over other IoT protocols were emphasized. Assaig et al. [6] aims to secure the application layer connection between IoT devices and servers. In this study, they presented a lightweight security system for security. Firdous et al. [7] examined DOS attacks against MQTT protocol. In a study by Andy et al.[8], attack scenarios and security analyzes on IoT devices were examined. In this study, it is suggested that besides the use of TLS for security, new studies should be focused on for restricted devices.

In this article, security vulnerabilities of IoT messaging protocols were examined. Specifically, we focused on the MQTT (Message Queuing Telemetry Transport) protocol [9] because of its wide adoption, and the attacks and security measures on systems using this protocol were discussed. We aimed to provide solutions to the problems encountered in this field by examining the studies in the literature within the scope of the measures taken for cyber attacks against IoT devices.

Rest of the paper was organized as follows. In the 2nd section, IOT messaging protocols are briefly mentioned and a detailed literature review for MQTT is given. In the 3rd section, the methodology of the article and the tools used in practice are given. Evaluations about the findings and the application were discussed in the 4th section. In the 5th section, the conclusion and suggestions for future work are given.

2. IOT MESSAGING PROTOCOLS

IoT devices are specialized systems designed to perform specific tasks. Different IoT systems use different messaging protocols according to their own requirements and constraints. Each messaging protocol has its strengths and weaknesses, making it more suitable for certain use cases. For example, some IoT systems target low-latency real-time communication, while others prioritize low power consumption or high security. In some systems, because of limited processing power and memory usage, appropriate protocols should be selected. In addition, different IoT systems can be built using different hardware and software components, which can also affect the choice of messaging protocol. For example, some protocols may be optimized for use with certain types of sensors or devices, or for integration with certain cloud platforms or applications. Shortly, the choice of messaging protocol for an IoT system depends on a number of factors, including the specific use case, hardware and software requirements, and the goals of the system developers [10].

The most commonly used messaging protocols in IoT systems are MQTT, CoAP, AMQP, HTTP and DDS. In this study, we will focus on the MQTT protocol. On the other hand, brief information will be given about other protocols, and their strengths and weaknesses and which systems they work on will be briefly mentioned.

2.1 COAP (Constrained Application Protocol)

CoAP is a lightweight messaging protocol designed for constrained environments, such as low-power, low-memory devices [11]. It is based on the REST architectural style and can operate over UDP (User Datagram Protocol) or TCP (Transmission Control Protocol) [12]. CoAP uses the request/response communication model designed to work on IoT devices with constrained hardware and generally limited bandwidth. Each IoT object running CoAP also acts as a CoAP server. CoAP clients send request packets with GET, POST commands to access the service on the server. The server also responds to incoming messages by sending response packets.

Pros:
- Specially designed for use in low bandwidth and power IoT systems
- Easy to integrate with Web services as it uses REST architecture
- Supports multiple transport protocols including UDP and TCP

Cons:
- Limited support for service quality options
- Low message reliability
- May require the implementation of additional security measures

CoAP protocol is used in smart home systems to control and monitor household appliances, and to communicate in lighting and HVAC (Heating, Ventilation, and Air Conditioning) systems [13]. It is also a preferred protocol in smart agriculture, systems that require real-time and precise tracking, and IoT systems used for instant monitoring of soil, weather and crop growth [14]. Moreover, CoAP is used for real-time inventory tracking and supply chain management in retail systems [15]. Finally, there are Smart City IoT systems where CoAP is preferred for real-time monitoring and control of traffic, parking and other infrastructure [16].

2.2 AMQP(Advanced Message Queuing Protocol)

AMQP was developed by John O'Hara in 2003 to meet the need for high durability, high volume and high degree of interoperability [17]. The improved version AMQP 1.0 was standardized by OASIS (Organization for the Advancement of Structured Information Standards) in August 2011 [18].

AMQP is a reliable and feature-rich messaging protocol that can handle a wide variety of use cases, enabling both simple point-to-point messaging and communication across complex distributed systems. Although it may seem like an advantage to have many features if needed, it includes many features and capabilities that may not be necessary or relevant for all applications. Therefore, AMQP is not generally considered a lightweight protocol, although its performance and resource usage can be optimized in certain configurations. For example, AMQP can be configured to use a small number of simple message brokers and can be optimized for low-latency messaging in high-performance applications.

AMQP is a messaging protocol that supports both publish/subscribe and request/respond communication models. It is designed to work together across different platforms and programming languages [19]. AMQP is designed to operate over TCP, a transport layer that provides features such as reliable message delivery, congestion control, and flow control [20]. When AMQP is used over TCP, a connection is established between the AMQP client and server, and messages are exchanged over that connection. The connection is managed by the AMQP protocol, which handles tasks such as establishing and terminating the connection, negotiating parameters, and ensuring reliable delivery of messages [21].

Pros:
- Supports both publish/subscribe and request/respond communication models
- Designed to be interoperable across different platforms
- Provides strong authentication and encryption features which ensures message secure transfer
- Provides message reliability by message acknowledgement, persistence, and transactional support.

Cons:
- More complex than some other messaging protocols, which may make it harder to implement and maintain
- May require additional resources to operate efficiently
- May not be well suited for mobile devices

As a security-focused messaging protocol, AMQP is mostly used in healthcare applications for patient monitoring and data exchange between medical devices [22]. In addition, in Financial services, AMQP is used for real-time data exchange, transaction processing, and messaging between financial institutions, traders, and market data providers [23]. Besides, it is used to provide instant data communication in various sectors such as Logistics and Supply Chain Management, Gaming, Energy Management [24].

2.3 HTTP (HyperText Transfer Protocol)

Although HTTP is not a protocol designed for IoT, it is a widely used protocol for data communication, especially between web servers and clients. HTTP uses a request/response model [25]. In this model, the client (such as a web browser) sends a request to the server for a resource (such as a webpage), and the server responds with the requested resource, along with any necessary status codes or headers. This protocol is not preferred in constrained-IoT devices as it sends many packets during the communication, which causes resource usage delays and traffic overload [26].

On the other hand, HTTP is a protocol used on the TCP/IP stack, TCP provides a reliable communication channel between client and server that HTTP can use to send and receive data. When a client sends an HTTP request to a server, a TCP connection is established with the server and the request is sent over that connection. The server responses over the same connection. This TCP connection ensures reliable and orderly transmission of data, even when sent in multiple packets [27].

Pros:
- Easy to integrate with web-based services since it is widely used and familiar
- Supports a variety of data formats, including JSON and XML
- For additional security, HTTP can be operated over SSL/TLS, which provides secure communication between devices
- Supports caching and other performance optimizations

Cons:
- HTTP has a high overhead which leads to increased latency and reduced performance, especially in low-power devices
- Lack of real-time communication capabilities

2.4 DDS (Data Distribution Service)

DDS is a standard that was developed by the Object Management Group (OMG) to address the need for a messaging protocol that could provide efficient and reliable data communication in distributed systems [28].

DDS utilizes a publish-subscribe messaging model to enable multiple subscribers to access data published by one or more publishers. This model is designed to separate publishers and subscribers from each other, which means that they don't need to be aware of each other's identity or location. Essentially, publishers publish data to topics, which subscribers can access based on their interest in those topics. DDS utilizes a system based on topics to facilitate communication between publishers and subscribers. Topics are named objects that signify a particular data stream, and publishers and subscribers can indicate their interest in specific topics. Additionally,

DDS offers various Quality of Service (QoS) policies that permit publishers and subscribers to define the preferred method of data transfer, including aspects like reliability, response time, and throughput.

DDS is used as the messaging protocol in variety of IoT systems such as Industrial automation systems, healthcare, smart grid systems, autonomous vehicles and unmanned aerial vehicles (UAVs) [29].

Pros:

- DDS is designed to provide high performance, with low latency and high throughput, making it well-suited for real-time applications in IoT systems.
- DDS is a reliable messaging protocol equipped with built-in mechanisms for data integrity control, flow control and error correction.
- DDS is a messaging protocol that provides a flexible and powerful Quality of Service (QoS) that can be tailored to the specific needs of an IoT system.
- DDS is highly scalable, capable of handling large volumes of data, and also supports communication between many different devices and components.

Cons:

- DDS requires special knowledge and expertise to configure and manage, which can be a hurdle for end users.
- DDS is not an open-source protocol and can be expensive for some IoT systems.
- DDS may require more network bandwidth and processing power compared to some other messaging protocols, resulting in a higher overhead. As a consequence, it may be necessary to use more expensive network infrastructure or hardware to support the DDS system.

2.5 MQTT (Message Queuing Telemetry Transport)

MQTT is a lightweight messaging protocol developed by Andy Stanford-Clark and Arlen Nipper from IBM in 1999 and is the most widely used in IoT systems [20]. This protocol first emerged with the need for a protocol with minimum bandwidth and minimum battery consumption to observe oil pipelines from satellite.

MQTT v3.1.1, the improved and standardized version of MQTT, was released in 2014 by the OASIS technical committee [30]. The latest enhanced version of the protocol is MQTT v5.0, which includes improvements for scalability and large-scale systems, and was released in 2019 [9]. Originally designed for lightweight IoT devices, the MQTT protocol is also used by major network service providers such as Facebook Messenger and Amazon. In addition, the availability of MQTT protocol libraries written in popular programming languages such as Arduino, Javascript, Python increases the popularity of MQTT and provides ease of use.

MQTT, which is designed to be used in constrained-devices, is also used effectively in machine-to-machine communication (M2M) between devices located at long distances [31]. Unlike the request-response structure of the HTTP protocol, it has been developed in publish-subscribe model [32].

The MQTT protocol has the publisher as the source of the data. The publisher undertakes the task of obtaining the data in the IoT systems and transmitting it to the subscribers. In this protocol, the client that wants to receive data from a particular source is called a subscriber. The intermediary server that provides the connection between the publisher and the subscriber in the system is called a broker. Specifically, an MQTT broker receives messages published by clients, filters the messages by topic, and distributes them to subscribers who subscribe to that topic [33] (Figure 1).

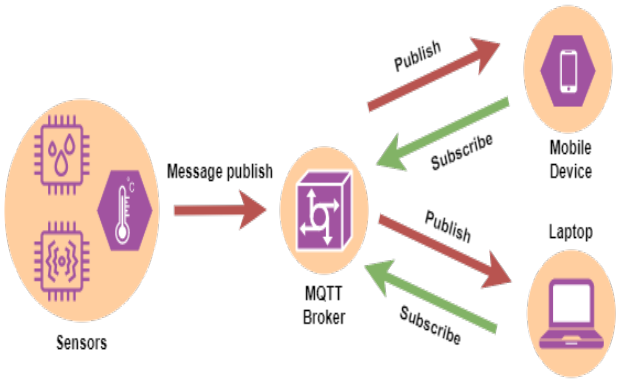


Figure 1. An example of MQTT Protocol Structure

A system must have at least one of each of the Broker, Publisher, and Subscriber to be able to perform messaging with the MQTT protocol. However, systems usually have more than one publisher and subscriber. In order for the message sent by the publisher to be transmitted to the subscriber via the broker, the subscriber must be connected to the broker. The subscriber is connected to the broker by socket. For TCP sockets, the subscriber and publisher use port 1883 by default on devices that work with the MQTT protocol. This port is used for unencrypted connections, which is the default feature of the MQTT protocol. For encrypted connections, port 8883 is used [34].

2.5.1 MQTT QoS Levels:

In the MQTT protocol, the control of whether the data reaches the receiver is made with three different quality of service (QoS) levels [35]. The QoS level is defined as the reliability and integrity of the communication between the publisher and the subscriber. Three different services are provided for this two-step process, which consists of sending the message from the publisher to the broker and transferring it from the broker to the client.

QoS_0: The publisher sends the message to the broker at most once (Figure 2). The sent message may be lost as a result of disconnection and may not reach the subscriber. QoS_0 does not control whether the message reaches the subscriber. For this reason, it is known as the most unsafe level of service quality. The message is not stored on the publisher and broker. The message is deleted after it is

sent. Another feature of QoS_0 is the service quality level with the lowest traffic.

Figure 2. QoS Level 0

QoS_1: The publisher sends the message to the broker at least once (Figure 3). The publisher keeps a copy of the message it sends until it receives acknowledgment of receipt, and transmits it more than once. The publisher deletes the message when it receives this confirmation. If the publisher does not receive acknowledgment of receipt for a certain period of time, the message is sent again. This cycle continues until the publisher receives acknowledgment. In this case, duplicate messages may occur. Messages are stored on the publisher and broker. Therefore, no message is lost.

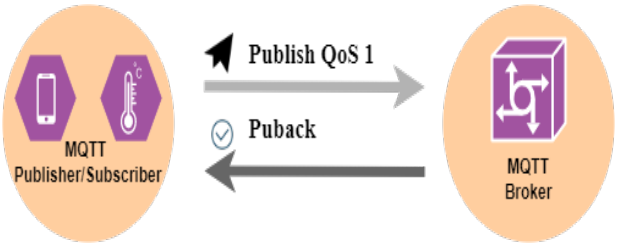


Figure 3. QoS Level 1

QoS_2: The publisher sends the message to the broker strictly once (Figure 4). Therefore, duplicate messages do not occur. Messages are stored on the publisher and broker, so no messages are lost. At this level of service quality, a kind of handshake takes place to confirm that the message has been sent and acknowledgment received. This handshake uses four packets transmitted in a specific order. After the handshake is complete, the publisher and the broker ensure that the message is sent exactly once. For this reason, it is accepted as the safest level of service quality. QoS_2 is the level of service quality with the most traffic.

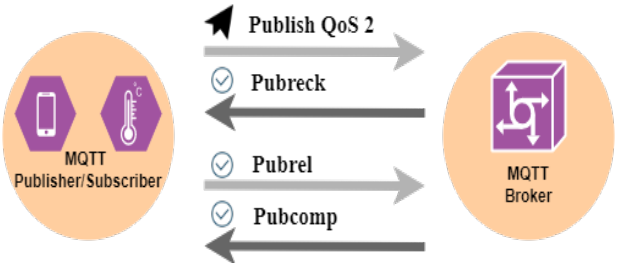


Figure 4. QoS Level 2

2.5.2 MQTT Brokers

The MQTT broker is an essential messaging server that serves as the central hub for message exchange among various MQTT clients, such as IoT devices, sensors, and actuators. Its main function is to enable reliable and efficient communication between these clients.

The broker uses topics to route messages, with each topic representing a specific data stream or subject of interest. It also maintains a message queue, allowing clients to publish messages even if the intended recipient is not available to receive them. This ensures that data is not lost and can be delivered as soon as the recipient is available. MQTT Broker is also responsible for managing the QoS of messages, ensuring that each message is delivered with the desired level of reliability. The QoS can be configured by the publisher, subscriber, or broker, depending on the requirements of the IoT system. Even though it is not default, MQTT broker provides security features, such as authentication and encryption, to protect the messages and ensure that only authorized clients can access them.

MQTT brokers can generally be categorized into two main classes: those designed for cloud systems and those designed for local systems. Cloud-based MQTT brokers are generally hosted in the cloud, and they can be accessed from any location that has an internet connection. Their primary function is to handle large volumes of traffic and can be easily scaled up or down based on the needs of the application. Several examples of cloud-based MQTT brokers include AWS IoT, Microsoft Azure IoT Hub, and Google Cloud IoT. In contrast, local MQTT brokers are designed to be deployed on-premises, and they facilitate local communication between IoT devices within a network. They are mainly used in situations where there is limited or no internet connectivity or when data needs to be kept within a private network. Some examples of local MQTT brokers are Mosquitto, HiveMQ, EMQX, RabbitMQ, Apache ActiveMQ [36].

2.5.3 MQTT Security

Although the MQTT protocol is a widely preferred protocol, it is the user's responsibility to solve the security problems in the MQTT protocol [37]. Many of the MQTT brokers do not require a password by default. This creates an essential vulnerability where anyone can subscribe and see the published data. This can cause hackers to spy on and even control devices in IoT systems. If a user wishes, they can authenticate through the connection package. However, sending your credentials in clear text, just like telnet, is a vulnerability. If an attacker eavesdrops on network traffic, they can see the username and password used in plain text.

Authentication is not a default feature in MQTT because the protocol was designed to be a lightweight and simple messaging protocol that minimizes the amount of data transmitted over the network. This means that MQTT was not intended to handle complex security mechanisms by default, as it could increase the overhead and complexity of the protocol [38].

For added security, some MQTT agents use SSL encryption mechanism when transmitting data. Authentication and Authorization can be done between

clients and the agent using SSL certificates and passwords. The MQTT agent authenticates subscribers using passwords as well as unique subscriber identifiers, which it typically assigns to each subscriber. In some applications, the subscriber verifies the publisher with DNS lookups. However, it may not be appropriate to use SSL for IoT systems with insufficient hardware. Instead, it has been suggested to use AES (Advanced Encryption Standard) to secure data [39]. In addition, some lightweight encryption mechanisms have also been developed to ensure MQTT messaging security in IoT systems consisting of devices with limited processing power and battery life [40][41].

In systems where security is vital, MQTTS (MQTT Secure) may be preferred over the MQTT. MQTTS is a modified version of the typical MQTT protocol that applies TLS (Transport Layer Security) encryption. TLS is more advanced than SSL and offers enhanced security features. The MQTTS requires clients to create a secure connection with the broker by exchanging certificates to guarantee the authenticity of both the server and client. This procedure helps to prevent unauthorized access and sustain data confidentiality in IoT systems.

Another problem is that there are no access restrictions in the MQTT protocol that prevent a client from subscribing or broadcasting to the topics they want. This may pose a problem in terms of data security. As a solution to this problem, it is suggested to use Access Control List (ACL) [5].

3. METHOD

In this study, an experimental environment consisting of agents, clients and sensors using the MQTT messaging protocol was established and the security of the MQTT protocol was tested. For different scenarios, MitM, DoS and Brute Force attacks were performed and the results were shared in detail. Appropriate measures have been proposed for security vulnerabilities in systems using MQTT protocol. In the experimental environment, the ESP8266 development board and a potentiometer were used as a sensor because it has the ability to generate and send instant data. Network traffic was carried out over WiFi.

3.1 Hardware

Esp8266 NodeMCU: ESP8266 is a high-performance, small-size and low-power IOT module developed by EspressifSystems that can enable Wi-Fi connectivity. ESP8266 is a module that can provide internet access in IoT systems, send and receive data, and control the system over the internet. It offers I/O units, PWM outputs and communication support.

3.2 Software

Tcpdump [42], a packet analysis program that runs on the command line in Linux operating systems, was used to capture and inspect TCP/IP packets or other packets transmitted or received over the network. It supports supporting many protocol format including MQTT. Wireshark [43] is an open source network monitoring tool widely used by network professionals for debugging network problems. It utilizes Tcpdump library and provide a user friendly graphical interface. Kali Linux [44], a Debian GNU/Linux based Linux distribution used for security purposes, was also utilized in this study for advanced penetration tests. Nmap [45], an open source tool for network scanning, was used to identify which computers on the network are using which services and applications.

3.3. Attack Tools

In this study, LOIC (Low Orbit Ion Cannon), Hping3 were used for DDoS attacks. Ettercap tool was used for MitM attack. Metasploit Framework is used for brute force attack.

LOIC (Low Orbit IonCannon): It is an open source Denial of Service (DoS) attack tool written in C#. LOIC is used to interrupt the service of a particular device. It can perform a DoS or DDoS attack on a target device by filling the target server with TCP, UDP or HTTP etc packets [46].

Hping3: Hping3 is an open source application for TCP/IP packet processing and analysis that can be used on the command line that comes ready on Kali Linux [47]. The difference of this tool from the ping command is that it not only sends ICMP echo requests, but also supports protocols such as TCP, UDP. It is a widely used tool for penetration testing of firewalls and networks, DoS and DDoS attacks.

Ettercap: It is an advanced tool for MitM attacks, eavesdropping and DoS attacks on local networks. In this study, it is used to perform a MitM attack between the publisher and the broker server in the local network. Ettercap is a tool that enables MitM attacks by allowing the target machine to use its MAC address to receive incoming packets on that machine.

Metasploit Framework: Metasploit is an open source penetration testing tool developed in the Ruby language [48]. It is a platform developed by Rapid7 company, where users can run exploits, and also has many network discovery and attack tools on it. This tool, which reveals the security vulnerabilities of services or applications running on the systems, comes preinstalled on Kali Linux. In this study, a brute force attack was carried out with the MQTT exploit tool on the Metasploit Framework.

4. TEST SCENARIOS and EXPERIMENTAL SETUP

4.1 Attack Scenarios

Man in the Middle, Denial of Service and Brute Force attacks were carried out in the test environment designed within the scope of this study.

4.1.1 Man in the Middle Attack

The attacker, who aims to get between two communicating devices and pass the network traffic over himself, can see and change the data in the communication and send the fake data to the devices if it is successful.

In the test environment established in our study, the attacker tried to capture the sensor data and MQTT session information by interfering between the IoT device and the subscriber, as shown in Figure 5. Man-in-the-middle attacks are handled in two different situations where username and password are used and not used in communication with MQTT protocol.

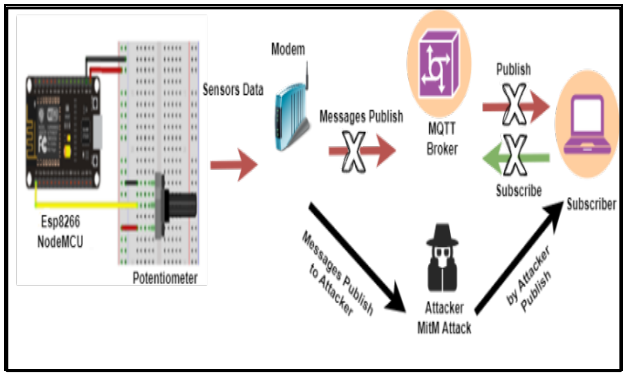


Figure 5. Man in the Middle Attack Scenario

4.1.2 Denial of Service (DoS)

DoS attacks are a type of attack that tries to weaken the connection between two devices in communication or to make users who use the system inaccessible to the system [49]. The purpose of these attacks is to damage the system by making the target unreachable.

In the test environment established in our study, as shown in Figure 6, the attacker sends packets to the IoT device that will keep the device busy with DoS attack tools and tries to weaken the data sent from the IoT device to the MQTT broker. Thus, it is tried to prevent data loss to the subscriber using the system.

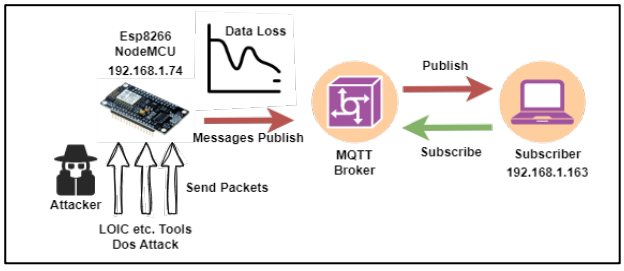


Figure 6. Denial of Service attack scenario

4.1.3 Brute Force Attack

Brute force attack is a type of attack that tries to obtain passwords used in systems. Since it is difficult to decipher passwords manually using trial and error method, automated tools using dictionaries are used to obtain passwords.

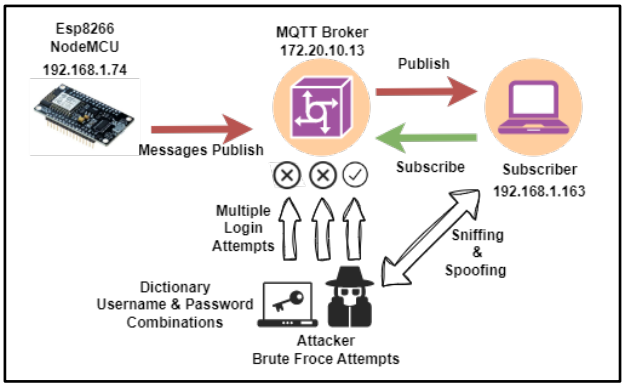


Figure 7. Brute force attack scenario

In the test environment established in our study, the attacker tries to capture user passwords with brute force attack tools as shown in Figure 7. If the attack is successful, malicious operations such as eavesdropping can now be performed with this information.

5. RESULTS

Authentication is not performed by default in the MQTT protocol. However, if desired, authentication mechanism with username and password can be used as shown in Figure 8. After authentication, the password information can be hidden.

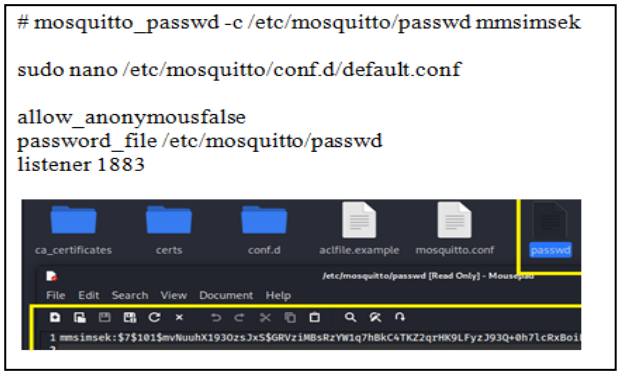


Figure 8. Authentication Mechanism in Mosquitto

In cases where MQTT is used in public networks or security is a concern, it is advised to activate authentication and other security measures to ensure that the transmitted data is protected from unauthorized access and maintain its confidentiality and integrity. It is the responsibility of the MQTT application developer to implement suitable security measures based on the specific use case and the type of threats they might face.

As shown in Figure 9, the authentication mechanism prevents attackers from publishing unauthorized data.

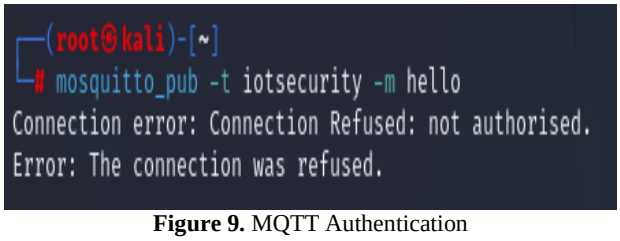


Figure 9. MQTT Authentication

However, since username and passwords are sent in plain text in MQTT, an attacker can intercept the packets with Wireshark-like tools after listening for the identity data and then modify them to send them to the MQTT server (Figure 10). This can potentially compromise the security and confidentiality of the data being transmitted, as well as the overall integrity of the system. To protect from such risks TLS encryption can be used. TLS can provide end-to-end encryption of MQTT messages, including the username and password. This can prevent eavesdropping and tampering by attackers who may attempt to intercept and modify MQTT packets. Another prevention could be using a more secure authentication mechanism. Instead of using usernames and passwords, an authentication mechanism such as JSON Web Tokens (JWT) or OAuth can be used. These mechanisms can provide more secure and robust authentication and authorization, and can prevent attackers from intercepting and modifying login credentials. Which of these methods to use may vary depending on the resource limitations of the relevant IoT device.

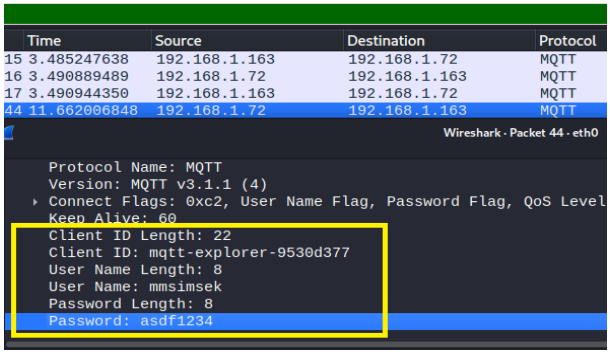


Figure 10. Listening with Wireshark

5.1 Man in the Middle (MitM) Attack

Network traffic of an IoT device working with the MQTT protocol is exposed to ARP poisoning, which is one of the MitM types. As seen in Figure 11, the attacker captures the data sent from the ESP8266 Sensor to the MQTT Broker, directs it to itself and reaches the data.

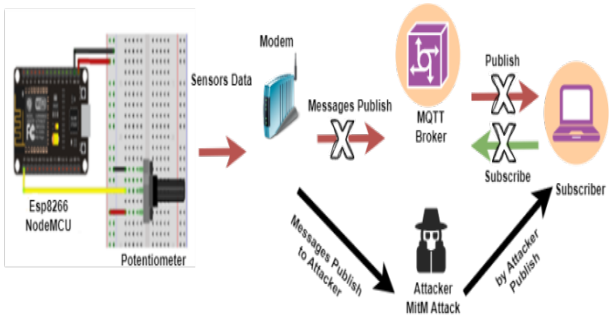


Figure 11. Experiment environment for MitM attack

As seen in Figure 12, it was determined that the device is an IOT device from the MAC addresses found by scanning the network with the Ettercap tool.

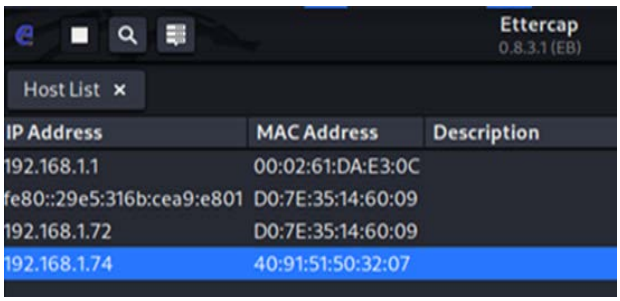


Figure 12. IoT device identification by MAC address

As shown in Figure 13, the attack phase was started after the IP address of our IOT device was found by following the connections in the Ettercap tool.

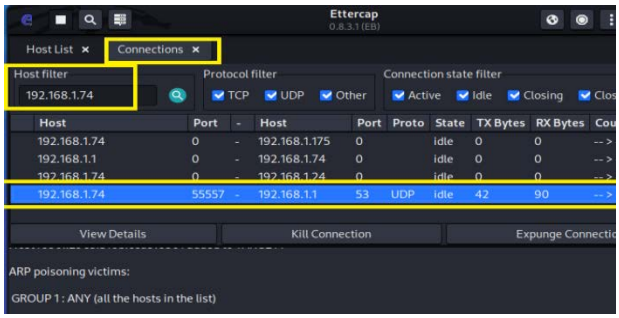


Figure 13. MQTT broker detection by following links

In our study, an attack was made with ARP poisoning, one of the MitM attack types found in the Ettercap tool. ARP (Address Resolution Protocol) [50] is a communication protocol used in computer networks to translate a network address into a physical address and store them in the ARP table. In ARP poisoning, the attacker manipulates this table and maps its own MAC address instead of the target device's MAC address. Then, the attacker intercepts and modifies the network traffic, and potentially gains access to sensitive information such as usernames, passwords, and other confidential data. With control over the network traffic, an attacker can inject malicious code or commands, and execute other malicious activities.

As shown in Figure 14, after the MitM attack started, the Wireshark tool captured the broker information that the IOT device sent data to.

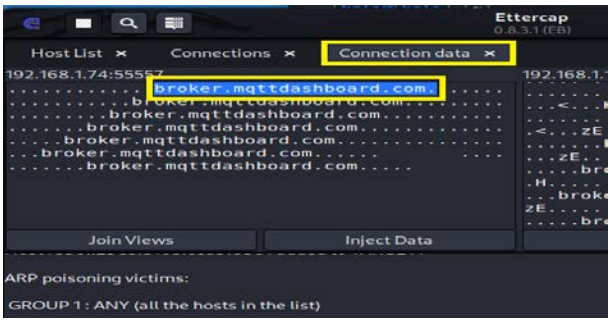


Figure 14. MQTT broker detection by following links

After the broker information was learned, the data sent from the IOT device to the broker was followed with the Wireshark tool and MQTT subject and message information was captured as shown in Figure 15.

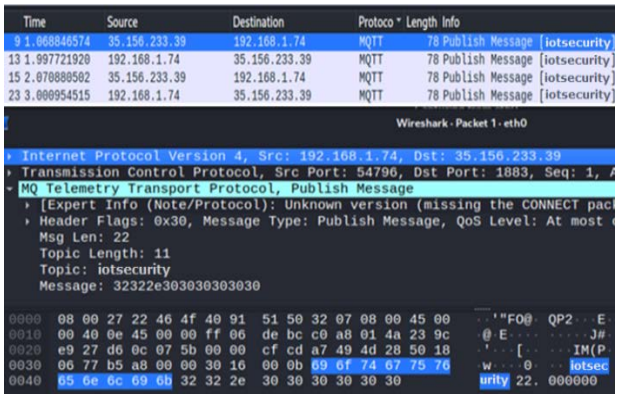


Figure 15. Listening to topics and messages with Wireshark

When the attacker captures the MQTT Topic and host information by listening to the network traffic, it can now replace the Broker (Figure 16). In tis attack, the attacker can easily change MQTT message and mislead the subscriber. Although MitM attack mainly targets the integrity of messages, it can be used to read the messages when MQTT system use encrypted communication.

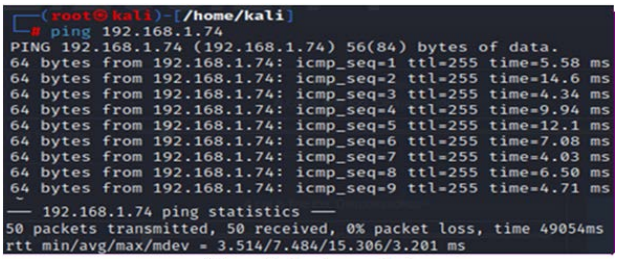
Figure 16. Fake data sent by the attacker

To reduce the risk of ARP poisoning attacks, network administrators can implement various security measures such as using encryption to protect network traffic, monitoring network traffic for suspicious behavior, and using tools such as Intrusion Detection Systems (IDS) to detect and prevent ARP poisoning attacks. Transport Layer Security (TLS) is recommended to secure confidentiality of MQTT traffic. Another countermeasure against MitM attacks is using MAC filtering to specify and limit known devices that can communicate with the MQTT broker. This can prevent unauthorized devices or users from accessing the MQTT broker and other devices on the network.

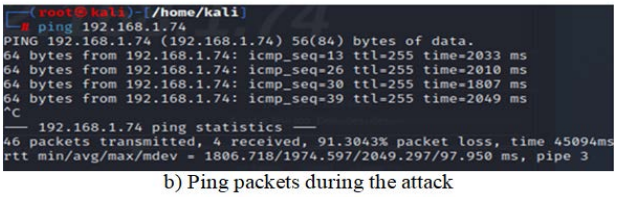
5.2 Denial of Service (DoS) Attacks

After the IP address of the ESP8266 device was detected with the Ettercap tool, the DoS attack was carried out with the LOIC tool.

The transmission times of the ping packets before the LOIC DoS attack on the 192.168.1.74 device is shown in Figure 17/a. Ping packet transmission times after the attack is started are shown in Figure 17/b. As seen in Figure 17, 50 packets reached the target in 49054 ms before the attack. It has been determined that there is no packet loss while sending ping packets.



a) Pre-attack ping packets



b) Ping packets during the attack

Figure 17. Ping packets tracking before and during DoS attack

When the DoS attack was started with the LOIC tool, 91% of the packets were lost on the way in the same time as seen in Figure 17/b.

To prevent the DoS attacks on IoT system, MQTT broker may limit the number of connections from each subscriber to prevent a single subscriber from overloading the system with requests. It is also recommended to use firewalls and IDS to monitor the MQTT network for suspicious traffic and activity. This can help to detect and prevent DoS attacks and other types of attacks.

5.3 Brute Force Attacks

In Brute Force attacks on MQTT protocol, the attacker tries all possible combinations of usernames and passwords to gain unauthorized access to an MQTT broker or subscribed topics. Attackers use automated software tools to generate a large number of username and password guesses and try each one until the correct combination is found. This type of attack can lead to unauthorized access, data theft, or disruption of the MQTT connection. In addition, the connection information such as client's IP address, port number and the version of the MQTT protocol, and the topics that the clients subscribed to can be revealed. In our study, Metasploit Framework tool was used to capture MQTT credentials with a brute force attack.

Before starting a brute-force attack, the Nmap tool inquires whether the target server uses authentication. The target IP is 172.20.10.13. From the scan, it is understood that we are not authorized to subscribe to any topic. This means that the target server is using authentication. Then MQTT Metasploit module is started in the Kali Linux terminal and the necessary parameters are set to find the user credentials. The Brute Force attack as shown in Figure 18 was initiated and the MQTT session information was captured.

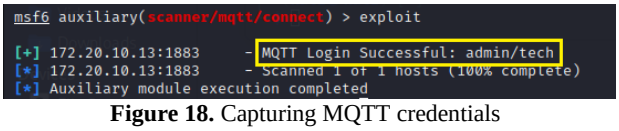


Figure 18. Capturing MQTT credentials

MQTT uses port 1883 for unencrypted communication. For encrypted communication, port 8883 is used by using Secure Sockets Layer (SSL). During SSL handshake, the client validates the server certificate and authenticates the server. As shown in Figure 19, although data was sent from port 8883, the user name and password information was captured by the Wireshark tool.

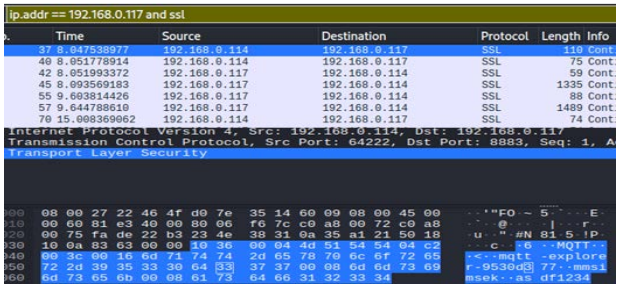


Figure 19. SSL protocol listening with Wireshark

To protect MQTT protocols from brute force attacks, implementing strong authentication and password policies should be the first line of defense. This may include implementing password complexity rules, and limiting the number of login attempts.

6. CONCLUSION

The study presented examines IoT middleware messaging protocols, beginning with an analysis of their benefits and drawbacks. The most frequently used messaging protocol, MQTT, is scrutinized in detail, including its operational principles, service quality, and security weaknesses. Additionally, various attack methods and tools that could be used against MQTT are analyzed, and their impacts are discussed. To demonstrate the risks, three attack types, including Man-in-the-Middle Attack, Denial of Service Attack, and Brute Force Attack, are selected, and attack scenarios are developed. The focus is on the availability of IoT devices that employ the MQTT protocol, and the tests are conducted using an ESP8266 device with both username and password authentication and without it. The study exposes security vulnerabilities in IoT devices utilizing MQTT and proposes ways to address these weaknesses and prevent such attacks.

Ethical Considerations

There are no ethical concerns associated with this article, and obtaining ethical permission was deemed unnecessary.

Funding

No funding related to this study has been received from any non-profit organization

Conflict of Interest

We, as authors, confirm that there is no conflict of interest with any person or institution related to this study.

REFERENCES

[1] Kevin, A. 2009, That 'Internet of Things' Thing, RFiD Journal, 22(7), 97-114.

[2] Oral, O., Çakır, M. 2017. Nesnelerin İnterneti Kavramı ve Örnek Bir Prototipin Oluşturulması. Mehmet Akif Ersoy Üniversitesi Fen Bilimleri Enstitüsü Dergisi, Özel Sayı 1, 172-177

[3] Hintaw, A. J., Manickam, S., Karuppayah, S., and Aboalmaaly, M. F. 2019. A brief review on MQTT's security issues within the internet of things (IoT), Journal of Communication, 14(6), 463–469, doi: 10.12720/jcm.14.6.463-469.

[4] Chen, F., Huo, Y., Zhu, J., and Fan, D. 2020. A Review on the Study on MQTT Security Challenge, In 2020 IEEE Int. Conf. Smart Cloud, SmartCloud, 128–133, doi: 10.1109/SmartCloud49737.2020.00032.

[5] Upadhyay, Y., Borole, A., and Dileepan, D. 2016. MQTT based secured home automation system, 2016 Symp. Colossal Data Anal. Networking, *CDAN 2016*, 14–17, doi: 10.1109/CDAN.2016.7570945.

[6] Assaig, F. A. A., Khalifa, O. O., Gunawan, T. S., Halbouni, A. H., Hamidi, E. A. Z., and Ismail, N. 2022. Development of A Lightweight IoT Security System, In 8th Int. Conf. Wirel. Telemat. ICWT 2022, doi: 10.1109/ICWT55831.2022.9935476.

[7] Firdous, S. N., Baig, Z., Valli, C., and Ibrahim, A. 2017. Modelling and evaluation of malicious attacks against the IoT MQTT protocol, In *IEEE Int. Conf. Internet Things, IEEE Green Comput. Commun. IEEE Cyber, Phys. Soc. Comput. IEEE Smart Data, iThings-GreenCom-CPSCom-SmartData*, 748–755. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.115.

[8] Andy, S., Rahardjo, B., and Hanindhito, B. 2017. Attack scenarios and security analysis of MQTT communication protocol in IoT system. In 4th Int. Conf. Electr. Eng. Comput. Sci. Informatics. 19–21, doi: 10.1109/EECSI.2017.8239179.

[9] OASIS, MQTT Version 5.0. 2019. OASIS Standard.

[10] Florea, I., Rughinis, R., Ruse, L., and Dragomir, D. 2017. Survey of Standardized Protocols for the Internet of Things. In 21st International Conference on Control Systems and Computer, CSCS 2017, 190–196. doi: 10.1109/CSCS.2017.33.

[11] Shelby, Z., Hartke, K., and Bormann, C. 2014. The constrained application protocol (CoAP). No. rfc7252.

[12] Dürkop, L., Czybik, B., and Jasperneite, J. 2015. Performance evaluation of M2M protocols over cellular networks in a lab environment. In 2015 18th International Conference on Intelligence in Next Generation Networks, ICIN 2015. 70–75. doi: 10.1109/ICIN.2015.7073809.

[13] Prayogo, S. S., Mukhlis, Y., and Yakti, B. K. 2019. The Use and Performance of MQTT and CoAP as Internet of Things Application Protocol using NodeMCU ESP8266. In 4th Int. Conf. Informatics Comput. ICIC 2019. doi: 10.1109/ICIC47613.2019.8985850.

[14] Farooq, M. S., Riaz, S., Abid, A., Abid, K., and Naeem, M. A. 2019. A Survey on the Role of IoT in Agriculture for the Implementation of Smart

Farming. IEEE Access, 7, 156237–156271, doi: 10.1109/ACCESS.2019.2949703.

[15] Kuladinithi, K., Bergmann, O., Thomas Pötsch, Becker, M., and Görg, C. 2011. Implementation of coap and its application in transport logistics. In Extending Internet to Low Power Lossy Networks, 1–7.

[16] Krimmling, J., and Peter, S. 2014. Integration and evaluation of intrusion detection for CoAP in smart city applications. In IEEE Conf. Commun. Netw. Secur., CNS 2014. 73–78, doi: 10.1109/CNS.2014.6997468.

[17] Uy, N. Q., and Nam, V. H. 2019. A comparison of AMQP and MQTT protocols for Internet of Things. In 6th NAFOSTED Conference on Information and Computer Science, NICS 2019. 292–297. doi: 10.1109/NICS48868.2019.9023812.

[18] McAteer, I. N., Malik, M. I., Baig, Z., and Hannay, P. 2017. Security vulnerabilities and cyber threat analysis of the amqp protocol for the internet of things. In 15th Aust. Inf. Secur. Manag. Conf. AISM 2017, 70–80, doi: 10.4225/75/5a84f4a695b4c.

[19] Keophilavong, T., Widyawan, and Rizal, M. N. 2019. Data transmission in machine to machine communication protocols for internet of things application: A review. In International Conference on Information and Communications Technology, ICOIACT 2019, IEEE, 899–904. doi: 10.1109/ICOIACT46704.2019.8938420.

[20] Naik, N. 2017. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In IEEE Int. Symp. Syst. Eng. ISSE 2017. 1–7. doi: 10.1109/SysEng.2017.8088251.

[21] Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., and Manzoni, P. 2014. Testing amqp protocol on unstable and mobile networks. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 8729, 250–260. doi: 10.1007/978-3-319-11692-1_22.

[22] Krishna, C. S., and Sasikala, T. 2019. Healthcare Monitoring System Based on IoT Using AMQP Protocol. Lecture Notes on Data Engineering and Communications Technologies, 15, 305–319. doi: 10.1007/978-981-10-8681-6_29.

[23] Ramana, S. 2022. A Three - Level Gateway protocol for secure M - Commerce Transactions using Encrypted OTP, no. Icaaic, 1408–1416.

[24] Javied, T., Huprich, S., and Franke, J. 2019. Cloud based Energy Management System Compatible with the Industry 4.0 Requirements, IFAC-PapersOnLine, 52(10), 171–175. doi: 10.1016/j.ifacol.2019.10.018.

[25] Bartolomeo, G., and Kovacikova, T. 1996. Hypertext Transfer Protocol, Identif. Manag. Distrib. Data, 31–48. doi: 10.1201/b14966-5.

[26] Wukkadada, B., and Wankhede, K. 2018. Comparison with HTTP and MQTT In Internet of Things (IoT). In International Conference on Inventive Research in Computing Applications (ICIRCA 2018), IEEE, 249–253.

[27] Jaafar, G. A., Abdullah, S. M., and Ismail, S. 2019. Review of Recent Detection Methods for HTTP DDoS Attack. Journal of Computer Networks and Communications, 2019, Article ID 1283472.

[28] Pardo-Castellote, G. 2003. OMG Data-Distribution Service: Architectural Overview. In 23rd International Conference on Distributed Computing Systems Workshops, 200-206. doi: https://doi.org/10.1109/ICDCSW.2003.1203555.

[29] Du, J., Gao, C., and Feng, T. 2023. Formal Safety Assessment and Improvement of DDS Protocol for Industrial Data Distribution Service. Futur. Internet, 15(1). doi: 10.3390/fi15010024.

[30] MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf (accessed Mar. 19, 2023).

[31] Bandyopadhyay, S., and Bhattacharyya, A. 2013. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. In Int. Conf. Comput. Netw. Commun. ICNC 2013, 334–340. doi: 10.1109/ICCNC.2013.6504105.

[32] Mishra, B., and Kertesz, A. 2020. The use of MQTT in M2M and IoT systems: A survey. IEEE Access, 8, 201071–201086. doi: 10.1109/ACCESS.2020.3035849.

[33] Soni, D., and Makwana, A. 2017. A Survey on MQTT: a protocol of internet of things (IoT). In International Conference on Telecommunication, Power Analysis and Computing Techniques (Ictpact - 2017), 173–177.

[34] Kant, D., Johannsen, A., and Creutzburg, R. 2021. Analysis of IoT security risks based on the exposure of the MQTT Protocol. Electronic Imaging, 2021(3), 96-1. doi: 10.2352/ISSN.2470-1173.2021.3.MOBMU-096.

[35] Chen, D., and Varshney, P. K. 2004. QoS support in wireless sensor networks: A survey. In Int. Conf. Wirel. Networks, ICWN'04. 233, 1–7.

[36] Bender, M., Kirdan, E., Pahl, M. O., and Carle, G. 2021. Open-source MQTT evaluation. IEEE 18th Annu. Consum. Commun. Netw. Conf. 1-4. doi: 10.1109/CCNC49032.2021.9369499.

[37] Ugalde, D. S. 2018. Security analysis for MQTT in Internet of Things. Master Thesis. 53p. Stockholm.

[38] Atilgan, E., Ozcelik, I., and Yolacan, E. N., MQTT Security at a Glance. 2021. In 14th International Conference on Information Security and Cryptology, ISCTURKEY 2021, Ankara, 138–142. doi: 10.1109/ISCTURKEY53027.2021.9654337.

[39] Setiawan, F. B. 2021. Securing Data Communication Through MQTT Protocol with AES-256 Encryption Algorithm CBC Mode on ESP32-Based Smart Homes. In Int. Conf. Comput. Syst. Inf. Technol. Electr. Eng. (COSITE). 166–170. doi: 10.1109/COSITE52651.2021.9649577.

[40] Manullang, I. T. 2021. The Implementation of XChaCha20-Poly1305 in MQTT Protocol, no. 13517044.

[41] Sadio, O., Ngom, I., and Lishou, C. 2019. Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In 6th Int. Conf. Internet Things Syst. Manag. Secur. IOTSMS 2019, 119–123. doi: 10.1109/IOTSMS48152.2019.8939177.

[42] Pallavi, A., and Hemlata, P. 2012. Network Traffic Analysis Using Packet Sniffer. Int. J. Eng. Res. Appl. 2(3), 854–856.

[43] Wireshark. https://www.wireshark.org/ (accessed Mar. 25, 2023).

[44] Hertzog, R., O'Gorman, J., and Aharoni, M. 2017. Kali Linux Revealed. Mastering the Penetration Testing Distribution

[45] Shah, M., Ahmed, S., Saeed, K., Junaid, M., Khan, H., and Ata-Ur-Rehman. 2019. Penetration testing active reconnaissance phase - Optimized port scanning with nmap tool. In 2nd Int. Conf. Comput. Math. Eng. Technol. iCoMET 2019. doi: 10.1109/ICOMET.2019.8673520.

[46] Nagpal, B., Sharma, P., Chauhan, N., and Panesar, A. 2015. DDoS tools: Classification, analysis and comparison. In 2nd Int. Conf. Comput. Sustain. Glob. Dev. INDIACom 2015, 342–346.

[47] Hping3, 2023. https://www.kali.org/tools/hping3/ (accessed Apr. 16, 2023).

[48] Valea, O., and Oprisa, C., Towards Pentesting Automation Using the Metasploit Framework. 2020. In IEEE 16th Int. Conf. Intell. Comput. Commun. Process. ICCP 2020, 171–178. doi: 10.1109/ICCP51029.2020.9266234.

[49] Özdemir, D., and Çavşi Zaim, H., Investigation of Attack Types in Android Operation System. 2021 J. Sci. Reports - A, no. 46, 34–58.

[50] Nachreiner, C. 2003. Anatomy of an ARP Poisoning Attack. (Accessed July 6, 2023).