MQTT Traffic Collection and Forensic Analysis Framework

Raymond Chan[1(✉)], Wye Kaye Yan[1], Jung Man Ma[1], Kai Mun Loh[1], Greger Chen Zhi En[1], Malcolm Low[1], Habib Rehman[2], and Thong Chee Phua[2]

[1] Singapore Institute of Technology, Singapore, Singapore
{Raymond.Chan,Wyekaye.Yan,Jungman.Ma,Malcolm.Low}@singaporetech.edu.sg,
{2100817,2100641}@sit.singaporetech.edu.sg
[2] Firefish Communications, Singapore, Singapore
{habib,thongchee}@firefishcomms.com

Abstract. Message Queue Telemetry Transport (MQTT) is a common protocol used for Internet-of-Things (IoT) devices communication. In recent years, IoT devices are deployed in Operational Technology (OT) systems such as building management system (BMS). It enables the capability to control the infrastructure within a building, and can be considered a miniature industrial control system. With the increased use of these devices to further enhance the functionality of such systems, there is also an increased risk of vulnerabilities that come with these devices. Cyber-security must be one of the top priorities to be taken into the consideration at the various stages when designing the BMS to achieve operational reliability. In this paper, we proposed a real-time MQTT logging and abnormal detection framework with push notifications. It can be used to collect digital evidence for forensic investigation and monitor cyber-attacks.

Keywords: Building management system · forensic analysis · MQTT · Internet of Things

1 Introduction

The use of IoT devices is increasing every year, and is considered one of the key factors that contributes to the huge volume of data exchanged in Information and Communication Technology (ICT) networks [1]. The MQTT protocol is commonly used within an IoT environment, and is designed to be a lightweight, machine to machine (M2M) network protocol, which at its core, uses a publish and subscribe communication model. The protocol functions on a server-client model, where the client publishes messages and the server, also known as the broker, will receive those messages filtering them by topic before relaying the messages to the subscribers. MQTT protocol is making significant gains into industrial automation and is also widely used in other applications and areas

S. Goel et al. (Eds.): ICDF2C 2022, LNICST 508, pp. 185–202, 2023.
https://doi.org/10.1007/978-3-031-36574-4_11

such as ECG monitoring system for healthcare [2], smart home automation [3], push notification system for smartphone applications [4], weather monitoring and smart farming for agriculture sector [5].

Compared to other protocols like HTTP, MQTT protocol has the ability to transfer data at a much faster rate [6]. Therefore, it is more ideal for resource constraint environments. However, although there are many advantages in using MQTT protocol, there are security risks involved with the increasing complexity of the IoT model, security flaws and vulnerabilities are highly common today in IoT devices, and there are a wider variety of attacks than in the past [7]. The security mechanism of MQTT protocol required further improvement and development [8] as many existing MQTT systems currently in use are still lacking basic security controls [9]. In [9], the authors performed Man-in-the-middle (MITM) attack on MQTT-based IoT devices. They concluded that their designed attack scheme had successfully avoided commonly used classification-based anomaly detection models. The security of the MQTT protocol was further investigated in [10] which the authors identified the weakness of the MQTT protocol to a slow denial of service attack. The configuration of the KeepAlive parameter and MQTT packets were two specific MQTT flaws that the authors exploited to launch a cyberattack against the MQTT broker. The results have shown that the attacks were successful and the vulnerability can be used to execute a denial of service against the IoT network by maintaining the connection for a very long time.

In a BMS setup that has IoT devices within the network, the devices are connected to a MQTT server and subscribe or publish to a topic. The messages are small in size, hence the popularity for its uses in IoT. Therefore, to safeguard the MQTT protocol from security risk, the proposed solution will consist of two parts: a network-based detection component for traffic collection and a methodology to analyze the traffic for anomaly detection, and a notification component for the front-end to alert users of the anomaly detected.

2 Related Work

There are several approaches to detect cyber threats and attacks on MQTT protocols. Budiana introduced a method to use a fuzzy logic algorithm embedded in a node to detect Denial of Service (DoS) in the MQTT protocol with feature selection nodes. The SUBSCRIBE and SUBACK traffic was monitored, and it provided the information to fuzzy input nodes to detect DoS attacks [11].

Another proposed way is to develop classification models that can use for an Intrusion Detection System (IDS), utilizing a specific dataset with particular attacks for the MQTT protocols. In their case study, machine learning techniques were used to classify the frames that an IDS can assign as attack or normal traffic [12]. In [13], the authors introduced ARTEMIS, an IDS for IoT that analyzes data from IoT devices using machine learning to detect changes from the system's typical behavior and sends alarms in the event of abnormalities. Hindy [14] discussed the effectiveness of six machine learning techniques to detect MQTT-based attacks. A MQTT simulated dataset is produced for the training and

evaluation processes. The authors concluded that the results highlighted how crucial it is to distinguish between MQTT-based attacks and legitimate traffic using flow-based features, whereas packet-based features are sufficient to identify traditional networking attacks.

In recent years, machine learning techniques have been utilised to detect cyber-attacks on networks and infrastructure. There are a number of research studies applying machine learning for MQTT attack detection. Vaccari presented MQTTset, a dataset that combines the legitimate dataset with cyber-attacks against the MQTT network [1]. The authors evaluated and validated the dataset by implementing and comparing it to different machine learning algorithms widely adopted in the cyber-security field. They concluded that MQTTset can be used for a possible detection system related to the MQTT protocol.

Through intrusion and DoS attacks on publicly available MQTT test brokers, Chunduri analysed the availability of MQTT network traffic and obtained sensitive information and validated its security implications [15]. The purpose of their research is to demonstrate the negative impact of security measures on MQTT by attacking the MQTT brokers. Similarly, Anthraper provided an overview of the security and privacy concerns of the IoT by analysing MQTT protocol [16]. Afterward, the paper discussed IoT forensics and concluded by highlighting the security challenges related to IoT. In [17], the author introduced a method Value-to-Keyed-Hash Message Authentication Code (Value-to-HMAC) mapping to ensure the confidentiality and integrity of information in MQTT.

Research studies in MQTT data analysis and attack detection areas are still considered very limited and insufficient. In order to fill this gap, we proposed a framework for traffic analysis based on MQTT traffic collected from an IoT-enabled BMS. Moreover, a push notification is created based on the proposed analysis framework to provide attack detection alerts to the users. The notification component alerts users in the event of unauthorised subscription, denial of service attack, brute force attack, and inactivity of Zigbee devices. Though device inactivity is uncommon, there are devices by default that are designed to operate in idle or inactive until a specific function call activates it. Also, though exploitable, MQTT has a KeepAlive feature that by nature is used to verify the device's connectivity with the system. However, there are scenarios like bridging Zigbee to MQTT and Modbus to MQTT where it does not have a KeepAlive feature or a function with a similar feature by default. The team has recognised it as a security risk due to the increasing complexity of the IoT model that compromises the integrity of the Confidentiality, Integrity, and Availability (CIA) triad and the Authentication, Authorization, and Accounting (AAA) security framework.

3 MQTT Traffic Collection and Analysis Methodology

3.1 Testbed Implementation

By leveraging on an existing IoT integrated BMS, various devices within the setup have been selected for traffic collection and analysis. As illustrated in

Fig. 1, these devices can be classified into two categories: MQTT devices such as a people counting radar and an elevator control system, and non MQTT devices that have been bridged to the MQTT protocol such as Zigbee-enabled smart light bulbs, smart air quality sensors, and smart locks.

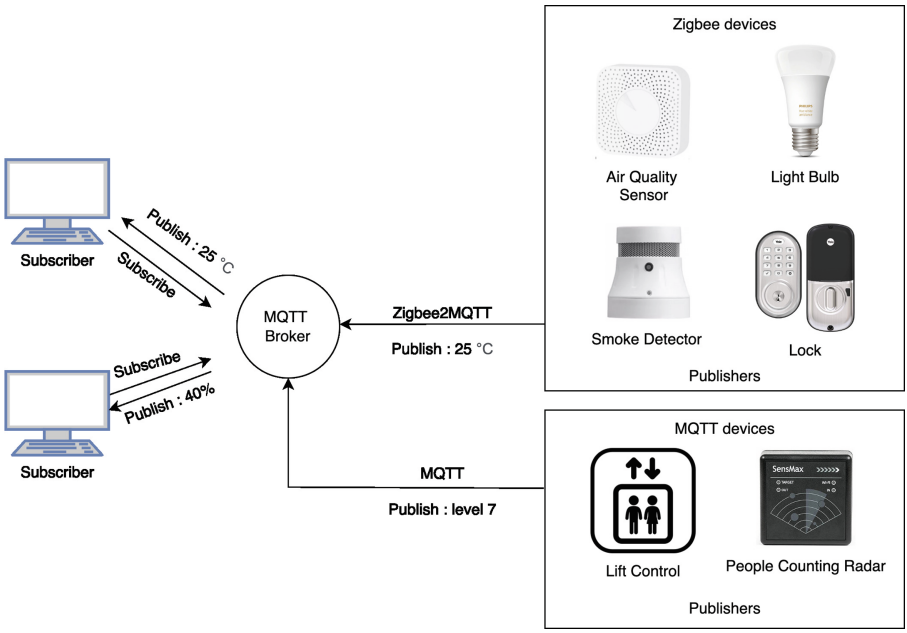

Fig. 1. Architecture of testbed

3.2 MQTT Protocol Implementation

As mentioned above, the selected devices are connected, controlled and monitored using a central control system called Home Assistant (HA). A locally integrated MQTT protocol is available in HA, it runs on a 3.1.1 specification by default and falls back to version 3.1 if the chosen server does not support it. For this experiment, Eclipse Mosquitto broker which supports MQTT version 3.1.1 is used. The Mosquitto broker acts as an intermediary between clients, in this case, between the devices selected. These clients will publish messages to other clients which have subscriptions established. As illustrated in Fig. 2, the description of the MQTT parties and their functions in the protocol are as follows:

- Broker (Centralised Server): Receives, filters, and distributes relevant messages to clients that are subscribed
- Publisher: Publishers send a message to a topic, which is sent via the broker. Subscribers will then be notified of the message

– Subscriber: Clients send a SUBSCRIBE message to the broker to receive messages on specific topic(s) of interest

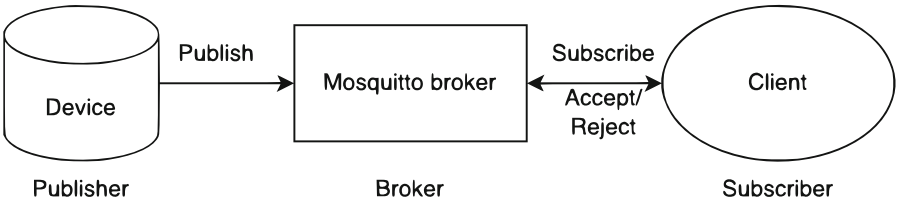

Fig. 2. MQTT subscriber and publisher model

3.3 MQTT Network-Based Forensic Framework

MQTT Traffic Collection. It is done through a containerised docker custom-made to run on HA that will leverage the current functionalities and Application Programming Interfaces (API) provided by HA. The docker container runs in a unique Alpine Linux environment and has controlled access to the HA host environment using the Bashio shell library and API. Each add-on is isolated from one another and can only communicate through the HA host or shared file system volumes. Within the docker container is the proposed network-based detection component, to streamline the support for MQTT with the Mosquitto broker, a MQTT client developed by Eclipse is used. The Paho Python client supports the same MQTT versions as the Mosquitto broker, and Python versions 2.7 and 3.x. Furthermore, Scapy, a packet manipulation tool written in Python that supports MQTT is included in the docker container and used to collect and analyse the traffic.

Similar to the research conducted by Husnain [18], a detection model that monitors the network for anything MQTT related protocols and MQTT Control packets is done through Scapy. There are 13 MQTT Control packet sub-layers that Scapy monitors:

– **MQTT**
– **MQTTConnack:** Connection acknowledgement from Server to Client
– **MQTTConnect:** Request to connect from Client to Server
– **MQTTDisconnect:** A disconnect message from Client to Server
– **MQTTPuback:** Publish acknowledgement (QoS 1 Response) from Client to Server and vice versa
– **MQTTPubcomp:** Publish complete (Last part of the QoS 2) from Client to Server and vice versa
– **MQTTPublish:** Publish message from Client to Server and vice versa
– **MQTTPubrec:** Publish received (First part of QoS 2) from Client to Server and vice versa

- **MQTTPubrel:** Publish released (Second part of QoS 2) from Client to Server and vice versa
- **MQTTSubscribe:** Request to subscribe from Client to Server
- **MQTTUnsuback:** Unsubscribe acknowledgement from Client to Server
- **MQTTUnsubscribe:** Request to unsubscribe from Client to Server

Out of the 13 MQTT Control packet sub-layers that Scapy monitors, only 4 signals from the packets are used by clients. Those signals are Publish, Subscribe, Unsubscribe, and Connect. The other signals are part of the server-client model protocol functions that MQTT makes use of through the use of the publish and subscribe communication model.

MQTT Forensic Analysis Methodology. is conducted by employing two methods: a log-based analysis method to analyse MQTT topics, and a rule-based engine for MQTT traffic analysis. In the log-based analysis method, there are two types of logs that are generated from the Mosquitto broker, namely System Status, and Information and Debugging logs. By default, the log types that are log are Error, Information, Notice, and Warning events. However, as Mosquitto broker allows its users to configure its logging for all log types, by doing so, all types of events for activities such as debugging can be monitored. These logs are generated into a log file. To leverage the contents in the log for anomaly detection, it has been configured for remote monitoring by routing the logs to the $SYS topic.

For the design and structure of the rule-based engine definitions, it is similar to Snort rules, which is a common method used in IDS and intrusion prevention system (IPS). This makes the rule definitions flexible and relatively simple to configure to detect threats on a case-by-case basis by defining the parameters. The rule definition structure consists of 4 rule options: Header, Name, Priority, and Additional options. Within each rule option are configurable parameters as illustrated in Fig. 3, description of the configurable parameters in the rule options are as follows:

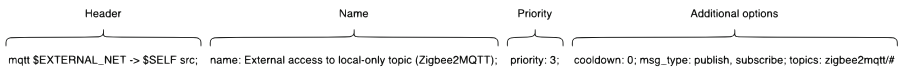

Fig. 3. Rule options parameters

- Header: In the Header, the user decides which protocol to analyse, from a source, to a destination, and the target to be identified as the threat actor.
  Protocol: TCP/MQTT
  Source: Source IP/$ALIAS
  Destination: Destination IP/$ALIAS
  Target: Source/Destination

– Name: Declares a rule name which will be included in the notification payload
– Priority: Classifies the priority of the threat.
  Priority 4: Critical
  Priority 3: High
  Priority 2: Medium
  Priority 1: Low
  Priority 0: Informational
– Additional: There are two types of additional parameters that can be configured to fine-tune the detection.
  General options: Threshold/Interval/Cooldown
  MQTT options: msg_type/QoS/Retcode/topics.

3.4 Attack Detection

This component that is part of the proposed solution is designed to complement the MQTT traffic collection and analysis framework. The purpose of the push notifications is to provide real-time alerts to the users about the abnormalities detected. These real-time alerts are essential for preventing possible attacks made on the system thus, WebSocket which is a low-latency bi-directional communication protocol between server and client is used.

Furthermore, by leveraging on WebSocket protocol, Socket.io which is a library based on WebSocket protocol that enables real-time, event-based bi-directional communication will be employed in the solution. As compared to pure WebSocket protocol, it provides extra features that reduce the complexity of producing WebSocket applications [19] thus, making it suitable for this setup.

Figure 4 illustrates the communication flow between the MQTT broker and front-end notifications. MQTT communications have a persistent connection with the back-end of the Notification component which is written in JavaScript running on Node.js. For the purpose of this experiment, SQLite3 database is chosen for the persistent structured data storage as the package is lightweight and relatively performant for this setup. At the front, web technologies written in HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript ES6 provide visualisation for the alerts panel.

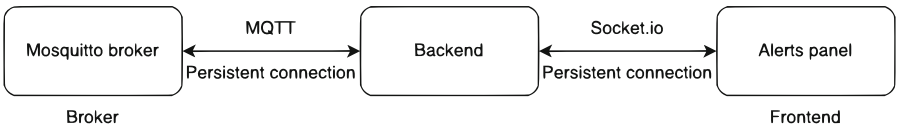

Fig. 4. Communication flow

3.5 Technology Stack

The architecture for the proposed solution leverages on an existing IoT integrated BMS as part of the testbed. With the use of a central control system, HA, a custom docker container that is designed for MQTT traffic collection and analysis works as a client with a locally installed Mosquitto broker in HA. Results from the traffic collection and analysis are pushed to the front-end alerts panel for easy visualisation. This is done so through a persistent connection from MQTT to the back-end of the notification component which has a database to store the persistent structured data before pushing the alerts to a web front-end through a persistent connection using Socket.io (Fig. 5).

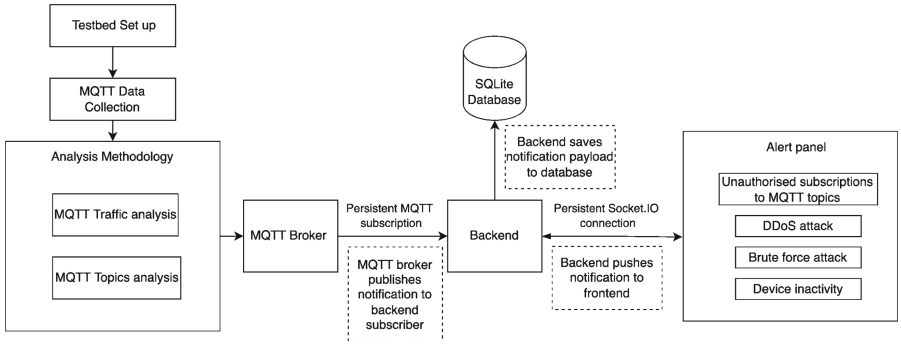

Fig. 5. MQTT traffic capture architecture

4 Experiment

In this section, we discussed the MQTT common vulnerabilities and further analysed the effectiveness of the proposed solution for detecting MQTT vulnerabilities in MQTT and Zigbee devices. With the proposed methodology, we are able to provide a front-end notifications component to alert users about abnormal detection. Our notifications component includes the following four vulnerability detection: Unauthorised Subscription Detection, Denial-of-Service Attack Detection, Brute Force Attack Detection and Zigbee2MQTT Devices Inactivity Detection. The detection methods are further explained in the following subsections.

4.1 Unauthorised Subscription Detection

In general, MQTT topics are created by subscribers or publishers, and the brokers use the topic of a message to decide which client receives which message. Apart from the typical topics, there is a special topic that contains information

about the broker itself, which is $SYS topics. Topics of this type are unique meta topics that allow the broker to communicate details about the broker and the MQTT client session.

An example of $SYS topics is $SYS/broker/clients/connected, this topic provides information about the total number of currently connected clients. $SYS topics also contain MQTT logs that reveal information such as broker version, IP addresses, client names, and subscriptions that should not be made known to any unauthorised users. $SYS topics are often used for developing and debugging MQTT applications. However, attackers can take advantage of this feature and exploit $SYS topics to expose internal information and lead to data breaches.

An access control list (ACL) can be built within the broker to make sure that all access is allowed in order to prevent unauthorised subscription of $SYS topics. The default ACL is intended to restrict the client's permissions on the system topic $SYS/# and all wildcard topics. The ACL, however, is insufficient to guarantee privacy when using devices that communicate using the MQTT protocol. This is because ACL is a useful security measure against attacks that target user credentials, but the attacker's aim isn't always the user's credentials [20]. In addition, if the attackers are able to bypass ACL, the confidentiality of the system will be compromised. Therefore, it's crucial to have an additional layer of security features in place that provides quick and effective detection for unauthorised subscriptions.

In our proposed solution, unauthorised subscription to $SYS topics detection is done by checking subscriptions to the $SYS and any subtopics of $SYS. The '$SYS/broker/log/M/subscribe' topic is subscribed in order to keep track of which clients are subscribed to which $SYS topics. This would help link the source of the client to any of the $SYS topics that it is subscribed to. There is a list of whitelisted IP addresses that are permitted to subscribe to any $SYS topics or a specified topic. Therefore, if an IP address that does not belong to the list of allowed IP addresses is detected to be subscribing to any of the $SYS topics, it is considered as an unauthorised subscription. The Fig. 6 shows how a notification for an unauthorised user subscribing to any $SYS topics or access to protected topics will appear in the dashboard, if it is detected every 30 s. As long as the IP is not whitelisted, it would be considered as unauthorised subscription. As shown in Fig. 7 and 8, the metadata of the unauthorised subscription will be shown in JSON format under the raw details section.

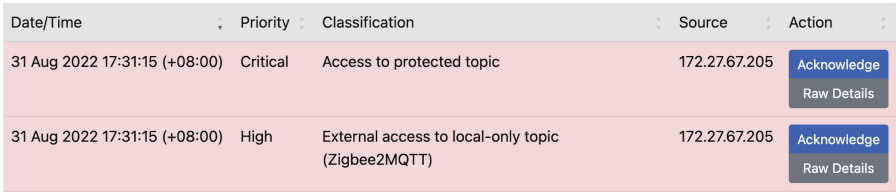

Fig. 6. Unauthorised subscription alert

```
{
 "uuid": "f3c6641a6ac54b7683f3a57d46a5e7eb",
 "datetime": "2022-08-31T17:31:15.639841+08:00",
 "classification": "Access to protected topic",
 "priority": "Critical",
 "src": "172.27.67.205",
 "rule_raw": "mqtt $EXTERNAL_NET -> $SELF src; name: Access to protected topic; priority: 4;
  cooldown: 0; msg_type: publish, subscribe; topics: $SYS/#, is2/mqtt_attack_notifications",
 "metadata": {
  "first_violation": "2022-08-31T17:31:15.639841+08:00",
  "total_violation": 1,
  "mqtt": {
   "client_id": [
    "mqtt-explorer-ddf55658"
   ],
   "auth": {
    "username": [
     "mqtt-user"
    ],
    "password": []
   },
   "topics": {
    "publish": {},
    "subscribe": {
     "# (QOS: 0)": 1
    },
    "unsubscribe": {}
   }
  }
 }
}
```


Fig. 7. Access to protected topic metadata

4.2 Denial of Service Attack Detection

DoS attacks aim to overload server resources and prevent legitimate clients from accessing the services. A massive volume of publishing, subscribing, and connecting messages that eventually exhaust node resources and prevent the node from delivering normal services is referred to as a DoS attack against MQTT [7].

In MQTT, DoS attacks can be performed in several ways depending on the control packet type and access level. The most common method is to flood the targeted MQTT broker with a large volume of CONNECT packets to overwhelm the broker with the processing of authentication requests. Another common method is by embedding a will payload on a CONNECT packet so the packet size increases. As a result, both the bandwidth and CPU resources of the victim server will be consumed, preventing new connections from being processed [21]. Additionally, it is also possible for an attacker to flood the broker with invalid subscriptions even with valid credentials but are not authorised to access various topics.

In order to simulate a DoS attack, a script was written that sends more than 100 TCP SYN network packets within 10 s, using hping3 to generate the packets. DoS attack detection is achieved by reviewing the logs and maintaining an IP address dictionary. As shown in Fig. 9, multiple new client connection messages will be shown in the MQTT log in the event of DoS attack. Every time a new IP address is discovered, it will be added to the dictionary as a key with the value of 1. Subsequently, if the IP address already exists in the dictionary as a key, its value will be incremented by 1. In Fig. 10, a notification alert will be triggered as a DoS if the total violation value is higher than the threshold of 100. Moreover,

```
{
 "uuid": "9a591f127ba347fe81f15206f19aaefe",
 "datetime": "2022-08-31T17:31:15.639841+08:00",
 "classification": "External access to local-only topic (Zigbee2MQTT)",
 "priority": "High",
 "src": "172.27.67.205",
 "rule_raw": "mqtt $EXTERNAL_NET -> $SELF src; name:
  External access to local-only topic (Zigbee2MQTT); priority: 3; cooldown: 0;
  msg_type: publish, subscribe; topics: zigbee2mqtt/#",
 "metadata": {
  "first_violation": "2022-08-31T17:31:15.639841+08:00",
  "total_violation": 1,
  "mqtt": {
   "client_id": [
    "mqtt-explorer-ddf55658"
   ],
   "auth": {
    "username": [
     "mqtt-user"
    ],
    "password": []
   },
   "topics": {
    "publish": {},
    "subscribe": {
     "# (QOS: 0)": 1
    },
    "unsubscribe": {}
   }
  }
 }
}
```


Fig. 8. External access to local-only topic metadata

monitoring and detection are not only focused on client connections but also on excessive MQTT messages being published to the broker. As shown in Fig. 12, the notification is triggered as excessive MQTT message publish activity if the total violation value is more than 20 within 5 s. As DoS attacks in MQTT can also be launched by flooding the broker with high QoS messages on the most subscribed topic. Therefore, it is important to monitor the number of messages being published to the brokers.

4.3 Brute Force Attack Detection

Brute force attacks usually consist of attempts to gain access to a system by employing a guessing technique for the authentication token, i.e. password, username, one-time token, etc. Attacks of this nature will be challenging to detect and investigate due to the high number of client devices. Furthermore, such attacks are extremely common in MQTT as the MQTT built-in authentication mechanism is very weak [22] and most users still use weak credentials that can be guessed easily [23]. It is therefore important to detect brute force attacks as early as possible to protect the system from security threats.

```
1662349165: Client mqtt-explorer-ddf55658 disconnected.
1662349166: New connection from 172.16.2.168:50407 on port 1883.
1662349166: New client connected from 172.16.2.168:50407 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349167: Client mqtt-explorer-ddf55658 disconnected.
1662349168: New connection from 172.16.2.168:50408 on port 1883.
1662349168: New client connected from 172.16.2.168:50408 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349169: Client mqtt-explorer-ddf55658 disconnected.
1662349170: New connection from 172.16.2.168:50410 on port 1883.
1662349170: New client connected from 172.16.2.168:50410 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349171: Client mqtt-explorer-ddf55658 disconnected.
1662349172: New connection from 172.16.2.168:50411 on port 1883.
1662349172: New client connected from 172.16.2.168:50411 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349173: Client mqtt-explorer-ddf55658 disconnected.
1662349174: New connection from 172.16.2.168:50412 on port 1883.
1662349174: New client connected from 172.16.2.168:50412 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349175: Client mqtt-explorer-ddf55658 disconnected.
1662349176: New connection from 172.16.2.168:50413 on port 1883.
1662349176: New client connected from 172.16.2.168:50413 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349178: Client mqtt-explorer-ddf55658 disconnected.
1662349179: New connection from 172.16.2.168:50415 on port 1883.
1662349179: New client connected from 172.16.2.168:50415 as mqtt-explorer-ddf55658 (p2, c1, k60, u'mqtt-user').
1662349180: Client mqtt-explorer-ddf55658 disconnected.
1662349969: Saving in-memory database to /data//mosquitto.db.
1662351770: Saving in-memory database to /data//mosquitto.db.
1662352367: New connection from 172.27.67.201:51237 on port 1883.
1662352367: Client <unknown> disconnected due to protocol error.
1662352375: New connection from 172.27.67.201:51239 on port 1883.
```

Fig. 9. Denial of service alert

Date/Time	Priority	Classification	Source	Action
05 Sep 2022 15:24:27 (+08:00)	High	Denial of Service	172.27.67.201	Acknowledge Raw Details

Fig. 10. Denial of service alert

A brute force attack was simulated using MQTT explorer, in which unauthorised connection attempts were made five times within 30 s. The detection method is by looking for a specific "client unknown" string in the logs. In Fig. 14, this message is shown after an unsuccessful connection to the broker due to invalid credentials. As these connections do not meet the broker's authentication requirements, they are deemed as unauthorised connections. As a result, if the broker receives a continuous stream of unauthorised connections, this might be a sign that an attacker is trying to brute force the password and username. A brute force attack notification will be flagged in the dashboard, as shown in Fig. 15, if more than 5 connection failures are detected in the logs within 60 s.

4.4 Zigbee2MQTT Devices Inactivity Detection

Device inactivity refers to when a device stops communicating or producing updates to a MQTT broker and the device status is unknown. The possible reasons for device inactivity could be due to devices that are operating in idle or inactive mode until a specific function call activates them, or devices that are actually disconnected from the broker due to connectivity issues with the Zigbee dongle, or even faulty devices. Usually, in such situations, MQTT devices have a default keep-alive feature that ensures the connection between the broker and the MQTT device is still connected by sending PINGREQ and PINGRESP packets.

```
{
  "uuid": "f6fa88b6ae6946cc9de8aaf999a62c6f",
  "datetime": "2022-09-05T15:24:27.194538+08:00",
  "classification": "Denial of Service",
  "priority": "High",
  "src": "172.27.67.201",
  "rule_raw": "tcp any -> $SELF src; name: Denial of Service; priority: 3;
   threshold: 100; interval: 10; cooldown: 10",
  "metadata": {
   "first_violation": "2022-09-05T15:24:17.271353+08:00",
   "total_violation": 433,
   "mqtt": {
    "client_id": [
     "mqtt-explorer-ddf55658"
    ],
    "auth": {
     "username": [
      "mqtt-user"
     ],
     "password": []
    },
    "topics": {
     "publish": {},
     "subscribe": {},
     "unsubscribe": {}
    }
   }
  }
}
```


Fig. 11. Denial of service metadata

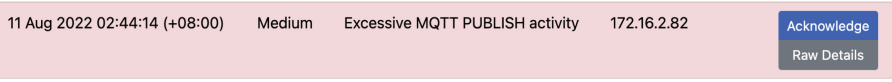

Fig. 12. Excessive MQTT publish activity alert

However, in Zigbee2MQTT devices, the default functionality does not include a keep-alive feature. Instead, Zigbee2MQTT devices use their own availability feature checks to identify between idle devices and disconnected devices (Figs. 11, 13, 16).

```
{
  "uuid": "2397b1fa00754031ac68cc3a5ae51170",
  "datetime": "2022-08-11T02:44:14.036049+08:00",
  "classification": "Excessive MQTT PUBLISH activity",
  "priority": "Medium",
  "src": "172.16.2.82",
  "rule_raw": "mqtt any -> $SELF src; name: Excessive MQTT PUBLISH activity;
   priority: 2; threshold: 20; interval: 5; msg_type: publish",
  "metadata": {
   "first_violation": "2022-08-11T02:44:10.176617+08:00",
   "total_violation": 20,
   "mqtt": {
    "client_id": [
     "ESP32_99C490"
    ],
    "auth": {
     "username": [
      "mqtt-user"
     ],
     "password": []
    },
    "topics": {
     "publish": {
      "smx/device/051001006/single_ts/20220812/151001006Z1 (QOS: 2)": 2,
      "smx/device/051001006/position (QOS: 2)": 9,
      "\"dst\":false, (QOS: 2)": 5,
      "smx/device/051001006/sync_list (QOS: 2)": 4,
      "smx/device/051001006/single_ts/20220812/251001006Z1 (QOS: 2)": 2
     },
     "subscribe": {},
     "unsubscribe": {}
    }
   }
  }
}
```


Fig. 13. Excessive MQTT publish activity metadata


```
1662361168: Client mqtt-explorer-ddf55658 disconnected.
1662362576: Saving in-memory database to /data//mosquitto.db.
1662363001: New connection from 172.27.67.201:54772 on port 1883.
error: received null username or password for unpwd check
1662363001: Client <unknown> disconnected, not authorised.
1662363004: New connection from 172.27.67.201:54773 on port 1883.
error: received null username or password for unpwd check
1662363004: Client <unknown> disconnected, not authorised.
1662363004: New connection from 172.27.67.201:54774 on port 1883.
error: received null username or password for unpwd check
1662363004: Client <unknown> disconnected, not authorised.
1662363005: New connection from 172.27.67.201:54775 on port 1883.
error: received null username or password for unpwd check
1662363005: Client <unknown> disconnected, not authorised.
1662363006: New connection from 172.27.67.201:54776 on port 1883.
error: received null username or password for unpwd check
1662363006: Client <unknown> disconnected, not authorised.
1662363006: New connection from 172.27.67.201:54777 on port 1883.
error: received null username or password for unpwd check
1662363006: Client <unknown> disconnected, not authorised.
1662363007: New connection from 172.27.67.201:54778 on port 1883.
error: received null username or password for unpwd check
```


Fig. 14. Unauthorised connection requests in MQTT log

Fig. 15. Brute force attack alert


```
{
  "uuid": "1572b51303214dd4bf30d3b2499a366c",
  "datetime": "2022-09-05T15:30:06.250113+08:00",
  "classification": "Brute Force (Auth)",
  "priority": "High",
  "src": "172.27.67.201",
  "rule_raw": "mqtt $SELF -> any dst; name: Brute Force (Auth);
    priority: 3; threshold: 5; interval: 60; msg_type: connack; retcode: 5",
  "metadata": {
    "first_violation": "2022-09-05T15:30:01.958344+08:00",
    "total_violation": 5,
    "mqtt": {
      "client_id": [
        "mqtt-explorer-ddf55658",
        "mqttx_88f6220f"
      ],
      "auth": {
        "username": [
          "mqtt-user"
        ],
        "password": []
      },
      "topics": {
        "publish": {},
        "subscribe": {},
        "unsubscribe": {}
      }
    }
  }
}
```


Fig. 16. Metadata of brute force attack

The detection is done by leveraging the availability feature of Zigbee2MQTT and monitoring the MQTT topic used by the device to communicate with Zigbee2MQTT as a fallback. A list of connected devices and their information is retrieved by subscribing to the topic "[zigbee2mqtt]/bridge/devices". In Zigbee2MQTT, the availability feature checks for a response when reading "zclVersion" from the "genBasic" cluster. A fallback mechanism is used if the availability feature is disabled, by subscribing to the device's MQTT topic and monitoring its last publish activity and the device is considered inactive if the topic remains quiet after a set period of time. As shown in Fig. 17, the Zigbee2MQTT device inactivity notification is triggered if the device does not respond to ping and no MQTT activity detected in the last 600 s (Fig. 18).

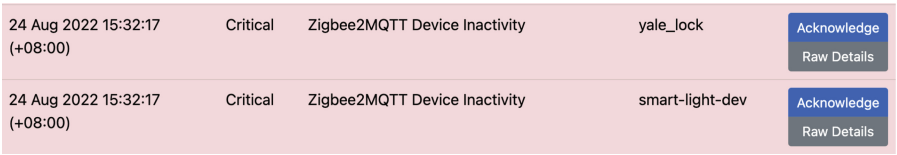

Fig. 17. Device inactivity detection


```
{
  "uuid": "55f62c9aa36a40d584a225e8ea6d8bb5",
  "datetime": "2022-08-24T15:32:17.261909+08:00",
  "classification": "Zigbee2MQTT Device Inactivity",
  "priority": "Critical",
  "src": "yale_lock",
  "rule_raw": "Device not responding to Zigbee2MQTT ping and
               no MQTT activity detected in the last 600 second(s)",
  "metadata": {
    "zigbee2mqtt_device_topic": "zigbee2mqtt/yale_lock",
    "friendly_name": "yale_lock",
    "ieee_address": "0x000d6f0010c9a6bc",
    "definition": {
      "description": "Assure lock",
      "model": "YRD226HA2619",
      "supports_ota": false,
      "vendor": "Yale"
    },
    "last_mqtt_activity": "2022-08-24T15:22:16.486078+08:00",
    "last_ping_activity": null
  }
}
```


Fig. 18. Metadata of device inactivity

4.5 Real-Time Detection Experiment

As per the mentioned experiments conducted above, this experiment for the real-time notification was conducted to benchmark if the proposed framework is reliable for use from the point of attack detection to notification of attacks in the alerts panel by utilizing a standardised benchmark suite to evaluate the experiment. The results are summarized as the following Table 1:

Table 1. Attack detection experiment results

Experiment results				
Attack type	Shortest delay timing	Longest delay timing	Mean delay timing	Number of attempts
Unauthorised Subscription Detection	1 ms	12 ms	6.1 ms	20
Denial-of-Service Attack Detection	1 ms	7 ms	2.85 ms	20
Brute Force Attack Detection	1 ms	9 ms	3.75 ms	20

5 Conclusion and Future Work

To conclude, this paper proposes a possible way to collect MQTT traffic for forensic investigation and analysis. We utilized the existing building management infrastructure and collected real-life MQTT communication for forensic analysis and analysis. Our experiment shows that it is possible to find out cyber-attacks from both MQTT traffic and MQTT logs.

For future work, we are planning to train a machine learning model for cyber-attacks detection. Hence we are going to collect a large MQTT communication dataset for further studies. Since the dataset contains actual BMS traffic, it is valuable to other researchers who would like to study the behavior of BMS communication. We are planning to release the BMS MQTT communication dataset in the coming future.

References

1. Vaccari, I., Chiola, G., Aiello, M., Mongelli, M., Cambiaso, E.: MQTTset, a new dataset for machine learning techniques on MQTT. Sensors **20**(22), 6578 (2020)
2. Yang, Z., Zhou, Q., Lei, L., Zheng, K., Xiang, W.: An IoT-cloud based wearable ECG monitoring system for smart healthcare. J. Med. Syst. **40**(12), 1–11 (2016)
3. Cornel-Cristian, A., Gabriel, T., Arhip-Calin, M., Zamfirescu, A.: Smart home automation with MQTT. In: 2019 54th International Universities Power Engineering Conference (UPEC), pp. 1–5. IEEE (2019)
4. Tang, K., Wang, Y., Liu, H., Sheng, Y., Wang, X., Wei, Z.: Design and implementation of push notification system based on the MQTT protocol. In: 2013 International Conference on Information Science and Computer Applications (ISCA 2013), pp. 116–119. Atlantis Press (2013)
5. Pooja, S., Uday, D., Nagesh, U., Talekar, S.G.: Application of MQTT protocol for real time weather monitoring and precision farming. In: 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), pp. 1–6. IEEE (2017)
6. Atmoko, R., Riantini, R., Hasin, M.: IoT real time data acquisition using MQTT protocol. J. Phys.: Conf. Ser. **853**(1), 012003 (2017)
7. Chen, F., Huo, Y., Zhu, J., Fan, D.: A review on the study on MQTT security challenge. In: 2020 IEEE International Conference on Smart Cloud (SmartCloud), pp. 128–133. IEEE (2020)
8. Andy, S., Rahardjo, B., Hanindhito, B.: Attack scenarios and security analysis of MQTT communication protocol in IoT system. In: 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), pp. 1–6. IEEE (2017)
9. Wong, H., Luo, T.: Man-in-the-middle attacks on MQTT-based IoT using BERT based adversarial message generation. In: KDD 2020 AIoT Workshop (2020)
10. Vaccari, I., Aiello, M., Cambiaso, E.: SlowTT: a slow denial of service against IoT networks. Information **11**(9), 452 (2020)
11. Budiana, M.S., Negara, R.M., Irawan, A.I., Larasati, H.T.: Advanced detection denial of service attack in the internet of things network based on MQTT protocol using fuzzy logic. Register: J. Ilmiah Teknol. Sist. Inform. **7**(2), 95–106 (2021)

12. Alaiz-Moreton, H., Aveleira-Mata, J., Ondicol-Garcia, J., Muñoz-Castañeda, A.L., García, I., Benavides, C.: Multiclass classification procedure for detecting attacks on MQTT-IoT protocol. Complexity **2019** (2019)
13. Ciklabakkal, E., Donmez, A., Erdemir, M., Suren, E., Yilmaz, M.K., Angin, P.: Artemis: an intrusion detection system for MQTT attacks in internet of things. In: 2019 38th Symposium on Reliable Distributed Systems (SRDS), pp. 369–3692. IEEE (2019)
14. Hindy, H., Bayne, E., Bures, M., Atkinson, R., Tachtatzis, C., Bellekens, X.: Machine learning based IoT intrusion detection system: an MQTT case study (MQTT-IoT-IDS2020 dataset). In: Ghita, B., Shiaeles, S. (eds.) INC 2020. LNNS, vol. 180, pp. 73–84. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-64758-2_6
15. Chunduri, N.V.H., Mohan, A.K.: A forensic analysis on the availability of MQTT network traffic. In: Thampi, S.M., Wang, G., Rawat, D.B., Ko, R., Fan, C.-I. (eds.) SSCC 2020. CCIS, vol. 1364, pp. 262–274. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-0422-5_19
16. Anthraper, J.J., Kotak, J.: Security, privacy and forensic concern of MQTT protocol. In: Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM). Amity University Rajasthan, Jaipur (2019)
17. Dinculeană, D., Cheng, X.: Vulnerabilities and limitations of MQTT protocol used between IoT devices. Appl. Sci. **9**(5), 848 (2019)
18. Husnain, M., et al.: Preventing MQTT vulnerabilities using IoT-enabled intrusion detection system. Sensors **22**(2), 567 (2022)
19. Introduction | Socket. IO. https://socket.io/docs/v4/
20. Yara, A.: Preventing vulnerabilities and MitigatingAttacks on the MQTT protocol (2020)
21. Syed, N.F., Baig, Z., Ibrahim, A., Valli, C.: Denial of service attack detection through machine learning for the IoT. J. Inf. Telecommun. **4**(4), 482–503 (2020)
22. Buccafurri, F., De Angelis, V., Nardone, R.: Securing MQTT by blockchain-based OTP authentication. Sensors **20**(7), 2002 (2020)
23. Agazzi, A.E.: Smart home, security concerns of IoT. arXiv preprint arXiv:2007.02628 (2020)