Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs

Harsha M S
Student, Department of CSE
PESIT – Bangalore South Campus
Bangalore, India
sridharsha598@gmail.com

Bhavani B M
Student, Department of CSE
PESIT – Bangalore South Campus
Bangalore, India
bhavani.bm97@gmail.com

K.R.Kundhavai
Assistant Professor, Dept. of CSE,
PESIT – Bangalore South Campus
Bangalore, India
kundhavai@pes.edu

Abstract— Among the technologies evolved in the recent years, a remarkable one is the IoT (Internet of Things), wherein the 'thing' in IoT could be smart phones, tablets, PCs and almost anything with a sensor on it like cars, people, machines in production plants, jet engines, oil drills, wearable devices and many more objects. A standardized, light-weight, session layer protocol with publish/subscribe architecture widely used for messaging and information exchange among IoT devices is the MQTT (MQ Telemetry Transport) protocol. In this paper, we identify various security loopholes in MQTT, using Shodan API and implementing an experimental setup on a Raspberry Pi as an MQTT Broker and python programs as publisher/subscriber clients. The experimental results with respect to the security issues in this protocol at packet and topic levels were studied and the corresponding security measures, consisting of authentication and authorization techniques (ACLs) were implemented. As a result, the Broker was then found to be immune to such attacks. This paper is a concise study of security inconsistencies in MQTT and its countermeasures.

Keywords—IOT, MQTT, Topics, Wildcards, security countermeasures, MQTT Broker, Wireshark, Shodan

I. INTRODUCTION

Internet of Things refers to the universal concept of things, i.e. everyday objects that are readable, recognizable, locatable, addressable, and/or controllable by means of the Internet [1]. Things can be connected through RFID, WLAN, WAN, or various other means. Day to day objects include not only the electronic gadgets or devices but also higher order technological products like machineries, vehicles, and also non-electronic things like processes, people, data which leads to Internet of Everything (IoE). There are many protocols existing and many under research with respect to IoT. Since it is a recent and emerging technology, security issues are increasing at a phenomenal degree on every tiny aspect right from data transfer protocols to connection protocols.

II. IOT PROTOCOLS

A. Protocols at different layers

There are different IoT protocols at different layers. Few protocols in the datalink layer are Zigbee Smart, Bluetooth Low Energy, LoRaWAN, WiFi, 802.11ah, etc. Some of the network layer encapsulation protocols are 6LoWPAN, 6Lo, IPv6 over Bluetooth Low Energy, etc. and session layer protocols are MQTT, SMQTT, AMQP, CoAP, DDS, XMPP. Since the paper focuses on security flaws in MQTT, the following topic is the discussion on session layer protocols.

B. Session Layer Protocols in IOT

1. **MQTT:** MQ Telemetry Transport is a standardized protocol designed to provide embedded connectivity between applications and middleware on one side and networks and communications on the other side [2]. Its goal is to collect data from many devices and transport that data to the IT infrastructure [3]. There are three elements in the architecture. 1. Publishers, 2. A Broker, and 3. Subscribers. The Publishers which can be a PC, Smartphone, a Sensor, etc., connects to the Broker to publish messages that they hold and goes back to sleep**.** The Subscribers, are applications/devices that register themselves with the Broker. In order to determine which message gets to which subscriber, MQTT uses 'Topics'. A Topic is a hierarchical structured string, which is used for message filtering and routing [4]. The MQTT Broker allows usage of wildcards in the topic names to grant access to other topics in the hierarchy at two levels: single level (+) and multi-level (#).
2. **SMQTT:** Secure MQTT[5] introduces encryption/ decryption to enhance security. The steps in SMQTT are setup, encryption, publish and decryption.
3. **AMQP:** Advanced Messaging Queuing Protocol [6] is all about queues. It helps in sending transactional

messages between servers, mainly in banking industry where thousands of queued transactions are processed.

4. **DDS:** Data Distribution Service is used by high-performance integrated devices. It is the only technology that delivers the flexibility, reliability and speed necessary to build complex, real-time applications [3]. Applications include military systems, wind farms, hospital integration, medical imaging, asset-tracking systems, and automotive test and safety. DDS connects devices together into working, distributed applications at physics speeds [3].
5. **CoAP:** The Constrained Application Protocol[7], uses a standard interface named Representational State Transfer (REST) between HTTP client and servers. REST runs on UDP and could result in significant overhead and power consumption because of its working nature.
6. **XMPP:** Extensible Messaging and Presence Protocol[8] is now extensively used in Software-Defined-Networks (SDN) which incorporates XML technology and can be used for both publish/subscribe architecture and client/server architecture.

For a survey of various application layer protocols for IoT see: [9].

III. MQTT ESSENTIALS

MQTT has been gaining momentum in recent years because of its nature to work with low powered devices. This protocol can transmit only 256 MB of data per message hence less affected by latency [10]. Since the data is small, transmission is quick and efficient. Messages are assigned at QoS level. There are three QoS levels:

1. Level-0: At most once
2. Level-1: At least once
3. Level-2: Exactly once

The QoS levels guarantees the delivery of messages between the sender and the receiver. The implementation details can be referred from [11].

A. Challenges identified in MQTT

i) Any client can publish or subscribe to any topic(s). There are no validation or verification techniques that authenticates the user to avail the services provided by the Broker and authorizes the user to the topics.

ii) In MQTT, if a subscriber forgets to collect the message from the Broker, the data stays in the Broker forever. Hence the Broker is overloaded and the performance degrades. [12]

iii) MQTT runs on TCP and security is most often provided by the Brokers based on SSL/TLS (Secure Socket Layer/Transport Layer Security). TLS provides security only on point to point basis.

iv) MQTT doesn't prioritize the messages. For example, ambulance services system messages are more important at a particular instance.

v) Ordering and resending of messages which are lost during transmission is a biggest challenge. [7]

vi) Handling huge data that is generated by all the IoT devices is a tough task.

vii) There is no access control mechanism to prevent any client from subscribing and/or publishing data to any topic(s) that it wishes to.

viii) The usage of wildcards in the topic names, can be exploited to steal data.

ix) DDoS attack can be launched by identifying the most subscribed topic and flooding the Broker with high QoS messages on that topic.

B. Countermeasures

- Authentication of the publisher and the subscriber by means of digital certificates, digital signatures and Encryption and Decryption of a Nonce. Encryption algorithms are to be chosen based on the need.
- Security enhancement by implementing IPSec tunnel mode for transferring the payload rather than TLS/SSL. [13]
- Timestamps can be included in the messages so that they don't stay in the Broker forever if unclaimed by any subscriber.
- Access Control Lists (ACLs) provide a mechanism by which the user access can be restricted.
- Tons and tons of data generated by the IoT devices may lead to many attacks. Hence a need of implementing Intrusion Detection Systems (IDS) and Intrusion Prevention Systems can help prevent major disasters.
- DDoS attacks are one of the threatening attacks in IoT network. A method of thwarting such an attack could be the deployment of a cache at the Broker to store, monitor and analyze the subscribers' and publishers' activities.
- Higher level QoS leads to flooding of messages and hence the Broker is overloaded. Therefore lower level QoS can be used as the default level to partially reduce this effect.

IV. METHODOLOGY

The experimental setup as shown in Fig. 1 includes a Raspberry Pi which hosts the MQTT Broker and publisher, subscriber clients implemented using python programs running on another computer. Raspberry Pi is a single board computer which runs on a Linux based OS. Eclipse Mosquitto, an open source MQTT Broker with support for versions 3.1 and 3.1.1, was implemented on the Raspberry Pi. The high level interpreted programming language Python was used to implement the Client (Both Publisher and Subscriber). Python provides APIs for communication with a MQTT Broker via the MQTT protocol. The open-source packet analyzer

Wireshark has been used to analyze the MQTT packets in the network.

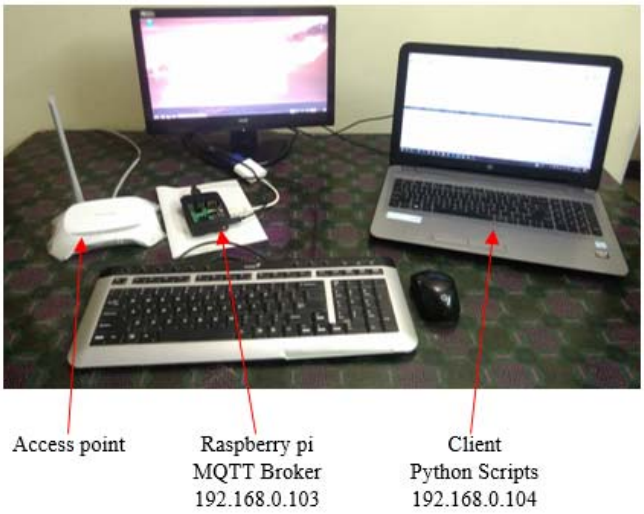

Fig. 1

The log file at the Broker end was used to record the response of the Broker to various kinds of attacks. The Broker has been implemented on Raspberry Pi and tested for various possible attacks. The mechanisms to prevent the attacks were implemented and the responses of the Broker were recorded and analyzed.

Source	Destination	Protocol	Length	Info
192.168.0.104	192.168.0.103	MQTT	68	Connect Command
192.168.0.103	192.168.0.104	MQTT	58	Connect Ack
192.168.0.104	192.168.0.103	MQTT	67	Subscribe Request
192.168.0.103	192.168.0.104	MQTT	59	Subscribe Ack

Fig. 2

Fig. 2 shows the sequence of actions when a client tries to subscribe to a topic. The client sends a CONNECT packet along with its id and credentials if required. The Broker on receiving it, sends the CONNECT ACK packet with the acknowledgement. The client then sends the SUBSCRIBE REQUEST with the topic name. The Broker on receiving it, adds the client to the list of subscribers it has and then sends SUBSCRIBE ACK with the acknowledgement.

Source	Destination	Protocol	Length	Info
192.168.0.104	192.168.0.103	MQTT	83	Connect Command
192.168.0.103	192.168.0.104	MQTT	58	Connect Ack
192.168.0.104	192.168.0.103	MQTT	70	Publish Message

Fig. 3

Fig. 3 shows the sequence of actions when a client tries to publish data to topic(s). The CONNECT and CONNECT ACK have the same meanings as in publishing. The client here uses QoS level 0 (Fire and forget) to publish data and hence there is no response from the Broker once the message has been sent from the client. The client sends the PUBLISH message along with the topic name and the payload. The Broker on receiving this message, forwards it to the subscribers of the topic.

V. SECURITY MECHANISMS

Security on a higher level can be implemented by two mechanisms: *Authentication* and *Authorization*. Authentication is a mechanism by which an entity proves its identity to the other party, whereas authorization is a mechanism by which the entity's credentials are used to either allow or deny access to the services provided.

For details on security perspective and challenges on IoT as a whole, see: [14].

This section describes the various loopholes and their corresponding security measures.

A. No Authentication Mechanism to authenticate users

When there is no Authentication mechanism in place, any client can publish and or subscribe to any topic. This poses a threat on the data confidentiality. The data that the Broker is dealing with, might have sensitive information and this information can be obtained by just providing the name of the topic. Authentication helps in the prevention of this attack by allowing only the registered entities to publish and subscribe data.

Authentication can be performed by different methods. Username and Password is the most common form of authentication that is used, Client certificates can also be used to prove its identity.

The challenge in using certificates for authentication is in having a process that takes care of the certificate provisioning (providing clients with certificates) and certificate revocation (to invalidate certain client certificates if found to be invalid or compromised). Hence there is a need of infrastructure for dealing with the aforementioned processes, which might not be feasible in many cases.

```
MQ Telemetry Transport Protocol, Connect Ack
  Header Flags: 0x20 (Connect Ack)
  Msg Len: 2
  Acknowledge Flags: 0x00
  Return Code: Connection Refused: not authorized (5)
```

Fig. 4

Fig. 4 shows the dissected CONNECT ACK of a client trying to access without valid credentials.

B. User Credentials sent as plain text in CONNECT Command Packet

The MQTT protocol was just designed with keeping connectivity in mind and not much attention was paid in securing the protocol. Enabling Authentication prevents unauthorized users from accessing the data, but the user credentials are passed in the form of plain text and this packet can be used by an attacker to impersonate the legitimate user and gain access to data.

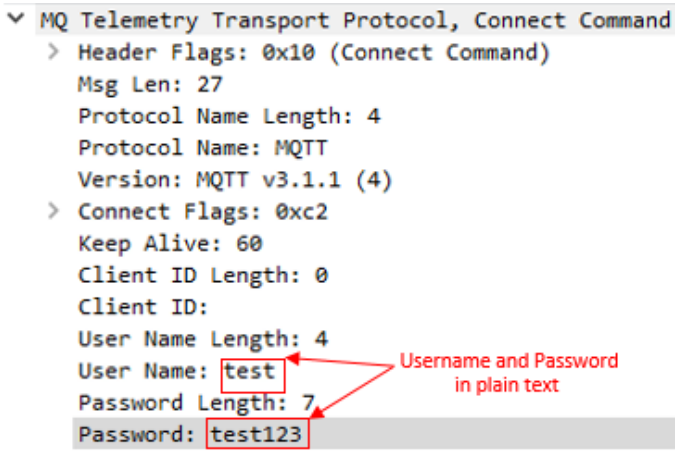

Fig. 5

Fig. 5 shows the dissected CONNECT COMMAND packet revealing the User name and Password in plain text.

Fig. 6 shows a possible attack scenario, where the attacker gets hold of the CONNECT packet and can use it to obtain the credentials to impersonate the legitimate client. The poor client is unaware of this attack.

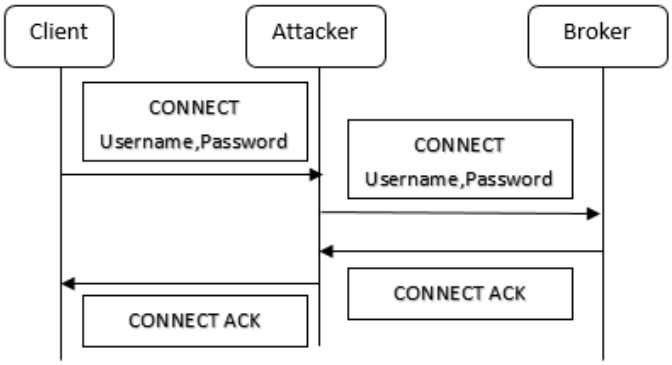

Fig. 6

This vulnerability in the protocol can be surpassed using the following mechanisms:

- Using Transport Layer Security or Secure Sockets Layer over port 8883(standard port used for secured MQTT connection), the packets between the client and the Broker are sent via a secure communication channel or tunnel as shown in Fig. 7.

  The challenge in using this mechanism is that, if the client performs frequent reconnects, then the overhead required in resuming the session would be really significant. Hence it is not recommended to use this mechanism in cases of frequent reconnects.
- The payload can be encrypted before it is sent on the network. This scheme requires the publisher to encrypt the message before sending it to the Broker. The decryption can be done either at the Broker or at the subscriber. Encryption and decryption can be done using light-weight cryptographic algorithms.

Another alternative would be to use SMQTT with your own choice of the encryption algorithm.

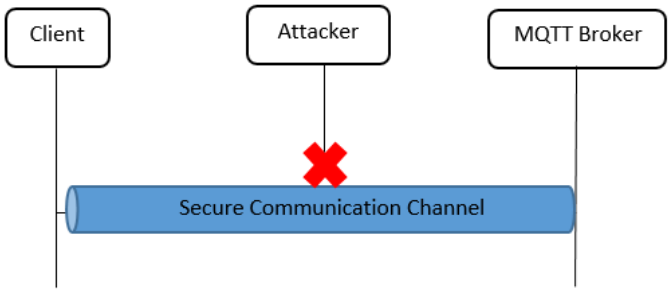

Fig. 7

- Message Authentication Code (MAC) based approaches can be used to overcome many challenges faced due to the public key cryptosystem. One such approach is mentioned in [15].

C. *Use of wildcards in topic names pose a threat to data security*

Say for instance we have a Broker which maintains the various topics in the hierarchy (ex: home/room1/temperature) as shown in Fig. 8.

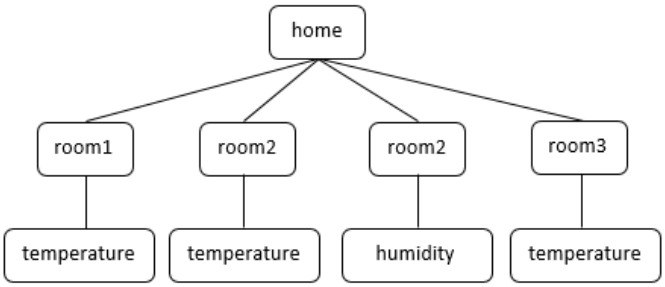

Fig. 8

A user can make use of the wild cards in topic names supported by the Broker to publish/subscribe data that he is not entitled to. With the topic name as 'home/+/temperature', the user will be able to access the temperature data of room 1, 2 and 3; similarly, with the topic name as 'home/#', the user will be able to access all the data in the hierarchy.

To prevent unauthorized data access, the authorization mechanisms are used to allow/deny access to the services.

Authorization can be implemented at three levels:

- *Per topic*: The user is given access to just the topics that the administrator allows him/her to access. This mechanism allows to prevent the unauthorized data access that can be done using the wild cards.
- *Per method*: Even after the user is given access to just the required topic(s), the user can still do both publish and subscribe, which is not favourable in many situations. This mechanism allows the user access to either publish or subscribe or both.

- *Per Quality of Service (QoS)*: MQTT Protocol allows three levels of QoS, and hence limitations can be imposed as to which level of QoS a particular user is entitled to.

The aforementioned mechanisms can be implemented using Access Control List (ACL), where the user name along with the restrictions is mentioned.

Fig. 9 depicts the log details in the MQTT Broker that has the ACL mechanism enabled, wherein a user tries to publish data (QoS 0) to the topic he/she is not allowed to. The first box shows the case when the user publishes data to the topic that he/she is entitled to, and when the user tries to publish data to a topic that he/she is not allowed to publish, it is straight away rejected by the Broker as indicated in the second box.

This work of identifying the vulnerabilities and proposing their associated security countermeasures is in progress.

There are other approaches that are used to identify the vulnerabilities, one such approach is the Novel Fuzzing Approach [16]. While our approach identifies the various loopholes that could be exploited by the attackers, the Novel Fuzzing approach can be used to observe the system's behaviour when it is fed with random inputs. This information can then be used to find the appropriate countermeasures.

```
pi@raspberrypi:~ $ sudo mosquitto -p 1883 -v -c /etc/mosquitto/mosquitt
.conf
1525012156: mosquitto version 1.3.4 (build date 2017-05-29 22:25:09+000
) starting
1525012156: Config loaded from /etc/mosquitto/mosquitto.conf.
1525012156: Opening ipv4 listen socket on port 1883.
1525012156: Opening ipv6 listen socket on port 1883.
1525012161: New connection from ::1 on port 1883.
1525012161: New client connected from ::1 as mosqpub/1943-raspberryp (c
, k60, utest).
1525012161: Sending CONNACK to mosqpub/1943-raspberryp (0)
1525012161: Received PUBLISH from mosqpub/1943-raspberryp (d0, q0, r0,
0, 'test', ... (5 bytes))
1525012161: Received DISCONNECT from mosqpub/1943-raspberryp
1525012178: New connection from ::1 on port 1883.
1525012178: New client connected from ::1 as mosqpub/1944-raspberryp (c
, k60, utest).
1525012178: Sending CONNACK to mosqpub/1944-raspberryp (0)
1525012178: Denied PUBLISH from mosqpub/1944-raspberryp (d0, q0, r0, m0
'test1', ... (5 bytes))
1525012178: Received DISCONNECT from mosqpub/1944-raspberryp
```

Fig. 9

VI. RESULTS

Shodan and its Python API has been used for the analysis wherein, Shodan is a search engine that crawls the Internet to find publicly accessible devices (computers, servers, etc.) using various filters (like protocol name, port numbers, etc.). The results are explained below.

Fig. 10 shows the total number of the publicly accessible devices that use the MQTT protocol. It also shows the distribution of the servers hosting MQTT Broker around the globe.

Fig. 11 shows the top services that deploy MQTT protocol. It is evident that 99.71% of the devices use port 1883 for communication via the MQTT Protocol. The rest of the devices use services, ports like HTTPS, Port 8081, Port 8883, etc. We used this information to identify servers that allow unauthorized access. For the analysis we used the Port 1883, and a sample of 200 devices out of 53396 devices. Here the client tries to subscribe to a topic named 'random' without specifying the user name and the password, the Broker on receiving the CONNECT message, might perform certain checks and then returns a Connection code in the CONNECT ACK.

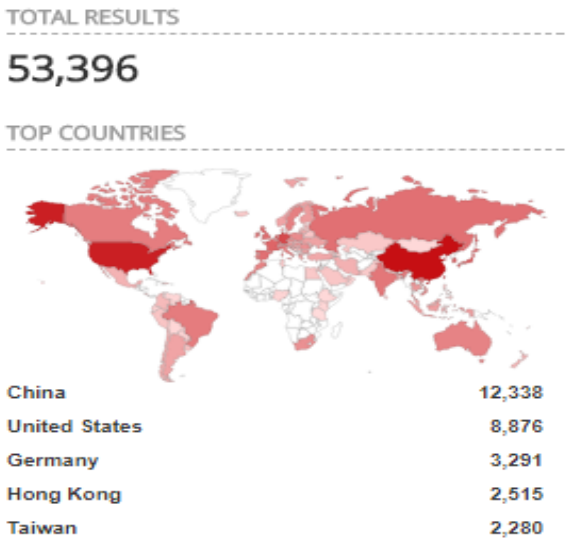

Fig. 10

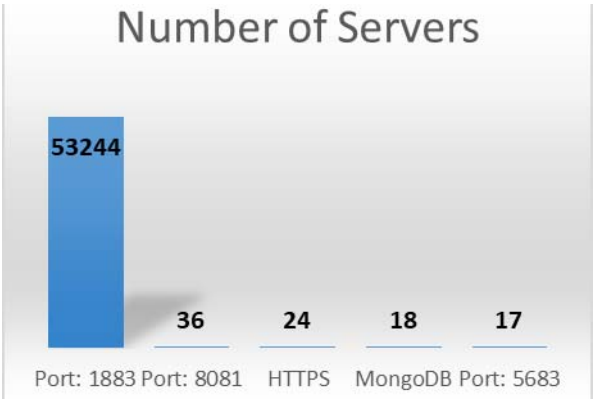

Fig. 11

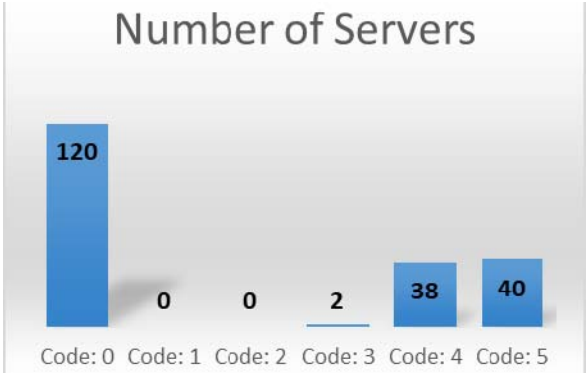

Fig. 12

Code	Info
0	Accepted
1	protocol version unacceptable
2	Identifier Rejected
3	Server unavailable
4	Bad User name or password
5	Not Authorized

Fig. 13

Fig. 12 and Fig.13 shows the result and the interpretation of the connection codes respectively. From the graph it is evident that 60% of the servers allow unauthorized access to topics i.e. a client can publish or subscribe to any topic without the credentials. Authentication being the first line of defense is not enabled in most of the servers, which can be exploited by attackers to perform various kind of attacks.

A python program to connect to one of these unauthorized servers was executed with the topic name '$SYS/#', this topic returns all the system related data. The result of the analysis is shown in Fig. 14, details like the Server version, the number of clients, number of messages, etc. are revealed. A similar mechanism can be used to steal all the data that passes via the Broker.

```
C:\Users\Harsha Sridhar\Documents\PythonMQTT>python tSub.py
Connected with result code 0
$SYS/broker/version: mosquitto version 1.4.15
$SYS/broker/timestamp: Sat, 07 Apr 2018 11:14:23 +0100
$SYS/broker/changeset: $Revision: e745e1ab5007 $
$SYS/broker/uptime: 1036255 seconds
$SYS/broker/messages/stored: 55
$SYS/broker/messages/received: 21969
$SYS/broker/messages/sent: 23884
$SYS/broker/messages/per second/received: 1
$SYS/broker/messages/per second/sent: 0
$SYS/broker/clients/total: 1
$SYS/broker/clients/inactive: 0
$SYS/broker/clients/active: 1
$SYS/broker/clients/maximum: 3
$SYS/broker/clients/disconnected: 0
$SYS/broker/clients/connected: 1
$SYS/broker/clients/expired: 0
$SYS/broker/heap/current size: 7400 bytes
$SYS/broker/heap/maximum size: 11376 bytes
$SYS/broker/heap/current: 18446744073709509744
$SYS/broker/heap/maximum: 18446744073709551608
$SYS/broker/bytes/received: 420388
$SYS/broker/bytes/sent: 564536
$SYS/broker/bytes/per second/received: 33
$SYS/broker/bytes/per second/sent: 7
$SYS/broker/subscriptions/count: 1
$SYS/broker/retained messages/count: 55
$SYS/broker/publish/messages/dropped: 0
```

Fig. 14

VII. CONCLUSION

Internet of Things (IoT) enables extension of capabilities of all kinds of devices by allowing them to connect and exchange data with other devices on the network. It is estimated that about 30 million devices will be connected to the Internet by 2020, which would drastically change our lifestyles. But there is a huge need to make sure that these devices that communicate via the Internet are safe from security vulnerabilities.

This paper talks about the various protocols used in IoT, but focuses on the widely used MQTT protocol, due to its features like small header size, low bandwidth requirements, efficiency and reliability in adverse conditions etc.

From the experiments that we have conducted on the MQTT protocol, it was evident that it consisted of its own set of vulnerabilities that can be exploited by attackers. Using Shodan search engine and Python APIs, we discovered that most of the servers hosting the MQTT Broker didn't have the authentication mechanism in place, which can be exploited to steal sensitive data or add redundant data which in turn could affect all the other systems that rely on this data.

Once the vulnerabilities were identified, security mechanisms were implemented on a Raspberry Pi serving a Broker and its configuration files were modified. Moreover, Access Control List (ACL) were created to ensure that the users are only given access to the data that they are entitled to. Once these security mechanisms were imposed on the Broker, all the attacks were rendered unsuccessful.

Therefore while implementing a Broker, these security mechanisms could be used in the initial phase to prevent the aforementioned attacks. Security should therefore be factored in at the inception, and not as an afterthought.

VIII. REFERENCES

[1] http://www.prismtech.com/sites/default/files/documents/Messaging-Whitepaper-051217.pdf

[2] https://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_prot/#MQTT

[3] http://www.electronicdesign.com/iot/understanding-protocols-behind-internet-things

[4] https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment

[5] Meena Singh, M.A. Rajan, V.L. Shivraj, "Secure MQTT for Internet of Things (IoT)", Fifth International Conference on Communication Systems and Network Technologies 2015.

[6] Steve Vinoski,"Advanced Message Queuing Protocol", IEEE Internet Computing (Volume: 10, Issue: 6, Nov.-Dec. 2006).

[7] Carsten Bormann, Angelo P. Castellani, Zach Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes", IEEE Internet Computing (Volume: 16, Issue: 2, March-April 2012).

[8] Micheal Kirsche, Ronny Klauck, “Unify to Bridge Gaps: Bringing XMPP into the Internet of Things”, Work in Progress(WIP) Session of PerCom(2012).
[9] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, Jesus Alonso-Zarate, “A Survey on Application Layer Protocols for the Internet of Things”, Transaction on IoT and Cloud Computing 2015.
[10] http://www.thinglogix.com/why-the-mqtt-protocol-is-ideal/
[11] http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[12] Dipa Soni ,Charotar University of Science and Tech and Ashwin Makwana, Charotar University of Science and Tech, “A Survey on MQTT: A Protocol of Internet Of Things(Iot)”- International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT - 2017).
[13] Harshal Sardeshmukh, Sardar Patel Institute of Technology and Dayanand Ambawade, Sardar Patel Institute of Technology; “Internet of Things: Existing Protocols and Technological challenges in security”, 2017 International Conference on Intelligent Computing and Control.
[14] Qi Jing, Athanasios V. Vasilakos , Jiafu Wan ,Jingwei Lu , Dechao Qiu, “Security of the Internet of Things: perspectives and challenges”, Wireless Netw - Springer Science+Business Media New York 2014.
[15] G. Gaubatz, J.-P. Kaps, and B. Sunar, “Public Key Cryptography in Sensor Networks—Revisited,” Lecture Notes in Computer Science, pp. 2–18, 2005.
[16] S. Hernández Ramos, M. T. Villalba, and R. Lacuesta, “MQTT Security: A Novel Fuzzing Approach,” Wireless Communications and Mobile Computing, vol. 2018, pp. 1–11, 2018.