DDoS Attack on MQTT based IoT network using Amazon Web Services IoT

Ravichandra Malapati*
Syracuse University
Syracuse, NewYork
United States-13210
rmalapat@syr.edu

Vir V. Phoha†
Syracuse University
Syracuse, New york
United States-13210
vvphoha@syr.edu

ABSTRACT

IoT has shown great promise to build a connected life with a relative ease of accessing any device through the Internet. The MQTT *(Message Queuing Telemetry Transport)* protocol, which can connect resource constrained devices to the Internet, is expanding to various applications as it is supported by major cloud service providers with new products like Amazon IoT suite and Microsoft Azure IoT hub. We present a DDoS attack exposing the vulnerability of MQTT protocol. Through an unprotected device, which can be a common scenario in IoT, we were able to do a DDoS by manipulating the number of clients and message payload size. Results using three attacker systems, each generating multiple clients, show the successful execution of the attack thereby denying the service to genuine clients.

CCS Concepts

•**Security and privacy** → **Denial-of-service attacks;** *Distributed systems security;*

Keywords

MQTT ; DDoS in IoT; AWS IoT; Amazon Web Services IoT

1. INTRODUCTION

Over the past few years IoT *(Internet of Things)* has gained popularity and the usage of IoT devices is expected to grow. Few of the many factors contributing to the deluge of IoT applications are comparably low cost of IoT devices, painless integration and low power consumption. MQTT *(Message Queuing Telemetry Transport)* [7] protocol makes the communication with these devices light weight and secured. When using these devices in remote locations, the threat posed by adversaries who steal the information from these devices physically can not be ignored.

*First Author
†Advisor

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

MQTT provides username/password and SSL/TSL based authentication to secure IoT devices. However, these authentication methods may not secure the IoT network in case of DoS *(Denial of Service)* and DDoS *(Distributed Denial of Service)* attacks using the stolen security credentials. So far, there has not been any extensive research on using the stolen certificates to conduct DDoS attacks in a MQTT protocol based IoT network.

In this paper we demonstrate the vulnerabilities to DoS/DDoS of one of the contemporary protocols MQTT which is used in IoT applications. This is also one of the protocols supported by major cloud service providers such as AWS *(Amazon Web Service*) IoT suite and Microsoft Azure IoT hub. AWS IoT suite is cloud based MQTT broker [5] supporting Cloud to Device (C2D) and Device to Cloud (D2C) communication. AWS IoT broker implements MQTT 3.1.1 version protocol.

We setup AWS IoT MQTT broker in section 4.1.1 and use this to demonstrate how an attacker can get access to the end destination address of the AWS IoT MQTT broker. End destination address is the unique address of MQTT broker provided by AWS IoT to publish and subscribe topics. Using this end destination address, in Section 4.2 of this paper, we show how attacker can perform DDoS (Distribute Denial of Service) attacks. To perform DDoS attacks in section 4.2, we use the locally setup server system which runs Mosquitto MQTT broker. This setup is described in Section 4.1.2

2. REVIEW OF DDOS IN IOT AND SECURITY

In IoT, cyber-attacks not only steal and debilitate the services but also trigger actions that are lethal. While DDoS attacks on Internet application layer and cloud computing have been well studied [8] [9] [4],very less research has been carried out for DDoS attacks on IoT [2]. This lack of research is perhaps because of limited use of IoT at this time. As IoT usage is expected to multiply in the coming years, the type of DDoS attacks that we propose can have a large adverse effect on IoT networks. To better understand our attacks, we briefly describe volume and protocol based attacks because, our attacks falls in these categories.

2.1 Volume based Attacks

In IoT, Volume based attacks [12] are the basic DDoS attack where the attacker sends colossal number of requests to the IoT network. There is a possibility that various components of IoT network such as Mosquitto MQTT Broker[5], MQTT Client[3], [7] can get affected due to these requests. As the memory queue available in Mosquitto MQTT Broker

is limited it can overflow easily.

Since the IoT network is heterogeneous in nature where different kinds of devices having different capability such as a MQTT client can act as MQTT broker on request can control and program other MQTT clients. Which means the vulnerability is increased as each device has multiple security flaws. Apart from being vulnerable to the DDoS there is also possibility that the IoT devices can be used to conduct the DDoS attacks by using these devices in botnet.

2.2 Protocol Attacks

The basic protocol attacks[6] target the server resource rather than the bandwidth, much advanced protocol attacks target the communication equipment including MAC, firewalls and load balancers. This attack exploits the generic design of any protocol. We use the attack models as described in Section 3 that fall in both volume and protocol based attacks.

3. PROPOSED ATTACK MODELS

We exploit the MQTT broker through four models (given in the next subsections 3.1, 3.2, 3.3, 3.4) of DDoS attacks as given below. These four models of attacks exploit the MQTT broker using two strategies. The first strategy overwhelms the number of client's connections, thereby disconnecting the genuine clients because, we use all the available resources of the MQTT broker.

The second strategy exploits the MQTT broker in terms of data processing. Here we alter the message payload size while generating the clients using sdkperf tool [11]. We maximize the message payload size for each client message, thereby keeping the MQTT broker busy processing large messages resulting in denying service for genuine clients. Here, we use two payload sizes 100 byte and 25Kbyte message payloads. The detailed specifications of the following attack models can be found in tables 1 and 2.Description of the four attack models is in the following sections.

In tables 1 and 2 Genuine client Message rate is the number of messages published by the genuine clients, however it is up to MQTT broker whether it honors the request of client or not. If the message is not acknowledged by the broker, then the client keeps on sending the same message with latency. However, this depends on the quality of service that is requested by publisher. We have three levels of QoS(Quality of Service), 0,1 and 2. If QoS of a message is 0 then message will be published at most once, if the QoS is 1 then the message will be published at least once and if the QoS is 2 then the message will be published exactly once. We request for 1000 messages per second for genuine clients to conduct DDoS attacks. Genuine client message size is the payload size that a client publishes. In all the four models of attacks we use 100Bytes as well as 25KBytes. Number of genuine publisher and subscriber clients are the number of clients that exist in IoT network before attack is performed. In all the four models of attacks we use 1, 10, 20, 40, 60, 80 and 100 publisher and subscriber clients.

Attacker message rate is the number of messages that an attacker client publish and subscribe to MQTT broker. We set the attacker message rate to 10,000 messages per second for Multiple attacker models and 100,000 for single attacker models.

Number of attacker publisher/subscriber clients are the number of clients that an attack involves. Based on the model of the attack this number changes. For SP/SA we use single attacker publisher client and single attacker subscriber client. For MP/SA we use 3 publisher attacker clients and 2 subscriber attacker clients. Whereas for MP/MA and SP/MA the number of attacker cleints vary and we use 600 attacker clients to cripple the MQTT broker.

3.1 Single Path Single Attacker(SP/SA)

When we say single path, that means publish and subscribe requests are originated from single ip address. In this model of attack we implement single client using single attacker system. This attack comes under DoS as the attacker is using single system. We conduct the attacks with two different message sizes, one with 100 Bytes of message size and another with 25KB message size. We request maximum number of publish/subscribe requests per second for single client, however the acceptance of the puslish requests depends on the network load and MQTT broker load. Which means the single attacker client requests maximum number of publish requests to the MQTT broker, thus keeping the MQTT broker busy.

3.2 Single Path Multiple Attacker(SP/MA)

In this model of attack we implement multiple clients using single attacker system. 600 clients are generated,even though the number of clients that an attacker system support depends on the number of threads a single process can use. The number of threads that a single process can support completely depends on the operating system and hardware. In our case we are able to generate 1017 clients per attacker system. This attack also comes under DoS attacks rather than DDoS attacks. As the distribution of the number of clients is concentrated at single system.

3.3 Multiple Path Single Attacker(MP/SA)

In multiple path single attacker we use more than one system and create MQTT clients in all the systems. In this paper we limit the number of paths to 3 which means all the attacker clients are originated from three addresses. Single attacker defines that each path will have only single client system. This is DDoS attack as the attacker clients are distributed in different systems or paths. The maximum number of clients that each system can generate is one, hence we use only three cleints for this model of client.

3.4 Multiple Path Multiple Attacker(MP/MA)

In multiple path multiple attacker model the attack will be more intense and can freeze the complete IoT network. We generate 200 clients per attacker system and in total 600 clients connections from three attacker systems. This comes under DDoS attack as there are 600 client connections coming from three attacker system each having 200 clients. As we discussed earlier each attacker system is capable of generating more than 200 clients, but for optimal demonstration of DDoS attacks we use 200 clients per system.

4. ATTACK EXPERIMENT

We setup a local MQTT broker which acts as our target to demonstrate all the four models of attacks 3.1,3.2, 3.3, 3.4. The design of this system is discussed in detail in the section 4.1. On successful execution, we show the MQTT IoT network is vulnerable to the DoS and DDoS attacks with the existing security model.

Table 1: Proposed attack specification with 100Bytes attacker message size

Specification	SP/SA	SP/MA	MP/SA	MP/MA
Genuine client Message Rate*(Message/Second/per client)*	1000	1000	1000	1000
Genuine client Message size*(bytes)*	100	100	100	100
No of Genuine publisher clients	1-100	1-100	1-100	1-100
No of Genuine subscriber clients	1-100	1-100	1-100	1-100
Attacker Message Rate	100000	10000	10000	10000
No of attacker publisher clients	1	600	3	600
No of attacker subscriber clients	1	600	3	600
No of paths	1	1	3	3

Table 2: Proposed attack specification with 25KBytes attacker message size

Specification	SP/SA	SP/MA	MP/SA	MP/MA
Genuine client Message Rate*(Message/Second/per client)*	500	500	500	500
Genuine client Message size*(bytes)*	25000	25000	25000	25000
No of Genuine publisher clients	1-100	1-100	1-100	1-100
No of Genuine subscriber clients	1-100	1-100	1-100	1-100
Attacker Message Rate	100000	10000	10000	10000
No of attacker publisher clients	1	600	3	600
No of attacker subscriber clients	1	600	3	600
No of paths	1	1	3	3

The general process of the attack an attacker uses is described below.

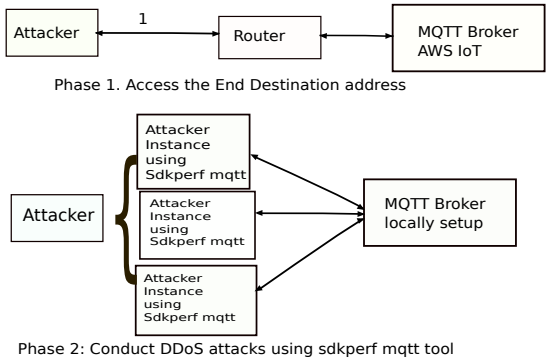
1) The attacker gets physical access to any of the IoT node in the network and acquires the X509 certificate, private key and device certificate. The attacker acquires these certificates by by accessing the root directory where the MQTT client is located in the operating system of the IoT node. we refer X.509 certificate as C1, private certificate as C2 and device certificate as C3 in this paper for simplicity purpose.

2) The attacker sniffs packet data using any of the network tools such as Wireshark. Later analyzing these packets the attacker can get the end destination address of the MQTT broker

3) Using the *sdkperf-mqtt* tool the attacker would create any of the four attack models(SP/SA, SP/MA, MP/MA and MP/MA) to create Denial of Service in the IoT network.

We show how to conduct all the four attacks in the following section.

Figure 1: Design Phase 1 Phase 2 overview



4.1 Design of Attack

The goal of our design is to automatically create the multiple publisher and subscriber clients originating from different systems and overwhelm the MQTT broker(see section 4.1) with the maximum number of publish and subscribe requests. As a result, to show the DoS and DDoS on MQTT broker affect the genuine client's service time. Our design is comprised of two main phases(see Figure 2).

The first phase design is used to show how the attacker can get the end destination address of AWS MQTT broker using networks tools. The second phase design includes a local IoT network which runs MQTT broker that acts as target to conduct the four DDoS attack models discussed in sections 3.1, 3.2, 3.3, 3.4. We use this design to demonstrate attack models.

The following are the assumptions for designing the Attack model.

1) The attacker has physical access to any one of the nodes in the IoT network and can read the stored certificates(C1, C2 and C3).

2) The attacker can connect to one of the router in the network and can sniff the packet using any of the network tools such as Wireshark.

Before we explain the design phases of the attack we walkthrough few AWS IoT modules.

Message Broker: - Message broker or MQTT Broker [1] is the server with which the things(IoT devices) in the internet can publish and subscribe to the topics. The MQTT broker in AWS is responsible for receiving the published messages, filtering them and identify who is interested and sending the messages to the interested subscribers. The MQTT broker is also responsible for holding the sessions of all the connected clients. One more responsibility of the broker is authentication and authorization of the clients.

Thing Registry:- Thing registry or Device registry [1] allows real devices or virtual applications to register with AWS IoT. Each device registered with the Thing Registry will have X.509 certificate along with device certificate and private key.

4.1.1 Phase 1

To setup AWS IoT test bed we create two IoT things in AWS IoT we name the things as DDoSthing1, DDoSthing2. we generate X.509 certificate, device certificate and device private key certificate in .pem format for one of the IoT devices DDoSthing1. These certificates are generated using the AWS command line interface. Using these certificates, in phase1 of the attack we develop a Mosquitto MQTT client in Ubuntu operating system using MQTT protocol version 3.1 and connect it with the MQTT broker running in AWS IoT. We show that different clients using the same set of certificates can connect to the AWS IoT concurrently by claiming different client Ids.

Using Wireshark network tool we sniff packets on the network while the client is connected to the AWS IoT. During our initial setup we found that DDoSthing2 can connect to MQTT broker with the stolen certificates of DDoSthing1 which means a single certificate can be used to connect any number of clients to a thing. This shows that MQTT broker can not identify the individual clients. Using this loop hole we conduct DDoS attacks on MQTT broker.

Table 3: MQTT server(AWS IoT) and client specifications

Name	Technical,specs
Server	AWS IoT
Mosquitto MQTT Broker version	1.4.4
MQTT version	3.1
Client	mosquitto_pub/mosquitto_sub Version 1.4.4 or MQTT.fx version 0.0.18
Client Hardware	Intel corei7 CPU @2.5GH 8 core
Client Operating system	Linux Ubuntu

The Mosquitto MQTT client we use support the following TLS implementations to successfully communicate with the AWS IoT client.

1. TLSv1.2
2. SHA-256 RSA certificate signature validation
3. All TLS cipher suite

Figure 2 describes the setup used in our experiment. We setup first test bed to demonstrate that using X.509 certificate, device certificate and private key of one client, multiple clients can connect to the IoT MQTT server by tapping the end destination address of the MQTT server using Wireshark tool. We use AWS IoT MQTT broker with multiple clients. The test bed built in phase1 is used to collect IP address and end destination address of the MQTT broker which is hosted in AWS IoT suite.

The MQTT broker hosted in the AWS IoT server has secure mechanism of authentication using 3 certificates X.509 certificate(C1), device certificate(C2) and private key of device(C3). We create a thing in AWS IoT which represents IoT device. Mosquitto MQTT client is used to publish/subscribe to a topic to MQTT broker in AWS IoT cloud.

Now we set up Wireshark instance in another system to sniff the packets of the router while the client is connected to the AWS IoT. By analyzing the sniffed packets the end destination address can be easily read. Using this end destination and three certificates (C1, C2 and C3), a trusted communication can be established and publish/subscribe information with AWS IoT broker with any number of clients using different client ids. This will be part of our DDoS attacks in the next section 4.2. This is explained in the appendix of this paper in much detail.

4.1.2 Phase 2

In the second experimental setup Figure. 3 Table. 4, we configure Mosquitto MQTT broker [5] and Paho MQTT client [3] in local system to evaluate the performance of MQTT broker and MQTT protocol with different types of message flood attacks. We use this set up to conduct stress testing with which we show that flooding messages from multiple clients can break the network and MQTT broker as well. To successfully setup the experiment we describe the following settings. Setting up Mosquitto MQTT broker as server Setting up Client system Generating multiple clients using '*sdkperf-mqtt*' Message structure Performance evaluation method

Table 4: MQTT server(attacker target) and client specifications

Name	Technical specs
Server hardware	Intel corei7 CPU @2.5GH 8 core
Primary memory capacity	8GB
Operating system	Ubuntu,14.04 64 bit
Mosquitto MQTT Broker version	1.4.4
MQTT protocol version	3.1
client(publisher and subscriber)	Paho Eclipse 1.0.3
Java version	1.8
Sdkperf-mqtt	Beta version

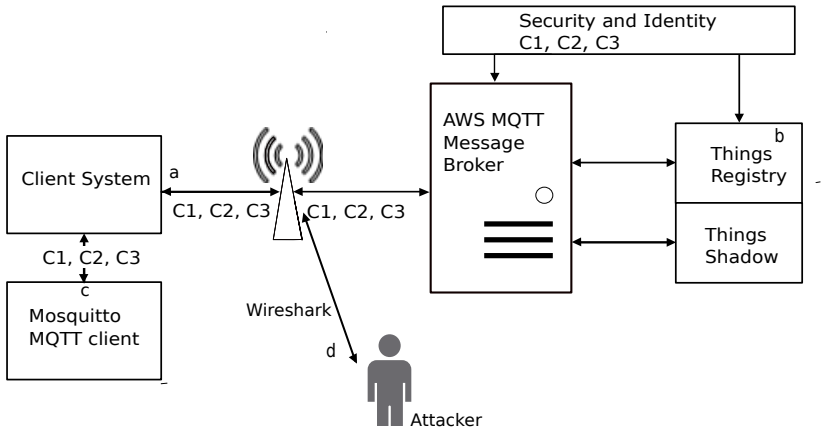
Setting up Mosquitto MQTT broker as server.

We set up Mosquitto MQTT broker 1.4.4 version running in Ubuntu operating system as our server. MQTT broker is configured to hold default number of queued messages as 100 which means the MQTT broker will not be able to hold the messages in buffer Queue more than 100 numbers. If the number of flooding messages make the queue full then the MQTT broker drops the messages. We also configure the maximum size limit for broker as default which means the MQTT broker will accept all the valid messages up to maximum payload size of 268435455 bytes. The security is disabled as we assume that the certificates (C1, C2, C3 or user-name and password) are already available for the attacker. The listener port is set to default which is 1883 as prescribed by the Mosquitto.org. Apart from these configuration settings all other options are set to default.

Setting up Attacker Client system .

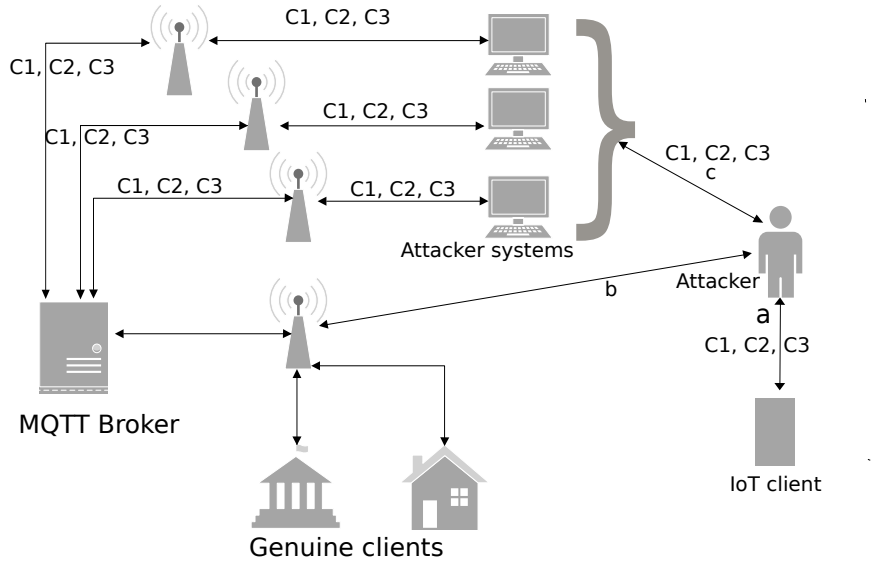
We set up Paho-mqtt[3] client running in Ubuntu as our

Figure 2: AWS *(Amazon Web Services)* IoT setup



a. C1, C2, C3 are the X.509 certificate, device certificate and device private key in .pem format b. DDoSthing1 is created things registry c. Mosquitto MQTT client passes the certificates to AWS MQTT broker to successfully get authenticated and publish/subscribe to topics. d. Using wireshark the attacker extracts the end destination address

Figure 3: Attacker target and client setup



a. Attacker gets physical access to any of the IoT node in the network to get access to the security credentials stored on the device b. Attacker sniffs the packet data to get the AWS IoT end destination address c. Attacker uses the end destination address and the stolen certificates to conduct DDoS attacks thus denying service to the genuine clients

client using MQTT protocol version 3.1 in our client system. This is configured to work with *sdkperf* tool [11] which is explained in the later section.

Generating multiple Attacker clients.

To generate multiple clients we have *malaria* tool [10] and *sdkperf-mqtt*. we use *sdkperf-mqtt* tool as it provides latency statistics. *sdkperf-mqtt* tool uses the Java SDKperf and implements every client using the Paho Java API which is set up in the previous section. We use the Sdkperf command line options to create clients, publish and subscribe to the MQTT broker topics. The Sdkperf also provides command line options to specify the target MQTT broker, number of publish clients, number of subscribe clients, size of each message payload, number of messages per second from each path, total number of messages to send per session and an option to select latency statistics.

Message Payload.

We use the auto generated messages using the sdkperf-mqtt tool by specifying the size of the message payload while generating the clients. We use two payload sizes to create

denial of service attacks one is with 100bytes and another one is with 25KB payload message size. For each attack model described in section 3

Connecting the Genuine publisher and subscriber client.

For connecting genuine publisher and subscriber client to the MQTT broker we use the sdkper-mqtt tool as described in the previous section. we set up 1 to 100 number of genuine clients with varying message payload sizes of 100Bytes and 25KBytes

4.2 Performing DDoS Attacks

To demonstrate DDoS attack models proposed in section 3, we assume that attacker has access to the security certificates like X.509 certificate(C1), device certificate(C2) and private key(C3) of any one device in the network. The attacker uses the stolen certificates (C1, C2, C3) to connect multiple publish/subscribe clients with different client ids to the MQTT broker. Here the attacker uses the MQTT broker inability to differentiate devices based on the certificates. As proposed in section 3 we conduct the following attacks on the target MQTT broker setup in phase2.

1. Multiple Path Multiple Attacker (MP/MA)
2. Single Path Single Attacker (SP/SA)
3. Single Path Multiple Attacker (SP/MA)
4. Multiple Path Single Attacker (MP/SA)

In order to accomplish the specifications mentioned in the tables 1, 2 , for each of the above attacks we use the test bed in section 4.1.2(phase 2) with two message payload sizes one with 25KB and another with 100Bytes.To instantiate multiple publisher/subscriber clients we use the sdkperf tool. Before we go into the details of the experiment we describe the below common setups and performance definitions for all the attack models mentioned above.

Message structure to publish to the server.

We choose the 100 Bytes and 25 KB message structure to publish to the MQTT broker by the client. The messages are randomly auto generated by sdkperf-mqtt tool. We select the message as 100 Bytes and 25KB as this is the optimum message payload size that we can transmit and create DoS. However increasing the message size of the client greater than 25KB we observe the latency of the messages drastically increased and the MQTT broker memory and CPU utilization increased to 100 percent hence making system freeze.

Avg. Message Rate.

Avg. Message rate of publisher is the number of messages that are published by the MQTT client and Avg. Message rate for subscriber is the number of messages received at the MQTT subscriber client.

Avg. Message Latency Rate.

While generating the message by sdkperf tool we embed the time stamp say TStampA when the message is generated in publishing client. When the message arrives at the subscriber client which is again instantiated by the sdkperf tool, the subscriber takes the time stamp say TStampB then the sdkperf tool API reads the values of TStampA and TStampB using the below formula it calculates the latency by using the the following formula.

$$\frac{TStampA - TStampB}{clockfrequencyofCPU}$$

..

The latency is measured from the time message generated by publisher API to the time the message pay load received at the subscriber API.

Number of Genuine publishers.

These are the publisher clients that are already connected to the MQTT broker before the attacker starts the attack.

Number of Genuine subscribers.

These are the subscriber clients that are already connected to the MQTT broker before the attacker starts the attack.

Errors.

There are three errors that we observe during our attack experiments. Time out error is reported by the client when the MQTT broker does not send acknowledgement when publisher publishes the data. Outgoing message error is reported by the MQTT broker if the broker is not able to connect and deliver the message to the subscriber due to overload. Socket error is reported by clients when the connection is closed by the broker when the client is still connected.

Before we start the experiment we setup genuine publisher and subscribers where we calculate the performance of the clients without any attacks. We choose to have different tests where the number of clients vary from 1 to 200 and the message payload of 100 Bytes, 25KBytes. It is observed during our testing, that when there is no attack happening in ideal test conditions there is no latency when the message payload is 100Bytes. However there is a message latency when the message payload is 25KB as shown in the table 5.

It can also be inferred from the table that when the number of clients increased more than 50 with 25KB of payload the latency is increased. Note that in the below attacks we use the same message payload size of the genuine clients. for ex: If the genuine client is using 100Bytes of message payload, the attacker client also uses the payload size of 100Bytes and the same applies for 25KB message payload also.

4.2.1 Multiple Path Multiple Attacker (MP/MA)

First a genuine client system is setup with publisher and subscriber of 100Bytes payload size. Now to conduct multiple path multiple attack, we use three systems with each system generating 100 paho publishers and 100 paho subscribers in total 200 clients per system. This makes total 600 clients attacking the target MQTT broker. We observed that the genuine client publish rate is decreased from 49 messages per second to 0-1 messages per second and the messages were indefinitely delayed until the attacker disconnects.

We also conducted the MP/MA attack on 10, 40, 60, 120 and 200 genuine clients simulated network and the results are as mentioned in the table 6 and plotted in graph . We could observe the clear denial of service for these genuine clients when we started the attacker instance from different paths. The above steps are repeated for message payload size of 25KB. As shown in the graph the attacks are conducted on MQTT target broker with clients of payload size 100Bytes and 25KBytes. In the case of MP/MA the denial

Table 5: Genuine Publisher, Subscriber readings without Attack

No. of Genuine publishers	No. of genuine subscribers	Avg Message Rate for publisher and subscriber	Avg Message Latency	Message Payload	Errors
1	1	49	0	100 Bytes	No Errors
10	10	150	0	100 Bytes	No Errors
20	20	190	0	100 Bytes	No Errors
40	40	210	0	100 Bytes	No Errors
60	60	253	0	100 Bytes	No Errors
100	100	300	0	100 Bytes	No Errors
1	1	4	0	25KBytes	No Errors
10	10	18	0	25KBytes	No Errors
20	20	60	0	25KBytes	No Errors
40	40	90	0	25KBytes	No Errors
60	60	110	0	25KBytes	No Errors
100	100	120	0	25KBytes	No Errors

Table 6: Multiple Path Multiple Attacker (MP/MA) statistics

No. of Genuine publishers	No. of genuine subscribers	Avg Message Rate for publisher per client	Avg Message Latency	Message Payload	Errors
1	1	7	>80%	100 Bytes	Data Drop
10	10	3	>90%	100 Bytes	Data drop
20	20	1	100%	100 Bytes	Data drop
40	40	1	100%	100 Bytes	Data drop
60	60	0	100%	100 Bytes	Socket Error
100	100	0	100%	100 Bytes	socket Error
1	1	5	100%	25K Bytes	No Errors
10	10	1	100%	25KBytes	Data drop
20	20	1	100%	25KBytes	Data drop
40	40	0	100%	25KBytes	socket Error
60	60	0	100%	25KBytes	socket Error
100	100	0	100%	25KBytes	socket Error

of service happened the moment we start the 200

4.2.2 *Single Path Single Attacker (SP/SA)*

In single path single attacker (SP/SA) we use single paho client system using sdkperf-mqtt to publish and subscribe requests indefinitely. We schedule publish and subscribe messages at the rate of 1000,000 messages per second, however the MQTT broker is able to support only 50 messages per second. We perform two attacks as mentioned in specification Table 1 and 2 one with message payload of 100 Bytes and another with message payload of 25KB.

First we establish a stable connection between the MQTT broker and the MQTT client, In the second step we start attacker instance from one system with the specifications mentioned in the table 1 and 2. The test is run for two minutes to check the stability and latency of the MQTT broker. The results are recorded in the table 7 for both 100Bytes of message size and 25KB of message size. We could observe that message rate of genuine publishers have not changed much as we use only single attacker client in this model.

We perform the above steps for 10, 20, 40, 60, 80 and 100 genuine clients as in table 7.

4.2.3 *Single Path Multiple Attacker (SP/MA)*

We setup one genuine publisher client and one subscriber client exchanging messages at maximum possible rate. The publish/subscribe client are set using the sdkperf-mqtt tool explained in the earlier sections. We use the 100Bytes of message payload size for these genuine clients. We observe that the messages are having no latency initially. Now, we start 300 publish clients and 300 subscriber clients from a single system with message size of 100Bytes and message rate as 100,000 thereby publishing more than the maximum number of messages that a MQTT broker could handle. As a result we see that many clients are dropped, CPU resources utilized to 100 percent and at times the MQTT broker dropped most of the messages. The latency rate has increased indefinitely resulting in messages not at all transmitting. The statistics of this attack can be referred at table 8. Now we repeat the above mentioned steps for 25KB message payload for both the genuine clients and attacker clients. We observe that the SP/MA and MP/MA show similar effect on the MQTT broker.

4.2.4 *Multiple Path Single Attacker (MP/SA)*

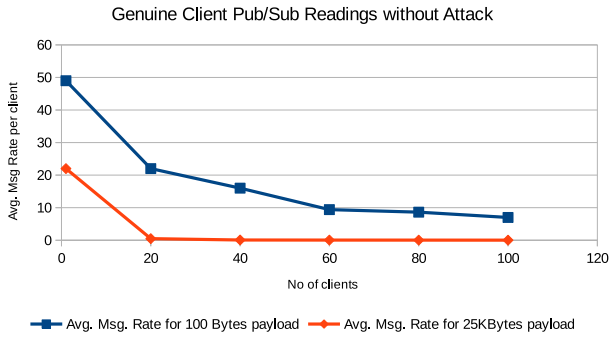
In this attack we setup single genuine publisher and subscriber client which publish and subscribe messages at a fixed rate. For first test specification with 100Bytes of data we publish at maximum rate using the genuine client. Now using sdkperf we start the attacker instances from multiple systems. The attacker clients are set to publish rate of 10,000 messages per second. Now that we observe the publish rate of the genuine client is reduced from 50 per second to less than 10 per second.

We repeat it by increasing the number of genuine clients as 10, 20, 40, 60, 80 and 100. We observer that Multiple path single Attacker is less intense when compared to Multiple path Multiple attacker and Single path multiple attacker.

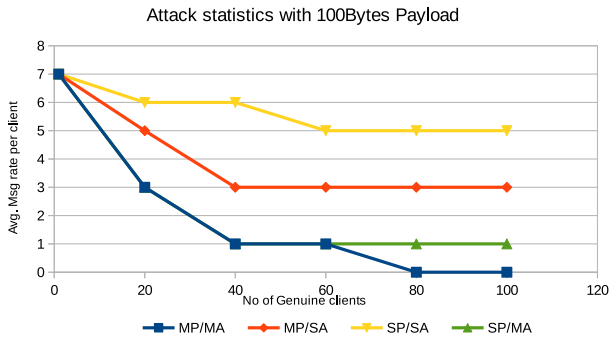
Now we change the message size and repeat the above process with the same number of clients and multiple paths but with message size of 25KB. the results of this attack can be seen in table 9.

Figure 4: Results plot

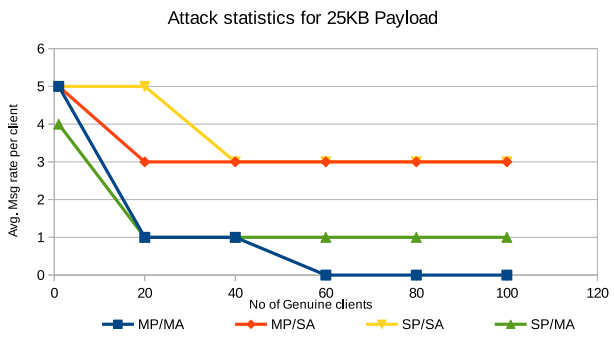
(a) Genuine client Pub/Sub without Attack



(b) After Attack statistics for 100Byte Message payload



(c) After Attack statistics for 25KByte Message payload



4.3 Results

From above graphs 4, we see that all four attack models SP/SA, SP/MA, MP/SA and MP/MA have impacted the MQTT broker service time. However we see from graphs 4.b that SP/MA and MP/MA impacted MQTT broker service time and resources greatly with respect to the number of attacker clients rather than number of paths. We could observe from graphs 4.b and 4.c that message payload size also plays considerable role in limiting the MQTT broker performance. We also can infer that as the number of gen-

Table 7: Single Path Single Attacker (SP/SA) statistics

No. of Genuine publishers	No. of genuine subscribers	Avg Message Rate for publisher and subscriber	Avg Message Latency	Message Payload	Errors
1	1	50	0%	100 Bytes	No Errors
10	10	20	0%	100 Bytes	No Errors
20	20	16	1%	100 Bytes	No Errors
40	40	16	2%	100 Bytes	No Errors
60	60	16	10%	100 Bytes	No Errors
100	100	13	10%	100 Bytes	No Errors
1	1	6	0%	25K Bytes	No Errors
10	10	5	5%	25KBytes	No Errors
20	20	5	30%	25KBytes	Data drop
40	40	3	99%	25KBytes	Socket Error
60	60	3	99%	25KBytes	Socket Error
100	100	3	99%	25KBytes	Socket Error

Table 8: Single Path Multiple Attacker (SP/MA) statistics

No. of Genuine publishers	No. of genuine subscribers	Avg Message Rate for publisher per client	Avg Message Latency	Message Payload	Errors
1	1	7	>80%	100 Bytes	Data drop
10	10	3	>90%	100 Bytes	Data drop
20	20	2	100%	100 Bytes	Socket Error
40	40	2	100%	100 Bytes	Socket Error
60	60	2	100%	100 Bytes	Socket Error
100	100	2	100%	100 Bytes	Socket Error
1	1	4	>99%	25K Bytes	Socket Error
10	10	1	100%	25KBytes	Socket Error
20	20	1	100%	25KBytes	Socket Error
40	40	1	100%	25KBytes	Socket Error
60	60	1	100%	25KBytes	Socket Error
100	100	1	100%	25KBytes	Socket Error

Table 9: Multiple Path Single Attacker (MP/SA) statistics

No. of Genuine publishers	No. of genuine subscribers	Avg Message Rate for publisher per client	Avg Message Latency	Message Payload	Errors
1	1	45	10%	100 Bytes	No Errors
10	10	18	10%	100 Bytes	No Errors
20	20	15	15%	100 Bytes	No Errors
40	40	12	15%	100 Bytes	Data drop
60	60	9	20%	100 Bytes	Data drop
100	100	8	100%	100 Bytes	Data drop
1	1	5	0%	25K Bytes	No Errors
10	10	3	5%	25KBytes	No Errors
20	20	3	30%	25KBytes	Data drop
40	40	3	99%	25KBytes	Socket Error
60	60	3	99%	25KBytes	Socket Error
100	100	3	99%	25KBytes	Socket Error

uine clients are increased the attacker system cripples the MQTT broker which means MQTT broker performance is limited to number of clients.

From table 6 we see when multiple path multiple attacker model is used the genuine client publish rate is reduced from 49 per second to 7 per second, when there are single publisher and subscribe clients. When the number of clients increased to 10 later 20 until 100 the data drop and socket errors were reported at broker end which means the broker is loaded and hence dropping the messages from clients which forces the clients to resend the same message which consumes considerable amount of resources, thereby creating DDoS. In table 8 the results of SP/MA are similar to the behavior of MQTT client which is seen with MP/MA, this shows that the MQTT broker is limited to number of clients rather than the number of paths the connections are coming from.

In tables 7 and 9 even though we are using multiple attacker systems in MP/SA the effect on the MQTT broker is not much different from the SP/SA attack model. During our experiment we observe that at times the new clients are not given connection during the attack hence forcing the genuine client to request continuously thereby exhausting all the resources. During intense attack such as MP/MA the MQTT broker was not able to reconnect with the genuine clients even after the attack was over. This shows that MQTT broker is not reliable and prone to attacks.

5. CONCLUSIONS

We conclude in this paper that the DDoS attacks are a viable threat for IoT devices and these DDoS attacks can consume CPU resources of IoT nodes and MQTT broker.The inability of MQTT broker to identify the genuine device can be used as loop hole to conduct DDoS attacks. We also observe that by generating the flood of messages the attacker can achieve Denial of Service in the IoT network. From this we infer that MQTT broker is unable to restrict one certificate for each client. In case, if the attacker gets access to these certificates stored on the device, there is always possibility that the genuine client will lose the connectivity.

We also see that the MQTT broker could crash with flood message requests from multiple malformed clients by attack. If the network has IoT devices which can act as both MQTT broker and MQTT client then all the IoT nodes(actuators, sensors and other smart devices) can be programmed and used for stealing private information and conduct unethical activities. Especially where actuators are part of the network, it can be lethal.

We also infer that a network of bots can be programmed to attack the MQTT broker which leads to increased latency in subscribed messages and published messages and can lead to complete network failure.

We show in this paper the security and privacy flaws existing with the present Mosquitto MQTT Broker. And there is a possibility of DDoS attacks. If the attacker gets access to any device physically and able to read the certificate information.

6. REFERENCES

[1] Amazon. https://aws.amazon.com/documentation/iot/, 2015.

[2] A. Aris, S. F. Oktug, and S. B. O. Yalcin. Internet-of-things security: Denial of service attacks. In *Signal Processing and Communications Applications Conference (SIU), 2015 23th*, pages 903–906. IEEE, 2015.

[3] eclipse. http://www.eclipse.org/paho/, 2015.

[4] S. Misra, P. V. Krishna, H. Agarwal, A. Saxena, and M. S. Obaidat. A learning automata based solution for preventing distributed denial of service in internet of things. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 114–122. IEEE, 2011.

[5] Mosquitto. http://mosquitto.org/, 2015.

[6] D. Moustis and P. Kotzanikolaou. Evaluating security controls against http-based ddos attacks. In *Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on*, pages 1–6. IEEE, 2013.

[7] MQTT. http://mqtt.org/documentation, 2015.

[8] P. Pongle and G. Chavan. A survey: Attacks on rpl and 6lowpan in iot. In *Pervasive Computing (ICPC), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[9] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly. Ddos-shield: Ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on Networking (TON)*, 17(1):26–39, 2009.

[10] remakeelectric. https://github.com/remakeelectric/mqtt-malaria, 2015.

[11] solacesystems. http://dev.solacesystems.com/downloads, 2015.

[12] X. Yang, T. Ma, and Y. Shi. Typical dos/ddos threats under ipv6. In *Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on*, pages 55–55. IEEE, 2007.

APPENDIX

A. AWS IOT

A.1 End destination address packet sniffing

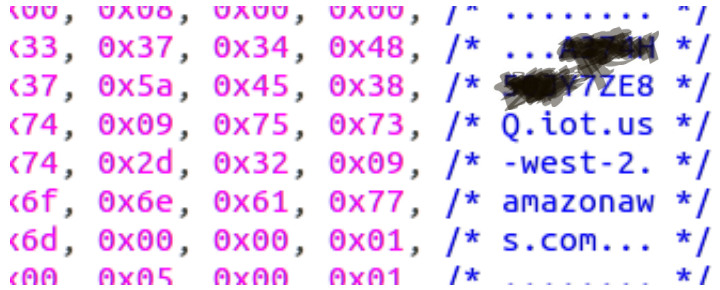
We use wireshark tool to get the end destination address of the AWS IoT MQTT broker. As we said in the phase 1 of the experiment, when we sniff the packet data using the server the following shown in the Figure 5 is the output.

A.2 Connecting to the AWS IoT end destination address

Using the end destination address collected in the above section A.1 we use the command to publish messages to the AWS MQTT Broker.

```
mosquitto_pub --cafile "/home/malz/AWS_Iot
/certs/rootCA.pem" --cert "/home/malz/AWS_Iot/
certs/11_04/cert.pem" --key "/home/malz/AWS_Iot
/certs/11_04/privateKey.pem" -h
 "A3HIxxxxxO2FF04X9.iot.us-west-2.amazonaws.com"
  -p 8883 -q 0 -d -t "topic/lightravi" -i
  "cleinttest" -m "{\"key1\":
 \"Hello, World1\"}"
```

Figure 5: A sample packet sniffed from the wireshark too which displays the end destination address.



The end destination address sniffed here is Axxxxxx7ZE8Q.iot.us-west-2.amazonaws.com

From above command we can see how we can make use of AWS end destination address "A3HIxxxxxx02FF04X9.iot.us-west-2.amazoonaws.com" to connect to the MQTT broker with multiple client Ids.

B. SDKPERF-MQTT TOOL

B.1 Generating clients using sdkperf-mqtt tool

We use the sdk perf tool to generate multiple clients concurrently using single system. the following is the command used to generate the clients.

```
./sdkperf_mqtt.sh  -cip localhost -ptl test
-stl test -mpq 1 -msq 1 -msa 250000 -mn
100000000 -mr 100000 -cc 200 -l
```

In this command 'cip' is the ip address of the MQTT broker or end destination address if the target is AWS MQTT broker. 'ptl/stl' is the publisher or subscriber topic list. Topic is the service or a destination inside the MQTT broker. A functionality which gets data from IoT devices is called topic. Every IoT node servers a topic to which it publishes or subscribes.

'msq/mpq' is the quality of service(QoS) of the publisher or subscriber.

'msa' is the payload size of the message that a publisher want to publish.

'mn' is the total number of message to sent and mr is the number of messages to send per second.