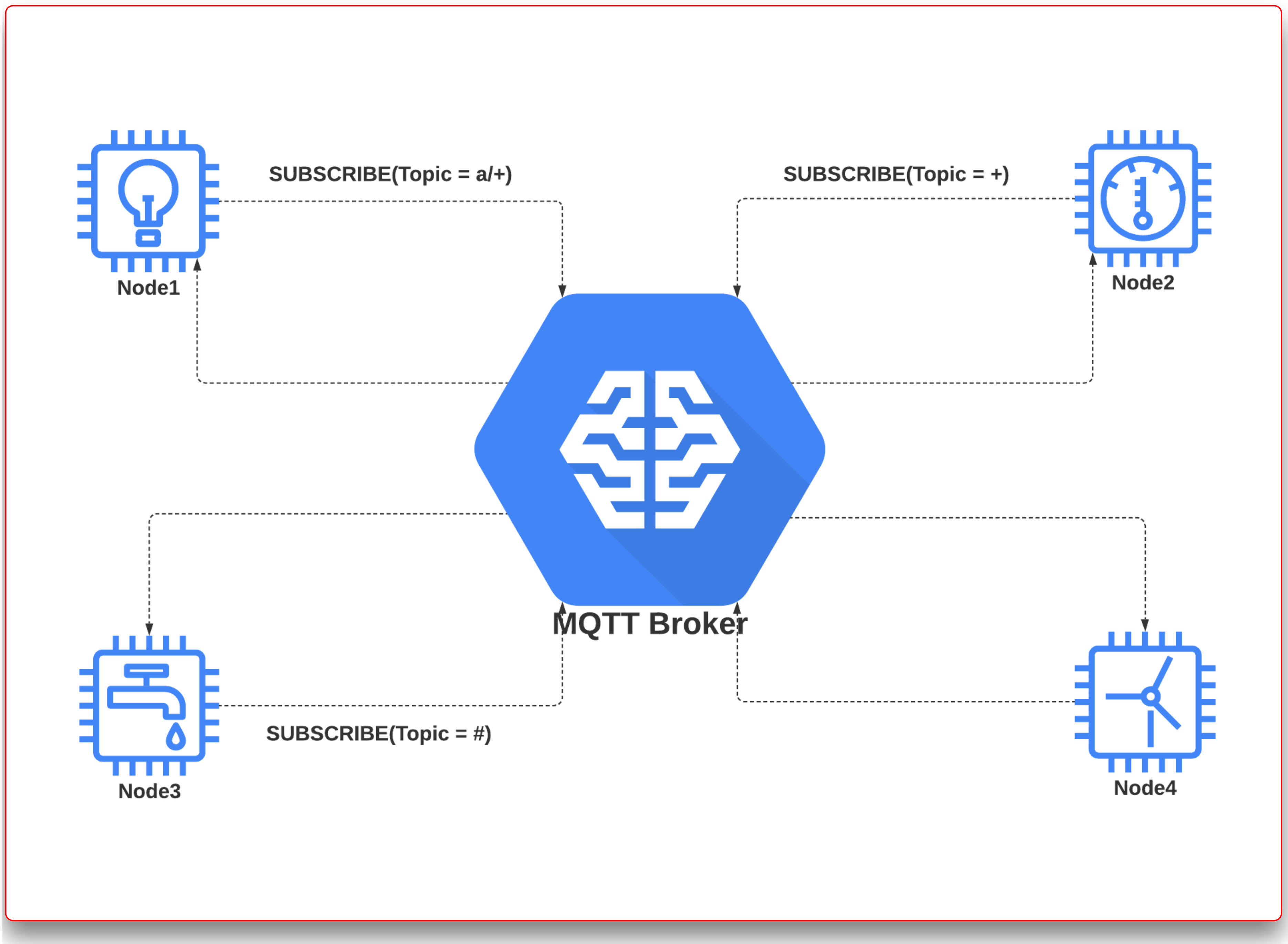
Payatu

IoT Security - Part 12 (MQTT Broker Security - 101)

Dattatray
30-August-2020



MQTT Broker Security - 101

This blog is part of IoT Security series where we discuss the basic concepts pertaining to the IoT/IIoT eco-system and its security. If you have not gone through the previous blogs in the series, I would urge you to go through those first. In case you are only interested in MQTT broker security, feel free to continue.

IoT Security - Part 1 (101 - IoT Introduction And Architecture)

IoT Security - Part 11 (Introduction To CoAP Protocol And Security)

In this blog, we are going to look at Client Authentication and other security methods used by MQTT broker for secure connections with MQTT clients

1 Introduction

MQTT protocol is becoming popular in many industries, it also fascinates IoT beginners, enthusiast IoT developers, and security researchers who want to explore this technology for new opportunities.

If you are new to MQTT and want to understand more about the protocol, please go through our blog on MQTT Protocol and Security.

IoT Security - Part 10 (Introduction To MQTT Protocol And Security)

There are many open-source and commercial implementation for MQTT Broker and client available to choose from. But before using any MQTT implementation of MQTT broker, one must understand the security capabilities of MQTT broker and clients.

This blog articulates the information required for beginners to start working on security of MQTT protocol and security capabilities required for MQTT Broker and client.

We are using Eclipse Mosquitto Broker to understand broker configuration and how to configure the Mosquitto broker for desired security mode. We will also look into AWS IoT Core for similar feature and configuration.

2 Data and Connection Security

2.1 Secure Connection

As per MQTT specs TCP ports 8883 and 1883 are reserved for MQTT TLS and non TLS communication respectively.

In non TLS communication i.e. on port 1883, MQTT messages are transferred in plain-text format between MQTT broker and client and should be considered as an unsecure connection. Whereas in TLS communication or secure connection i.e on TCP port 8883, MQTT messages are encrypted by TLS security layer before transmission.

It is always a good idea to use secure communication between MQTT broker and client.

2.1.1 Mosquitto Broker

Mosquitto Broker provides an option in mosquitto.conf file to select listener port for MQTT connection, default value is 1883. It should be changed to 8883 for secure (TLS) communication.

```
# Port to use for the default listener.
port 8883
```

This is just the first step and you still need to configure broker for using certificates to make it work.

2.1.2 AWS IoT Core

AWS IoT core by default uses TLS connection for MQTT on port 8883

2.2 Application data or MQTT Payload security

MQTT protocol is messaging protocol and does not provide any encryption for the application payload it is carrying. It is responsibility of the application to encrypt/decrypt application data (payload) before sending to or after receiving from the MQTT layer to achieve end to end data encryption. Please note that application level encryption is independent of TLS packet encryption.

3 Client Authentication

For any MQTT broker, there are three types of client authentication methods available to verify the identity of MQTT client 1. Client Identifier (Client Id) 2. Username and Password 3. Client Certificates

3.1 Client Identifier (Client Id)

All MQTT clients must have a unique Client Id as per the MQTT Spec, and the client must send this Client Id with the CONNECT packet. The broker uses this Client Id to create and maintain the session state.

The Client Id is not part of MQTT authentication and is only used to uniquely identify the MQTT connection. However, some broker implementations provide some way to establish client identity using Client Id or something derived from Client Id. However, using client ID alone for authentication or verification is a bad idea because it can be manipulated/spoofed at the client side.

We will see how Mosquitto Broker and AWS IoT Core use client id for basic security in following examples

3.1.1 Mosquitto Broker

Mosquitto Broker provides an option called clientid_prefixes in mosquitto.conf file to configure Client ID prefixes, which allow clients with specified prefixes in their Client ID to connect to the mosquitto broker.

```
# =================================================================
# Security
# =================================================================

# If set, only clients that have a matching prefix on their
# clientid will be allowed to connect to the broker. By default,
# all clients may connect.
# For example, setting "secure-" here would mean a client "secure-
# client" could connect but another with clientid "mqtt" couldn't.
clientid_prefixes C_ID-
```

Here in above example, clientid_prefixes is set to **C_ID-** and a client with client id let's say **C_ID_Client01** will be allowed to connect, but a client with the Client ID **Client01** will not be allowed to connect.

3.1.2 AWS IoT Core

AWS IoT provides the option to set thing id while creating thing in IoT core, here thing id is not the same as Client ID. However, AWS IoT allows connection for registered thing when connection policy is configured for the corresponding thing id.

> Note: If the client with client id as client_1 is connected and the second client with the same client id i.e. client_1 is trying to connect the broker, then the first client is disconnected by the broker. This is the bare minimum security provided for any client session by MQTT protocol and keeps a client with intermittent connectivity from spawning multiple MQTT sessions. However, as discussed in our previous blog post on MQTT, it also opens the door for Denial of Service attacks on the MQTT network.

3.2 Username and Password

MQTT spec defines the requirement of a username and password for MQTT client authentication. A client sends the username and password with the CONNECT packet to the MQTT broker and the broker validates the username and password before accepting the MQTT session.

The username and password is sent in the CONNECT packet to the broker in cleat text format unless encrypted at the transport layer i.e. using port 8883 for connection.

> Note: According to the MQTT spec it is not mandatory to use authentication.

3.2.1 Mosquitto Broker

Mosquitto Broker provides two parameters in mosquitto.conf file to enable client authentication by client - username and password.

3.2.1.1 Allow Anonymous

(Please provide a sentence explaining what is allow anonymous) To enable client authentication using credentials, you also need to set allow_anonymous parameter to false in mosquitto.conf file.

```
# Defaults to true if no other security options are set. If `password_file` or
# `psk_file` is set, or if an authentication plugin is loaded which implements
# username/password or TLS-PSK checks, then `allow_anonymous` defaults to
# false.
#
allow_anonymous false
```

3.2.1.2 Password file Path

To create a password file, the mosquitto broker comes with mosquitto_passwd utility which creates a password file.

> Example:
>
> a. Create/append password file and add user john with hashed password
>
> ```
> $ sudo mosquitto_passwd -c /etc/mosquitto/pwfile john <somepassword>
> ```
>
> b. Delete a user from a password file
>
> ```
> $ sudo mosquitto_passwd -d /etc/mosquitto/pwfile john <somepassword>
> ```

Once password file has been created, set the current location of password file using password_file parameter in mosquitto.conf file.

```
# See the TLS client require_certificate and use_identity_as_username options
# for alternative authentication options. If an auth_plugin is used as well as
# password_file, the auth_plugin check will be made first.
password_file /etc/mosquitto/pwfile
```

Note: The default location of the mosquitto password file for Linux is /etc/mosquitto/pwfile and for windows is the mosquitto installation folder ~\mosquitto\passwords.txt

3.2.2 AWS IoT Core

AWS IoT core does not support username and password authentication but it provides option for custom authentication.

Note: Another use of username is to allow or deny topic access to clients based on their username, we will see it in following section 4.1.3.2

3.3. Client SSL Certificate

Considered as the most secure method of client authentication, in certificate-based authentication client sends an SSL certificate which is signed by a trusted root CA to the server to authenticate the client.

3.3.1 Mosquitto Broker

Mosquitto broker provides below options in mosquitto.conf file to enable certificate-based client authentication.

3.3.1.1 Client Certificate require

When require_certificate is set to true, MQTT clients that provide valid certificate during connection handshake are allow to connect.

```
# By default an TLS enabled listener will operate in a similar fashion to a
# https enabled web server, in that the server has a certificate signed by a CA
# and the client will verify that it is a trusted certificate. The overall aim
# is encryption of the network traffic. By setting require_certificate to true,
# the client must provide a valid certificate in order for the network
# connection to proceed. This allows access to the broker to be controlled
# outside of the mechanisms provided by MQTT.
require_certificate true
```

3.3.1.2 Root CA file Path

The cafile or capath must be set to a trusted CA certificate that has signed your server certificate, to enable certificate-based client authentication.

```
# At least one of cafile or capath must be defined to enable certificate based
# TLS encryption. They both define methods of accessing the PEM encoded
# Certificate Authority certificates that have signed your server certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificates.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must run
# "openssl rehash <path to capath>" each time you add/remove a certificate.
cafile /etc/mosquitto/certs/ca.crt
#capath
```

3.3.1.3 Server Certificate and keyfile Path

The certfile is requested by MQTT client for server authentication during TLS handshake.

```
# Path to the PEM encoded server certificate.
certfile /etc/mosquitto/certs/server.crt
# Path to the PEM encoded keyfile.
keyfile /etc/mosquitto/certs/server.key
```

3.3.2 AWS IoT Core

AWS IoT core uses x.509 certificate-based authentication as default client authentication method.

Read more about AWS IoT client authentication here.

4 Restrict access to server resources

Every MQTT broker must have some access control mechanism to restrict MQTT clients from accessing server resources based on information provided by the client such as User Name, Client Identifier, the hostname/IP address of the client, or the outcome of authentication mechanisms.

4.1 Mosquitto Broker

Mosquitto broker has a mechanism called an access control list or ACL for MQTT topics, ACL is used to control the client access to subscribe and publish to topics on the broker. User name or Client ID is used to restrict access to broker topics.

The acl_file parameter in mosquitto.conf is a path to a file that contains an access control list. Valid file path enables ACL for topic restriction.

```
# If an auth_plugin is used as well as acl_file, the auth_plugin check will be
# made first.
acl_file /etc/mosquitto/aclfile
```

> Note: Location of default ACL file i.e aclfile.example for linux is /usr/share/doc/mosquitto/examples/aclfile.example and for windows it is the mosquito installation folder ~\mosquitto\aclfile.example

Default content of aclfile.example file

```
 # This affects access control for clients with no username.
topic read $SYS/#

# This only affects clients with username "roger".
user roger
topic foo/bar

# This affects all clients.
pattern write $SYS/broker/connection/%c/state
```

There are three types of rules by which ACL is established and saved in acl_file

4.1.1 topic name

This rule uses the topic keyword and is applicable to all anonymous clients. If a client uses credentials for the connection then this rule is ignored for that client.

Below is the format used to define this rule -

```
topic [read|write|readwrite] <topic>
```

> Example:
>
> Below line gives read (subscribe) access to any anonymous client and access to any other topic would be denied.
>
> ```
> topic read $SYS/#
> ```

4.1.2 user name

This rule uses the user keyword and is for the clients that use user credentials for the connection. Below is the format used to define this rule -

```
user <username>
topic [read|write|readwrite] <topic>
```

Here the username is the same as in the password file

Example:

Below lines gives read (subscribe) and write (publish) access to topic foo/bar to a client with user name roger and ignored for other clients.

```
user roger
topic foo/bar
```

4.1.3 pattern substitution with the topic

This rule uses the pattern keyword. Using pattern substitution with the topic, a broker allows access to a client using its client id or username.

Below is the format used to define this rule -

```
pattern [read|write|readwrite] <topic>
```

Note: As per Eclipse Mosquitto documentation following patterns are available for substitution

1. %c to match the client id of the client
2. %u to match the username of the client

The following line is default entry in the ACL file and grants write (publish) access to all clients.

```
pattern write $SYS/broker/connection/%c/state
```

4.1.3.1 Client ID substitution

Following example will illustrate how to use client id substation for topic access

Example:

Below line allow read access to topic pattern write sensor/<clientid>/data to all clients

```
pattern write sensor/%c/data
```

Below line will allow write access to client with client id c1 to access this topic and denies access to all other clients

```
pattern write sensor/c1/data
```

4.1.3.2 User name substitution

Following example will illustrate how to use username substation for topic access

Example:

Below line allow read access to topic user/<username>/logintime to all client

```
pattern read user/%u/logintime
```

Below line will allow read and write access to the topic user/john/lastlogin to user john

```
pattern read | write user/john/lastlogin
```

4.2 AWS IoT Core

In AWS IoT, permissions to access the broker's resources are controlled by AWS IoT Core policies. Read more on AWS IoT Core policies here.

5 Certificate Revocation List

Though certificate based authentication is considered as most secured, certificate once issued, it cannot be modified and controlling one's access when it is not expired is a challenge. And then there are scenarios where certificates form cloned or decommissioned devices, are used to get system access.
To avoid such scenarios, Certificate Revocation List (CRL) are issued by certificate authority CA. It contains information of the revoked certificates from decommissioned or compromised devices that no longer be trusted and prevent from any future use.

5.1 Mosquitto Broker

The CRL file is must when certificate-based client authentication is used i.e. require_certificate set to true. It contains information of the revoked certificates from decommissioned or compromised client devices to prevent any future use.

You can use the crlfile parameter from mosquitto.conf to point to the PEM encoded revocation file.

```
# If you have require_certificate set to true, you can create a certificate
# revocation list file to revoke access to particular client certificates. If
# you have done this, use crlfile to point to the PEM encoded revocation file.
crlfile /etc/pki/mosquitto/content/crl/mosquitto_crl.pem
```

5.2 AWS IoT Core

Using AWS IoT console, the user can deactivate or revoke the client certificates. Read more on client certificate activation or deactivation here.

6 Conclusion

There are serval security mechanisms available out there to secure MQTT communication and it varies from implementation to implementation. These security mechanisms implemented at the MQTT broker side, correct configuration of MQTT broker and MQTT client reduce the attack surface in an IoT ecosystem and give a better edge over known attacks. We hope this blog post gave you a good insight about the security mechanisms used to secure MQTT connection between a broker and a client.

Continue to the next part - IoT Security-Part 13 (Introduction To Hardware Recon)

7 References

- **Mosquitto broker configuration file**
- **mosquitto_passwd utility**
- **Configure SSL/TLS support for Mosquitto**
- **Security in AWS IoT**