Seguridad EMQX

Autenticación

Introducción

Este proceso se emplea para verificar la identidad del cliente MQTT y prevenir que terceros puedan publicar o suscribirse sin autorización. Para almacenar los datos necesarios para autenticar clientes MQTT, se utiliza una base de datos MySQL a la cual el servidor EMQX consulta para verificar la coincidencia entre el usuario y contraseña que el cliente utiliza para publicar o suscribirse y las que están almacenadas.

MySQL

Es una base de datos, se utiliza para estructurar y organizar los datos. Estos datos se almacenan en una estructura de "matriz" donde las columnas, conocidas como atributos, representan el tipo de dato almacenado, y las filas, llamadas registros, contienen los datos organizados en cada columna correspondiente dentro de una matriz de datos.

Tipo de Autenticación: basada en contraseña

En esta sección, implementamos la autenticación para clientes MQTT utilizando nombre de usuario y contraseña, como mencionamos anteriormente. Las contraseñas se almacenan de manera "hasheada". Esto significa que la contraseña se combina con el algoritmo SHA256 y se guarda en la base de datos como una cadena de caracteres, donde la contraseña original no es reconocible. Debido a que este proceso es unidireccional, no es matemáticamente posible recuperar la contraseña original a partir de los datos almacenados en la base de datos, incluso conociendo el algoritmo y el salt.

Configuración de Autenticación con MySQL

Paso 1: Archivo init.sql para Configurar la Base de Datos

Esta configuración se explica en caso de que se desee agregar nuevas tablas o registros, aunque ya se tenga el archivo. Creo archivo:

```
touch init.sql
```

entro a editarlo:

```
nano init.sql
```

dentro de el pongo tabla con atributos de interes:

```
  CREATE TABLE `mqtt_user` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT,
`username` varchar(100) DEFAULT NULL,
`password_hash` varchar(100) DEFAULT NULL,
`salt` varchar(35) DEFAULT NULL,
`is_superuser` tinyint(1) DEFAULT 0,
`created` datetime DEFAULT NULL,
PRIMARY KEY (`id`),
UNIQUE KEY `mqtt_username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Ahora agrego la informacion que voy almacenar en cada atributo, agrego un registro por cliente:

```
  INSERT INTO mqtt_user(username, password_hash, salt, is_superuser) VALUES
('emqx_u', SHA2(concat('public', 'slat_foo123'), 256), 'slat_foo123', 1);
Query OK, 1 row affected (0,01 sec)
```

Paso 2: orden de almacenamiento

1. Creo un directorio para el proyecto:

   ```
   mkdir mi_proyecto
   cd mi_proyecto
   ```

2. Guardo los siguientes archivos en el directorio creado:

   - `docker-compose.yaml`
   - `init.sql`

3. Creo directorio data y le doy permisos

   ```
   mkdir data
   chmod 777 data
   ```

Paso 3: Configuración de Docker

1. Creo una red Docker llamada `mi_red`:

   ```
   docker network create mi_red
   ```

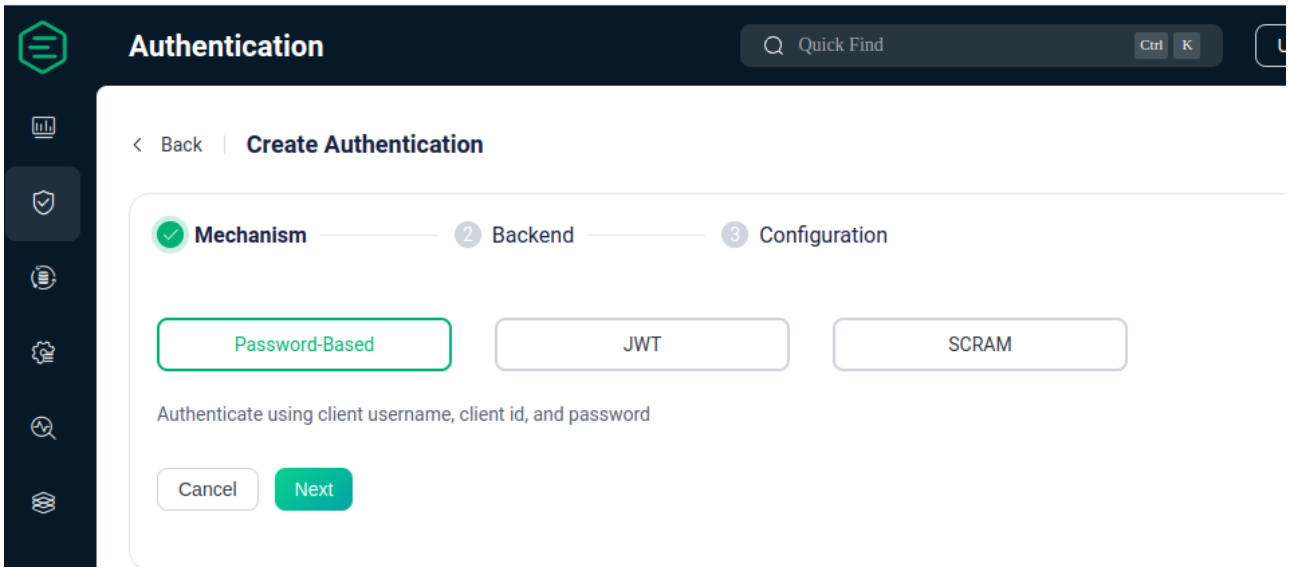
2. Levanto los contenedores utilizando `docker-compose`:

```
docker-compose up -d
```

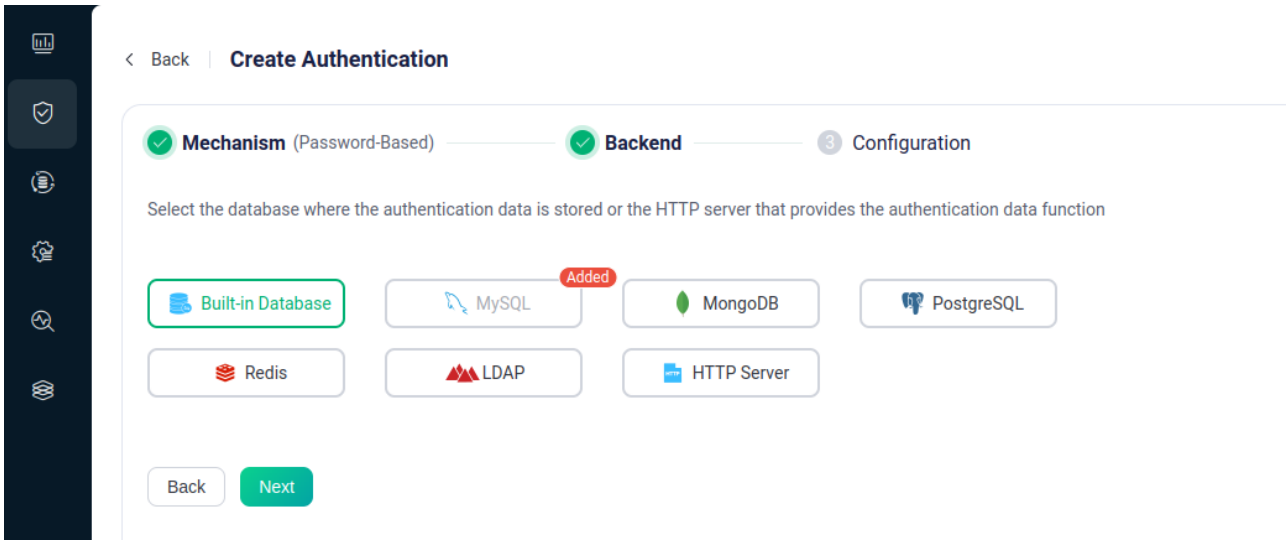
Paso 4: Configuración del Dashboard EMQX

Accede al dashboard de EMQX desde http://0.0.0.0:18083.

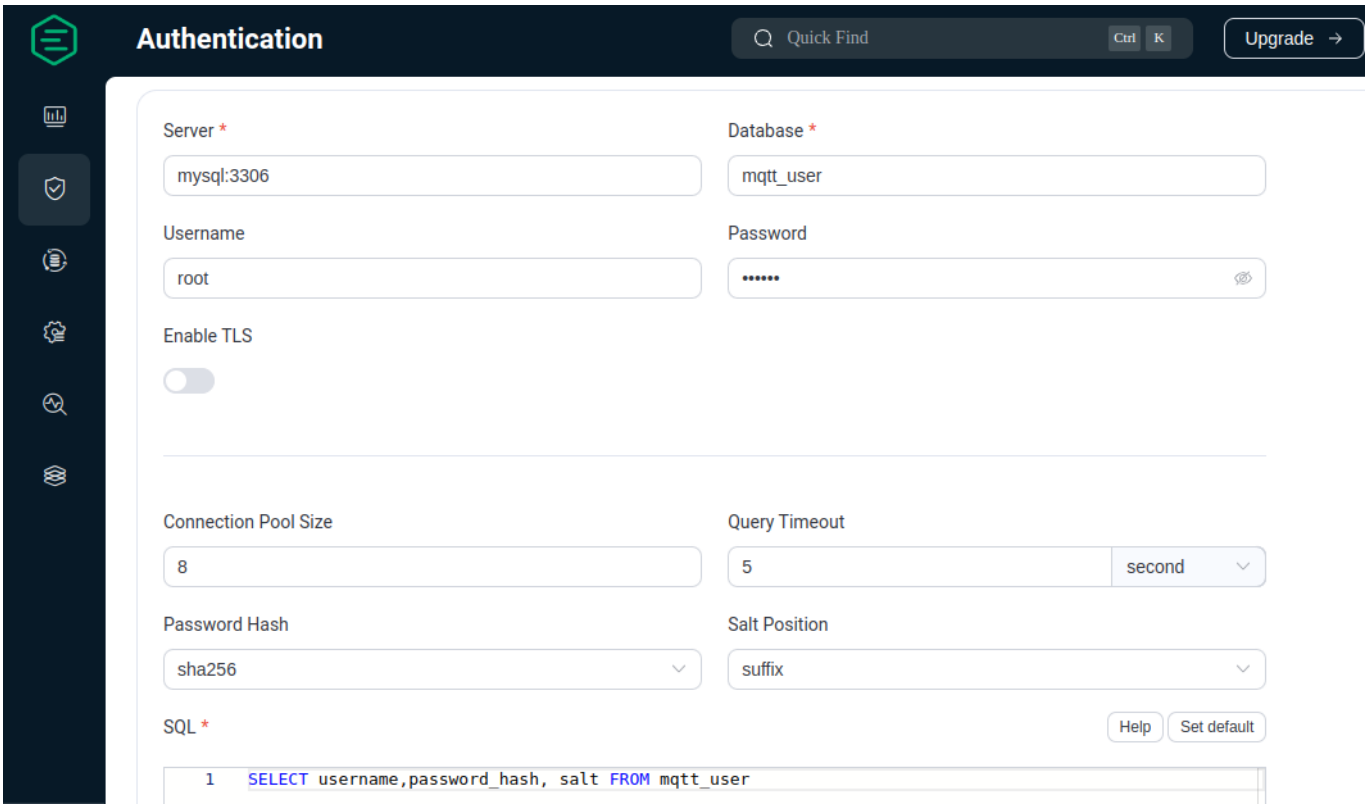
1. Selecciono la opción de autenticación y aprieto "create" y



2. luego "MySql"

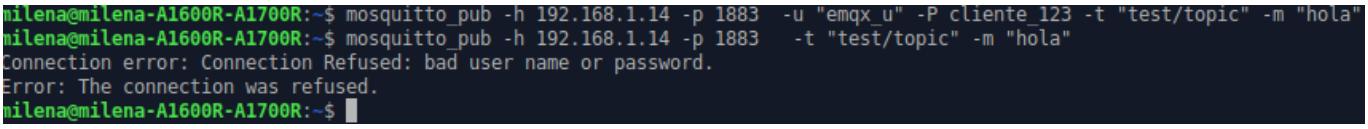


3. Presiono settings y configuro de la siguiente forma:



Pruebas con Mosquitto: Verificación de Funcionamiento

previo instalamos el mosquito y desactivamos el servidor, ya que usa el mismo puerto que usamos para las pruebas.



estructura del pub y sub:

```
  mosquitto_pub -h localhost  -p 1883  -u "usuario" -P contra -t
"algun/topico" -m "agun_mensaje"
```

```
   mosquitto_pub -h localhost  -p 1883  -u "usuario" -P contra -t
"algun/topico" -m "agun_mensaje"

# Certificados


## Teoria de funcionamiento:

Los portocolos utilizados en este caso son TLS (Transport Layer Security) y
SSL (Secure Sockets Layer) son protocolos criptográficos que proporcionan
autenticación y cifrado de datos entre el cliente ( que puede ser pub o
sub)  y el servidor (broker emqx).
Cuando nos referimos a que son protocolos criptograficos es por el hecho de
que en este protocolo se busca garantizar lo siguiente a la hora de
```

comunicarnos:

Confidencialidad
Los datos transmitidos están cifrados, lo que significa que solo las partes autorizadas pueden entender el contenido de los mensajes. Esto se logra mediante algoritmos criptográficos que transforman los datos en un formato ilegible sin la clave de descifrado correspondiente.

Integridad
TLS/SSL utilizan funciones criptográficas como hashes y firmas digitales para asegurar que los datos no sean alterados durante la transmisión. Esto garantiza que los destinatarios reciban los datos exactamente como fueron enviados, sin modificaciones no autorizadas.

Autenticación
Los protocolos TLS/SSL permiten la autenticación de las partes involucradas en la comunicación mediante el uso de certificados digitales. Estos certificados son emitidos por autoridades de certificación y contienen claves públicas que pueden ser verificadas para confirmar la identidad del emisor.

Seguridad en la Comunicación
En conjunto, estos principios criptográficos aseguran que las comunicaciones en redes, como las transacciones en línea, la transferencia de datos sensibles y la navegación web segura, estén protegidas contra la interceptación y la manipulación por parte de terceros malintencionados.

Proceso de cifrado

El proceso de comunicación en el protocolo TLS/SSL consta de dos partes. La primera parte es el protocolo de handshake (saludo). El propósito de este protocolo de handshake es identificar la identidad de la otra parte y establecer un canal de comunicación seguro. Después del handshake, ambas partes negociarán la suite de cifrado y la clave de sesión para la comunicación siguiente. La segunda parte es el protocolo de record (registro). Record es muy similar a otros protocolos de transmisión de datos. Lleva tipos de contenido, versión, longitud, carga, etc., y la diferencia es que la información llevada por este protocolo está cifrada.

La siguiente imagen describe el proceso del protocolo de handshake de TLS/SSL, desde el "hello" (saludo) del cliente hasta el "finished" (terminado) del broker.

Tipo de cifrado utilizado en este caso es "two- way"
La certificación bidireccional es que se requiere un certificado para el servicio y el cliente durante la autenticación de la conexión. Ambas partes deben realizar la autenticación para garantizar que se confíe en ambas partes involucradas en la comunicación. Ambas partes comparten sus certificados públicos y luego realizan la verificación y confirmación en función del certificado

```
## Pasos de la configuracion

#### creacion de certificados

En primer lugar creo en mi_proyecto un directorio llamado certs y  me
posisiono dentro de esa carpeta.

```bash
    mkdir certs
```

le doy a esa carpeta todos los permisos:

```
    chmod 777 certs
```

instalo open SSL

```
    sudo apt install openssl
```

Genero el certificado ca.pem autofirmado:

primero genero clave privada para firmarlo:

```
    openssl genrsa -out ca.key 2048
```

luego lo creo

```
   openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.pem
```

Ahora genero certificados para el servidor (emqx)

en principio clave privada:

```
    openssl genrsa -out emqx.key 2048
```

luego creo archivo openssl.cnf

```
    touch openssl.cnf
```

```
    nano openssl.cnf
```
```

lo edito con el siguiente archivo cambio SOLO donde dice broker_address, alli pongo la direccion ip de mi emqx :

```
    [req]
default_bits  = 2048
distinguished_name = req_distinguished_name
req_extensions = req_ext
x509_extensions = v3_req
prompt = no
[req_distinguished_name]
countryName = CN
stateOrProvinceName = Zhejiang
localityName = Hangzhou
organizationName = EMQX
commonName = CA
[req_ext]
subjectAltName = @alt_names
[v3_req]
subjectAltName = @alt_names
[alt_names]
IP.1 = BROKER_ADDRESS
DNS.1 = BROKER_ADDRESS
```

luego uso esa clave y configuracion para solicitar un certificado

```
openssl req -new -key ./emqx.key -config openssl.cnf -out emqx.csr
```

por ultimo uso el ca.pem para generar el certificado de emqx:

```
    openssl x509 -req -in ./emqx.csr -CA ca.pem -CAkey ca.key -
CAcreateserial -out emqx.pem -days 3650 -sha256 -extensions v3_req -extfile
openssl.cnf
```

Luego genero certificado del cliente clave del cliente:

```
openssl genrsa -out client.key 2048
```

archivo de solicitud de cliente:

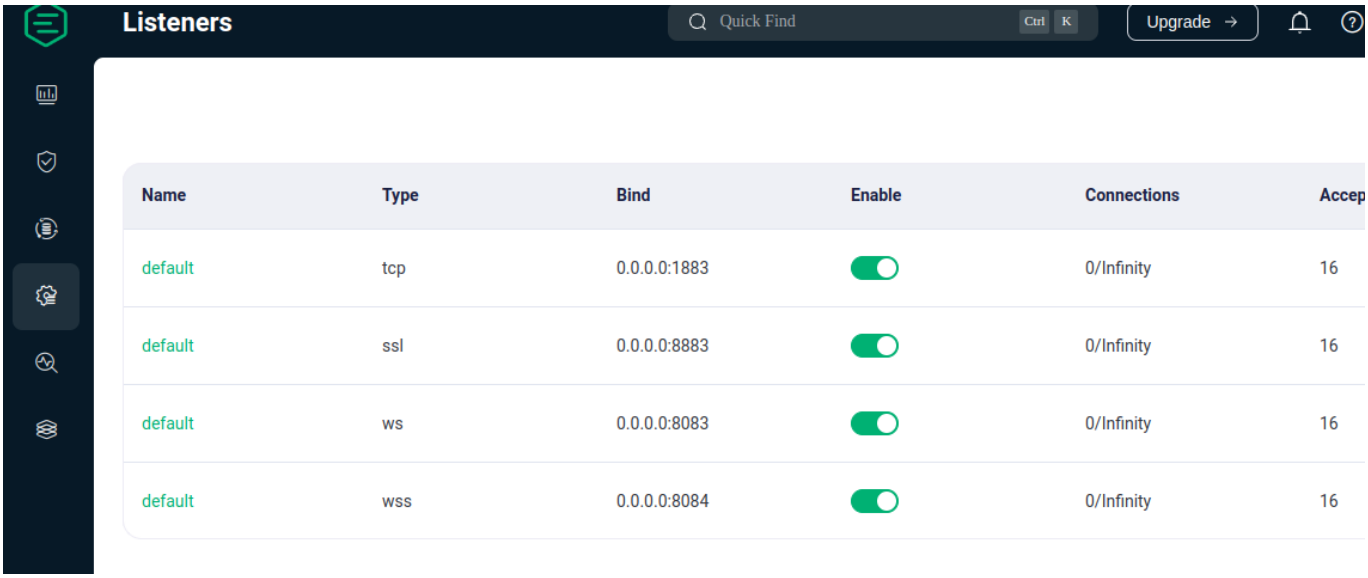
```
   openssl req -new -key client.key -out client.csr -subj
"/C=CN/ST=Zhejiang/L=Hangzhou/O=EMQX/CN=client"
```

firmo y genero certificado de cliente:

```
   openssl x509 -req -days 3650 -in client.csr -CA ca.pem -CAkey ca.key -
CAcreateserial -out client.pem
```

configuracion del dashboard

Dentro de "Listeners" en la columna izquierda, selecciona el listener por default para SSL.



Carga los archivos de encriptación en el siguiente orden: `emqx.pem`, `emqx.key`, `ca.pem`, luego presiona "Update". y pongo Force Verify Peer Certificate en true y habilito verify Peer