MQTT Manager Documentation

Release 0.2.0a1

Matheus Magalhaes

Apr 06, 2018

Contents:

This container bundles mosquitto with MQTT Manager. Mosquitto is an open source message broker that implements the MQTT protocol. MQTT Manager provides a REST service to update mosquitto access control list (ACL) and TLS options easily and 'on the fly'.

CHAPTER 1

MQTT-TLS Tutorial

This document describes how to configure dojot to use MQTT over TLS.

Table of Contents

1.1 tl;dr

For a device to connect using TLS with Mosquitto, it must possess:

- A key pair (.key file);
- A certificate signed by a Certificate Authority (CA) trusted by Mosquitto (.crt file);
- The certificate of this CA (.crt file);
- An entry on Mosquitto Access Control List (ACL), allowing the device to publish on a specific topic;
- (optional) A Certificate Revocation List (CRL).

When a device is created, DeviceManager will automatically notify the following components:

- IoTAgent: will register the new device on its internal cache.
- MQTT-Manager: will create an entry on the ACL, allowing the device to publish on a specific topic.
- EJBCA: will create an end entity so a certificate can be created on the future.

By default, dojot uses clear MQTT. To activate TLS, docker-compose.yml must be changed:

- The image for service 'mqtt' must be changed from 'ansi/mosquitto' to 'dojot/mqtt-manager';
- A new entry must be added to 'mqtt' service exported port list: "9010:9010"
- The public port for 'mqtt' service must be changed from '1883:1883' to '8883:8883';
- The MQTT_TLS variable of 'iotagent' service must be set to true (lowercase).

On the configuration file 'iotagent/config.json':

- The flag 'secure' should be changed to true

1.2 Components

1.2.1 EJBCA-REST

EJBCA is a complete Private Key Infrastructure (PKI) capable to manage CAs, cryptography keys and certificates. EJBCA provides a SOAP, web and a command line interface. EJBCA-REST is an wrapper on top of EJBCA that provides modern interfaces, like REST and Kafka.

EJBCA provides SOAP, web and command line interfaces. EJBCA-REST is a wrapper on top of EJBCA that complements those, allowing the CA to be configured using REST. When used within dojot, it also listens to Kafka events, allowing its automatic configuration.

What is a certificate?

A certificate contains the public key for an entity (a user, device, website), along with information about this entity, about the CA which signs the certificate, the allowed certificate usage and a checksum. When a entity wants a certificate to be signed, the entity should create a CSR file and send it to the desired CA. The CSR file is an 'intention of certification'. The file contains the information required from the entity and some information about the certificate use, hostnames and IPs where the certificate will reside, alternative names for the entity, etc. EJBCA can decide, using its configured policies, what information to keep, to discard and to overwrite of the received CSR. EJBCA can refuse to sign a CSR if it concludes that it is not safe enough according to its policies.

These configurable policies are called 'Certificate Profiles'. One Certificate profile named CFREE, specialized for MQTT TLS, is provided out of the box.

In short, CFREE have the following configurations (and many more):

- Cryptography keys must have between 2048 and 8192 bits;
- Certificate expires in 730 days;
- Entities can define hostnames and IPs;
- Key usage is marked as not critical (for now);
- The hash algorithm is SHA256. The sign algorithm is RSA.

I have a CSR. How can I ask EJBCA to sign it for me?

Calm down! EJBCA will not allow strangers to ask for certification. You need to authenticate yourself. EJBCA use a username+password authentication system. The term 'end entity' will be used to refer to EJBCA users to follow the terms on EJBCA documentation and to avoid ambiguities between EJBCA users and dojot users. An administrator should create the end entity. An entity that was just created has the state 'New' an can generate a certificate. After signing a certificate for an entity, the end entity's state changes to 'Generated' and will no longer accept this username and password. EJBCA 'End entities' can create only one certificate.

So, how does EJBCA work in dojot?

When creating a new device, an associated end entity is created in EJBCA. Its name will be the device's ID (like '8fa3') and its password will be always 'dojot'.

A certificate can be signed by sending a HTTP POST request to host:1234/sign/<cname>/pkcs10. CName is the end entity's name (or device). The payload sent with this request should be a JSON containing the end entity password and a CSR file (certificate intention) in base64 format.

Note that the URL is 'routed' by the API gateway. As in other APIs in dojot, a JWT is needed. You can find how to generate and how to use such token in User Guide.

In order to create the CSR file and ask for a certificate signature, a user can use a helper script called 'Certificate Retriever', which is detailed in *Certificate retriever* section.

1.2.2 MQTT Manager

MQTT-Manager is a helper service used to configure Mosquitto MQTT broker in a simple and 'on-the-fly' way. It can be configured using REST interfaces and Kakfa. Thus, HTTP requests or Kafka messages can be used to create and remove devices, as well as update CRL file (certification revogation list). This service is distributed as a docker container for easy deploy and its source code repository can be accessed in MQTT Manager repository.

Mosquitto by itself doesn't generate nor revoke certificates, it only rely upon a CA and implements TLS protocol. The 'creation' of a particular device consists only in adding a new rule to ACL file in Mosquitto. Such file looks like:

```
user iotagent
topic read /#
user 24f6
topic write /admin/24f6/attrs
```

Each rule is composed by two lines: the first one specifies the user (device) and the second one defines which action (write or read) is allowed to which topic. In the example above, the user iotagent can read all topics (# is a wildcard). Also, the device with ID 24f6 can write to topic /admin/24f6/attrs. The device ID is retrieved in 'Common name' certificate field.

If a device sends data to a topic which it has no write permissions, then all data is discarded. Mosquitto won't log any errors related to this.

When the ACL is changes, Mosquitto must be restarted (or a SIGDUP signal can be sent to its process). MQTT-Manager does this automatically when creating or removing devices.

A script is executed when firing the container up. This script will generate a pair of keys to Mosquitto, retrieves the certificate and CRL from a CA and asks it to sign its public key. ALl generated files are placed in /usr/local/src/mosquitto-1.4.13/certs (inside the container).

Mosquitto will only accept device connections that have certificate signed by its trusty CA.

Also note that MQTT-Manager is used only in case when a TLS-enabled broker is needed. If this is not the case, then the vanilla Mosquitto docker image can be used.

1.3 Mosquitto configuration files

Checkout this commented Mosquitto configuration file:

```
# network port on which Mosquitto will accept new connections
port 8883

# Trusted CA certificate
cafile /usr/local/src/mosquitto-1.4.13/certs/ca.crt

# Mosquitto certificate
certfile /usr/local/src/mosquitto-1.4.13/certs/mosquitto.crt

# Mosquitto key par
keyfile /usr/local/src/mosquitto-1.4.13/certs/mosquitto.key

tls_version tlsv1.2

# If false, a device will check Mosquitto certificate, but Mosquitto won't check
# the device counterparts.
# If true, both checks are performed (2-way TLS)
require_certificate true

# Certificate Common Name field will be used as username.
# Thus, a device with 'CN=abc1' will have a 'user abc1' entry in Mosquitto's ACL
use_identity_as_username true

# Permission list file
acl_file /usr/local/src/mosquitto-1.4.13/certs/access.acl

# CA CRL.
crlfile /usr/local/src/mosquitto-1.4.13/certs/ca.crl
```

Note that for all configuration updates, it is mandatory to restart Mosquitto or to send a SIGDUP signal to its process.

1.4 Certificate retriever

This component is a helper script for device certificates creation. It is available at Certificate Retriever GitHub repository and it coded using Python 3.

A user can use it by executing:

```
./certificate-retriever.py HOST DEVICE-NAME CA [OPTIONS]
```

The mandatory parameters are:

- HOST: where dojot is. Example: http://localhost:8000
- DEVICE-NAME: device name that will get a new certificate. Example: ac32
- CA: CA which will sign the certificate. Example: IOTmidCA (this is the CA name used in dojot)

Other options are:

- -u or –username USERNAME: dojot's username. If this parameter is not specified here, it will be asked iteratively.
- -w or –overwrite: overwrites any certificate files or criptographic keys if already existent.
- -k or –key KEYLENGTH: size of the criptographic key being generated (in bits).
- -d or –dns: Hostname where the certificate owner can be reached out. Note that this has no relation with DNS (Domain Name System) servers - this name was kept because x509 certificates have an attribute that is called DNS.
- -i or –ip: same as -d, buto to specify IP address.
- –skip-https-check: if dojot accepts HTTPS connections but it has no valid certificate, then this option will allow the connection to be made.

Note that authentication is performed in dojot. The script will ask for user credentials and will invoke user authentication automatically. The user needs permission for certificate signing to be able to use this script.

An end entity must exist in EJBCA in 'New' state before asking for a new certificate signature. When a new device is created, an end entity is automatically created in EJBCA by DeviceManager. This new end entity's name is the device ID itself. Its password is 'dojot'.

The script authenticate users with given username and password, retrieve CA certificate, generate a key pair as well as a CSR file and asks for certificate signature, in this order. Any error in any step will halt its execution.

After successfully executed, all certificates can be found in './certs' folder.

1.5 Important Notes

These are a few but important notes related to device security and associated subjects.

1.5.1 CRL (Certification Revocation List)

A CRL is a list which contains all revoked certificates. It is used to indicate which certificates are no longer valid (administratively set to invalid) as a normal certificate can be used for 1 to 5 years. This list is signed by CA and also has an expiration date - 1 day by default. In TLS protocol, if CRL is expired then the recommended action to be taken is to refuse all incoming connections, as there is no way to check if the certificates used in those connections are invalid or not. This procedure is implemented in Mosquitto.

Therefore, CA must generate a new list periodically. All components that use it must be updated.

1.5.2 Debugging

TLS error might be not so verbose as other problems. If an error occurrs, the user might not know what went wrong because no component indicates any problem. In this section there are some tips, frequent problems and debugging tools to find out what's happening.

How to read a certificate

A certificate file can be in two formats: PEM (base64 text) or DER (binary). OpenSSL offers tools to read such formats:

```
openssl x509 -noout -text -in certFile.crt
```

To read a CRL:

```
openssl crl -inform PEM -text -noout -in crlFile.crl
```

Errors in secure connection handshake between device and Mosquitto

If any errors occur during connection handshake, something like the following error might appear in Mosquitto's logs:

```
1514550332: New connection from 172.20.0.1 on port 8883.
1514550332: OpenSSL Error: error:140940E5:SSL routines:ssl3_read_bytes:ssl handshake␣
↪failure
```

If this happens, try to establish connection using 'openssl client', as it is more verbose in error description.

```
openssl s_client -connect localhost:8883 -CAfile ca.crt -cert device.crt -key device.
↪key
```

Common errors are shown by openssl_client (and _server as well):

- SSL alert number 45: this error indicates that a certificate expired. Keep in mind that CRL also expires.
- SSL alert number 48: received a valid certificate chain or partial chain, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA. This message is always fatal.
- Alert unknown CA: check whether sent CA certificate is correct. If it is a sub-CA, check if all of its certificate chain was sent. This error also occurs if the CA certificate data (specially common name attribute) is the same as those from client certificate.

Handshake is OK, but no published data reaches iotagent

You can check whether the device could connect to MQTT broker by checking Mosquitto's log:

```
1514482004: New client connected from 172.20.0.10 as mqttjs_c011c22d (c1, k10, u
↪'deviceName')
```

If that line shows up, it means that the TLS handshake worked and the device successfully connected to Mosquitto. Check if the device has an ACL entry in Mosquitto to allow it to publish data in the specified topic. Keep in mind that if a device publishes something in another topic (which it has no permission to publish) all data is discarded by Mosquitto with no warnings.

CHAPTER 2

How to build/update/translate documentation

If you have a local clone of this repository and you want to change the documentation, then you should follow this simple guide.

2.1 Build

The readable version of this documentation can be generated by means of sphinx. In order to do so, please follow the steps below. Those are actually based off ReadTheDocs documentation.

```
pip install sphinx sphinx-autobuild sphinx_rtd_theme sphinx-intl
make html
```

For that to work, you must have pip installed on the machine used to build the documentation. To install pip on an Ubuntu machine:

```
sudo apt-get install python-pip
```

To build the documentation in Brazilian Portuguese language, run the following extra commands:

```
sphinx-intl -c conf.py build -d locale
make html BUILDDIR=build/html-pt_BR O='-d build/doctrees/ -D language=pt_BR'
```

2.2 Update workflow

To update the documentation, follow the steps below:

1. Update the source files for the english version
2. Extract translatable messages from the english version

```
make gettext
```

3. Update the message catalog (PO Files) for pt_BR language

```
sphinx-intl -c conf.py update -p build/gettext -l pt_BR
```

4. Translate the messages in the pt_BR language PO files

This workflow is based on the Sphinx i18n guide.

CHAPTER 3

How to build

The recommended way to build MQTT Manager is to build a Docker image:

```
docker build -t mqtt-manager .
```

If you'd like to run it by hand, MQTT Manager has the following dependencies (these packages are related to Ubuntu. Other Linux distributions might have different names for them):

- python-openssl
- python-pip

And there are a few other dependencies from pip:

- flask
- requests
- kafka

There are a few things that must be moved around in order to flask properly find them. You can check the Dockerfile to find out what they are and where do they should go.

CHAPTER 4

How to run

All MQTT Manager and mosquitto dependencies should be automatically downloaded and configured when building the container.

If running in standalone mode (without Docker and other elements from dojot's docker-compose), check the entrypoint script.

Both methods depend on a running instance of EJBCA-REST.