SSL/TLS Client Certs to Secure MQTT

When I started to experiment with MQTT it was pretty easy to find information on using SSL/TLS Certificates to encrypt communications with the broker. The OwnTracks project even provides a script to setup a certificate authority (CA) and sign the certificates. However the certificates are for the broker (server) and not the client.

I was less successful in finding easy to follow instructions for creating and using SSL certificates to authenticate MQTT clients. This exercise attempts to remedy that lack of instructions.

A note on terminology: TLS (Transport Layer Security) is the new name for SSL (Secure Sockets Layer). The StackExchange answer to "What's the difference between SSL, TLS, and HTTPS?" provides a more nuanced explanation.

Requirements

- The certificates will be generated with `openssl` which is probably already installed by your distribution (I used Ubuntu 14.04 LTS to develop and test this procedure).
- Demonstrations will be done with the open source MQTT broker, `mosquitto`. It was easy to install using `apt-get`:

```
$ sudo apt-get install mosquitto
```

Procedure

NOTE: This procedure assumes all the steps will be performed on the same system.

1. Setup a protected workspace

Warning: the keys for the certificates are not protected with a password. This is not the best practice, but it does make it easier to use them with daemons and embedded devices. It is important that you keep them secret.

For this exercise we'll create and use a directory that does not grant access to other users.

```
$ mkdir myCA
$ chmod 700 myCA
$ cd myCA
```

2. Setup a CA and generate the server certificates

Download and run the `generate-CA.sh` script from the OwnTracks project. The script creates the certificate authority (CA) files, generates server certificates, and uses the CA to sign the certificates.

NOTE: Before using this procedure in a production environment, you should probably customize the `generate-CA.sh` script, but that is beyond the scope of this exercise.

```
$ wget https://github.com/owntracks/tools/raw/master/TLS/generate-CA.sh
.
$ bash ./generate-CA.sh
```

`generate-CA.sh` produces 6 files: `ca.crt`, `ca.key`, `ca.srl`, `myhost.crt`, `myhost.csr`, and `myhost.key`. There are certificates (`.crt`), keys (`.key`), a request (`.csr`) and a serial number record file (`.slr`) used in the signing process. Note that the `myhost` files will have different names on your system!

Three of them get copied to the `/etc/mosquitto/` directories:

```
$ sudo cp ca.crt /etc/mosquitto/ca_certificates/
$ sudo cp myhost.crt myhost.key /etc/mosquitto/certs/
```

They are referenced in the `/etc/mosquitto/mosquitto.conf` file like this:

```
# mosquitto.conf
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/myhost.crt
keyfile /etc/mosquitto/certs/myhost.key
```

After copying the files and modifying the `mosquitto.conf` file, restart the server:

```
$ sudo service mosquitto restart
```

3. Checkpoint

You can verify the work to this point by using `mosquitto_sub` client:

```
$ mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile ca.crt
$SYS/broker/bytes/received 65
$SYS/broker/bytes/sent 67
$SYS/broker/bytes/received 130
$SYS/broker/bytes/sent 196
^C
$
```

The topics are updated every 10 seconds. If debugging is needed you can add the `-d` flag to `mosquitto_sub` and/or look at `/var/logs/mosquitto/mosquitto.log`.

4. Generate client certificates

At this point you could try what I did and use `generate-CA.sh` to generate another certificate, but don't! It won't work because the certificates it creates have nsCertType set to server so they won't work for a client. After playing with the script a bit and reading the manual pages for openssl, I found the following three commands would create the certificates I need:

```
$ openssl genrsa -out client.key 2048
$ openssl req -new -out client.csr \
```

```
  -key client.key -subj "/CN=client/O=example.com"
$ openssl x509 -req -in client.csr -CA ca.crt \
  -CAkey ca.key -CAserial ./ca.srl -out client.crt \
  -days 3650 -addtrust clientAuth
```

The important argument is `-addtrust clientAuth`. It makes the resulting signed certificate suitable for use with a client.

That's simple enough, but you will probably want to have a number of different client certificates. One per client is not unreasonable. So the attached script is a complement to `generate-CA.sh`. Use it like this:

```
$ bash ./generate-client.sh client2
```

5. Reconfigure

Change the `mosquitto` configuration to require client certificates by adding the `require_certificate` line to the end of the `/etc/mosquitto/mosquitto.conf` file so that it looks like this:

```
# mosquitto.conf
pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/SVE14A1HFXB.crt
keyfile /etc/mosquitto/certs/SVE14A1HFXB.key
require_certificate true
```

Restart the server:

```
$ sudo service mosquitto restart
```

6. Test

The `mosquitto_sub` command we used above now fails:

```
$ mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile ca.crt
Error: Protocol error
```

I recently (2015/11/01) retried this and the error message was: `Error: A TLS error occurred`.

Adding the `--cert` and `--key` arguments satisfies the server:

```
$ mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile ca.crt --cert
client.crt --key client.key
$SYS/broker/bytes/received 65
$SYS/broker/bytes/sent 67
$SYS/broker/bytes/received 130
$SYS/broker/bytes/sent 136
^C
$
```

The second client certificates work too:

```
$ mosquitto_sub -t \$SYS/broker/bytes/\# -v --cafile ca.crt --cert
client2.crt --key client2.key
$SYS/broker/bytes/received 130
$SYS/broker/bytes/sent 198
$SYS/broker/bytes/received 195
$SYS/broker/bytes/sent 269
^C
$
```

Conclusion

Using client certificates, when you can, provides another layer of security to your MQTT system. Now in addition to having an encrypted communications channel, the server will only accept connections with a properly signed certificates.

There is still a lot of room for improving the security of your MQTT system, but encrypted communications is a good start.

Follow-up

Contact me about this exercise by commenting on my Google+ post.

generate-client.sh (3k) JERRY DUNMIRE, Feb 13, 2015, 7:46 v.1

Comments

You do not have permission to add comments.