GSoC Phase 1

Author: Vedang Joshi

Published: July 17, 2021 · 5 min read

In the past month, I worked on a feature in DRG, which enables a user to create and add Trust anchors to the application object, and use them to sign the device certificates.

Implementation

Drogue cloud tries to make it as easy as possible for devices to connect to the cloud. In today's IoT world, a convenient and secure solution to authenticate devices is to use digitally signed certificates. We first need to obtain a public and private key pair which can be generated by algorithms like RSA, DSA, or ECDSA. An X.509 certificate is the envelope that holds a public key along with some other information about the usage and ownership of the certificate and the issuer. In Drogue IoT, we have applications (aka apps), which are containers for devices. The flow for X.509 authentication looks like this:

- We create a new key for an application. This is the root CA key, it needs to be stored securely. Let's name it - app key.
- We create a certificate and sign it using the app key. This is a self-signed certificate, and it would work are the Trust anchor or root CA.
- We store this certificate in the application object on the cloud side.

Now, we have a trust anchor set up and ready to sign our device certificates. To do that:

- We download the Trust anchor certificate from the application object. This helps us add "issuer" information on the device certificate and also verify the private key.
- We take the app key as input.
- Then, we create a CSR i.e certificate signing request and sign it with the app key to create the device certificate.
- We can now export the certificate and private key for the device either to a file or to the terminal.

This concludes the flow, now we can use this device certificate and its key to authenticate and send data to the drogue cloud. This may sound like a lot to do, especially if you are not familiar with tools like OpenSSL, don't worry, DRG is here to rescue. DRG abstracts this entire process in not more than 2 commands.

- `drg trust create` : Creates an app key, Trust anchor, and adds it to the application object.
- `drg trust enroll` : Uses the app key to sign and export device certificate and key.

We have focused on having a pure rust implementation for all the cryptography we have used here. `rcgen` crate is just amazing and makes handling X.509 certificates very easy. Currently, We using the ECDSA algorithm to generate the key pairs, this is the default for the rcgen crate, but we are working on generating and using RSA key pairs as well.

While implementing this, I started by creating a script that uses OpenSSL to generate certificates and authenticate devices. For a few days, I would create a Trust Anchor correctly but couldn't authenticate any device. On sharing this with mentors, we discovered that it is because of a broken dependency but it got fixed pretty quickly. Once, my bash script worked successfully, I knew exactly what I had to do with `rcgen` . The majority of the work on this feature has been done in this PR. I am happy to say it is now merged to the master branch.

Here's a quick demo

To use X.509 authentication, please follow the following steps. Assuming that you are logged into drogue cloud if not use `drg login <drogue-cloud-endpoint>` .

- `drg create app <app_name>` - This commands create an application.
- `drg create device <device_name> --app <app_name> --cert` - This command creates a new device. Notice, the `--cert` flag used here, it formats the device name according to what is needed by Drogue cloud for certificate authentication.
- `drg trust create <app_name> --key-output <app-key-filename.pem>` - This command generates a new key and a certificate, uses the key to sign the cert, finally uploads the certificate to the application object, and writes the key to the file. Optionally, you may specify `--days <no_of_days>` argument, for the validity, by default it is 365.
- `drg trust enroll <device_name> --app <app_name> --ca-key <app_key_filename.pem> --key-output <deivce_key_filename.pem> --out <deivce_cert_filename.pem>` - This commands uses the app key to sign a newly generated device certificate + key. This certificate and key are exported to the respective files.

Now, we can use this device certificate and key to authenticate the devices. An example to do the same over MQTT using the MQTT-CLI (v4.6.2) tool would look like.

```
mqtt pub -h <mqtt_host> -p 443 --cert device-cert.pem --key device-key.pem
```

To Do

This implementation has many things to improve, some of them are:

- RSA Key Pair, This is currently "work in progress".
- Using existing private keys and existing certificates, in that case, DRG works only as a medium to upload them to the cloud.
- Trying to use it with a key store solution like LastPass, could be a good topic for a blog post.
- Improving the implementation.

Thank you

Thanks to @jbtrystram, @ctron and everybody in the community for helping me and as always Thanks for reading.

GSOC DrogueIoT IoT Github Open source