Julien Vermillard Follow
Software developer, opensourcer. Eclipse IoT fellow.
Sep 6, 2016 · 4 min read

Setup Cloudflare CFSSL with OCSP responder

CFSSL is both an HTTP based Public Key Infratructure (PKI) Certificate Authority server, an Online Certificate Status Prootcol responder and a PKI toolkit.

It's a very flexible and simple software stack but the doc is somewhat, hum missing or outdated :)

First, the installation:

Follow the first step of https://github.com/cloudflare/cfssl

Install go (https://golang.org)

```
$ go get -u github.com/cloudflare/cfssl/cmd/...
```

You have all the cfssl commands ready to use! Now we want to setup the database for storing the issued certificates and the associated revocation status.

Don't go with MySQL, I found some bug with cfssl support for MySQL. So if you don't want to debug cfssl I advise you to start with PostgreSQL.

Start a PostgreSQL container:

```
$ sudo docker run -p 5432:5432 --name=postgres-cfssl -e
POSTGRES_PASSWORD=cfssl -d postgres:latest
```

Then you need to create the database schema. For that you need to use the goose utility:

```
$ go get -u bitbucket.org/liamstask/goose/cmd/goose
```

Run the migration scripts:

```
$ goose -path
$GOPATH/src/github.com/cloudflare/cfssl/certdb/pg up
```

You need to create a database configuration file for cfssl server to be able to connect to the database. My **db-pg.json** contains:

```
{"driver":"postgres","data_source":"postgres://postgres:cfss
l@localhost/postgres?sslmode=disable"}
```

Now we want to create our root CA and our intermediate CA. The Root CA will be just used for signing the intermediate CA. The intermediate CA will be used for signing the end entity certificates.

CA operation is out of scope of this document but this Root CA must be generated and stored offline.

Now create a signature request configuration file **ca-csr.json**

```
{
 “CN”: “My PoC root CA”,
 “key”: {
 “algo”: “ecdsa”,
 “size”: 256
 },
 “names”: [
 {
 “C”: “FR”,
 “L”: “Toulouse”,
 “OU”: “ACME operations”
 }
 ]
}
```

You need to customize the different fields. Here I’m using ECDSA signature, but you can choose to use RSA depending of your needs.

Then run the certificate generation command:

```
$ cfssl gencert -initca ca-csr.json -loglevel=0| cfssljson -
bare ca -
```

It'll produce two important files: the private key **ca-key.pem** and the self signed certificate **ca.pem**

Now before creating the intermediate CA which will use for signing the end entities, we need to create a cfssl config file for defining all the different profiles of certificates we are going to produce.

```
{
    "signing": {
        "default": {
            "ocsp_url": "http://localhost:8889",
            "crl_url": "http://localhost:8888/crl",
            "expiry": "26280h",
            "usages": [
                "signing",
                "key encipherment",
                "client auth"
            ]
        },
        "profiles": {
            "ocsp": {
                "usages": ["digital signature", "ocsp
signing"],
                "expiry": "26280h"
            },
            "intermediate": {
                "usages": ["cert sign", "crl sign"],
```

```
                "expiry": "26280h",
                "ca_constraint": {"is_ca": true}
            },
            "server": {
                "usages": ["signing", "key encipherment",
"server auth"],
                "expiry": "26280h"
            },
            "client": {
                "usages": ["signing", "key encipherment",
"client auth"],
                "expiry": "26280h"
            }
        }
    }
}
```

The default properties will be shared by all the produced certificates. Be careful to correctly configure the OCSP and CRL URLs. They must point to the final externally accessible URL for your intermediate CA CRL (certificate revocation list) and OCSP HTTP endpoint. I'm using localhost here for the proof-of-concept, but all the generated certificate will need to be reissued when you deploy the real production CA with non localhost URLs.

Again create a signature request configuration file **server-ca.csr.json**:

```
{
    "CN": "My intermediate CA",
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
```

```
        {
            "C": "FR",
            "L": "Toulouse",
            "OU": "ACME operations"
        }
    ]
}
```

Create the intermediate key and root CA signed certificate:

```
cfssl gencert -ca ca.pem -ca-key ca-key.pem -config="cfssl-
config.json"  -profile="intermediate" server-ca.csr.json|
cfssljson -bare ca-server -
```

Again the important created files are: ca-server-key.pem and ca-server.pem.

Now for signing OCSP response we need to create a specific certificate. This certificate must be signed by the intermediate CA because it's delivering status for the intermediate CA.

```
cfssl gencert -ca ca-server.pem -ca-key ca-server-key.pem -
config ../cfssl-config.json -profile="ocsp" ocsp.csr.json|
cfssljson -bare server-ocsp -
```

Now we can run our Certificate Authority API server and start delivering end-entity certificates!

```
cfsssl serve -db-config=db-pg.json -loglevel=0  -ca-key=ca-
server-key.pem -ca=ca-server.pem -config=./cfssl-config.json
-responder=server-ocsp/server-ocsp.pem -responder-
key=server-ocsp/server-ocsp-key.pem
```

How to use our brand new CFSSL server to process a certificate request: first generate a key and a CSR for the end entity:

```
#create openssl key configuration file for using ECSA
openssl ecparam -name prime256v1 -out prime256v1.pem
```

```
#generate a private key and a certificate request
openssl req -new -newkey ec:prime256v1.pem -nodes -keyout
client.key -out client.csr
```

```
#request a certificate signature from the cfssl server
 cfssl sign -remote "localhost:8888" -profile "client"
client.csr |cfssljson -bare my-client -
```

Now since we have a certificate in our database, we want to be able to answer OCSP requests.

CFSSL don't generate OCSP response on the fly, but use pre computed file, ready to serve. So from time to time you need to generate this file

and restart your OCSP responder.

Pre-generate the OCSP response:

```
cfssl ocspdump -db-config db-pg.json> ocspdump.txt
```

By default OCSP response are expiring in 96hours.

Now start the OCSP responder:

```
cfssl ocspserve -port=8889 -responses=ocspdump.txt  -
loglevel=0
```

How to test our OCSP responder ?

```
openssl ocsp -issuer bundle.pem -no_nonce -cert my-
client.pem -CAfile ca-server.pem -text -url
http://localhost:8889
```

And that's all!