Building an Internal PKI

Using Open-Source Tools

INTRODUCTION

Who Am I?

Craine Runton

Denver-metro local

Have worked in IT and Security for over ten years

Hold CISSP, CISSP-ISSAP, and CISA certifications, if you care about that sort of thing…

Occasionally write technical things at `www.runtonsecurity.com`

Occasionally put projects I work on at `craine.gitlab.io`

What Do I Do?

Manager of Security @ Distil Networks

- Manage the Security Operations & Engineering efforts
- Security architect for Distil's infrastructure

Founder of Log the Crux

- Web application for rock climbers

Open Source programmer

Project Background

The Startup Way

Used a number of wild card certificates from a number of CA vendors

Managing the private keys is a nightmare

Single-level wildcards means you end up with a mishmash of hostnames, domains, and FQDNs

- Was that host `staging-ops-api.example.it`? Or was it `ops-api-staging.example.us`?

Renewals across a large number of certificates is manual and high-risk

- 250 servers all have a certificate expire on the same day and it needs to be renewed manually? Sure thing. Where's the whiskey?
- Wait, which servers are using the **.it** cert and which are using the **.us** cert? What do you mean the hostname doesn't match the app's FQDN? Where's the kerosene?

Why we needed to change

Implementing a number of projects that required TLS certificates

- Needed to support both Server & Client Authentication certificates

Wanted to advance our security posture, and control our certificates & private keys better

- Single private key per host & per application
- Rotate private keys more often than one per year
- Certificates that matched the DNS name of an application or host

Overview of Available Tools

What's Out there?

Active Directory Certificate Services

- Built into Microsoft AD
- (Mostly…) Windows dependent

Cloudflare SSL (CFSSL)

- Open sourced internal CF project
- Cross-platform

EJBCA

- Complicated docs & setup
- Java-based

The OpenCA Project

- Last stable-branch update in 2013
- No bug fixes since 2014

CFSSL - Cloudflare's contribution to usable crypto

Written in Go and can be compiled for essentially any platform

Can run as a CA server to sign certificates, generate CRLs, act as OCSP responder

On a client, you can generate keys & CSRs, submit CSR to server, and write out the signed certificate from the response

Many other capabilities, including certificate bundling, site scanning, and PKCS#11/ HSM support

Database Software

PostgreSQL

- High-availability/replication, decent documentation, easy setup
- Already in use extensively at Distil

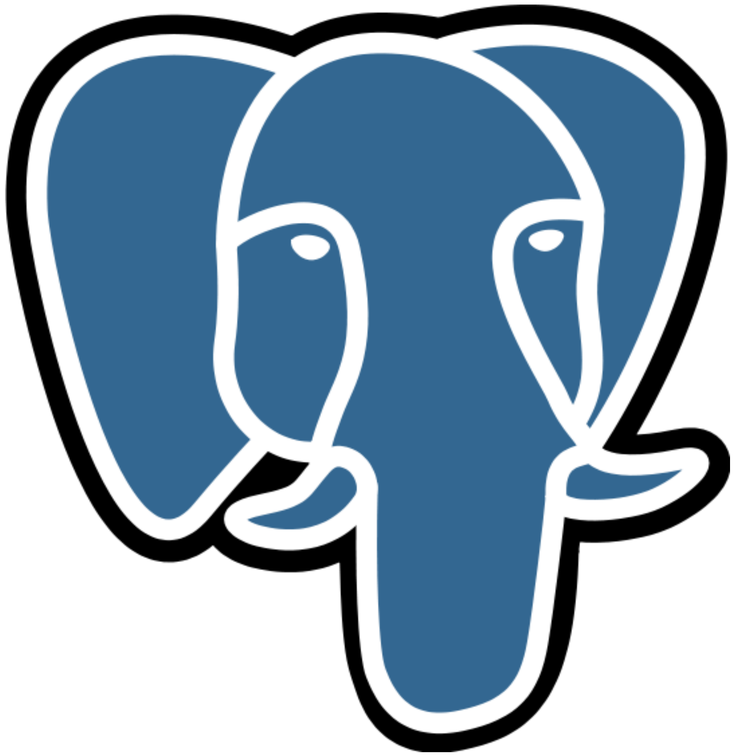
MySQL

- Excellent documentation, HA (but it's complicated/not simple)
- Majority of my previous DB experience is with MySQL

SQLite

- Hey, I didn't say all the options were good… ¯_(ツ)_/¯

PostgreSQL - Because elephants can remember...



Easier (in my opinion) setup and configuration than MySQL

Can set up clustered high availability using streaming replication between the master and slave

The HA slave can also be used as a read-only node

I was going to be using PostgreSQL databases for other projects and wanted to reuse code

Automation Software

Chef

- Write all your code in Ruby,
- Cookbooks & recipes make distribution easy from a central server

Ansible

- Reasonably easy to write for, from what the docs say

Puppet

- Has its own language

Bash/CRON

- It’s bash. Everyone knows bash, right?

Chef - There can never be too many cooks in the kitchen... wait...

At Distil we use Chef for all of our server configuration management

Maintain all cookbooks and recipes in a private git repo

Can create wrapper recipes with includes and dependencies to ensure that baselines are upheld

Chef Vault can be integrated to maintain control over secrets

CHEF™

Developing the Architecture

Goals for the new PKI

Scalability

- Will the PKI be able to support 2x, 5x, or 10x servers?

High Availability

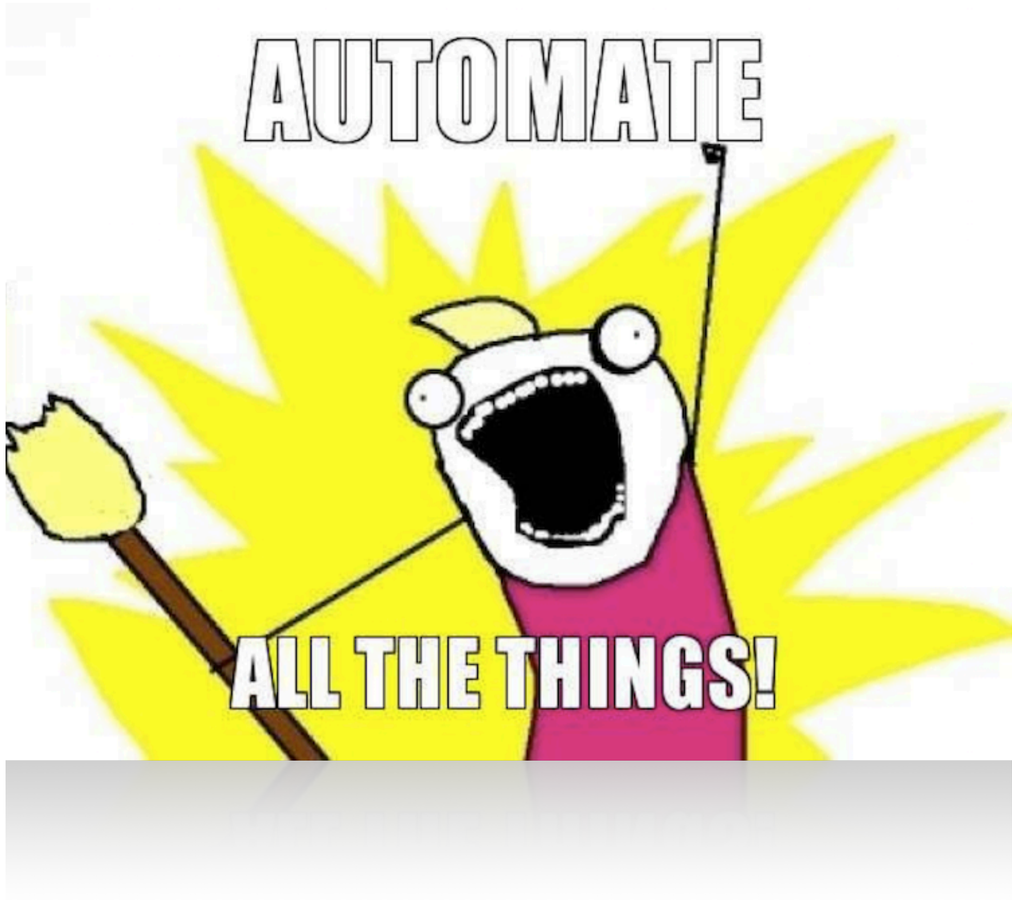
- If a datacenter goes down, will it still work?

Data Redundancy

- If a CA goes down, can I still see the certificates it issued?

Limited manual intervention

- I don't want to be issuing certificates myself



Goals for the new PKI

$0.00

2 months

Root CA

Server is:

- Physical
- Something that provides data redundancy
- Offline & always shutdown
- Stored in a physically separate, secure location from the rest of the infrastructure

Root certificate is valid for 10 years

Intermediate CA

Servers are:

- Part of the existing virtual infrastructure
- Shut down when they aren't in use

Intermediate Certificates are valid for 4 years

Logical way of grouping Issuing CAs

- Type of end-entity (Server, Client, Network Device, etc.)

Issuing CA

Servers are:

- Virtualized
- Always online
- Distributed across data centers for availability

Issuing Certificates are valid for 2 years

Multiple Issuing CAs under each Intermediate CA

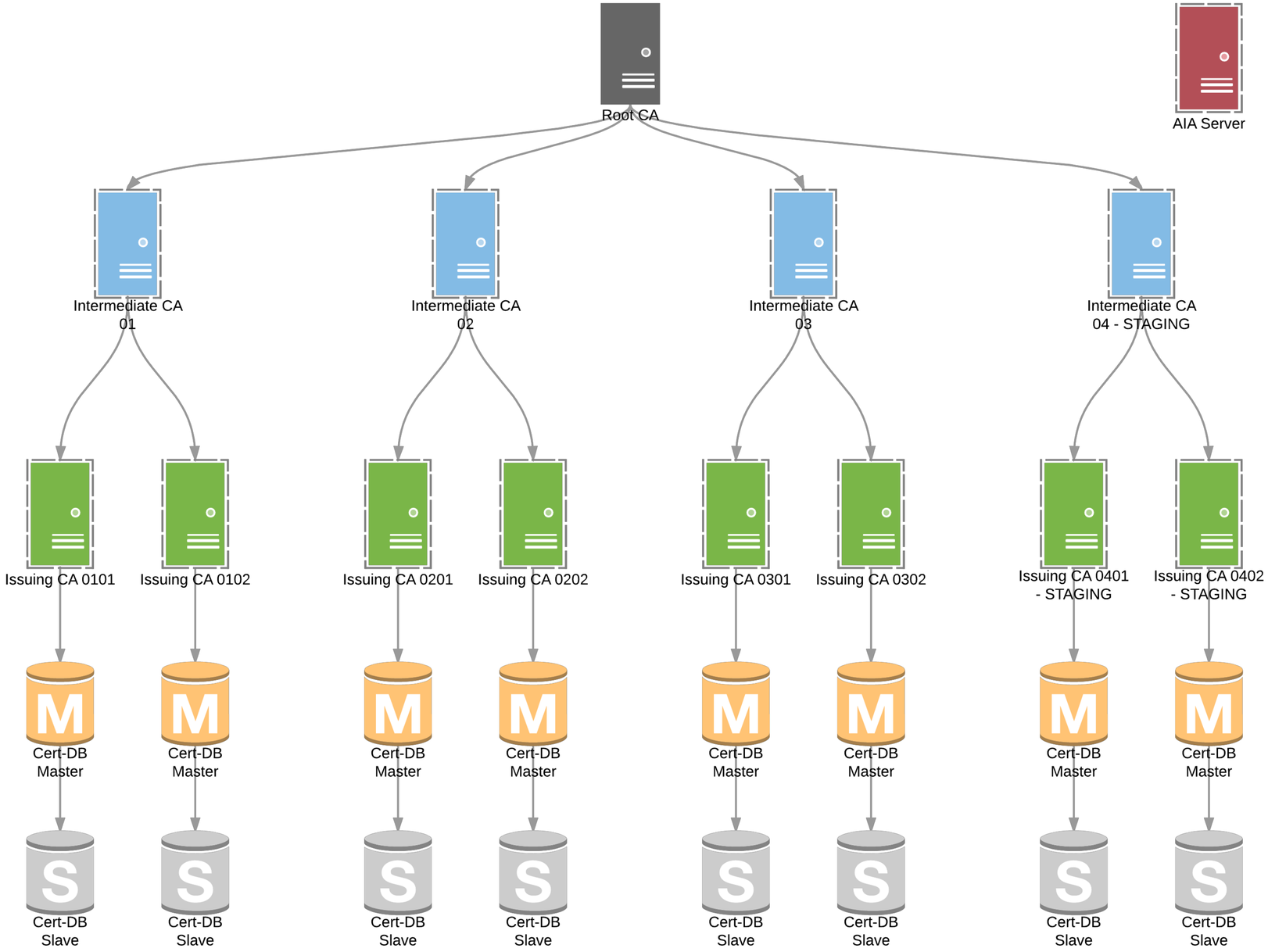
- Provides multiple endpoints for end-entities to reach
- Redundancy/availability in the event of a failure

Issuing CA - cont.

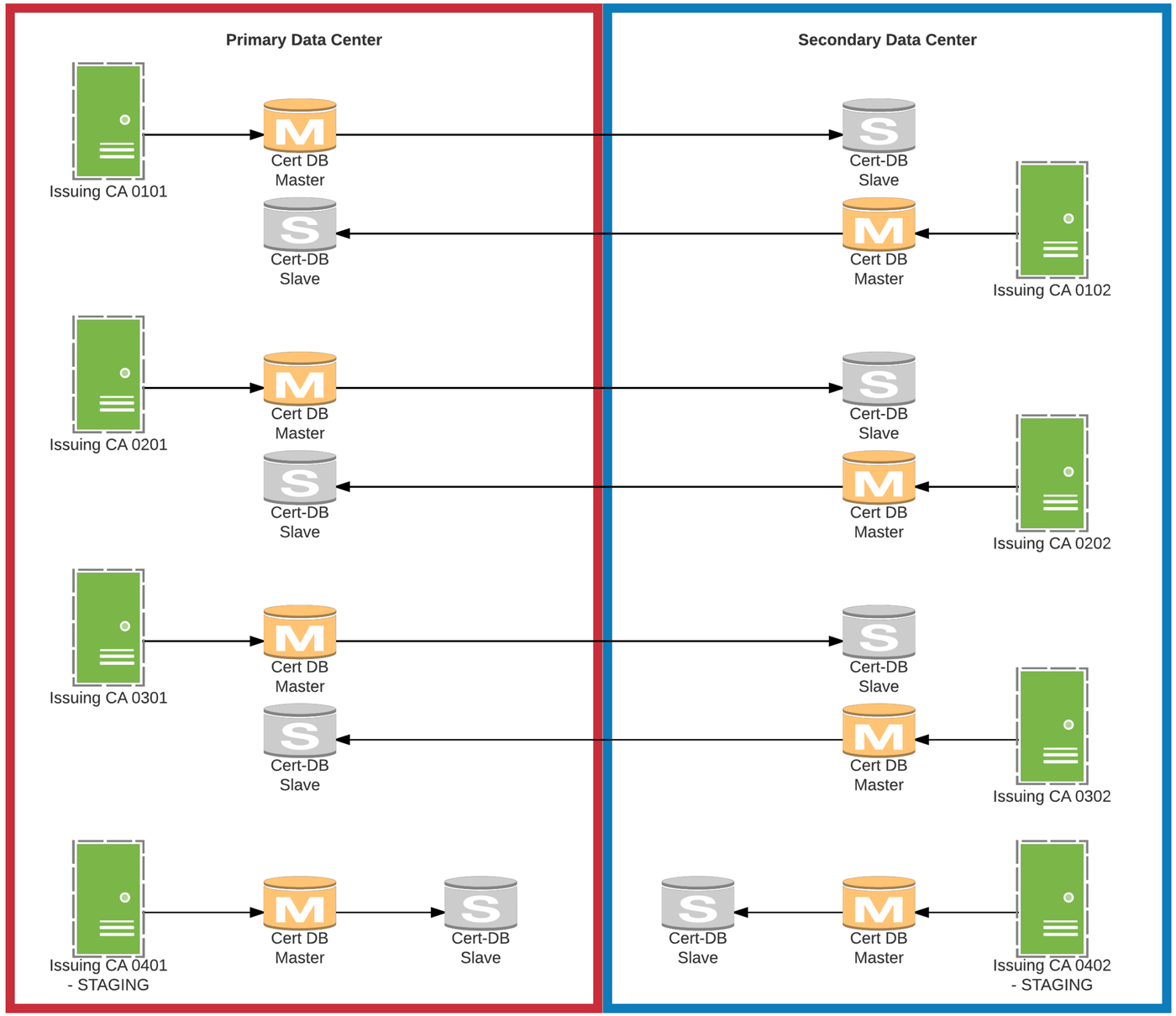
Each Issuing CA has a Certificate Database attached

- Certificate DBs are set up in a master-slave configuration for data redundancy
- Master and Slave database instances should be split across data centers or availability zones

DEVELOPING THE ARCHITECTURE



DEVELOPING THE ARCHITECTURE



Building out the Components

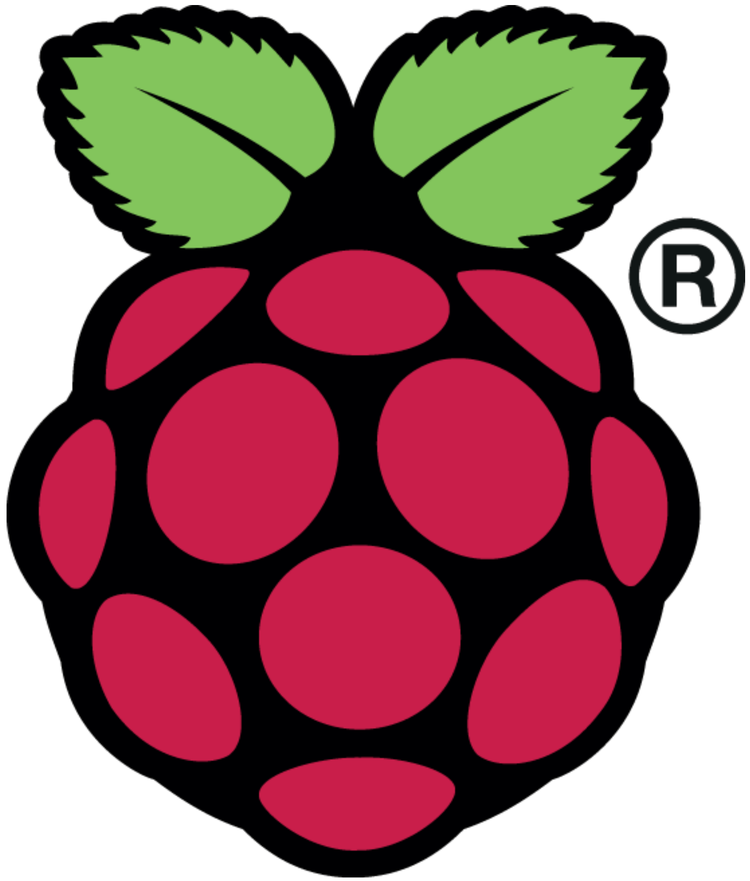
Root CA Hardware & Software

Raspberry Pi 3 Model B

2x 32 GB microSD cards

Hard shell case to protect the Pi

Stored in safe-deposit box/other secure offsite location



Raspbian Jessie with Pixel

OpenSSL (ships with Raspbian)

CFSSL

- Compiled for Linux platform, **ARM architecture**

Using OpenSSL vs CFSSL to create the Trust Anchor

The `cfssl gencert` command has an `-init-ca` flag that is available to generate a self-signed certificate that is supposed to be suitable as a Trust Anchor

When you use that flag to generate a self-signed certificate, CFSSL adds in some EKUs (Extended Key Usages) to the certificate

This is a problem because Mozilla's NSS (used in Firefox) has issues with root certificates that have EKUs in them. Microsoft also can throw similar errors. Generating the Root CA certificate with OpenSSL gets around this issue

Root CA CFSSL Configuration

CFSSL acts as the actual CA software

CFSSL configuration files are all in JSON format

On any given CFSSL Certificate Authority, the configuration file that is used determines the parameters for certificates *issued & signed* by that CA

```
{
  "signing": {
    "default": {
      "expiry": "35040h",
      "usages": [
        "cert sign",
        "crl sign",
        "ocsp signing",
        "server auth",
        "client auth",
        "ipsec user",
        "ipsec tunnel",
        "ipsec end system",
        "digital signature",
        "key encipherment",
        "key agreement"
      ],
      "ca_constraint": {
        "is_ca":true,
        "max_path_len":2
      },
      "issuer_urls": ["http://internal-pki.example.com/certs/root-ca.pem"],
      "crl_url": "http://internal-pki.example.com/crls/root-ca.crl"
    }
  }
}
```

Generating a Private Key & CSR on the Intermediate CAs

```
{
  "CN": "Example Org Internal Intermediate CA 01",
  "key": {
    "algo": "rsa",
    "size": 4096
  },
  "hosts": [
    "intermediate-ca-01.example.com",
    "104.236.166.118"
  ],
  "names": [
     {
       "C": "US",
       "L": "Denver",
       "O": "Example Org, Ltd",
       "OU": "IT",
       "ST": "Colorado"
     }
  ]
}
```

`csr.json` file sets the parameters used to create the CSR and private key on the server

Attributes include:

- Common Name
- Key algorithm and size
- SANs
- Entity names

Configuring CFSSL on Intermediate CAs

`config.json` file sets the parameters for signing the Issuing CA certificates

Sets:

- Issuing CA validity
- Usages (EKUs)
- CA Constraints
- AIA information

```
{
  "signing": {
    "default": {
      "expiry": "17520h",
      "usages": [
        "cert sign",
        "crl sign",
        "ocsp signing",
        "server auth",
        "digital signature",
        "key encipherment",
        "key agreement"
      ],
      "ca_constraint": {
        "is_ca": true,
        "max_path_len":0,
        "max_path_len_zero": true
      },
      "crl_url": "http://internal-pki.examp
      "issuer_urls": [
        "http://internal-pki.example.com/c
      ]
    }
  }
}
```

Issuing CA - Setting up the basics

Private Key and CSR generation is the same as on the Intermediate CAs:

- `csr.json` file sets the parameters used to create the CSR and private key on the server
- `config.json` adds profiles & authentication keys

We add a third JSON file on the issuing CA

- `cert_db.json` contains the information required to connect to the master certificate database

```
{
  "signing": {
    "default": {
      "expiry": "720h",
      "usages": ["digital signature"],
      "is_ca": false
    },
    "profiles": {
      "server": {
        "expiry": "2160h",
        "usages": [
          "server auth",
          "key encipherment",
          "key agreement",
          "digital signature"
        ],
        "auth_key": "server_auth",
        "crl_url": "http://internal-pki.example
        "issuer_urls": ["http://internal-pki.ex
        "is_ca": false
      }
    }
  },
  "auth_keys": {
    "server_auth": {
      "type":"standard",
      "key":"0123456789ABCDEF0123456789ABCDEF"
    }
  }
}
```

Issuing CAs - Making it a service

```
description "Runs the CFSSL web server on port 8888 as a service"
author "Craine Runton - craine@runtondev.com"

start on runlevel [2345]
stop on shutdown

respawn
respawn limit 10 5

console log

chdir /opt/cfssl/example-org

script

    export PATH="/opt/cfssl/bin"
    cfssl serve -address 127.0.0.1 \
    -ca issuing_ca.pem \
    -ca-key issuing_ca-key.pem \
    -config config_issuing_ca.json \
    -db-config cert_db.json

end script

pre-start script
    echo "`date +%Y/%m/%d\ %H:%M:%S` [Info] CFSSL server Starting" >> /var/log/upstart/cfssld.log
end script

pre-stop script
    echo "`date +%Y/%m/%d\ %H:%M:%S` [Critical] CFSSL server Stopping" >> /var/log/upstart/cfssld.log
end script
```

We use Upstart to “service-ify” CFSSL

- Starts the service when an Issuing CA boots
- Restarts the service when it dies
- Logs stdout & stderr to a logfile for troubleshooting

Issuing CAs - Securing the front door

Use Nginx to proxy connections to the service

We add HTTPS to the proxy to ensure that all requests to the CA are encrypted

Also gives us HTTP/S access and error logging

```
19  server {
20    listen 443 ssl default_server;
21    server_name <%= @hostname %>;
22
23    access_log /var/log/nginx/<%= @hostname %>/https/access.log;
24    error_log /var/log/nginx/<%= @hostname %>/https/error.log;
25
26    ssl on;
27    ssl_certificate /opt/cfssl/example-org/<%= @hostname %>/cert_bundle.pem;
28    ssl_certificate_key  /opt/cfssl/example-org/<%= @hostname %>/cert-key.pem;
29    ssl_session_timeout 5m;
30    ssl_protocols TLSv1.2;
31    ssl_prefer_server_ciphers on;
32    ssl_ciphers
      'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AE
      CDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-
      A384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:
      AES256-GCM-SHA384:AES256-SHA256:AES256-SHA';
33    ssl_ecdh_curve secp521r1;
34    ssl_session_cache shared:SSL:10m;
35
36    location / {
37      proxy_pass http://127.0.0.1:8888;
38      proxy_set_header X-Forwarded-Host $server_name;
39      proxy_set_header X-Real-IP $remote_addr;
40      proxy_set_header X-Forwarded-Proto $scheme;
41    }
```

Certificate Databases - Remembering all the little details

Required if you want to use CRLs, OCSP responders, want to revoke issued certificates

Stores records of all the certificates issued by a CA:

- Serial Number
- Authority Key Identifier
- Labels
- Certificate Status
- Expiration Date
- Revocation date & reason
- PEM

```
cfssl=# SELECT serial_number, status, expiry FROM certificates LIMIT 15;
                              serial_number                                    |     status     |        expiry
-------------------------------------------------------------------------------+----------------+-----------------------
 \x35323530383832313537323034313438393031373239393832383930343033383735343030343033373638373633323 31 | \x676f6f64     | 2017-01-16 08:43:00-07
 \x33353835343938313530363634323933363838383933303339343132373038343930323931383638383135373832333 39 | \x676f6f64     | 2017-01-16 08:43:00-07
 \x33393030373334313833303733323737343934333234353038323836373131313734333632383633303530323638363 37 | \x676f6f64     | 2017-01-16 08:43:00-07
```

Authority Information Access - Because what I say goes

Uses Nginx to serve static files over plain HTTP

- Root, Intermediate, and Issuing CA PEMs
- CRLs

Do NOT use HTTPS to server the files

- Can cause lookup loops on systems trying to build the certificate chain

End Entities - You get a certificate! and you get a certificate! Everybody gets a

By default every server gets one end-entity certificate

- Part of the baseline server configuration
- CN is the FQDN and includes the server's primary IP as a SAN
- 90 day validity, new certificate and key every 60 days

Certificates for additional hostnames can be provided by adding a recipe to the Chef run list

All CA certificates for are installed and trusted by the OS

Up to application owners to point their apps to new certificates, restart their applications at least every 60 days

User machines - One Certificate to Rule Them All

JumpCloud for client platform management (macOS, Windows, Linux)

1 script for each platform installs the Root CA certificate and has the OS trust it

- Covers Safari, Chrome, IE, & Edge browsers
- Firefox users must install Root CA cert manually

1 script for each platform can set up client certificates for

- Downloads binaries from build server
- Generates private key & CSR, submits to CA, saves signed certificate to user's cert store

Manual requests - Let's get hands on

Not every request can be automated

- Network devices
- Systems that can't have CFSSL binaries installed

Can run CFSSL locally to generate a key/certificate pair

- Upload/install at needed location

Protecting the Infrastructure

Keep it Secret. Keep it Safe.

Keep it Secret. Keep it Safe.

Encrypt the Private Keys using OpenSSL once all signings are complete

- `openssl rsa -aes256 -in cert-key.pem -out cert-key.encrypted.pem`
- Make it sufficiently long and split it between at least two groups of people
- Delete the unencrypted/original key
- Save the encrypted key in another Vault (Hashicorp, Bitium, etc.) just in case

When you need to sign another CSR, decrypt to `/tmp` and ensure you delete when your done, then always power off the instance/VM

Ensure appropriate file permissions for all unencrypted private keys or use an HSM

Audit trails & whitelists are useful

Audit things like power-on/power-off events in your virtualization environment for Intermediate CAs

Log & audit file access attempts for the encrypted/unencrypted private key files

Add CN & SAN whitelists to your Issuing CA configurations to ensure only certificates with appropriate domain names can be issued

```
22          "name_whitelist":"^([^\\*]+)\\.(distil|distilnetworks)\\.(it|us|com)$"
```

QUESTIONS?

CONTACT INFO

Craine Runton

Twitter: `@craine_runton`

GitLab: `craine`

GitHub: `cmrunton`

Email: `craine@runtonsecurity.com`