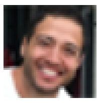

**Fabian Lee : Software Engineer**

Cloud Operations and Development

# Ubuntu: Creating a trusted CA and SAN certificate using OpenSSL

There are numerous articles I've written  where a certificate is a prerequisite for deploying a piece of infrastructure.

This article will guide you through creating a trusted CA (Certificate Authority), and then using that to sign a server certificate that supports SAN (Subject Alternative Name).

Operationally, having your own trusted CA is advantageous over a self-signed certificate because once you install the CA certificate on a set of corporate/development machines, all the server certificates you issue from that CA will be trusted.   If you manage a larger sized internal environment where hosts, services, and containers are in constant flux, this is an operational win.

CA trust also had advantages to self-signed certs because browsers like Chrome 58 and Firefox 48 have limitations on trusting self-signed certificates.   The Windows version of Chrome is the only flavor that allows self-signed certs to be imported as a trusted root authority, all other OS do not trust the self-signed certificate.  And Firefox allows you to add a permanent exception, but needs a trusted CA in order to show a fully green trust lock icon.

If you just want a self-signed SAN certificate with no backing CA, then read my article here instead, but note that it has limitations that are overcome by using a trusted CA.

## Overview

If you are familiar with commercial certificates, you know that a certificate does not live in isolation.  It it just the beginning of a chain of trust, where the root certificate is ultimately trusted because it sits on your local system.

We can create a SAN certificate with the same features, issued and signed by a Certificate Authority that *we create.*   This has several benefits:

1. Better emulation of production – production certs also consist of a chain
2. Ease of administration – once a user trusts our CA, then any other SAN certificate we generate will also be trusted
3. Better browser support – not all browsers allow self-signed certs to be added into the trusted root authorities list

In this article, first we will create our own CA (Certificate Authority).  Then we will use that CA to create a SAN server certificate that covers  "mydomain.com" as well as any of its subdomains, "*.mydomain.com".

If you want to test the certs we generate here, I would recommend using HAProxy.  Here is a page where I describe how to do a quick HAProxy install.

## Prerequisite

As a prerequisite, ensure the SSL packages are installed:

```
$ sudo apt-get install libssl1.0.0 -y
```

### Customized openssl.cnf

The first step is to grab the openssl.cnf template available on your system.  On Ubuntu this can be found at "/usr/lib/ssl/openssl.cnf".  You may find this in "/System/Library/OpenSSL/" on MacOS, and "/etc/pki/tls" on Redhat variants.

```
export prefix="mydomain"

cp /usr/lib/ssl/openssl.cnf $prefix.cnf
```

“$prefix.cnf” needs be modified with the specific information about the cert we are going to generate.

Under the [ v3_ca ] section, add the following values.  For the CA, this signifies we are creating a CA that will be used for key signing.

```
[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = critical, CA:TRUE, pathlen:3
keyUsage = critical, cRLSign, keyCertSign
nsCertType = sslCA, emailCA
```

Then under the “[ v3_req ]” section, set the following along with all the valid alternative names for this certificate.

```
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
#extendedKeyUsage=serverAuth
subjectAltName = @alt_names

[ alt_names ]
DNS.1 =  mydomain.com
DNS.2 =  *.dydomain.com
```

Also uncomment the following line under the “[ req ]” section so that certificate requests are created with v3 extensions.

```
req_extensions = v3_req
```

When we generate each type of key, we specify which extension section we want to use, which is why we can share $prefix.cnf for creating both the CA as well as the SAN certificate.

## Create CA certificate

Now we will start using OpenSSL to create the necessary keys and certificates. First generate the private/public RSA key pair:

```
openssl genrsa -aes256 -out ca.key.pem 2048

chmod 400 ca.key.pem
```

This encodes the key file using an passphrase based on AES256.  Then we need to create the self-signed root CA certificate.

```
openssl req -new -x509 -subj "/CN=myca" -extensions v3_ca -days 3650 -key
ca.key.pem -sha256 -out ca.pem -config $prefix.cnf
```

You can verify this root CA certificate using:

```
openssl x509 -in ca.pem -text -noout
```

This will show the root CA certificate, and the 'Issuer' and 'Subject' will be the same since this is self-signed. This is flagged as "CA:TRUE" meaning it will be recognized as a root CA certificate; meaning browsers and OS will allow it to be imported into their trusted root certificate store.

```
Issuer: CN=myca
...
Subject: CN=myca
...
X509v3 Basic Constraints:
 critical CA:TRUE, pathlen:3
X509v3 Key Usage:
 critical Certificate Sign, CRL Sign
Netscape Cert Type:
 SSL CA, S/MIME CA
```

## Create Server certificate signed by CA

With the root CA now created, we switch over to the server certificate.  First generate the private/public RSA key pair:

```
openssl genrsa -out $prefix.key.pem 2048
```

We didn't put a passphrase on this key simply because the CA is more valuable target and we can always regenerate the server cert, but feel free to take this extra precaution.

Then create the server cert signing request:

```
openssl req -subj "/CN=$prefix"  -extensions v3_req  -sha256 -new -key
$prefix.key.pem -out $prefix.csr
```

Then generate the server certificate using the: server signing request, the CA signing key, and CA cert.

```
openssl x509 -req  -extensions v3_req  -days 3650 -sha256 -in $prefix.csr -CA ca.pem
-CAkey ca.key.pem -CAcreateserial -out $prefix.crt -extfile $prefix.cnf
```

The "$prefix.key.pem" is the server private key and "$prefix.crt" is the server certificate.  Verify the certificate:

```
openssl x509 -in $prefix.crt -text -noout
```

This will show the certificate, and the 'Issuer' will be the CA name, while the Subject is the prefix.  This is not set to be a CA, and the 'Subject Alternative Name' field contains the URLs that will be considered valid by browsers.

```
Issuer:
  CN=myca
...
Subject:
  CN=mydomain
...
X509v3 Basic Constraints:
  CA:FALSE
X509v3 Key Usage:
 Digital Signature, Non Repudiation, Key Encipherment
X509v3 Subject Alternative Name:
 DNS:mydomain.com, DNS:*.mydomain.com
```

## Server deployment

Servers like HAProxy want the full chain of certs along with private key (server certificate+CA cert+server private key). While Windows IIS wants a .pfx file. Here is how you would generate those files.

```
cat $prefix.crt ca.pem $prefix.key.pem > $prefix-ca-full.pem

openssl pkcs12 -export -out $prefix.pfx -inkey $prefix.key.pem -in $prefix.crt -
certfile ca.pem
```

## Browser Evaluation

When you first point Chrome or Firefox at the site with your SAN cert with CA signing, it will throw the same type of exceptions as a self-signed SAN cert. This is because the root CA cert is not known as a trusted source for signed certificates.

In Chrome settings (chrome://settings), search for "certificates" and click on "Manage Certificates". On Windows this will open the Windows certificate manager and you should import the "ca.pem" file at the "Trusted Root Certification Authorities" tab. This is equivalent to adding it through mmc.exe, in the "local user" trusted root store (not the computer level). On Linux, Chrome manages its own certificate store and again you should import "ca.pem" into the "Authorities" tab. This should now make the security icon turn green.

In Firefox Options (about:preferences), search for "certificates" and click "View Certificates". Go to the "Authorities" tab and import "ca.pem". Check the box to have it trust websites, and now the lock icon should turn green when you visit the page.

Although there is a little friction doing this import, it is a one-time cost because any other certificates that you sign with this CA are now trusted.  So if a cert expires and you have to replace it, or you need to change the URLs in a SAN and refresh it, none of the browsers will have an issue with trust.

REFERENCES

https://stackoverflow.com/questions/21488845/how-can-i-generate-a-self-signed-certificate-with-subjectaltname-using-openssl

http://grokify.github.io/security/wildcard-subject-alternative-name-ssl-tls-certificates/

https://gist.github.com/jhamrick/ac0404839b5c7dab24b5 (script for CA and SAN)

http://wiki.cacert.org/FAQ/subjectAltName

https://github.com/stanzgy/wiki/blob/master/network/openssl-self-signed-certs-cheatsheet.md (exact commands for CA, intermediate, chain, server cert, validating cert+key)

https://gist.github.com/akailash/7ec96e39d6951dd2293308e1d8055307

https://gist.github.com/bitoiu/9e19962b991a71165268 (original source of quick SAN with no intermediate)

https://jamielinux.com/docs/openssl-certificate-authority/create-the-root-pair.html (multiple pages of detailed lead through for CA, intermediate, and cert)

https://security.stackexchange.com/questions/38782/ssl-tls-distinction-between-self-signed-cert-and-self-signed-ca-and-other-que (discussion on CA vs Self signed)

https://stackoverflow.com/questions/5244129/use-rsa-private-key-to-generate-public-key (discussion on RSA public/private pair and info inside)

https://stackoverflow.com/questions/5935369/ssl-how-do-common-names-cn-and-subject-alternative-names-san-work-together (explains how RFC 6125 from 2011 says SAN checked first)

https://github.com/webpack/webpack-dev-server/issues/854 (self signed cert no longer valid chrome 58)

https://bugs.chromium.org/p/chromium/issues/detail?id=700595&desc=2 (chrome 58 needs SAN for self-certs)

https://gist.github.com/akailash/7ec96e39d6951dd2293308e1d8055307 (wildcard SAN with CA, also shows how to add as trusted cert at linux level)

https://www.chromium.org/administrators/policy-list-3#EnableCommonNameFallbackForLocalAnchors (temporary workaround for chrome 58)

Reddit discussion on Chrome 58

http://users.skynet.be/pascalbotte/art/server-cert.htm (use Jetty jar to transform pkcs12 to jks java keystore)

NOTES

On Ubuntu, trusted root certificates sit in the directory "/etc/ssl/certs", and can be updated using "sudo update-ca-certificates". On Windows it is managed through the MMC Certificate Snap-In.

If you don’t want to manually type the password, you can use passin/passout:

```
openssl genrsa -des3 -out CA.key -passout file:capass.txt 2048
```

Now use that CA to create the root CA certificate.

```
openssl rsa -in CA.key -passin file:capass.txt -out CA.pem
```

Fabian / February 17, 2018 / DNE, Linux / CA, cert, certificate, chrome, crt, haproxy, java, key, keytool, openssl, pem, pfx, san, self, self-signed, signed, subject alternate name, ubuntu, wildcard