Get your certificate chain right

Sebastiaan van Steenis Follow
Aug 17 · 9 min read

As many know, certificates are not always easy. If you have a self created Certificate Authority and a certificate (self signed), there is not that much that can go wrong. It gets more troublesome when there are one or more intermediate certificates are in the chain. The application serving the certificate has to send the complete chain, this means the server certificate itself and all the intermediates. The CA certificate is supposed to be known by the receiving end (either manually imported because it is self signed or built in because it's from a recognized Certificate Authority)

One of the problems encountered is that the chain sent from the application is incomplete, this usually leads to errors like `x509: certificate signed by unknown authority` or `server certificate verification failed`. Usually certificates are tested using a browser, visiting the URL by going to https://yourwebsite.com and see if it shows as green (or if it's not showing `Not Secure` in the latest version of Google Chrome). Problem using this approach is that browsers tend to complete the chain if it's not sent from the server using their embedded certificate store (or from the operating system). This means that even an incomplete chain will show as valid in the browser. For example, go to https://incomplete-chain.badssl.com and see how the

browser will show it as valid. If you try to connect to the same URL using command line tools, it will fail:

```
$ openssl s_client -connect incomplete-chain.badssl.com:443
-servername incomplete-chain.badssl.com
    Verify return code: 21 (unable to verify the first
certificate)
```

```
$ curl -v https://incomplete-chain.badssl.com
curl: (60) server certificate verification failed. CAfile:
/etc/ssl/certs/ca-certificates.crt CRLfile: none
```

Let's see how we can check the certificates before applying them, so we can know for sure that the certificate chain is complete. I divided the post in two options, one when you are using your own Certificate Authority (usually called self-signed) and one when using certificates from a recognized Certificate Authority (yes, they use intermediates as well).

Make sure you have the required certificate files:

- CA certificate file (usually called `ca.pem` or `cacerts.pem`)
- Intermediate certificate file (if exists, can be more than one. If you don't know if you need an intermediate certificate, run through the steps and find out)
- Server certificate file

For the purpose of this blog post, we will walk through the case with no intermediate certificates and with one intermediate certificate. In the example commands, the following filenames are used:

- Root CA certificate file: `ca.pem`
- Intermediate CA certificate file: `intermediate.pem`
- Server certificate file: `cert.pem`

Validate certificate chain when using your own Certificate Authority

Root CA certificate file and server certificate file (no intermediates)

Let's start validating. Run the following command:

```
$ openssl verify cert.pem
cert.pem: C = Country, ST = State, O = Organization, CN =
FQDN
error 20 at 0 depth lookup:unable to get local issuer
certificate
```

As you can see, the chain cannot be verified. The Root CA certificate is unknown and the chain cannot be validated. If you want to know what CA issued this certificate (`issuer`), you can use the following command:

```
$ openssl x509 -in cert.pem -noout -issuer
issuer= /CN=the name of the CA
```

Now that we know the `issuer`, we can check if the Root CA certificate file we have is the correct one by retrieving the `subject` of the Root CA certificate file. This should match the `issuer` on the server certificate file.

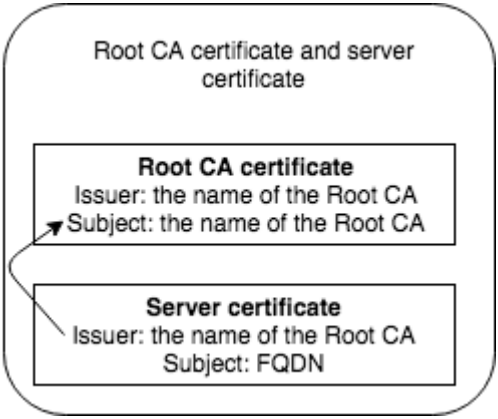
Note: If it is not showing the expected issuer, it might be issued by an intermediate CA. Scroll down to see how to deal with intermediate certificates.

Retrieve the `subject` of the Root CA certificate file using this command:

```
$ openssl x509 -noout -subject -in ca.pem
subject= /CN=the name of the CA
```

Good, this adds up. Now verify the certificate chain by using the Root CA certificate file while validating the server certificate file by passing the `CAfile` parameter:

```
$ openssl verify -CAfile ca.pem cert.pem
cert.pem: OK
```



Issuer should match subject in a correct chain

The past example was on a Root CA certificate and a server certificate, if you still see `error 20 at 0 depth lookup:unable to get local issuer certificate` or the `issuer` and `subject` don't add up, you probably need to include an intermediate certificate.

Root CA certificate file, intermediate CA certificate and server certificate file

Let's start by using the base command to validate a certificate:

```
$ openssl verify cert.pem
cert.pem: C = Countrycode, ST = State, O = Organization, CN
= yourdomain.com
error 20 at 0 depth lookup:unable to get local issuer
certificate
```

This was to be expected, we don't supply anything to validate the certificate chain so it fails. We can discover the issuer of this certificate as follows:

```
$ openssl x509 -in cert.pem -noout -issuer
issuer= /CN=the name of the intermediate CA
```

The issuer shown should match up with the `subject` of our intermediate certificate. Retrieve the `subject` of the intermediate certificate:

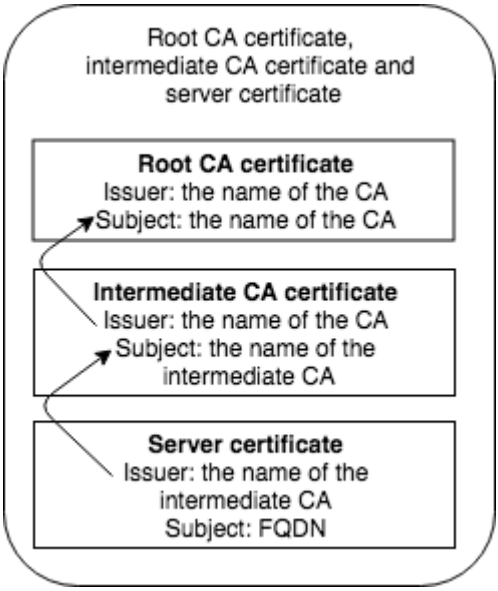
```
$ openssl x509 -in intermediate.pem -noout -subject
subject= /CN=the name of the intermediate CA
```

This should match with the issuer of the certificate. We can do the same validation on the intermediate certificate, as the `issuer` on the intermediate should match the `subject` of the CA certificate.

```
$ openssl x509 -in intermediate.pem -noout -issuer
issuer= /CN=the name of the CA
```

And this should match the `subject` of the CA certificate:

```
$ openssl x509 -in ca.pem -noout -subject
subject= /CN=the name of the CA
```



Issuer should match subject in a correct chain

To complete the validation of the chain, we need to provide the CA certificate file and the intermediate certificate file when validating the server certificate file. We can do that using the parameters `CAfile` (to provide the CA certificate) and `untrusted` (to provide intermediate certificate):

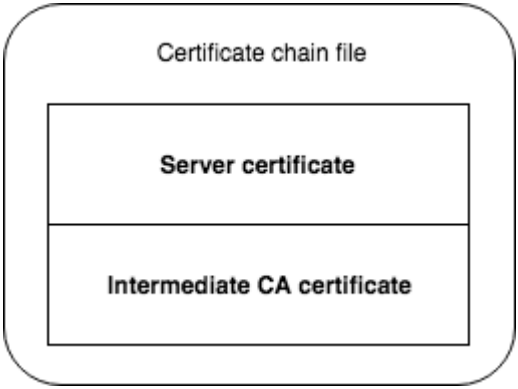
```
$ openssl verify -CAfile ca.pem \
                 -untrusted intermediate.cert.pem \
                 cert.pem
cert.pem: OK
```

Note: If you have multiple intermediate CA certficates, you can use the `untrusted` parameter multiple times like `-untrusted intermediate1.pem -untrusted intermediate2.pem` .

Ordering certificate when using intermediates

If you are using intermediate certificate(s), you will need to make sure that the application using the certificate is sending the complete chain (server certificate and intermediate certificate). This depends on the application you are using that uses the certificate (always check the documentation), but usually you have to create a file containing the server certificate file and the intermediate certificate file. It is required to put the server certificate file first, and then the intermediate certificate file(s). When using the files in our example, we can create the correct file for the chain using the following command:

```
$ cat cert.pem intermediate.pem > chain.pem
```



Server certificate comes first in the chain file, then the intermediates

Always double check if everything went well, we can do so by using this command which will list each certificate in order with the `issuer` and `subject`.

```
$ openssl crl2pkcs7 -nocrl -certfile chain.pem | openssl
pkcs7 -print_certs -noout
subject=/C=Countrycode/ST=State/O=Organization/CN=FQDN
issuer=/C=Countrycode/ST=State/O=Organization/CN=the name of
the intermediate CA

subject=/C=Countrycode/ST=State/O=Organization/CN=the name
of the intermediate CA
issuer=/C=Countrycode/ST=State/O=Organization/CN=the name of
the CA
```

In this output, the order should be:

- `subject` : Server certificate file subject (your FQDN usually)

- `issuer` : Intermediate CA certificate name
- `subject` : Intermediate CA certificate name (this should match with the previous issuer value)
- `issuer` : CA certificate subject

Validate certificate chain when using a recognized Certificate Authority

If you use a certificate from a recognized Certificate Authority, you could think that all certificate chains are taking care off. As mentioned in the beginning of this post, most users will test certificates using the browser which will complete most of the certificate chains if they are not being sent from the server. I will walk through an example using Let's Encrypt certificates.

Let's Encrypt certificates

As this is not a guide for Let's Encrypt, we won't go into detail of getting the certificates. I created a DNS record for FQDN pointing to the host where I will request the certificates using Docker. The command I used to get the certificates is:

```
docker run -p 80:80 -p 443:443 \
           -v /etc/letsencrypt:/etc/letsencrypt \
           certbot/certbot \
           certonly --standalone \
           --agree-tos \
           --reinstall  \
           --force-renewal \
           --non-interactive \
```

```
            --text \
            --rsa-key-size 4096 \
            --email your@email.com \
            --domains "FQDN"
```

This should give you multiple files in `/etc/letsencrypt/live/FQDN` :

```
$ ls /etc/letsencrypt/live/FQDN/
cert.pem  chain.pem  fullchain.pem  privkey.pem  README
```

You can already see that Let's Encrypt provides you with basically all files you need. Let's go over them by validating them, starting with the `openssl verify` command:

```
$ openssl verify /etc/letsencrypt/live/FQDN/cert.pem
/etc/letsencrypt/live/FQDN/cert.pem: CN = FQDN
error 20 at 0 depth lookup:unable to get local issuer
certificate
```

You see that even with a certificate from a recognized Certificate Authority, it still fails to validate the chain. When using self signed certificates, you need to provide the Root CA certificate (and possible intermediates) to validate the chain. When using a recognized Certificate Authority, you usually only need to provide the intermediate CA certificate (the command above will succeed if there is no

intermediate CA). If you are not sure if your Certificate Authority is using intermediates, usually Googling with `your certificate provider intermediates` shows a page describing the so called Chain of Trust. For Let's Encrypt, you can visit their Chain of Trust page.

To complete our chain, let's find out what `issuer` is in the server certificate file:

```
$ openssl x509 -in /etc/letsencrypt/live/FQDN/cert.pem -
noout -issuer
issuer= /C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
```

If you look at the files generated, you can see the file `chain.pem` which indicates containing the intermediate to validate the certificate chain. You can retrieve the `subject` from the `chain.pem` file and see if that matches the `issuer` on the certificate:

```
$ openssl x509 -in /etc/letsencrypt/live/FQDN/chain.pem -
noout -subject
subject= /C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
```

This should all match, and to confirm that the `chain.pem` file completes our chain, we can run the following command:

```
$ openssl verify -untrusted
/etc/letsencrypt/live/FQDN/chain.pem
/etc/letsencrypt/live/FQDN/cert.pem
/etc/letsencrypt/live/FQDN/cert.pem: OK
```

The other file that stands out is `fullchain.pem`, the difference between `chain.pem` and `fullchain.pem` is that `chain.pem` only contains the intermediate certificate. The file `fullchain.pem` contains both your server certificate file and the intermediate (conveniently placed in the correct order). This means that you should always use `fullchain.pem` when configuring a server certificate in an application. The only exception here is if the application uses a dedicated file for providing the chain.

Last but not least, you can show the entire chain by using the following command:

```
$ openssl crl2pkcs7 -nocrl -certfile fullchain.pem | openssl
pkcs7 -print_certs -noout
subject=/CN=FQDN
issuer=/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3

subject=/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
issuer=/O=Digital Signature Trust Co./CN=DST Root CA X3
```

This output also confirms the correct order of certificates in `fullchain.pem`, as it shows the `subject` of server certificate (`FQDN`),

with `issuer` being the intermediate. In the following lines, it shows the `subject` of the intermediate (which should match the `issuer` of the previous section), and the `issuer` being the Root CA certificate.

Completing certificate chains automatically

First and foremost, you should be able to contact the company where you bought your certificate and they should be able to point you to the correct intermediate certificates needed. There are tools out there that can grab the intermediate certificates by using the `CA Issuers` / `IssuingCertificateURL` in the certificate to find the intermediates needed for the chain. Example is https://github.com/zakjan/cert-chain-resolver and the web service based upon it, https://certificatechain.io/. This usually only works for certificates from a recognized Certificate Authority.

Check certificate chain using Docker image

To simplify testing certificate chains, I created https://github.com/superseb/cert-check. This shell script checks most common issues with certificate files provided. You can use it as follows:

Self signed certificate, providing CA certificate

```
docker run -v /mylocation/cert.pem:/certs/cert.pem \
           -v /mylocation/key.pem:/certs/key.pem \
           -v /mylocation/ca.pem:/certs/cacerts.pem \
           superseb/cert-check:latest \
           test.yourdomain.com
```

Certificate signed by recognized Certificate Authority, not providing CA certificate

```
docker run -v /mylocation/cert.pem:/certs/cert.pem \
           -v /mylocation/privkey.pem:/certs/key.pem \
           superseb/cert-check:latest \
           test.yourdomain.com
```

Let's Encrypt certificate without chain, retrieve intermediate certificate(s) and save in /cert-check/cert-check-fullchain.pem

```
docker run -v /yourlocation/cert.pem:/certs/cert.pem \
           -v /yourlocation/privkey.pem:/certs/key.pem \
           -v /yourlocation/cert-check:/cert-check \
           superseb/cert-check \
           test.yourdomain.com \
           resolv
```

Sources/references

https://backreference.org/2010/03/06/check-certificate-chain-file/
https://www.itsfullofstars.de/2016/02/verify-certificate-chain-with-openssl/
https://security.stackexchange.com/questions/56697/determine-if-private-key-belongs-to-certificate
https://github.com/zakjan/cert-chain-resolver
https://stackoverflow.com/questions/20983217

https://gist.github.com/stevenringo/2fe5000d8091f800aee4bb5ed1e800a6
https://serverfault.com/questions/590870/how-to-view-all-ssl-certificates-in-a-bundle