OpenSSL CA on Ubuntu server 20.04

OpenSSL is a free, open-source library that you can use for digital certificates. One of the things you can do is build your own CA (Certificate Authority).

A CA is an entity that signs digital certificates. An example of a well-known CA is Verisign. Many websites on the Internet use certificates for their HTTPS connections that were signed by Verisign.

Besides websites and HTTPS, there are some other applications/services that can use digital certificates. For example:

- VPNs: instead of using a pre-shared key you can use digital certificates for authentication.
- Wireless: WPA 2 enterprise uses digital certificates for client authentication and/or server authentication using PEAP or EAP-TLS.

Instead of paying companies like Verisign for all your digital certificates. It can be useful to build your own CA for some of your applications. In this lesson, you will learn how to create your own CA.

Configuration

In this example, I will use an Ubuntu server. The OpenSSL configuration will be similar as on other distributions like CentOS.

Prerequisites

Before we configure OpenSSL, Set the correct hostname/FQDN correctly and make sure that time, date and timezone are correct.

```
$ hostname
ca
```

The hostname is “ca”. Let's check the FQDN:

```
$ hostname -f
ca
```

It’s also “ca”. Let's change the FQDN; you need to edit the following file for this:

```
$ sudo vim /etc/hosts
#Change the following line:
127.0.1.1       ca
#To:
127.0.1.1       ca.example.local ca
```

Let’s verify the hostname and FQDN again:

```
$ hostname
ca

$ hostname -f
ca.example.local
```

The hostname and FQDN is looking good.

Change timezone

```
$ sudo timedatectl
               Local time: Wed 2022-02-02 09:36:44 UTC
           Universal time: Wed 2022-02-02 09:36:44 UTC
                 RTC time: Wed 2022-02-02 09:36:44
                Time zone: Etc/UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
$ sudo timedatectl set-timezone Europe/Zagreb
$ sudo timedatectl
               Local time: Wed 2022-02-02 10:38:30 CET
           Universal time: Wed 2022-02-02 09:38:30 UTC
                 RTC time: Wed 2022-02-02 09:38:30
                Time zone: Europe/Zagreb (CET, +0100)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
$
```

Time and date could be configured manually, but it is a better idea to use NTP. You can synchronize the time/date with this command:

```
$ sudo vim /etc/systemd/timesyncd.conf
# uncomment and modify line: #NTP=
[Time]
NTP=hr.pool.ntp.org
#save the file
$ sudo timedatectl set-ntp off
$ sudo timedatectl set-ntp on
$ sudo systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
     Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled;
vendor preset: enabled)
     Active: active (running) since Wed 2022-02-02 10:42:04 CET; 15s ago
       Docs: man:systemd-timesyncd.service(8)
   Main PID: 15260 (systemd-timesyn)
     Status: "Initial synchronization to time server 162.159.200.1:123
(hr.pool.ntp.org)."
      Tasks: 2 (limit: 918)
     Memory: 1.4M
     CGroup: /system.slice/systemd-timesyncd.service
```

```
             └─15260 /lib/systemd/systemd-timesyncd

Feb 02 10:42:04 ca systemd[1]: Starting Network Time Synchronization...
Feb 02 10:42:04 ca systemd[1]: Started Network Time Synchronization.
Feb 02 10:42:04 ca systemd-timesyncd[15260]: Initial synchronization to time
server 162.159.200.1:123 (hr.pool.ntp.org).
$
```

OpenSSL Configuration

OpenSSL uses a configuration file that is easy to read. There are a couple of things that we will change in it:

```
# vim /usr/lib/ssl/openssl.cnf
```

Look for the following section:

```
[ CA_default ]

dir        = ./demoCA

#And change it, so it looks like this:

[ CA_default ]

dir            = /root/ca

#Also, find and uncomment:
copy_extensions = copy
#Be careful with the copy_extensions
```

The “/root/ca” folder is where we will store our private keys and certificates.

You might also want to take a look at the default policy:

```
[ policy_match ]
countryName     = match
stateOrProvinceName = match
organizationName    = match
organizationalUnitName  = optional
commonName      = supplied
emailAddress        = optional
```

Some fields like country, state/province, and organization have to match. If you are building your CA for a lab environment like I am then you might want to change some of these values:

```
[ policy_match ]
countryName             = match
stateOrProvinceName     = optional
```

```
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

I've changed it so that only the country name has to match.

Root CA

The first thing we have to do is to create a root CA. This consists of a private key and root certificate. These two items are the "identity" of our CA.

Let's switch to the root user:

```
$ sudo su
```

We will create a new folder which stores all keys and certificates:

```
$ mkdir /root/ca
```

In this new folder we have to create some additional sub-folders:

```
$ cd /root/ca
$ mkdir newcerts certs crl private requests
```

We also require two files. The first one is called "index.txt". This is where OpenSSL keeps track of all signed certificates:

```
$ touch index.txt
```

The second file is called "serial". Each signed certificate will have a serial number. I will start with number 1234:

```
$ echo '1234' > serial
```

All folders and files are in place. Let's generate the root private key:

```
$ openssl genrsa -aes256 -out private/cakey.pem 4096
Generating RSA private key, 4096 bit long modulus
..++
...................++
e is 65537 (0x10001)
Enter pass phrase for private/cakey.pem:
Verifying - Enter pass phrase for private/cakey.pem:
```

The root private key that I generated is 4096 bit and uses AES 256 bit encryption. It is stored in the private folder using the "cakey.pem" filename.

Anyone that has the root private key will be able to create trusted certificates. Keep this file secure!

We can now use the root private key to create the root certificate:

```
# openssl req -new -x509 -key /root/ca/private/cakey.pem -out cacert.pem -
days 3650 -set_serial 0
Enter pass phrase for /root/ca/private/cakey.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:HR
State or Province Name (full name) [Some-State]:Croatia
Locality Name (eg, city) []:Zagreb
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:CA.example.local
Email Address []:admin@example.local
```

The root certificate will be saved as the “cacert.pem” filename and is valid for 10 years. If “cacert.pem” is renamed to “cacert.crt”, you can import it to windows trusted root store

Create a certificate - short and with conf and DNS alt names

config.txt

```
[req]
default_bits = 4096
prompt = no
default_md = sha256
req_extensions = req_ext
distinguished_name = dn

[ dn ]
countryName=HR
stateOrProvinceName=Croatia
localityName=Zagreb
organizationName=Company
emailAddress=person@example.com
commonName=fqdn

[ req_ext ]
```

```
subjectAltName=@alt_names

[ alt_names ]
DNS.1=fqdn1
DNS.2=fqdn2
```

```
// generate key
openssl genrsa -aes256 -out pwd.key 4096
// apache will ask for key password on each service restart, so you can
remove the password
openssl rsa -in pwd.key -out pwdPASSWORDLESS.key
// generate CSR using the config file
openssl req -new -key pwd.key -out pwd.csr -sha256 -config <( cat config.txt
)
// check id DNS entry is present
openssl req -noout -text -in pwd.csr | grep DNS:
// sign the CSR
openssl ca -in pwd.csr -out pwd.pem
// check if DNS entry is present in signed certificate
openssl x509 -noout -text -in pwd.pem | grep DNS:
// Combine the full chain (if you can't specify each file separately)
cat {pwd.pem,cacert.pem,pwdPASSWORDLESS.key} > starCOMBINED.pem
```

Create a certificate

Our root CA is now up and running. Normally when you want to install a certificate on a device (a web server for example), then the device will generate a CSR (Certificate Signing Request). This CSR is created by using the private key of the device.

On our CA, we can then sign the CSR and create a digital certificate for the device.

Another option is that we can do everything on our CA. We can generate a private key, CSR and then sign the certificate…everything “on behalf” of the device.

That’s what I am going to do in this example; it’s a good way to test if your CA is working as expected.

I’ll generate a private key, CSR and certificate for an imaginary “web server”.

Let’s use the requests folder for this:

```
$ cd /root/ca/requests/
```

First, we have to generate a private key:

```
$ openssl genrsa -aes256 -out some_serverkey.pem 2048
Generating RSA private key, 2048 bit long modulus
..............................+++
....+++
```

```
e is 65537 (0x10001)
Enter pass phrase for some_server.pem:
Verifying - Enter pass phrase for some_server.pem:
```

The private key will be 2048 bit and uses AES 256 bit encryption. With the private key, we can create a CSR:

```
root@ca:~/ca/requests# openssl req -new -key some_serverkey.pem -out
some_server.csr
Enter pass phrase for some_serverkey.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:HR
State or Province Name (full name) [Some-State]:Croatia
Locality Name (eg, city) []:Zagreb
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:some_server.example.local
Email Address []:admin@example.local

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Now we can sign the CSR that we just created:

```
$ openssl ca -in some_server.csr -out some_server.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for /root/ca/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4660 (0x1234)
        Validity
            Not Before: Apr  1 09:08:59 2016 GMT
            Not After : Apr  1 09:08:59 2017 GMT
        Subject:
            countryName               = HR
            stateOrProvinceName       = Croatia
            organizationName          = Example
            commonName                = some_server.example.local
            emailAddress              = admin@example.local
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

```
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                57:A7:7A:41:3E:3F:B3:EE:0D:CF:46:D0:A7:A5:9B:46:92:D1:F0:AD
            X509v3 Authority Key Identifier:
keyid:1B:38:B6:9F:82:46:72:5A:04:07:76:C2:DA:A5:5D:EB:95:83:81:30

Certificate is to be certified until Apr  1 09:08:59 2017 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

That's all there is to it. The "some_server.pem" file is the signed digital certificate for our web server. If you want you can delete the CSR, move the private key to the "private" folder, and move the new certificate to the "certs" folder:

```
$ rm some_server.csr
$ mv some_serverkey.pem /root/ca/private/
$ mv some_server.pem /root/ca/certs/
```

The "some_server.pem" certificate can now be installed on your web server.

Security

Protecting your CA is important. Anyone that has access to the private key of the CA will be able to create trusted certificates.

One of the things you should do is reducing the permissions on the entire /root/ca folder so that only our root user can access it:

```
$ chmod -R 600 /root/ca
```

In this example, we used the root CA to sign the certificate of an imaginary web server directly. This is fine for a lab environment but for a production network, you should use an intermediate CA.

The intermediate CA is another server that signs certificates on behalf of the root CA.

The root CA signs the certificate of the intermediate CA. You can then take the root CA offline which reduces the chance of anyone getting their hands on your root private key.

Verification

We created some private keys and generated some certificates. Let's take a closer look at some of our work.

Here's the index.txt file:

```
$ cat /root/ca/index.txt
V       170401090859Z           1234    unknown
/C=HR/ST=Croatia/O=Example/CN=some_server.example.local/emailAddress=admin@e
xample.local
```

Above you can see the certificate that we created for our web server. It also shows the serial number that I stored in the serial file. The next certificate that we sign will get another number:

```
$ cat /root/ca/serial
1235
```

Let's take a closer look at the certificates. We can verify them with OpenSSL, but it might be nice to see them on your computer. I'll use a Windows computer for this.

Windows doesn't recognize the .PEM file extension so you might want to rename your certificates to .CRT.

Here's the root certificate: You can see the name of our root CA and the validity (10 years). If we want to trust certificates that are signed by our root CA, then we'll have to install this certificate. Here's how:

1. Hit the Install Certificate button and you will see this wizard:
    1. It's up to you if you want to install it for your current user or the entire computer. Click Next to continue:
2. Make sure you select the Trusted Root Certification Authorities store and click Next and Finish:
3. Windows will give you one more big security warning, click Yes to continue:
4. The root certificate is now installed and trusted.

Now open the certificate that we assigned to "some server"; You can see that it was issued by our root CA, it's valid for one year. When you look at the certification path then you can see that Windows trusts the certificate. If a web server would present this certificate to your computer, then it will trust it from now on.

If you use a service that doesn't accept separate certificate files (like HaProxy; unlike Apache), you have to create a full chain certificate file (cat everything into a single file: site cert, ca cert, site key).

Conclusion

You have now learned how to build your own CA using OpenSSL and are ready to sign certificates for your servers, routers, firewalls, clients or any other devices that you have.

I hope you enjoyed this lesson. If you have any questions feel free to ask in our forum.