**Exercise 11**

*Setting up SSL security for some of our existing services*

**Prior Knowledge**

Previous exercises

**Objectives**

Understanding SSL certificates

Understanding TLS configuration

How to use Docker Compose to install certificates securely

**Software Requirements**

(see separate document for installation of these)

* **TBD**

**Overview**

*TBD*

**Steps**

1. Check that openssl is installed on your Ubuntu server:  
   sudo apt-get install openssl
2. The first step is that we are going to create a certificate authority (CA). This is because we don’t want to have to go through the rigmarole of paying a real CA.   
     
   *Hint: if you are willing to host a real server on a real fully-qualified DNS name, then you can get a free certificate from the EFF via certbot. However, we aren’t able to do that today.  
     
   Hint 2: If you really do want to create a CA, do not follow these instructions. They are far too insecure. You should read widely, but this is a good starting place:* [*https://jamielinux.com/docs/openssl-certificate-authority/*](https://jamielinux.com/docs/openssl-certificate-authority/)
3. Make some directories:  
   mkdir -p ~/sec/ca/private  
   mkdir -p ~/sec/server/keys/private  
   mkdir -p ~/sec/client/keys/private
4. We are going to act as several different roles in this lab. The first role is going to be the **CA Administrator**.  
     
   Let’s make a private key for the CA:  
   cd ~/sec/ca  
   openssl genrsa -aes256 -out private/ca.key.pem 4096  
   Generating RSA private key, 4096 bit long modulus

..........................................................................................++

..................................................................++

e is 65537 (0x10001)

Enter pass phrase for private/ca.key.pem:

Enter a password. Probably best to use something insecure like “password” since this is not for real.   
  
Verifying - Enter pass phrase for private/ca.key.pem:

Re-enter the password.

We put the key into the private directory so we can keep track of which parts need security and which don’t.

1. We now need a certificate for the CA.   
   openssl req -key private/ca.key.pem -new -x509 -days 8000 -sha256 -out ca.cert.pem  
   *(All on one line)*  
     
   First enter your password.  
   Now enter the following **bold items**. Hit enter for the others.

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]:**UK**

State or Province Name (full name) [Some-State]:**Oxfordshire**

Locality Name (eg, city) []:**Oxford**

Organization Name (eg, company) [Internet Widgits Pty Ltd]:**Comlab CA**

Organizational Unit Name (eg, section) []:

Common Name (e.g. server FQDN or YOUR name) []:

Email Address []:

This *ca*.*cert*.*pem* file doesn’t need securing. In fact we want to share this certificate as widely as possible.

1. That is our CA created. We can now “switch hats” and be the **Server administrator**.  
   cd ~/sec/server/keys
2. The server needs a private key. This doesn’t need to be a secure as the CA key (lasts a year instead of 20 years!) so we can use 2048 bits.  
     
   openssl genrsa -aes256 -out private/server.key.pem 2048  
   I propose you use “password” again.

Generating RSA private key, 2048 bit long modulus

.................+++

..............................................................................................................................................................................+++

e is 65537 (0x10001)

Enter pass phrase for private/server.key.pem:

Verifying - Enter pass phrase for private/server.key.pem:

1. No-one will trust this key because it hasn’t been signed. In order to create trust we need to get a CA to sign this key. Luckily we know a friendly CA (ourselves). To ask the CA to sign the key, we create a Certificate Signing Request (csr).   
     
   openssl req -key private/server.key.pem -new -sha256 -out server.csr.pem

*Again all on one line*   
Now use the following **bold** entries. The only really important one is the FQDN (**localhost**) since this will be checked against the DNS name of the server.

Enter pass phrase for private/server.key.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]:**UK**

State or Province Name (full name) [Some-State]:**Oxfordshire**

Locality Name (eg, city) []:**Oxford**

Organization Name (eg, company) [Internet Widgits Pty Ltd]:**Localhost Website**

Organizational Unit Name (eg, section) []:

Common Name (e.g. server FQDN or YOUR name) []:**localhost**

Email Address []:

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:

1. We now need to “send” that CSR to the CA:  
   cp server.csr.pem ~/sec/ca/
2. Now we need to switch back to being the CA:  
   cd ~/sec/ca
3. We now need to sign the CSR (as the CA)

openssl x509 -req -days 365 -in server.csr.pem -CAkey private/ca.key.pem -CA ca.cert.pem -out server.cert.pem   
-CAcreateserial  
*(All on one line)*

Signature ok

subject=/C=UK/ST=Oxfordshire/L=Oxford/O=Localhost Website/CN=localhost

Getting CA Private Key

Enter pass phrase for private/ca.key.pem:

1. Now “send” the certificate back to the Server Admin:  
   cp server.cert.pem ~/sec/server/keys
2. The server also needs the CA’s certificate:  
   cp ca.cert.pem ~/sec/server/keys
3. Switch back to being a Server Administrator:  
   cd ~/sec/server
4. We are now ready to use our keys and certificates in an application.   
   There is an updated version of our random app:  
   This should all be fairly obvious.

var https = require('https'),

fs = require('fs'),

express = require('express'),

app = express();

app.get("/",function(req,res){

obj = {random : Math.floor((Math.random() \* 100) + 1)};

res.json(obj);

});

var secureServer = https.createServer({

key: fs.readFileSync('./keys/private/server.key.pem'),

cert: fs.readFileSync('./keys/server.cert.pem'),

ca: fs.readFileSync('./keys/ca.cert.pem'),

requestCert: true,

passphrase: "password",

ciphers: "TLSv1.2",

rejectUnauthorized: false

}, app).listen('8443', function() {

console.log("Secure Express server listening on port 8443");

});

1. Download this code.  
   cd ~/sec/server  
   curl -L [http://freo.me/sec-rand -o server.js](http://freo.me/sec-rand%20-o%20server.js)  
     
   Your directory should look like:

> tree

.

├── ca

│   ├── ca.cert.pem

│   ├── ca.srl

│   ├── private

│   │   └── ca.key.pem

│   ├── server.cert.pem

│   ├── server.crt

│   └── server.csr.pem

├── client

│   └── keys

│   └── private

└── server

├── keys

│   ├── ca.cert.pem

│   ├── private

│   │   └── server.key.pem

│   ├── server.cert.pem

│   └── server.csr.pem

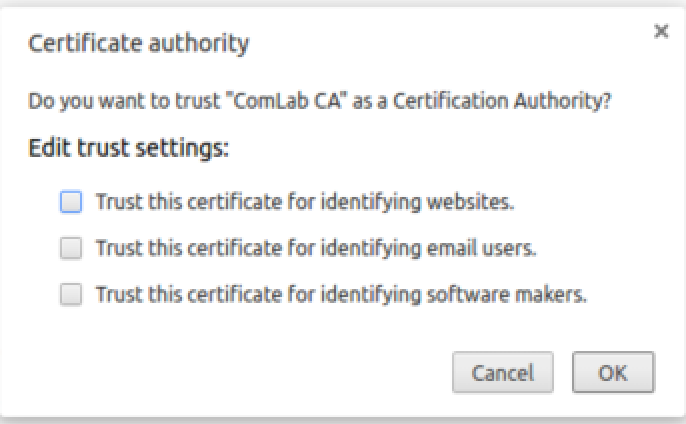
└── server.js

8 directories, 11 files

1. Make sure express is installed:  
   npm install express
2. Run it:  
   node server.js
3. Browse to <https://localhost:8443>

You should get a security error. Do not accept it but instead add the certificate. In Chromium:  
**Edit -> Preferences.  
Show advanced settings**

**HTTPS/SSL -> Manage Certificates  
Import**Browse to **ca.cert.pem**

1. You should see:
2. Click Trust this certificate for identifying websites.
3. Click Finished and then close the settings.
4. Now try browsing again. You should have a lovely green padlock next to the server.  
     
   *Client*
5. Now we’d like to get our client working with this encryption.