Lab 1

Digital Certificates

Contents

1.1 Objective

The objective is twofold. Firstly, get familiar with `openssl` which is a tool to manage digital certificates in PKI, with the format of digital certificates and the information contained in them. Secondly, learn how to configure an Apache web server in order to use secure HTTP connections with Secure Sockets Layer and Transport Layer Security (SSL/TLS) protocols with only server authentication or both client/server authentication.

As you may already know, Apache is an HTTP open source server for Unix and Windows operating systems. Apache uses *mod_ssl* as a module to support the SSL v2/v3 and TLS v1. It also uses the openssl tool to support the cryptographic functionalities.

1.2 Setting up the VM and the Apache server

For this session, we will use the Ubuntu Virtual Machine (VM) image "Ubuntu64-18LTSv1.zip" available in the FIB software repository: https://softdocencia.fib.upc.edu/software. It is a VM, so then you just need to configure an empty VM in either VMWare player or Virtual Box with the "I will install the OS later" option checked. Then, in the configuration option, you can load the VM image as a virtual CD drive, start the VM, and select the "run without installation" starting option. Username and password are **alumne** and **sistemes**, respectively.

Once the VM is set up, you need to update it with

```
# sudo apt update
```

The next step is the installation of the Firefox browser

```
# sudo apt install firefox
```

Openssl is a set of utilities to issue and manage asymmetric key pairs, digital certificates in X.509v3 format, CRL, etc. [1]. The Ubuntu version installed in this VM image has a version already installed at: */usr/bin/openssl*. Note: in case you cannot find it there, you can use the command *whereis openssl* to find it.

As commented previously, the other software we are going to use is Apache, which is not included in this VM image. Therefore, the first step is to install Apache2

```
# sudo apt install apache2
```

Apache server will be installed at */usr/sbin/apache2*. To configure apache2, we will use the files *ports.conf* at */etc/apache2* and the VirtualHost configuration files that should be at */etc/apache2/sites-availabe*. The first file contains the ports that the server uses to listen for incoming connections. The second file is used to configure the behaviour of the website. In particular for our case, it will include the options that the server uses for opening SSL connections and find its private key, its certificate, the trusted CAs, etc.

To start apache2 and edit some of the configuration files we need root access. Enable the root user:

```
# sudo passwd root
```

enter password **toor**, and switch to the root user (su).

Finally, we need to create a directory structure to facilitate the management of the data:

ssl.key : directory to store the cryptographic keys.

ssl.csr : directory to store the digital certificate requests.

ssl.crt : directory to store the digital certificates.

[1] http://www.openssl.org/docs/apps/openssl.html

1.3 Scheme

Figure 1.1 shows an outline of this practice. In particular, we can see the different entities involved and their relationships as well as the values of the attributes that need to be setup.

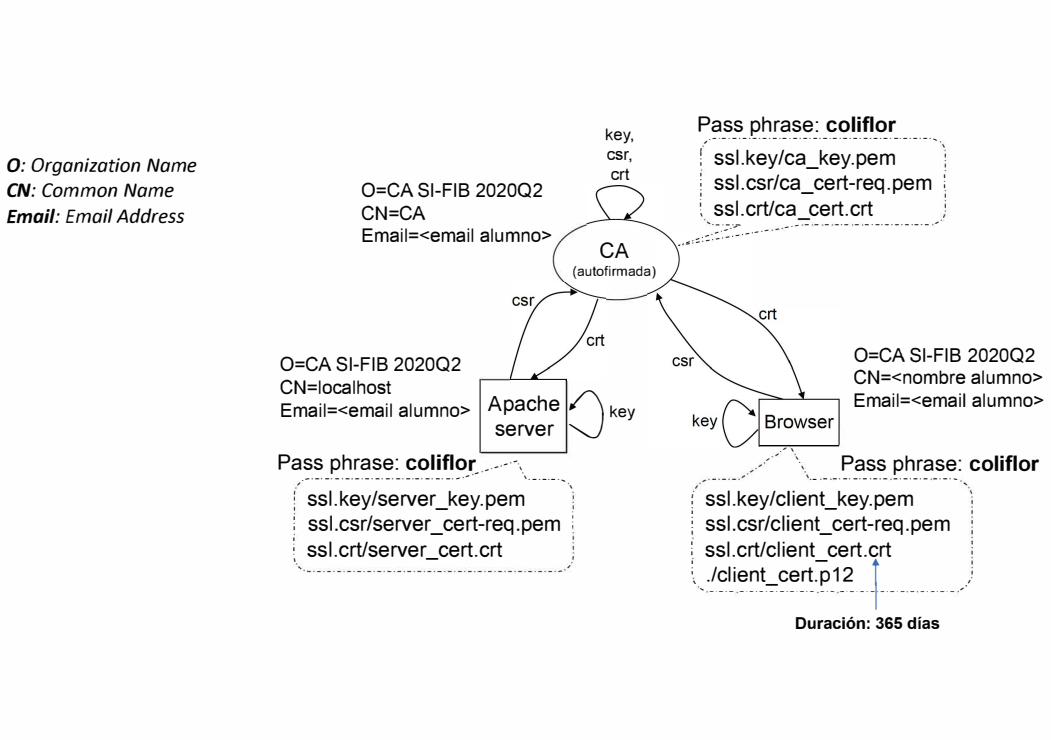

Figure 1.1: Configuration of the different entities.

1.4 Creating the certification hierarchy

The first step consists of creating our certification hierarchy with only one Certification Authority (CA). That means creating the root certificate of the CA.

Warning: we recommend to write down each and every password you use for each certificate. As a suggestion to facilitate this task, you can use the same passphrase and password **coliflor** in every place. This is NOT a good practice and, in real scenarios, it is absolutely NOT recommended. In this practice, we suggest this only to avoid any possible problem related with our memory (or lack of it).

1.4.1 Generating the certificate request for the CA

This step consists of generating a key pair. The private key will be stored in a file protected by a password (*ca_key.pem*), whilst the public key (together with

the information we want the certificate to contain) will be stored in the certificate request file (*ca_ cert-req.pem*).

```
# openssl req -new -extensions v3_ca -keyout
ssl.key/ca_key.pem -out ssl.csr/ca_cert-req.pem
```

For the generation of these files, you will be requested to indicate a passphrase (twice for security) and the different attributes of the certificate, e.g., Country, State, Organization Name, Common Name, etc. As commented above, we suggest to use the pass phrase **coliflor**. Regarding the attributes, we are going to use only the values indicated in Fig. 1.1 for the CA, i.e., Organization Name, Common Name and the email address. The remaining attributes will not be specified: when requested, you need to enter a dot ".", which indicates an empty attribute.

You can check the contents of the two files: one containing the private key and the other containing the certificate request.

The next step is to parse the certificate request using the *asn1parse* utility from openssl:

```
# openssl asn1parse -i -dump -in ssl.csr/ca_cert-req.pem >
ssl.csr/ca_cert-req.txt
```

You can check the new text file and identify the fields that contain the information given when the request was generated and the public key.

1.4.2 Issue the signature for the CA certificate

This step consists of generating the self signed certificate of the CA

```
# openssl req -new -x509 -in ssl.csr/ca_cert-req.pem -out
ssl.crt/ca_cert.crt -days 365 -key ssl.key/ca_key.pem
```

You need to use the password set at the previous step.

You can parse the new certificate and identify the fields containing the information introduced and the public key using

```
# openssl asn1parse -i -dump -in ssl.crt/ca_cert.crt >
ssl.crt/ca_cert.txt
```

Once the CA is set up, we can move on to the server configuration.

1.4.3 Generating the certificate request for the server

Issue the certificate request for the server:

```
# openssl req -new -extensions v3_req -keyout
ssl.key/server_key.pem -out ssl.csr/server_cert-req.pem
```

For this certificate, you have to use the option *-extensions* v3_req and not v3_ca used before for the CA. Openssl provides a set of certificate profiles described in its configuration file *openssl.cnf*. In this case, we use the profile for an entity that is not a CA.

Warning:

- When issuing the new request be careful with the file name not to rewrite the previous requests;
- Use the attribute values indicated in Fig. 1.1 for the web server.

Once the request is issued, you can check if it is well formed using the command:

```
# openssl req -in ssl.csr/server_cert-req.pem -text -verify
```

1.4.4 Issue the signature of the server certificate

This certificate is clearly not self signed anymore: it will be signed by the CA. Therefore, the following command generates the server certificate signed by the CA with a duration of 1 year. The CA will sign this certificate with its pass phrase.

```
# openssl x509 -req -in ssl.csr/server_cert-req.pem -out
ssl.crt/server_cert.crt -days 365 -CA ssl.crt/ca_cert.crt
-CAkey ssl.key/ca_key.pem -CAcreateserial
```

You can parse the new certificate to check that the information is correct:

```
# openssl asn1parse -i -dump -in ssl.crt/server_cert.crt
> ssl.crt/server_cert.txt
```

We can verify that the certificate is well formed using the command:

```
# openssl x509 -in ssl.crt/server_cert.crt -text
```

1.4.5 Generating a certificate for the user

In this step, we will generate the certificate for the user (indicated as client) to be correctly authenticated by the web server. The process is similar to what we already did for the web server certificate.

```
# openssl req -new -extensions v3_req -keyout
ssl.key/client_key.pem -out ssl.csr/client_cert-req.pem
```

Once the request is issued, you can check if it is well formed using the command:

```
# openssl req -in ssl.csr/client_cert-req.pem -text -verify
```

1.4.6 Issue the signature of the user certificate

As we already did for the server, we now need the signature of the CA for the client certificate.

```
# openssl x509 -req -in ssl.csr/client_cert-req.pem -out
ssl.crt/client_cert.crt -days 365 -CA ssl.crt/ca_cert.crt
-CAkey ssl.key/ca_key.pem -CAcreateserial
```

You can also parse the certificate to check the different fields and verify their correctness as you already did for the server (specific commands are omitted here).

1.4.7 Export the user certificate and its private key

This step consists of exporting the user certificate so then we can import it afterwards in the browser. The next command creates a single file that follows the PKCS#12 and will be protected by a password.

```
# openssl pkcs12 -export -in ssl.crt/client_cert.crt
-inkey ssl.key/client_key.pem -out client_cert.p12
-name "clientCert"
```

Warning! Keep this '.p12' file safe and do not forget the password because it will be necessary for the browser client configuration. In addition, if you created this file with root credentials, you need to change its reading permission (*chmod*) so then Firefox can read and import the file.

1.4.8 Install the certificates in the browser

Firstly we need to import the CA certificate in the Firefox browser and afterwards the user certificate.

Open the Firefox browser. Open the menu with the 3-horizontal lines icon on the right hand side on and select Preferences. Go to Privacy & Security, scroll down until Certificates and click on View Certificates. You can see the list of trusted CAs in the Authorities tab. You need to scroll down and find the Import button. Here you need to import the CA certificate: find the *ca_ cert.crt* file and select Open. You can examine the CA certificate (and double check if the attributes are still correct). At the question "Do you want to trust CA for the following purposes?", select both options and click OK. You can check if our CA has been imported. If so, click OK.

Now we can import the user certificate. Click View Certificates again. Go to Your Certificates, scroll down and click Import. Find the file *client_ cert.p12* and select Open. The file's password will be required. You can open the certificate and double check if the attributes are correct.

1.5 Apache Configuration

1.5.1 Start/Stop/Restart the apache web server

```
sudo service apache2 start
```

You will need to have root access to do it. Then start the web browser and connect to localhost (or *http://127.0.0.1*). If you see a presentation page, it means that apache is correctly installed.

To stop the web server

```
sudo service apache2 stop
```

You can also restart the webserver

```
sudo service apache2 restart
```

Warning! Every time you change the configuration of any file you need to restart the web server.

1.5.2 Apache configuration to authenticate the server

We want the server to offer its certificate to the browser, then we need to enable SSL in apache2.

Edit the contents of *ports.conf* (in */etc/apache2*) for apache to listen to HTTPS requests: usually such requests are addressed to ports 443, 4443, etc. Verify that the file already contains:

```
Listen 80
Listen 443
```

You can use the following script to enable the ssl module in apache

```
sudo a2enmod ssl
```

Now restart apache to load the module.

1.5.3 Configure a VirtualHost to use SSL

Check the default file at */etc/apache2/sites-available* directory to become familiar with the structure of such files

1. Open the file *default-ssl.conf*
2. The *DocumentRoot* variable indicates where is the directory tree containing the web pages for this VirtualHost [2]. You can change the home page contents to make sure that you are connecting to the SSL configured VirtualHost, i.e., change from the default site */var/www/html* to */var/www-ssl*.

[2] The `<VirtualHost>` label defines a mask which is used to identify which requests must be addressed to each VirtualHost. Using '*' we include all requests. Note: depending on the Apache version, it may be not necessary to specify '*' (the default in the current version of Apache is already '*').

3. Create the directoty */var/www-ssl* and a new file *index.html* with the following content

```
<html>
 <body><h1>SSI</h1>
  <hr>
  <h2>Server with SSL active</h2><ul>
 </body>
</html>
```

4. Verify that it contains *SSLEngine on* and add just afterwards the option *SSLProtocol -all +TLSv1.1 +TLSv1.2*.

5. Configure the certificate variables in order to point to the server and CA certificate.

```
SSLCertificateFile /home/alumne/si/ssl.crt/server_cert.crt
SSLCertificateKeyFile /home/alumne/si/ssl.key/server_key.pem
SSLCACertificateFile /home/alumne/si/ssl.crt/ca_cert.crt
```

   Warning: Check that */home/alumne/si/* is your absolute path where the ssl.crt and ssl.key directories reside.

6. Apache is performing a control access so then we need to enable the access to its web pages.

```
<Directory "/var/www-ssl">
  AllowOverride All
  Require all granted
</Directory>
```

1.5.4 Enable the new website

The next step enables this new website in the server

```
#a2ensite default-ssl
```

You now need to restart apache2 to reload all changes. If you do not have any error, a request for a password to unlock the server's private key will be prompted.

You can now try to connect to the new website *https://localhost* using a secure connection (note that it is **https**). If the configuration is correct, you should be able to see the new webpage. You can click on the padlock icon just on the left hand side of the url, and check that the connection is secure. You can click on the right arrow to know the connection details and verify the name of the entity that certifies the connection. Clicking on More information you can see the details of the certificate.

Warning! if you see a warning saying that the name of the server is not the one on the certificate you can change the file *default-ssl* to correct the name of the server at the *ServerName* variable.

1.5.5 Client authentication using a digital certificate

We will now configure the web server to require a digital certificate from the user willing to visualise a subdirectory called private.

1. Create the new subdirectory in /var/www-ssl/private and a new file *index.html* with the following content

```
<html>
 <body><h1>SSI</h1>
  <hr>
  <h2>Private: Server with SSL client auth active</h2><ul>
 </body>
</html>
```

2. Open the file *default-ssl.conf*, create a new *<Directory>* directive in the VirtualHost with the following content

```
<Directory "/var/www-ssl/private">
  AllowOverride All
  Require all granted
  SSLVerifyClient require
  SSLVerifyDepth 1
</Directory>
```

   These options mean that the Server requires a certificate from the user, and the maximum length of the certification chain is 1 (i.e., the CA directly signs the user certificate).

3. Restart the server again and try to establish a secure connection to the root directory and to the 'private' directory. See that the server requests a client certificate for the 'private' directory.

You can now try to open the url *https://localhost/private* in Firefox. A User Identification Request should pop up with the details of the user certificate. If you click OK, you will see that the server accepts the user and shows the "private" webpage.