INGI2347: Certificates with HTTPS

Xiao Chen, Marco Canini

May 2, 2015

The learning objective of this lab is for students to get familiar with how to install your own Web server supporting HTTPS, and to get you more familiar with Public Key Infrastructure. After finishing the lab, in addition to gaining a deeper understanding of the concepts, students should be able to use tools to create certificates and setup a Web server that authenticates with the created certificates.
NOTE: This lab has to be done individually. However, please team up in advance with a colleague because a part of the lab will be done in pairs.

1 Setting up the environment

Typically, a webserver is run by root. Since some of you use machines from the lab (on which you are not root), this tutorial will cover the installation of a webserver from a regular user's perspective. For the sake of simplicity, we will use a lightweight and easily-configurable webserver, `lighttpd`. If you work on your laptop and/or if you are more comfortable with another webserver, feel free to install the webserver as root on your laptop, and/or use any alternative.

Start by downloading `lighttpd` into your home directory. Open a terminal and type in the following commands.

```
wget http://download.lighttpd.net/lighttpd/releases-1.4.x/lighttpd-1.4.39.tar.gz
tar -xf lighttpd-1.4.39.tar.gz
```

Next, install the software in your home directory by doing:

```
cd lighttpd-1.4.39/
./configure --prefix=$HOME/lighttpd --with-openssl \
--with-openssl-libs=/usr/lib64/openssl --without-pcre --without-bzip2
make
make install
```

You might encounter such errors, install the necessary software respectively:

```
configure: error: C compiler cannot create executables ...
--> sudo apt-get install libc6-dev.
configure: error: configure: error: pcre-config not found ...
---> sudo apt-get install libpcre3-dev
configure: error: zlib-headers and/or libs where not found ...
---> sudo apt-get install zlib1g-dev
configure: error: bzip2-headers and/or libs where not found ...
---> sudo apt-get install libbz2-dev
```

Make sure it works as expected by typing:

```
cd
lighttpd/sbin/lighttpd -v
```

You should see the line "`lighttpd/1.4.39 (ssl) - a light and fast webserver`". Make sure the "`(ssl)`" bit is there, too. If there is not, check your openssl directory. You need to change /usr/lib64/openssl to wherever the openssl library is. Probably, the library is located in "/usr/lib/openssl".

You can also install it by:

```
# Upto Fedora 21 #
# yum install lighttpd lighttpd-fastcgi

# Fedora 22 & later #
# dnf install lighttpd lighttpd-fastcgi

# For openSUSE & suse #
# zypper install lighttpd

# For Ubuntu, Debian & Mint #
$ apt-get install lighttpd
```

2 Getting the webserver running

Create a sample HTML document, for example by doing:

```
cd
mkdir html/
echo '<html>Hello, world!</html>' > html/index.html
```

Then, create the configuration file for `lighttpd` (we will name it `server.conf`) in your home directory, and fill it with:

```
server.document-root = "/path/to/your/home/html/"
server.port = 64080
index-file.names = ("index.html")
mimetype.assign = (".html" => "text/html")
```

Notice that we set the server port to 64080. Normally, HTTP runs on port 80 but ports from 0 to 1023 are restricted to root. Moreover, only a few ports are open to students on machines of the Intel room.

Finally, run your server by doing:

```
cd
lighttpd/sbin/lighttpd -D -f server.conf
```

Open your favorite browser and go to `http://localhost:64080/`. You should be able to see your greeting message. Verify that you can access your buddy's webpage as well by replacing `localhost` with her/his IP or hostname.

3 Self-signed certificate and first HTTPS test

Start by generating a self-signed certificate. That is, we will create a key for the server and then we will create a certificate that is signed with the **same** key.

Be sure to understand what each of the following commands does.

Let's start with creating a new key:

```
openssl genrsa 2048 > server.key
```

Using the req command[1], let's create a certificate signing request (CSR).

This command will ask you to enter several information that relate to the subject identity for the certificate we are trying to create. You can enter '.' to leave the field blank, if the field is optional.

The information requested includes the following:

- Common Name: this is the fully qualified domain name of the Web server. For example, if you intend to secure https://www.example.com, the the Common Name must be www.example.com.
- Email Address: the email of the person creating the CSR.

[1] `https://www.openssl.org/docs/manmaster/apps/req.html`

- Organization Name: typically the legal name of your organization.
- Organizational Unit Name: typically a division of the organization.
- Country Name: a 2-letter code of the organization's country.
- State or Province Name: the organization's state.
- Locality Name: the organization's city.

WARNING: Leave the challenge password blank (press enter).

Think about what Common Name (CN) you should put in order to avoid errors.

```
openssl req -new -sha256 -key server.key -out server.csr
```

Now, let's proceed to create the actual certificate, `server.crt` file, by signing the CSR with the server key. We will use the x590 command[2] of OpenSSL.

```
openssl x509 -req -in server.csr -signkey server.key -out server.crt
```

The output of this command might look something like this:

```
Signature ok
subject=/C=BE/O=My Org/CN=www.my.org
Getting Private key
```

Finally, we can bundle the certificate together with the (private) server key into a single Privacy Enhanced Mail (PEM) file. This is a standard data format that the web server can use.

```
cat server.crt server.key > server.pem
```

When you think you have a correct `server.pem` file, it is time to enable SSL on your Web server.

Let's append the following lines to `server.conf`:

```
ssl.engine = "enable"
ssl.pemfile = "/path/to/server.pem"
```

Also, change the server port to `server.port = 64443`. Remember to restart your server whenever you make changes to the configuration files. If it seems changes have not taken effect, try emptying the cache of the testing browser.

Visit the page of your friend (in `https`).

Is everything alright? Observe the traffic with **wireshark** if possible, and identify the certificate.

4 A certificate signed by a CA

Instead of signing your own CSR, send it to a CA (represented for the sake of the exercise by `xiao.chen@uclouvain.be`) to receive your certificate. Please name your CSR explicitly by following the naming convention: **firstname.surname.crt** (where firstname.surname matches your UCL email minus the part "@student.uclouvain.be").

What is also necessary to complete the chain of trust?

Try it out on your server and the one of your friend. Make sure no error messages remain.

Be sure that your info is following exactly the same format:

```
Country Name (2 letter code) [AU]:BE
State or Province Name (full name) [Some-State]:Wallonie
Locality Name (eg, city) []:Louvain-la-Neuve
Organization Name (eg, company) [Internet Widgits Pty Ltd]:uclouvain
Organizational Unit Name (eg, section) []:ingi
Common Name (e.g. server FQDN or YOUR name) []:xiao
Email Address []:xiao.chen@uclouvain.be
```

[2] `https://www.openssl.org/docs/manmaster/apps/x509.html`

5 Client authentication

You will now create your own Certification Authority. Use the following commands:

```
openssl genrsa 2048 > ca.key
openssl req -new -key ca.key -out ca.csr
openssl x509 -req -in ca.csr -signkey ca.key -out ca.crt
```

What does each of them do ? For the purpose of signing CSR's as a CA, the default configuration file of SSL suggests using the following file structure. Make sure you create the following files and directories (`man ca`):

```
./demoCA                        - main CA directory
./demoCA/cacert.pem             - CA certificate
./demoCA/private/cakey.pem      - CA private key
./demoCA/serial                 - CA serial number file
./demoCA/index.txt              - CA text database file
./demoCA/newcerts/              - where new certificates will be placed
```

The file `index.txt` is empty, but the file `serial` should contain an integer, say `01`. Once this has been set up, you are ready to sign CSR's.

Import this authority into your browser, as well as the certificate of your friend's CA . Then, generate a client CSR by using:

```
openssl genrsa 1024 > client.key
openssl req -new -key client.key -out client.csr
```

Sign your own CSR as well as your friend's using the CA you created above:

```
openssl ca -in client.csr -out client.crt
```

Note: in some SSL installations, the default configuration file does not use `demoCA`. You may copy the default configuration file (usually `/etc/.../openssl.cnf`) locally, modify the `dir` option in the local copy to `demoCA` and finally use the `-config` option to specify the new configuration file.

For your own certificate, use the following command to export it to a format browsers use (and import it to your browser):

```
openssl pkcs12 -export -out client.p12 -in client.crt -inkey client.key
```

Finally, in order to enable client authentication, add the following to `server.conf`:

```
ssl.verify-peer = "enable"
ssl.verify-depth = 1
```

Browse your website and your colleague's website and make sure everything works properly.