cs3235:11: Lucky last laboratory for the final week, 2019

This may be done individually, or in pairs (ie - a group of 2).

1 Lab11, part 1: Creating a certificate authority

There is a tool installed on your machine that allows you to perform all kinds of cryptographic tasks. It is called `openssl` and you can find out about its large variety of functions by typing "`man openssl`". Before beginning this lab, it may be useful to create a directory heirarchy in which to store things:

```
cd
mkdir -p LAB11/demoCA/private
mkdir LAB11/demoCA/newcerts
cd LAB11
```

Use openssl to create a private (RSA) key, and a public key signed by your own private key. This is called a *self-signed* certificate. When you run the following command, it will ask you for a password that will be used to encrypt the private key on disk. You should remember this password, as you will be asked for it later.

```
openssl req -new -x509 -keyout cakey.pem -out cacert.pem -days 365
```

Look at the private key and the certificate, using `cat` or an editor. Move the files into your directory heirarchy:

```
mv cacert.pem demoCA/
mv cakey.pem demoCA/private/
```

Creating public and private key pairs

Imagine now that you are a company running a webserver that will use SSL to protect it's web-served data (using https). Create a public and private key pair for your company. You can do so by:

```
openssl req -nodes -new -x509 -keyout coPrivKey.pem -out coReq.pem -days 365
```

This creates a private key in `coPrivKey.pem` and an unsigned public key "certificate" in `coReq.pem`. You will now use the CA to validate the public key. To do this you have to ask `openssl` to create a certificate request for you, asking the CA to validate your key. You can do so by:

```
openssl x509 -x509toreq -in coReq.pem -signkey coPrivKey.pem -out coCertReq.pem
```

Look at the `coCertReq.pem` file. To have your CA sign certificate requests, set up `openssl` to work with your CA structure by creating a file `openssl.cnf`. This file is something like this:

```
[ ca ]
default_ca              = CAdefault
[ CAdefault ]
dir                     = ./demoCA
database                = $dir/index.txt
new_certs_dir           = $dir/newcerts
certificate             = $dir/cacert.pem
serial                  = $dir/serial
private_key             = $dir/private/cakey.pem
RANDFILE                = $dir/private/.rand
default_days            = 365
default_crl_days        = 30
default_md              = sha1
policy                  = policy_any
[ policy_any]
countryName             = supplied
stateOrProvinceName     = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

You must also create an empty file `index.txt` at the beginning in the directory specified in the config file (`$dir`). In addition, you need to create a file called `serial` containing "`00`". Openssl will use this as a starting serial number.

You can now have your CA sign this certificate. When you are asked for the "`commonName`", enter the name `localhost`. Look at the file created...

```
openssl ca -config openssl.cnf -out coCertSigned.pem -infiles coCertReq.pem
```

Testing your CA

You can now test your CA! First, add your new CA to the list of trusted CAs in your webbrowser. You can see how to do this at:

https://wiki.wmtransfer.com/projects/webmoney/wiki/Installing_root_certificate_in_Mozilla_Firefox

Openssl itself comes with a very primitive test webserver that we will use as a test SSL webserver. Create a small `index.html` file for testing purposes that should be displayed when people connect via SSL. Run the test webserver the first time only to see information exchanged between the webserver and the client. You can start the SSL webserver on port 4443 by typing:

```
openssl s_server -accept 4443 -www -state -cert coCertSigned.pem -key coPrivKey.pem
```

Use `firefox` to connect to `https://localhost:4443`. Look at the information given on the server side, as well as on the client side. Now run the same command but with "`-WWW`" in place of "`-www`" in the directory in which you created your `index.html` file.

Connect to `https://localhost:4443/index.html`. Look to see the details of your encrypted connection.

Finally - modify/extend your certificates so that someone from an adjacent computer can access your web server (using `https://192.168.100.XXX:4443/index.html`).

Once you have finished the lab, you can access the cs3235 grading website, and answer these three questions:

1. What is the purpose of the extra information in the (CA) certificate you created in at the beginning?
2. What is the purpose of the two files you created: **coPrivKey.pem** and **coReq.pem**. (What is each file used for?)
3. Why is the webserver using both the certificate as well as the private key? (What does it use each for?)

Here is the grading site:

`https://hugh.comp.nus.edu.sg/cs3235/lab11/gradeslab11-1.php`

2 Lab 11, part 2: SEED Return-to-libc Attack lab

Have a look at this paper

`http://www.infosecwriters.com/Papers/C0ntex_return-to-libc.pdf`

It provides some useful background to the SEED Return-to-libc Attack lab, which you are going to do, found at

`http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Software/Return_to_Libc/`

There is a lot in this laboratory, but I would like you to just try task 1, a simple return to libc attack. Once you have finished the lab, you can upload your exploit.c code to the IVLE upload folder for Lab11-2. Please name your file e0XXXXXXX.c (your login id followed by .c).

Access the cs3235 grading website, and describe how you decide the values for X, Y and Z. Either show us your reasoning, or if you use trial-and-error approach, show your trials. After your attack is successful, change the file name of retlib to a different name, making sure that the length of the file names are different. For example, you can change it to newretlib. Repeat the attack (without changing the content of badfile). Is your attack successful or not? If it does not succeed, explain why.

`https://hugh.comp.nus.edu.sg/cs3235/lab11/gradeslab11-2.php`

3 Lab 11, part 3: (Optional/Ungraded) SEED DNS attack lab

Have a look at the SEED DNS attack lab, found at

`http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Networking/DNS_Local/`

There is a lot in this laboratory, but you could just try task 5, spoofing a DNS response. Once you have finished the lab, you can access the cs3235 grading website, and enter in a brief description of the steps to follow to recreate your attack:

`https://hugh.comp.nus.edu.sg/cs3235/lab11/gradeslab11-3.php`