Project 2: Https and Certificate

Due date: Oct. 31

Preparation

1. Read L10 slides
2. Understand public key cryptosystem, signature, certificate, SSL/TLS protocol

Submission

1. Write a report as a Word/PDF file, use filename "**hw2_lastname_studentID**"
2. Submit your report in Blackboard/Homework/Assignment2

Program Requirement

1. (3 points) Follow the instruction below, and successfully setup a https connection with your local web server. Show a screen shot of your web browser that connects to the server. (Demo will be provided in the class)
2. (7 points) Answer the following questions in the report.
    a. Check the certificate file ca.crt and answer the following questions
        i. What cryptographic algorithm was used to generate the certificate? (1 point)
        ii. What's the valid duration of the certificate? (1 point)
    b. Check the certificate file server.crt and answer the following questions
        i. What cryptographic algorithm was used to generate the certificate? (1 point)
        ii. What's the valid duration of the certificate? (1 point)
        iii. What's the serial number of the certificate? Does it match the number you put in the file 'serial'? Check the file 'serial' again, and describe your observations. (1 point)
    c. At the machine, try https://cs642.com:4433 and https://localhost:4433, to access the same page. Does the certificate work properly in both cases? Why? (1 point)
    d. Summarize all the files you have generated in this task and describe the purpose of each of them. (1 point)

Ubuntu Instructions (Demo will be shown in the class). Use Virtual Box, and Seed
https://www.virtualbox.org/

https://seedsecuritylabs.org/

1. Becoming a Certificate Authority (CA)

A Certificate Authority (CA) is a trusted entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. A number of commercial CAs are treated as root CAs, e.g., VeriSign. Users who want to get digital certificates issued by the commercial CAs need to pay those CAs. In this lab, we need to create digital certificates, but we are not going to pay any commercial CA. We will become a root CA ourselves, and then use this CA to issue certificate for others (e.g., web servers). Unlike other certificates, which are usually signed by another CA, the root CA's certificates are self-signed. Root CA's certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on Public Key Infrastructure (PKI). Root CA's certificates are unconditionally trusted.

In order to use OpenSSL to create certificates, you have to have a configuration file which will be used by three OpenSSL commands: **ca**, **req** and **x509**. You can get a copy of the configuration file from /usr/lib/ssl/openssl.cnf. After copying this file into your current directory, you need to create several sub-directories and files as specified in the configuration file (look at the [CA default] section):

```
dir = ./demoCA                  # Where everything is kept
certs = $dir/certs              # Where the issued certs are kept
crl_dir = $dir/crl              # Where the issued crl are kept
new_certs_dir = $dir/newcerts        # default place for new certs.
database = $dir/index.txt            # database index file.
serial = $dir/serial         # The current serial number
```

For the index.txt file, simply create an empty file. For the serial file, put a single number in string format (e.g. 1000) in the file. Now you can create and issue certificates.

As we described before, we need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

```
$openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The outputs of the command are stored in two files: **ca.key** and **ca.crt**. The file **ca.key** contains the CA's private key, while **ca.crt** contains the public-key certificate.

2. **Create a certificate for cs642.com**

Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called cs642.com. For this company to get a digital certificate from a CA, it needs to go through three steps.

Step 1: Generate public/private key pair. The company needs to first create its own public/private key pair. We can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys. The keys will be stored in the file server.key:

```
$openssl genrsa -des3 -out server.key 1024
```

Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it should generate a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). Please use cs642.com as the common name of the certificate request.

```
$openssl req -new -key server.key -out server.csr -config openssl.cnf
```

Step 3: Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, we will use our own trusted CA to generate certificates:

```
$openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config
openssl.cnf
```

3. **Use certificates for web sites**

In this lab, we will explore how public-key certificates are used by web sites to secure web browsing. Let us use cs642.com as our domain name. To get our computers recognize this domain name, let us add the following entry to /etc/hosts; this entry basically maps the domain name cs642.com to our localhost (i.e., 127.0.0.1): **127.0.0.1 cs642.com**

Next, let us launch a simple web server with the certificate generated in the previous task. OpenSSL allows us to start a simple web server using the s server command:

$cp server.key server.pem
$cat server.crt >> server.pem # Combine the secret key and certificate into one file
$openssl s_server -cert server.pem –www # Launch the web server using server.pem

By default, the server will listen on port 4433. You can alter that using the -accept option. Now, you can access the server using the following URL: https://cs642.com:4433/. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following:
“cs642.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown”.

Had this certificate been assigned by VeriSign, we will not have such an error message, because VeriSign’s certificate is very likely preloaded into Firefox’s certificate repository already. Unfortunately, the certificate of cs642.com is signed by our own CA (i.e., using **ca.crt**), and this CA is not recognized by Firefox. We can manually add our CA’s certificate to the Firefox browser by clicking the following menu sequence:

Edit -> Preference -> Advanced -> View Certificates.

You will see a list of certificates that are already accepted by Firefox. From here, we can “import” our own certificate. Please import ca.crt, and select the following option: “Trust this CA to identify web sites”. You will see that our CA’s certificate is now in Firefox’s list of the accepted certificates. Now, point the browser to https://cs642.com:4433 and you should be able to access the web page without the error message.