Cryptography - SSL - X509 Certificate

Lab Overview

The main objective for this lab is to gain first-hand experience on applications of cryptography in cyber security and to get familiar with the concepts in the Public-Key encryption and Public-Key Infrastructure (PKI).

Introduction to SSL

The following is simplified view of how SSL is implemented and what part the certificate plays in the entire process.

Normal web traffic is sent unencrypted over the Internet. That is, anyone with access to the right tools can snoop all that traffic. This can lead to problems, especially where security and privacy is necessary, such as in credit card data and bank transactions. The Secure Socket Layer is used to encrypt the data stream between the web server and the web client (the browser).

SSL makes use of asymmetric cryptography, commonly referred to as public key cryptography (PKI). With PKI, two keys are generated, one public, one private. Anything encrypted with either key can only be decrypted with its corresponding key. Thus, if a message or data stream were encrypted with the server's private key, it can be decrypted only using its corresponding public key, ensuring that the data only could have come from the server.

If SSL utilizes public key cryptography to encrypt the data stream traveling over the Internet, why is a certificate necessary? The technical answer to that question is that a certificate is not really necessary - the data is secure and cannot easily be decrypted by a third party. However, certificates do serve a crucial role in the communication process. The certificate, signed by a trusted Certificate Authority (CA), ensures that the certificate holder is really who he claims to be, and his public key is the one included in the certificate. Without a trusted signed certificate, the data may be encrypted, however, the party you are communicating with may not be whom you think. This is guaranteed through the message digest in the certificate. Without certificates, impersonation attacks would be much more common.

You can either buy an SSL (X.509) certificate or generate your own (a self-signed certificate) for testing or, depending on the application, even in a production environment. If you self-sign your certificates you may save money. Bad news is if you self-sign your certificates nobody but you and your company (perhaps) may trust them. In today's lab we generate a self-signed certificate and we implement SSL on a website.

OpenSSL

OpenSSL is an open source project that provides a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library. More information is available at openssl

Lab Tasks

Task 1: Become 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; VeriSign is the largest CA at the time of writing. 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. servers). In this task, we will make ourselves a root CA, and generate a certificate for this CA.

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 PKI. Root CA's certificates are unconditionally trusted.

Step 1: The Configuration File

In order to use OpenSSL to create certificates, you must have a configuration file. The configuration file usually has an extension "**.cnf**". It is used by three OpenSSL commands: **ca, req** & **x509**.

Before getting started, let's create a directory on our Desktop to store the files for the CA and move into the folder. Use:

```
mkdir ~/CA
cd CA
```

The config file of `openssl.cnf` can be copied using the following command:

```
cp /usr/lib/ssl/openssl.cnf ./
```

(the "**./**" refers to the current directory)

As we want our copy of the **openssl.cnf** file to run independently of the one in the system, we need to change/modify the file. To do so, use any editor you like and change the highlighted **dir** path as shown:

```
dir             =./             # Name of your local directory.
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.
```

As specified in the configuration file, we also need some files and folders to store information. To create them use:

```
mkdir crl certs newcerts
touch index.txt
echo '01' > serial
```

Important: Make sure to change the **dir** value to the path of your **openssl.cnf** file to '**./**'.

Step 2: Certificate Authority (CA)

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., use the following details and keep in mind they are case-sensitive.

- Country Name (2 letter code) [AU]: **AU**
- State or Province Name (full name) [Some-State]: **NSW**
- Locality Name (eg, city) []: **SYD**
- Organization Name (eg, company) [Internet Widgits Pty Ltd]: **UTS**
- Organizational Unit Name (eg, section) []: **FEIT**
- Common Name (e.g. server FQDN or YOUR name) []: **cybersec.com.au**
- Email Address []: **root@cybersec.com.au**

The output of the command is 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.

The **ca.key** is an encoded text file (also encrypted), so you will not be able to see the actual content, such as the modulus, private exponents, etc. To see those, you can run the following command:

```
openssl rsa -in ca.key -text
```

Task 3: Create a Certificate for cybersec.com.au

Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called **cybersec.com.au**. 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 encrypt the private key (using the AES-128 encryption algorithm, as is specified in the command option). The keys will be stored in the file **server.key**:

```
openssl genrsa -aes128 -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), which basically includes the company's public key. 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 identity). Please use **cybersec.com.au** as the common name of the certificate request.

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

You will be once again prompted for information and a password, use the following details and keep in mind they are case-sensitive and should be same as the ones used before.

- Country Name (2 letter code) [AU]: **AU**
- State or Province Name (full name) [Some-State]: **NSW**
- Locality Name (eg, city) []: **SYD**
- Organization Name (eg, company) [Internet Widgits Pty Ltd]: **UTS**
- Organizational Unit Name (eg, section) []: **FEIT**
- Common Name (e.g. server FQDN or YOUR name) []: **cybersec.com.au**
- Email Address []: **root@cybersec.com.au**
- Please enter the following 'extra' attributes
- to be sent with your certificate request
- A challenge password []: **< LEAVE BLANK >**
- An optional company name []: **< LEAVE BLANK >**

It should be noted that the above command is quite similar to the one we used in creating the self-signed certificate for the CA. The only difference is the **-x509** option. Without it, the command generates a request; with it, the command generates a self-signed certificate.

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. The following command turns the certificate signing request (**server.csr**) into an X509 certificate (**server.crt**), using the CA's **ca.crt** and **ca.key**:

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

Task 4: Use PKI for Web Sites

In this lab, we will explore how public-key certificates are used by web sites to secure web browsing. First, we need to get our domain name. Let us use **cybersec.com.au** 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 **cybersec.com.au** to our localhost (i.e., 127.0.0.1):

127.0.0.1 cybersec.com.au

Once the hosts file has been updated, restart the system for the settings to take effect.

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:

Combine the secret key and certificate into one file

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

Launch the web server using server.pem

```
openssl s_server -cert server.pem -www
```

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://cybersec.com.au:4433/**

Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following:

"The owner of cybersec.com.au has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.".

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 **cybersec.com.au** is signed by our own CA (i.e., using **ca.crt**), and this CA is not recognized by Firefox. Therefor we will do the following to include it.

Loading ca.crt into Firefox:

We can manually add our CA's certificate to the Firefox browser by clicking the following menu sequence:

Menu -> Preference -> Advanced -> Certificates -> 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://cybersec.com.au:4433.** Please describe and explain your observations. Please also do the following tasks: