The exercises are designed for students to finish in an individual capacity. The exercises are not designed to be completed in tutorial sessions but rather to give you some tasks and a starting point to continue and complete on your own.

Lab Overview

Public key cryptography is the foundation of today's secure communication, but it is subject to man-in-the-middle attacks when one side of communication sends its public key to the other side. The fundamental problem is that there is no easy way to verify the ownership of a public key, i.e., given a public key and its claimed owner information, how do we ensure that the public key is indeed owned by the claimed owner? The Public Key Infrastructure (PKI) is a practical solution to this problem. The learning objective of this lab is for students to gain the first-hand experience on PKI. By doing the tasks in this lab, students should be able to gain a better understanding of how PKI works, how PKI is used to protect the Web, and how Man-in-the-middle attacks can be defeated by PKI. Moreover, students will be able to understand the root of the trust in the public-key infrastructure, and what problems will arise if the root trust is broken.

Lab Tasks

Open SecureCorp network configuration in GNS3 and start all nodes. We will use `Server-1` from Server LAN as Certificate Authority (CA) and `Web` from DMZ as web server.

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; 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.

   We will use `Server-1` container as web server and access the website using one of the clients. The openssl configuration file is located in: `/usr/lib/ssl/openssl.cnf`. According to configuration file we need to create directories for our certificates, see `[CA_default]` part in the configuration file. Let's make directories in `Server-1`, open terminal (right click on the `Server-1` node and select `console`), and execute the following:

```
apt update
apt install openssl
apt install nano
cd
mkdir pki
cd pki
mkdir demoCA
cd demoCA
mkdir certs
mkdir crl
mkdir newcerts
touch index.txt
echo 1000 > serial
cd ..
```

By default OpenSSL configures itself as an internal CA which is only signing certificates for the same organisation. We need to change this configurations so that our CA can sign certificate for any domain name. Find the following section in the configuration file `/usr/lib/ssl/openssl.cnf`. and change them as below.

```
[ policy_match ]
countryName = optional
stateOrProvinceName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
```

demoCA ⇒ Where everything is kept.
certs ⇒ Where the issued certs are kept.
crl ⇒ Where the issued crl are kept.
newcerts ⇒ default place for new certs.
index.txt ⇒ database index file.
serial ⇒ The current serial number

Alright, we are ready 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 a self-signed certificate for the CA:

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

2. **Creating a Certificate for networksecurity.com**
Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called `networksecurity.com`. We assume that the web server of `networksecurity.com` is `Web` node in `DMZ`. Perform the following three steps in `Web` node.

   (a) **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). The keys will be stored in the file `server.pem`:

   ```
   cd
   apt install openssl
   openssl genrsa -out server.pem 2048
   ```

   The `server.pem` is an encoded text file, 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 server.pem -text
   ```

   (b) **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 true identity). Please use **networksecurity.com** as the common name of the certificate request.

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

   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.

(c) **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. Copy the content of `server.csr` from `Web` to `Server-1`. Execute the following command on `Server-1` to turn 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
```

Copy the certificate file (`server.crt`) to the `Web` server.

3. **Deploying Certificate in an HTTPS Web Server (Apache)**
Perform the following on `Web` node.

Install Apache server and a text editor (nano):

```
apt install nano
apt install apache2
```

An Apache server can simultaneously host multiple websites. It needs to know the directory where a website's files are stored. This is done via its `VirtualHost` file, located in the `/etc/apache2/sites-available` directory. To add an HTTP website, we add a `VirtualHost` entry to the file `000-default.conf` (using nano `/etc/apache2/sites-available/000-default.conf`). Add the following lines at the end of the file:

```
<VirtualHost *:80>
ServerName networksecurity.com
DocumentRoot /var/www/networksecurity
DirectoryIndex index.html
</VirtualHost>
```

and to the file `/etc/apache2/sites-available/default-ssl.conf` (before line `</IfModule>`):

```
<VirtualHost *:443>
ServerName networksecurity.com
DocumentRoot /var/www/networksecurity
DirectoryIndex index.html

SSLEngine On
SSLCertificateFile /etc/apache2/server.crt
SSLCertificateKeyFile /etc/apache2/server.pem
</VirtualHost>
```

The `ServerName` entry specifies the name of the website, while the `DocumentRoot` entry specifies where the files for the website are stored. The above example sets up the HTTPS site `https://networksecurity.com` (port 443 is the default HTTPS port). In the setup, we need to tell Apache where the server certificate (`/etc/apache2/server.crt`) and private key (`/etc/apache2/server.pem`) are stored.

First, we will create our `DocumentRoot` directory and copy our cert/private key files in apache directory; then we will run few commands to test our apache configuration and finally we will restart our apache server:

```
mkdir /var/www/networksecurity
cp server.crt /etc/apache2/server.crt
cp server.pem /etc/apache2/server.pem
echo 'ServerName 127.0.0.1'>> /etc/apache2/apache2.conf
apachectl configtest
a2enmod ssl
a2ensite default-ssl
service apache2 restart
```

Let us also create one test html file:

```
nano /var/www/networksecurity/index.html
```

Copy the following in html file and press `ctrl+x` to close and save.

```
<!DOCTYPE html>
<html>
<body>

<h1>Network Security</h1>
<p>This is a test page.</p>

</body>
</html>
```

4. **Browsing** `networksecurity.com`
We can use one of our clients to visit the newly created HTTPS website. Open on `Internal-Client` and edit the `/etc/hosts` file. We point the URL **neworksecurity.com** to our server:

```
nano /etc/hosts
```

And add the following line at the end of file:

```
10.10.2.80 networksecurity.com
```

Press `ctrl+x` to save and exit. We will use text-based browser `lynx` to browse the website:

```
apt install lynx
lynx https://networksecurity.com
```

Why `lynx` is complaining about the certificate? Press `y` to accept the risk. You should see your webpage (`index.html`).

Had our certificate been assigned by VeriSign, we will not have such an error message, because VeriSign's certificate is very likely preloaded into lynx's certificate repository already. Unfortunately, the certificate of `networksecurity.com` is signed by our own CA (i.e., using `ca.crt`), and this CA is not recognized by `lynx`.

We can include our CA certificate in `lynx`. In `Server-1`, copy the content of `ca.crt`:

```
cat ca.crt
```

Select the contents and copy it. In `Internal-Client`, open ca-certificates file:

```
apt install ca-certificates
nano /etc/ssl/certs/ca-certificates.crt
```

Paste contents of certificate, which you copied from server, at the top of `ca-certificates.crt` file. Now again `lynx` to `networksecurity.com`, and this time it should not complain about the certificate.

Acknowledgement
Parts of this lab is based on the SEED project (Developing Instructional Laboratory for Computer Security Education) `https://seedsecuritylabs.org`.