Eric Wong: 501174088
Rohan Manoharan: 501189408
Thylane Rossi: 501340990

Labs 3 - Public-Key Infrastructure (PKI) Lab

Task 1

```
[10/13/24]seed@VM:~$ cp /usr/lib/ssl/openssl.cnf ./openssl.cnf
```

Creating the openssl config file in home directory

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

```
[10/13/24]seed@VM:~$ mkdir ./demoCA
[10/13/24]seed@VM:~$ cd ./demoCA
[10/13/24]seed@VM:~/demoCA$ mkdir certs
[10/13/24]seed@VM:~/demoCA$ mkdir crl
[10/13/24]seed@VM:~/demoCA$ mkdir newcerts
[10/13/24]seed@VM:~/demoCA$ touch index.txt
[10/13/24]seed@VM:~/demoCA$ echo "1000" > serial
```

Creating sub-directories as specified by [CA_default]

mkdir ./demoCA
cd ./demoCA
mkdir certs
mkdir crl
mkdir newcerts
touch index.txt
echo "1000" > serial

Generating self-signed certificate for CA

*openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 *
*-keyout ca.key -out ca.crt *
*-subj "/CN=www.modelCA.com/O=Model CA LTD./C=US" *
-passout pass:dees

Doing this means that CA is totally trusted, certificate will be treated as root certificate

```
[10/15/24]seed@VM:~$ openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \
> -keyout ca.key -out ca.crt \
> -subj "/CN=www.modelCA.com/O=Model CA LTD./C=US" \
> -passout pass:dees
Generating a RSA private key
.....................++++
.....................................................................................
................++++
writing new private key to 'ca.key'
-----
```

Set password as dees with the command, however can be anything the user desires
Press enter to skip all other fields – gets left as blank

openssl x509 -in ca.crt -text -noout

```
[10/15/24]seed@VM:~$ openssl x509 -in ca.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3d:41:83:a3:21:4a:a3:07:14:9d:90:78:f0:68:93:aa:65:54:9e:e8
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = www.modelCA.com, O = Model CA LTD., C = US
        Validity
            Not Before: Oct 15 16:16:14 2024 GMT
            Not After : Oct 13 16:16:14 2034 GMT
        Subject: CN = www.modelCA.com, O = Model CA LTD., C = US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:


        X509v3 extensions:
            X509v3 Subject Key Identifier:
                50:04:8F:A6:58:C9:ED:8F:64:F7:C1:58:0E:68:16:88:0D:23:25:B2
            X509v3 Authority Key Identifier:
                keyid:50:04:8F:A6:58:C9:ED:8F:64:F7:C1:58:0E:68:16:88:0D:23:25:B2

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
```

Modulus and Signature Algorithm is omitted in the above screenshots

openssl rsa -in ca.key -text -noout

Running this command results in reading the modulus, publicExponent, privateExponent, prime1, prime2, exponent1, exponent2, and the coefficient

What part of the certificate indicates this is a CA's certificate?

- The part where it says CA:TRUE under the X509v3 Basic Constraints

What part of the certificate indicates this is a self-signed certificate?

- Since the Issuer and Subject fields are identical, this suggests that the certificate is self-signed

In the RSA algorithm, we have a public exponent e, a private exponent d, a modulus n, and two secret numbers p and q, such that n = pq. Please identify the values for these elements in your certificate and key files.

- After running the command that reads *ca.key* this information is outputted.
  - publicExponent e: 65537 (0x10001)
  - privateExponent d: listed in the privateExponent section
  - modulus n: listed in the Modulus section
  - secret number p: prime1 listed in its respective section
  - secret number q: prime2 listed in its respective section

Task 2

Run following command to generate RSA key pair for the company

openssl req -newkey rsa:2048 -sha256 \
-keyout server.key -out server.csr \
-subj "/CN=www.test24.com/O=Test24 Inc./C=US" \
-passout pass:dees

```
[10/15/24]seed@VM:~$ openssl req -newkey rsa:2048 -sha256 \
> -keyout server.key -out server.csr \
> -subj "/CN=www.bank32.com/O=Bank32 Inc./C=US" \
> -passout pass:dees
Generating a RSA private key
....+++++
..............+++++
writing new private key to 'server.key'
-----
```

You can view this by inputting the following commands:

openssl req -in server.csr -text -noout
openssl rsa -in server.key -text -noout

```
[10/15/24]seed@VM:~/demoCA$ openssl req -newkey rsa:2048 -sha256 \
> -keyout server.key -out server.csr \
> -subj "/CN=www.bank32.com/O=Bank32 Inc./C=US" \
> -passout pass:dees \
> -addext "subjectAltName = DNS:www.bank32.com, \
> DNS:www.bank32A.com, \
> DNS:www.bank32B.com"
Generating a RSA private key
.....+++++
.................+++++
writing new private key to 'server.key'
-----
```

Adding two alternative names to the CSR via the following command:

*openssl req -newkey rsa:2048 -sha256 *
*-keyout server.key -out server.csr *
*-subj "/CN=www.test24.com/O=Test24 Inc./C=US" *
*-passout pass:dees *
*-addext "subjectAltName = DNS:www.test24.com, *
*DNS:www.test24A.com, *
DNS:www.test24B.com"

Task 3: Generating a Certificate for The Server

Step 1: Generating a Certificate

Copy openssl to the demoCA folder. Unable to generate a certificate without it and the server key in the same directory

```
ca.crt  ca.key  demoCA  Desktop  Documents  Downloads  Music  openssl.cnf  Pictures  Public  Templates  Videos
[10/15/24]seed@VM:~$ cp openssl.cnf demoCA/
```

The command following was used to generate a certificate. The passphrase *dees* was inputted, along with the common name of *test24.com.* This issues information of the certificate, though most information is left blank as it is not important for the purposes of a lab.

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

```
[10/15/24]seed@VM:~/.../Labsetup$ openssl ca -config openssl.cnf -policy policy_anything -md sha256 -days 3650 -in
 server.csr -out server.crt -batch -cert ca.crt -keyfile ca.key
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Oct 16 02:20:58 2024 GMT
            Not After : Oct 14 02:20:58 2034 GMT
        Subject:
            countryName               = US
            organizationName          = Test24 Inc.
            commonName                = www.test24.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                C7:BF:E4:AD:BD:F1:78:8C:F8:F0:FB:A7:B1:84:0D:8F:F2:DB:D5:CC
            X509v3 Authority Key Identifier:
                keyid:D2:D5:4B:08:D4:94:FE:25:9D:04:1A:8C:02:EF:97:DB:A7:05:58:B5

Certificate is to be certified until Oct 14 02:20:58 2034 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
```

Step 2: Copying Extension Field

Edited line 68 of *openssl.cnf* to be uncommented, allowing extensions to be copied

```
# Extension copying option: use with caution.
copy_extensions = copy
```

Task 4: Deploying Certificate in an Apache-Based HTTPS Website

Step 1: Hosting the Website

Edited both the dockerfile and the apache_ssl file in the lab setup folder to match with the certificates, keys and server names affiliated with test24. The following changes were made in the screenshot, then the docker container was built.

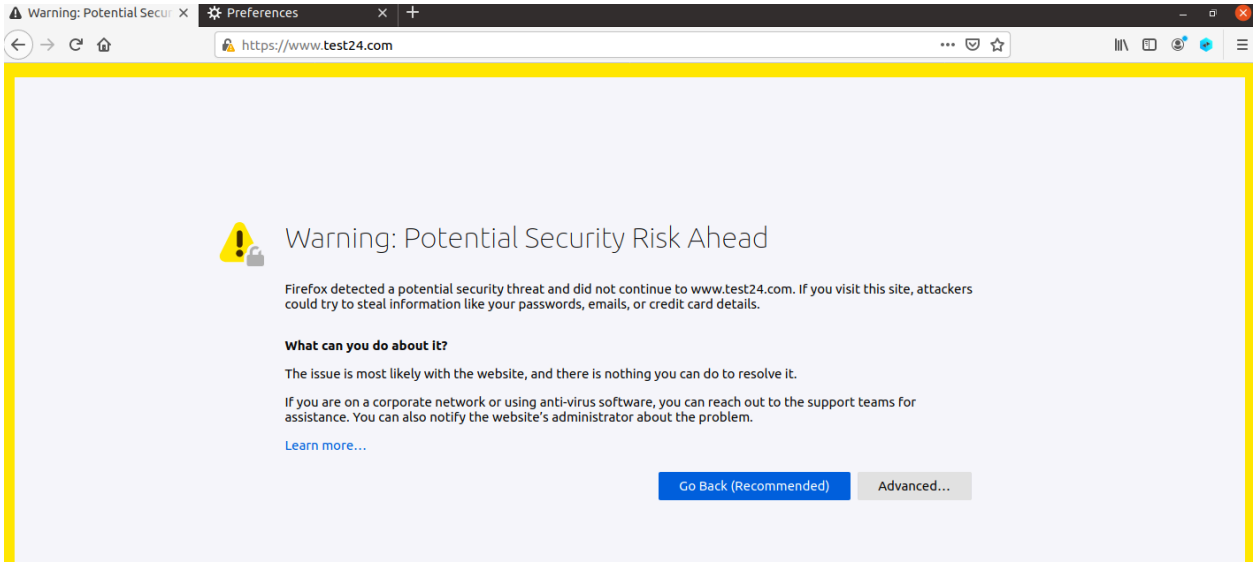
Open *Dockerfile ~/Desktop/Labsetup/image_www

```
1 FROM handsonsecurity/seed-server:apache-php
2
3 ARG WWWDIR=/var/www/test24
4
5 COPY ./index.html ./index_red.html $WWWDIR/
6 COPY ./test24_apache_ssl.conf /etc/apache2/sites-available
7 COPY ./certs/ca.crt ./certs/ca.key  /certs/
8
9 RUN  chmod 400 /certs/ca.key \
10      && chmod 644 $WWWDIR/index.html \
11      && chmod 644 $WWWDIR/index_red.html \
12      && a2ensite test24_apache_ssl
13
14 CMD  tail -f /dev/null
15
```

Open *bank32_apache_ssl.conf ~/Desktop/Labsetup/image_www

```
1 <VirtualHost *:443>
2     DocumentRoot /var/www/bank32
3     ServerName www.test24.com
4     ServerAlias www.test24A.com
5     ServerAlias www.test24B.com
6     ServerAlias www.test24W.com
7     DirectoryIndex index.html
8     SSLEngine On
9     SSLCertificateFile /certs/test24.crt
10    SSLCertificateKeyFile /certs/test24.key
11 </VirtualHost>
12
13 <VirtualHost *:80>
14    DocumentRoot /var/www/test24
15    ServerName www.test24.com
16    DirectoryIndex index_red.html
17 </VirtualHost>
18
19 # Set the following gloal entry to suppress an annoying warning message
20 ServerName localhost
```

The command *service apache2 start* was then used, allowing www.test24.com to be hosted, resulting in a page deemed a potential security risk.



Step 2: Add Certificate To Website

Going to “certificate” in Firefox’s preferences page, then navigating to View Certificates > Certificate Manager > Authorities and importing modelCA.crt will fix the issue, resulting in this

Task 5: Launching a Man-In-The-Middle Attack

Step 1:
We take the instagram site to be attacked, so we replace it in the ServerName file instagram.com.

Open *bank32_apache_ssl.conf
~/Desktop/Labsetup/image_www

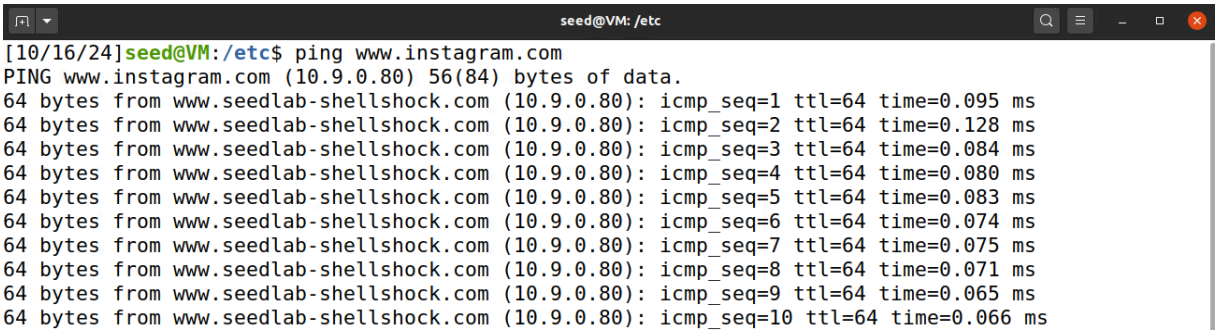
```
<VirtualHost *:443>
    DocumentRoot /var/www/bank32
    ServerName www.instagram.com
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile /certs/bank32.crt
    SSLCertificateKeyFile /certs/bank32.key
</VirtualHost>

<VirtualHost *:80>
    DocumentRoot /var/www/bank32
    ServerName www.instagram.com
    DirectoryIndex index_red.html
</VirtualHost>

# Set the following gloal entry to suppress an annoying warning message
ServerName localhost
```

Step 2:
Here's the addition of the line *10.9.0.80 www.instagram.com* in the file hosts to simulate the DNS attack.

seed@VM: /etc

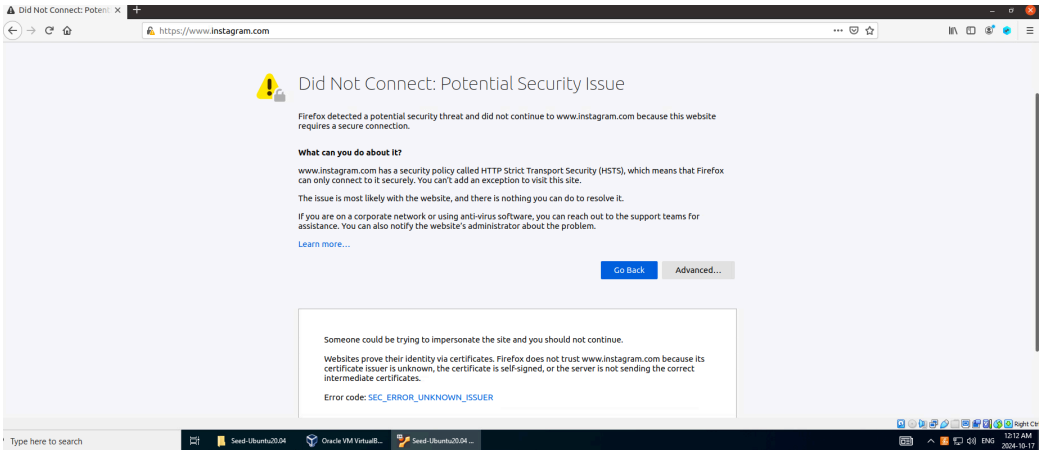
```
 GNU nano 4.8                    hosts
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

# For CSRF Lab
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

# For Shellshock Lab
10.9.0.80       www.seedlab-shellshock.com
10.9.0.80       www.instagram.com

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^_ Go To Line
```

We can check with the ping command that the instagram site is indeed redirected to the site whose address is 10.9.0.80.

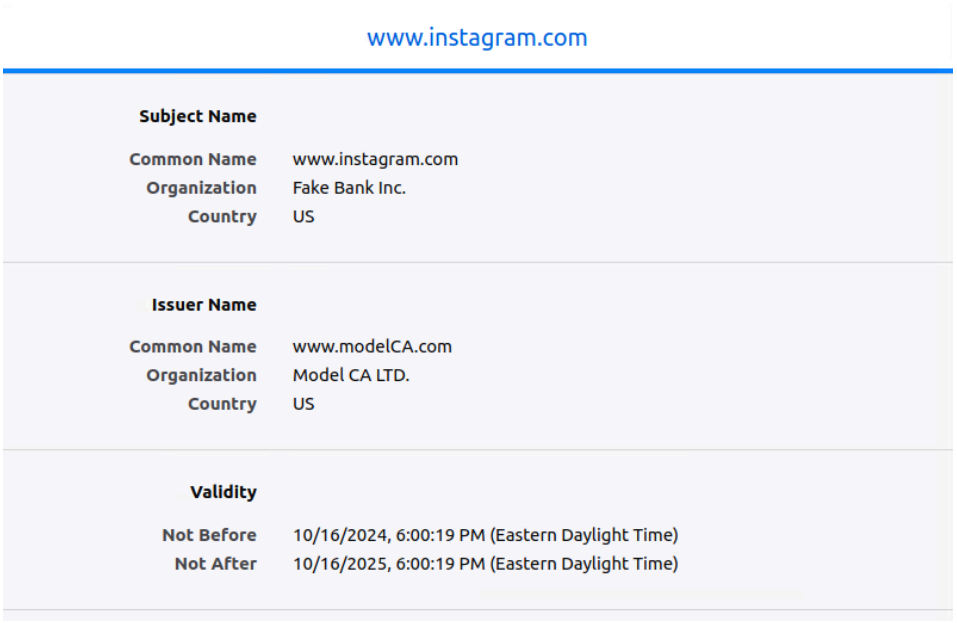


Step 3 ：

When I tried to access www.instagram.com on my computer, I was redirected to my “malicious site”, although there is a security warning.



Here's the site certificate.



By adding the certificate to the firefox page (Task 6), you can access this non-malicious page (the same as in task 4). But if you create a fake instagram authentication page, it's easy to retrieve user credentials.

Task 6: Launching a Man-In-The-Middle Attack with a Compromised CA

Here's my new file, this time with a certificate for instagram and the SSLCACertificateFile line that lets me add the CA.

```
<VirtualHost *:443>
    DocumentRoot /var/www/bank32
    ServerName www.instagram.com
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile /certs/instagram.crt
    SSLCertificateKeyFile /certs/instagram.key
    SSLCACertificateFile /certs/ca.crt
</VirtualHost>

<VirtualHost *:80>
    DocumentRoot /var/www/bank32
    ServerName www.instagram.com
    DirectoryIndex index_red.html
</VirtualHost>

# Set the following gloal entry to suppress an annoying warning message
ServerName localhost
```

After recovering the certificate used in the docker, as in task 4, it was added to the trusted certificates of the firefox search engine. After that, the www.instagram.com page was directly redirected to the site whose ip address is 10.9.0.80. Here we can see that Instagram has been replaced by “Hello world”. But someone malicious could create a site with exactly the same interface as instagram to fool the user.