Public-Key Infrastructure (PKI) Lab

1 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. SEED labs have a series of labs focusing on the public-key cryptography, and this one focuses 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. This lab covers the following topics:

- Public-key encryption
- Public-Key Infrastructure (PKI)
- Certificate Authority (CA) and root CA
- X.509 certificate and self-signed certificate
- Apache, HTTP, and HTTPS
- Man-in-the-middle attacks

2 Lab Tasks

2.1 Task 1: Becoming a Certifificate Authority (CA)

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

The Confifiguration File openssl.conf**.**

In order to use OpenSSL to create certifificates, you have to
have a confifiguration fifile. The confifiguration fifile usually has an extension .cnf. It is used by three OpenSSL
commands: ca, req and x509. The manual page of openssl.conf can be found using Google search. You can also get a copy of the confifiguration fifile from /usr/lib/ssl/openssl.cnf. After copying
this fifile into your current directory, you need to create several sub-directories as specified in the confifiguration fifile (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
```

```
abe@abe-VirtualBox:~$ cp /usr/lib/ssl/openssl.cnf openssl.cnf
abe@abe-VirtualBox:~$ ls
Desktop    Downloads  openssl.cnf  Public     Videos
Documents  Music      Pictures     Templates
abe@abe-VirtualBox:~$ mkdir demoCA
abe@abe-VirtualBox:~$ cd demoCA/
abe@abe-VirtualBox:~/demoCA$ mkdir certs crl newcerts
abe@abe-VirtualBox:~/demoCA$ touch index.txt serial
abe@abe-VirtualBox:~/demoCA$ echo 1000 > serial
abe@abe-VirtualBox:~/demoCA$ ls
certs  crl  index.txt  newcerts  serial
abe@abe-VirtualBox:~/demoCA$ cd ..
abe@abe-VirtualBox:~$ ls
demoCA   Documents  Music        Pictures  Templates
Desktop  Downloads  openssl.cnf  Public    Videos
abe@abe-VirtualBox:~$ 
```

For the index.txt fifile, simply create an empty fifile. For the serial fifile, put a single number in string format (e.g. 1000) in the fifile. Once you have set up the confifiguration fifile openssl.cnf, you can
create and issue certifificates.

Certifificate Authority (CA).

As we described before, we need to generate a self-signed certifificate for our CA. This means that this CA is totally trusted, and its certifificate will serve as the root certifificate. You can run the following command to generate the self-signed certifificate 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 fifill in some information, such as the Country Name, Common Name, etc. The output of the command are stored in two fifiles: ca.key and ca.crt. The fifile ca.key contains the CA's private key, while ca.crt contains the public-key certifificate.

```
openssl.cnf
Generating a RSA private key
.............................................+++++
................................................................................
...............................................+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ET
State or Province Name (full name) [Some-State]:Addis Ababa
Locality Name (eg, city) []:Addis Ababa
Organization Name (eg, company) [Internet Widgits Pty Ltd]:AAiT
Organizational Unit Name (eg, section) []:SiTE
Common Name (e.g. server FQDN or YOUR name) []:abe
Email Address []:abrahamshimekt4096@gmail.com
abe@abe-VirtualBox:~$
```

The above provide the screenshot of the provided information. The output of the file is stored in two files: ca.key and ca.crt(as provided in the command). ca.key contains the CA's private key and ca.crt contains the CA's public key. The following provides the content of the ca.crt.

abe

Identity: abe
Verified by: abe
Expires: 07/19/2022

▾ Details

Subject Name

C (Country):	ET
ST (State):	Addis Ababa
L (Locality):	Addis Ababa
O (Organization):	AAiT
OU (Organizational Unit):	SiTE
CN (Common Name):	abe
EMAIL (Email Address):	abrahamshimekt4096@gmail.com

Issuer Name

C (Country):	ET
ST (State):	Addis Ababa
L (Locality):	Addis Ababa
O (Organization):	AAiT
OU (Organizational Unit):	SiTE
CN (Common Name):	abe
EMAIL (Email Address):	abrahamshimekt4096@gmail.com

Issued Certificate

Version:	3
Serial Number:	6C 9C 7A D2 69 75 5F 91 80 7B C2 11 A6 E5 37 BA 14 8C DD CE
Not Valid Before:	2022-06-19
Not Valid After:	2022-07-19

Certificate Fingerprints

SHA1:	6F 36 E8 D1 2F 54 2E 3C 70 DB 9D 8C 02 8A 26 41 2A 56 AA 11
MD5:	98 FF C4 70 D9 2F 59 3B 33 0D 38 FC EB 96 B2 78

Public Key Info

Key Algorithm:	RSA
Key Parameters:	05 00
Key Size:	2048
Key SHA1 Fingerprint:	71 8C CC AA FA 20 48 9F B5 EC 09 0A D4 3D 0F EE AE 92 DC 89
Public Key:	30 82 01 0A 02 82 01 01 00 DC 99 17 4D 67 69 3D 8F C1 16 A2 92 0D 9E 7F 06 86 86 D3 2B 8B DE CB 00 BB 5B 17 AA 13 0E 33 C5 10 3A 29 D2 3E C7 F3 C5 C8 18 FC 7E D7 7F C2 62 DC F2 92 94 7C 0B 0F 66 D3 20 44 FD 71 F5 AF E7 01 0F A0 D2 B6

Key Algorithm:	RSA
Key Parameters:	05 00
Key Size:	2048
Key SHA1 Fingerprint:	71 8C CC AA FA 20 48 9F B5 EC 09 0A D4 3D 0F EE AE 92 DC 89
Public Key:	30 82 01 0A 02 82 01 01 00 DC 99 17 4D 67 69 3D 8F C1 16 A2 92 0D 9E 7F 06 86 86 D3 2B 8B DE CB 00 BB 5B 17 AA 13 0E 33 C5 10 3A 29 D2 3E C7 F3 C5 C8 18 FC 7E D7 7F C2 62 DC F2 92 94 7C 0B 0F 66 D3 20 44 FD 71 F5 AF E7 01 0F A0 D2 B6 62 7D EC 97 8B BD 56 25 7B C0 7B A5 04 80 BF 81 3A 3D 68 03 90 C8 36 56 F3 97 C9 67 46 E0 80 B9 1E 5B 8C 96 79 F2 C2 ED 5F E8 D9 F6 BF FC F7 94 DD 5B A1 48 59 97 E2 69 0D 30 1E 19 A7 58 63 BA 3A 0B E2 79 81 00 64 E2 3D 52 40 57 DE C7 8F A4 CE 71 89 16 7E 42 6E 4A 47 4F BB B4 F2 EE 1F D3 E2 47 9A 66 54 88 80 B6 29 3F 74 1E 42 75 0F 02 91 9C AB CB BC A8 5C A9 50 19 8D D6 E2 A9 D5 1B C5 86 5B A5 7A B7 2D 73 EE 8A A2 68 CE 1D FB 27 48 A1 67 C0 FF 85 77 0C D7 D9 58 62 D1 7F 62 94 63 CE FF E0 DA D2 15 65 2E A3 D8 45 40 11 59 1A 0B 32 C6 5A 28 DC 2C B0 DD 4F D3 02 03 01 00 01
Subject Key Identifier	
Key Identifier:	52 11 53 0A 14 0F 4C E5 01 11 62 90 5E D7 A8 D2 D6 27 D3 E1
Critical:	No
Extension	
Identifier:	2.5.29.35
Value:	30 16 80 14 52 11 53 0A 14 0F 4C E5 01 11 62 90 5E D7 A8 D2 D6 27 D3 E1
Critical:	No
Basic Constraints	
Certificate Authority:	Yes
Max Path Length:	Unlimited
Critical:	Yes
Signature	
Signature Algorithm:	1.2.840.113549.1.1.11
Signature Parameters:	05 00
Signature:	8A BD 8D B4 2F 35 C0 07 DB 65 B4 8B 40 41 C3 70 D3 11 77 13 76 A1 BF D9 09 35 2B 84 D1 28 E7 EC 66 E4 5D 9E 4E C9 93 C3 02 C1 EC 20 38 1C E0 FC 6E 35 BA F8 9D 7B C7 46 3B 95 57 22 24 6B AD 2D 0C D8 4E 43 DB 5C 3D 85 32 76 DC B5 FB F4 15 09 4F 31 01 82 BF 93 E2 CC 5F CF 99 18 6A 54 F7 41 14 3C 9A C1 FA 0B AB BA 1E EC 58 88 96 2D BC 4C 50 49 52 B4 78 BC 1F A4 B9 F6 18 FD 02 2F C1 86 C3 86 55 C5 79 D3 8D B8 7D 57 9E 6D D5 F7 AF 3F 75 70 3F DD 92 15 57 F8 35 0B C0 BC 25 C0 BF 39 CA CD ED BA 17 EA B9 C5 CA 4B A3 B4 43 71 46 3C 87 66 7E FC EB 50 4A D0 9E BD 0C 9F C7 60 26 24 FB 7C 6A 45 A5 98 F1 11 03 F5 9E 6A 90 32 FC D0 52 DD AB CA 36 B2 91 C2 5C 9D 7A 49 B6 15 15 01 CF 08 C6 85 A6 A8 18 2B 6D 4C 02 8B 30 FC E8 BB 86 7A 1F 15 18 B2 4C 24 60 F4 C8 70 74 46 F6 C0

We see that the subject and the issuer are the same, indicating that a self -signed certificate. Also the Certificate authority is set to Yes in the basic constraint, indicating that this certificate can be used to issue and sign another certificate, hence becoming the certificate of the certificate authority(CA).this certificate can be used as the root certificate in this lab.

2.2 Task 2: Creating a Certifificate for SEEDPKILab2020.com

Now, we become a root CA, we are ready to sign digital certifificates for our customers. Our fifirst customer is a company called SEEDPKILab2020.com. For this company to get a digital certificate from a CA, it needs to go through three step

Step 1: Generate public/private key pair.

The company needs to fifirst 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 fifile server.key:

$ openssl genrsa -aes128 -out server.key 1024

```
abe@abe-VirtualBox:~$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.........................+++++
.................................+++++
e is 65537 (0x010001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
abe@abe-VirtualBox:~$
```

The server.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 server.key -text

Step 2: Generate a Certifificate Signing Request (CSR).

Once the company has the key fifile, it should generates a Certifificate Signing Request (CSR), which basically includes the company's public key. The CSR will be sent to the CA, who will generate a certifificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). Please use SEEDPKILab2020.com as the common name of the certifificate request.

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

It should be noted that the above command is quite similar to the one we used in creating the self-signed certifificate 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 certifificate.

```
abe@abe-VirtualBox:~$ openssl req -new -key server.key -out server.csr -config o
penssl.cnf
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ET
State or Province Name (full name) [Some-State]:Addis Ababa
Locality Name (eg, city) []:Addis Ababa
Organization Name (eg, company) [Internet Widgits Pty Ltd]:AAiT
Organizational Unit Name (eg, section) []:SiTE
Common Name (e.g. server FQDN or YOUR name) []:SEEDPKILAB2020.com
Email Address []:abrahamshimekt4096@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:abraham
An optional company name []:
abe@abe-VirtualBox:~$
```

The above CSR is then sent to the CA to generate a certificate for the key and company name.

Step 3: Generating Certifificates.

The CSR fifile needs to have the CA's signature to form a certifificate. In the real world, the CSR fifiles are usually sent to a trusted CA for their signature. In this lab, we will use our own trusted CA to generate certifificates. The following command turns the certifificate signing re
quest (server.csr) into an X509 certifificate (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

If OpenSSL refuses to generate certifificates, it is very likely that the names in your requests do not match with those of CA. The matching rules are specified in the confifiguration fifile (look at the [policy match] section). You can change the names of your requests to comply with the policy, or you can change the policy. The confifiguration fifile also includes another policy (called policy anything), which is less restrictive. You can choose that policy by changing the following line: "policy = policy_match" change to "policy = policy_anything".

```
abe@abe-VirtualBox:~$ openssl ca -in server.csr -out server.crt -cert ca.crt -ke
yfile ca.key -config openssl.cnf
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: Jun 19 10:10:53 2022 GMT
            Not After : Jun 19 10:10:53 2023 GMT
        Subject:
            countryName               = ET
            stateOrProvinceName       = Addis Ababa
            organizationName          = AAiT
            organizationalUnitName    = SiTE
            commonName                = SEEDPKILAB2020.com
            emailAddress              = abrahamshimekt4096@gmail.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                67:5C:B7:70:CB:36:08:85:BD:38:94:DE:F4:09:DD:91:1E:DB:21:1A
            X509v3 Authority Key Identifier:
                keyid:89:BE:CD:66:13:20:FF:90:83:6B:B4:91:96:00:FE:E8:CB:48:AB:8
B

Certificate is to be certified until Jun 19 10:10:53 2023 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

2.3 Task 3: Deploying Certifificate in an HTTPS Web Server

In this lab, we will explore how public-key certifificates are used by websites to secure web browsing. We will set up an HTTPS website using openssl's built-in web server.

Step 1: Confifiguring DNS.

We choose SEEDPKILab2020.com as the name of our website. To get our
computers recognize this name, let us add the following entry to /etc/hosts; this entry basically maps

the hostname **SEEDPKILab2020.com** to our localhost (i.e., 127.0.0.1):
127.0.0.1 SEEDPKILab2018.com

```
abe@abe-VirtualBox:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       abe-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
abe@abe-VirtualBox:~$
```

Step 2: Confifiguring the web server.

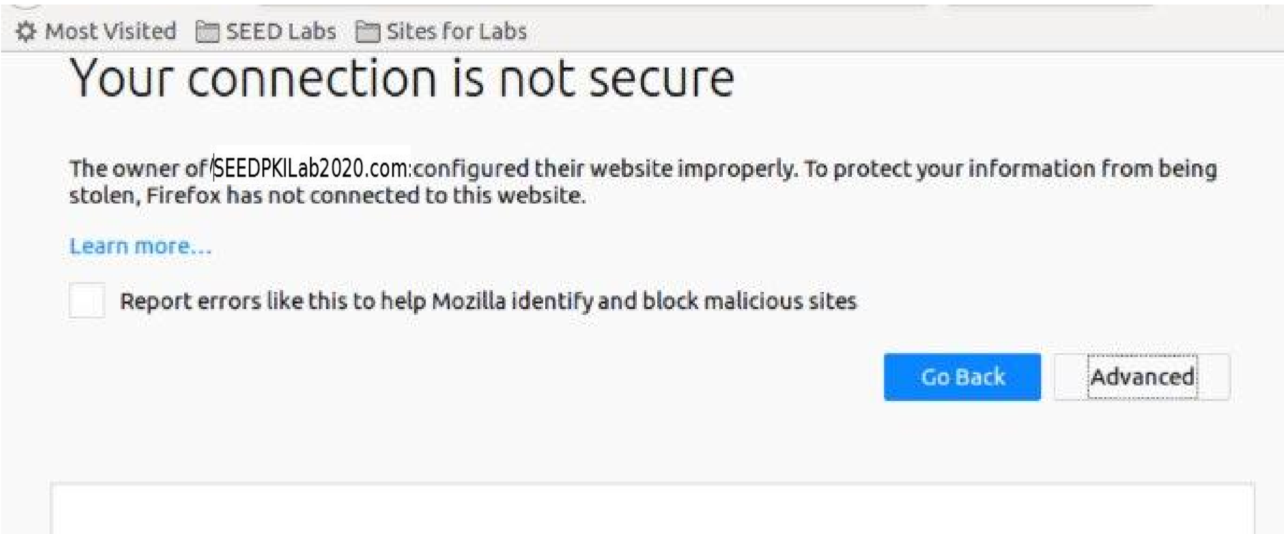
Let us launch a simple web server with the certifificate generated in
the previous task. OpenSSL allows us to start a simple web server using the s server command:

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://SEEDPKILab2020.com:4433/. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following:

*“**seedpkilab2018.com:4433 uses an invalid security certifificate**. The certifificate is not trusted because the*
issuer certifificate is unknown”.

```
abe@abe-VirtualBox:~$ cp server.key server.pem
abe@abe-VirtualBox:~$ cat server.crt >> server.pem
abe@abe-VirtualBox:~$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
error setting certificate
140607458067776:error:0909006C:PEM routines:get_name:no start line:../crypto/pem
/pem_lib.c:745:Expecting: DH PARAMETERS
140607458067776:error:140AB18F:SSL routines:SSL_CTX_use_certificate:ee key too s
mall:../ssl/ssl_rsa.c:310:
abe@abe-VirtualBox:~$
```



Step 3: Getting the browser to accept our CA certificate.

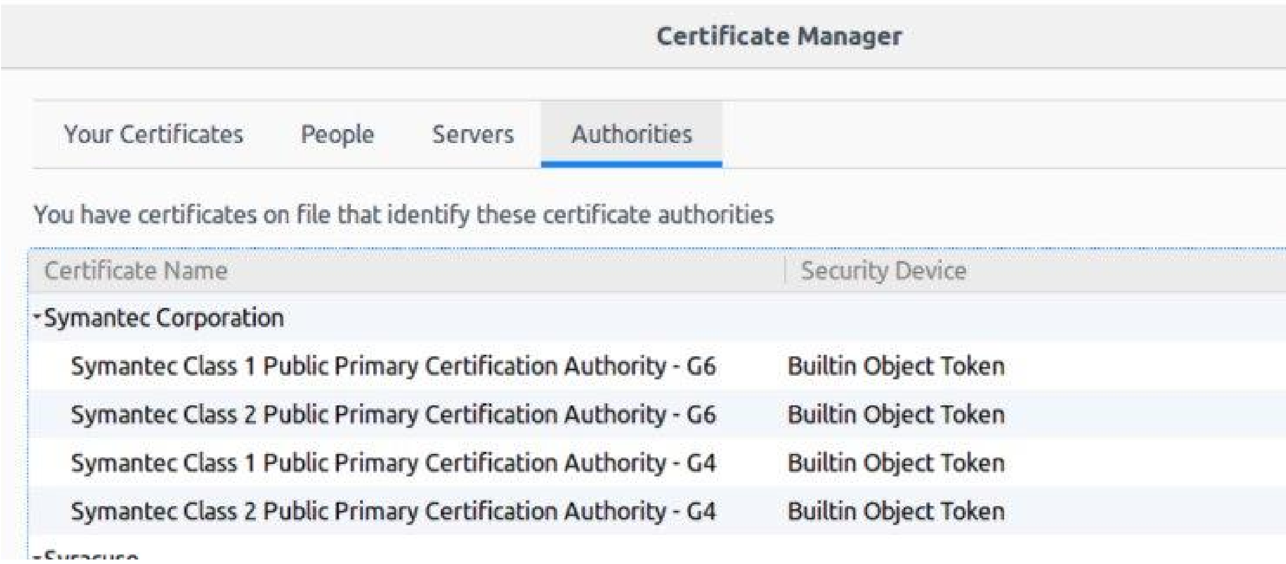
Had our certificate been assigned by VeriSign, we will not have such an error message, because VeriSign's certificate is very likely preloaded into Firefox's certificate repository already. Unfortunately, the certificate of SEEDPKILab2020.com is signed by our own CA (i.e., using ca.crt), and this CA is not recognized by Firefox. There are two ways to get Firefox to accept our CA's self-signed certificate.

• We can request Mozilla to include our CA's certificate in its Firefox software, so everybody using Firefox can recognize our CA. This is how the real CAs, such as VeriSign, get their certificates into Firefox. Unfortunately, our own CA does not have a large enough market for Mozilla to include our certificate, so we will not pursue this direction.

• **Load** ca.crt **into Firefox:** We can manually add our CA's certificate to the Firefox browser by clicking the following menu sequence:

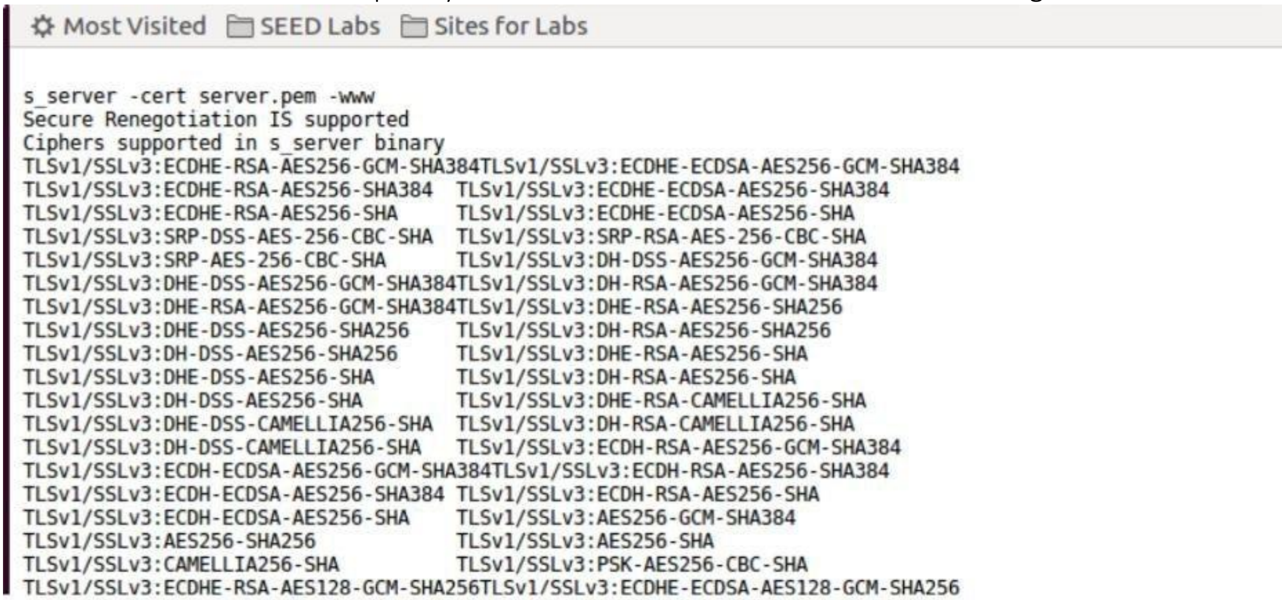
Edit -> Preference -> Privacy & Security -> View Certificates.

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.

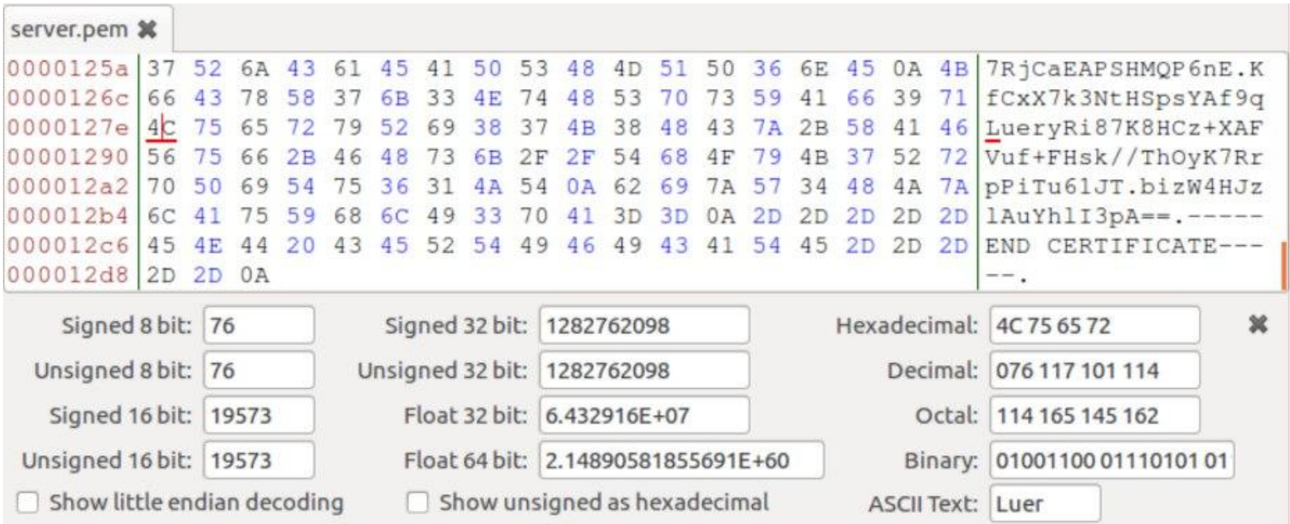


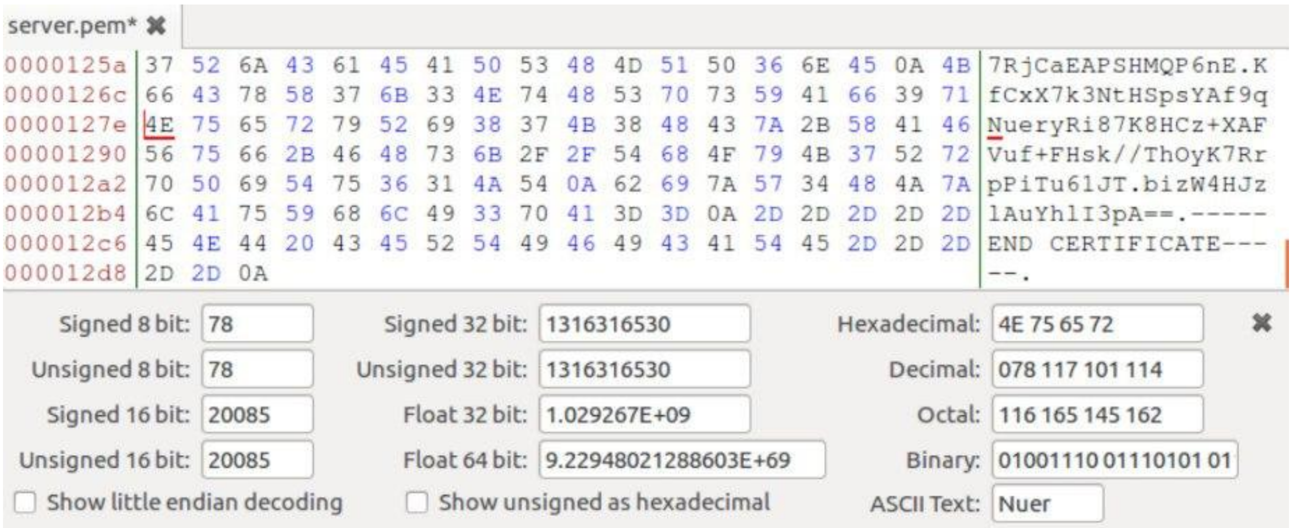
Step 4. Testing our HTTPS website.

Now, point the browser to https://SEEDPKILab2018.com:
4433. Please describe and explain your observations. Please also do the following tasks:



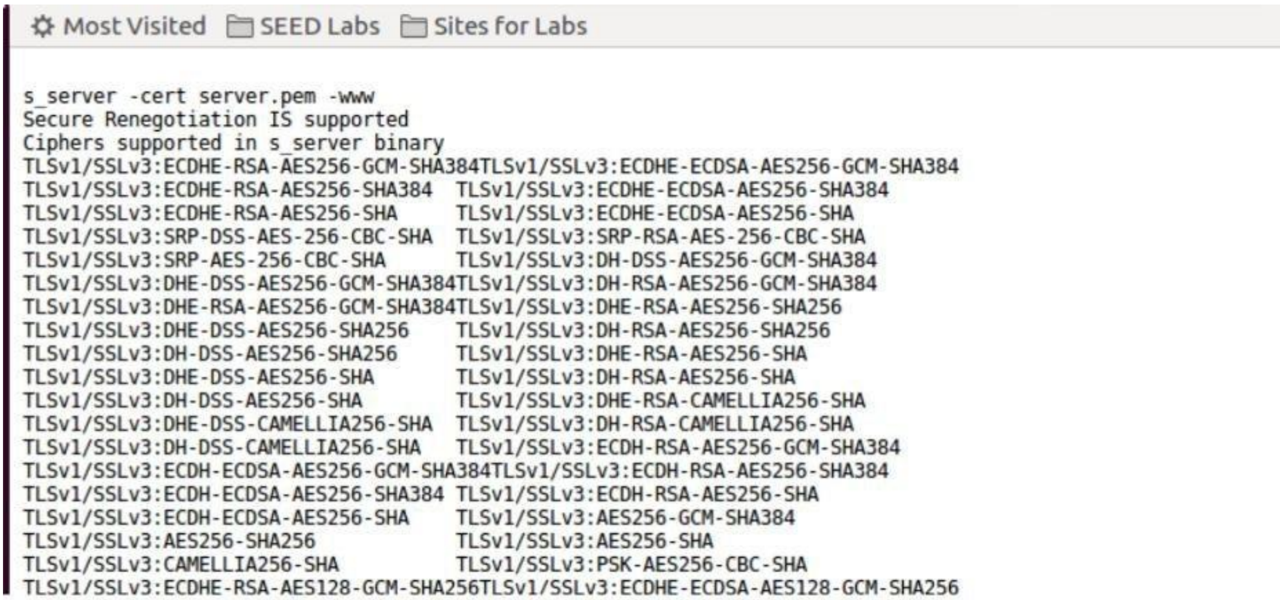
1. Modify a single byte of server.pem, and restart the server, and reload the URL. What do you observe?

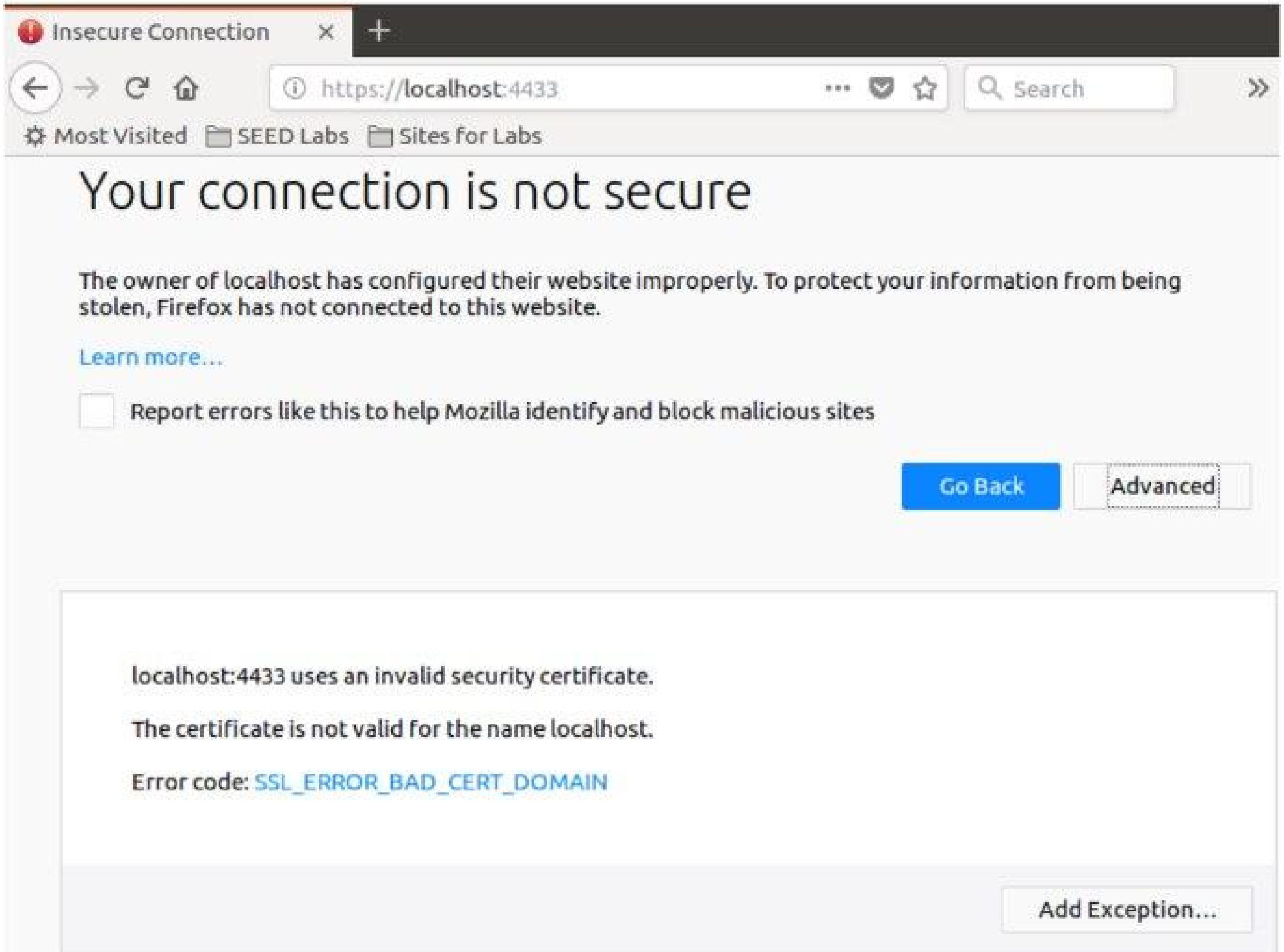


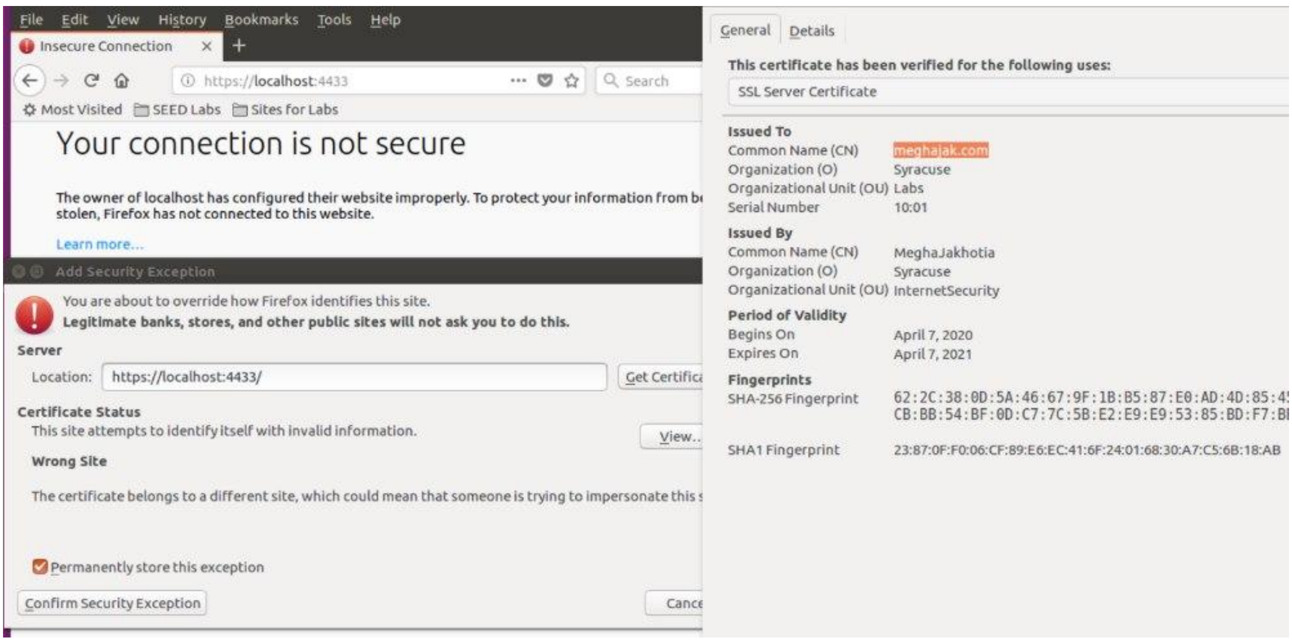


Make sure you restore the original server.pem afterward. Note: the server may not be able to restart if certain places of server.pem is corrupted; in that case, choose another place to modify.

2. Since SEEDPKILab2020.com points to the localhost, if we use https://localhost:4433 instead, we will be connecting to the same web server. Please do so, describe and explain your observations.




Insecure Connection
https://localhost:4433
Search
Most Visited
SEED Labs
Sites for Labs
Your connection is not secure
The owner of localhost has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.
Learn more...
Report errors like this to help Mozilla identify and block malicious sites
Go Back
Advanced
localhost:4433 uses an invalid security certificate.
The certificate is not valid for the name localhost.
Error code: SSL_ERROR_BAD_CERT_DOMAIN
Add Exception...


File Edit View History Bookmarks Tools Help
Insecure Connection
https://localhost:4433
Search
Most Visited
SEED Labs
Sites for Labs
Your connection is not secure
The owner of localhost has configured their website improperly. To protect your information from be
stolen, Firefox has not connected to this website.
Learn more...
Add Security Exception
You are about to override how Firefox identifies this site.
Legitimate banks, stores, and other public sites will not ask you to do this.
Server
Location: https://localhost:4433/
Get Certifica
Certificate Status
This site attempts to identify itself with invalid information.
View...
Wrong Site
The certificate belongs to a different site, which could mean that someone is trying to impersonate this s
Permanently store this exception
Confirm Security Exception
Cance
General
Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN) meghajak.com
Organization (O) Syracuse
Organizational Unit (OU) Labs
Serial Number 10:01
Issued By
Common Name (CN) MeghaJakhotia
Organization (O) Syracuse
Organizational Unit (OU) InternetSecurity
Period of Validity
Begins On April 7, 2020
Expires On April 7, 2021
Fingerprints
SHA-256 Fingerprint 62:2C:38:0D:5A:46:67:9F:1B:B5:87:E0:AD:4D:85:4
CB:BB:54:BF:0D:C7:7C:5B:E2:E9:E9:53:85:BD:F7:B
SHA1 Fingerprint 23:87:0F:F0:06:CF:89:E6:EC:41:6F:24:01:68:30:A7:C5:6B:18:AB

2.4 Task 4: Deploying Certifificate in an Apache-Based HTTPS Website

The HTTPS server setup using openssl's s server command is primarily for debugging and demon stration purposes. In this lab, we set up a real HTTPS web server based on Apache. The Apache server, which is already installed in our VM, supports the HTTPS protocol. To create an HTTPS website, we just need to confifigure the Apache server, so it knows where to get the private key and certifificates. We give an example in the following to show how to enable HTTPS for a website www.example.com. You task is to do the same for SEEDPKILab2020.com using the certificate generated from previous tasks.

An Apache server can simultaneously host multiple websites. It needs to know the directory where a website's fifiles are stored. This is done via its VirtualHost fifile, located in the /etc/apache2/ sites-available directory. To add an HTTP website, we add a VirtualHost entry to the fifile 000-default.conf. See the following example.

```
abe@abe-VirtualBox:~$ cp server.crt SEEDPKILAB2020_cert.pem
abe@abe-VirtualBox:~$ cp server.key SEEDPKILAB2020_key.pem
abe@abe-VirtualBox:~$ ls
ca.crt   Desktop    Music        Public                 server.crt  server.pem
ca.key   Documents  openssl.cnf  SEEDPKILAB2020_cert.pem  server.csr  Templates
demoCA   Downloads  Pictures     SEEDPKILAB2020_key.pem   server.key  Videos
abe@abe-VirtualBox:~$
```

<VirtualHost *:80>
ServerName one.example.com
DocumentRoot /var/www/Example_One
DirectoryIndex index.html
</VirtualHost>

To add an HTTPS website, we need to add a VirtualHost entry to the default-ssl.conf fifile in the same folder.

```
abe@abe-VirtualBox:/$ cd var
abe@abe-VirtualBox:/var$ cd www
abe@abe-VirtualBox:/var/www$ cd SEEDPKILAB2020
abe@abe-VirtualBox:/var/www/SEEDPKILAB2020$ ls
index.html
abe@abe-VirtualBox:/var/www/SEEDPKILAB2020$
```

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

In this task, we will show how PKI can defeat Man-In-The-Middle (MITM) attacks. Figure 1 depicts how MITM attacks work. Assume Alice wants to visit example.com via the HTTPS protocol. She needs to get the public key from the example.com server; Alice will generate a secret, and encrypt the secret using the server's public key, and send it to the server. If an attacker can intercept the communication between Alice and the server, the attacker can replace the server's public key with its own public key. Therefore, Alice's secret is actually encrypted with the attacker's public key, so the attacker will be able to read the secret. The attacker can forward the secret to the server using the server's public key. The secret is used to encrypt the communication between Alice and server, so the attacker can decrypt the encrypted communication.

The goal of this task is to help students understand how PKI can defeat such MITM attacks. In the task, we will emulate an MITM attack, and see how exactly PKI can defeat it. We will select a target website fifirst. In this document, we use example.com as the target website, but in the task, to make it more meaningful, students should pick a popular website, such as a banking site and social network site

Step 1: Setting up the malicious website.

In Task 4, we have already set up an HTTPS website for SEEDPKILab2020.com. We will use the same Apache server to impersonate example.com (or the site chosen by students). To achieve that, we will follow the instruction in Task 4 to add a VirtualHost entry to Apache's SSL confifiguration fifile: the ServerName should be example.com, but the rest of the confifiguration can be the same as that used in Task 4. Our goal is the following: when a user tries to visit example.com, we are going to get the user to land in our server, which hosts a fake website for example.com. If this were a social network website, The fake site can display a login page similar to the one in the target website. If users cannot tell the difference, they may type their account credentials in the fake webpage, essentially disclosing the credentials to the attacker

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

    SSLEngine On
    SSLCertificateFile      /home/seed/shared/Lab10/meghajak_cert.pem
    SSLCertificateKeyFile   /home/seed/shared/Lab10/meghajak_key.pem
</VirtualHost>
```

Step 2: Becoming the man in the middle

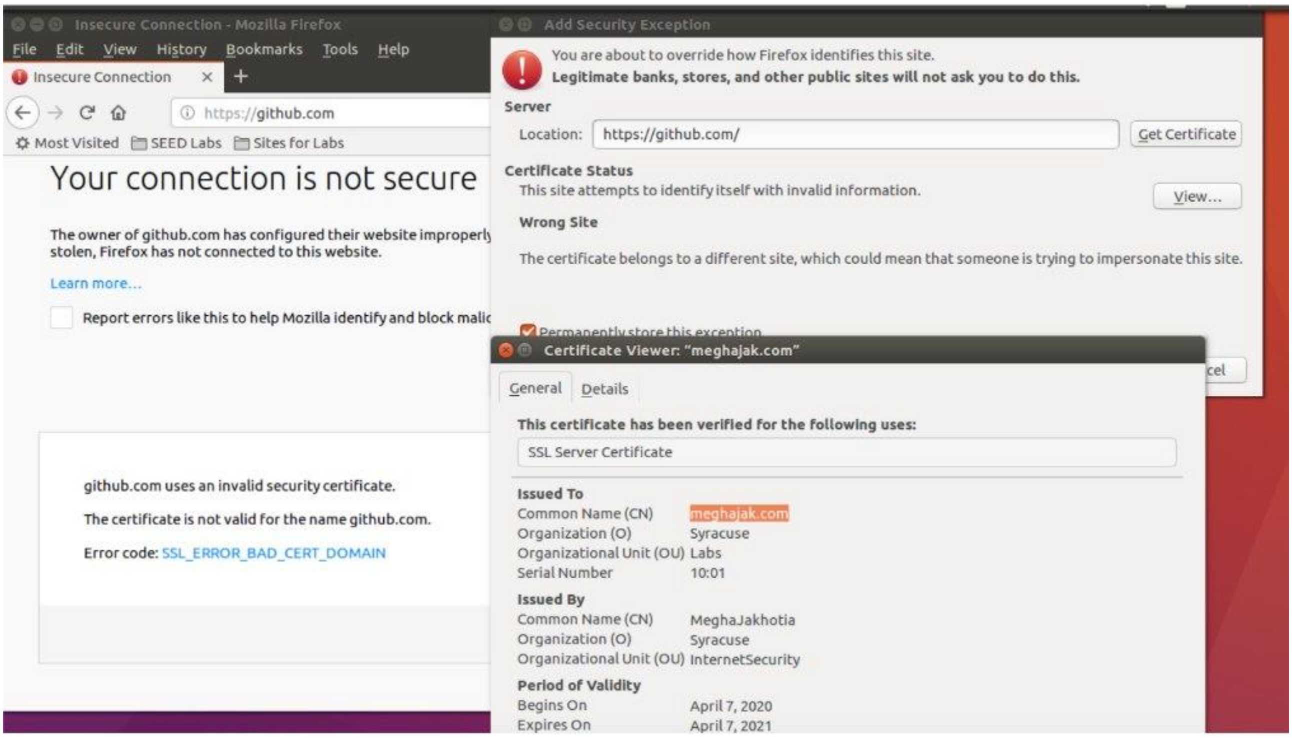
There are several ways to get the user's HTTPS request to land
in our web server. One way is to attack the routing, so the user's HTTPS request is routed to our web server. Another way is to attack DNS, so when the victim's machine tries to fifind out the IP address of the target web server, it gets the IP address of our web server. In this task, we use "attack" DNS. Instead of launching an actual DNS cache poisoning attack, we simply modify the victim's machine's /etc/hosts fifile to emulate the result of a DNS cache positing attack (the IP Address in the following should be replaced by the actual IP address of the malicious server). <IP_Address> example.com

```
abe@abe-VirtualBox:/var/www/SEEDPKILAB2020$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       abe-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Step 3: Browse the target website.

With everything set up, now visit the target real website, and see what your browser would say. Please explain what you have observed.

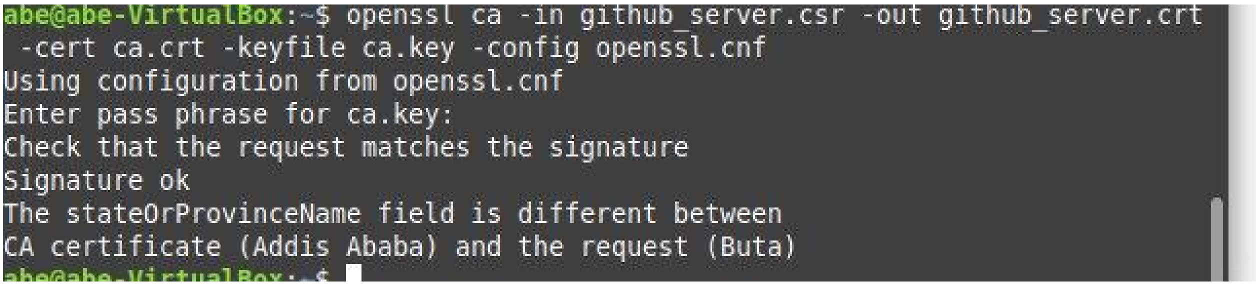


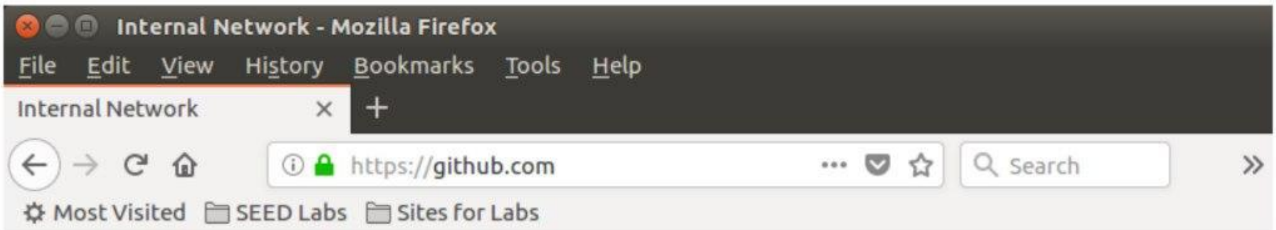
2.6 Task 6: Launching a Man-In-The-Middle Attack with a Compromised CA

Unfortunately, the root CA that we created in Task 1 is compromised by an attacker, and its private key is stolen. Therefore, the attacker can generate any arbitrary certifificate using this CA's private key. In this task, we will see the consequence of such a compromise. Please design an experiment to show that the attacker can successfully launch MITM attacks on any HTTPS website. You can use the same setting created in Task 5, but this time, you need to demonstrate that the MITM attack is successful, i.e., the browser will not raise any suspicion when the victim tries to visit a website but land in the MITM attacker's fake website.

```
abe@abe-VirtualBox:~$ openssl genrsa -aes128 -out github_server.key 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....................................+++++
..................+++++
e is 65537 (0x010001)
Enter pass phrase for github_server.key:
Verifying - Enter pass phrase for github_server.key:
abe@abe-VirtualBox:~$
```


```
abe@abe-VirtualBox:~$ openssl genrsa -aes128 -out github_server.key 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....................................+++++
..................+++++
e is 65537 (0x010001)
Enter pass phrase for github_server.key:
Verifying - Enter pass phrase for github_server.key:
abe@abe-VirtualBox:~$
```





This because the certificate is valid due to being signed by root CA and its common name is github.com. If the root CA's key is compromised, then anyone can create a certificate of themselves and impersonate any other website.