CALIFORNIA STATE UNIVERSITY
LONG BEACH

Lab #3

CECS 378 – Fall 2021 – Cappel

Due: Monday, September 27th by 11:59 PM

This lab will leverage one of the virtual machines you built in Lab #1. This lab is based on the Public-Key Infrastructure (PKI) Lab written by Dr. Wenliang Du, Syracuse University.

We will be using the following VM in Lab #3: SeedUbuntu1 VM

Public Key Infrastructure (PKI)

Public key cryptography is the foundation of today's secure communication, but it is subject to Man-in-the-Middle (MitM) 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 MitM 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
- MitM attacks

This lab will be performed on the SEEDUbuntu1 virtual machine and will use O*penSSL* commands and libraries that have already been installed on the virtual machine.

Task 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 CA's are treated as root CA's; IdenTrust is the largest CA at the time of this writing. Users who want to get digital certificates issued by the commercial CA's need to pay those CA's.

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 certificates 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

normally signed by another CA, the root CA's certificates are self-signed. Commercial root CA certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on PKI. Root CA certificates are unconditionally trusted.

The Configuration File openssl.cnf. In order to use *OpenSSL* to create certificates, you must have a configuration file. In Linux, configuration files normally have an extension of *.cnf*. The *openssl.cnf* configuration file is used by three *OpenSSL* commands: *ca*, *req* and *x509*. We will create a copy of the default configuration file located at */usr/lib/ssl/openssl.cnf*. After copying this file into the proper directory, we will create several sub-directories 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, we will simply create an empty file. For the *serial* file, we will simply place a single number in string format (e.g., 1000) in the file. Once we have set up the configuration file *openssl.cnf*, we can create and issue certificates.

1. Turn on your SeedUbuntu1 virtual machine in VirtualBox.

   Note: You will only require the SeedUbuntu1 virtual machine in this lab.

2. Open the Terminator (command line) program.
3. We will be using the /home/seed/Desktop directory as our primary directory for this lab.

   Note: You can leverage the pwd command to "print working directory" in case you lose track of your current command line focus. All the openssl commands you will execute must be done from the /home/seed/Desktop directory.

4. Execute the following to make a backup copy of the openssl.cnf file & place it in the primary directory:

   cp -b /usr/lib/ssl/openssl.cnf /home/seed/Desktop

5. Execute the following command to move focus to the Desktop directory:

   cd /home/seed/Desktop

6. Execute the following command to create the demoCA directory:

   mkdir demoCA

7. Execute the following command to shift focus to the demoCA directory:

   cd demoCA

8. Execute the following three commands to create the 3 sub-directories:

   mkdir certs

   mkdir crl

mkdir newcerts

9. Execute the following commands to create two empty files:

   touch index.txt

   touch serial

10. Open the *serial* file using the following command:

    sudo gedit serial

11. Enter the number "1000" on the first line and save the *serial* file by selecting save in the upper right corner. Close the file by selection the orange x in the upper left corner.
12. Execute the following command to shift focus back to the Desktop directory:

    cd ..

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.

13. Execute 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

    Note: You will be prompted for the PEM pass phrase. 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 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.

    Enter the following information when prompted:

    Country Name: **US**

    State or Province Name: **California**

    Locality Name: **Long Beach**

    Organization Name: **CSULB**

    Organizational Unit Name: **CECS**

    Common Name: **csulb.edu**

    Email Address: **postmaster@csulb.edu**

    Note: You should see the *ca.key* and *ca.crt* files appear on your desktop.

The ca.key is an encoded file (also encrypted), so you will not be able to see the actual content, such as the modulus, private exponents, etc.

This is the private key for the CA and should be heavily protected.

14. Execute the following command which will allow you to see the contents of the file:

    openssl rsa -in ca.key -text

    Note: Prime1 is p, Prime 2 is q, Modulus is N, Public exponent is e, and the Private exponent is d.

Screen Shot 1) Right-click on the *ca.crt* file and Open with View file. Expand the details section and take a snip (screen shot) of the contents of the file and then close the file.

Question 1) How long is the *ca.crt* valid? What RSA key size was used for the Public key found in *ca.crt*?

Task 2 – Creating a Certificate for SEEDPKILab2020.com

Now that we have become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called *SEEDPKILab2020.com*. For this company to be able to receive a digital certificate from a CA, it needs to go through these three steps.

Step 1: Generate a public/private key pair. The company needs to first create its own public/private key pair. We can run a 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*.

15. Execute the following command to generate the RSA key pair:

    openssl genrsa -aes128 -out server.key 1024

    Note: You should see the *server.key* file appear on your desktop.

The server.key is an encoded file (also encrypted), so you will not be able to see the actual content, such as the modulus, private exponents, etc.

16. Execute the following command which will allow you to see the contents of the file:

    openssl rsa -in server.key -text

Question 2) How many bits is the private key generated in the *server.key*?

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 information in 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 identify). Please use *SEEDPKILab2020.com* as the common name of the certificate request.

17. Execute the following command which will generate the CSR:

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

Enter the following information when prompted:

Country Name: **US**

State or Province Name: **California**

Locality Name: **Long Beach**

Organization Name: **SEEDLabs Inc.**

Organizational Unit Name: Leave Blank

Common Name: **SEEDPKILab2020.com**

Email Address: Leave Blank

Note: You will need to enter a challenge password and you can leave the optional company name blank. You will then see the *server.csr* file appear on your desktop.

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. We will execute a command that turns the certificate signing request (*server.csr*) into an X.509 certificate (*server.crt*), using the CA's *ca.crt* and *ca.key*.

18. Execute the following command which will generate the certificate:

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

You probably were not able to generate the certificate and received the following message:

The organizationName field needed to be the same in the CA certificate (CSULB) & the request (SEEDLabs Inc.)

If *openssl* refuses to generate the certificate, it is very likely that the names in your requests do not match with those of the CA. The matching rules are specified in the configuration file (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 configuration file also includes another policy (called *policy_anything*), which is less restrictive.

19. Edit the openssl.cnf file by executing the following command:

    sudo gedit /home/seed/Desktop/openssl.cnf

20. Change the matching policy by modifying the following content in the openssl.cnf file:

    "*policy = policy_match*" change to "*policy = policy_anything*"

21. Save the changes to the openssl.cnf file.
22. Re-run the following command which will generate the certificate:

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

Note: You will need to sign the certificate and commit the certificate. You should then see the *server.crt* file appear on your desktop.

Question 3) Who verified the *server.crt* certificate? How long is the *server.crt* valid?

Task 3 – Deploying a Certificate in an HTTPS Web Server

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

Step 1: Configuring DNS. We choose *SEEDPKILab2020.com* as the name of our website. To get our computers to 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) IP address. Your computer will leverage the local hosts file by default for domain name resolution if it is not configured to use a DNS Server.

1. Execute the following command to add the SEEDPKILab2020.com entry to the hosts file:

   sudo gedit /etc/hosts

2. Add the following line in the hosts file and then save the file:

   127.0.0.1 SEEDPKILab2020.com

3. Execute the following command to ensure the website responds with the proper address (127.0.0.1):

   ping SEEDPKILab2020.com

Step 2: Configuring the web server. 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.

4. Execute the following commands to first combine the secret key and certificate into one file:

   cd /home/seed/Desktop

   cp server.key server.pem

   cat server.crt >> server.pem

   Note: You should see the *server.pem* file appear on your desktop.

5. Execute the following command to launch the web server using the *server.pem* file:

   openssl s_server -cert server.pem -www

   Note: You will need to provide the passphrase used for the *server.key* file. Once the web server starts you will see "using default temp DH parameters and ACCEPT". The web server is running at this point.

By default, the web server will listen on port 4433. You can alter that using the *-accept* option. We will now try accessing the web server to see what happens.

6. Open the Firefox web browser on the SEEDUbuntu1 virtual machine.

7. Attempt to access the following website by entering this URL in the text box at the top of the browser:

   https://SEEDPKILab2020.com:4433

   Note: Most likely, you will receive an error message from the browser. In Firefox, you will see a message like the following: "**Your connection is not secure.** The owner of seedpkilab2020.com has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website."

Step 3: Getting the browser to accept our CA certificate. Had our certificate been assigned by VeriSign or some other commercial CA, we will not receive such an error message, because VeriSign's certificate is highly likely already preloaded into Firefox's certificate repository. 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 commercial CA's, 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 option.
- We can manually load our CA's certificate into Firefox.

8. Select the Edit menu within the Firefox browser.
9. Select Preferences, Privacy & Security, and View Certificates.

You will see a list of certificates that are already accepted by Firefox. We will now import our certificate and tell Firefox to trust this CA to identify web sites.

10. Select the Import option in the Manage Certificates window.
11. Select the Desktop folder on the left, select the *ca.crt* file, and select Open.
12. Select the Trust this CA to identify web sites option, select OK, and then OK again to close the certificate manager window.

Step 4: Testing our HTTPS website. Now, point the browser at the HTTPS website.

13. Enter the *https://SEEDPKILab2020.com:4433* web site in the web browser URL text box.

    Note: You no longer receive the error message. If the web site loaded, you will see the following:

```
s_server -cert server.pem -www
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1/SSLv3:ECDHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:ECDHE-ECDSA-AES256-GCM-SHA384
And many additional ciphers…
```

14. Stop the web server using the Ctrl^C command.
15. Make a copy of the server.pem file using the following command:

    cp server.pem server.bak

16. Modify the server.pem file by changing one value in the modulus section of the file then save the file.

sudo gedit server.pem

17. Restart the web server using the following command:

    openssl s_server -cert server.pem -www

18. Stop the web server using the Ctrl^C command.
19. Modify the server.pem file by changing one value in the certificate section of the file then save the file.

    sudo gedit server.pem

20. Restart the web server using the following command:

    openssl s_server -cert server.pem -www

Question 4) Why do you think the web server failed to start when changing the certificate information but not the modulus information?

21. Restore the original server.pem file using the following command:

    cp server.bak server.pem

22. Delete the server.bak file by executing the following command:

    rm server.bak

Since *SEEDPKILab2020.com* points to the localhost, if we use *https://localhost:4433* instead, we will be connecting to the same web server. Now let us try accessing that URL to see if we are successful.

23. Enter the *https://localhost:4433* web site in the Firefox web browser.

Question 5) Did your browser allow you to connect to the website? Why or why not?

Task 4 – Deploying a Certificate in an Apache-Based HTTPS Website

The HTTPS server setup using openssl's *s_server* command we used in the previous task is primarily for debugging and demonstration purposes. In this task, we setup a real HTTPS web server based on Apache. The Apache server, which is already installed in our VM, supports the HTTPS protocol. We will use the certificate generated in the previous task to establish the SEEDPKILab2020.com website running on the Apache server.

Step 1: Configure the Apache web server. 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 HTTPS website, we need to add a *VirtualHost* entry to the *default-ssl.conf* file in this directory.

1. Execute the following commands to backup and then modify the Apache configuration file:

cd /etc/apache2/sites-available

sudo cp default-ssl.conf default-ssl.orig

sudo gedit default-ssl.conf

2. Edit the file so it only contains the following lines:

```
<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerName  SEEDPKILab2020.com
                DocumentRoot /var/www/SEEDPKILab2020
                DirectoryIndex index.html
                SSLEngine on
                SSLCertificateFile      /home/seed/Desktop/server.crt
                SSLCertificateKeyFile /home/seed/Desktop/server.key
        </VirtualHost>
</IfModule>
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Note: The ServerName entry specifies the name of the website, while the DocumentHost entry specifies where the files for the website are stored. The above configuration sets up the HTTPS site *https://SEEDPKILab2020.com* (port 443 is the default HTTPS port for Apache). In the configuration, we also need to tell the Apache service where the server certificate and private key are stored.

3. Execute the following commands to create the directory containing the website contents:

sudo mkdir /var/www/SEEDPKILab2020

cd /var/www/SEEDPKILab2020

sudo touch test.txt

sudo gedit test.txt

(Enter some random text in this test text file and save the file)

Step 2: Enable SSL & Start Apache Service. After the *default-ssl.conf* file is modified, we need to run a series of commands to enable SSL. Apache will ask us to type the password used for encrypting the private key (*server.key*). Once everything is setup properly, we can browse the web site, and all the traffic between the browser and the server will be encrypted.

4. Enter the following command to test the apache configuration file for errors:

sudo apachectl configtest

5. Enter the following command to enable the SSL module:

sudo a2enmod ssl

6. Enter the following command to enable the site we have just edited:

   sudo a2ensite default-ssl

7. Enter the following command to restart the apache we service:

   sudo service apache2 restart

   Note: If all is well, you will be prompted for the passphrase for the *server.key.*

8. Open the Firefox web browser on the SEEDUbuntu1 virtual machine.
9. Attempt to access the following website by entering this URL in the text box at the top of the browser:

   https://SEEDPKILab2020.com

   Note: If your Apache website loads, you will see the test.txt file you created earlier.

Task 5 – Launching a Man-in-the-Middle (MitM) Attack

In this task, we will show how PKI can defeat 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 the 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 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.

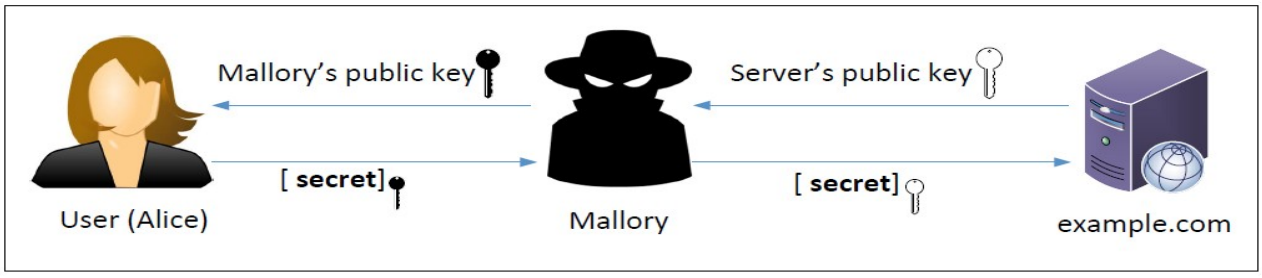

Figure 1: A Man-In-The-Middle (MITM) attack

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 first. In this lab we will use example.com as the target website.

Step 1: Setting up the malicious website. In Task 4, we have already setup an HTTPS website for *SEEDPKILab2020.com*. We will use the same Apache server to impersonate *example.com*. To achieve that, we

will follow the instruction in Task 4 to add a VirtualHost entry to Apache's SSL configuration file: the ServerName should be set to example.com, but the rest of the configuration 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 an actual social network website, the fake site can display a login page like 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.

1. Execute the following commands to backup and then modify the Apache configuration file:

   cd /etc/apache2/sites-available

   sudo cp default-ssl.conf default-ssl.bak

   sudo gedit default-ssl.conf

2. Edit the file and change the ServerName field only:

   "ServerName SEEDPKILab2020.com" change to "ServerName example.com"

Step 2: Becoming the man in the middle. There are several ways to get the user's HTTPS request to land on 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 find 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 /etc/hosts file to emulate the result of a DNS cache poisoning attack (the IP_Address in the following should be replaced by the actual IP address of the malicious server).

1. Execute the following commands to add the example.com entry to the hosts file:

   sudo gedit /etc/hosts

2. Add the following line in the hosts file and then save the file:

   127.0.0.1 example.com

3. Attempt to access the malicious website by entering this URL in the text box at the top of the browser:

   https://example.com

Question 6) Did your browser allow you to connect to the website? Why or why not?

Task 6 – Launching a Man-in-the-Middle (MitM) 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 certificate using this CA's private key. In this task, we will see the consequences of such a compromise.

Step 1: Generate a public/private key pair. The attacker needs to first create the public/private key pair which they can do on any device.

1. Execute the following command to generate the RSA key pair:

   cd /home/seed/Desktop

   openssl genrsa -aes128 -out example.key 1024

   Note: You should see the *example.key* file appear on your desktop.

Step 2: Generate a Certificate Signing Request (CSR). Once the attacker has the key file, a Certificate Signing Request (CSR) can be generated, which basically includes the websites public key.

3. Execute the following command which will generate the CSR:

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

   Enter the following information when prompted:

   Country Name: **US**

   State or Province Name: **California**

   Locality Name: **Long Beach**

   Organization Name: **Example Inc.**

   Organizational Unit Name: Leave Blank

   Common Name: **example.com**

   Email Address: Leave Blank

   Note: You will need to enter a challenge password and you can leave the optional company name blank. You should then see the *example.csr* file appear on your desktop.

Step 3: Generate the Certificate. The CSR file needs to have the CA's signature to form a certificate.

4. Execute the following commands to create the certificate:

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

   Note: You will need to sign the certificate and commit the certificate. You should then see the *example.crt* file appear on your desktop.

Step 4: Configure the malicious web server. Modify the Apache configuration file to leverage the new certificate and key files.

1. Execute the following commands to backup and then modify the Apache configuration file:

   cd /etc/apache2/sites-available

sudo cp default-ssl.conf default-ssl.bkp

sudo gedit default-ssl.conf

2. Edit the file and change the SSLCertificateFile and SSLCertificateKeyFile fields:

   Change to: *SSLCertificateFile /home/seed/Desktop/example.crt*

   Change to: *SSLCertificateKeyFile /home/seed/Desktop/example.key*

3. Enter the following command to restart the apache we service:

   sudo service apache2 restart

   Note: If all is well, you will be prompted for the passphrase for the *example.key.*

4. Attempt to access the malicious website by entering this URL in the text box at the top of the browser:

   https://example.com

Question 7) How important is it to protect the private key of your CA? Explain…

5. Enter the following command to examine the contents of the index.txt file for the CA:

   sudo gedit /home/seed/Desktop/demoCA/index.txt

Question 8) What information is being stored in the index.txt file?

Screen Shot 2) Take a snip (screen shot) of your SEEDUbuntu1 desktop showing *all* the files and folders you created during this lab. **Hint:** Your screen shot should have 1 folder and 10 files.