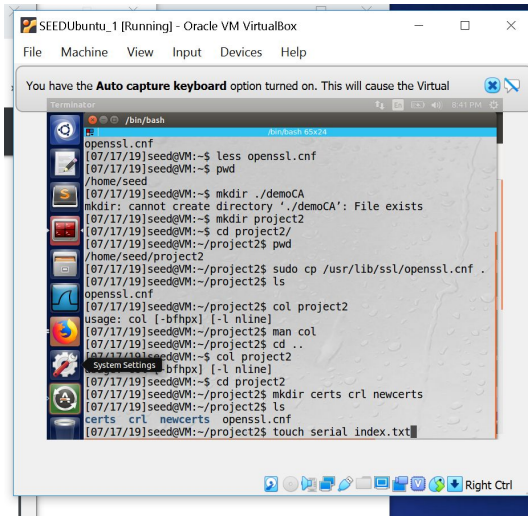
Rose Reiner and Vlad Smirnov
Introduction to Computer Security
7/19/19

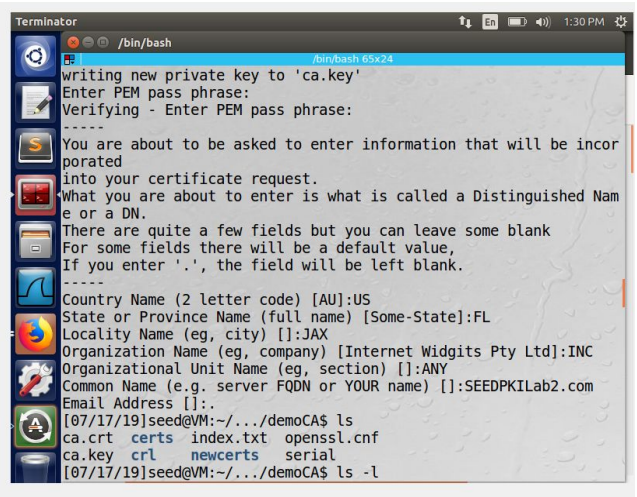
Public Key Infrastructure

Introduction:

In the Public Key Infrastructure lab, we became a root CA which is a trusted certificate authority that is able to sign certificates for other companies or websites. This lab demonstrates what a root CA is and how to sign a certificate for a company. It also shows us how to use our certificate for a web server when the browser doesn't recognize it yet. Lastly, we also see how a Man-in-the-middle attack works.

Task 1: Becoming a Certificate Authority





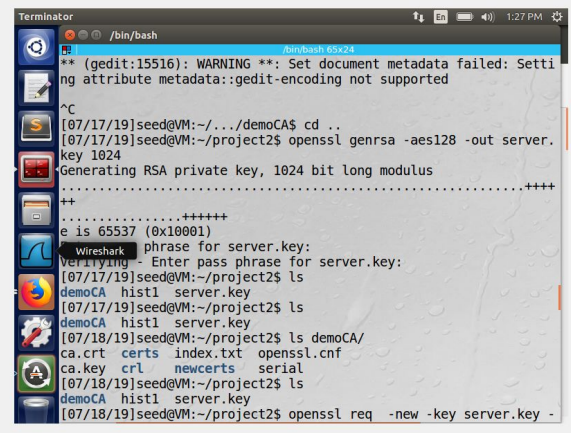
In Task 1, our goal was to become a root certificate authority (CA) in order for us to issue certificates and self-sign our own certificate. To create the certificate we need to use OpenSSL and create a configuration file. In the terminal, we made a directory to store the configuration file to use OpenSSL. Then we made subdirectories to store information such as issued certificates and new certificates. After we completed these steps, we typed in the terminal openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf to make our certificate a self-signed one. It asked us to create a password and fill in some information that was then stored in the files we created earlier: the private key (ca.key) and public key (ca.crt).

Task 2: Creating a Certificate for SEEDPKILab2018.com

In Task 2, we sign certificates to companies since we became a root CA in the previous task.

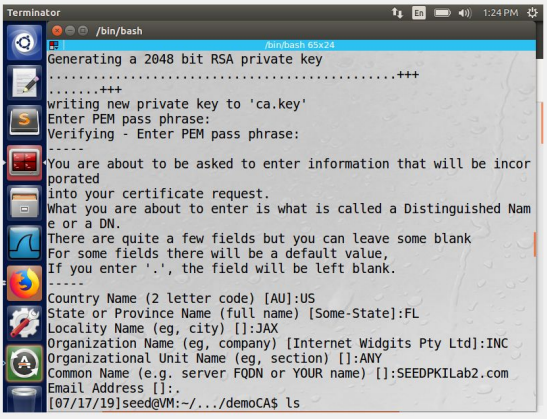
Step 1:

The first step to creating a certificate for SEEDPKILab2018.com is creating a public and private key for the company. We use the command openssl genrsa -aes128 -out server.key 1024 which prompts us with a password in order to encrypt the key. This command generates an RSA private key, encrypts it using AES-128, and stores it in the encoded text file server.key.



Step 2:

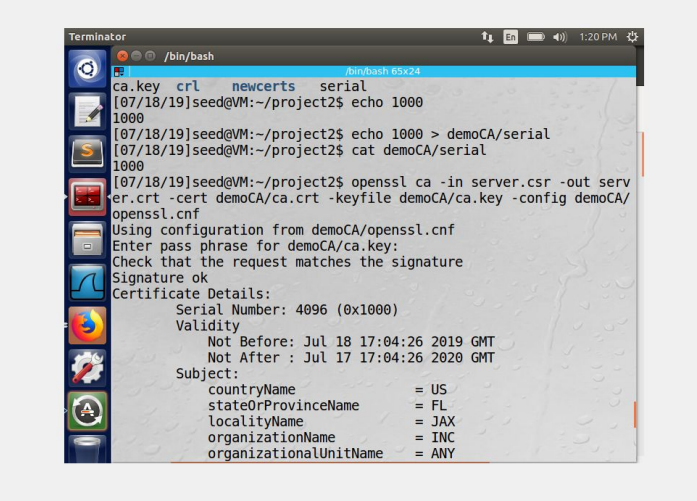
The second step is to generate a certificate signing request (CSR). This is done to get the certificate for the private and public key of the company. We used the command openssl req - new -key server.key - out server.csr -config openssl.cnf which just requests the CSR.

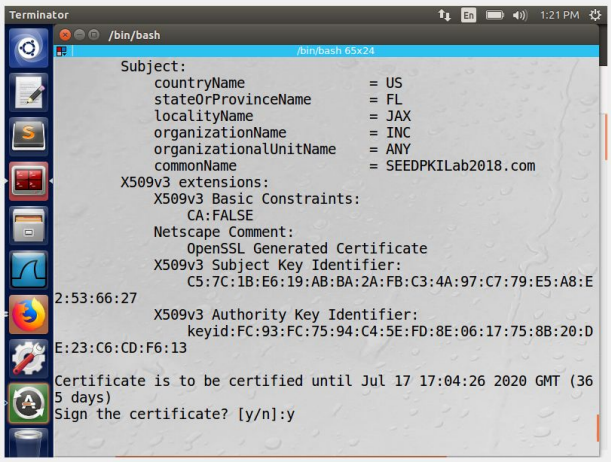


Step 3:

The third step is to generate the certificate. We opened the configuration file to change the matching rules to policy_match in order for the request file of the company's to match our CA.

We used the command openssl ca -in server.csr -out server.crt -cert demoCA/ca.cert -keyfile demoCA/ca.key -config demoCA/openssl.cnf which makes server.csr to server.crt by using our own public and private keys. Then it checks if the request server.csr matches our signature and prompts us to sign the certificate.





Task 3: Deploying Certificate in an HTTPS Web Server

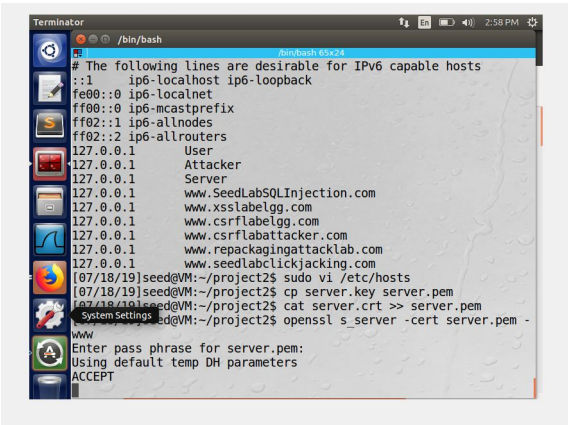
Task 3 demonstrates how CA's can sign a certificate to an HTTPS web server for security.

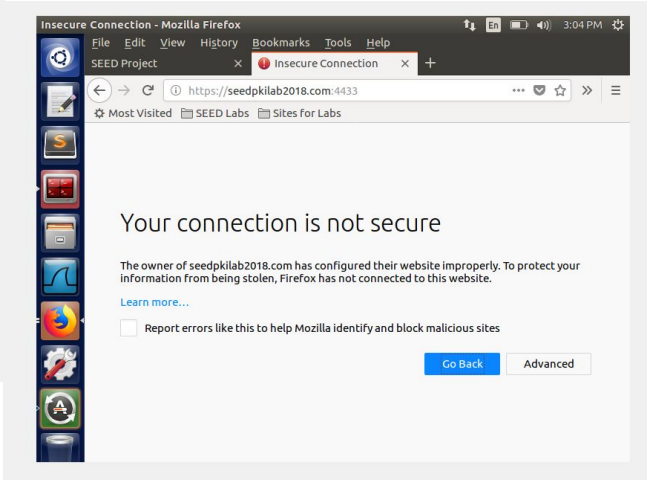
Step 1:

```
[07/18/19]seed@VM:~/project2$ sudo gedit /etc/hosts
[sudo] password for seed:
```

In step one we needed to make the computer recognize the name of our website which we named SEEDPKILab2018.com , which we want to add to the follow /etc/hosts so it will go to the localhost.

Step 2:

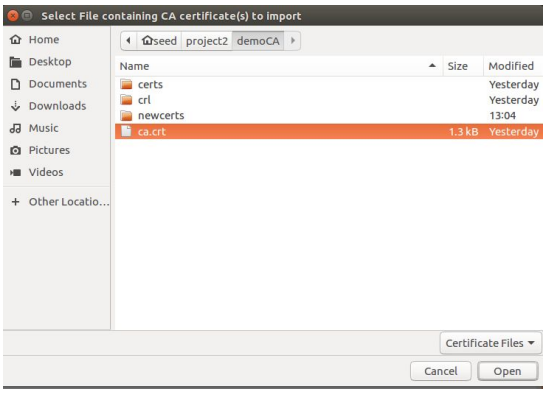


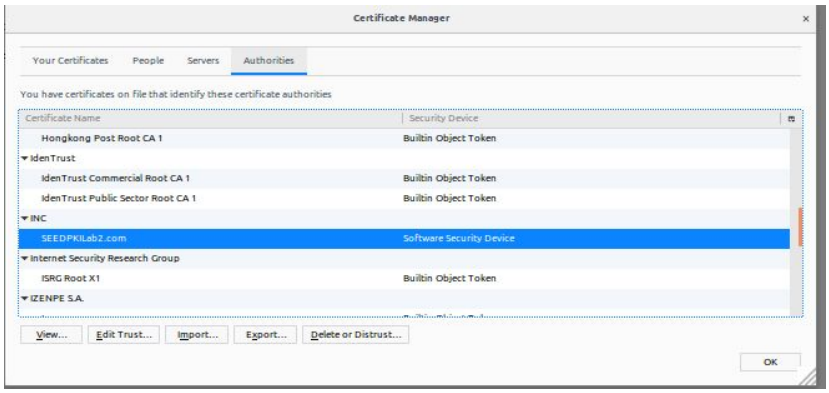


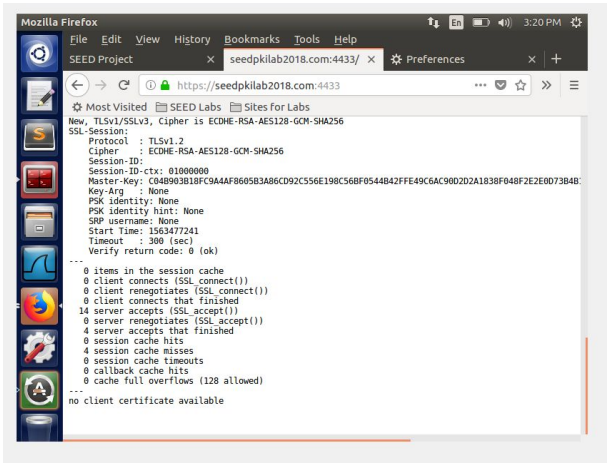
For step 2 we want to configure the web server. By using the openssl s_server -cert server.pem -www command to launch a web server with the certificate we made. Then we can go to the website https://seedpkilab2018.com:4433 and find that it says our connection is not secure. Therefore, shows that it hasn't accepted our certificate yet.

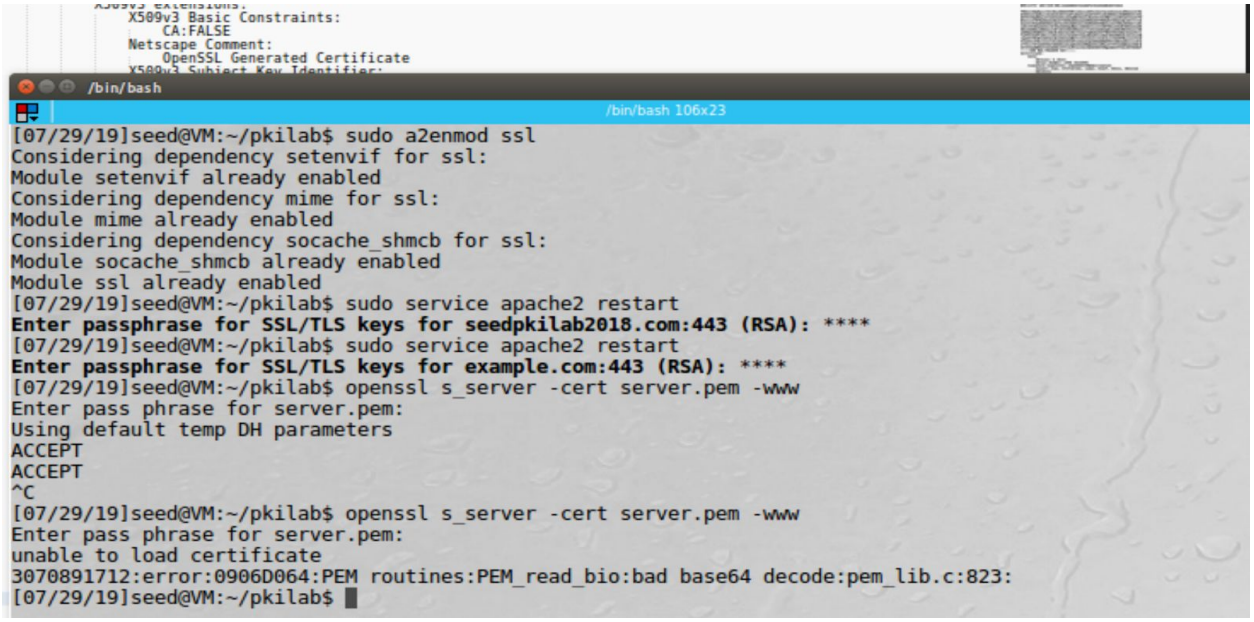
Step 3:

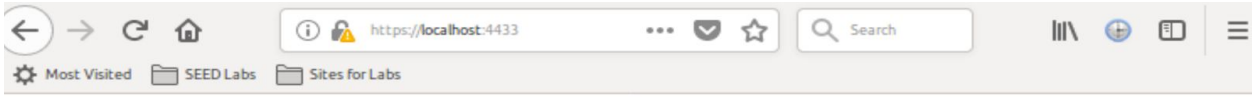
In the final step, we need the web browser to accept our certificate because our own CA is not recognizable to it and that's why we generate a security error when we try to go to the webpage. To do this we have to add our certificate (ca.crt) to the browser.






Mozilla Firefox
3:20 PM
File Edit View History Bookmarks Tools Help
SEED Project
seedpkilab2018.com:4433/
Preferences
https://seedpkilab2018.com:4433
Most Visited
SEED Labs
Sites for Labs
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
SSL-Session:
Protocol : TLSv1.2
Cipher : ECDHE-RSA-AES128-GCM-SHA256
Session-ID:
Session-ID-ctx: 01000000
Key-Arg : None
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1563477241
Timeout : 300 (sec)
Verify return code: 0 (ok)
0 items in the session cache
0 client connects (SSL_connect())
0 client renegotiates (SSL_connect())
0 client connects that finished
14 server accepts (SSL_accept())
0 server renegotiates (SSL_accept())
4 server accepts that finished
0 session cache hits
4 session cache misses
0 session cache timeouts
0 callback cache hits
0 cache full overflows (128 allowed)
no client certificate available


X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
/bin/bash
/bin/bash 106x23
[07/29/19]seed@VM:~/pkilab$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
[07/29/19]seed@VM:~/pkilab$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for seedpkilab2018.com:443 (RSA): ****
[07/29/19]seed@VM:~/pkilab$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for example.com:443 (RSA): ****
[07/29/19]seed@VM:~/pkilab$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
ACCEPT
^C
[07/29/19]seed@VM:~/pkilab$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
unable to load certificate
3070891712:error:0906D064:PEM routines:PEM_read_bio:bad base64 decode:pem_lib.c:823:
[07/29/19]seed@VM:~/pkilab$




```
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
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA384   TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA      TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA
TLSv1/SSLv3:SRP-DSS-AES-256-CBC-SHA   TLSv1/SSLv3:SRP-RSA-AES-256-CBC-SHA
TLSv1/SSLv3:SRP-AES-256-CBC-SHA       TLSv1/SSLv3:DH-DSS-AES256-GCM-SHA384
TLSv1/SSLv3:DHE-DSS-AES256-GCM-SHA384TLSv1/SSLv3:DH-RSA-AES256-GCM-SHA384
TLSv1/SSLv3:DHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:DHE-RSA-AES256-SHA256
TLSv1/SSLv3:DHE-DSS-AES256-SHA256     TLSv1/SSLv3:DH-RSA-AES256-SHA256
TLSv1/SSLv3:DH-DSS-AES256-SHA256      TLSv1/SSLv3:DHE-RSA-AES256-SHA
TLSv1/SSLv3:DHE-DSS-AES256-SHA        TLSv1/SSLv3:DH-RSA-AES256-SHA
TLSv1/SSLv3:DH-DSS-AES256-SHA         TLSv1/SSLv3:DHE-RSA-CAMELLIA256-SHA
TLSv1/SSLv3:DHE-DSS-CAMELLIA256-SHA   TLSv1/SSLv3:DH-RSA-CAMELLIA256-SHA
TLSv1/SSLv3:DH-DSS-CAMELLIA256-SHA    TLSv1/SSLv3:ECDH-RSA-AES256-GCM-SHA384
TLSv1/SSLv3:ECDH-ECDSA-AES256-GCM-SHA384TLSv1/SSLv3:ECDH-RSA-AES256-SHA384
TLSv1/SSLv3:ECDH-ECDSA-AES256-SHA384  TLSv1/SSLv3:ECDH-RSA-AES256-SHA
TLSv1/SSLv3:ECDH-ECDSA-AES256-SHA     TLSv1/SSLv3:AES256-GCM-SHA384
TLSv1/SSLv3:AES256-SHA256             TLSv1/SSLv3:AES256-SHA
TLSv1/SSLv3:CAMELLIA256-SHA           TLSv1/SSLv3:PSK-AES256-CBC-SHA
TLSv1/SSLv3:ECDHE-RSA-AES128-GCM-SHA256TLSv1/SSLv3:ECDHE-ECDSA-AES128-GCM-SHA256
TLSv1/SSLv3:ECDHE-RSA-AES128-SHA256   TLSv1/SSLv3:ECDHE-ECDSA-AES128-SHA256
TLSv1/SSLv3:ECDHE-RSA-AES128-SHA      TLSv1/SSLv3:ECDHE-ECDSA-AES128-SHA
TLSv1/SSLv3:SRP-DSS-AES-128-CBC-SHA   TLSv1/SSLv3:SRP-RSA-AES-128-CBC-SHA
TLSv1/SSLv3:SRP-AES-128-CBC-SHA       TLSv1/SSLv3:DH-DSS-AES128-GCM-SHA256
TLSv1/SSLv3:DHE-DSS-AES128-GCM-SHA256TLSv1/SSLv3:DH-RSA-AES128-GCM-SHA256
TLSv1/SSLv3:DHE-RSA-AES128-GCM-SHA256TLSv1/SSLv3:DHE-RSA-AES128-SHA256
TLSv1/SSLv3:DHE-DSS-AES128-SHA256     TLSv1/SSLv3:DH-RSA-AES128-SHA256
TLSv1/SSLv3:DH-DSS-AES128-SHA256      TLSv1/SSLv3:DHE-RSA-AES128-SHA
TLSv1/SSLv3:DHE-DSS-AES128-SHA        TLSv1/SSLv3:DH-RSA-AES128-SHA
```

Step 4:

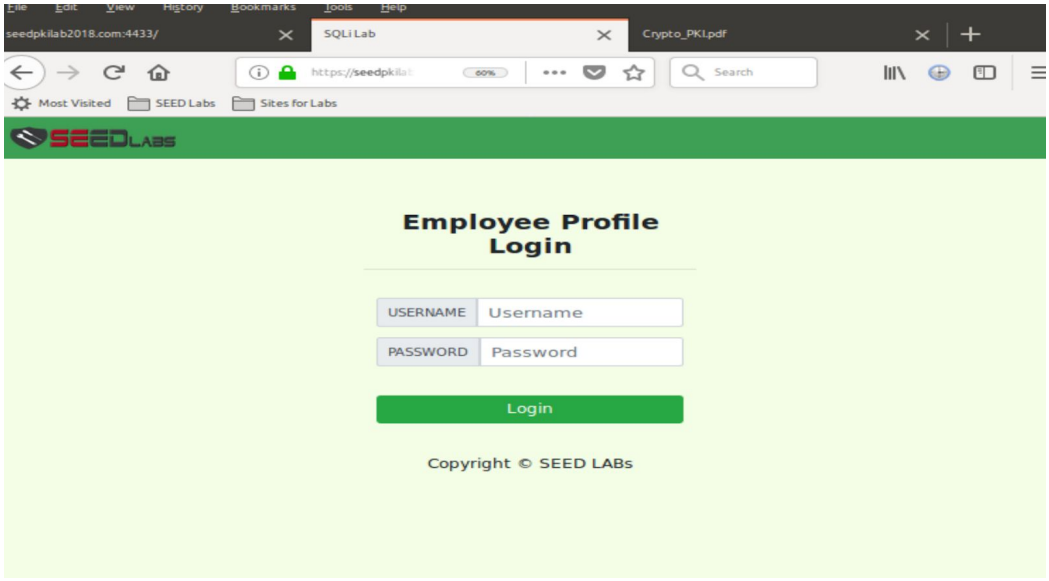
Lastly, we have to make sure FireFox accepted our certificate after we imported in manually. We should go back to the webpage https://SEEDPKILab2018.com:4433 here we get the certificate. If we modify a single byte of server.pem such as the key or the algorithm, the server will not be able to restart. However, anything besides that the server will be able to restart. If we go to the webpage http://localhost:4433 we get the same certificate because we directed that website to the localhost in a previous step.

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

For the apache server to host the website **seedpkilab2018.com** we need to add it into a virtual host in the 000-default.config folder. Apache server can host several websites, to host this one we had to create a virtual host. That is for a http website but for a https website where keys and certificates are used we need to edit the default-ssl-config file and add into the virtual host a new servername, document root, certificate and certificate key so it actually will be able to host the website. The certificate file paths point to the certificate location which we made in task 2.

When the ssl file changes need to be enabled and for that we had to run a series of commands. *Sudo apachect1 configtest* tested the config file for errors. *Sudo a2enmod ssl* enabled the ssl module. *Sudo a2ensite default-ssl* enabled the default file which enabled the site we created there. *Sudo service apache2 restart*, restarted the server after asking for the seed password and

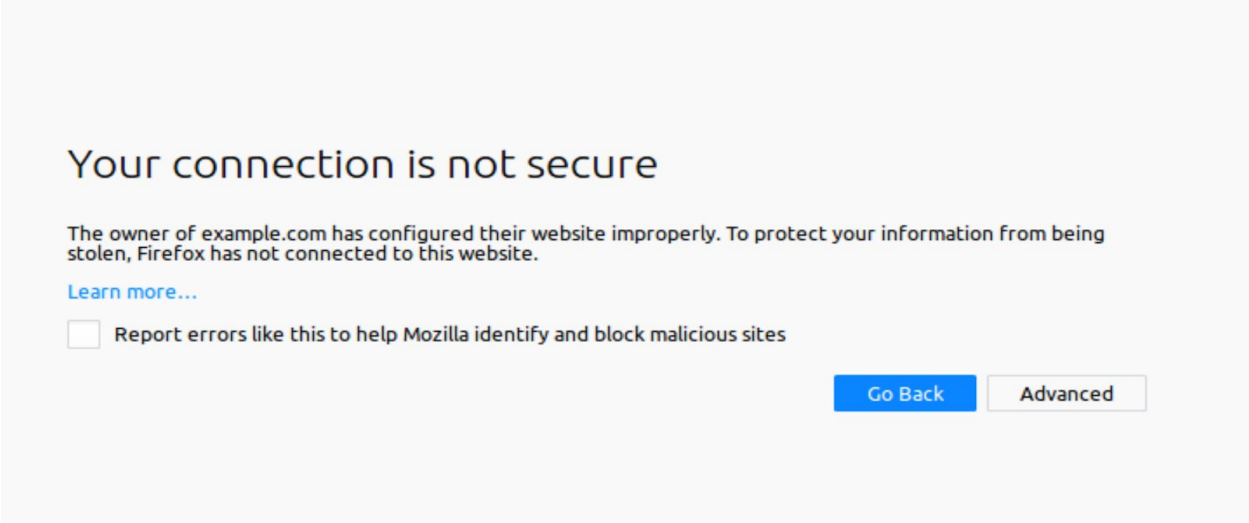
for the password on the certificate.



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

```
        the HTTP
        #   keep-alive facility, so you usually additionally want to
        disable
        #   keep-alive for those clients, too. Use variable "nokeepalive"
        for this.
        #   Similarly, one has to force some clients to use HTTP/1.0 to
        workaround
        #   their broken HTTP/1.1 implementation. Use variables
        "downgrade-1.0" and
        #   "force-response-1.0" for this.
        # BrowserMatch "MSIE [2-6]" \
        #       nokeepalive ssl-unclean-shutdown \
        #       downgrade-1.0 force-response-1.0


        ServerName seedpkilab2018.com
        DocumentRoot /var/www/SQLInjection
        DirectoryIndex index.html
        SSLEngine On

        SSLCertificateFile      /home/seed/pkilab/server.crt
        SSLCertificateKeyFile   /home/seed/pkilab/server.key


#this is the fake

        ServerName seedpkilab2018.com
        DocumentRoot /home/seed/pkilab/test
        DirectoryIndex index.html
        SSLEngine On

        SSLCertificateFile      /home/seed/pkilab/server.crt
        SSLCertificateKeyFile   /home/seed/pkilab/server.key

    </VirtualHost>
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

To impersonate seedpkilab2018.com we have to make another virtual host and we used example.com which will route the user to the seedlabpki2018.com. When the user goes to the website they will be redirected to the fake malicious website. After doing all of that and trying to go to the website what we end up getting is that the connection is not secure. This is because of certificate issues therefore the browser does not allow the connection because it sees it as malicious.



Conclusion:

Throughout this lab we learned how certificates are set up with the CA and the client. How to set up the CA. How to get the browser to accept other certificates. How to host a webpage using the certificate and key and also how to initiate a man in the middle attack. Step one is where CA makes certificate, server requests certificate from CA to make a certificate. The CA also makes a key for signing the certificate. Step two is to make a certificate for seedpkilab2018.com. Step three is to get the browser to recognize that certificate. Step 4 is to host a webpage on that webpage using the certificate for seedpki2018.com. Step 5 is to initiate a man in the middle attack but it wont work because the certificates don't work for the malicious website.