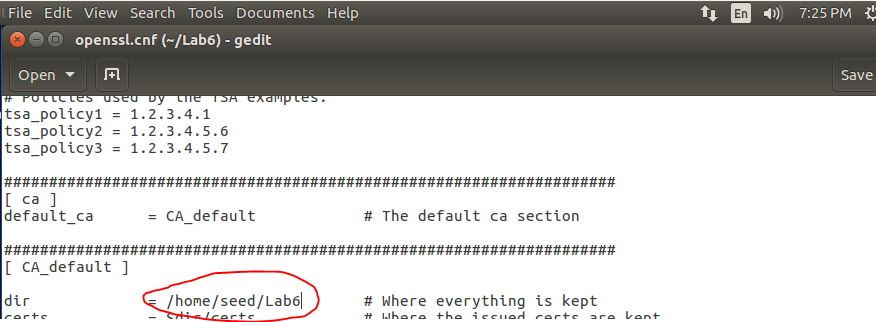
Lab 6 Report

**Task 1: Becoming a Certificate Authority (CA)**

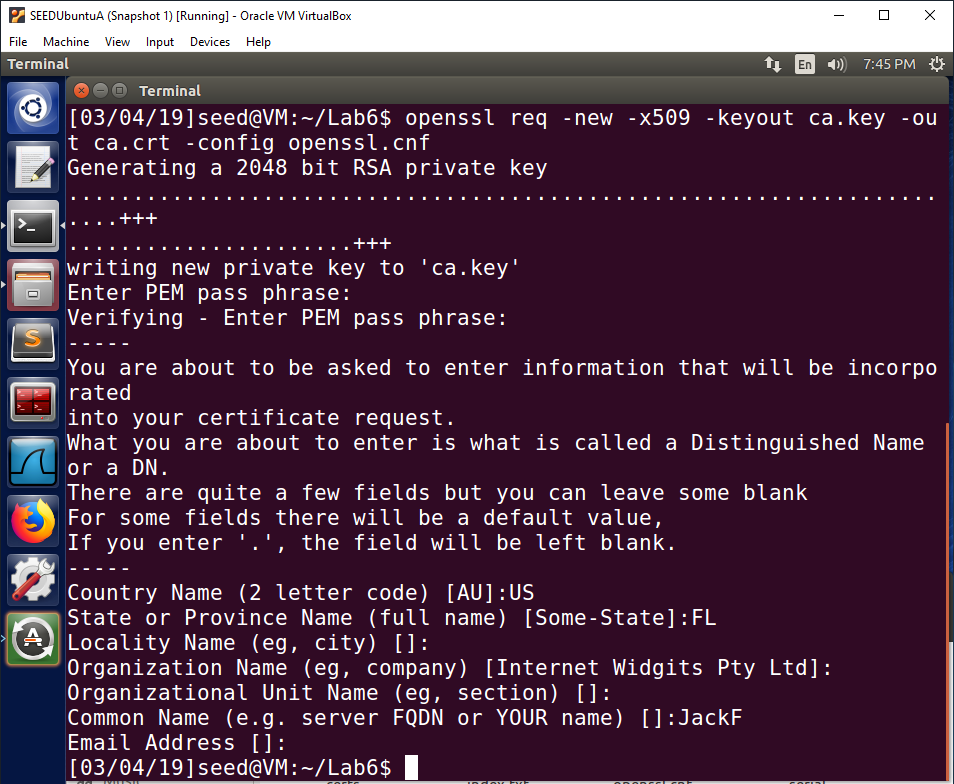


Observation:

The default directory for the certificate authority “openssl.cnf” file has been changed to “/home/seed/Lab6”.

Explanation:

I changed the default directory in the “openssl.cnf” configuration file to a folder I created specifically for this lab. This will allow the certificate generation process to look in this specific directory since the configuration file now specifies this directory as the default directory for finding specific files and directories that are needed. The “openssl.cnf” configuration file will be used to create certificates. This “openssl.cnf” file was copied from the “/usr/lib/ssl” directory already on the image.



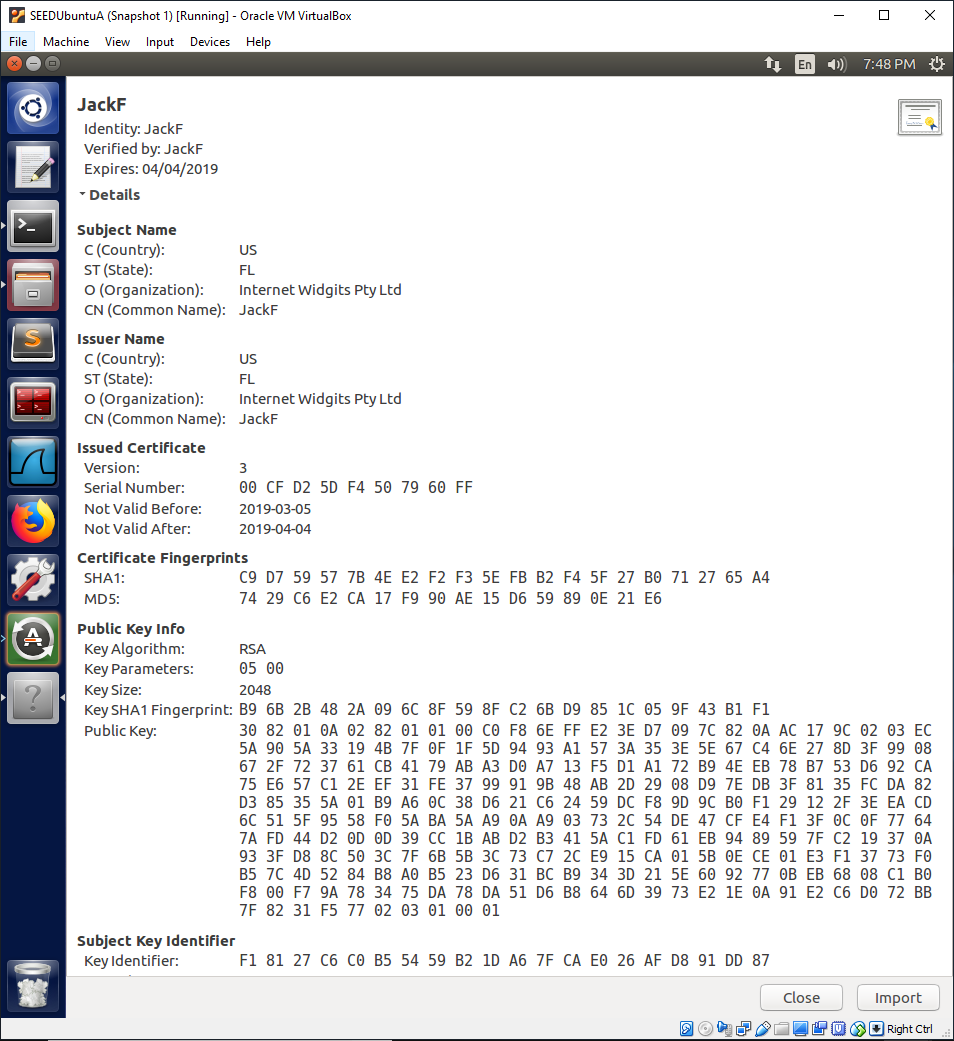
**Note: PEM pass phase is “dees”.**

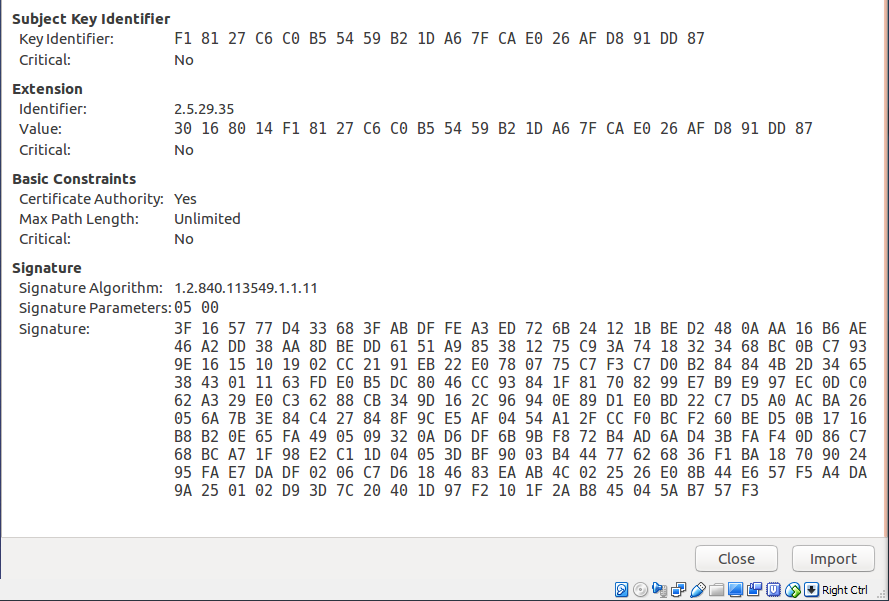
Observation:

A certificate request is generated using the openssl command. The command contains multiple options. The full command is “openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf”. A pass phrase is requested, which will be used to encrypt the private key. I have filled the country name, state, and common name field in with information and left the other fields blank.

Explanation:

The “-new” option is used to generate a new request. The “-x509” option is used to generate a self-signed certificate instead of a general certificate request. This is important since we want to generate a self-signed certificate to be used by us so we can act as a certificate authority. Root certificate authorities self-sign their own certificates since there is no organization that is above the root certificate authority. Root certificate authorities are at the highest level of trust. The “-keyout ca.key” option will output the private key that was generated to the “ca.key” file. The “ca.key” file contains the private key in an encrypted form. The “-out ca.crt” option will output the generated self-signed certificate to the “ca.crt” file. The “-config openssl.cnf” option tells the command to look at the “openssl.cnf” configuration file for any configuration settings that it needs during the process. This configuration file will give a target directory for where to find certain information as well as where to output the newly generated files.



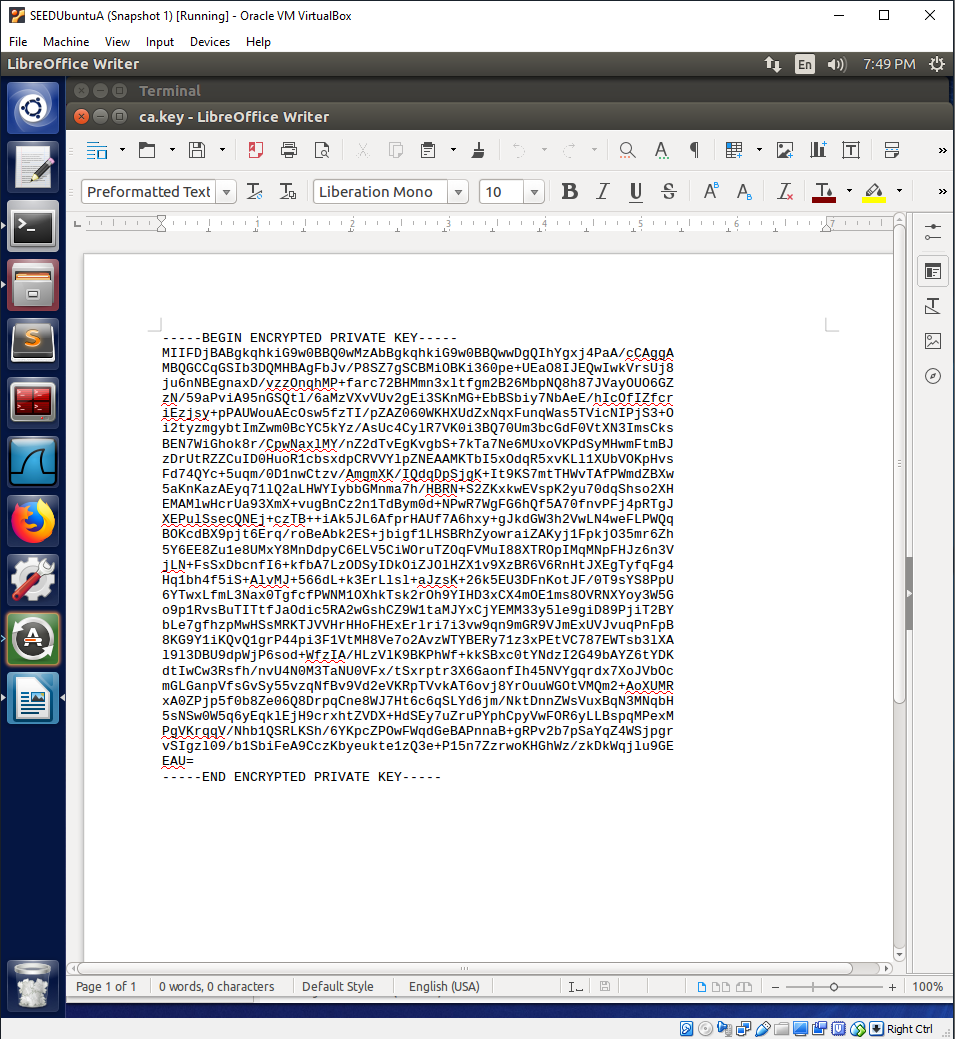


Observation:

The two images together are the entire self-signed certificate that was just generated. This certificate contains the information that was provided when prompted for information. The certificate contains important information such as common name and the certificate public key.

Explanation:

We have successfully generated a self-signed certificate to act as a root certificate authority. We will now be able to sign certificates for other entities to vouch for their credibility. A self-signed certificate is usually not trusted unless it is from a root certificate authority, which is what we are acting as during this lab. It is also important that the root certificate authority’s certificate is given to the user in a secure manner so that it is ensured to be trustworthy. Therefore, many operating systems and browsers are installed with a list of trusted certificates. In this case, we are generating the certificate on our own machine so we know it can be trusted. Since the certificate is trusted, the public key can be used by other entities to verify certificates that are signed by the root certificate authority.



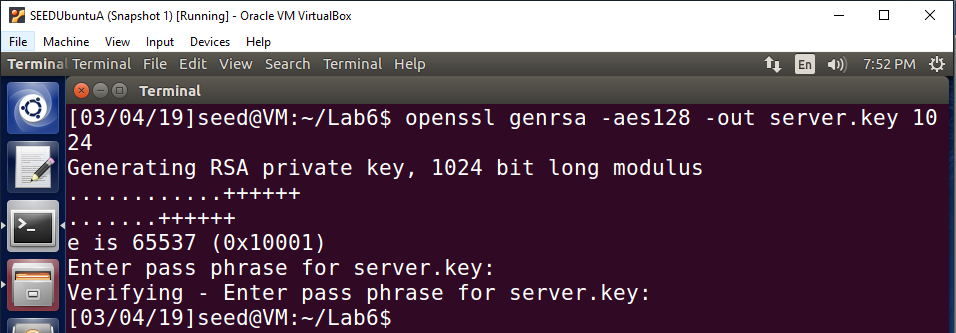
Observation:

This is the “ca.key” file, which contains the private key that has been encrypted.

Explanation:

This “ca.key” file contains our root certificate authority private key. This private key is encrypted for security. When we need to use the private key, we must provide the password to decrypt the private key so that it can be used. The unencrypted private key was used to self-sign our own root certificate authority certificate. This private key will also be used to sign other certificates when we are requested to do so. The public key in our root certificate authority certificate will be used to verify our signature on certificates we sign for other entities.

**Task 2: Creating a Certificate for SEEDPKILab2018.com**



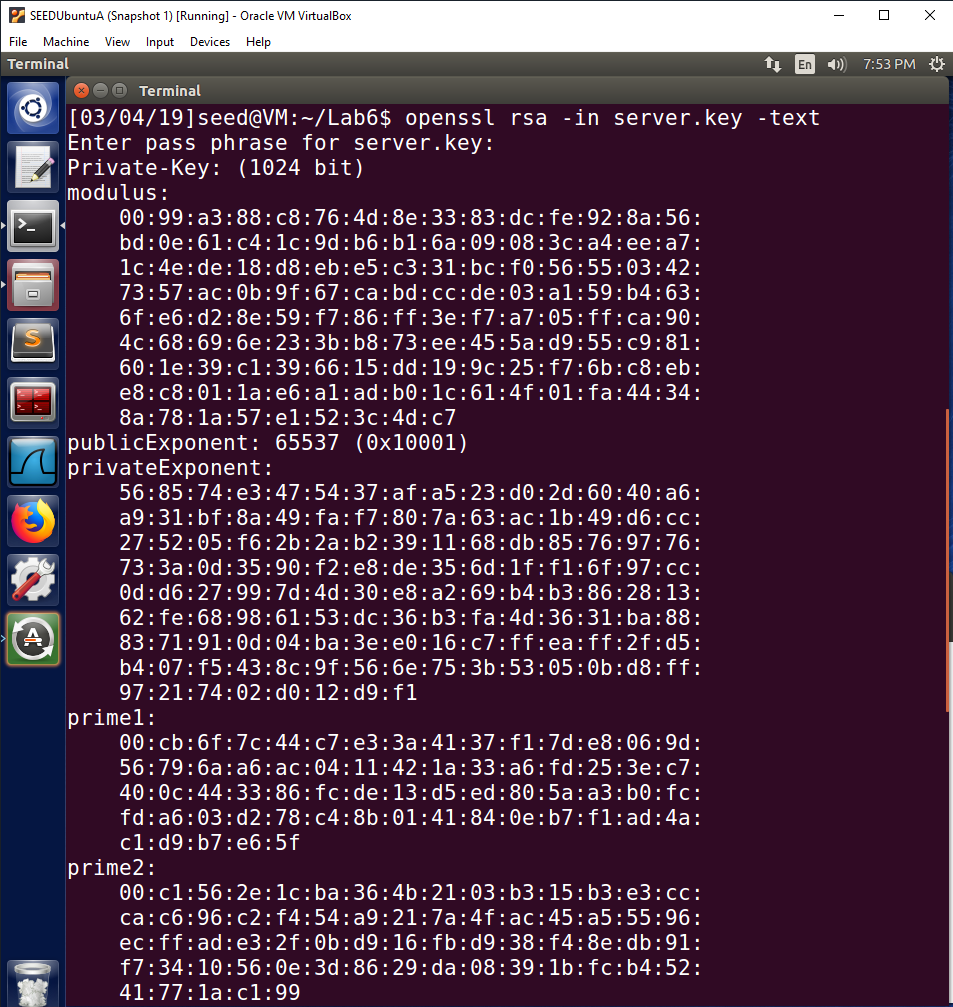
**Note: Pass phrase is “dees”.**

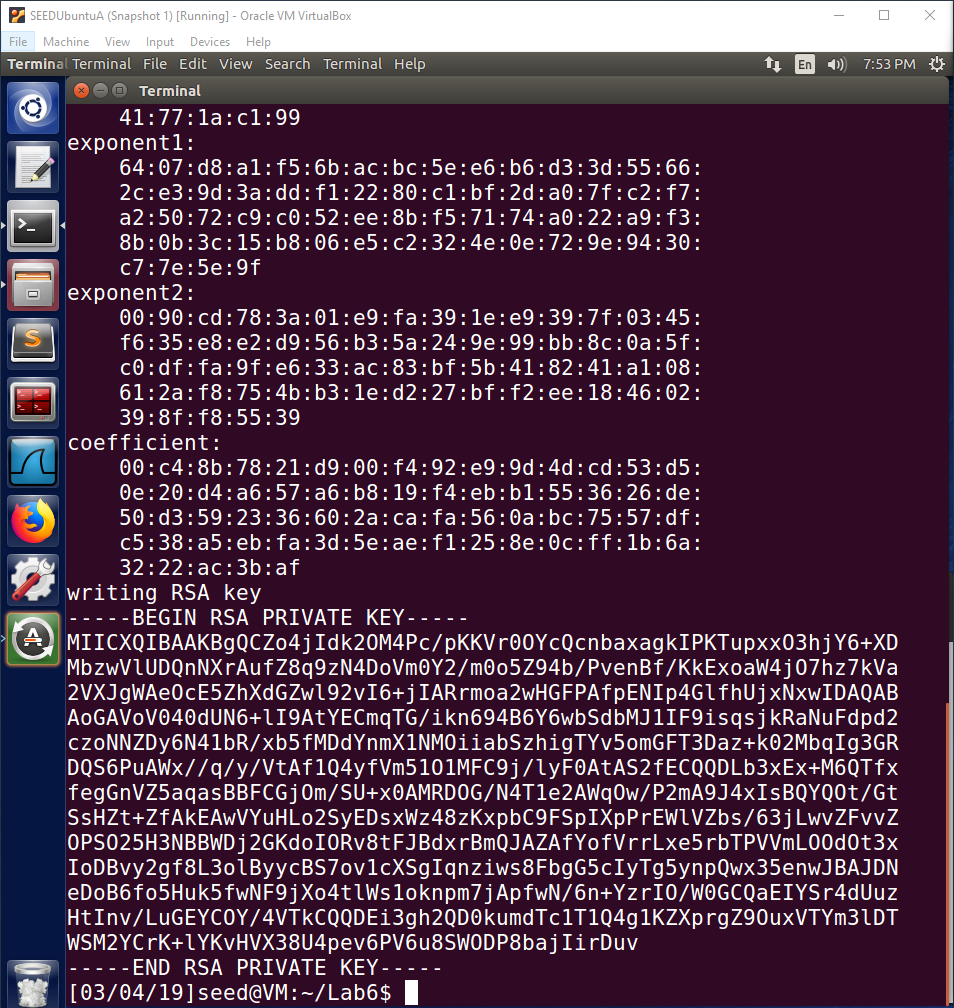
Observation:

The owner of SEEDPKILab2018.com generates a public/private key pair using the command “openssl genrsa -aes128 -out server.key 1024”. A pass phrase is required to password protect the generated “server.key” file.

Explanation:

The owner of SEEDPKILab2018.com must generate a public/private key pair that they want to use before they apply for a x509 certificate from a certificate authority. This command generates a public/private key pair that will be used by the SEEDPKILab2018.com owner. The command uses AES 128-bit encryption to generate the key pair. The output of the command is to the “server.key” file. We will decrypt this file and view the contents in the next image. The passphrase used to encrypt the file is required to decrypt and view the contents.



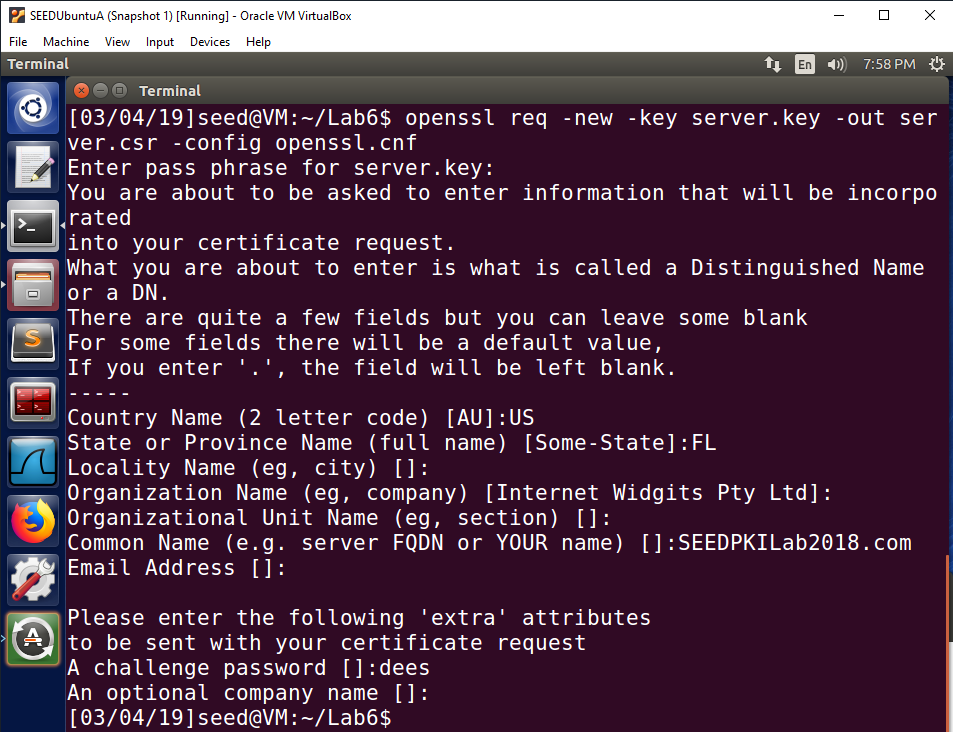


Observation:

The two images are the contents of the “server.key” file containing the public/private key pair.

Explanation:

The openssl command was used to decrypt the “server.key” file using the password we used to encrypt the file. This allows us to view the unencrypted contents of the file. It is important that this file is encrypted and is never viewed by anyone other than the owner, otherwise an attacker will be able to exploit the private and public key to perform attacks on communications to and from SEEDPKILab2018.com. It is definitely a bad idea to actually print out the file contents like this, but it is useful to see the contents of this file specifically for this lab to learn how it works.



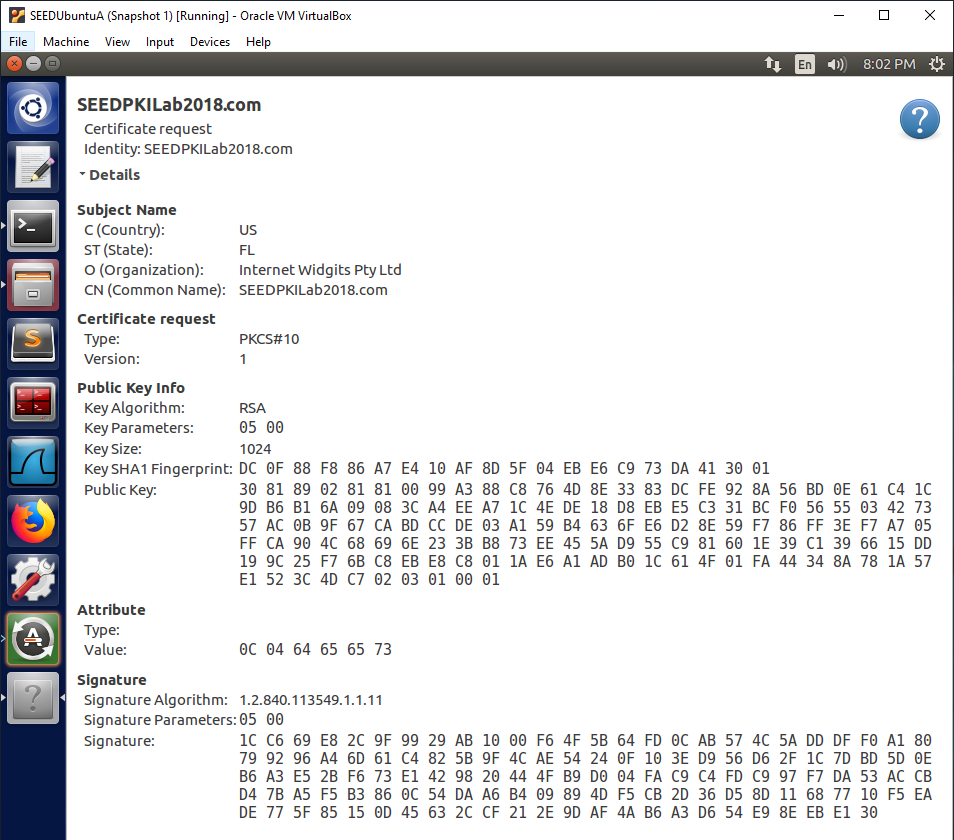
**Note: server.key pass phase is “dees”.**

Observation:

The openssl command is used to generate a certificate request that will be sent to the root certificate authority to be validated and signed. The input is our public/private key pair file that we just created for SEEDPKILab2018.com. The password used to encrypt the “server.key” file was requested so that the command can decrypt the file and use the contents in it. The output is the “server.csr” file. The configuration file used is “openssl.cnf”. Information such as country name, state, and common name has been filled in and a challenge password “dees” has been included.

Explanation:

Now that the owner of SEEDPKILab2018.com has generated a public/private key pair, in this step they are generating a certificate request. The input is the “server.key” file that contains the public/private key pair for their website. The output file is “server.csr”, which is the certificate request file. The configuration file used is “openssl.cnf”. The password used to encrypt the key pair file is needed to decrypt the file so the contents can be used in the certificate request. The private key from the public/private key pair for SEEDPKILab2018.com is used to sign the certificate request. The receiving certificate authority will verify this signature using the public key of the requester. This will allow the certificate authority to verify that the public key is in fact from the requester. This is important, because it protects the requester from attackers creating fake certificates for the public key of the real website owner.



Observation:

This is the contents of the “server.csr” certificate request file. The important parts of this process such as the public key and signature can be seen in the file as a long string of hexadecimal data.

Explanation:

The “server.csr” is the certificate request file that will be sent to the root certificate authority to ask the certificate authority to validate it and sign it. Since we are also the root certificate authority in this lab, we will be able to do this on the same machine. The public key in the certificate request file will be used to verify the signature in the certificate request file. Since the signature is generated using the private key from the SEEDPKILab2018.com website owner, it will be able to be verified by using their public key in the certificate request. This ensures the root certificate authority that the public key is owned by the requester who is the real owner of SEEDPKILab2018.com. Upon verifying this, the root certificate authority will sign the certificate request and the owner of SEEDPKILab2018.com will now have a valid certificate.

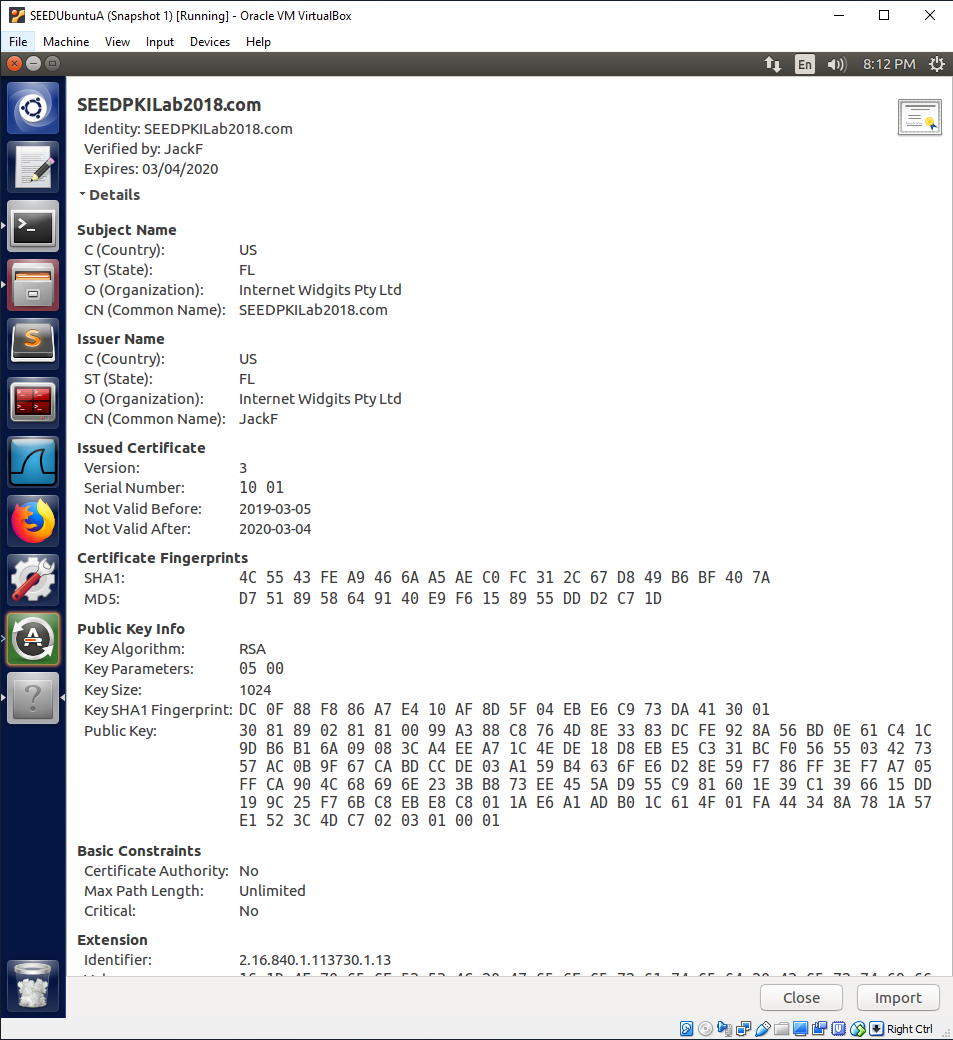


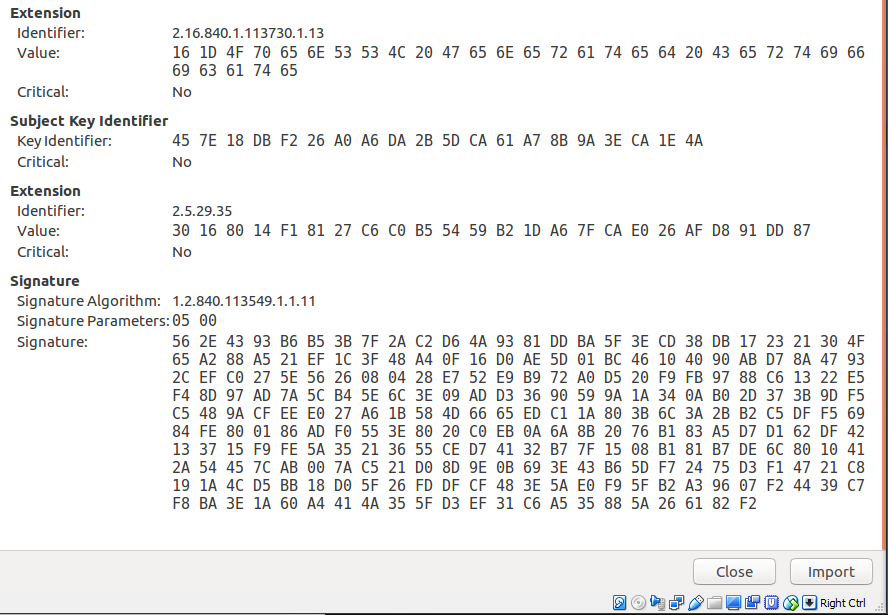
Observation:

In this step we are signing the certificate request while acting as the root certificate authority. The openssl command is used to sign the certificate request. The entire command is “openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf”. The pass phrase for the “ca.key” is required since this file is encrypted by a pass phrase that we entered earlier when creating the file. The request is checked to see if it matches the provided signature. The certificate details are displayed for the user to confirm the signing of the certificate. The second confirmation is to commit the changes to the data base that tracks certificate records, which is in the /home/seed/Lab6 directory on our machine.

Explanation:

Now we act as the root certificate authority and will assess the certificate request and sign it if we can validate the credentials of the request. In the real world, the certificate authority would also confirm the ownership of the common name, but in this case we are also the owner of the SEEDPKILab2018.com website so we can continue with the signing process. The openssl command has multiple important options that we used. The “-in server.csr” is the certificate request file that will be assessed. The “-out server.crt” is the certificate output file that will be generated upon a successful signing of the certificate. The “-cert ca.crt” is the certificate from us, the root certificate authority, that will be used along with the information from the certificate request to generate a new x509 certificate containing information from both. The “-keyfile ca.key” is the file containing the root certificate authority public/private key pair. We will use the private key from this file to sign the new certificate. The “ca.key” is a password protected, encrypted file that we created earlier in this lab, so the password is required in this step to decrypt the file and use the private key inside it. The “config openssl.cnf” is the configuration file used for this step. This tells the command where to locate the files and where to place newly generated files. The signature in the request is from the requester and goes with the requester’s public key in the request. The public key in the request can be used to check that the signature in the request is a valid signature. Once the public key is used to verify the signature in the certificate request, we can review the details and sign the new certificate. We agreed to sign the certificate and also agreed to commit the changes to the data base. We have now successfully signed the new certificate for SEEDPKILab2018.com while acting as the root certificate authority. The owner of SEEDPKILab2018.com can now use this certificate for their website.





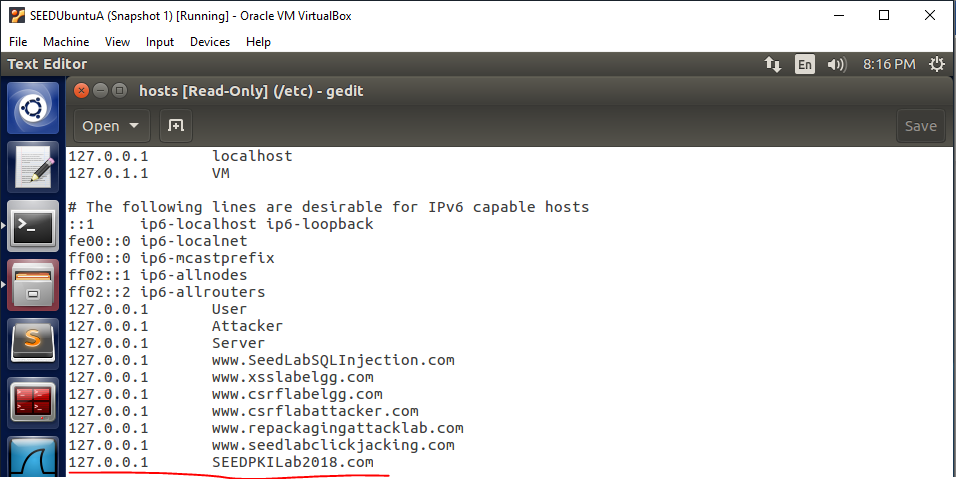
Observation:

These two images are the newly generated certificate for SEEDPKILab2018.com that has been issued by the root certificate authority, “JackF”. The certificate shows that it is for “SEEDPKILab2018.com”. It also shows that the issuer is “JackF”, which is the root certificate authority. The public key is the public key that the website owner wanted to use for their website when they generated their private/public key pair. The last part of important information is the signature, which was signed using the root certificate authority’s private key.

Explanation:

The new certificate show she correct subject information, issuer information, public key, and signature. The subject information is for “SEEDPKILab2018.com”, the issuer is “JackF”, the public key is the public key the website owner generated when they created their private/public key pair, and the signature is created using the root certificate authority’s private key. The root certificate authority’s certificate contains the public key that can be used to verify the signature on this new certificate. When someone needs to verify this certificate, they will verify the signature using the public key from the root certificate authority’s certificate. This will tell the person verifying the certificate that the public key in the SEEDPKILab2018.com certificate is the correct public key to use for encryption with that website.

**Task 3: Deploying Certificate in an HTTPS Web Server**

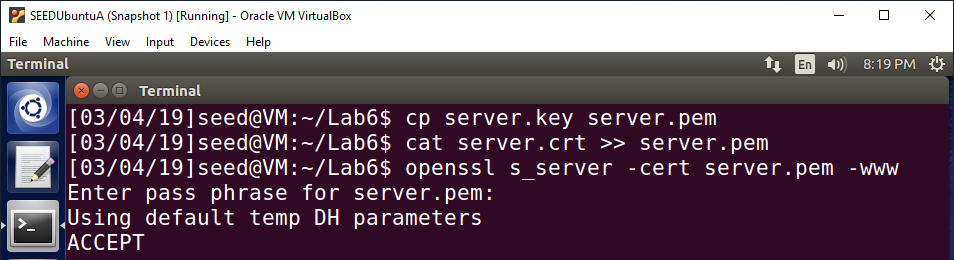


Observation:

The entry “127.0.0.1 SEEDPKILab2018.com” is added to the “/etc/hosts” file.

Explanation:

Since we are running the SEEDPKILab2018.com website on our local machine, we want to add this entry to the hosts file so that our browser uses the local machine IP address to access the website instead of trying to reach out to an outside DNS server to look for the IP address.

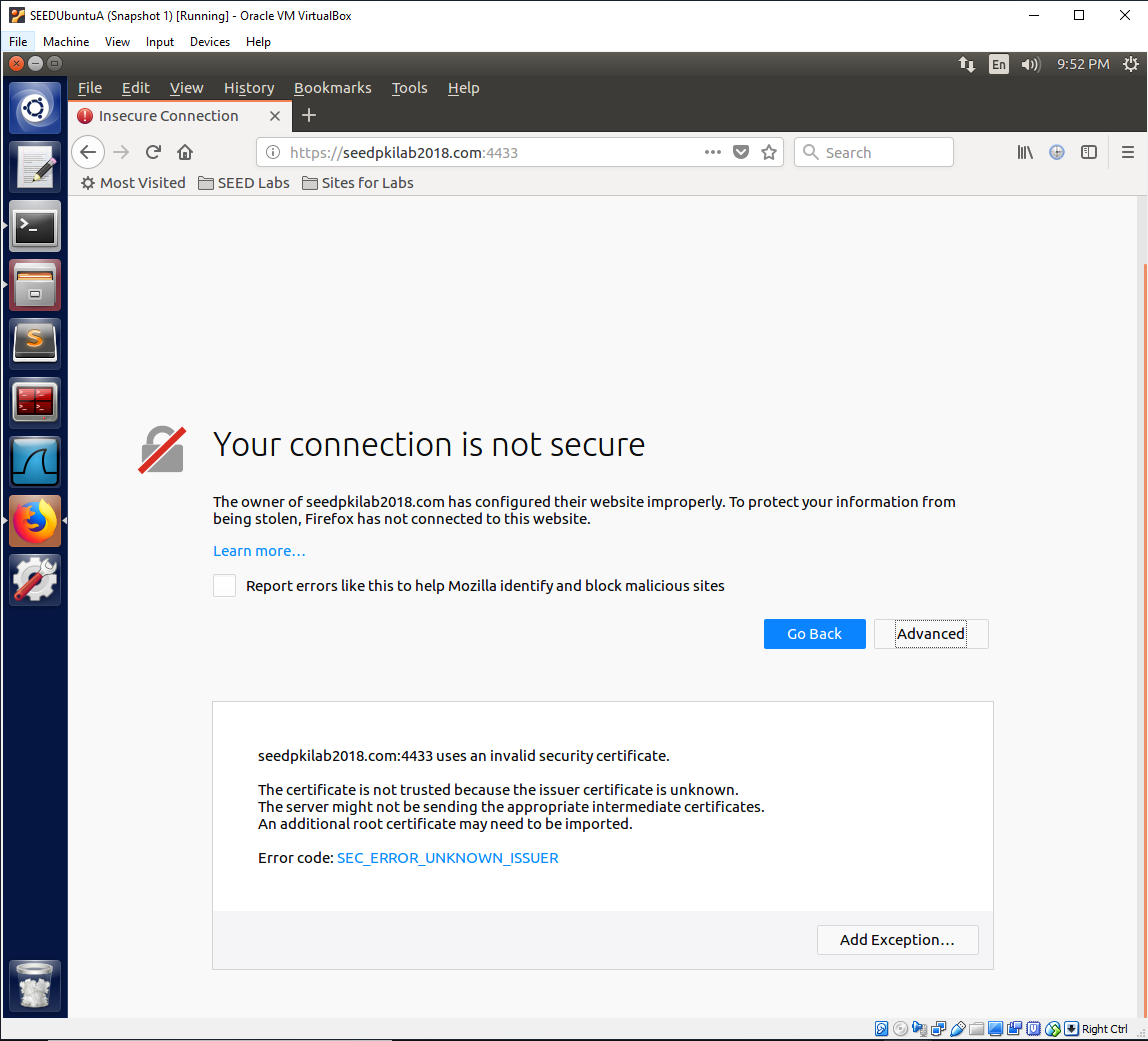


Observation:

The server key is combined with the server certificate by first copying the contents of “server.key” into the “server.pem” file and then using the cat command to append the contents of the “server.crt” file onto the “server.pem” file. The result is a file that contains the contents of both files. The final action is the openssl s\_server command, which starts a web server using the certificate file we just created. The default listening port is 4433.

Explanation:

The server private key and server certificate are combined into one file to be used by the web server. This is so the web server has a certificate containing the public key for the website as well as access to the private key for decrypting information that was sent to it and encrypted using its public key. The web server is started using the openssl s\_server command and is provided the new “server.pem” file as input for the certificate file. The web server will listen to port 4433 by default as a result of this command so we will connect to the website using port 4433 in the next steps.

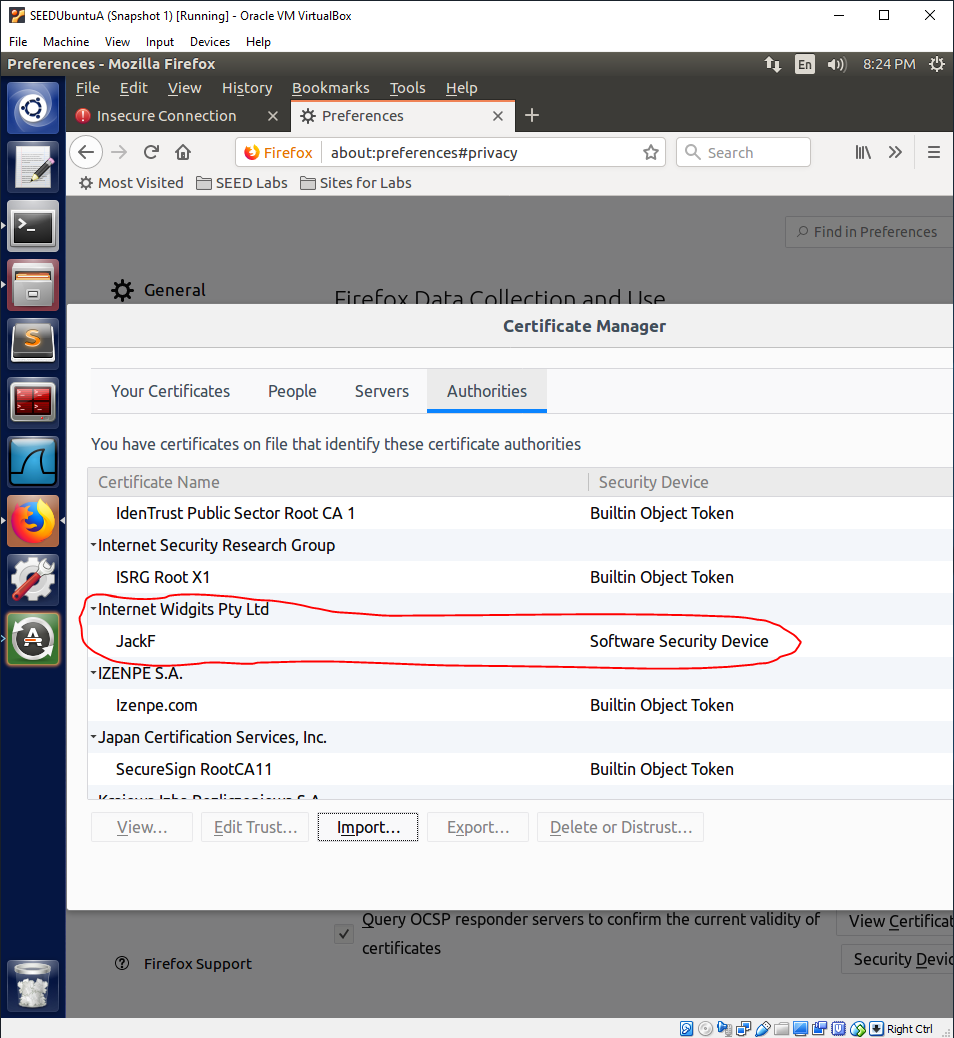


Observation:

We try to connect to the website using port 4433, but the connection is deemed insecure by the Firefox browser and it does not connect. The message “The certificate is not trusted because the issuer certificate is unknown” is displayed in the additional information section.

Explanation:

The connection to the website is deemed insecure so Firefox does not connect to the website. Firefox comes with preinstalled trusted root certificate authority certificates for verifying other certificates. Our recently new root certificate authority certificate is not included in these certificates because Firefox does not know we exist and has not authorized our certificate as trusted. For this reason, when the website’s certificate is checked, the issuer is not recognized as being on the trusted certificate authority list in Firefox, so the website certificate is not trusted. This is the reason the connection is rejected. What we will need to do is manually add our issuer certificate from our role as a root certificate authority to the list of trusted certificates in the Firefox browser. This will tell Firefox to trust the certificate from the root certificate authority we made and will see the website’s certificate as valid since it now trusts the certificate’s issuer, which is us.



Observation:

We have added our root certificate authority certificate to the list of trusted authorities in the Firefox browser.

Explanation:

We manually added our certificate for our root certificate authority, “JackF”, to the list of trusted certificate authorities in the Firefox browser. Now when the Firefox browser receives a certificate that is issued by us, “JackF”, it will check the list and see that we are a trusted certificate issuer. This will allow Firefox to connect to the website since it will now see the certificate issuer as a trusted certificate authority.

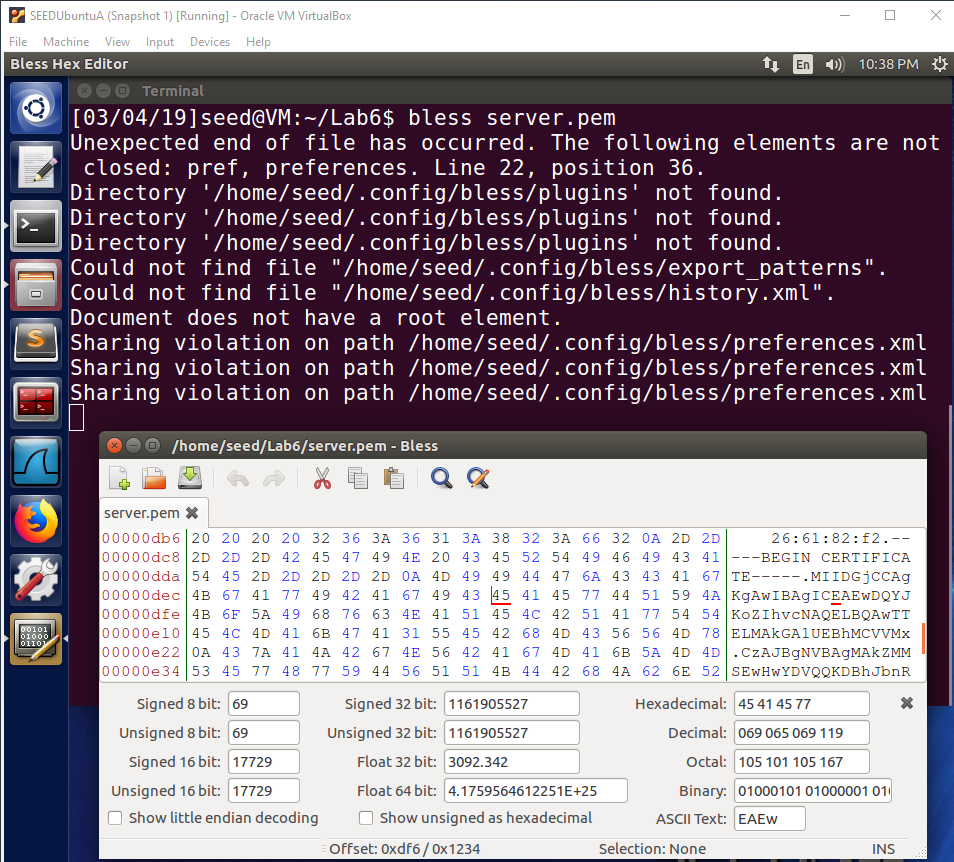


Observation:

Firefox has deemed the connection as secure since it now trusts the certificate of the website as a result of trusting the certificate issuer. The website was loaded successfully.

Explanation:

The Firefox browser checked its list of authorized certificate authorities and saw that the issuer of the website’s certificate was a trusted certificate authority from its list. This is a direct result of us manually adding our root certificate authority’s certificate to the list of trust certificate authorities in the Firefox browser. The website connection was seen as secure and the Firefox browser allowed us to connect to the website. The connection to this website uses the HTTPS protocol and uses the public key from the certificate to encrypt communications between the website host and the user.

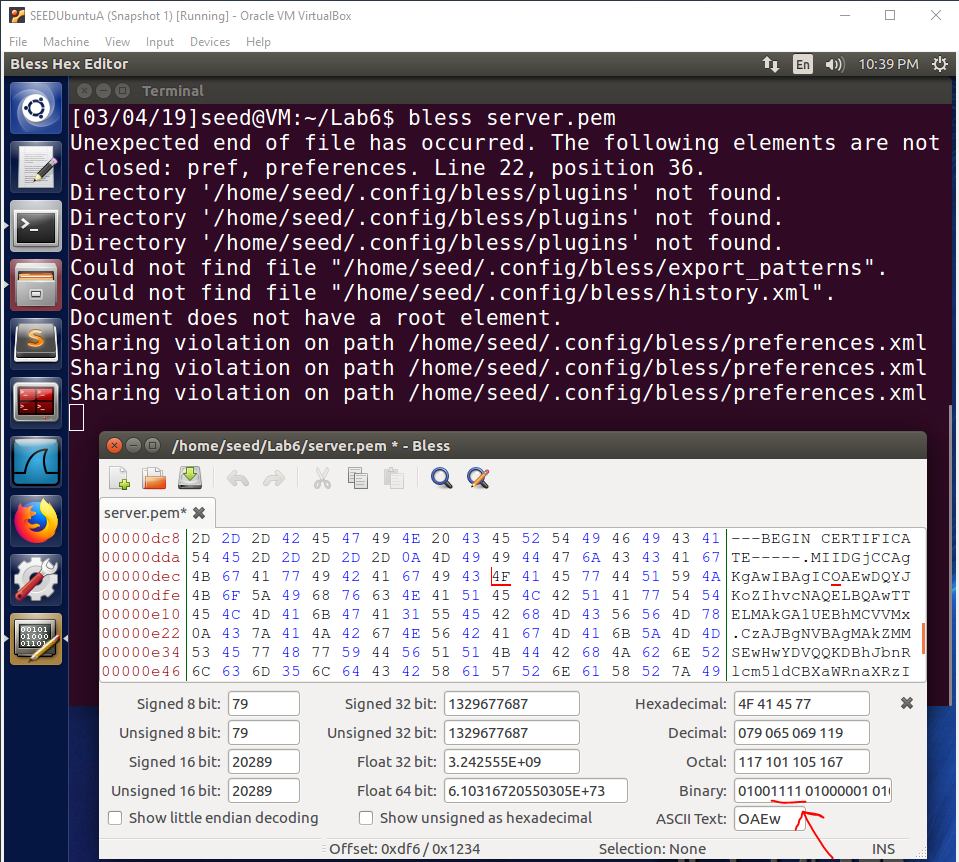


Observation:

This is the “server.pem” file before any changes viewed as hexadecimal data. The targeted byte is the ‘5’ in “45”. The targeted byte is underlined in red in the picture.

Explanation:

One byte of the “server.pem” file will be corrupted to see how this affects the connection to the website. The ‘5’ byte will be changed, which is a byte that is a part of the peer certificate signature. This should render the signature as invalid and the connection will be denied because it is seen as insecure due to an invalid signature.

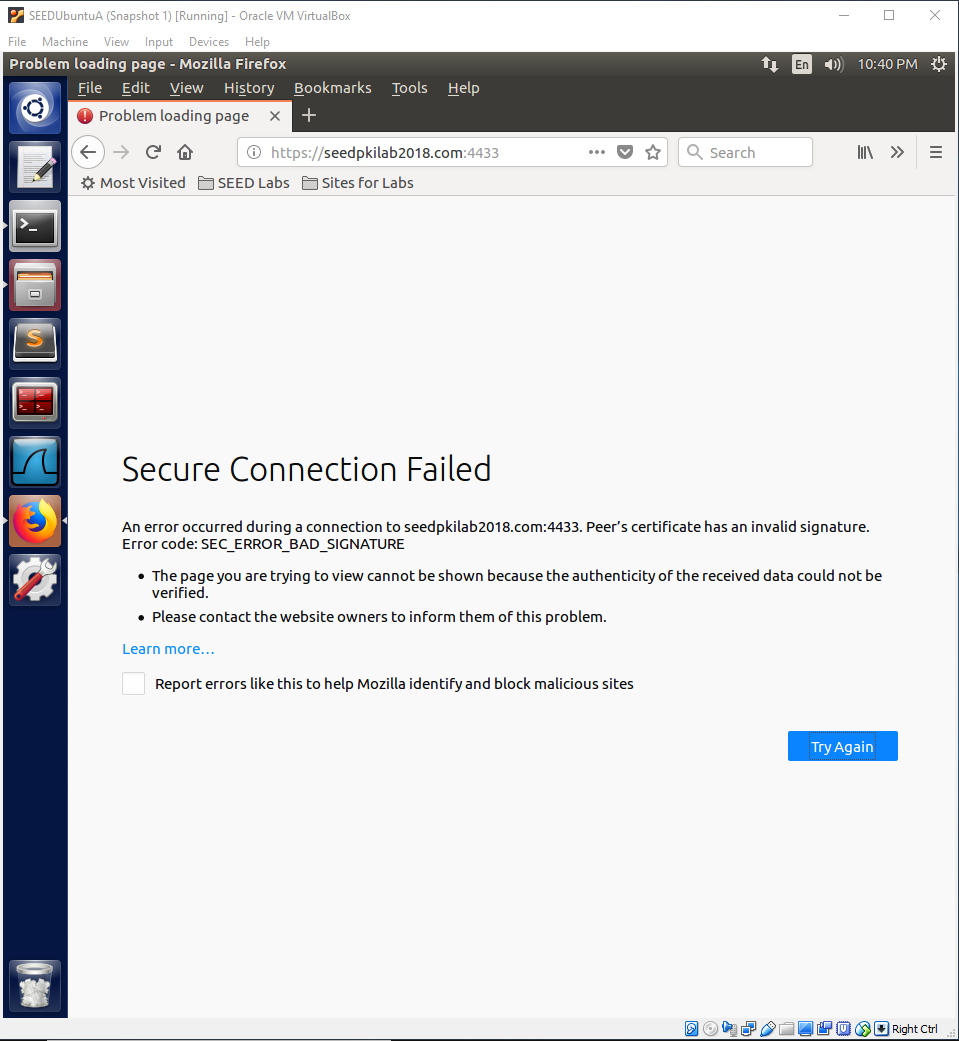


Observation:

The ‘5’ byte is changed to ‘F’.

Explanation:

The ‘5’ byte has been corrupted and now has the value ‘F’. This will alter the signature of the peer certificate. This will render the signature invalid, because during the signature validation process the signature will no longer be the exact value necessary to have a successful signature check outcome. This will render the connection insecure and the Firefox browser will reject the connection.

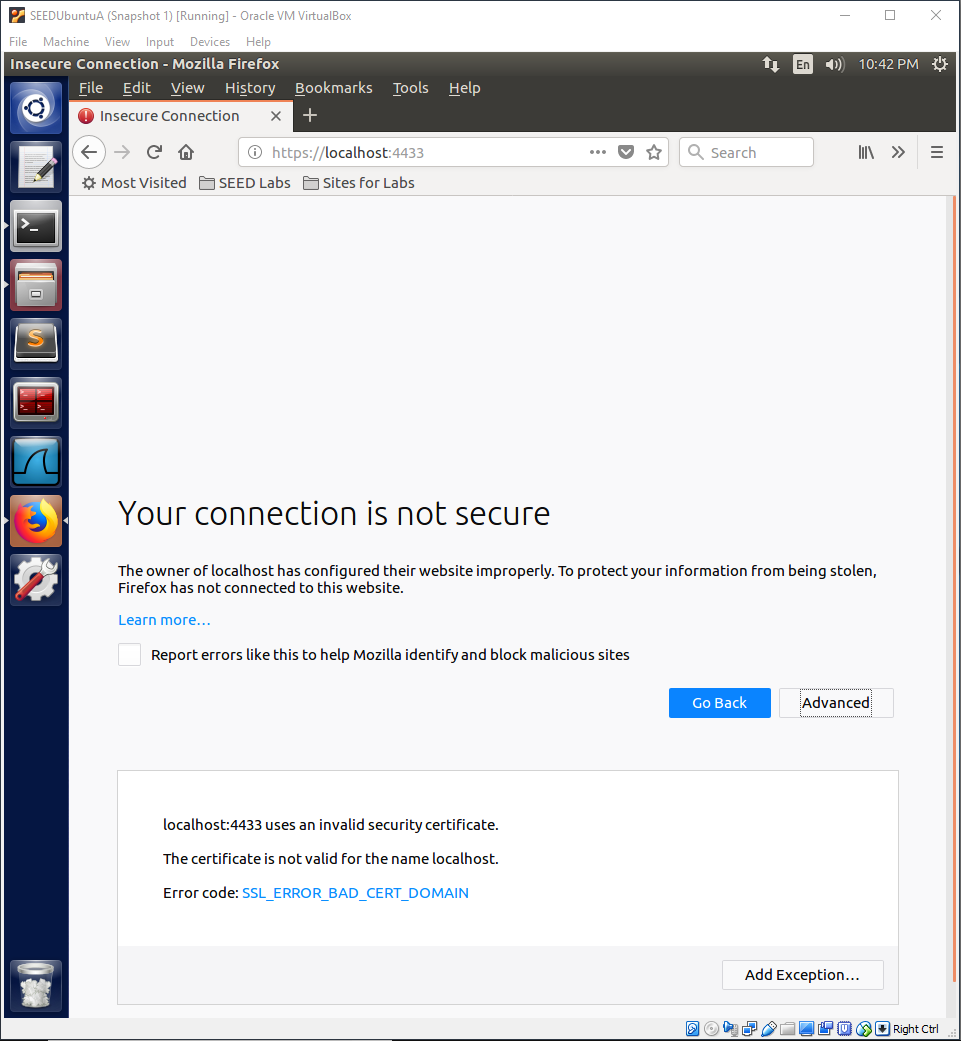


Observation:

The connection was rejected due to an invalid peer certificate signature.

Explanation:

As stated before, the peer certificate now has an invalid signature since we corrupted a byte of the signature. The corrupted signature no longer evaluates to the correct value when it is checked using the public key. The Firefox browser is unable to verify the signature, so for security reasons it rejects the connection.



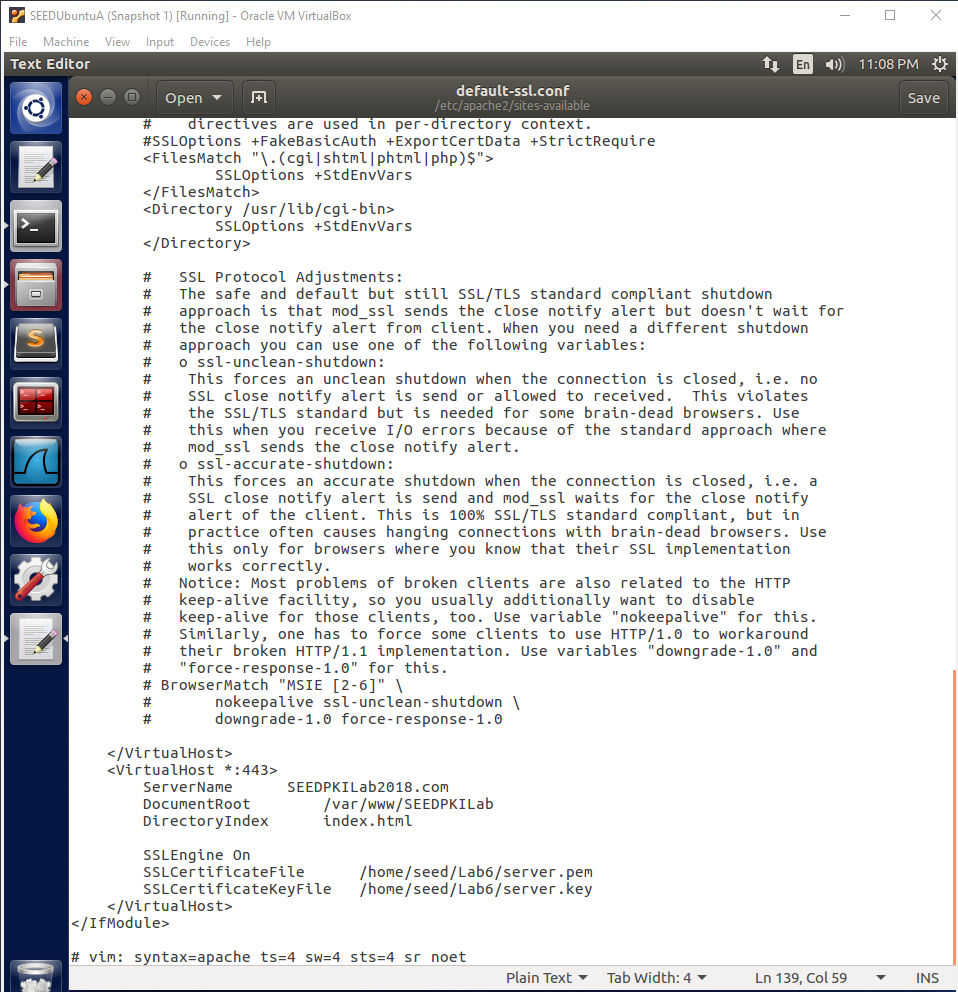
Observation:

We try to connect to the website using “localhost”. The certificate is deemed invalid since the certificate was not created for the name “localhost”.

Explanation:

This failed connection shows the importance of the common name field. The Firefox browser was given a valid certificate for the SEEDPKILab2018.com website, but the Firefox browser recognized the name being used was “localhost” instead of “SEEDPKILab2018.com”. For this reason, the Firefox browser deemed that the website name we were trying to use, “localhost”, was not the same as “SEEDPKILab2018.com” and was insecure since the certificate was not meant to be used for the name “localhost”. For security reasons related to the importance of the common name, the connection was deemed insecure and the connection was refused. If “localhost” were actually a malicious website, there are two outcomes of this type of situation as described in the lecture. If the malicious website provides their valid certificate with their common name, the user must be sure to check the common name manually so they can see that they are not connecting to the correct website. The other outcome is the malicious website providing the real website’s certificate. The malicious website does not have the real private key so it would not be able to pass any encryption challenges. This shows that to protect against attacks on common name, it is important to check the common name like Firefox has done in this situation.

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

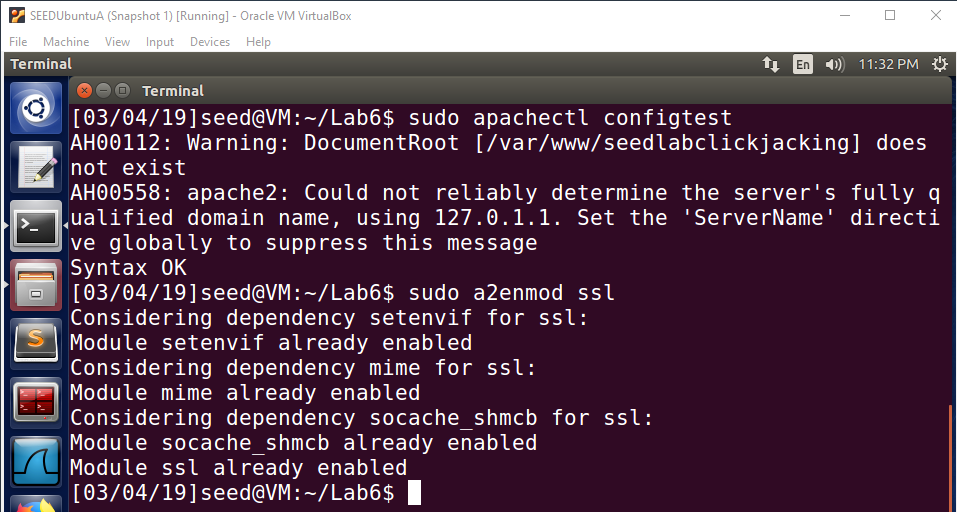


Observation:

The entry for “SEEDPKILab2018.com” was added to the “default-ssl.conf” file.

Explanation:

We needed to add an entry for our website to the default sll configuration file. This will add our website to the list of websites that our apache server supports and hosts. The port specified is “443”, since this is the port that HTTPS listens on. The server name is the name of our website, “SEEDPKILab2018.com”. The document root is the base directory where the apache server can find the website files such as html files for the web pages. The directory index is the basic html file to serve as the basic frontpage for our website. I created a basic html page with my name and the class name to be used as the index html file. We want SSL to be used so we set the SSL Engine to, “On”, so it is active. We then provide the paths to the certificate file and the key file to be used for the website. This entry will tell the apache server where to locate all of these files that are necessary for hosting the website.

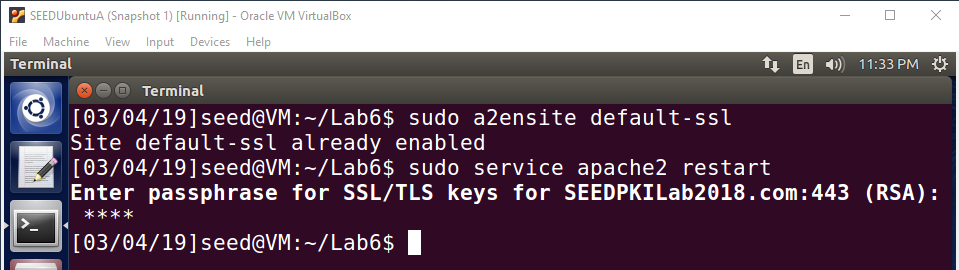


Observation:

The commands “sudo apachectl configtest” and “sudo a2enmod ssl” are used.

Explanation:

The “sudo apachectl configtest” command checks for any errors in the configuration file. The configuration file error check passed so we can continue with the setup of the apache server. The “sudo a2enmode ssl” command enables the SSL module. This ensures that SSL is being used for the apache web server, which is essential to ensure our website communications are secure.



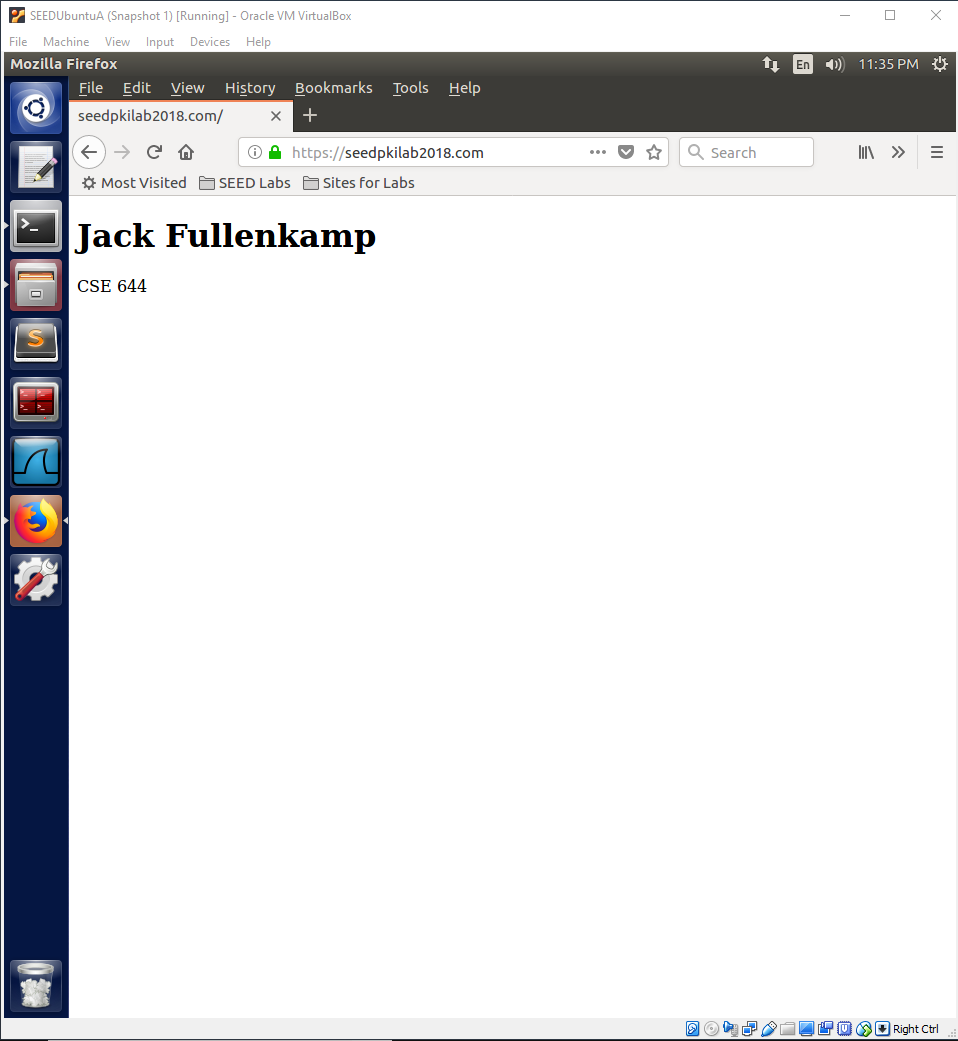
**Note: Passphrase for SSL/TLS keys is “dees”.**

Observation:

The “sudo a2ensite default-ssl” command is used and then the server is restarted using “sudo apache2 restart”.

Explanation:

First, we enable the website by using “sudo a2ensite default-ssl”. This command checks for entries in the “default-ssl.conf” file to enable the websites that it has entries for. Since many changes have been made, to make those changes active the apache server must be restarted. The server is restarted using “sudo service apache2 restart”. When the apache server restarts, it will reload all of the configuration files and the new changes will now be effective in the apache server. A passphrase for the SSL/TLS keys is required, so we provide the password “dees”.



Observation:

The website running on the apache server is able to be successfully accessed using the Firefox browser.

Explanation:

This website is running on the apache server and is using the same certificate that we used when running the website using the openssl command. The website is able to be accessed since the certificate is issued by a trusted issuer since we manually added our root certificate authority to Firefox’s trusted authorities list earlier in this lab. As a result of our certificate being trusted by Firefox, we are able to successfully connect to the website and view the index.html page that I created. The connection uses HTTPS so all communications to and from the website are encrypted as a result. This shows another way that the website can be hosted using the same certificate. This also shows that regardless of how the website is hosted, the certificate is still trusted by Firefox since the issuer is in Firefox’s list of trusted certificate authorities.