Gabriel Naples
2/8/2019
Lab 2

Section 2.1, Creating the Certificate Authority

The first step that had to be taken in creating the certificate authority was to copy the openssl.cnf to the local directory. The next step was to create the sub directories that the openssl.cnf file uses when creating keys. The subdirectories are created within the same directory that the openssl.cnf file was copied to.

```
[01/31/19]seed@VM:~/demoCA$ cp /usr/lib/ssl/openssl.cnf .
[01/31/19]seed@VM:~/demoCA$ ls
ca.key  crl        newcerts    private
certs   index.txt  openssl.cnf  serial
```

After these directories were made then I was able to generate a self-signed certificate for the certificate authority. This certificate is the root certificate and is used to sign other certificates and allow access. This root certificate uses its private key to sign certificates so that when a certificate is submitted it can check for authenticity by decrypting the private key with the public key which assures that the user was authorized initially by the CA.

Below is the code that creates the root certificate:

```
[01/31/19]seed@VM:~/demoCA$ openssl req -new -x509 -keyout ca.key -out ca.crt -confi
g openssl.cnf
Generating a 2048 bit 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]:US
State or Province Name (full name) [Some-State]:Pennsylvania
Locality Name (eg, city) []:State College
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Penn State
Organizational Unit Name (eg, section) []:451
Common Name (e.g. server FQDN or YOUR name) []:GabeCert
Email Address []:gjn5070@psu.edu
[01/31/19]seed@VM:~/demoCA$
```

The output, ‘ca.key’, is the private key that is used to sign certificates. When the root certificate is created it also requires the creation of a password to allow it to use the private key to sign certificates.

Section 2.2 Creating a Certificate for SEEDPKILab2018.com

For the website SEEDPKILab2018.com to have a secure SSL connection it needs to be recognized by a certificate authority. After the CA signs the certificate for this site it will then be able to be configured a secure SSL connection. To do this the first thing I had to do was generate a key pair for SEEDPKILAb2018.com. The keys were encoded RSA keys that were encoded into the same file, 'server.key'.

Below is the code used and the outcome:

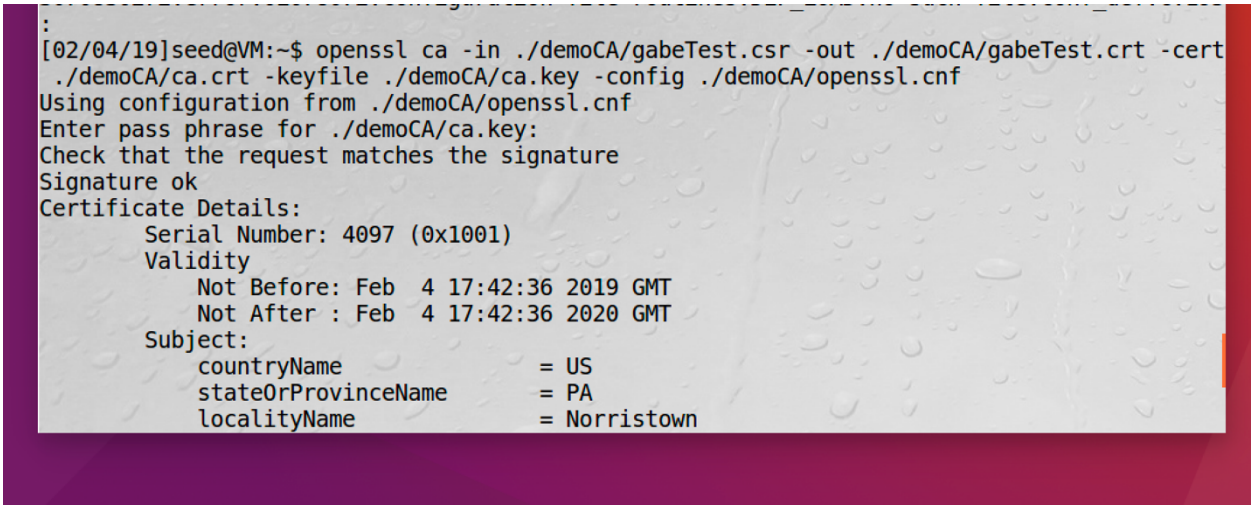
```
[01/31/19]seed@VM:~/demoCA$ openssl genrsa -aes128 out server.key 1024
usage: genrsa [args] [numbits]
 -des            encrypt the generated key with DES in cbc mode
 -des3           encrypt the generated key with DES in ede cbc mode (168 bit key)
 -seed
                 encrypt PEM output with cbc seed
 -aes128, -aes192, -aes256
                 encrypt PEM output with cbc aes
 -camellia128, -camellia192, -camellia256
                 encrypt PEM output with cbc camellia
 -out file       output the key to 'file
 -passout arg    output file pass phrase source
 -f4             use F4 (0x10001) for the E value
 -3              use 3 for the E value
 -engine e       use engine e, possibly a hardware device.
 -rand file:file:...
                 load the file (or the files in the directory) into
                 the random number generator
[01/31/19]seed@VM:~/demoCA$
```

The below code shows the actual contents of the key file, and the RSA key, after using the command: openssl rsa -in server.key -text

```
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQCw124bngkF66/fLOPdsIdyj/1UxvGCAnBH+Ahxv569iqt/Xeye
FPPSd5cKGExiYOT0KspUANTtxLCU2kQChHim2bjBJ/D9mvreKHn0w8hiikW1OOjF
r7FIO3Vsm9smjRGQBFErm8qY7DbWMwgNaFWCrMzERlqLW8IPdFBZEuartwIDAQAB
AoGAf55IQLfwrQwkNHuRycd+R+evm0Giwf7+hoFxnpz5crGjB9yBSKe4OXfCUS+t
2AUy1R4YOz0KbLhX8mP9Vo/A/Pyga0+hx1e3TBmT44s/DuwLqIHkNB5hSgizSzWB
16iK4lsTyjizraekA1s700FqjFYLWp6hVx6+Oak1afJw7VECQQDdtyEzZ8uIm+UQ
0nrQVLX2Mnbwr2NPAkNZ8hA92YS2Eaiy5gcgdQu1U4ZUm+EehvOOHmrUHPqEQgBM
T6sA43bfAkEAzC/qbLo4QujqKLiPAZTdXoBPFKFuAxlV8Y4UJncKhD1nnmCeS4HF
xXmAfbG4SNgt04wTwCy47yhpk0d6n0WeKQJBANDsjECggk7MeGINtOUzrfO9Qg8z
dmUxQL41N9DQdBoNMebCk9yl7s6DUoXQDaZyjHdTel8+n2KZvyYpvH4t2gkCQGtl
3JD51fPjnJ50jMuRKGq3ZGD+k+XPYrQePh0JSh/KSIE2YqxmgsbepKX4X2z5QAlg
SVcwbslkYTuVQjsaHBkCQQCe0rsEE3Bi96IMFpdddCR9Xgs7c/rPvwuOkOgyiOvI
AtDrWEh7vYu6SOSFLTZx0kPco9pgter5x0n4U4cialsy
-----END RSA PRIVATE KEY-----
[01/31/19]seed@VM:~/demoCA$
```

The next step is to create certificate signing request. This step creates the server.csr file which is essentially the public key and common name for the SEEDPKILab2018.com server. The certificate authority uses this public key to create a certificate for the server. The certificate authority also checks that the identity in the certificate signing request matches the server's true identity. This is for authenticity purposes.

Below is the process of the CA signing the certificate request that is sent to it. The example code below entered into the command line is exactly what was used to sign the 'server.csr' and create the 'server.crt.' So everywhere it says 'gabeTest' I had 'server' on the initial request. (I forgot to take a screenshot of the actual command when making it, but I have screenshots of the rest of the process from the original.) The code below says that the in statement to the CA is the certificate sign request (the csr) and it creates the 'crt' file which is the actual certificate. It specifies that it is using the 'ca.crt' certificate to sign the server's certificate and 'ca.key' is the certificates authorities' private key which it signs the certificate with. This private key is then decrypted later by the certificate authority when the server uses the certificate and is how authentication is provided. In the code below I also had to add the full file paths to the command because I was having issues getting it to point to the correct files in the directories.



The subsequent lines of code show the details that the certificate authority sees when signing and creating the server.crt. It sees the information within the 'csr' and makes sure everything is accurate and then creates the certificate. The CA authority needs to enter their password to confirm making the certificate and signing it with the private key.

```
Enter pass phrase for ./demoCA/ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Jan 31 15:36:07 2019 GMT
            Not After : Jan 31 15:36:07 2020 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Pennsylvania
            localityName              = State College
            organizationName          = Penn State
            organizationalUnitName    = 451
            commonName                = SEEDPKILab2018.com
            emailAddress              = gjn5070@psu.edu
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                6E:45:DE:0A:EC:E4:A8:6E:F4:DF:C2:34:AC:18:6D:D3:EF:02:13:96
            X509v3 Authority Key Identifier:
                keyid:63:BB:6A:DD:CD:D0:23:88:52:FC:EC:33:E2:C5:D5:76:A1:F4:98:AA

Certificate is to be certified until Jan 31 15:36:07 2020 GMT (365 days)
Sign the certificate? [y/n]:
```

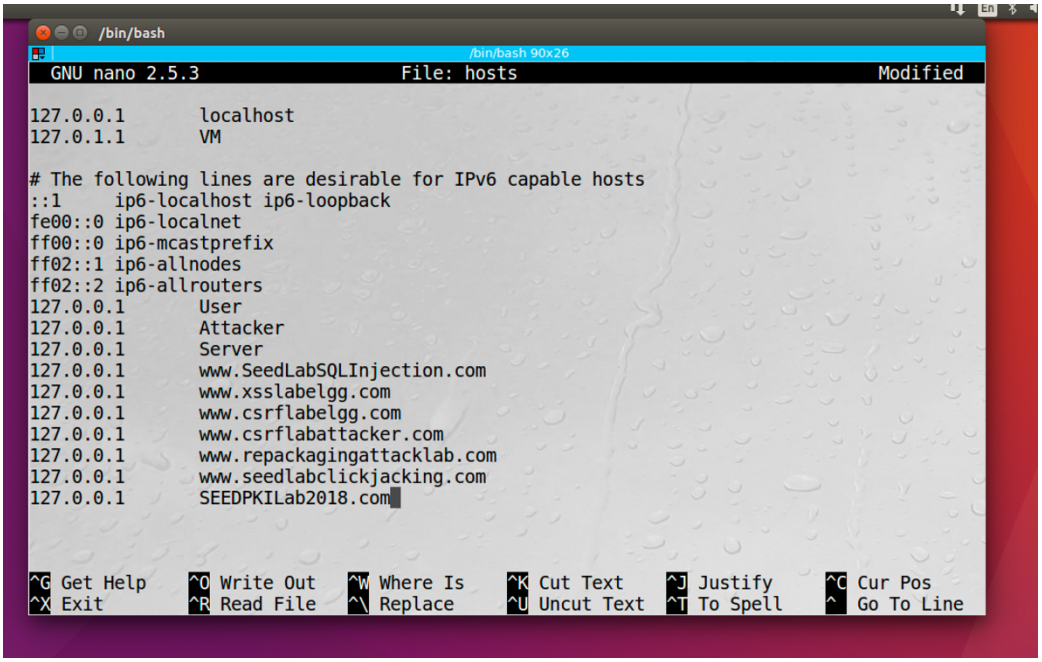
```
/bin/bash 101x29
        Validity
            Not Before: Jan 31 15:36:07 2019 GMT
            Not After : Jan 31 15:36:07 2020 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Pennsylvania
            localityName              = State College
            organizationName          = Penn State
            organizationalUnitName    = 451
            commonName                = SEEDPKILab2018.com
            emailAddress              = gjn5070@psu.edu
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                6E:45:DE:0A:EC:E4:A8:6E:F4:DF:C2:34:AC:18:6D:D3:EF:02:13:96
            X509v3 Authority Key Identifier:
                keyid:63:BB:6A:DD:CD:D0:23:88:52:FC:EC:33:E2:C5:D5:76:A1:F4:98:AA

Certificate is to be certified until Jan 31 15:36:07 2020 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
[01/31/19]seed@VM:~$
```

To get the certificate to be generated I had to go into the openssl.cnf file to adjust the “policy” configuration since I was initially having trouble generating the certificate. Below is the code I adjusted to change it to “policy = policy_anything”

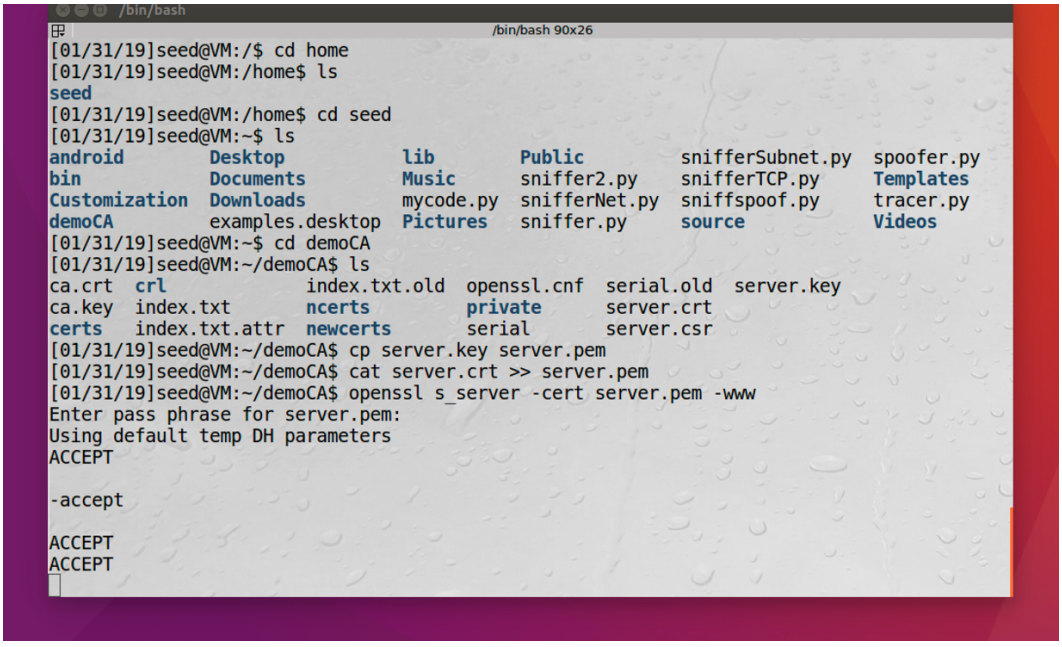
```
# and supplied fields are just that :-)
policy          = policy_anything

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied

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

Section 2.3 Deploying Certificate in HTTPS Web Server

The first step is to configure the DNS. For the computer to recognize the name of the server it must be put into the /etc/hosts file. I assigned it the IP of 127.0.0.1 and gave it the hostname SEEDPKILab2018.com. This maps the server to the localhost and enables it to run.

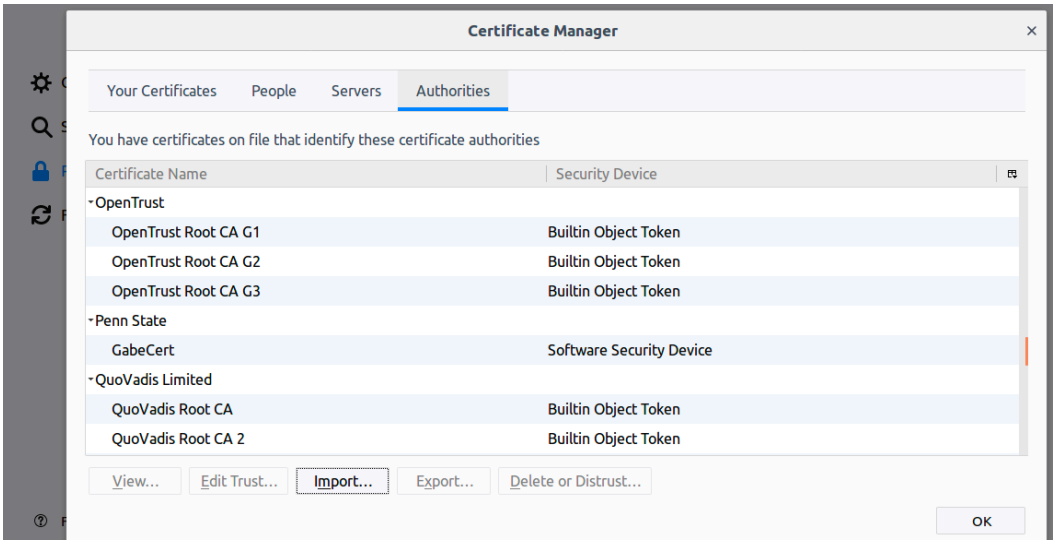


The next step was to combine the server secret key with the certificate to create one file ‘cat server.crt >> server.pem’ (‘server.pem is just a copy of server.key’ in the Privacy Enhanced Mail Format). After that the server was ran with the code ‘openssl s_server -cert server.pem -www’

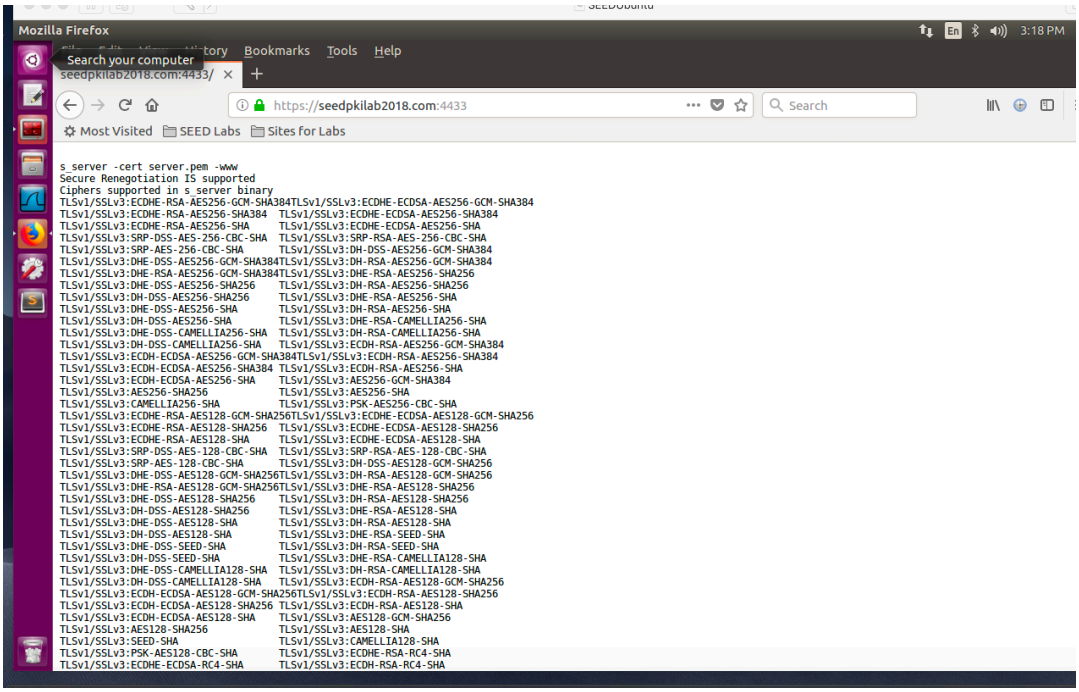


For this server to be recognized I first had to load the certificate authorities' certificate into Firefox. If this was signed with a globally recognized certificate authority, then this step wouldn't be required but since its signed by a local authority I created I have to manually put the ca.crt on Firefox before it can start authorizing certificates that are signed by the CA.

Below is the menu where the ca.crt can be uploaded and where my certificate authority is contained. To get to this menu open Firefox and got through the menus, Edit > Preference > Privacy & Security > View Certificates, and then hit "import." Below you can see my certificate listed under Penn State.

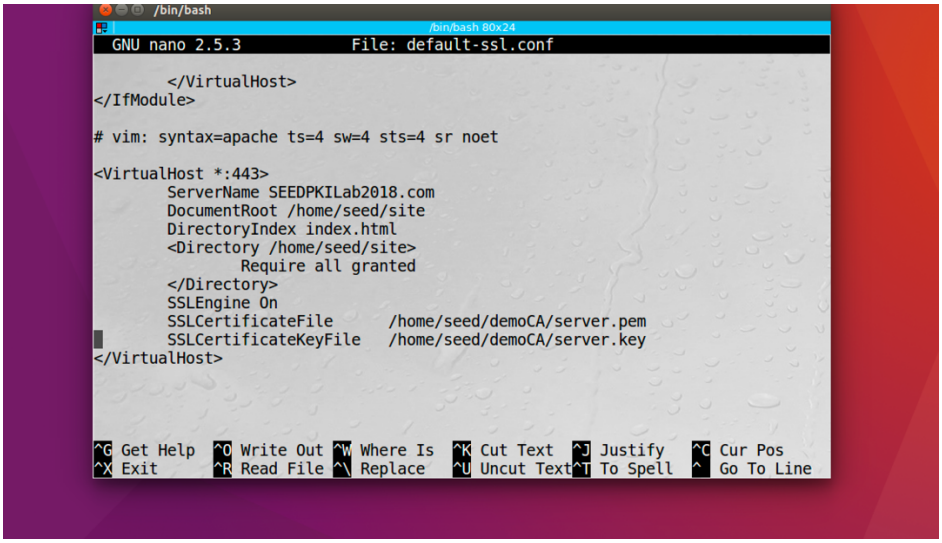


Next, I pointed the firefox browser to https://SEEDPKILab2018.com:4433. The web browser then shows certificate information.



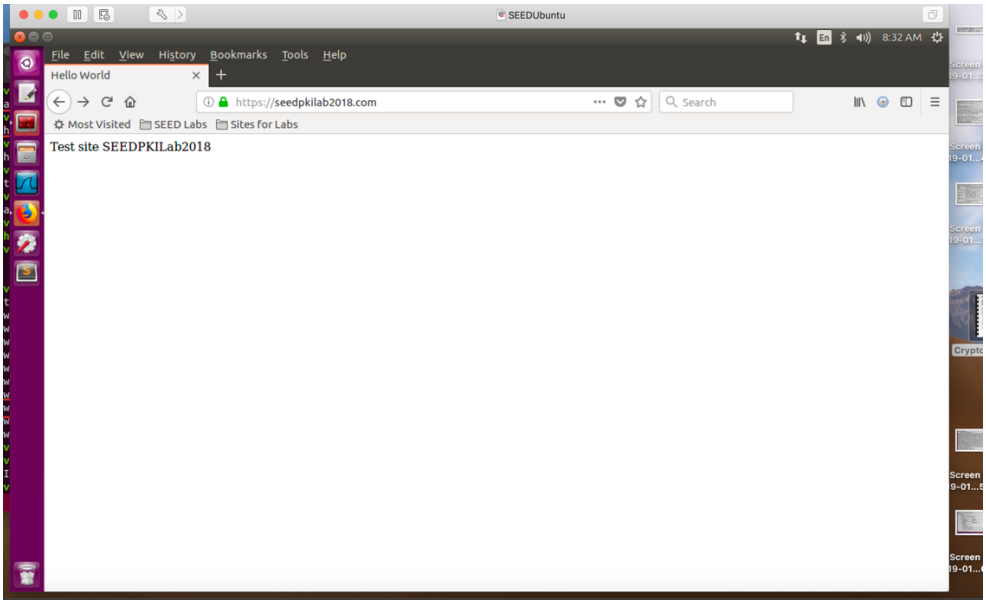
Section 2.4: Deploying a Certificate in an Apache-Based HTTPS Website

In this section I used the apache server to host the SEEDPKILab2018.com website, and it uses the certificates to ensure that it has a secure and authentic connection. The first step to this was editing the 'default-ssl.conf' which is located in the /etc/apache2/sites-available directory. Within this directory is also the '000-default.conf' which can be edited if you want to create a site and not use SSL.

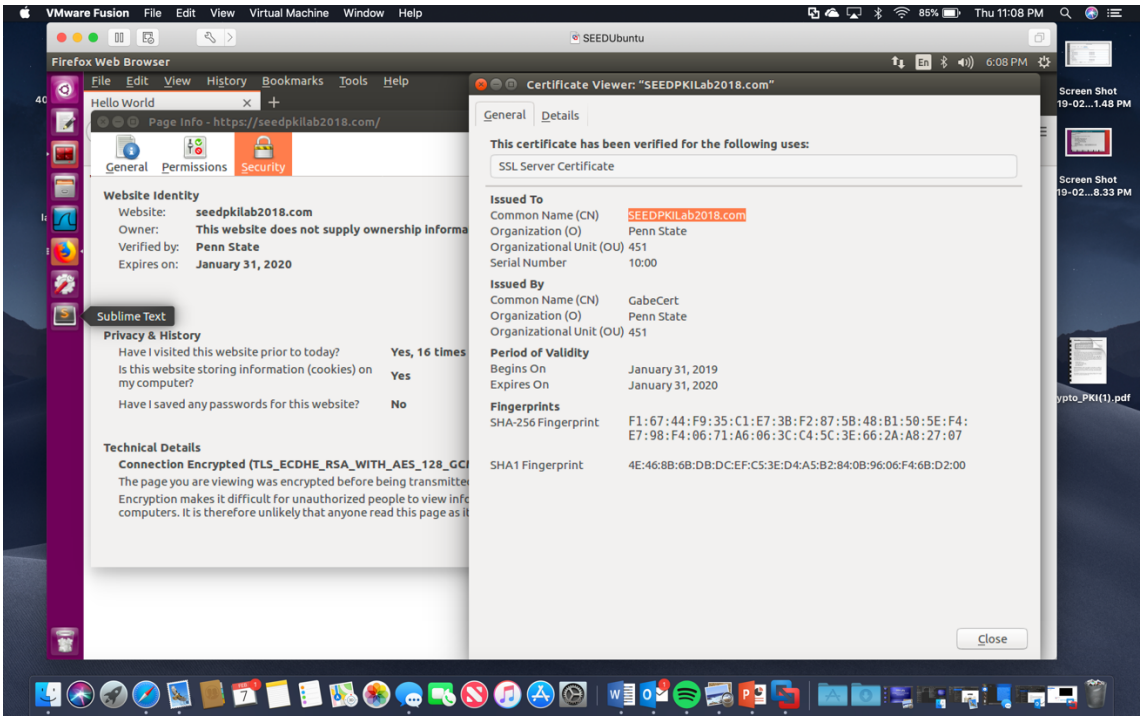


The above code states the ServerName DocumentRoot and DirectoryIndex. The DocumentRoot has the file path to the directory that contains my index.html file. The DirectoryIndex links to the index.html file. The '<Directory /home/seed/site>' section contains the line 'Require all granted'. This allows all IP addresses to access the site folder and HTML file. Without this line I was not able to connect to the website and would get an access denied message when trying to connect. The message stated "Permission Denied: access to / denied" which made me believe that it was a permissions issue. With the above code I was able to access the site with a secure connection as shown by the green lock. The 'SSLCertificateFile' links to the servers' certificate which was signed by the CA. This will be used by the CA to authorize the connection. The 'SSLCertificateKeyFile' links to the private key of the server. To use public key encryption apache needs both of the keys in the pair to decrypt and encrypt information.

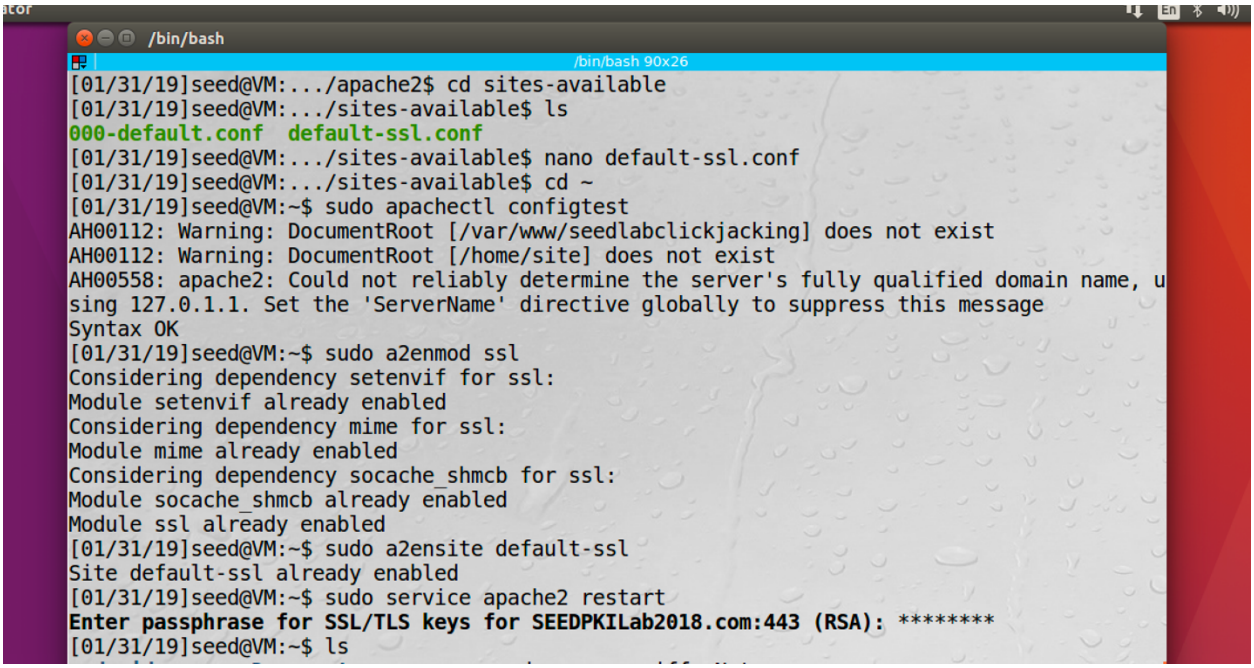
Below is the SEEDPKILab2018.com site loaded on Firefox with an SSL connection:



Below is how the certificate is viewed by the user:



Below is the code that enables, checks, and runs the server:



Explanation

For there to be an SSL connection with a web server there has to be a certificate authority to authenticate the connection. First the certificate authority creates its own self-signed certificate. This is the root certificate and ensures that the CA is fully trusted. The certificate authority uses this certificate to sign certificate requests from sites that want to be authenticated for an SSL connection. When a company wants to have their server authenticated with a certificate, they need to create a key pair. The company sends their signing request which includes their public key and other data like their common name. The Certificate authority takes this request and makes sure its authentic and then the CA signs it with its private key. The private key is how authentication is carried out when the site accesses it later. Now that the company has its certificate signed from the certificate authority it can use it to establish an SSL connection. When the company sets up a connecting it takes the certificate signed by the CA's private key and encrypts it with its own public key. The CA can then decrypt this certificate with the users' private key which proves that it did come from that user. Next the CA can use its own public key to decrypt the private CA key which proves that the CA signed the certificate and that it is valid.

If a man in the middle attack were to take place, then a malicious attacker would have to act as a proxy between the user and the server they're trying to securely communicate with. The man in the middle would intercept communication and send his own public key to the user while posing as the server. The user would then mistakenly use this malicious public key to encrypt the secret message which the attacker could then read. The attacker can then forward the request to the server the user was initially trying to reach and establish itself in the middle of communications. This way the man in the middle can passively intercept all of the information that was supposed to be secret.