East West University

Summer-2022

Project-1 Report

Course Title: Cyber Security, Law and Ethics

Course Code: CSE87

Submitted To: Rashedul Amin Tuhin (RDA)

Senior Lecturer,

Department of Computer Science and Engineering

Submitted By:

Name	ID
Lata Rani Saha	2018-2-60-013
Shajnin Alam Sarnali	2018-2-60-015
Sadia Afrin	2018-2-60-011

Date of Submission: 10 August 2022

Title: Securing a networked system with Public Key Infrastructure Implementing Transport Layer Security on HTTP for https:// connection

Tools: Linux Mint, Virtual Machine, OpenSSL, bind9 and dnsutils Libraries, Python, Django server.

The work of this project was done on a Linux (Ubuntu) virtual machine using OpenSSL tool and bind9 and dnsutils libraries.
We have a server on the machine address 192.168.0.193, which we want to turn into a website named "**verysecureserver1.com**" and have the site SSL encrypted and secure.
For this purpose, we need to configure 2 main things,
1. A local name resolving DNS server
2. Creating signed SSL certificates to ensure security

Install Python and Django and Make verysecureserver1

Install Python and Django from terminal in Linux Mint. Then run our server following below command.

Command:

1. sudo apt install python3-pip
2. sudo apt install python3-django
3. sudo python3 manage.py migrate
4. sudo python3 manage.py runserver 192.168.0.193:80

Figure 1: Runing Our Server

Configure Local DNS Server:

2 main libraries on linux are needed to configure the DNS name lookup server. These are bind9 and dnsutils. To install these, run

Command:

1. sudo apt install bind9
2. sudo apt install dnsutils

After installing is done, we need to restart the bind9 service.

Command:

1. sudo systemctl restart bind9.service

After that the /etc/resolve.conf file should be edited to make sure our localhost is being used as the DNS server.

Command:

1. sudo nano /etc/resolv.conf

Add these lines at the end of the file and save:

```
nameserver 192.168.0.193
options edns0 trust-ad
search localdomain
```

Next a named.conf file should be created at the /etc/ directory and a DNS caching server should be configured so that our machine isn't completely cut off from the internet. This ensures that our local DNS server is listening to port 53 of localhost for any DNS lookup, and the forwarders are using 8.8.8.8 and 8.8.4.4 which is Google's DNS for looking up addresses which are not available locally.

Command:

1. sudo nano /etc/named.conf

Add these lines of the file and save:

```
//
//named.conf
// Provided by Red Hat bind package to configure the ISC BIND named(8) DNS
// server as a caching only name server (as a localhost DNS resolver only).
//    See    /usr/share/doc/bind*/sample/    for    example    named configuration files.
//

options {
        listen-on port 53 { 127.0.0.1;};
//      listen-on-v6 port 53 { ::1; };
        forwarders { 8.8.8.8; 8.8.4.4; };
```

```
        directory       "/var/named";
        dump-file       "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
        memstatistics-file "/var/named/data/named_mem_stats.txt";
        allow-query     { localhost; 192.168.0.0/24, 127.0.0.1 };
        recursion yes;


        dnssec-enable yes;
        dnssec-validation yes;
        dnssec-lookaside auto;


        /* Path to ISC DLV key */
        bindkeys-file "/etc/named.iscdlv.key";


        managed-keys-directory "/var/named/dynamic";
};
logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};
zone "." IN {
        type hint;
        file "named.ca";
};
include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
```

For now, the local DNS server only works as a caching server. To have it working as a complete lookup and name resolving server, next let us create a zone file verysecureserver1.com.zone

Command:

1. sudo nano /etc/bind/verysecureserver1.com.zone

Add these lines of the file and save:

```
; Authoritative data for verysecureserver1.com zone
;
$TTL 1D
@ IN SOA verysecureserver1.com root.verysecureserver1.com. (
2022041301 ; serial
1D ; refresh
1H ; retry
1W ; expire
```

```
3H ) ; minimum
$ORIGIN verysecureserver1.com.
Verysecureserver1.com. IN NS verysecureserver1.com.
@ IN A 192.168.0.193
```

Here, it is being said to redirect any lookups for a verysecureserver1.com to 127.0.0.1; Which is the address of the server we are using.
Next add the path of the file to named.conf.local in the same directory.

Command:

1. sudo nano /etc/bind/named.conf.local

Add these lines of the file and save:

```
zone "verysecureserver1.com" IN {
        type master;
        file "/etc/bind/verysecureserver1.com.zone";
};
```

Now the DNS server should be completely configured. Just restart the named service and check if the server is working correctly. If everything has gone properly looking up verysecureserver.com should redirect to 127.0.0.1 now.

Command:

1. dig verysecureserver1.com
2. nslookup verysecureserver1.com

The work with the DNS server is now done. Next, we need to create the actual certificate authorities and certificates to secure the server.

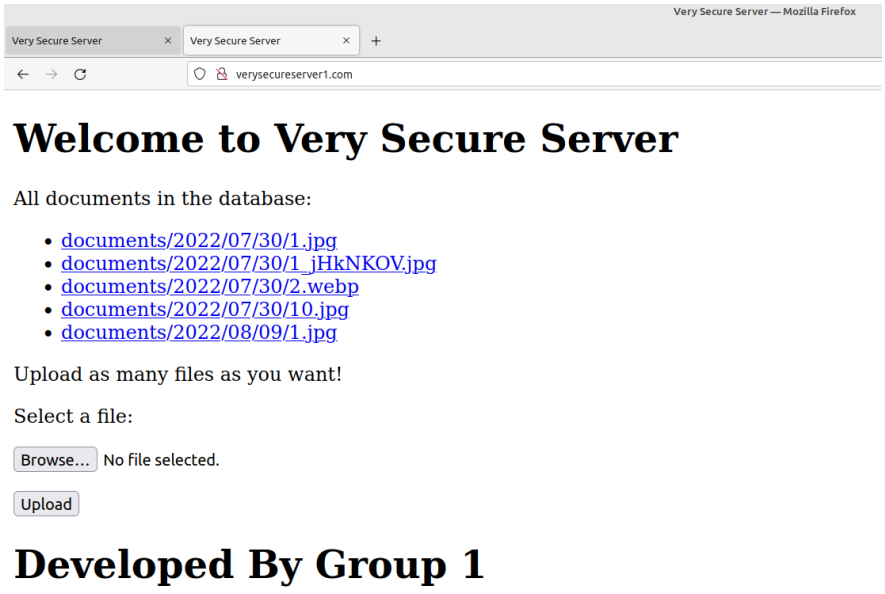

Figure 2: Server Without Lock Icon

Configuring Certificates with OpenSSL:

To create the certificates with OpenSSL in this project first we need to select a base directory for our work and then create some sub directories. So first, open the command line in a suitable directory and then run.

Command:

1. mkdir {root-ca,sub-ca,server}
2. mkdir {root-ca,sub-ca,server}/{private,certs,newcerts,crl,csr}

to create the 3 directories and their required subdirectories. root-ca for the top-level certificate authority, sub-ca for intermediate certificate authority and the server for the server files.

Then execute these commands to create the required files in these directories

Command:

1. touch root-ca/index
2. touch sub-ca/index

Then we generate the RootCA (Acme-RootCA) and Sub-CA(Acme) private keys as well as the local server's private key which can be done using the genrsa command.

Command:

1. openssl genrsa -aes256 -out root-ca/private/ca.key 4096
2. openssl genrsa -aes256 -out sub-ca/private/sub-ca.key 4096
3. openssl genrsa -out server/private/server.key 2048

Since we are using aes256 encryption on CA and Sub-CA for stronger protection a password will be needed. This password should be remembered as it will be needed for signing certificates later.

Next, we need move to the root-ca directory and create the top-level certificate for the CA from the previously generated key. A root-ca.conf configuration file will be needed in the directory and we need to create a file.

File Code:

```
[ca]
#/root/ca/root-ca/root-ca.conf
#see man ca
default_ca    = CA_default

[CA_default]
dir     = /home/lata/openssl/root-ca
certs     =  $dir/certs
crl_dir    = $dir/crl
new_certs_dir   = $dir/newcerts
database   = $dir/index
serial    = $dir/serial
RANDFILE   = $dir/private/.rand
```

```
private_key   = $dir/private/ca.key
certificate   = $dir/certs/ca.crt

crlnumber   = $dir/crlnumber
crl    =  $dir/crl/ca.crl
crl_extensions   = crl_ext
default_crl_days    = 30

default_md   = sha256

name_opt   = ca_default
cert_opt   = ca_default
default_days   = 365
preserve   = no
policy    = policy_strict

[ policy_strict ]
countryName   = supplied
stateOrProvinceName  =  supplied
organizationName  = supplied
organizationalUnitName  =  optional
commonName   =  supplied
emailAddress   =  optional

[ policy_loose ]
countryName   = optional
stateOrProvinceName  = optional
localityName   = optional
organizationName  = optional
organizationalUnitName   = optional
commonName   = supplied
emailAddress   = optional

[ req ]
# Options for the req tool, man req.
default_bits   = 2048
distinguished_name  = req_distinguished_name
string_mask   = utf8only
default_md   =  sha256
# Extension to add when the -x509 option is used.
x509_extensions   = v3_ca

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
```

```
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address
countryName_default  = BD
stateOrProvinceName_default = Dhaka
0.organizationName_default = Acme

[ v3_ca ]
# Extensions to apply when createing root ca
# Extensions for a typical CA, man x509v3_config
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid:always,issuer
basicConstraints  = critical, CA:true
keyUsage   =  critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions to apply when creating intermediate or sub-ca
# Extensions for a typical intermediate CA, same man as above
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid:always,issuer
#pathlen:0  ensures  no  more  sub-ca  can  be  created  below  an
intermediate
basicConstraints  = critical, CA:true, pathlen:0
keyUsage   = critical, digitalSignature, cRLSign, keyCertSign
crlDistributionPoints           = @crl_dist_points

[ server_cert ]
# Extensions for server certificates
basicConstraints  = CA:FALSE
nsComment   =  "OpenSSL Generated Server Certificate"
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid,issuer:always
keyUsage   =  nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage  = serverAuth
subjectAltName                          = @alt_names


[alt_names]
DNS.1 = verysecureserver1.com
DNS.2 = www.verysecureserver1.com
IP.1 = 192.168.0.193

[crl_dist_points]
URI.0 = http://localhost:8086/rev.crl
```

Command:

1. cd root-ca

2. openssl req -config root-ca.conf -key private/ca.key -new -x509 - days 7200 -sha256 -extensions v3_ca -out certs/ca.crt

Here the days mean how long the certificate will remain valid for.
We will be asked for some information including country code, state and common name. The common name is very critical, and it should be set to what we want the domain to be. Here we have named it Acme-RootCA.

Then we change to the sub-ca directory and generate the CSR file for signing the sub-ca certificate by RootCA. It is similar to creating the RootCA and we need to provide similar information for SubCA. The sub-ca.conf file should be copied to the directory too before executing the command. we need to create a file.

File Code:

```
[ca]
#C:/openssl/root-ca/root-ca.conf
#see man ca
default_ca    = CA_default

[CA_default]
dir     = /home/lata/openssl/sub-ca
certs     =  $dir/certs
crl_dir    = $dir/crl
new_certs_dir   = $dir/newcerts
database   = $dir/index
serial    = $dir/serial
RANDFILE   = $dir/private/.rand

private_key   = $dir/private/sub-ca.key
certificate   = $dir/certs/sub-ca.crt

crlnumber   = $dir/crlnumber
crl    =  $dir/crl/ca.crl
default_crl_days    = 30

default_md   = sha256

name_opt   = ca_default
cert_opt   = ca_default
default_days   = 365
preserve   = no
policy    = policy_loose

[ policy_strict ]
countryName   = supplied
stateOrProvinceName  =  supplied
organizationName  = supplied
organizationalUnitName  =  optional
```

```
commonName   =  supplied
emailAddress   =  optional

[ policy_loose ]
countryName   = optional
stateOrProvinceName  = optional
localityName   = optional
organizationName  = optional
organizationalUnitName   = optional
commonName   = supplied
emailAddress   = optional

[ req ]
# Options for the req tool, man req.
default_bits   = 2048
distinguished_name  = req_distinguished_name
string_mask   = utf8only
default_md   =  sha256
# Extension to add when the -x509 option is used.
x509_extensions   = v3_ca


[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address
countryName_default  = BD
stateOrProvinceName_default = Dhaka
0.organizationName_default = Acme

[ v3_ca ]
# Extensions to apply when createing root ca
# Extensions for a typical CA, man x509v3_config
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid:always,issuer
basicConstraints  = critical, CA:true
keyUsage   =  critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions to apply when creating intermediate or sub-ca
# Extensions for a typical intermediate CA, same man as above
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid:always,issuer
```

```
#pathlen:0 ensures no more sub-ca can be created below an
intermediate
basicConstraints  = critical, CA:true, pathlen:0
keyUsage   = critical, digitalSignature, cRLSign, keyCertSign
crlDistributionPoints           = @crl_dist_points

[ server_cert ]
# Extensions for server certificates
basicConstraints  = CA:FALSE
nsComment   =  "OpenSSL Generated Server Certificate"
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid,issuer:always
keyUsage   =  nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage  = serverAuth
subjectAltName                          = @alt_names


[alt_names]
DNS.1 = verysecureserver1.com
DNS.2 = www.verysecureserver1.com
IP.1 = 192.168.0.193

[crl_dist_points]
URI.0 = http://localhost:8086/rev.crl
```

Command:

1. cd ../sub-ca/
2. openssl req -config sub-ca.conf -new -key private/sub-ca.key -sha256 -out csr/sub-ca.csr

We need to change back to the root-ca directory again for signing it with RootCA certificate using the “cd” command after generation of the CSR. It is important the previously generated index file in root-ca and sub-ca directory is an empty text file or the ca command might not work.

Command:

1. cd ../root-ca
2. openssl ca -config root-ca.conf -extensions v3_intermediate_ca -days 3650 -notext -in ../sub-ca/csr/sub-ca.csr -out ../sub-ca/certs/sub-ca.crt -rand_serial

And after that we need to configure the private key and certificate for the server. A server.conf file will be needed.
we need to create a file.

File Code:

```
[ca]
#C:/openssl/root-ca/root-ca.conf
#see man ca
default_ca    = CA_default
```

```
[CA_default]
dir     = /home/lata/openssl/server
certs     =  $dir/certs
crl_dir    = $dir/crl
new_certs_dir   = $dir/newcerts
database   = $dir/index
serial    = $dir/serial
RANDFILE   = $dir/private/.rand

private_key   = $dir/private/sub-ca.key
certificate   = $dir/certs/sub-ca.crt

crlnumber   = $dir/crlnumber
crl    =  $dir/crl/ca.crl
crl_extensions   = crl_ext
default_crl_days    = 30

default_md   = sha256

name_opt   = ca_default
cert_opt   = ca_default
default_days   = 365
preserve   = no
policy    = policy_loose

[ policy_strict ]
countryName   = supplied
stateOrProvinceName  =  supplied
organizationName  = supplied
organizationalUnitName  =  optional
commonName   =  supplied
emailAddress   =  optional

[ policy_loose ]
countryName   = optional
stateOrProvinceName  = optional
localityName   = optional
organizationName  = optional
organizationalUnitName   = optional
commonName   = supplied
emailAddress   = optional

[ req ]
# Options for the req tool, man req.
default_bits   = 2048
distinguished_name  = req_distinguished_name
string_mask   = utf8only
```

```
default_md   =  sha256
# Extension to add when the -x509 option is used.
x509_extensions   = v3_ca

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address
countryName_default  = BD
stateOrProvinceName_default = Dhaka
0.organizationName_default = Acme

[ v3_ca ]
# Extensions to apply when createing root ca
# Extensions for a typical CA, man x509v3_config
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid:always,issuer
basicConstraints  = critical, CA:true
keyUsage   =  critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions to apply when creating intermediate or sub-ca
# Extensions for a typical intermediate CA, same man as above
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid:always,issuer
#pathlen:0  ensures  no  more  sub-ca  can  be  created  below  an
intermediate
basicConstraints  = critical, CA:true, pathlen:0
keyUsage   = critical, digitalSignature, cRLSign, keyCertSign
crlDistributionPoints           = @crl_dist_points

[ server_cert ]
# Extensions for server certificates
basicConstraints  = CA:FALSE
nsComment   =  "OpenSSL Generated Server Certificate"
subjectKeyIdentifier  = hash
authorityKeyIdentifier  = keyid,issuer:always
keyUsage   =  nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage  = serverAuth
subjectAltName                          = @alt_names


[alt_names]
```

```
DNS.1 = verysecureserver1.com
DNS.2 = www.verysecureserver1.com
IP.1 = 192.168.0.193

[crl_dist_points]
URI.0 = http://localhost:8086/rev.crl
```

Command:

1. cd ../server
2. openssl req -config server.conf -key private/server.key -new -sha256 -out csr/server.csr

And finally, to sign the certificate using the SubCA, the openSSL cd command will be used again.

Command:

1. cd ../sub-ca
2. openssl ca -config sub-ca.conf -extensions server_cert -days 365 -notext -in ../server/csr/server.csr -out ../server/certs/server.crt -rand_serial

And with that our certificate creation process is complete. We can merge the SubCA and the actual server certificate file using the cat command and check the final created certificate with the x509 command to ensure it is all the required information

Command:

1. cd ..
2. cat ./server/certs/server.crt ./sub-ca/certs/sub-ca.crt > chained.crt

Adding Acme-RootCA as Trusted Root CA, Configuring and Verifying Secure Server

Now the after certificate-creation tasks. First the Acme-RootCA certificate should be added as a trusted Certificate Authority in the machine. For linux machines, the easier method to add certificates as trusted Root CA is to directly add them from browsers. It can be done by going from browser settings.

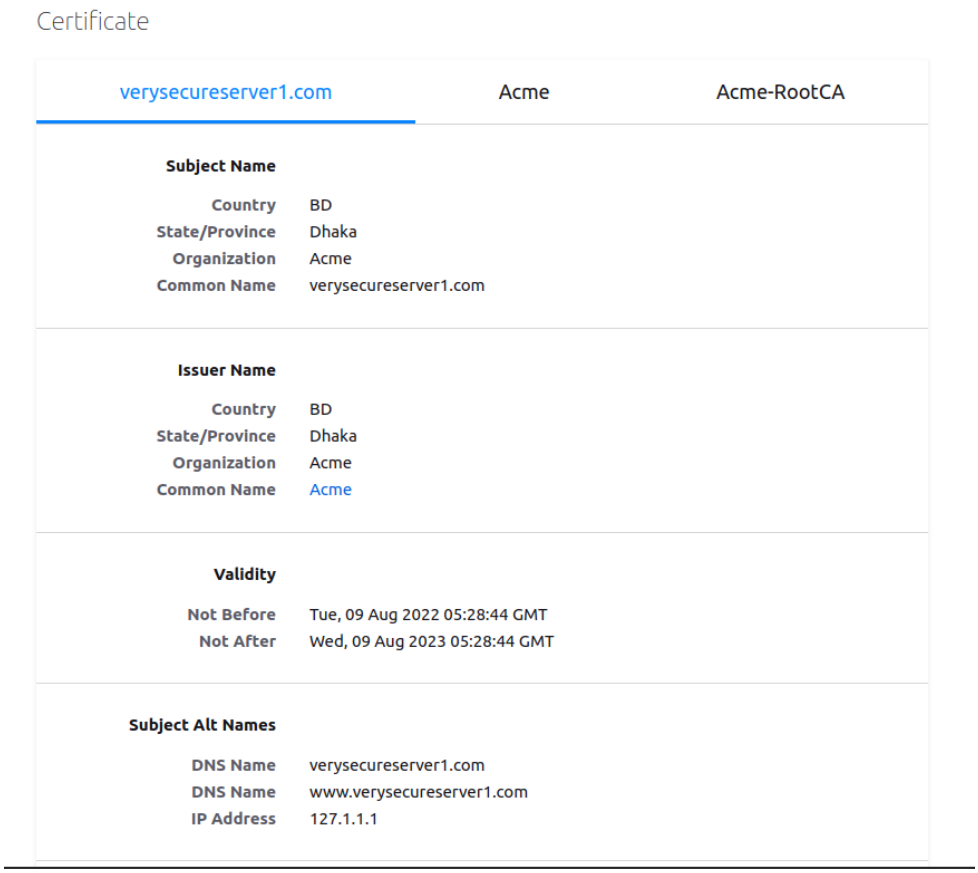

Figure 3: Show our Certificate in Website

Then we just need to configure our local server to use these generated server certificate and the Intermediate certificate to verify it. I am using a Django file upload server so for this first we need to install some new packages like django-sslserver and pyOpenSSL. And we also need to configure Django settings.py to add "sslserver" to the installed apps list**.**

Command:

1. sudo pip install pyOpenSSL
2. sudo pip install django-sslserver

Then we just need to copy the server private key (server\private\server.key) and signed server certificate and sub-ca certificate chained file chained.crt inside the projects root directory. This server can then be securely deployed using the runsslserver command.

Command:

1. sudo python3 manage.py runsslserver 192.168.0.193:443 –certificate chained.crt --key server.key

If everything was done correctly, now we should have our very own secure file upload and download server with complete certificate chains in place.

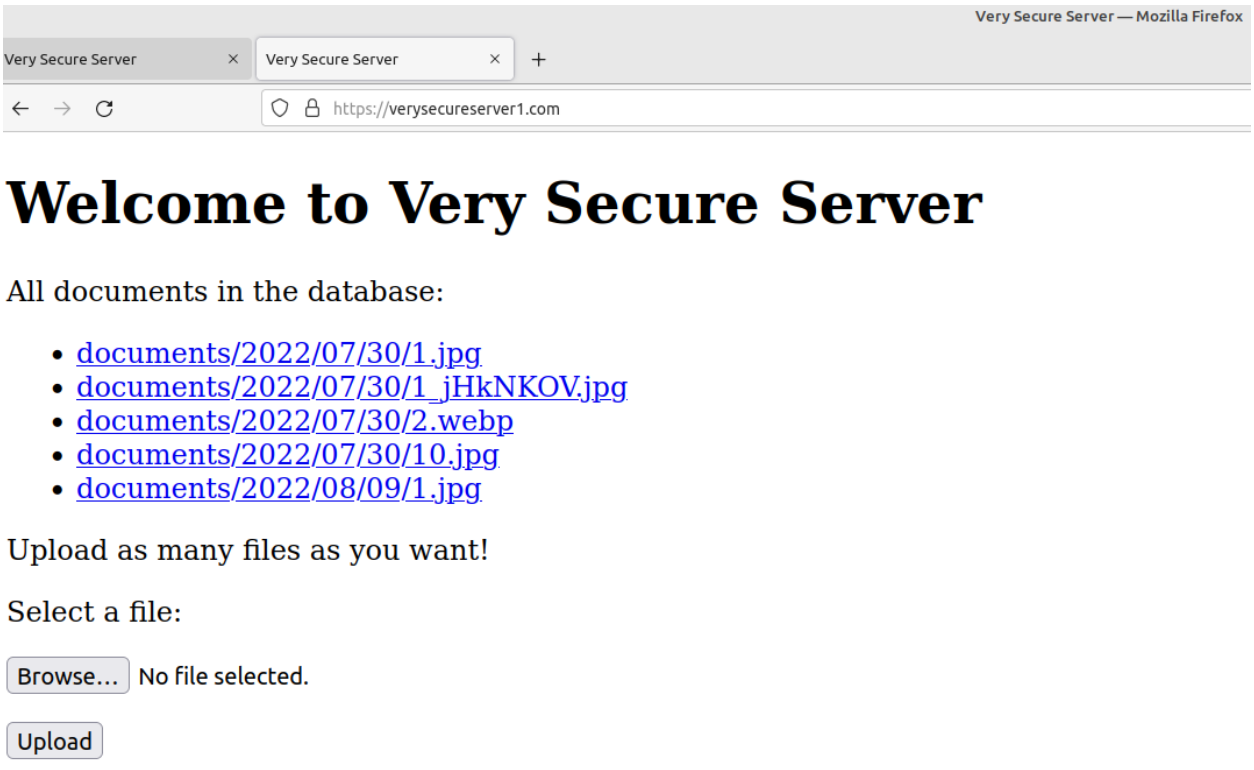

Figure 4: Server with Lock Icon

From the given figures it can be seen that the local website is valid and secure, and its authenticity is ensured by multi-level certificate authorities. Thus, no other site can pretend to be this site and this site cannot claim to not be this site thanks to this verified chain of certificates.

Revoking the Certificate:

Finally, if the intermediate CA wants to revoke the certificate granted to verysecureserver.com, it can also be done quite easily using openSSL. We just need to change to the sub-ca directory and revoke the certificate by pointing to it.

Command:

1. cd sub-ca
2. openssl ca -config sub-ca.conf -revoke ../server/certs/server.crt

And then to distribute the information that his server's certificate has been revoked, a certificate revocation list needs to be generated. A number should be given as crlnumber so that the serial of certificate can be kept count of. We can even check if the revocation and CRL generation work was done properly using the CRL command.

Command:

1. nano crlnumber
2. openssl ca -config sub-ca.conf -gencrl -out crl/rev.crl

And then this CRL can easily be posted to the linked CRL distribution address attached to the Intermediate CA.

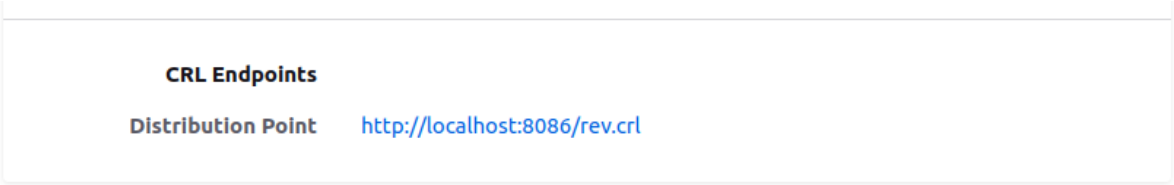

Figure 5: Revoke file

And that concludes the work of this project. Going through following these steps we have successfully managed to secure a local file upload server using SSL certificates and have managed to do various other things with it.

Wireshark Result:

From the given figures the local website is valid and secure, and its authenticity is ensured by multi-level certificate authorities. Thus, no other site can pretend to be this site and this site cannot claim to not be this site thanks to this verified chain of certificates. The security can be further verified using Wireshark. From the Wireshark output we can see that the server is clearly using TLSv1.2 protocol for transferring data securely.

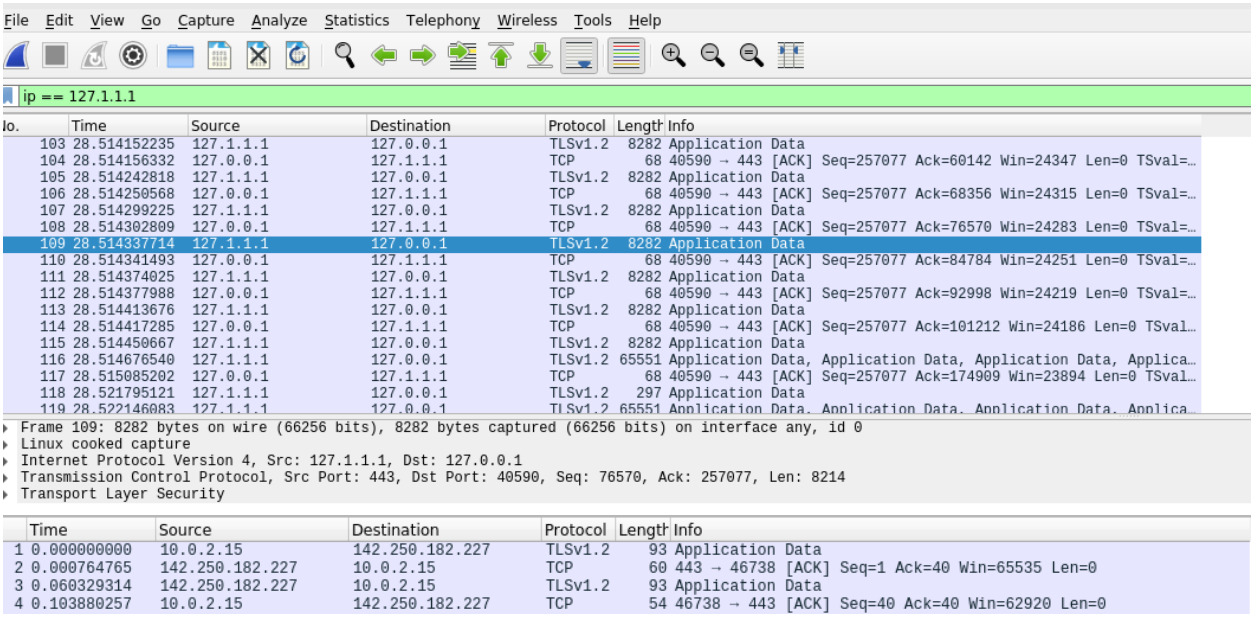

Figure 6: TLS and TCP

For Client Server Configuration

First of all we need to edit network manager file in etc, netplan folder.
We need to write following code:
Command: sudo nano /etc/netplan/1-network-manager-all.yaml

```
# Let NetworkManager manage all devices on this system
network:
    version: 2
    renderer: NetworkManager
    ethernets:
        enp0s3:
            dhcp4: no
            addresses: [192.168.0.191/24]
            routes:
                - to: default
                  via: 192.168.0.1
            nameservers:
                addresses: [192.168.0.193]
                search: [verysecureserver1.com]
```

Then we need to restart netplan
Command: sudo netplan try

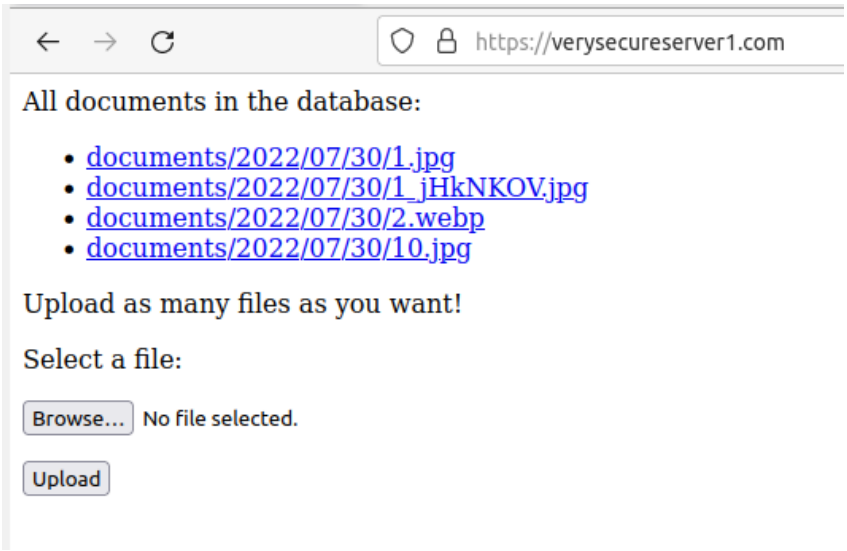

Figure 7: Lock in Client Machine