EAST WEST UNIVERSITY

Mini Project - 1

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

Course Code: CSE487

Course Title: Computer and Cyber Security

Section: 01

Semester: Summer-22

Submitted By,

Name: Al Shammas

ID: 2018-3-60-016

Name: Fardin Alam

ID: 2018-3-60-030

Name: Tahmid Alam

ID: 2018-3-60-031

Submitted To,

Rashedul Amin Tuhin

Senior Lecturer

Department of Computer Science & Engineering

Submission Date: 25/08/2022

1: First, we have to open terminal and enter our system as root user.

sudo -i

2: Create all the necessary CA directories.

mkdir -p ca/{root-ca,sub-ca,server}/{private,certs,newcerts,crl,csr}

3: Check all directories created successfully with tree command.

tree

4: Give read, write and execute permission for only root user.

chmod -v 700 ca/{root-ca,sub-ca,server}/private

5: Create index file

touch ca/{root-ca,sub-ca}/index

6: Generate serial file for root-ca and sub-ca

openssl rand -hex 16 > ca/root-ca/serial

openssl rand -hex 16 > ca/sub-ca/serial

7: Generate private keys for root-ca, sub-ca and server

openssl genrsa -aes256 -out root-ca/private/ca.key 4096

openssl genrsa -aes256 -out sub-ca/private/sub-ca.key 4096

openssl genrsa -out server/private/server.key 2048

8: Create root-ca.conf file and paste some line of codes

vim root-ca/root-ca.conf

Paste this following code inside the root-ca.conf file:

[ca]

#/root/ca/root-ca/root-ca.conf

```
#see man ca
default_ca    = CA_default

[CA_default]
dir     = /root/ca/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  = match
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 = Very Secure Ltd

[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

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

9: Change directory to root-ca

cd root-ca/

10: Generate root-ca certificate using root-ca.conf file and ca.key file
openssl req -config root-ca.conf -key private/ca.key -new -x509 -days 7305 -sha256 -extensions
v3_ca -out certs/ca.crt

11: Check whether root-ca certificate created successfully or not.

openssl x509 -noout -in certs/ca.crt -text

12: Change directory to sub-ca

cd ../sub-ca/

13: Create sub-ca.conf file and paste some line of codes

vim sub-ca.conf

Paste this following code inside the sub-ca.conf file:

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

[CA_default]    dir               =
/root/ca/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 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 =

match

organizationalUnitNam

e = 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 = Very Secure Ltd
[ 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
[ server_cert ]
# Extensions for server certificates basicConstraints   =
CA:FALSE  nsCertType      =  server  nsComment      =
"OpenSSL        Generated        Server        Certificate"
```

subjectKeyIdentifier = hash authorityKeyIdentifier = keyid,issuer:always keyUsage = critical, digitalSignature, keyEncipherment extendedKeyUsage = serverAuth

14 Generate sub-ca certificate signing request using sub-ca.conf file and sub-ca.key file

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

15: Back to previous working directory.

cd -

16: Accept sub-ca certificate signing request using root-ca.conf file and generate sub-ca.crt

openssl ca -config root-ca.conf -extensions v3_intermediate_ca -days 3652 -notext -in ../subca/csr/sub-ca.csr -out ../sub-ca/certs/sub-ca.crt

17: Check whether sub-ca certificate generated successfully or not.

openssl x509 -noout -text -in ../sub-ca/certs/sub-ca.crt

18: Go to server directory for generating csr file

cd ../server/

19: Generate server certificate signing request using server.key file

openssl req -key private/server.key -new -sha256 -out csr/server.csr

20: Go to sub-ca directory for signing server certificate.

cd ../sub-ca/

21: Accept server certificate signing request using sub-ca.conf file and generate server.crt

openssl ca -config sub-ca.conf -extensions server_cert -days 365 -notext -in

../server/csr/server.csr -out ../server/certs/server.crt

22 Go to server/certs directory for chaining both server.crt and sub-ca.crt files.

cd ../server/certs/

23: Concatenate both server.crt and sub-ca.crt files and name as chained.crt.

cat server.crt ../../sub-ca/certs/sub-ca.crt > chained.crt

24: Go back to server directory

cd ..

25: Append localhost IP address and domain in /etc/hosts file.

echo "127.0.0.2 www.verysecureserver.com" >> /etc/hosts

26: Start ping command to check whether localhost domain returns the correct IP or not.

ping www.verysecureserver.com

27: Use the SSL port 443 for our server

Openssl s_server -accept 443 -www -key private/server.key -cert certs/server.crt -CAfile ../sub-ca/certs/sub-ca.crt

Now we cannot get access this terminal window anymore.

28: So, we have to open another terminal window and enter as a root user.

sudo -i

29 With this command, we can see all the ports which are now used in our system and also we can verify whether 443 port is used or not.

ss -ntl

30: Use curl command to check whether our localhost is verified successfully or not.

curl https://www.verysecureserver.com

This command gives us an error: curl failed to verify the legitimacy of the server …
This error occurs because we have created our own RootCA, so our OS does not trust the RootCA.

31: Copy our ca.crt certificate to our system's ca-certificate directory

cp ca/root-ca/certs/ca.crt /usr/local/share/ca-certificates/

32: Update the system's ca-certificate directory

update-ca-certificates -v

33: Now again use the curl command
!cu

This time curl command gives no error. So trust relationship now has been established.

Now we can stop our old terminal window with CTRL+C command.

For testing our localhost in a real web server, we have to install a server

34: Install nginx server

sudo apt update

sudo apt install nginx

35 Commands for check nginx activation status, stop nginx and start nginx

sudo systemctl status nginx

sudo systemctl stop nginx

sudo systemctl start nginx

36: Edit nginx.conf file and paste some line of codes

vim /etc/nginx/nginx.conf

Inside the nginx.conf file paste the following code in HTTP section:

```
# HTTPS server server
{
        listen          443 ssl;
        server_name     www.verysecureserver.com;
        ssl_certificate         /root/ca/server/certs/chained.crt;
        ssl_certificate_key     /root/ca/server/private/server.key;
        ssl_protocols           TLSv1.2;
    ssl_session_cache shared:SSL:1m;
ssl_session_timeout  5m;  ssl_ciphers
HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;       location / {
root  /srv/www/htdocs/;             index
index.html index.htm;
```

```
        }
}
include vhosts.d/*.conf;
```

37 echo the line in our localhost index.html file

```
echo "" > /srv/www/htdocs/index.html
```

38: Use curl command to enter our localhost.

```
curl https://www.verysecureserver.com
```

Now, we have to tell our browser to trust our CA certificate:

39: Copy root-ca pem file in our user account.

```
cp /root/ca/root-ca/newcerts/file.pem ~tahmid/
```

40: Finally, import the pem file in our browser.

Go to browser's settings → Privacy and Security → Security → Manage certificates → Authorities section → Click Import.

Import the pem file and give permission only identify websites.

DONE!

Restart the system.

Now go to our localhost: https://www.verysecureserver.com We can see the SSL Certificate in our localhost.

Adding firewall:

ufw status

ufw enable

ufw status numbered

ufw allow 443

ufw allow 80

ufw allow 53

: