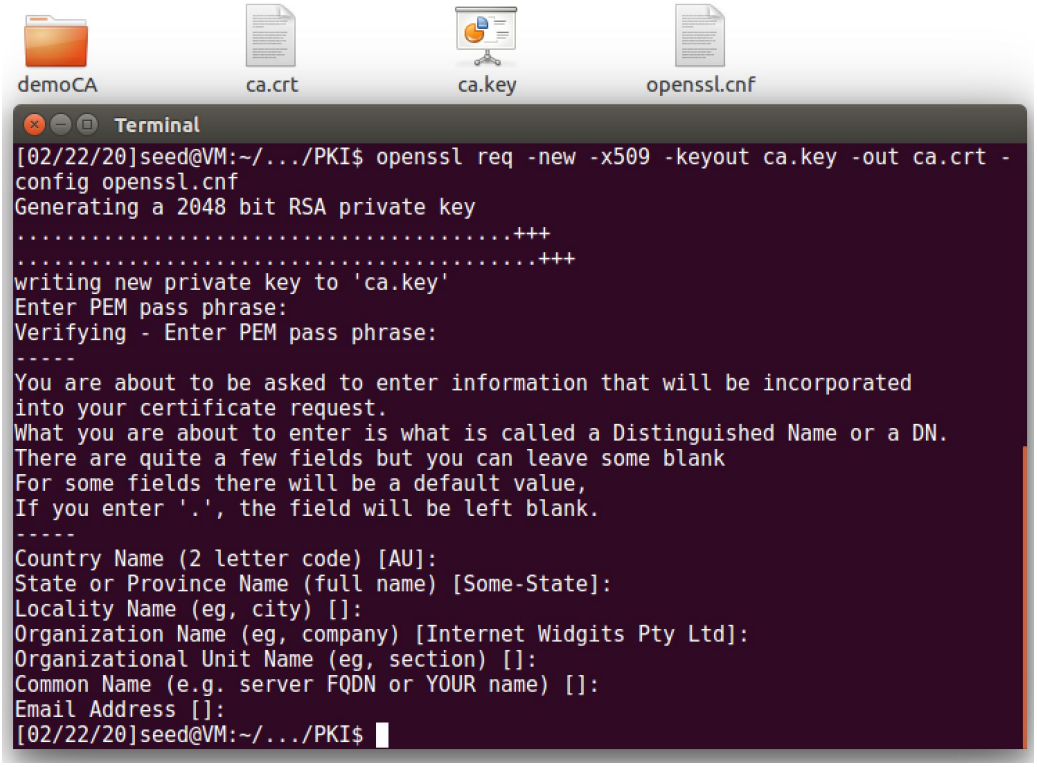
Midterm Crypto PKI Lab Report

Parag Mhatre

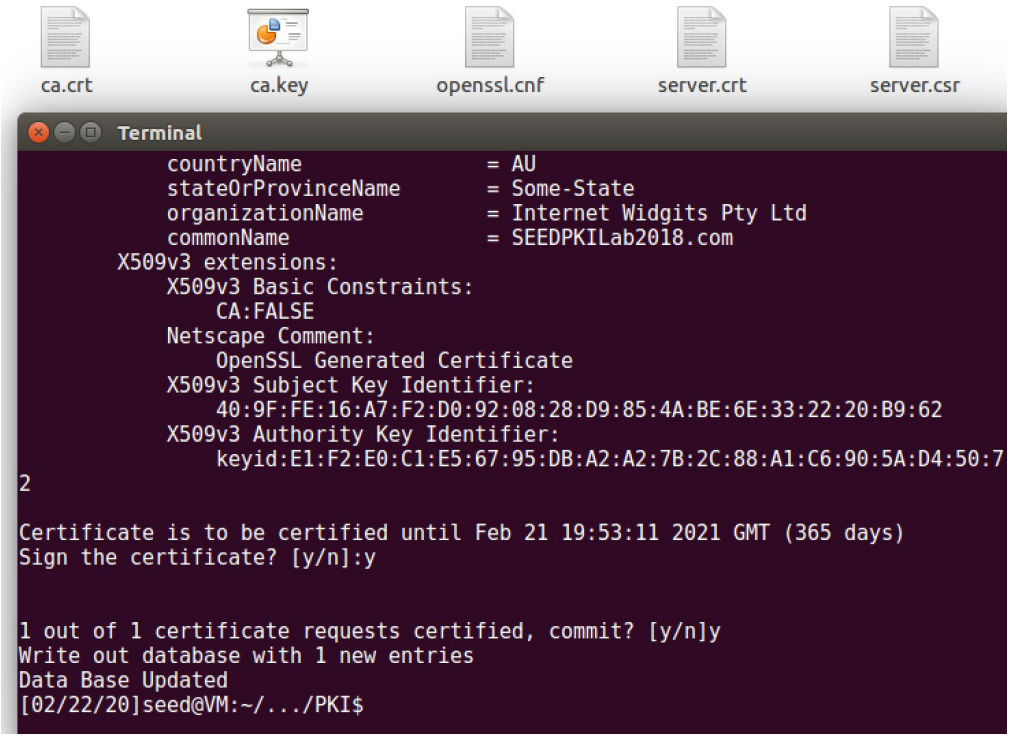
Task 1

In this task, we create a Certificate Authority (CA) to self sign our certificates. The first thing that we do is to get our own `openssl.cnf` file. We do this by copying the file from `/usr/lib/ssl/`. The next thing that we do is to create a file tree as shown on the right. The content of the serial file is `1000`. The file `index.txt` is an empty file. We have to provide a password while creating the CA. This step outputs two files - `ca.key` and `ca.crt`.



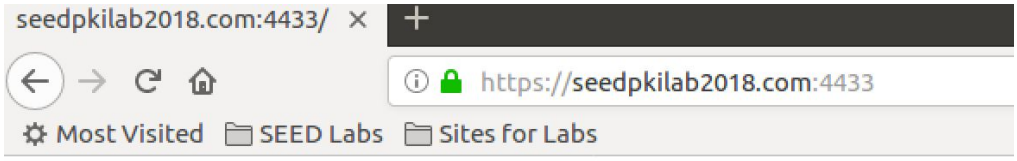
Task 2

In this task, we create a certificate for the website - SEEDPKILab2018.com. To do this, we first create a private/public key pair which is outputted in `server.key`. The next step is to generate a certificate signing request for the CA to sign the certificate. We do this and the request is generated in a file called `server.csr`. The final step is to actually sign the certificate using the CA that we created. We change OpenSSL policy to a less restrictive policy called policy_anything and sign the certificate thereby creating a file called server.crt.



Task 3

We deploy a certificate in an HTTPS server provided in openssl in this task. We want to host the website with the domain name - SEEDPKILab2018.com. So, first thing that we do is add an entry in the hosts file in `/etc/hosts`. Then we create a `server.pem` by concatenating `server.key` and `server.crt`. This .pem file is required to host an HTTPS website. Then we test the file by hosting the server via the openssl's built-in feature. The server is started on port 4433. When we check the website via a browser, we see an warning - *"seedpkilab2018.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown".* This is because the browser does not have our custom CA as a registered root CA like other popular CAs like VeriSign. Once we add the CA to the registered CAs on the browser, the browser will recognize the CA and open the website.



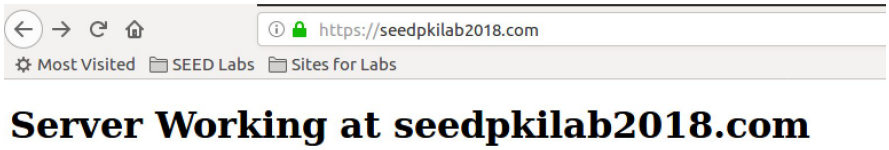

```
s_server -cert server.pem -www
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1/SSLv3:ECDHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA384  TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA     TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA
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
```

When we modify a single byte in server.pem, openssl server shoots an error and it doesn't start successfully.

If we use localhost instead of seedpkilab2018.com, then the browser shoots up an warning. I think that this is because the domain name is different than that of in the pem file.

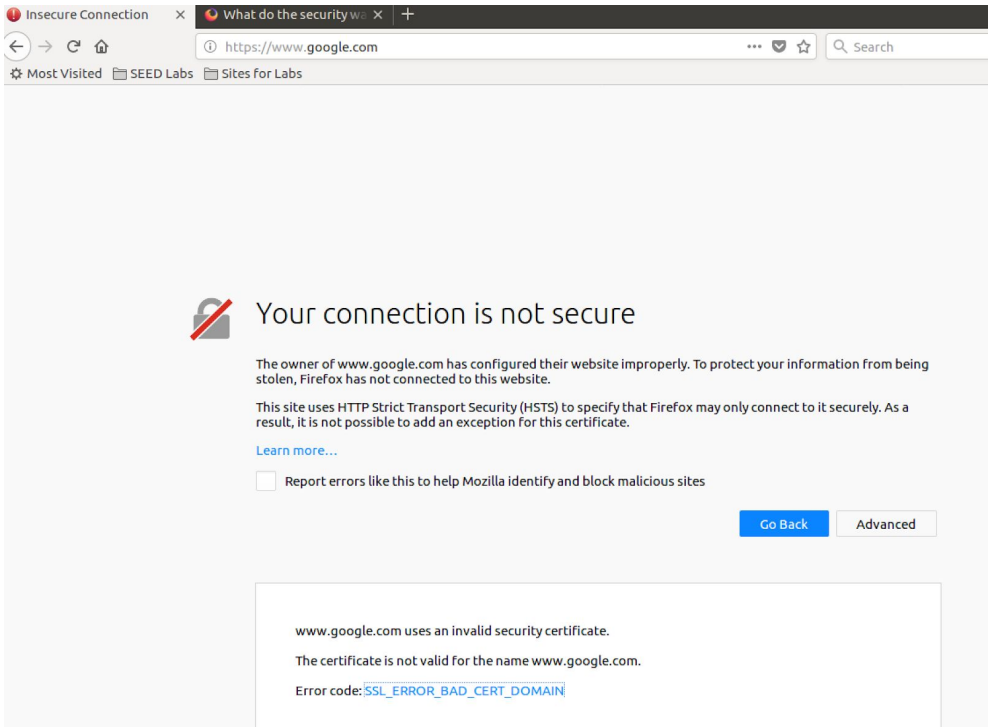
Task 4

In this task, we deploy an HTTPS website using an certificate via an apache web server. We modify the `000-default.conf` and `default-ssl.conf` available in the `/etc/apache2/sites-available`. We set the ServerName attribute as the domain name of the website and create a new folder for the website in `/var/www/`. When the configuration is done, we can perform the configtest, enable ssl and restart the apache server. Once the server starts successfully, visit the website from your browser.



Task 5

In this task, we try to simulate a Man-in-the-middle attack scenario. We first setup a malicious website using the apache web server's virtual host feature. Now, to make the user land on our malicious website, we will simply manually poison the dns cache of the user by editing the `/etc/hosts` file. Now, when we try to browse the genuine website, we are directed to the malicious website and it is opened as shown below. We can see that it shoots an error saying that we have a bad cert domain error. Thus PKI saves the victim from visiting the malicious website as the website does not have a certificate that matches its domain name.



Task 6

In this task, we try to repeat the above, but considering that the attacker has compromised the root CA and he/she can generate certificates. So we generate a certificate that matches the domain of the malicious server and redo the whole man-in-the-middle simulation as above. We can see that the victim can now land on the malicious website successfully without him knowing that the website is malicious. Thus we understand that if an attacker gains access to the root CA, then he/she can circumvent the protections that the PKI ensures for the users.