Public Key Cryptography and PKI Lab

The development of this document is/was funded by three grants from the US National Science Foundation: Awards No. 0231122 and 0618680 from TUES/CCLI and Award No. 1017771 from Trustworthy Computing.

1 Overview

The learning objective of this lab is for students to become familiar with the concepts of Public Key Cryptography (PKC) and Public Key Infrastructure (PKI). Students will gain hands-on experience with public key certificates, certificate authorities, and authentication based on PKI.

2 Lab Environment

Installing OpenSSL. In this lab, we will use `openssl` commands and libraries. `openssl` has already been installed on the SEEDUbuntu 12.04 VM. However, you may install `openssl` and complete the lab on your own machine if you prefer.

3 Lab Tasks

3.1 Task 1: Become a Certification Authority (CA)

A Certification Authority (CA) is a trusted entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. A number of commercial CAs are treated as root CAs; VeriSign is the largest CA at the time of writing. Commercial CAs charge a fee to issue a certificate.

In this lab, we will create digital certificates, but we are not going to pay a commercial CA. We will become a root CA ourselves, and then use this CA to issue certificates for others (e.g. servers). For this first task, we will create a root CA and generate a certificate for the CA. Unlike other certificates, which are usually signed by another CA, the root CA's certificate is self-signed. Root CAs' certificates are pre-loaded into most operating systems, web browsers, and other software that rely on PKI. Root CA's certificates are trusted unconditionally.

The Configuration File `openssl.conf`. In order to use `OpenSSL` to create certificates, you have to have a configuration file. The configuration file usually has an extension `.cnf`. It is used by three `OpenSSL` commands: `ca`, `req` and `x509`. The manual page for `openssl.conf` can be found using Google search. Copy the configuration file from `/usr/lib/ssl/openssl.cnf` to your working directory. After copying the file, you will need to create several sub-directories and files as specified in the configuration file (look at the `[CA_default]` section):

```
dir             = ./demoCA          # Where everything is kept
certs           = $dir/certs        # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
new_certs_dir   = $dir/newcerts     # default place for new certs.

database        = $dir/index.txt    # database index file.
serial          = $dir/serial       # The current serial number
```

For the `index.txt` file, simply create an empty file. For the `serial` file, put a single number in string format (e.g. 1000) in the file. Once you have set up the configuration file `openssl.cnf`, you can create and issue certificates.

Certificate Authority (CA). As we described before, we need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as a root certificate. The following command will generate the self-signed certificate for the CA:

```
openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

You will be prompted for information and a password. Do not lose the password, as you will need it each time you want to use this CA to sign certificates for others. Additionally, do not choose a password that you use for any other systems, as you will be required to include the passwords in your lab submission. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command is stored in two files: `ca.key` and `ca.crt`. The file `ca.key` contains the CA's private key, while `ca.crt` contains the public key certificate.

3.2 Task 2: Create a Certificate for `PKILabServer.com`

Now that we have created a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called `PKILabServer.com`. There are three steps required to create a certificate for the customer:

Step 1: Generate public/private key pair. The company needs to first create its own public/private key pair. We can run the following command to generate an RSA key pair (both private and public keys). You will be required to provide a password to protect the keys. As before, do not choose a password that you use for any other systems. The keys will be stored in the file `server.key`:

```
openssl genrsa -aes128 -out server.key 1024
```

Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it must generate a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches the server's true identity).

```
openssl req -new -key server.key -out server.csr -config openssl.cnf
```

When prompted to provide a Common Name, enter the name of the customer's server, `PKILabServer.com`.

Step 3: Generate a Certificate. The CSR file must be signed by the CA to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, we will use our own CA to generate certificates:

```
openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \
           -config openssl.cnf
```

If `OpenSSL` refuses to generate certificates, it is very likely that the names in your requests do not match with those of CA. The matching rules are specified in the configuration file (look at the `[policy_match]` section). You can change the names of your requests to comply with the policy, or you can change the policy. The configuration file also includes another policy (called `policy_anything`), which is less restrictive. You can choose that policy by changing the following line:

```
"policy = policy_match"  change to "policy = policy_anything".
```

3.3 Task 3: Using PKI with Web Sites

In this task, we will explore how public-key certificates are used by web sites to secure web browsing. Our server's domain name is `PKILabServer.com`. To get our computers to recognize this domain name, we must add the following entry to `/etc/hosts`; this entry maps the domain name `PKILabServer.com` to our localhost (i.e., 127.0.0.1):

```
127.0.0.1  PKILabServer.com
```

Next, we will launch a simple web server with the certificate generated in the previous task. `openssl` allows us to start a simple web server using the `s_server` command:

```
# Combine the secret key and certificate into one file
cp server.key server.pem
cat server.crt >> server.pem

# Launch the web server using server.pem
openssl s_server -cert server.pem -www
```

By default, the server will listen on port `4433`. You can alter that using the `-accept` option. You can access the server using the following URL: `https://PKILabServer.com:4433/`. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following: *"This Connection is Untrusted. You have asked Firefox to connect securely to pkilabserver.com:4433, but we can't confirm that your connection is secure."*

Had this certificate been signed by VeriSign or another major commercial CA, we would not receive such an error message, because the commercial CA's certificate would most likely be preloaded into Firefox's certificate repository. Unfortunately, the certificate of `PKILabServer.com` is signed by our own CA (i.e., using `ca.crt`), and this CA is not recognized by Firefox. There are two ways to get Firefox to accept our CA's self-signed certificate:

- We can request that Mozilla include our CA's certificate in its Firefox software, so everybody using Firefox can recognize our CA. This is how the real CAs, such as VeriSign, get their certificates into Firefox. Unfortunately, our own CA does not have a large enough market for Mozilla to include our certificate, so we will not pursue this direction.

- **Load `ca.crt` into Firefox:** We can manually add our CA's certificate to the Firefox browser by clicking the following menu sequence:

  ```
  Edit -> Preference -> Advanced -> Certificates -> View Certificates
  ```

  You will see a list of certificates that are already accepted by Firefox. From here, we can import our own CA certificate. Import `ca.crt`, and select the following option: "Trust this CA to identify web sites." You will see that our CA's certificate is now in Firefox's list of the accepted certificates.

Now, point the browser to `https://PKILabServer.com:4433`. *Describe and explain your observations*. Complete the following additional tasks:

1. Modify a single byte of `server.pem`, restart the server, and reload the URL. *What do you observe?* Make sure you restore the original `server.pem` afterward. Note: the server may not be able to restart if certain locations in `server.pem` are corrupted; in that case, choose another place to modify.
2. Since `PKILabServer.com` points to the localhost, if we use `https://localhost:4433` instead, we will be connecting to the same web server. Try to connect to `https://localhost:4433`. *Describe and explain your observations.*

4 Submission

Your lab must be submitted on Blackboard and will include a lab report and a zip file containing the keys and certificates that you created. The lab report is to be submitted in DOC, DOCX, or PDF format and must include a detailed description of what you did and what you observed; you should explain any interesting or surprising observations. In particular, your lab report must include the following:

1. Screenshots of browser access to `https://PKILabServer.com:4433` *before and after* adding the CA certificate to Firefox's certificate repository.
2. Explanations for what is observed in each screenshot and why they are different.
3. The effect of modifying a byte in `server.pem` and an explanation for what you observed.
4. The effect of browsing to `https://localhost:4433` and an explanation for what you observed.
5. Descriptions of any obstacles encountered and their solutions.
6. Descriptions of any interesting or surprising observations and explanations of those observations.

The ZIP file must contain the key and certificate files you produced as well as a text README. Specifically, the zip file must include:

1. CA key and certificate – `ca.key`, `ca.crt`.
2. Server key, csr, certificate, and pem file – `server.key`, `server.csr`, `server.crt`, `server.pem`.
3. `README` file containing the passwords for the CA and server keys.