**USC UPSTATE**

**CSCI 455: Computer Security**

**Spring 2019**

**Lab 7**

**PKI lab**

**Task 1: Becoming a Certificate Authority (CA)**

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

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. You can get a copy of the configuration file from searching openssl.conf. After copying this file into your current directory, you need to create several sub-directories as specified in the configuration file (look at the [CA default] section):

**A screenshot of a cell phone

Description automatically generated**

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 the root certificate. You can run the following command to generate the self-signed certificate for the CA:

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

If you encounter an error message like this “problem creating object tsa\_policy1=1.2.3.4.1”, go to the file openssl.conf and comment the following three lines.

tsa\_policy1 = 1.2.3.4.1

tsa\_policy2 = 1.2.3.4.5.6

tsa\_policy3 = 1.2.3.4.5.7

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command are 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.

**Task 2: Creating a Certificate for PKILab.com**

Now, we become a root CA, we are ready to sign digital certificates for our customers. Our first customer is a company called PKILab.com. For this company to get a digital certificate from a CA, it needs to go through three steps.

**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 also be required to provide a password to encrypt the private key (using the AES-128 encryption algorithm, as is specified in the command option). The keys will be stored in the file server.key:

genrsa -aes128 -out server.key 1024

The server.key is an encoded text file (also encrypted), so you will not be able to see the actual content, such as the modulus, private exponents, etc. To see those, you can run the following command:

rsa -in server.key -text

**Step 2: Generate a Certificate Signing Request (CSR).** Once the company has the key file, it should generates a Certificate Signing Request (CSR), which basically includes the company’s public key. 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 with the server’s true identity). Please use PKILab.com as the common name of the certificate request.

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

It should be noted that the above command is quite similar to the one we used in creating the self-signed certificate for the CA. The only difference is the -x509 option. Without it, the command generates a request; with it, the command generates a self-signed certificate.

**Step 3: Generating Certificates.** The CSR file needs to have the CA’s signature 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 trusted CA to generate certificates. The following command turns the certificate signing request (server.csr) into an X509 certificate (server.crt), using the CA’s ca.crt and ca.key:

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".