UMBCCD Week 4 - Asymmetric Cryptography Lab

Background

Easily the most prevalent use of asymmetric cryptography in the real world is for HTTPS (through TLS).

When they need a TLS certificate, most people resort to copy-pasting `openssl` commands from the internet (that's what I'm doing to build this lab), but my hope is that after you do this lab, you'll have some idea as to what's going on.

Outline

You're going to create a CA certificate and a server certificate. You'll poke through the certificates with some commands that show you what fields are there. Then, in a snapshotted VM, you're going to install the CA cert as a trusted root CA and then configure Apache to use the server certificate. You'll then visit the site and see HTTPS work. Finally, you'll revert your VM, because having extra root CAs can be a security problem if you aren't careful.

Instructions

Take a snapshot of your Kali VM

We're going to be adding our test Root CA cert, and that's a security problem if we aren't careful. So take a snapshot and revert to it when you're done this lab.

Configure OpenSSL

`openssl` is probably the ugliest command line tool I've ever used that wasn't written by Oracle. Sorry about that. However, most stuff you'll find online uses it because it exists. So we'll be using it as well.

(Instructions shamelessly stolen and simplified from: https://jamielinux.com/docs/openssl-certificate-authority/create-the-root-pair.html). If you want to learn more about this, that site seems pretty informative. I've simplified the instructions slightly, because we don't need this to actually be secure. To be secure, you'd need hardware security modules and a whole lot more care than any of us have.

1. `mkdir ~/ca`

Everything related to this lab will be in that directory.

2. `cd ~/ca`
3. `mkdir certs private newcerts`
4. `chmod 700 private`
5. Set up some files:

```
touch index.txt
echo "01" > serial
```

The `openssl ca` scripts will update these files with metadata about the certs.

6. Now create `~/ca/openssl.cnf`. Most of this stuff is kinda random, don't sweat it too much. Just look at it and try to follow it.

```
[ ca ]
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = /home/USERNAME/ca
certs             = $dir/certs
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/private/ca.key.pem
certificate       = $dir/certs/ca.cert.pem

# SHA-1 is deprecated, so use SHA-2 instead.
default_md        = sha256

name_opt          = ca_default
cert_opt          = ca_default
default_days      = 375
preserve          = no
policy            = policy_strict

[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of `man ca`.
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

```
[ policy_loose ]
# Allow the intermediate CA to sign a more diverse range of certificates.
# See the POLICY FORMAT section of the `ca` man page.
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

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

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

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
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

# Optionally, specify some defaults.
countryName_default             = US
stateOrProvinceName_default     = MD
localityName_default            = Baltimore
0.organizationName_default      = UMBCCD Lab

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
```

```
[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
```

Create your CA cert

1. Generate a key: `openssl genrsa -aes256 -out private/ca.key.pem`
2. Create the certificate: `openssl req -config openssl.cnf -key private/ca.key.pem -new -x509 -days 3650 -sha256 -extensions v3_ca -out certs/ca.cert.pem`
3. Take a look at the new cert. Recognize anything? `openssl x509 -noout -text -in certs/ca.cert.pem`

Create your CSR

1. Generate a key: `openssl genrsa -out private/server.key.pem`
2. Create a certifcate signing request (CSR). Real CAs will ask you to do this. `openssl req -config openssl.cnf -key private/server.key.pem -new -sha256 -out server.csr`
3. Look at your request and see what's in it: `openssl req -noout -text -in server.csr`

Use your CA to sign your CSR

Now you get to pretend to be a CA and sign your CSR. Note that the CA doesn't actually need the private key matching the CSR.

1. Sign it! `openssl ca -config openssl.cnf -extensions server_cert -days 365 -notext -md sha256 -in server.csr -out server.crt.pem`
2. Your cert is in server.crt.pem. Let's look at it, shall we? `openssl x509 -noout -text -in server.crt.pem`

Look for: basicConstraints, DN, etc.

Configure Apache

1. Apache should be installed by default on Kali Linux

2. To add your certificate to the website and enable HTTPS, edit /etc/apache2/sites-available/default-ssl.conf and replace the following lines:

```
SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

to read:

```
SSLCertificateFile   /path/to/ca.cert.pem
SSLCertificateKeyFile /path/to/ca.key.pem
```

3. Run the following commands as root: `service apache2 restart` `apachectl startssl` `apachectl restart`
4. Visit `127.0.0.1:443`
5. Accept the certificate warning **(don't permanently confirm)** and proceed to the web page
6. Quit firefox

Now, we'll add the certificate

1. Open up firefox and follow the instructions here to permanently store the certificate in firefox - https://www.godaddy.com/help/importing-a-code-signing-certificate-into-firefox-4784
2. Visit `127.0.0.1:443`
3. Notice the difference?