Creating Certificate Authorities and self-signed SSL certificates

Following is a step-by-step guide to creating your own CA (Certificate Authority) -- and also self-signed SSL server certificates -- with openssl on Linux. Self-signing is the simpler route to take, but making one's own CA allows the signing of multiple server certificates using the same CA and involves only a few extra steps.

After using openssl to generate the necessary files, you'll need to integrate them into Apache. This process differs between Linux distros and versions of Apache. Additional references exist at the end of this document. My instructions for Setting up SSL: Debian and Apache 2 are kept most current, and will carry you through to completion.

Making a homemade CA or self-signed certificate will cause the client web browser to prompt with a message whether to trust the certificate signing authority (yourself) permanently (store it in the browser), temporarily for that session, or to reject it. The message "web site certified by an unknown authority... accept?" may be a business liability for general public usage, although it's simple enough for the client to accept the certificate permanently.

Whichever route you take, you'll save the periodic expense of paying a recognized signing authority. This is purely for name recognition -- they've paid the major browser producers to have their CA pre-loaded into them. So if you're on a budget, have a special need or small audience, this may be useful.

Before you start
You need Apache and openssl. Compiling them from source, handling dependencies, etc. is beyond the scope of this document. You can consult their documentation, or go with a mainstream Linux distro that will do the preliminary work for you.

Now you need to decide whether you'll make a CA (Certificate Authority) and sign a server certificate with it -- or just self-sign a server certificate. Both procedures are detailed below.

(1A) Create a self-signed certificate.

Complete this section if you do NOT want to make a CA (Certificate Authority). If you want to make a CA, skip 1A entirely and go to 1B instead.

Some steps in this document require priviledged access, and you'll want to limit access to the cert files to all but the root user. So you should su to root and create a working directory that only root has read/write access to (for example: mkdir certwork, chmod 600 certwork). Go to that directory.

Generate a server key:

```
openssl genrsa -des3 -out server.key 4096
```

Then create a certificate signing request with it. This command will prompt for a series of things (country, state or province, etc.). Make sure that "Common Name (eg, YOUR name)" matches the registered fully qualified domain name of your box (or your IP address if you don't have one). I also suggest not making a challenge password at this point, since it'll just mean more typing for you.

The default values for the questions ([AU], Internet Widgits Pty Ltd, etc.) are stored here: /etc/ssl

/openssl.cnf. So if you've got a large number of certificate signing requests to process you probably want to carefully edit that file where appropriate. Otherwise, just execute the command below and type what needs to be typed:

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

Now sign the certificate signing request. This example lasts 365 days:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Make a version of the server.key which doesn't need a password:

```
openssl rsa -in server.key -out server.key.insecure
mv server.key server.key.secure
mv server.key.insecure server.key
```

These files are quite sensitive and should be guarded for permissions very carefully. Chown them to root, if you're not already sudo'd to root. I've found that you can chmod 000 them. That is, root will always retain effective 600 (read) rights on everything.

Now that you've just completed Step 1A, skip ahead to Step 2.

(1B) Generate your own CA (Certificate Authority).

Complete this section if you want to make a CA (Certificate Authority) and sign a server certificate with it. The steps for making a server certificate are also included here. If you'd rather one-time self-sign a server certificate, skip this step entirely and go to 1A instead.

Some steps in this document require priviledged access, and you'll want to limit access to the cert files to all but the root user. So you should su to root and create a working directory that only root has read/write access to (for example: mkdir certwork, chmod 600 certwork). Go to that directory.

In this step you'll take the place of VeriSign, Thawte, etc. You'll first build the CA key, then build the certificate itself.

The Common Name (CN) of the CA and the Server certificates must NOT match or else a naming collision will occur and you'll get errors later on. In this step, you'll provide the CA entries. In a step below, you'll provide the Server entries. In this example, I just added "CA" to the CA's CN field, to distinguish it from the Server's CN field. Use whatever schema you want, just make sure the CA and Server entries are not identical.

CA:
Common Name (CN): www.somesite.edu CA
Organization (O): Somesite
Organizational Unit (OU): Development

Server:
Common Name (CN): www.somesite.edu
Organization (O): Somesite
Organizational Unit (OU): Development

If you don't have a fully qualified domain name, you should use the IP that you'll be using to access your SSL site for Common Name (CN). But, again, make sure that something differentiates the entry of the CA's CN from the Server's CN.

```
openssl genrsa -des3 -out ca.key 4096
openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

Generate a server key and request for signing (csr).

This step creates a server key, and a request that you want it signed (the .csr file) by a Certificate Authority (the one you just created in Step #1B above.)

Think carefully when inputting a Common Name (CN) as you generate the .csr file below. This should match the DNS name, or the IP address you specify in your Apache configuration. If they don't match, client browsers will get a "domain mismatch" message when going to your https web server. If you're doing this for home use, and you don't have a static IP or DNS name, you might not even want worry about the message (but you sure will need to worry if this is a production/public server). For example, you could match it to an internal and static IP you use behind your router, so that you'll never get the "domain mismatch" message if you're accessing the computer on your home LAN, but will always get that message when accessing it elsewhere. Your call -- is your IP stable, do you want to repeat these steps every time your IP changes, do you have a DNS name, do you mainly use it inside your home or LAN, or outside?

```
openssl genrsa -des3 -out server.key 4096
openssl req -new -key server.key -out server.csr
```

Sign the certificate signing request (csr) with the self-created Certificate Authority (CA) that you made earlier.

Note that 365 days is used here. After a year you'll need to do this again.

Note also that I set the serial number of the signed server certificate to "01". Each time you do this, especially if you do this before a previously-signed certificate expires, you'll need to change the serial key to something else -- otherwise everyone who's visited your site with a cached version of your certificate will get a browser warning message to the effect that your certificate signing authority has screwed up -- they've signed a new key/request, but kept the old serial number. There are a couple ways to rectify that. crl's (certificate revocation list) is one method, but beyond the scope of the document. Another method is for all clients which have stored the CA certificate to go into their settings and delete the old one manually. But for the purposes of this document, we'll just avoid the problem. (If you're a sysadmin of a production system and your server.key is compromised, you'll certainly need to worry.)

The command below does a number of things. It takes your signing request (csr) and makes a one-year valid signed server certificate (crt) out of it. In doing so, we need to tell it which Certificate Authority (CA) to use, which CA key to use, and which Server key to sign. We set the serial number to 01, and output the signed key in the file named server.crt. If you do this again after people have visited your site and trusted your CA (storing it in their browser), you might want to use 02 for the next serial number, and so on. You might create some scheme to make the serial number more "official" in appearance or makeup but keep in mind that it is fully exposed to the public in their web browsers, so it offers no additional security in itself.

```
openssl x509 -req -days 365 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
```

To examine the components if you're curious:

```
openssl rsa -noout -text -in server.key
openssl req -noout -text -in server.csr
openssl rsa -noout -text -in ca.key
```

```
openssl x509 -noout -text -in ca.crt
```

Make a server.key which doesn't cause Apache to prompt for a password.

Here we create an insecure version of the server.key. The insecure one will be used for when Apache starts, and will not require a password with every restart of the web server. But keep in mind that while this means you don't have to type in a password when restarting Apache (or worse -- coding it somewhere in plaintext), it does mean that anyone obtaining this insecure key will be able to decrypt your transmissions. Guard it for permissions VERY carefully.

```
openssl rsa -in server.key -out server.key.insecure
mv server.key server.key.secure
mv server.key.insecure server.key
```

These files are quite sensitive and should be guarded for permissions very carefully. Chown them to root, if you're not already sudo'd to root. I've found that you can chmod 000 them. That is, root will always retain effective 600 (read) rights on everything.

(2) Copy files into position and tweak Apache.

Some professors like to pause for a moment after a long lecture, and do a little recap. It's a good pedagogical tool, so let's do so here. If you took route 1A above, you should have four files in a working directory:

server.crt: The self-signed server certificate.
server.csr: Server certificate signing request.
server.key: The private server key, does not require a password when starting Apache.
server.key.secure: The private server key, it does require a password when starting Apache.

If you took route 1B and created a CA, you'll have two additional files:

ca.crt: The Certificate Authority's own certificate.
ca.key: The key which the CA uses to sign server signing requests.

The CA files are important to keep if you want to sign additional server certificates and preserve the same CA. You can reuse these so long as they remain secure, and haven't expired.

At a bare minimum, the following considerations must now be addressed:

- You'll need a virtual host and document root set up for the SSL instance.
- You'll need to turn on the SSL engine and enable/load the SSL module.
- Apache must reference server.crt and server.key somewhere in its configuration.
- Apache must be listening to a port for which SSL is enabled (443 is default).

The particulars differ between Linux distros and versions of Apache. I'm only able to keep the Setting up SSL: Debian and Apache 2 documentation current due to time constraints. Those steps should apply broadly to other Debian-based distros with little or no modification. Red Hat and openSUSE commentary is kept online here for historical purposes.

Setting up SSL: Debian and Apache 2
Setting up SSL: Ubuntu and Apache 2
Setting up SSL: Red Hat and Apache 1.3.x
Setting up SSL: openSUSE

Paul Bramscher

E-mail: brams006@umn.edu

Updated: June 15, 2012