USING CLIENT CERTS WITH APACHE

Implementation Notes from caelyx.networks

caelyx

Simon Brown
projects at caelyx dot net

22 February, 2005

Why?

I was in a bind - I needed to be able to put an application on a system connected to the internet, but guarantee that only authorised people could see it. While the app had password security, I didn't trust it. And it's a lot easier to guess a password than a large secret key. Give me crypto over user memory any day.

So I went to implement this thing, but found a distinct lack of **useful** information on the Internet. Every howto and guide was tangential to what I actually needed to do. In the end, I wound up buying O'Reilly publishing's Network Security with OpenSSL (found here) and some of this document is paraphrased from that book. This book was a great crutch, and I totally recommend it if you have to do any serious work with OpenSSL.

This document is a summary of what I did to implement this functionality. It's here so that I can find it quickly and easily, but in the hope that you find it useful. If you do, or if you have a comment or correction, please send me an email at projects@caelyx.net.

Assumptions & Requirements

This guide is based upon the work I did - I know it works on RedHat 9 and Fedora Core 1, with their respective standard openssl and httpd installs. I recommend just using the packages for ease of uninstall, but you can compile from source if you fancy. I'm not going to walk you through running ./configure; make; su; make install. I use the default paths throughout this document. If you compile from source and put things in a different spot, please replace the default paths with your own.

At the very least, you will need:

- OpenSSL 0.9.7+;
- Apache HTTPD 2.x;
- a recent mod_ssl;
- a browser that accepts pkcs12 certificates (e.g: Firefox, Internet Explorer, Safari);
- time; and
- lots of patience.

I'll put anything you should type in a text like this.

```
# <command to run as root>
```

Now, on to the goods.

Create an OpenSSL CA

1. Create the directory where you are going to store the CA. I recommend /opt/ca or /opt/ca/<name> if you want multiple CAs on one box. I will use /opt/ca

```
# mkdir /opt/ca
# cd /opt/ca
```

2. Create the directory structure

```
# mkdir certs private requests
# touch index.txt
# cat 01 > serial
```

3. Copy a template config file in, and edit it to taste

```
# cp /usr/share/ssl/openssl.cnf /opt/ca
# export OPENSSL_CONF=/opt/ca/openssl.cnf
```

I recommend you increase the default certificate validity and keysize, both for the CA and the generated certificates.

4. Create the root CA cert and self-sign it

```
# openssl req -x509 -out cacert.pem -outform PEM
-newkey rsa:2048
```

Please use a decent passphrase for your private key - if your box gets owned, this is the only thing protecting you.

Your CA is now ready to go.

Setup Apache to require client certs

1. Copy the CA's public key to apache's SSL certificates directory

```
# cp /opt/ca/cacert.pem
/etc/httpd/conf/ssl.crt/localca.crt
```

2. Update the certificate links

```
# cd /etc/httpd/conf/ssl.crt
# make -f Makefile.crt update
```

3. Edit /etc/httpd/conf.d/ssl.conf to require client certs.

```
# ex -c ":%s/#[ \t]*\(VerifyClient\)/\1/"
/etc/httpd/conf.d/ssl.conf
```

Or just find the line that has 'VerifyClients" in it, uncomment it, and make sure the value is 'Require'

4. Restart httpd

```
    # service httpd restart
```

If all has gone to plan, you should no longer be able to connect to https without a cert, signed by our CA.

Create the client certificates

For each user who needs to be able to access the https service, follow these steps. In each step, replace <name> with an easily recognisable id for that user. Eg: <name>key.pem becomes simonkey.pem

1. Create the certificate request.

```
# cd /opt/ca/requests
# openssl req -newkey rsa:1024 -out <name>req.pem -outform
PEM -keyout <name>key.pem -keyform PEM
```

Make sure you fill out the information correctly, especially the common name.

2. Sign the request using the CA's key

```
# openssl ca -in <name>req.pem -out <name>.crt
```

3. Export the certificate as a PKCS12 cert that most browsers can understand.

```
# openssl pkcs12 -export -in <name>.crt -keyin
<name>key.pem -out <name>.p12
```

4. Get the .p12 file to a user securely. I recommend flash disks, floppies or GPG encrypted email.

5. Import the certificate into the browser of choice.I'll eventually publish the easy way to do this.

And you're done. Rinse repeat for each user.

Conclusion

Once you've done all this, you should have an HTTPS server that requires clients to have a certificate issued by a CA under your control.

Note that I've excluded some important security concerns. You should never store the CA root cert, or any private key on a bastion host like an HTTPS server. Preferably use your workstation, or a secured server as the CA. If you're paranoid like me, you can even store the CA private cert on a flash disk or CDR.

Please feel free to send me email with comments, corrections or suggestions. All are welcome.