



Securing Web Access with a Private Certificate Authority

Presented by Paul Weinstein, Waubonsie Consulting, <pdw@waubonsie.com>

ApacheCon US 2002 - November 20, 2002

Session Notes

Waubonsie Consulting

Introduction

Before we dive into the main part of this presentation I want to take a moment and introduce myself such that everyone understands where I'm coming from with this presentation in terms of my personal experience and also to introduce what I plan to actually cover within this session.

As I mentioned my name is Paul Weinstein and I currently work as an independent computer consultant, focusing on web engineering and of course web security. Before striking out on my own I work as a web engineer for Red Hat and C2Net Software where I developed and maintained a number of web-based systems such as a sales and support system and public e-commerce site.

This session is broken down into two main parts:

The Basics

With the basics will quickly review what makes up a **digital certificate** and where a **private certificate authority** maybe be of use to us as web engineers and system administrators and;

The Nit and Gritty

The nit and gritty is where will get into the details on using **OpenSSL** for **creating a private certificate authority**, using **Apache** to **publish our private certificate authority** and of course configuring **Apache with mod_ssl** for **using our private certificate authority**.

Notice

One last note before we move on, this presentation is really about **Access Control** and **Authentication** and while both of these topics do indeed deal with **network security**, this talk should not be considered a network security talk. We won't be discussing how to **build a secure Apache server**, how to **maintain a private network**, how to **create a security policy** to work from or even how the **SSL protocol** works. As such I strongly recommend that if you haven't already, to take the time to understand the concepts behind network security and the SSL protocol before deploying a solution such as the one we are about to cover.

Digital Certificates

The SSL protocol's two main features, **Encryption** and **Authentication**, provide the foundation from which we will be building upon, since the method of authentication that the SSL protocol uses is that of using **Digital Certificates**.

A Digital Certificate provides unique information about some party, a client, a server, or some third, outside party, a public key belonging to the party and a digital signature belonging to a **certificate authority**.

In terms of today's talk there are three "types" of digital certificates that concern us. A **root certificate**, which is a digital certificate that identifies a certificate authority, a **server certificate**, which is a digital certificate that identifies a web server and finally, a **client**

certificate which is a digital certificate identifying a web client.

Certificate Authorities

A certificate authority is some party that has check the integrity and signs the information within the digital certificate. There are two classifications for certificate authorities,

Public Certificate Authorities

Public Certificate Authorities, such as **Verisign**, **Thawte** or **GeoTrust** are no doubt what comes to mind when one talks about certificate authorities. These authorities act as outside agents validating certificate requests and in turn issues digital certificates to individuals and organizations who, having no other method connection, depend upon in order to authenticate each other. By default most web clients and web servers have these certificate authorities' root certificates installed allowing them to recognized digital certificates issued by these authorities.

A perfect example of course is **Amazon.com**. If for example, I had no other connection to Amazon.com and wished to purchase goods from them I first need to answer some basic questions of trust. How do I know that the server I'm sending personal information to belonging to Amazon.com? Moreover, how do I know that Amazon.com will indeed send the goods promised on their website? Luckily for me, my web browser has detected a server certificate identifying the server as belonging to Amazon.com. Moreover, this digital certificate has been recognized as signed by Verisign. If I trust that Verisign has confirmed that the server and domain in question as belonging to Amazon.com and that Amazon.com is indeed a legal business, I can trust that I can indeed send my information to server in question and that I will indeed get the goods I requested.

Private Certificate Authorities

A **Private Certificate Authority**, on the other hand, is most commonly used when some sort of trust model already exists. For example;

A PCA in Action

Intranet

Consider an **Intranet** within a medium or large corporation that has multiple web servers running within a private network. How can an employee trust that the **Human Resources** server is indeed the HR server and that they can trust that any personal information they communicate via a web-based application is indeed getting to HR? If our corporation has setup an internal certificate authority that can "sign-off" on HR's server by issuing a digital certificate allow the employees to trust that the server in question is indeed the HR's server and not a server belonging to some other department or individual.

Extranet

Another example can be found in maintaining an **Extranet**. Consider that our would-be corporation has a business partner for which we want to dedicate a private section of our public web server to. We can issue client certificates, signed by our own certificate authority to employees of our business partner, who, when needed can authenticate themselves to our

web server using their digital certificates in order to access the restricted area. We can of course go on with various examples, mixing and matching solutions as needed. However, on thing remains in all of these examples a basic model of trust already exists from which we can work with.

Creating a Private Certificate Authority

A self-signed root certificate:

```
[pca]$ openssl req -new -x509 -keyout private/ca.key -out certs/ca.cert -
config openssl.cnf
Using configuration from openssl.cnf
Generating a 1024 bit RSA private key
.....+++++
...+++++
writing new private key to 'private/ca.key'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Richmond
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Waubonsie
Consulting
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:Paul Weinstein
Email Address []:pdw@waubonsie.com
```

Consider, for example, that we are a manager within the Security Department of our corporation. We've been charged with creating an internal certificate authority. As such one of our first steps in setting up a private certificate authority is creating a root certificate that identifies our certificate. We can do this using OpenSSL, creating a public-private key pair, assigning a pass-phrase to our private key and inputting information about our certificate authority for our digital certificate which we will **self-signing** using our newly creating private key.

Configuring OpenSSL:

```
[ CA_default ]

dir                = /usr/local/pca           # Where everything is kept
certs              = $dir/certs               # Where the issued certs are kept
crl_dir            = $dir/crl                 # Where the issued crl are kept
database           = $dir/index.txt          # database index file.
new_certs_dir      = $dir/certs               # default place for new certs.

certificate        = $dir/certs/ca.cert      # The CA certificate
```

```

serial      = $dir/serial      # The current serial number
crl         = $dir/crl.pem     # The current CRL
private_key = $dir/private/ca.key # The private key
RANDFILE    = $dir/private/.rand # private random number file

x509_extensions = usr_cert      # The extensions to add to the cert

```

From here on out everything we do with OpenSSL is done in relation to our new certificate authority we just created. As such we need to configure OpenSSL such that it knows where to find the digital certificate and private key we just created for our certificate authority.

```

policy      = policy_match

# For the CA policy
[ policy_match ]
countryName      = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName       = supplied
emailAddress     = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName      = optional
stateOrProvinceName = optional
localityName     = optional
organizationName = optional
organizationalUnitName = optional
commonName       = supplied
emailAddress     = optional

```

We also need to define a **policy** as to what information is required and optional for inclusion in the digital certificate requests we sign with our internal certificate authority.

Publishing the Private Certificate Authority

Setting MIME-type in Apache:

```

#
# Some MIME-types for downloading Certificates and CRLs
#
AddType application/x-x509-ca-cert .crt

```

Lastly, we need to distribute the root certificate of our certificate authority such that web clients and servers within our corporation can recognize and trust digital certificates signed by us. As such we need to add the following **AddType** to our security department's Apache server so that our root certificate can be downloaded and installed.

Using Our Private Certificate Authority

Server Certificate

Creating a Certificate Signing Request:

```
[pca]$ openssl req -new -keyout private/server.key -out certs/server.csr -
days 365 -config openssl.cnf
Using configuration from openssl.cnf
Generating a 1024 bit RSA private key
..+++++
.....+++++
writing new private key to 'private/server.key'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Richmond
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Waubonsie
Consulting
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:www.waubonsie.com
Email Address []:webmaster@waubonsie.com
```

Now let us switch roles, let us consider ourselves as the administrator of an Apache server, and say the HR department's server and we need to meet this new corporate requirement of getting a server certificate for authentication of our department's server. To get a server certificate for our web server we need to put in a **certificate signing request** to our security department manager running our private certificate authority.

To create a certificate signing request we use OpenSSL to create a public-private key pair, assign a pass-phrase for our server's private key and supply information for our server certificate. Lastly, we can pass our **server.csr** file onto our security department for signing.

Signing the Certificate Signing Request:

```
[pca]$ openssl ca -out certs/server.cert -config openssl.cnf -infile
certs/server.csr
Using configuration from openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName             :PRINTABLE:'US'
stateOrProvinceName     :PRINTABLE:'California'
localityName            :PRINTABLE:'Richmond'
organizationName        :PRINTABLE:'Waubonsie Consulting'
commonName              :PRINTABLE:'www.waubonsie.com'
emailAddress            :IA5STRING:'webmaster@waubonsie.com'
Certificate is to be certified until Oct  7 04:16:40 2003 GMT (365 days)
```

```
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y  
Write out database with 1 new entries  
Data Base Updated
```

Back in our role as manager in our security department we now have a certificate signing request from the administrator in the HR department. Using the OpenSSL config file we defined before and after we verify the information within our certificate request we go ahead and sign the request with our certificate authorities' private key creating a digital certificate that will be valid for one year's time.

Configuring Apache:

```
# Server Certificate:  
# Point SSLCertificateFile at a PEM encoded certificate. If  
# the certificate is encrypted, then you will be prompted for a  
# pass phrase. Note that a kill -HUP will prompt again. A test  
# certificate can be generated with 'make certificate' under  
# built time. Keep in mind that if you've both a RSA and a DSA  
# certificate you can configure both in parallel (to also allow  
# the use of DSA ciphers, etc.)  
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.cert  
  
# Server Private Key:  
# If the key is not combined with the certificate, use this  
# directive to point at the key file. Keep in mind that if  
# you've both a RSA and a DSA private key you can configure  
# both in parallel (to also allow the use of DSA ciphers, etc.)  
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
```

Switching roles once again, back as the administrator of our HR server and armed with a server certificate we need to configure our Apache server with mod_ssl to where our new server certificate is with the directive **SSLCertificateFile**. We also need to point to the private key we created when we made our certificate signing request using the **SSLCertificateKeyFile** directive.

Client Certificate

Creating a Certificate Signing Request:

If we want to authenticate our clients using digital certificates we need to first get client certificates to our users and that of course means getting certificate signing requests from them for processing.

Signing the Certificate Signing Request:

```
[pca]$ openssl ca -spkac certs/client.csr -out certs/client.cert -days 365 -  
config openssl.cnf  
Using configuration from openssl.cnf  
Enter PEM pass phrase:  
Check that the SPKAC request matches the signature  
Signature ok  
The Subjects Distinguished Name is as follows
```

```

CommonName      :PRINTABLE:'Employee Certificate for Paul Weinstein'
emailAddress     :IA5STRING:'pdw@waubonsie.com'
organizationName :PRINTABLE:'Waubonsie Consulting'
organizationalUnitName:PRINTABLE:''
stateOrProvinceName :PRINTABLE:'California'
localityName     :PRINTABLE:'Richmond'
countryName      :PRINTABLE:'US'
Certificate is to be certified until Oct  7 04:16:40 2003 GMT (365 days)

Write out database with 1 new entries
Data Base Updated

```

As with the processing the certificate signing request for our HR server we need to first, as the manager of our certificate authority, verify the accuracy of the information within the certificate requests before processing. Then, using OpenSSL we can actually sign the request for a client certificate.

Configuring Apache:

```

# Certificate Authority (CA):
# Set the CA certificate verification path where to find CA
# certificates for client authentication or alternatively one
# huge file containing all of them (file must be PEM encoded)
# Note: Inside SSLCACertificatePath you need hash symlinks
#       to point to the certificate files. Use the provided
#       Makefile to update the hash symlinks after changes.
SSLCACertificatePath /usr/local/apache/conf/ssl.crt/ca.cert

# Client Authentication (Type):
# Client certificate verification type and depth. Types are
# none, optional, require and optional_no_ca. Depth is a
# number which specifies how deeply to verify the certificate
# issuer chain before deciding the certificate is not valid.
SSLVerifyClient require
SSLVerifyDepth 10

```

Since the SSL protocol doesn't require a client to authenticate itself with a digital certificate, we need to, as our HR server administrator, configure Apache to request a digital certificate from our client using the **SSLVerifyClient** directive.

We also need to point to the root certificate for our certificate authority with the **SSLCACertificatePath** such that when Apache is presented with a client certificate it can recognize who validate the information contained within the digital certificate.

Certificate Revocation List

Revoking an Existing Digital Certificate:

```

[pca]$ openssl ca -revoke certs/client.cert -config openssl.cnf
Using configuration from openssl.cnf
Enter PEM pass phrase:
Revoking Certificate 10.
Data Base Updated

[pca]$ openssl ca -gencrl -out ca.crl -config openssl.cnf

```



```
Using configuration from openssl.cnf
Enter PEM pass phrase:
```

What happens if the information within a digital certificate expires before the certificate itself expires? Say for example an employee who was issues a client certificate leaves the employment of the company. Or, by no fault of the employee's the laptop that contained the employee's digital certificate is stolen. What can we do, as the manager of our private certificate authority, to maintain that no one unauthorized to access our companies' servers tries to use one of these client certificates?

We can prevent such a situation by **revoking** the validity of the digital certificate our certificate authority created and then by distributing a list of revoked certificates such that one can scan the list before finally accepting a digital certificate bearing the signature of our certificate authority for authorization.

Our first step, as the manager of our certificate authority, is to revoke the privileges of the digital certificate in question using OpenSSL. Then we create a **certificate revocation list**; again with OpenSSL that we can distribute, such as we did with our root certificate, for installation in our web clients and servers.

Setting MIME-type in Apache

```
#
# Some MIME-types for downloading Certificates and CRLs
#
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl .crl
```

Now we need to configure our Security Department's web server to distribute our revocation list by adding the proper MIM-Type for downloading.

Configuring Apache:

```
# Certificate Revocation Lists (CRL):
# Set the CA revocation path where to find CA CRLs for client
# authentication or alternatively one huge file containing all
# of them (file must be PEM encoded)
# Note: Inside SSLCARevocationPath you need hash symlinks
# to point to the certificate files. Use the provided
# Makefile to update the hash symlinks after changes.
SSLCARevocationFile /usr/local/apache/conf/ssl/ssl.crl/ca.crl
#SSLCARevocationPath /usr/local/apache/conf/ssl/ssl.crl
```

In our role as the administrator of our HR server, once we get a copy of the certificate revocation list from our friends in the Security department, we need to configure our Apache server to read the revocation list(s) using the **SSLCARevocationFile** directive.

Review

The Basics

In our presentation today we covered the basics on what makes up a **digital certificate**, the role of a certificate authority in the creation of a digital certificate and where a **private**

certificate authority maybe be of use to us as web engineers and system administrators in our networks and;

The Nit and Gritty

How we can use the **OpenSSL** toolkit for **creating a private certificate authority**, how to using **Apache** to **publish our private certificate authority's** root certificate and of course how configuring **Apache with mod_ssl** for **using our private certificate authority** in authenticating web clients and web servers within a closed environment.

Citation

Hirsch, Frederick *Introducing SSL and Certificates using SSLeay*. 8 Oct 2002
<<http://www.pseudonym.org/ssl/wwwj-index.html>>.

Engelschall, Ralf *User Manual mod_ssl Version 2.8* 9 Oct. 2002
<<http://www.modssl.org/docs/2.8/>>

Resources

This Presentation

Online:

<<http://www.weinstein.org/work/presentations/apacheconus02/pca/>> (HTML)
<<http://www.weinstein.org/work/presentations/apacheconus02/pca.pdf>> (PDF)

CD-ROM:

Whitepaper
Handout

Upcoming Publication:

Mobily, Tony, Paul Weinstein, Mark Wilcox, Debashish Bhattacharjee, Sandip Bhattacharya, Brian Rickabaugh. *Apache Security*. Birmingham: Wrox Press, 2003.

Apache HTTP Server Project

<<http://httpd.apache.org>>

Apache Week

<<http://www.apacheweek.com>>

mod_ssl Project

<<http://www.modssl.org>>

Mailing Lists, List Archives:

Warbonise Consulting

<modssl-announce@modssl.org>
<modssl-users@modssl.org>
<<http://marc.theaimsgroup.com/?l=apache-modssl>>

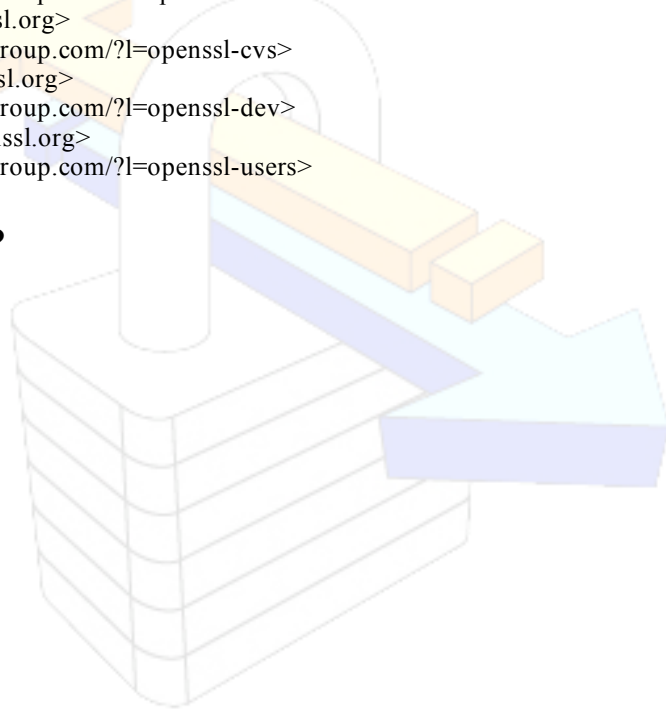
OpenSSL Project

<<http://www.openssl.org>>

Mailing Lists, List Archives:

<openssl-announce@openssl.org>
<<http://marc.theaimsgroup.com/?l=apache-modssl>>
<openssl-cvs@openssl.org>
<<http://marc.theaimsgroup.com/?l=openssl-cvs>>
<openssl-dev@openssl.org>
<<http://marc.theaimsgroup.com/?l=openssl-dev>>
<openssl-users@openssl.org>
<<http://marc.theaimsgroup.com/?l=openssl-users>>

Any Questions?



Waubonsie Consulting