Apache based WebDAV Server with LDAP and SSL

Saqib Ali

<saqib@seagate.com>

Offshore XML/XHTML Development

Revision History

Revision v4.1.2	2003–10–17	Revised by: sa
Added the SSL performance tuning section.		
Revision v4.1.1	2003–09–29	Revised by: sa
Updated the SSL section based on the feedback received from readers.		
Revision v4.1.0	2003–09–02	Revised by: sa
Updated the SSL section based on the feedback received from readers.		
Revision v4.0.2	2003–08–01	Revised by: sa
Minor updates to the Apache configure cmd line. /dev/random referenced in the SSL section.		
Revision v4.0.1	2003–07–27	Revised by: sa
Added more information to the SSL section.		
Revision v4.0	2003–06–29	Revised by: sa
Updated the HOWTO for Apache 2.0. Also the source is in XML		

.This document is an HOWTO on installing a Apache based WebDAV server with LDAP for authentication and SSL encryption.

Table of Contents

Table of Contents

1. Introduction

The Objective of this document in to Setup a Apache + mySQL + PHP + WebDAV based Web Application Server, that uses LDAP for Authentication. The documentation will also provide details on the encrypting LDAP transactions.

Note:

If you encounter any problems installing Apache or any of the modules please feel free to contact me @ `<saqib@seagate.com>`

1.1. About this document

This document was originally written in 2001. Since then many updates and new additions have been made. Thanks to all the people who submitted updates and corrections.

The XML source of this document is available at http://www.xml−dev.com:8080/cocoon/mount/docbook/Apache−WebDAV−LDAP−HOWTO.xml.

The latest version of the document is available at http://www.xml−dev.com:8080/cocoon/mount/docbook/Apache−WebDAV−LDAP−HOWTO.html.

1.2. Contributions to the document

If you like to contribute to the HOWTO, you can d/l the XML source from http://www.xml−dev.com:8080/cocoon/mount/docbook/Apache−WebDAV−LDAP−HOWTO.xml , and send in the updated source to saqib@seagate.com ALONG WITH YOUR NAME IN THE LIST OF AUTHORS AND REVISION HISTORY :). That makes it easier for me contact the person if there are any updates/corrections. Thanks.

1.3. What is Apache?

The Apache HTTP Server is an open−source HTTP server for modern operating systems including UNIX and Windows NT. It provides HTTP services in sync with the current HTTP standards.

Thei Apache WebServer is available for free download from http://httpd.apache.org/

1.4. What is WebDAV?

WebDAV stands for Web enabled Distributed Authoring and Versioning. It provides a collaborative environment for users to edit/manage files on web−servers. Technically DAV is an extension to the http protocol.

Here is a brief description of the extensions provided by DAV:

Overwrite Protection: Lock and Unlock mechanism to prevent the "lost update problem". DAV protocol support both shared and exclusive locks.

Properties: Metadata (title, subject, creater, etc)

Name−space management: Copy, Rename, Move and Deletion of files

Access Control: Limit access to various resources. Currently DAV assumes access control is already in place, and does not provide strong authentication mechanism.

Versioning: Revision control for the documents. Versioning is not implemented yet.

1.5. What is PHP?

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely−used Open Source general−purpose scripting language that is especially suited for Web development and can be embedded into HTML.

PHP is available from http://www.php.net

1.6. What is mySQL?

MySQL, the most popular Open Source SQL database, is developed, distributed, and supported by MySQL AB

mySQL DB Engine can be downloaded from http://www.mysql.com/

1.7. What do we need?

The tools needed to achieve this objective are:

i. C Compiler e.g. GCC
ii. Apache 2 Web Server
iii. LDAP Module for Apache
iv. iPlanet LDAP lib files
v. SSL engine
vi. PHP
vii. mySQL DB Engine

Note:

All of these packages are free and are available for download on the net.

1.8. Assumptions

This document assumes that you have the following already installed on your system.

i. gzip or gunzip − available from http://www.gnu.org
ii. gcc and GNU make − available from http://www.gnu.org

2. Requirements

You'll have to download and compile several packages. This document will explain the compilation process, but you should be fimiliar with installing from source code.

2.1. Basics

You will need a machine running Solaris / Linux and GCC Compiler. GNU gnzip and GNU tar is also needed.

2.2. Apache 2.0.46

Apache is the HTTP server, it will be used to run the Web Application Server. Please download the Apache 2.0.46 source code from http://www.apache.org/dist/httpd/.

2.3. OpenSSL

You will need to download the OpenSSL from http://www.openssl.org/source/ . Please download the latest version. OpenSSL installation will be used for SSL libraries for compiling mod_ssl with Apache, and for managing SSL certificates on the WebServer. Please download the OpenSSL source code gzipped file into /tmp/downloads

2.4. iPlanet LDAP Library

Download the iPlanet LDAP SDK from http://wwws.sun.com/software/download/products/3ec28dbd.html. We will use iPlanet LDAP SDK, because it includes libraries for ldaps:// (LDAP over SSL)

2.5. mod_auth_ldap

mod_auth_ldap will be used for compiling LDAP support into Apache. Please download mod_auth_ldap from http://www.muquit.com/muquit/software/mod_auth_ldap/mod_auth_ldap_apache2.html

2.6. mySQL DB Engine

Download the appropriate mySQL build for your platform from http://www.mysql.com/downloads/index.html

2.7. PHP

Download the PHP source code from http://www.php.net/downloads.php

3. Installation

First we hve take care of the few pre–requisites, and then we will get into the main installtion.

3.1. Pre–requisites

The application server as we plan to install, requires the SSL libraries and LDAP libraries. SSL engine is also required for managing the SSL certs for Apache 2.x

3.1.1. iPlanet LDAP SDK

Become root by using the su command:

```
$ su -
```

Create the `/usr/local/iplanet-ldap-sdk.5` directory. Copy the `ldapcsdk5.08-Linux2.2_x86_glibc_PTH_OPT.OBJ.tar.gz` form `/tmp/downloads` to `/usr/local/iplanet-ldap-sdk.5` directory.

```
# mkdir /usr/local/iplanet-ldap-sdk.5
# cp /tmp/downloads/ldapcsdk5.08-Linux2.2_x86_glibc_PTH_OPT.OBJ.tar /usr/local/iplanet-ldap-sdk.5
# cd /usr/local/iplanet-ldap-sdk.5
# tar -xvf ldapcsdk5.08-Linux2.2_x86_glibc_PTH_OPT.OBJ.tar
```

Now you should have all the required iPlanet LDAP lib files in the correct directory

3.1.2. OpenSSL Engine

Next we need to install the OpenSSL Engine

OpenSSL is an open source implementation of the SSL/TLS protocol. It is required to create and manage SSL certificates on the webserver. The installion is also necessary for the lib files that will be used by the SSL module for apache.

Change to the directory where you placed the OpenSSL source code files

```
# cd /tmp/download
# gzip -d openssl.x.x.tar.gz
# tar -xvf openssl.x.x.tar
# cd openssl.x.x
# make
# make test
# make install
```

Upon successful completion of the **make install** the openssl binaries should reside in `/usr/local/ssl`

3.2. mySQL

Installaing mySQL is quite simple. The downloaded binaries have to be place in appropriate directory.

We start creating a user:group for mysql daemon, and copying the files to appropriate directories.

```
# groupadd mysql
# useradd -g mysql mysql
# cd /usr/local
# gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
# ln -s full-path-to-mysql-VERSION-OS mysql
```

Next run the install_db script, and change permission on the files

```
# cd mysql
# scripts/mysql_install_db
# chown -R mysql .
```

3.2.1. Starting mySQL

Now start the mySQL server to verify the installation

```
# bin/mysqld_safe --user=mysql &
```

Verify mySQL daemon is running, by using the ps –ef command. You should see the following output:

```
# ps -ef | grep mysql
root      3237     1  0 May29 ?        00:00:00 /bin/sh bin/safe_mysqld
mysql     3256  3237  0 May29 ?        00:06:58 /usr/local/mysql/bin/mysqld --defaults-extra-file=/usr/local/mysql/data/my.cnf --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data --user=mysql --pid-file=/usr/local/mysql/data/downloa
```

3.2.2. Stopping mySQL

To stop the MySQL server, follow the instructions below

```
# cd /usr/local/mysql
# ./bin/mysqladmin -u root -p shutdown
```

3.2.3. Locating Data Directory

mySQL deamon stores all the information in a direcory called "Data Directory". If you followed the installation instructions above, your Data Directory should be located under `/use/local/mysql/data`.

To locate where your Data Directory is located, use the **mysqladmin** utility as follows:

```
# /usr/local/mysql/bin/mysqladmin variables -u root --password={your_password} | grep datadir
```

3.3. Apache 2.0

Start by setting some FLAGS for the compiler

```
# export LDFLAGS="-L/usr/local/iplanet-ldap-sdk.5/lib/ -R/usr/local/iplanet-ldap-sdk.5/lib/:/usr/local/lib"
# export CPPFLAGS="-I/usr/local/iplanet-ldap-sdk.5/include"
```

Next UNTAR the apache 2.0 source files, and execute the `configure` script.

```
# cd /tmp/download
# gzip -d httpd-2.0.46.tar.gz
# tar -xvf httpd-2.0.46.tar
# cd httpd-2.0.46
#./configure --enable-so  --with-ssl --enable-ssl  --enable-rewrite   --enable-dav
```

Next run the make command

```
# make
# make install
```

3.3.1. Starting Apache

```
# /usr/local/apache2/bin/apachectl start
```

3.3.2. Stopping Apache

```
# /usr/local/apache2/bin/apachectl stop
```

3.4. mod_auth_ldap

Untar modauthldap_apache2.tar.gz

```
cd /tmp/download
# gzip -d modauthldap_apache2.tar.gz
# tar -xvf modauthldap_apache2.tar
# cd modauthldap_apache2
```

Now configure and install mod_auth_ldap

```
# ./configure --with-apxs=/usr/local/apache2/bin/apxs  --with-ldap-dir=/usr/local/iplanet-ldap-sdk.5/
# make
# make install
```

3.5. CERT DB for LDAPS://

You will also need to get the cert7.db and key7.db from http://www.xml–dev.com/xml/key3.db and http://www.xml–dev.com/xml/cert7.db and place it in the `/usr/local/apache2/sslcert/`directory.

3.6. PHP

Unzip the PHP Source Files

```
gzip -d php-xxx.tar.gz
tar -xvf php-xxx.tar
```

Configure and run the make command

```
cd php-xxx
./configure --with-mysql --with-apxs=/usr/local/apache2/bin/apxs
```

Compile the source code

```
# make
# make install
```

Copy the php.ini file to the appropriate directory

```
cp php.ini-dist /usr/local/lib/php.ini
```

4. Configuring and Setting up the WebDAV services

Now for the easy part. In this section we will WebDAV enable a directory under Apache root.

4.1. Modifications to the `/usr/local/apache/conf/httpd.conf`

Please verify that the following Apache directive appears in the `/usr/local/apache/conf/httpd.conf` :

```
Addmodule mod_dav.c
```

If it does not please add it. This directive informs Apache about DAV capability. The directive must be placed outside any container.

Next we must specify where Apache should store the DAVLockDB file. DAVLockDB is a lock database for the WebDAV. This directory should be writable by the httpd process.

I store the DAVLock file under `/usr/local/apache/var`. I use this directory for other purposes as well. Please add the following line to your `/usr/local/apache/conf/httpd.conf` to specify that the DAVLockDB file will be under `/usr/local/apache/var` :

```
DAVLockDB      /usr/local/apache/var/DAVLock
```

The directive must be placed outside any container.

4.2. Creating a directory for DAVLockDB

As mentioned above a directory must be created for DAVLockDB that can be written by the web server process. Usually web server process runs under the user '*nobody*' . Please verify this for your system using the command:

```
ps -ef | grep httpd
```

Under `/usr/local/apache` create the directory and set the permissions on it using the following commands:

```
# cd /usr/local/apache
  # mkdir var
  # chmod -R 755 var/
  # chown -R nobody var/
  # chgrp -R nobody var/
```

4.3. Enabling DAV

Enabling DAV is a trivial task. To enable DAV for a directory under Apache root, just add the following directive in the container for that particular directory:

```
DAV On
```

This directive will enable DAV for the directory and its sub−directories.

The following is a sample configuration that will enable WebDAV and LDAP authentication on `/usr/local/apache/htdocs/DAVtest`. Place this in the `/usr/local/apache/conf/httpd.conf` file.

```
 DavLockDB /tmp/DavLock
<Directory "/usr/local/apache2/htdocs/DAVtest">
Options Indexes FollowSymLinks
AllowOverride None
order allow,deny
allow from all
AuthName "SMA Development server"
AuthType Basic
LDAP_Debug On
#LDAP_Protocol_Version 3
#LDAP_Deref NEVER
#LDAP_StartTLS On
LDAP_Server you.ldap.server.com
#LDAP_Port 389
# If SSL is on, must specify the LDAP SSL port, usually 636
LDAP_Port 636
LDAP_CertDbDir /usr/local/apache2/sslcert
Base_DN "o=SDS"
UID_Attr uid
DAV On
#require valid-user
require valid-user
#require roomnumber "123 Center Building"
#require filter "(&(telephonenumber=1234)(roomnumber=123))"
#require group cn=rcs,ou=Groups
</Directory>
```

4.4. Create a Directory called DAVtest

As mentioned in a earlier section, all DAV directories have to be writable by the WebServer process. In this example we assume WebServer is running under username '*nobody*'. This is usually the case. To check httpd is running under what user, please use:

```
# ps -ef | grep httpd
```

Create a test directory called '`DAVtest`' under `/usr/local/apache2/htdocs` :

 # mkdir /usr/local/apache/htdocs/DAVtest

Change the permissions on the directory to make it is read−writable by the httpd process. Assuming the httpd is running under username '*nobody*', use the following commands:

```
# cd /usr/local/apache/htdocs
# chmod -R 755 DAVtest/
# chown -R nobody DAVtest/
# chgrp -R nobody DAVtest/
```

4.5. Restart Apache

Finally you must run the configuration test routine that comes with Apache to verify the syntax in `httpd.conf`:

```
# /usr/local/apache/bin/apachectl configtest
```

If you get error messages please verify that you followed all of the above mentioned steps correctly. If you can not figure out the error message feel free to email me with the error message (saqib@seagate.com).

If the configtest is successful start the apache web-server:

 # /usr/local/apache/bin/apachectl restart

Now you have WebDAV enabled Apache Server with LDAP authentication and SSL encryption.

4.6. WebDAV server protocol compliance testing

It is very important that the WebDAV that we just implemented be fully complaint with the WebDAV-2 protocol. If it is not fully compatible, the client side WebDAV applications will not function properly.

To test the complaince we will use a tool called Litmus. Litmus is a WebDAV server protocol compliance test suite, which aims to test whether a server is compliant with the WebDAV protocol as specified in RFC2518.

Please download the Litmus source code from http://www.webdav.org/neon/litmus/ and place it in the /tmp/downloads directory.

Then use gzip and tar to extract the files:

```
# cd /tmp/downloads
# gzip -d litmus-0.6.x.tar.gz
# tar -xvf litmus-0.6.x.tar
# cd litmus-0.6.x
```

Compiling and installing Litmus is easy:

```
# ./configure
# make
# make install
```

make install will install the Litmus binary files under `/usr/local/bin` and the help files under `/usr/local/man`

To the test the complaince of the WebDAV server that you just installed, please use the following command

```
# /usr/local/bin/litmus http://you.dav.server/DAVtest userid passwd
```

5. WebDAV server management

In this section we will discuss about the various management task – e.g. using LDAP for access control, and working with DAV method on Apache

Most of the configuration changes for the DAV will have to done using the `httpd.conf` file. This file is located at `/usr/local/apache/conf/httpd.conf`

`httpd.conf` is a text based configuration file that Apache uses. It can b editted using any text editor – I preffer using vi. Please make backup copy of this file, before changing it.

After making changes to the `httpd.conf` the Apache server has to be restarted using the **/usr/local/apache/bin/apachectl restart** command. However before restarting you test for the validity of the `httpd.conf` by using the **/usr/local/apache/bin/apachectl configtest** comand.

5.1. Restricting access to DAV shares

In the previous section when we created the DAVtest share, we used the LDAP for authentication purposes. However anyone who can authenticates using their LDAP useri/passwd will be able to access that folder.

Using the **require** directive in the httpd.conf file, we can limit access to certain individuals or groups of individuals.

If we look at the DAVtest configuration from the previosu section:

```
<Directory /usr/local/apache/htdocs/DAVtest>
Dav On
#Options Indexes FollowSymLinks

AllowOverride None
order allow,deny
allow from all
AuthName "LDAP_userid_password_required"
AuthType Basic
<Limit GET PUT POST DELETE PROPFIND PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
Require valid-user
</Limit>
LDAP_Server ldap.server.com
LDAP_Port 389
Base_DN "o=ROOT"

UID_Attr uid
</Directory>
```

We see that the **require** is set to **valid−user**. Which means any valid authenticated user has access to this folder.

5.1.1. Restricting access based on Individual UID(s)

LDAP UID can be used to restrict access to DAV folder.

require valid−user directive can be changed to **require user 334455 445566**

This will restrict access to individuals with UID 334455 and 445566. Anyone else will not be able to access this folder.

5.1.2. Restricting access based on groups of individuals.

require can also be used to restrict access to groups of individuals. This can be either done using LDAP groups or LDAP filters. The filter must be valid LDAP filter syntax.

5.2. Restricting write access to DAV shares

It maybe be required that the editting for the resources on the DAV shares be restricted to certain individual, however anyone can view the resources. This can be easily done using the **<Limit>** tags in the httpd.conf file

```
<Directory /usr/local/apache/htdocs/DAVtest>
Dav On
#Options Indexes FollowSymLinks

AllowOverride None
order allow,deny
allow from all
AuthName "LDAP_userid_password_required"
AuthType Basic
<Limit GET PUT POST DELETE PROPFIND PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
Require valid-user
</Limit>
LDAP_Server ldap.server.com
LDAP_Port 389
Base_DN "o=ROOT"

UID_Attr uid
</Directory>
```

You restrict write access to certain individuals by changing the **<limit>** to

```
<Limit PUT POST DELETE PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
Require 334455
</Limit>
```

Basically we are limiting the PUT POST DELETE PROPPATH MKCOL COPY MOVE LOCK and UNLOCK to an individual who has the UID of 334455. Everone else will be able to use the methods GET and PROPFIND on the resources, but not any other method.

6. Implementing and using SSL to secure HTTP traffic

Security of the data stored on a file server is very important these days. Compromised data can cost thousands of dollars to company. In the last section, we compiled LDAP authentication module into the Apache build to provide a Authentication mechanism. However HTTP traffic is very insecure, and all data is transferred in clear text − meaning, the LDAP authentication (userid/passwd) will be transmitted as clear text as well. This creates a problem. Anyone can sniff these userid/passwd and gain access to DAV store. To prevent this we have to encrypt HTTP traffic, essentially HTTP + SSL or HTTPS. Anything transferred over HTTPS is encrypted, so the LDAP userid/passwd can not be easily deciphered. HTTPS runs on port 443. The resulting build from the last section's compilation process will have Apache to listen to both port 80 (normal HTTP) and 443 (HTTPS). If you are just going to use this server for DAV, then I will highly suggest that you close port 80. In this section of the HOWTO I will provide some information regarding SSL and maintaining SSL on a Apache HTTP server.

6.1. Introduction to SSL

SSL (Secure Socket Layer) is a protocol layer that exists between the Network Layer and Application layer. As the name suggest SSL provides a mechanism for encrypting all kinds of traffic − LDAP, POP, IMAP and most importantly HTTP.

The following is a over−simplified structure of the layers involved in SSL.

```
+-------------------------------------------+
|   LDAP    |     HTTP     |    POP    |    IMAP   |
+-------------------------------------------+
|                    SSL                    |
+-------------------------------------------+
|               Network Layer               |
+-------------------------------------------+
```

6.1.1. Encryption algorithms used in SSL

There are three kinds of cryptographic techniques used in SSL: Public−Private Key, Symmetric Key, and Digital Signature.

Public−Private Key Crytography − Initiating SSL connection: In this algorithm, encryption and decryption is performed using a pair of private and public keys. The Web−server holds the private Key, and sends the Public key to the client in the Certificate.

1. The client request content from the Web Server using HTTPS.
2. The web server responds with a Digital Certificate which includes the server's public key.
3. The client checks to see if the certificate has expired.
4. Then the client checks if the Certificate Authority that signed the certificate, is a trusted authority listed in the browser. This explains why we need to get a certificate from a a trusted CA.
5. The client then checks to see if the Fully Qualified Domain Name (FQDN) of the web server matches the Comman Name (CN) on the certificate?
6. If everything is successful the SSL connection is initiated.

Note:

Anything encrypted with Private Key can only be decrypted by using the Public Key. Similarly anything encrypted using the Public Key can only be decrypted using the Private Key. There is a common mis−conception that only the Public Key is used for encryption and Private Key is used for decryption. This is not case. Any key can be used for encryption/decryption. However if one key is used for encryption then the other key must be used for decryption. e.g. A message can not encrypted and then decrypted using only the Public Key.

Using Private Key to encrypt and a Public Key to decrypt ensures the integrity of the sender (owner of the Private Key) to the recipients. Using Public Key to encrypt and a Private Key to decrypt ensures that only the inteded recipient (owner of the Private Key) will have access to the data.(i.e. only the person who holds the Private Key will be able to decipher the message).

Symmetric Cryptography − Actual transmission of data: After the SSL connection has been established, Symmetric cryptography is used for encrypting data as it uses less CPU cycles. In symmetric cryptography the data can be encrypted and decrypted using the same key. The Key for symmetric cryptography is exchanged during the initiation process, using Public Key Cryptography.

Message Digest The server uses message digest algoritm such as HMAC, SHA−1, MD5 to verify the integrity of the transferred data.

6.1.2. Ensuring Authenticity and Integrity

Encryption Process

```
             Sender's                 Receiver's
            PrivateKey                 PublicKey
           ,-.                      ,-.
          (   )..........          (   )..........
           `-' ''''|'|'||           `-' ''''''''||
                   | |                      |
                   | |                      |
    .----------.   | |    .----------.      |     .----------.
    |          |   V |    |          |      V     |          |
    |Clear Text|--------->|CipherText|--------->|CipherText|
    |          |  Step1   |     1    |  Step2   |     2    |\
    `----------'     |    `----------'          `----------' \     __
          |          |                                        \   [_'
          |          |                                 step5 \    |
          |Step3     |                                     __   --|--
          |          |                                 _.--'      |
          V          |                            _..-''         / \
     .---------.     |    .---------.        _..-''           Receiver
     |  SHA 1  |     V    | Digital |  _..-''
     |MsgDigest|--------->|Signature|'             _
     `---------'  Step4   `---------'         _  (_)
         ______ _____    _____  _____ _   _ _____ _| |_ _   ____  _____
        | ___ |  _ \ / ___)/ ___) | | |  _ (_   _) |/ _ \|  _ \
        | _____| | | ( (___| |   | |_| | |_| || |_| | |_| | | | |
        |______)_| |_|\_____)_|    \__  |  __/  \__)_|\___/|_| |_|
                                (____/|_|
```

- Step1: In this step the Original "Clear Text" message is encrypted using the Sender's Private Key, which results in Cipher Text 1. This ensures the Authenticity of the sender.

- Step2: In this step the "CipherText 1" is encrypted using Receiver's Public Key resulting in "CipherText 2". This will ensure the Authenticity of the Receiver i.e. only the Receiver can decipher the Messsage using his Private Key.
- Step3: Here the SHA1 Message Digest of the "Clear Text" is created.
- Step4: SHA1 Message Digest is then encrypted using Sender's Private Key resulting in the Digital Signature of the "ClearText". This Digital Signature can be used by the receiver to ensure the Integrity of the message and authenticity of the Sender.
- Step5: The "Digital Signature" and the "CipherText 2" are then send to the Receiver.

Decryption Process

```
         Receiver's               Sender's
         PrivateKey               PublicKey
       ,-.                     ,-.
      (   ).........          (   ).........
       `-' '''''''''||         `-' ''''''''|||
              |                       |    |
              |                       |    |
.----------.  |     .----------.      |    |  .----------.
|          |  V     |          |      V    |  |          |      .---#1----.
|CipherText|------->|CipherText|--------------->|ClearText |----->|  SHA 1  |
|    2     | Step1  |    1     |   Step2   |  |          | Step3|MsgDigest|
`----------'        `----------'           |  `----------'      `---------'
                                           |                        ||
                                           |                        ||Step5
                                           |                        ||
                                           |                        ||
                           .---------.     |                    .---------.
                           | Digital |     V                    |  SHA 1  |
                           |Signature|------------------------->|MsgDigest|
                           `---------'     Step4                `---#2----'
               _                                  _
              | |                           _  (_)
            __| |_____  ____  ____ _   _ ____ _| |_ _  ___  ____
           / _  | ___ |/ ___)/ ___) | | |  _ (_   _) |/ _ \|  _ \
          ( (_| | ____( (___| |   | |_| | |_| || |_| | |_| | | | |
           \____|_____)\____)_|    \__  |  __/  \__)_|\___/|_| |_|
                                  (____/|_|
```

- Step1: In this step the "CipherText 2" message is decrypted using the Receiver's Private Key, which results in Cipher Text 1.
- Step2: In this step the "CipherText 1" is decrypted using Sender's Public Key resulting in "ClearText".
- Step3: Here the SHA1 Message Digest of the "Clear Text" is created.
- Step4: The "Digital Signature" is then decrypted using Sender's Public Key, resulting the "SHA 1 MSG Digest".
- Step5: The "SHA1 MsgDigest #1" is then compared against "SHA1 MsgDigest #2". If they are equal, the data was not modified during transmission, and the integrity of the Original "Clear Text" has been maintained

6.2. Test Certificates

While compiling Apache we created a test certificate. We used the makefile provided by mod_ssl to create this custom Certificate. We used the command:

```
# make certificate TYPE=custom
```

This certificate can be used for testing purposes.

6.3. Certificates for Production use

For production use you will need a certificate from a Certificate Authority (hereafter CA). Certificate Authorities are certificate vendors, who are listed as a Trusted CA in the user's browser. As mentioned in the Encryption Algorithms section, if the CA is not listed as a trusted authority, your user will get a warning message when trying to connect to a secure location.

Similarly the test certificates will also cause a warning message to appear on the user's browser.

6.4. How to generate a CSR

CSR or Certificate Signing Request must be sent to the trusted CA for signing. This section discusses howto create a CSR, and send it to the CA of your choice. **# openssl req** command can be used to a CSR as follows:

```
# cd /usr/local/apache/conf/
# /usr/local/ssl/bin/openssl req -new -nodes -keyout private.key -out public.csr
Generating a 1024 bit RSA private key
...........++++++
....++++++
writing new private key to 'private.key'
-----
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) []:San Jose
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Seagate
Organizational Unit Name (eg, section) []:Global Client Server
Common Name (eg, YOUR name) []:xml.seagate.com
Email Address []:saqib@seagate.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:badpassword
An optional company name []:
```

"PRNG not seeded"

If you do not have `/dev/random` on your system you will get a *"PRNG not seeded"* error message. In that case you can use the following command:

```
# /usr/local/ssl/bin/openssl req -rand some_file.ext -new -nodes -keyout private.key -out public.csr
```

Replace some_file.ext with the name of a existing file on your file system. Any file can be specified. Openssl will use that file to generate the seed

Solaris 9 comes with `/dev/random`. However on Solaris you might have to install the 112438 patch to get the /dev/random

At this point you will be asked several questions about your server to generate the Certificate Singning Request

Note: Your Common Name (CN) is the Fully Qualified DNS (FQDN) name of your webserver e.g. dav.server.com . If you put in anything else, it will NOT work. Remember the password that you use, for future reference.

Once the process is complete, you will have `private.key` and a `public.csr` . You will need to submit the `public.csr` to the Certification Authority. At this pointe the public.key is not encrypted. To encrypt:

```
# mv private.key private.key.unecrpyted
# /usr/local/ssl/bin/openssl rsa -in private.key.unecrpyted -des3 -out private.key
```

6.5. Installing Server Private Key, and Server Certificate

Once the Certification Authority processes your request, they will send an encoded certificate (Digital Certificate) back to you. The Digital Certificate is in the format defined by X.509 v3. The following shows the structure of a typical X509 v3 Digital Certificate

- Certificate
    - Version
    - Serial Number
    - Algorithm ID
    - Issuer
    - Validity
    -
        - Not Before
        - Not After
    - Subject
    - Subject Public Key Info
    -
        - Public Key Algorithm
        - RSA Public Key
    - Extensions
- Certificate Signature Algorithm
- Certificate Signature

6.5.1. Verifying a Digital Certificate

To verify a X.509 Certificate use the following command

```
# openssl verify server.crt
server.crt: OK
```

Where `server.crt` is the name of the file that contains the Digital Certificate

6.5.2. Viewing the contents of a Digital Certificate

The contents of a Digital Certificate can be viewed by using the **# openssl x509** command as follows:

```
# openssl x509 -text -in server.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 312312312 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, O=GTE Corporation, CN=GTE CyberTrust Root
        Validity
            Not Before: Feb  8 03:25:50 2000 GMT
            Not After : Feb  8 03:25:50 2001 GMT
            Subject: C=US, ST=New York, L=Pelham, O=xml-dev, OU=web, CN=www.xml-dev.com/Email=saqib@xml-dev.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                ...........
                ...........
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        ...........
        ...........
```

6.5.3. Modifying the httpd.conf to Install the Certificates

You will need to place this certificate on the server, and tell Apache where to find it.

For this example, the Private Key is placed in the `/usr/local/apache2/conf/ssl.key/` directory, and the Sever Certificate is placed in the `/usr/local/apache2/conf/ssl.crt/`.

Copy the file received from the Certification to a file called `server.crt` in the `/usr/local/apache2/conf/ssl.crt/`.

And place the private.key generated in the previous step in the `/usr/local/apache2/conf/ssl.key/`

Then modify the `/usr/local/apache2/conf/ssl.conf` to point to the correct Private Key and Server Certificate files:

```
#   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.  Keep
#   in mind that if you have both an RSA and a DSA certificate you
#   can configure both in parallel (to also allow the use of DSA
```

```
#   ciphers, etc.)
SSLCertificateFile /usr/local/apache2/conf/ssl.crt/server.crt
#SSLCertificateFile /usr/local/apache2/conf/ssl.crt/server-dsa.crt

#   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/apache2/conf/ssl.key/private.key
#SSLCertificateKeyFile /usr/local/apache2/conf/ssl.key/server-dsa.key
```

6.6. Removing passphrase from the RSA Private Key

RSA Private Key stored on the webserver is usually encrypted, and you need a passphrase to parse the file. That is why you are prompted for a passphrase when start Apache with modssl:

```
# apachectl startssl
Apache/1.3.23 mod_ssl/2.8.6 (Pass Phrase Dialog)
Some of your private key files are encrypted for security reasons.
In order to read them you have to provide us with the pass phrases.
Server your.server.dom:443 (RSA)
Enter pass phrase:
```

Encrypting the RSA Private Key is very important. If a cracker gets hold of your "Unencrypted RSA Private Key" he/she can easily impersonate your webserver. If the Key is encrypted, the cracker can not do anything without brute forcing the passphrase. Use of a strong (ie: long) passphrase is encouraged.

However encrypting the Key can sometimes be nuisance, since you will be prompted for a passphrase everytime you start the web−server. Especially if you are using rc scripts to start the webserver at boot time. The prompt for a passphrase will stop the boot process, waiting for your input.

You can get rid of the passphrase prompt easily by decrypting the Key. However make sure that no one can hold of this Key. I would recommend Hardening and Securing guidelines be followed before decrypting the Key on the webserver.

To decrypt the Key:

First make a copy of the encrypted key

```
# cp server.key server.key.cryp
```

Then re−write the key with encryption. You will be prompted for the original encrypted Key passphrase

```
# /usr/local/ssl/bin/openssl rsa −in server.key.cryp −out server.key
read RSA key
Enter PEM pass phrase:
writing RSA key
```

One way to secure the decrypted Private Key is to make readable only by the root:

```
# chmod 400 server.key
```

6.7. SSL Performance Tuning

6.7.1. Inter Process SSL Session Cache

Apache uses a multi–process model, in which all the request are NOT handled by the same process. This causes the SSL Session Information to be lost when a Client makes multiple requests. Multiple SSL HandShakes causes lot of overhead on the webserver and the client. To avoid this, SSL Session Information must be stored in a inter–process Session Cache, allowing all the processes to have access to to handshake information. SSLSessionCache Directive the in `/usr/local/apache2/conf/ssl.conf` file can be used to specify the location of the SSL Session Cache:

```
SSLSessionCache        shmht:logs/ssl_scache(512000)
#SSLSessionCache        shmcb:logs/ssl_scache(512000)
#SSLSessionCache         dbm:logs/ssl_scache
SSLSessionCacheTimeout  300
```

Using dbm:logs/ssl_scache creates the Cache as DBM hashfile on the local disk.

Using shmht:logs/ssl_scache(512000) creates the Cache in Shared Memory Segment

shmht vs shmcb

shmht: uses a Hash Table to Cache the SSL HandShake Information in the Shared Memory

shmht: uses a Cyclic Buffer to Cache the SSL HandShake Informationin the Shared Memory

Note:

Not all platforms/OS support creation of Hash table in the Shared Memory. So dbm:logs/ssl_scache must be used instead

6.7.2. Verifying SSLSession Cache

To verify if the SSLSessionCache is working properly, you can use the **openssl** utility with the **–reconnect** as follows:

```
# openssl s_client -connect your.server.dom:443 -state  -reconnect

CONNECTED(00000003)
.......
.......
Reused, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
SSL-Session:
.....
Reused, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
SSL-Session:
.....
Reused, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
SSL-Session:
.....
Reused, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
SSL-Session:
.....
Reused, TLSv1/SSLv3, Cipher is EDH-RSA-DES-CBC3-SHA
```

```
SSL-Session:
.....
```

–reconnect forces the s_client to connect to the server 5 times using the same SSL session ID. You should see 5 attempts of Reusing the same Session–ID as shown above.

A. HTTP/HTTPS Benchmarking tools

The following is a list of some of the OpenSource BenchMarking tools for WebServers

i. SSLswamp – For stress–testing/benchmarking connction to a SSL enable server
ii. HTTPERF – A Tool for Measuring Web Server Performance
iii. ab – Apache HTTP server benchmarking tool

B. Hardware based SSL encryption solutions

The following is a Hardware Based SSL encryption solution available:

i. CHIL (Cryptographic Hardware Interface Library) by nCipher
ii. ab – Apache HTTP server benchmarking tool

C. Certificate Authorities

The following is list of Certificate Authorities that are trusted by the various browsers:

i. Baltimore
ii. Entrust
iii. GeoTrust
iv. Thawte
v. TrustCenter

Glossary of PKI Terms

A

Asymmetric Cryptography

In this Cryptography a Key Pair – Private and Public Key is used. Private Key is kept secret and the Public Key is Widely distributed.

C

Certificate

A Data Record that contains the information as defined in the X.509 Format.

Certificate Authority (CA)

(CA)

Issuer of the Digital Certificate. Also validates the Identity of the End−Entity that posseses the Digital Certificate.

Certificate Signing Request (CSR)

(CSR)

Certificate Signing Request (CSR) is what you send to a Certifiate Authority (CA) to get enrolled. A CSR contains the Public Key of the End−Entity that is a requesting the Digital Certificate.

Common Name (CN)

(CN)

Common Name is the name of the End−Entity e.g. Saqib Ali. If the End−Entity is a WebServer the CN is the Fully Qualified Domain Name (FQDN) of the WebServer

D

Digital Certificate

A certificate that binds a Public Key to a Subject (end–entity). This certificate also contains other indentifying information about the subject as defined in the X.509 Format. It is signed by Issuing CA, using CA's pivate key. e.g. of a digital certificate

Digital Signature

A Digital Signature is created by signing the Message Digest (Message Hash) using the Private Key. It ensures the Identity of the Sender, and the Integrity of the Data.

E

End−Entity

An entity that participates in the PKI. Usually a Server, Service, Router, or a Person. A CA is not a End−Entity. An RA is an End−Entity to the CA

H

Hash

A hash is Hexadecimal number generated from a string of text such that, no two different strings can produce the same hash.

HMAC: Keyed Hashing for Message Authentication

(HMAC)

HMAC is an implementation of Message Authentication Code Algorithm.

M

Message Authentication Code

(MAC)

Similar to a Message Digest (Hash/Fingerprint), except the Shared Secret Key is used in the process of calculating the Hash. Since a shared secret key is used, an attacker can not change the Message Digest. However the shared secret key has to be first communicated to the participating entities, unlike Digital Signature where Message Digest is signed using the Private Key. HMAC is an example of a Message Authentication Code Algorithm.

Message Digest 5 − MD5

(MD5)

Message Digest 5 (MD5) is a 128−bit one−way hash function

P

Private Key

Private Key is the Key in Asymmetric Cryptography that is kept secret by the owner (End−Entity). Can be used for encryption or decryption

Public Key

Public Key is the Key in Asymmetric Cryptography that is widely distributed. Can be used for encryption or decryption

Public Key Infrastructure (PKI)

(PKI)
Public Key Infrastructure

S

SHA−1: Secure Hash Algorithm

(MD5)

Secure Hash Algorithm (SHA−1) is a 160−bit one−way hash function. Maximum message is 2^64 bits.

Secure Socket Layer (SSL)

(SSL)

Secure Socket Layer (SSL) is a security protocol that provides authentication (Digital Certificate), confidentiality (encryption), and data integrity (Message Digest − MD5, SHA etc).

Symmetric Cryptography

In this cryptography the message the encrypted and decrypted by the same key. (((n^2−n))/2) keys are required for n users who want to participate in this system of cryptography.