The Soothingly Seamless Setup of Apache, SSL, MySQL, and PHP

By Israel Denis Jr. and Eugene Otto

Table of Contents

Objective

Our objective is to install a web server that will allow us to host sites, that would be secure for e−commerce solutions, and that could be driven via scripts to connect to a database server and extract its data.

The tools for the job are:

- Apache −− A Web Server
- Mod_SSL −− A module for Secure Sockets Layer
- OpenSSL −− Open Source Toolkit (required for Mod_SSL)
- RSARef −− Only necessary for those in the US
- MySQL −− A Database Server
- PHP −− A Scripting Language

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

We are using just one of the many configurations that will fill our objective. We choose it because it is one of the simplest and fastest. We chose Apache because it is probably the most versatile web server ever to exist. Apache has the largest server market share partly because it is free, but mostly because it is the best. Mod_SSL and OpenSSL are added for the ability to conduct secure connections and because they are among the fastest and easiest to configure and setup. PHP is a simple, yet powerful scripting language which integrates easily with MySQL a powerful relational database server.

Hopefully after completing this guide, we will have achieved the following:

- Installed and setup the MySQL database server
  −> Know how to check the status of the MySQL server
  −> Know how to access the MySQL server using the command line client
  −> Know how to access your database server from the web
- Installed and setup the Apache web server with SSL
  −> Configure a simple virtual web site
  −> Know how to stop and start the server
  −> Know how to do some basic hosting configuration
- Installed and setup the PHP 4.0 Hypertext Preprocessor for server−side−scripting
  −> Know how to write simple php code
  −> Know how to connect to a DB using php
  −> Create a simple site that is enabled with PHP to talk to a database
- Create some sample certificates to use with Apache SSL
  −> Know how to generate a CSR file
  −> know how to encrypt a key
  −> Know how to sign your own certificates

In this article, quite a bit of information will be covered. We mean for it to be an introductory guide to get you started in the world of e−commerce, web site scripting, and Secure Sockets Layer (SSL), for the purpose of creating secure web sites driven by dynamic information stored in databases.

By no means is this article meant to be a detailed comprehensive document. We hope that it will help you get up and running with the aforementioned products, and hopefully get a better understanding of how everything

works.

Our goal is to write this document so that any newbie can understand what we are talking about. If we have accomplish this, then we have done a good job. If you walk away wanting to create e−commerce sites, than we have done an excellent job.... :−) send us some comments.

Assumptions

We assume that you have the following already installed on your system.

- Perl (Preferably version 5+)
- gzip or gunzip
- gcc and GNU make

If you don't have these installed, you will need to take the necessary steps to get them installed before any of the procedures in this document will make sense.

Prior programming knowledge is not required, however we assume that you are somewhat computer literate. You will also need a basic understanding of UNIX commands, HTML, and SQL. You should have a basic understanding of how to administer your Linux box. You will need a fully working Linux box on which you will install the software. Of course you'll need the necessary software packages listed above to compile the source code. Lastly, be sure that you don't already have MySQL, Apache, or PHP pre−installed on your Linux box.

Prerequisites

- Apache (Web Server)
- Mod_SSL (Secure Server Layer)
- OpenSSL (Toolkit for SSL)
- MySQL (SQL Database Server)
- PHP (Scripting Language)
- RSAref (Encryption for U.S. citizens) − search for at http://ftpsearch.lycos.com

The exact versions we used were: Apache 1.3.12, Mod_SSL 2.6.4−1.3.12, OpenSSL 0.9.5a, MySQL 3.22.32, PHP 4.0.0, and RSAref20.tar. We expect that the following procedures may be followed for future and past versions of these packages and will try to keep this tutorial updated.

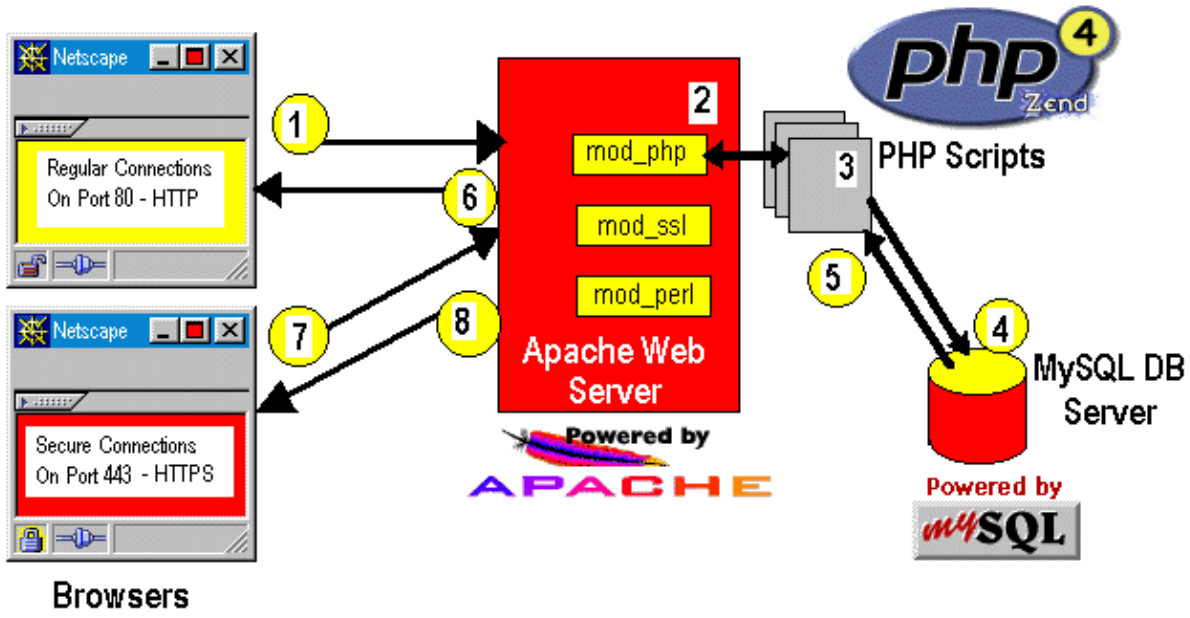
Download all tar file sources to a temporary directory. Make sure you put them somewhere with plenty of space. You should download them as root to avoid permissions problems.

If you'd like a quick and easy way to install your server, check out Ralph I. Zeller's tutorial on how to use RPMs. Thanks Ralph!

How it Works

It is helpful to understand what happens behind the scenes. Here is an over−simplification of how things work. The diagram below and the explanation that follows aren't entirely accurate but serve as a quick overview for now:

The scenario: We have a web page that pulls some data out of a DB. John Doe requests this page from his browser, the request is sent to the web server which in turn calls a PHP script. The PHP script is interpreted by the PHP preprocessor and pulls data from the database. The results are then processed by the rest of the PHP script and turned into HTML. The final HTML then gets sent back to the user's browser.



Let's look at this step by step:

1. John Doe clicks on a link to from his web browser; his web browser sends the request for http://localhost/test.php.
2. Apache gets the request for test.php. It knows that .php files are handled by the PHP preprocessor (mod_php) because we specified it in the Apache configuration file, so it tells PHP to deal with it.
3. test.php is a PHP script that contains commands. One of these commands is to open a connection to a database and grab data. PHP handles the connection to the database, and interprets the SQL calls to extract data from the database.
4. The database server gets the connection requests from the PHP interpreter, and processes the request. The request could be something like a simple select statement, or a table creation.
5. The database then sends the response and results back to PHP interpreter.
6. Apache sends the PHP information back to John Doe's browser, as the response to his request. John Doe now sees a web page containing some information from a database.

If this had been a request for https://localhost/test.php, then the process would be a little different.

1. Every request and response is encrypted and decrypted at both ends. That is, the browser connects to

Apache, obtains its encryption key, encrypts the request and sends it over.

2. The server sees the request decrypts it and authenticates it. It processes the file, encrypts it and sends it over. The browser then decrypts it with the server's key. Keep in mind that since the connections are encrypted, different ports are used. Port 80 used in the non−secure connection, while port 443 is used in the secure connection.

Again, that's not 100% correct but it's a very simplistic overview of what goes on behind the scenes. Now that you have a basic understanding of what we are trying to accomplish, let's get on to installing the software.

Game Plan

Our plan is first to install the MySQL server and ensure that it works. Then we'll install PHP and Mod_SSL. We will then install the Apache web server. Finally, after we have installed Apache we will test to see if PHP and Mod_SSL are functioning correctly.

You should note that:

- /usr/local/apache
- /usr/local/mysql
- /usr/local/ssl

are the locations we installed Apache, MySQL, and Mod_SSL/OpenSSL. You can install to different directories by changing the "prefix" option before installation.

Text that is indented and between horizontal rules is the expected input and output to and from your computer. The black text is what you enter. The red text means that you may have to change it to fit your circumstances. The green text is what the computer should display by itself.

MySQL Source Installation (*NIX)

The basic commands to unpack and install the MySQL source distribution from a `tar' file:

Become ROOT by using su.

```
# su
```

Change directly to where you have the tar file. (use a temp directory. I used /tmp/download/)

```
# cd /tmp/download/
```

Extract the files using the following command.

```
# gunzip −dc mysql−3.22.32.tar.gz | tar xvf −
```

Change to the NEW directory which was created during the extract.

```
# cd mysql−3.22.32
```

Now you can run "configure" for the MySQL server. You can specify many options with the configure command. Type "configure −−help" to see all options. We're using the "−−prefix" option to specify the direct path to the installation location. Configure will check for your compiler and a couple of other things. If you have any errors you can check the config.cache file to see the errors.

```
# configure −−prefix=/usr/local/mysql
```

After you are done with the configure. You can make the actual binaries by executing the following line (this will take a while).

```
# make
```

Now you are ready to install all the binaries. Run the following lines to install the binaries to the directory you specified with the configure "−−prefix" option.

```
# make install
```

Now it is time to create the mysql tables used to define the permissions. Make sure you replace "new−password" with something of your choice, otherwise, new−password will be your root password.

```
# scripts/mysql_install_db # cd /usr/local/mysql/bin # ./safe_mysqld & # ./mysqladmin −u root password 'new−password'
```

You can ensure that MySQL is working by running some simple tests to verify that the server is working. The output should be similar to what is shown below:

```
# BINDIR/mysqlshow −p
Enter password:
+−−−−−−−−−−−−−−−−−−−−+
| Databases          |
+−−−−−−−−−−−−−−−−−−−−+
| mysql              |
+−−−−−−−−−−−−−−−−−−−−+
```

Once you install MySQL, it will automatically create two databases. One is the mysql table which controls users, hosts, and database permissions in the actual server. The other is a test database. We could use the test database, however, we want to give you a quick and simple overview of some of the command line options available with MySQL. Also, this will ensure that root has been set up with full access to the database server (i.e. root has permission to create databases, tables, etc.) We will create a "test2" database that we will use later for our testing after logging into the MySQL server.

```
#mysql −u root −p
Enter password:
mysql> show databases;
+−−−−−−−−−−−−−−−−−−−−+
| Database           |
+−−−−−−−−−−−−−−−−−−−−+
| mysql              |
| test               |
+−−−−−−−−−−−−−−−−−−−−+
2 rows in set (0.00 sec)
mysql> create database test2;
Query OK, 1 row affected (0.00 sec)
```

Now select the test2 database, and create a new table called tst_tbl, with the two following fields. Field 1, which is an id field which lets you know the id of the record. Essentially, this is just a row number for simplicity. The second field is a name field in which you will store name information about books. The formats for these fields are.. field 1 (id) is an integer (int) of length 3, and field 2 (name) is a character (char) field of length 50. We assign id to be the key for searching and indexing the data.

y NOTE: MySQL commands are not case−sensitive. For example, CREATE and cReatE will be interpreted the same way. Also, remember to add a semi−colen after your commands.

```
mysql> use test2;
Database changed
mysql> CREATE TABLE books (
−> id int(3) not null auto_increment,
−> name char(50) not null,
−> unique(id),
−> primary key(id)
−> );
Query OK, 0 rows affected (0.00 sec)
```

Now we can verify that indeed everything is correct with the following commands.

```
mysql> show tables;
+--------------------------------+
| Tables in text2                |
+--------------------------------+
| books                          |
+--------------------------------+
1 row in set (0.00 sec)

mysql> describe books;

+--------+-------------+------+------+----------+-------------------------+
| Field  | Type        | Null | Key  | Default  | Extra                   |
+--------+-------------+------+------+----------+-------------------------+
| id     | int(3)      |      | PRI  | 0        | auto_increment          |
| name   | char(50)    |      |      |          |                         |
+--------+-------------+------+------+----------+-------------------------+

2 rows in set (0.00 sec)
```

Notice that the describe command basically "describes" the table layout. Pretty cool hey?!

Ok, time for some really useful SQL commands, inserting and selecting data from the database. It's time to add several records to the new table. Remember these are simple records of book names, but once you have gained enough experience with SQL you can create really complex databases for some great e−commerce sites. Let's create two records of two fictitious books. The first record is the name of a book I plan on writing some day, "PHP 4 Newbies." The other is a useful book for Linux, "Red Hat Linux 6 Server", by Mohammed J. Kabir.

```
mysql> INSERT INTO books (name) values('PHP 4 Newbies');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO books (name) values('Red Hat Linux 6 Server');
Query OK, 1 row affected (0.00 sec)
```

Now we can check the new records, and issue a select all command.

```
mysql> SELECT * from books;
+----+---------------------------------------+
| id | name                                  |
+----+---------------------------------------+
| 1  | PHP 4 Newbies                         |
| 2  | Red Hat Linux 6 Server                |
+----+---------------------------------------+
2 rows in set (0.00 sec)
```

Great, the MySQL server is fully functional. We could continue to add records, but it makes no sense at this time. Notice how you did not have to specify the id number when you inserted the record in the database. This is because you created the id field with the auto_increment option enabled.

Let's learn how do a quick delete. This is just for your info, remember that you can find everything you may need about the MySQL commands and the server at the mysql web site at http://www.mysql.com.

```
mysql> delete from books where id=1;
Query OK, 1 row affected (0.00 sec)
mysql> select * from books;
+----  +----------------------------------  +
| id     | name                               |
+----  +----------------------------------  +
| 2      | Red Hat Linux 6 Server             |
++----++-----------------------------------+
1 row in set (0.00 sec)
```

Ok, exit MySQL and continue with the setup...you can play with MySQL later after you have completed all the installations and everything is working properly.

PHP Installation (*NIX)

Now it's time to install the PHP (Hypertext Prerocessor) language.

You are still supposedly root, if not su back to root.

PHP requires that you have Apache pre−configured so that it knows where where everything is. You will come back to this later in the section when you setup the Apache server. Change back to the directory where you have the sources.

```
# cd /tmp/download
# gunzip −c apache_1.3.12.tar.gz | tar xf −
# cd apache_1.3.12
# ./configure −−prefix=/usr/local/apache
# cd ..
```

Ok, now you can start with PHP. Extract the source files and change to the new directory.

```
# gunzip −c php−4.0.0.tar.gz | tar xf −
# cd php−4.0.0
```

Configure will always be your friend if you are compiling code :−) So, again there are many options with the configure command. Use "configure −−help" to determine what you want to add. We just want MySQL and of course Apache.

```
# ./configure −−with−mysql=/usr/local/mysql \
−−with−xml \
−−with−apache=../apache_1.3.12 \
−−enable−track−vars
```

Make and install the binaries.

```
# make
# make install
```

Copy the ini file to the lib directory.

```
# cp php.ini−dist /usr/local/lib/php.ini
```

You can edit the PHP file to set PHP options. You could for example increase the "max_execution_time" in PHP by inserting the following line in your php.ini file.

```
max_execution_time = 60;
```

Apache &Mod_SSL

So far so good, time to configure and install Mod_SSL and Apache. If your server is going to be in the U.S., you will need to have the rsaref−2.0 files. You can search on http://ftpsearch.lycos.com/ for "rsaref20.tar.Z" Make sure you get the *nix distribution.

```
+-----------------------------------------------------------------------------------------------------------+
| IGNORE THE RSAREF SECTION IF YOU ARE NOT IN THE U.S.                                                      |
+-----------------------------------------------------------------------------------------------------------+
```

Create the "rsaref" directory where you will extract the files. Note that this assumes you have downloaded to the temp directory where you are currently at.

```
# mkdir rsaref−2.0
# cd rsaref−2.0
# gzip −dc ../rsaref20.tar.Z | tar xvf −
```

Now configure and build the OpenSSL library. When you're a U.S. citizen you have to build OpenSSL in conjunction with the RSAref library.

```
# cd rsaref−2.0
# cp −rp install/unix local
# cd local
# make
# mv rsaref.a librsaref.a
# cd ../..,
```

```
+-----------------------------------------------------------------------------------------------------------+
| END OF THE RSAREF SECTION, PLEASE CONTINUE READING                                                        |
+-----------------------------------------------------------------------------------------------------------+
```

Time to set up OpenSSL. Remember, this is what you will use to create temporary certificates and CSR files. The "−−prefix" specifies the main installation directory.

NOTE: Only include the "−L'pwd'/../rsaref−2.0/local/rsaref −fPIC'" line if you are a U.S. citizen.

```
# gunzip −dc openssl−0.9.5a.tar.gz | tar xvf −
# cd openssl−0.9.x
#./config −−prefix=/usr/local/ssl \
−L`pwd`/../rsaref−2.0/local/rsaref −fPIC
```

Now make it, test it, and install it.

```
# make
# make test
# make install
# cd ..
```

We will configure the Mod_SSL module and then specify it to be a loadable module with the Apache configuration.

```
# gunzip −dc mod_ssl−2.6.4−1.3.12.tar.gz |tar xvf −
# cd
# ./configure −−with−apache=../apache_1.3.12
# cd ..
```

Now we can add more Apache modules to the Apache source tree. The optional "−−enable−shared=ssl" option enables the building of mod_ssl as a DSO `libssl.so." Read the INSTALL and htdocs/manual/dso.html documents in the Apache source tree for more information about DSO support in Apache. We strongly advise ISPs and package maintainers to use the DSO facility for maximum flexibility with Mod_SSL, but notice that DSO is not supported by Apache on all platforms.

```
# cd apache_1.3.12
# SSL_BASE=../openssl−0.9.x \
RSA_BASE=../rsaref−2.0/local \
./configure \
−−enable−module=ssl \
−−activate−module=src/modules/php4/libphp4.a \
−−enable−module=php4 \
−−prefix=/usr/local/apache \
−−enable−shared=ssl
[...you can add more options here...]
```

Make Apache, then make certificates, and install...

```
# make
```

If you have done everything right you will a message similar to the following:

```
+−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
| Before you install the package you now should prepare the SSL
| certificate system by running the 'make certificate' command.
| For different situations the following variants are provided:
|
| % make certificate TYPE=dummy (dummy self−signed Snake Oil cert)
| % make certificate TYPE=test (test cert signed by Snake Oil CA)
| % make certificate TYPE=custom (custom cert signed by own CA)
| % make certificate TYPE=existing (existing cert)
| CRT=/path/to/your.crt [KEY=/path/to/your.key]
| Use TYPE=dummy when you're a vendor package maintainer,
| the TYPE=test when you're an admin but want to do tests only,
| the TYPE=custom when you're an admin willing to run a real server
| and TYPE=existing when you're an admin who upgrades a server.
| (The default is TYPE=test)
```

```
|
| Additionally add ALGO=RSA (default) or ALGO=DSA to select
| the signature algorithm used for the generated certificate.
|
| Use 'make certificate VIEW=1' to display the generated data.
|
| Thanks for using Apache & mod_ssl. Ralf S. Engelschall
| rse@engelschall.com
| www.engelschall.com
|
+-------------------------------------------------------------------------------------------------------
```

Now you can create a custom certificate. This option will prompt your for location, company, and a couple other things. Certificates are explained in a separate tutorial.

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

Now install Apache..

```
# make install
```

If everything went well, the message that you should see is something similar to this:

```
+-------------------------------------------------------------------------------------------------+
| You now have successfully built and installed the                                               |
| Apache 1.3 HTTP server. To verify that Apache actually                                          |
| works correctly you now should first check the                                                  |
| (initially created or preserved) configuration files                                            |
|                                                                                                 |
| /usr/local/apache/conf/httpd.conf                                                               |
| and then you should be able to immediately fire up                                              |
| Apache the first time by running:                                                               |
|                                                                                                 |
| /usr/local/apache/bin/apachectl start                                                           |
| Or when you want to run it with SSL enabled use:                                                |
|                                                                                                 |
| /usr/local/apache/bin/apachectl startssl                                                        |
| Thanks for using Apache. The Apache Group                                                       |
| http://www.apache.org/                                                                          |
+-------------------------------------------------------------------------------------------------+
```

Now it's time to see if Apache and PHP are working, however, we need to edit the httpd.conf of srm.conf to ensure we added the PHP type to the configuration.

Look at the httpd.conf and uncomment the following lines. If you have followed along with exactly the same instructions as this document, your httpd.conf file will be located in the "/usr/local/apache/conf" directory. The file has the addtype for PHP4 commented out, please uncomment it out at this time. It should look something like this:

```
>
> # And for PHP 4.x, use:
> #
---> AddType application/x-httpd-php .php
---> AddType application/x-httpd-php-source .phps
>
```

```
>
```

Now we are ready to start the Apache server to see if its working. First we will start the server without the SSL support to see if it comes up. We will check for PHP support and then we will stop the server and start it with the SSL support enabled and see if we got everything working. The configtest will check that all the configuration is setup properly.

```
# cd /usr/local/apache/bin
# ./apachectl configtest
Syntax OK
# ./apachectl start
./apachectl start: httpd started
```

Testing Our Work: Is Apache working?

If it worked correctly you will see something similar to this screen capture, when you connect to the server with Netscape. This is basically the Apache default installed page. Hopefully you will create a cool looking site, and change this page.

NOTE: You can connect to the server with a DOMAIN name or using the actual IP of the box. Check both cases, to ensure that everything is working properly.

Is PHP Support Working?

Now will test for PHP support. Create a file called test.php information below in it. The file needs to be located in the document root path which should be setup by default to /usr/local/apache/htdocs. Note that this is dependent on the prefix that we chose initially. However, this could be changed in the httpd.conf. Setting up multiple virtual hosting will be another article, keep your eyes open for it, as it will cover some very basic options in setting up Apache and its directives.

<? phpinfo(); ?>

This will display information about server, php, the environment, and a couple of other things. Here is a screen capture of the top portion of the output page.

Pretty cool. PHP rules!!!

Is SSL Working?

Ok, now we are ready to test for SSL. First, stop the server, and restart with the SSL option enabled.

```
# /usr/local/apache/bin/apachectl stop
# /usr/local/apache/bin/apachectl startssl
```

Test to see if it works, by connecting to the server with a Netscape and selecting the https protocol, (i.e. https://yourserver.yourdomain.com or http://yoursever.yourdomain.com:443 again). Also, try your server's ip (i.e. https://xxx.xxx.xxx.xxx: and http://xxx.xxx.xxx.xxx:443).

If it worked, the server will send the certificate to the browser to establish a secure connection which will make the browser prompt you for accepting the self signed certificate. If it were a certificate from VeriSign or Thawte then the browser will not prompt you because their certificates are coming from a trusted Certificate Authority (CA). In our case we created and signed our own certificates. We didn't want to buy one right away because we wanted first to ensure that everything worked properly.

You will see the following option enabled in Netscape. This tells you that a secure connection has been established.

Are PHP and MySQL Working Together?

Now, we can confirm that PHP is working with MySQL by creating a simple script to do some inserting and deleting of data from the "test2" database. Again this is just a simple script to see if it worked. In another article we will talk about PHP scripting to connect to a MySQL database. Remember, we already created the database and a table. We could had done it here, but chose not to. We wanted to double check that root had privileges to create DB and tables. However, PHP provides support for MySQL so we can easily write code to create a testing database and several records.

Remember, we created the "books" table prior to getting to this point. This portion will not work if you skipped prior sections. We created the "test2" database with a "books" table, and inserted a record for a book.

This script basically goes through the table and list all the fields by name. It's very simple.

```
<?

$dbuser = 'root';

$dbhost = 'localhost';

$dbpass = 'password';

$dbname = 'test2';

$dbtble = 'books';

$mysql_link = mysql_connect($dbhost,$dbuser,$dbpass);

$column = mysql_list_fields($dbname,$dbtble,$mysql_link);

for($i=0; $i< mysql_num_fields($column); $i++ )
{
print mysql_field_name($column,$i )."<br>";
}

?>
```

A more complex example will show you some of the cool features of PHP.

```
<html>
<head>
<title>Example 2 −− more details</title>
</head>
<body bgcolor="white">
<?

$dbuser = 'root';

$dbhost = 'localhost';

$dbpass = 'password';

$dbname = 'test2';

$dbtable = 'books';

//−−−−−− DATABASE CONNECTION −−−−−−−−//
$mysql_link = mysql_connect($dbhost,$dbuser,$dbpass);
$column = mysql_list_fields($dbname,$dbtable,$mysql_link);
```

```
$sql = "SELECT * FROM $dbtable";
$result = mysql_db_query($dbname,$sql);
?>
<table bgcolor="black">
<tr><td>
<table><!−−−− Inside Table −−−−>
<?
while($value = mysql_fetch_array($result))
{
print "<tr BGCOLOR=YELLOW>";
//This loop goes through the colums and prints
//each value
for($i=0; $i< mysql_num_fields($column); $i++ )
{
print "<td> $value[$i] </td>";
}
print "</tr>";
}
mysql_free_result($result);
mysql_close();
?>
</table><!−−−− Inside Table −−−−−>
</td></tr>
</table>

</body>
</html>
```

Notice, how you can have both HTML and PHP commands inside the same file, one of the nice things about PHP scripts. Because you begin with "

Conclusion

That was a lot of information for one article! Remember, there are quite a few assumptions in this simple guide, but we hope we have given you some understanding of how you can install Apache, PHP, Mod_SSL, and MySQL. We hope we have also given you some clue of how it all fits together.

Please share your comments with others by clicking on the discuss link associated with this document. For more information please visit the sites on the references section.

References:
www.apache.org
www.modssl.org
www.openssl.org
www.php.net
www.mysql.com
www.perl.com
www.cpan.org