OAMP
vs. 0.3

Rev3rse - www.revunix.tk
email: Rev3rse@revunix.tk

January 2005

1 License

2 Intro

OAMP= OpenBSD 3.5 + Apache 1.3 + MySQL 4 + PHP 4.3 :-)
The server Web will run in chroot jail with mod_ssl/2.8.16, OpenSSL/0.9.7c by default; we will also add mod_security.
This document born from personal notes of my studies, and it doesn't claim to be fully exhaustive. Therefore, I assume that whoever uses this informations, knows what he is doing...so, I will not speak about hardening the box, firewall rules or other stuff.
Thanks to BSD users for theirs passion. Excuse me for my bad english :P.
If you want contribute, write me an email to: Rev3rse@revunix.tk

3 Remember...

The size of the partitions on a server system greatly depends on the use of the system. If the server will be used for only one task, such as being a web server, then space considerations are easy to resolve. If, however, the system will be a database server, a mail server, and a web server, then obviously more must be considered. For each service that the machine will be running, space will be needed.
In a web server, for example, how much space is needed for logging?
Will web log processing software be running on the system that will generate more data as well?
In general, a lot of space should be allocated for the /var filesystem to allow for logging, even with log rotation in operation. Because the system will be a server, it won't need XF86 installed and may not even need the system source code or ports. Thus the /usr filesystem can be smaller and the /home filesystem can even be ignored given that there should be no users on the machine.

You can use symbolic links in the user's home directories pointing to subdirectories in /var/www, but you can NOT use links in /var/www pointing to other parts of the file system. That will not work due to the chroot(2) which Apache runs in.

If you want your users to have chrooted FTP access, this will not work, as the FTP chroot will (again) prevent you from accessing the targets of the symbolic links. A solution to this is to not use /home as your home directories for these users, but rather use something similar to /var/www/home.
FTP can be a real security risk for non-anonymous environments, but th ability to browse for files is a much-needed feature. For this reason, the folks who developed SSH also have developed *sftp* ;-)
Regarding sftp, I have written a document on how to make particular chrooted sftp accounts on OpenBSD, with scponly. You can find it on my site.

Apache's Httpd(8) daemon can only open new log files if restarted, or by other strategies, such as logging to a pipe(2), and using an external log rotator at the other end of the pipe.
One important use of piped logs is to allow log rotation without having to restart the httpd server.

The Apache HTTP Server includes a simple program called rotatelogs for this purpose.

For example, to rotate the logs every 24 hours, you can add the following lines in the httpd.conf file:

```
CustomLog "|/usr/sbin/rotatelogs \
/var/www/log/access_log 86400" common
```

After that you just have to remember to copy the rotatelogs bin file (/usr/sbin/rotatelogs) into the jail to: /var/www/usr/sbin.

Most CGIs will NOT work as is. They may need programs or libraries outside Apache's chroot, /var/www. Some can be fixed by compiling so they are statically linked (not needing libraries in other directories), most may be fixed by populating the /var/www directory with the files required by the application, though this is non-trivial and requires considerable programming knowledge.

In general, the minimum files needed to run an application should be copied into Apache's chroot at /var/www. Not every application can or should be chrooted.

The Apache server must run under a unique UID/GID, not used by any other system process; by default, in OpenBSD, Apache processes run with privileges of user *www* (except the main process, which runs with root privileges): uid=67(www) gid=67(www) groups=67(www), with shell /sbin/nologin.
In case of a successful "break-in", an intruder can obtain access to all other processes that run under the same UID/GID; hence, the optimum solution is to run Apache under the UID/GID of a unique regular user/group, dedicated to that software.

If there are other users who use the box I can choose to modify the permissions of the /var/www subdirectory. So, I have change permissions with:
`# chmod -R 750 /var/www/myweb`
`# chown -R mynick:www /var/www/mywebsite`.
I have also thought, for security reasons, that it is not a good thing that the www 'Apache' user has a home directory in /var/www. So, I have changed:

```
# chsh www
   ...
   Home directory:  /dev/null
   ...
```

3.1 About modules

The choice of modules is one of the most important steps of securing Apache, in general the rule is: "the less the better". That way we can avoid potential break-ins when new security vulnerabilities are found in one of them.
Apache on OpenBSD is compiled with some static modules and you can list installed modules with: `httpd -l`

The most important modules are:

- httpd_core
  The core Apache features, required in every Apache installation.
- mod_access
  Provides access control based on client hostname, IP address, or other characteristics of the client request. This module is needed to use "order", "allow" and "deny" directives too.
- mod_auth
  Required in order to implement user authentication using text files (HTTP Basic Authentication), which was specified in functionality assumptions.
- mod_dir
  Required to search and serve directory index files: index.html, default.htm, etc...
- mod_log_config
  Required to implement logging of the requests made to the server.
- mod_mime
  Required to set the character set, content- encoding, handler, content-language, and MIME types of documents.

Static compilation of modules instead of dynamic compilation will result in improved web server performance, but upgrading to a newer version of them later on will require recompilation of the whole web server. Compiling as dynamic modules, hasn't got this disadvantage but the performance of the web server is decreased by approximately 5%. Additionally, one more module would be needed: mod_so.
For now, we'll not recompile the software and use the dynamic approach.

3.2 General configuration

To run Apache automatically with SSL support if the system is restarted:

/etc/rc.conf.local

```
httpd_flags="-DSSL"
```

Edit httpd's configuration file. Following some main settings, in order like on configuration file; if you don't know about one option reade about it in httpd.conf or `man httpd`.

/var/www/conf/httpd.conf

```
ServerType standalone
ServerRoot "/var/www"

Port 80
```

```
<IfDefine SSL>
   Listen 80
   Listen 443
</IfDefine>

User www
Group www

ServerAdmin webmaster@yourname.bofh
ServerName www.yourname.bofh
DocumentRoot "/var/www/myweb"

<Directory />
   Options FollowSymLinks
   AllowOverride None
   Order deny,allow
   Deny from all
</Directory>

<Directory "/var/www/myweb">
   Order allow,deny
   Allow from all
</Directory>

<Directory "/var/www/myweb/download">
   Options Indexes
</Directory>

DirectoryIndex index.html index.htm

ServerTokens Full
UseCanonicalName On
ServerSignature Email

<Directory "/var/www/icons/">
   Options None
</Directory>

IndexOptions FancyIndexing
LanguagePriority en it fr de

NameVirtualHost *
<VirtualHost *>
   ServerName www.yourname.bofh
   DocumentRoot /var/www/myweb
</VirtualHost>
<VirtualHost *>
   ServerAdmin webmaster@yourname.bofh
   DocumentRoot /var/www/anotherweb
```

```
      ServerName www.secondname.bofh
      ErrorLog logs/anotherweb/anotherweb-error_log
      CustomLog logs/anotherweb/anotherweb-access_log common
   </VirtualHost>
```

If we want to create a directory with password of access:

```
# touch /var/www/password_file
# chown root:www /var/www/password_file
# chmod 640 /var/www/password_file
# htpasswd -c /var/www/password_file your_nick
...

# mkdir /var/www/myweb/private
# touch /var/www/myweb/private/.htaccess
# chown -R mynick:www /var/www/myweb/private
# chmod -R 750 /var/www/myweb/private

# vi /var/www/myweb/private/.htaccess
   AuthType Basic
   AuthName "Private Area"
   AuthUserFile /var/www/password_file
   require valid-user
```

We now insert the following lines in *http.conf* file:

```
   <Directory "/var/www/myweb/private">
      Options Indexes
      AllowOverride AuthConfig
   </Directory>

   AccessFileName .htaccess
   IndexIgnore .htaccess
```

That's all for right now. We'll further modify httpd.conf later.

4 SSL

Apache on OpenBSD is compiled to support SSL by default. Although the server and the configuration file are able to use SSL, keys need to be generated and the server needs to be started in a different way.

Generation of the SSL keys for the server is done in three steps:

1. Creation of the private key for the server.

   ```
   # openssl genrsa -out /etc/ssl/private/server.key 1024
   ```

2. Creation of the certificate request key; information about the site that this key will be protecting must be entered.

```
# openssl req -new -key /etc/ssl/private/server.key \
-out /etc/ssl/private/server.csr
```

3. The security behind SSL assumes that a trusted third party will sign the key. One such CA is Thawte Certification which you can reach at http://www.thawte.com/. As an alternative, the openssl tool, can allow the user to sign the key rather than having a public company sign it.

```
# openssl x509 -req -days 365 -in /etc/ssl/private/server.csr\
-signkey /etc/ssl/private/server.key -out /etc/ssl/server.crt
```

5 MySQL

MySQL is one of the most popular databases on the Internet and it is often used in conjunction with PHP. Besides its undoubted advantages such as ease of use and relatively high performance, MySQL offers simple but very effective security mechanisms. Unfortunately, the default installation of MySQL, and in particular the empty root password and the potential vulnerability to buffer overflow attacks, makes the database an easy target for attacks.
However, MySQL database is enough for our "little" web server, otherwise try to use postgresql. ;-)
We'll install this software from the OpenBSD package.

Base assumption:

- The MySQL database will be used only by PHP applications, installed on the same host
- The default administrative tools, such as MySQLAdmin, mysql, mysqldump etc...will be used to manage the database
- Remote data backup will be performed by utilizing the SSH protocol

The installation and configuration of MySQL should be performed in accordance with the following security requirements:

- MySQL processes must run under a unique UID/GID that are not used by any other system process
- Only local access to MySQL will be allowed
- MySQL root's account must be protected by a hard to guess password
- The administrator's account will be renamed
- Anonymous access to the database (by using the nobody account) must be disabled
- All sample databases and tables must be removed

Installing MySQL from OpenBSD package:

pkg_add *ftp://openbsd.org/...*

- *mysql-client-4.0.18*
- *p5-DBI-1.38*
- *mysql-server-4.0.18*
- *p5-DBD-mysql-2.90.03 (mysql-server dependence)*

The package automatically creates the user '_mysql' (uid 502) and group '_mysql' (gid 502) which are used for running the sql server.
To configure OpenBSD to automatically start mysql at every system start-up, I edit the files rc.conf.local and rc.local as follows:
/etc/rc.conf.local

```
# Start MySQL automatically at start-up:
mysql=YES
```

Somewhere in the life of mysql development, the libraries were moved from /usr/local/lib to their own directory /usr/local/lib/mysql. Because of this, we need to specify its location for the machine startup routines. We make these changes in rc.conf.local by modifying the reference to shlib_dirs:

```
httpd_flags="-DSSL"
shlib_dirs="$shlib_dirs /usr/local/lib/mysql"
```

Or, if you have more than one directory there, use the curly-braces like this:

```
shlib_dirs="$shlib_dirs /usr/local/lib/{db4,mysql,pkgconfig...}"
```

/etc/rc.local

```
# MySQL Settings
#
   rm -R /var/www/var/run/mysql
   mkdir -p /var/www/var/run/mysql
#
   if [ X"${mysql}" == X"YES" -a -x /usr/local/bin/mysqld_safe ];
      then echo -n "mysqld";
      /usr/local/bin/mysqld_safe --user=_mysql \
      --bind-address=127.0.0.1 --log=/var/mysql/
#
      for i in 1 2 3 4 5 6;
         do if [ -S /var/run/mysql/mysql.sock ];
            then break
            else sleep 1 echo -n "."
         fi done
#
      sleep 5
#
      ln -f /var/run/mysql/mysql.sock /var/www/var/run/mysql/mysql.sock
   fi
```

Start MySql and change the password for root. Set the root access password for the database.

```
# /usr/local/bin/mysqld_safe &
# /usr/local/bin/mysql -u root
mysql> SET PASSWORD FOR root@localhost=PASSWORD(″new_password″);
```

It is good practice not to change passwords from the command line, for example, by using the "mysqladmin password" command. This is especially important when other users work on the server. In that case the password could be easily revealed, e.g. by using the "ps aux" command or reviewing history files (/.history, /.bash_history etc), when improper access rights are set to them.

```
mysql> quit
# reboot
```

Verify the server is running by using the 'fstat' in the following example:

```
# fstat | grep mysql | grep 127
```
_mysql mysqld 22190 5 internet stream tcp 0xd0bc25a4 127.0.0.1:3306*

Now we know through fstat that the mysql daemon (mysqld) is running with user privileges of _mysql and listening on port 3306.
Our first test for validating the installation is to access the MySQL database server and look at the server maintenance database 'mysql.' We log in to the system through mysql interactive interface to the server.

```
# mysql -u root -p
Enter password:  ***********
Welcome to the MySQL monitor
...
mysql>
```

The `mysql>` prompt allows sql statements and MySQL commands to be entered. Most commands are completed by using the " ; " semi-colon delimiter. We check whether the initial database creation was successful (mysql, and test). The MySQL package should have created the system database 'mysql' and a sample database 'test'.

```
mysql> show databases;
...
```

We can check whether the mysql database has been installed by looking at the installed tables.

```
mysql> use mysql;
mysql> show tables;
...
```

The user table is the system wide table to record what users are allowed onto the MySQL database system and with what privileges. By using the 'describe' command we can see a list of the table fields and data-types.

```
mysql> describe user;
...
```

In this table it shows us the different levels of privileges available on the MySQL server.

Grabbing a set of information from the user table lets us see who has been given access to the system. Note the blank users is used by mysql for 'anonymous' and at the beginning only −−user=root has privileges to do anything on the system. Note that the password field is encrypted with a one-way encryption system similar but not identical to the unix crypt() function.

Check to see if user root has access to 127.0.0.1

```
mysql> select host, user, select_priv, grant_priv, password from user;
...
```

If you make a mistake you can always delete user access by the following code:

```
mysql> use mysql;
mysql> delete from user where user = user to delete;
```

First to quit, we must remove the simple database (test) and alla accounts, except the local root account:

```
mysql> use mysql;
mysql> drop database test;
mysql> delete from user where user=″;
mysql> delete from db where user=″;
mysql> flush privileges;
```

This will prevent the database from establishing anonymous connections and remote connection as well.

```
mysql> quit
# mysqladmin -p shutdown
```

5.1 MySQL security

- *Remove history*
  Before proceeding any further, we should remove the content of the MySQL history file (/.mysql_history), in which all executed SQL commands are being stored (especially passwords, which are stored as plain text):

  ```
  cat /dev/null >  /.mysql_history
  ```

Now we must configure the database server and add some security.
The main configuration file is /etc/my.cnf.
You can use /usr/local/share/mysql/my-medium.cnf or create a new one; following the most important settings.

- *Disable remote access*
  The first change applies to the 3306/tcp port, on which MySQL listens by default. Because, according to the initial assumptions, the database will be used only by locally installed PHP applications, we can freely disable listening on that port. This will limit possibilities of attacking the MySQL database by direct TCP/IP connections from other hosts. Local communication will still be possible through the mysql.sock socket. In order to disable listening on the mentioned port, the following parameter should be added to the `[mysqld]` section

  ```
  skip-networking
  ```

  If, for some reason, remote access to the database is still required (e.g. to perform remote data backup), the SSH protocol can be used as follows:

  ```
  ssh mysqlserver /usr/local/mysql/bin/mysqldump -A > backup
  ```

- *Improve local security*
  The next change is to disable the use of LOAD DATA LOCAL INFILE command, which will help to prevent against unauthorized reading from local files. This matters especially when new SQL Injection vulnerabilities in PHP applications are found.
  For that purpose, the following parameter should be added in the `[mysqld]` section:

  ```
  set-variable=local-infile=0
  ```

- *Change admin name*
  It is also recommended to change the default name of administrator's account (root), to a different, harder to guess one. Such a change will make it difficult to perform brute-force and dictionary attacks on the administrator's password. In this case the intruder will have to guess not only the password, but first and foremost, the name of the administrator's account.

  ```
  mysql> update user set user=″mydbadmin″ where user=″root″;
  mysql> flush privileges;
  ```

Some options on system's stability:

- *binary logging and slowly query*
  First option is a good practice to restore from breaking off. Second option will help us on application's optimize.

  ```
  log-bin
  log-slow.queries
  ```

  If you are not practical with mysql, and you don't know whether to create databases, you can initially leave the following lines for InnoDB Tables commented.

- *InnoDB Tables*

  ```
  innodb_data_file_path=ibdata1:10M:autoextend
  set-variable = innodb_buffer_pool_size=70M
  set-variable = innodb_additional_mem_pool_size=10M
  set-variable = innodb_log_file_size=20M
  set-variable = innodb_log_buffer_size=8M
  innodb_flush_log_at_trx_commit=1
  ```

Finally, we'll add the following lines:

```
[client]
socket = /var/www/var/run/mysql/mysql.sock
```

At this point we can create all databases and accounts which will be used by specific PHP applications.
It should be emphasized that these accounts should have access rights only to the databases which are used by the PHP applications. In particular, they should not have any access rights to the mysql database, or any system or administrative privileges (FILE, GRANT, ALTER, SHOW DATABASE, RELOAD, SHUTDOWN, PROCESS, SUPER etc.).

6 PHP

PHP, one of the most popular scripting languages used to create dynamic web pages. For that:

- The web server must handle the PHP scripting language
- The PHP component must be able to read and write users' data in a locally installed MySQL database

Security assumptions are:

- The Apache server must reject all requests (GET and POST), which contain HTML tags (possible Cross-Site-Scripting attack) or apostrophe/quotation marks (possible SQL Injection attack).
- No PHP warning or error messages should be shown to the web application's regular users.

- It should be possible to store incoming GET and POST requests into a text file which will make it possible to use additional, host-based intruder detection system (HIDS).

The main package you need to install is *php4-core-4.3.5RC3*, which contains the basic engine (plus gettext, iconv and recode). Next, take a look at the module packages, such as *php4-mysql-4.3.5RC3* or *php4-imap-4.3.5RC3*. You need to use the phpxs command to activate and deactivate these modules in your php.ini.
Another package is *mod_security-1.7.5*: the module will be used to implement the protection against CSS and SQL injection attacks.

pkg_add *ftp://openbsd.org/...*

- *mod_security-1.7.5*

  Check if the following line has been added in the httpd.conf file, otherwise you insert this by hand:

  ```
  LoadModule security_module /usr/lib/apache/modules/mod_security.so
  ```

  One configuration base:

  ```
  <IfModule mod_security.c>
  AddHandler application/x-httpd-php .php
  SecAuditEngine On
  SecAuditLog logs/audit_log
  #SecFilterScanPOST On
  SecFilterEngine On
  SecFilterDefaultAction deny,log,status:500
  SecFilterCheckURLEncoding On
  SecFilterCheckUnicodeEncoding Off
  SecFilterForceByteRange 1 255
  SecFilterDebugLevel 0
  # Command execution attacks
  SecFilter /etc/password
  SecFilter /bin/ls
  # Directory traversal attacks
  SecFilter "\.\./"
  # XSS attacks
  SecFilter "<(.|\n)+>"
  SecFilter "'"
  SecFilter "\" "
  SecFilter "<[[:space:]]*script"
  # SQL injection attacks
  SecFilter delete[[:space:]]+from"
  SecFilter insert[[:space:]]+into"
  SecFilter select.+from"
  </IfModule>
  ```

pkg_add *ftp://openbsd.org/...*

- *php4-core-4.3.5RC3*
  ```
  # /usr/local/sbin/phpxs -s
  # cp /usr/local/share/doc/php4/php.ini-recommended /var/www/conf/php.ini
  # chown root:www /var/www/conf/php.ini
  # chmod 640 /var/www/conf/php.ini
  ```
- *php4-mysql-4.3.5RC3*
  ```
  # /usr/local/sbin/phpxs -a mysql
  ```
- *c-client-4.53*
- *php4-imap-4.3.5RC3*
  ```
  # /usr/local/sbin/phpxs -a imap
  ```
- *php4-pear-4.3.5RC3*
  ```
  mkdir /var/www/php
  mkdir /var/www/php/includes
  cp -pR /usr/local/lib/php/* /var/www/php/includes
  ```

Check if the following line has been added in the httpd.conf file, otherwise you insert this by hand:

```
LoadModule php4_module /usr/lib/apache/modules/libphp4.so
```

Then check this:

```
<IfModule mod_php4.c>
   AddType application/x-httpd-php .php .php4 .php3 .htm .html
   AddType application/x-httpd-php-source .phps
</IfModule>

DirectoryIndex index.html index.htm index.php index.php4 index.php3
```

Finally check:

```
# apachectl stop
# apachectl startssl
# apachectl status
```

6.1 Common problems

The default install of Apache runs inside a chroot(2) jail, which will restrict PHP scripts to accessing files under /var/www. You will therefore need to create a /var/www/tmp directory for PHP session files to be stored, or use an alternative session backend.
In addition, database sockets need to be placed inside the jail or listen on the localhost interface. If you use network functions, some files from /etc such as /etc/resolv.conf and /etc/services will need to be moved into /var/www/etc.
The OpenBSD PEAR package automatically installs into the correct chroot directories, so no special modification is needed there.

6.2 Test PHP

Make your test file:

```
# vi /var/www/htdocs/phptest.html

   <?php
   phpinfo();
   ?>
```

or try to run Apache and check if PHP can properly communicate with MySQL. We can achieve this by using the sample "test.php" script with the following content (the "user_name" and "password" values should be changed in accordance with installed database):

```
   <html><body>
   <?php
      $link = mysql_connect(‘‘localhost’’, ‘‘mydbadmin’’, ‘‘password’’)
      or die;
      print ‘‘Everything works OK!’’;
      mysql_close($link);
   ?>
   </body></html>
```

Save it and restart apache or reboot the computer.

```
# apachectl restart
```

The above web page can be viewed by using any Internet browser. If PHP instructions are properly interpreted and a connection to MySQL is established, we can start securing the software. If not, we should analyze the Apache and MySQL log files and eliminate the cause of the problems.
Use your favorite web browser to check to see if it worked.

```
# lynx localhost/test.php
```

A few changes must also be made in the PHP configuration file (php.ini).
The most important changes that should be made to improve PHP security are follows:

- `safe_mode= On`
  By enabling safe_mode parameter, PHP scripts are able to access files only when their owner is the owner of the PHP scripts. This is one of the most important security mechanisms built into the PHP. Effectively counteracts unauthorized attempts to access system files (e.g. /etc/paswd) and adds many restrictions that make unauthorized access more difficult.

- `safe_mode_gid = Off`
  When safe_mode is turned on and safe_mode_gid is turned off, PHP scripts are able to access files not only when UIDs are the same, but also when the group of the owner of the PHP script is the same as the group of the owner of the file.

- `open_basedir = directory[:...]`
  When the open_basedir parameter is enabled, PHP will be able to access only those files, which are placed in the specified directories (and subdirectories).

- `safe_mode_exec_dir = directory[:...]`
  When safe_mode is turned on, system(), exec() and other functions that execute system programs will refuse to start those programs, if they are not placed in the specified directory.

- `expose_php = Off`
  Turning off the "expose_php" parameter causes that PHP will not disclose information about itself in HTTP headers that are being sent to clients in responses to web requests.

- `register_globals = Off`
  When the register_globals parameter is turned on, all the EGPCS (Environment, GET, POST, Cookie and Server) variables are automatically registered as global variables. Because it can pose a serious security threat, it is strongly recommended to turn this parameter off (starting from the version 4.2.0, this parameter is turned off by default)

- `display_errors = Off`
  If the display_errors parameter is turned off, PHP errors and warnings are not being displayed. Because such warnings often reveal precious information like path names, SQL queries etc., it is strongly recommended to turn this parameter off on production servers.

- `log_errors = On`
  When log_errors is turned on, all the warnings and errors are logged into the file that is specified by the error_log parameter. If this file is not accessible, information about warnings and errors are logged by the Apache server.

- `error_log = filename`
  This parameter specifies the name of the file, which will be used to store information about warnings and errors (attention: this file must be writeable by the user or group apache)

In addition, changing the file extension can be taken into account. For example *.php to *.asp, *.dhtml or even *.html. Such a change will make it difficult for any potential intruders to recognize the server-side technology that is being used. In order to change the extensions, all the *.php files should be renamed to *.dhtml (for example), and the following line should be changed in /var/www/conf/httpd.conf:

```
AddType application/x-httpd-php .php
```

to the new one:

```
AddType application/x-httpd-php .dhtml
```

Thanks to that, web users will not see *.php extension in the URL address which is what immediately suggests that the PHP technology is being used at the server side.

7 Regards...

...to you that you have read this article, to all of my friends, to the unix developers.
Bye!

Rev‘