

# Securing Apache: Step-by-Step

Artur Maj 2003-05-14

This article shows in a step-by-step fashion, how to install and configure the Apache 1.3.x Web server in order to mitigate or avoid successful break-in when new vulnerabilities in this software are found.

## Functionality

Before we start securing Apache, we must specify what functionality we expect from the server. Variety of Apache's use makes it difficult to write a universal procedure to secure the server in every case. That's why in this article we'll base on the following functionality:

- The Web server will be accessible from the Internet; and,
- Only static HTML pages will be served
- the server will support name-based virtual hosting mechanism
- specified Web pages can be accessible only from selected IP addresses or users (basic authentication)
- the server will log all the Web requests (including information about Web browsers)

It is worth emphasizing that the above model doesn't support PHP, JSP, CGI or any other technologies that make it possible to interact with Web services. The use of such technologies may pose a large security threat, so that even a small, inconspicuous script can radically decrease the server's security level. Why? Primarily, ASP/CGI applications may contain security vulnerabilities (e.g. SQL injection, cross-site-scripting). Secondly, the technology itself can be dangerous (vulnerabilities in PHP, Perl modules etc.). That's why I strongly recommend using such technologies only when an interaction with a Web site is absolutely necessary.

## Security Assumptions

One of the most important elements of every computer project is the specification of security assumptions. This must be fulfilled before the project is implemented. The security assumptions for our Web server are as follows:

- The operating system must be hardened as much as possible, both against local and remote attacks;
- The server must not offer any network services except HTTP: (80/TCP);

- Remote access to the server must be controlled by a firewall, which should block all outbound connections, and allow inbound connections only to the 80/TCP port of the Web server;
- The Apache Web server must be the only service available on the system;
- Only absolutely necessary Apache modules should be enabled;
- Any diagnostic Web pages and automatic directory indexing service must be turned off;
- The server should disclose the least amount of information about itself (security by obscurity);
- The Apache server must run under a unique UID/GID, not used by any other system process;
- Apache's processes must have limited access to the file systems (chrooting); and,
- No shell programs can be present in the Apache's chrooted environment (/bin/sh, /bin/csh etc.).

## Installing the Operating System

Before installing Apache we must choose an operating system, upon which the server will run. We've got a broad choice here, because Apache can be compiled and installed on almost every operating system. The rest of the article instructs how to secure the Apache Web server on FreeBSD (4.7), however the described methods are possible to apply in case of most UNIX/Linux systems. The only operating system I do not recommend using is MS Windows - mainly because of the limited capabilities of securing the Apache.

The first step in securing the Web server is hardening the operating system. A discussion of hardening the operating system is beyond the scope of this article. However, there are a lot of documents on the Net describing how to perform that. Readers are encouraged to conduct their own issue on this topic.

After the system is installed and hardened, we have to add a new group and regular user called "apache" like this (an example from FreeBSD):

```
pw groupadd apache
pw useradd apache -c "Apache Server" -d /dev/null -g apache -s /sbin/nologin
```

By default, Apache processes run with privileges of user *nobody* (except the main process, which runs with *root* privileges) and GID of group *nogroup*. This might pose a significant security threat. In case of successful break-in, the 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.

## Preparing the Software

The next step is to download the latest version of the [Apache Web server](#). Some of Apache's options can be enabled only during compilation time, thus it is important to download the source code instead of the binary version.

After downloading the software, we must unpack it. Then we must decide which modules should remain enabled. A short description of all modules available in the latest version of Apache 1.3.x (1.3.27) can be found at <http://httpd.apache.org/docs/mod/>.

## Apache's Modules

The choice of modules is one of the most important steps of securing Apache. We should go by the rule: the less the better. To fulfill the functionality and security assumptions, the following modules must remain enabled:

Module's name	Description
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. Because this module is needed to use "order", "allow" and "deny" directives, it should remain enabled.
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.

All other Apache's modules must be disabled. We can safely turn them off, mainly because we do

not need them. By disabling unneeded modules, we can avoid potential break-ins when new security vulnerabilities are found in one of them.

It is also worth to note that two of Apache's modules can be more dangerous than others: *mod\_autoindex* and *mod\_info*. The first module provides for automatic directory indexing, and is enabled by default. It is very easy to use this module in order to check if Apache runs on a server (e.g. `http://server_name/icons/`) and to get the content of the Web server's directories, when no index files are found in them. The second module, *mod\_info*, should never be accessible from the Internet, mainly because it reveals the Apache server's configuration.

The next question is how to compile modules. The static method seems to be a better choice. If new vulnerabilities in Apache are found, we will probably recompile not just the vulnerable modules, but the whole software. By choosing the static method, we eliminate the need of one more module - *mod\_so*.

## Compiling the software

First of all - if exist - any security patches must be applied. Then, the server should be compiled and installed as follows:

```
./configure --prefix=/usr/local/apache --disable-module=all --server-  
uid=apache --server-gid=apache --enable-module=access --enable-  
  
module=log_config --enable-module=dir --enable-module=mime --enable-module=auth  
  
make  
su  
umask 022  
make install  
chown -R root:sys /usr/local/apache
```

## Chrooting the server

The next step is to limit Apache processes' access to the filesystems. We can achieve that by chrooting it's main daemon (httpd). Generally, the chrooting technique means creating a new root directory structure, moving all daemon files to it, and running the proper daemon in that new environment. Thanks to that, the daemon (and all child processes) will have access only to the

new directory structure.

We'll start this process by creating a new root directory structure under the /chroot/httpd directory:

```
mkdir -p /chroot/httpd/dev
mkdir -p /chroot/httpd/etc
mkdir -p /chroot/httpd/var/run
mkdir -p /chroot/httpd/usr/lib
mkdir -p /chroot/httpd/usr/libexec
mkdir -p /chroot/httpd/usr/local/apache/bin
mkdir -p /chroot/httpd/usr/local/apache/logs
mkdir -p /chroot/httpd/usr/local/apache/conf
mkdir -p /chroot/httpd/www
```

The owner of all above directories must be root, and the access rights should be set to the 0755. Next, we'll create the special device file: /dev/null:

```
ls -al /dev/null
  crw-rw-rw-  1 root wheel  2,  2 Mar 14 12:53 /dev/null
mknod /chroot/httpd/dev/null c 2 2
chown root:sys /chroot/httpd/dev/null
chmod 666 /chroot/httpd/dev/null
```

A different method must be used to create a /chroot/httpd/dev/log device, which is also needed for the server to work properly. In case of the FreeBSD system, the following line should be added to the /etc/rc.conf:

```
syslogd_flags="-l /chroot/httpd/dev/log"
```

We must restart the system or the syslogd daemon itself for the changes to take effect. In order to create a /chroot/httpd/dev/log device on other operating systems, we must take a look at the proper manuals (man syslogd).

The next step is to copy the main httpd program into the new directory tree with all necessary binaries and libraries. In order to do that, we must prepare the list of all required files. We can make such list by using the following commands (their presence depends on particular operating

system):

Command	Availability	Description
ldd	All	Lists dynamic dependencies of executable files or shared libraries
ktrace/ktruss/kdump	*BSD	Enables kernel process tracing, Displays kernel trace data
sotruss	Solaris	Traces shared library procedure calls
strace/ltrace	Linux	Traces system calls and signals
strings	All	Finds the printable strings in binary files
trace	AIX	Records selected system events
trace (freeware)	HP-UX <10.20	Print system call and kernel traces of processes
truss	FreeBSD, Solaris, AIX 5L, SCO Unixware	Traces system calls and signals
tusc (freeware)	HP-UX >11	Traces the system calls a process invokes in HP-UX 11

Examples of using ldd, strings and truss commands are shown below:

```
localhost# ldd /usr/local/apache/bin/httpd
/usr/local/apache/bin/httpd:
    libcrypt.so.2 => /usr/lib/libcrypt.so.2 (0x280bd000)
    libc.so.4 => /usr/lib/libc.so.4 (0x280d6000)
```

```
localhost# strings /usr/local/apache/bin/httpd | grep lib
/usr/libexec/ld-elf.so.1
libcrypt.so.2
libc.so.4
```

```
localhost# truss /usr/local/apache/bin/httpd | grep open
```

```
(...)
open("/var/run/ld-elf.so.hints",0,00)           = 3 (0x3)
open("/usr/lib/libcrypt.so.2",0,027757775370) = 3 (0x3)
open("/usr/lib/libc.so.4",0,027757775370)     = 3 (0x3)
open("/etc/spwd.db",0,00)                     = 3 (0x3)
open("/etc/group",0,0666)                     = 3 (0x3)
open("/usr/local/apache/conf/httpd.conf",0,0666) = 3 (0x3)
(...)
```

The above commands should be applied not only to the httpd program, but also to all of the libraries and binaries required (libraries often require other libraries). In case of FreeBSD system, the following files have to be copied to the new root directory structure:

```
cp /usr/local/apache/bin/httpd /chroot/httpd/usr/local/apache/bin/
cp /var/run/ld-elf.so.hints /chroot/httpd/var/run/
cp /usr/lib/libcrypt.so.2 /chroot/httpd/usr/lib/
cp /usr/lib/libc.so.4 /chroot/httpd/usr/lib/
cp /usr/libexec/ld-elf.so.1 /chroot/httpd/usr/libexec/
```

By using the truss command we can also discover that the following configuration files must be present in the chrooted environment as well:

```
cp /etc/hosts /chroot/httpd/etc/
cp /etc/host.conf /chroot/httpd/etc/
cp /etc/resolv.conf /chroot/httpd/etc/
cp /etc/group /chroot/httpd/etc/
cp /etc/master.passwd /chroot/httpd/etc/passwords
cp /usr/local/apache/conf/mime.types /chroot/httpd/usr/local/apache/conf/
```

Note, that from /chroot/httpd/etc/passwords we have to remove all the lines except "nobody" and "apache". In a similar way, we must remove all the lines except "apache" and "nogroup" from /chroot/httpd/etc/group. Next, we have to build the password database as follows:

```
cd /chroot/httpd/etc
pwd_mkdb -d /chroot/httpd/etc passwords
rm -rf /chroot/httpd/etc/master.passwd
```

The next step is to test if the httpd server runs correctly in the new chrooted environment. In order to perform that, we have to copy the default Apache configuration file and sample index.html:

```
cp /usr/local/apache/conf/httpd.conf /chroot/httpd/usr/local/apache/conf/
cp /usr/local/apache/htdocs/index.html.en /chroot/httpd/www/index.html
```

After copying the aforementioned files, we must change the DocumentRoot directive as presented below (in /chroot/httpd/usr/local/apache/conf/httpd.conf):

```
DocumentRoot "/www"
```

Next, we can try to run the server:

```
chroot /chroot/httpd /usr/local/apache/bin/httpd
```

If any problems occur, I recommend analyzing Apache's log files precisely (/chroot/httpd/usr/local/apache/logs). Alternatively, the following command can be used:

```
truss chroot /chroot/httpd /usr/local/apache/bin/httpd
```

The truss program should show the cause of the problems. After eliminating any eventual faults, we can configure the Apache server.

## Configuring Apache

The first step is to remove the /chroot/httpd/usr/local/apache/conf/httpd.conf file and create a new one in its place, with content similar to the following:

```
# =====
# Basic settings
# =====
ServerType standalone
ServerRoot "/usr/local/apache"
PidFile /usr/local/apache/logs/httpd.pid
ScoreBoardFile /usr/local/apache/logs/httpd.scoreboard
ResourceConfig /dev/null
```

AccessConfig /dev/null

# =====

# Performance settings

# =====

Timeout 300

KeepAlive On

MaxKeepAliveRequests 100

KeepAliveTimeout 15

MinSpareServers 5

MaxSpareServers 10

StartServers 5

MaxClients 150

MaxRequestsPerChild 0

# =====

# Apache's modules

# =====

ClearModuleList

AddModule mod\_log\_config.c

AddModule mod\_mime.c

AddModule mod\_dir.c

AddModule mod\_access.c

AddModule mod\_auth.c

# =====

# General settings

# =====

Port 80

User apache

Group apache

ServerAdmin Webmaster@www.ebank.lab

UseCanonicalName Off

ServerSignature Off

HostnameLookups Off

ServerTokens Prod

<IfModule mod\_dir.c>

```
    DirectoryIndex index.html
</IfModule>
DocumentRoot "/www/vhosts"

# =====
# Access control
# =====

<Directory />
    Options None
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
<Directory "/www/vhosts/www.ebank.lab">
    Order allow,deny
    Allow from all
</Directory>
<Directory "/www/vhosts/www.test.lab">
    Order allow,deny
    Allow from all
</Directory>

# =====
# MIME encoding
# =====

<IfModule mod_mime.c>
    TypesConfig /usr/local/apache/conf/mime.types
</IfModule>
DefaultType text/plain

<IfModule mod_mime.c>
    AddEncoding x-compress Z
    AddEncoding x-gzip gz tgz
    AddType application/x-tar .tgz
</IfModule>

# =====
```

```

# Logs
# =====
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
ErrorLog /usr/local/apache/logs/error_log
CustomLog /usr/local/apache/logs/access_log combined

# =====
# Virtual hosts
# =====
NameVirtualHost *
<VirtualHost *>
    DocumentRoot "/www/vhosts/www.ebank.lab"
    ServerName "www.ebank.lab"
    ServerAlias "www.e-bank.lab"
    ErrorLog logs/www.ebank.lab/error_log
    CustomLog logs/www.ebank.lab/access_log combined
</VirtualHost>
<VirtualHost *>
    DocumentRoot "/www/vhosts/www.test.lab"
    ServerName "www.test.lab"
    ErrorLog logs/www.test.lab/error_log
    CustomLog logs/www.test.lab/access_log combined
</VirtualHost>

```

The above configuration includes only the commands that are necessary to fulfill the functionality and security assumptions. In the configuration presented, there are two virtual hosts supported by the Web server:

- www.ebank.lab (www.e-bank.lab)
- www.test.lab

The content of the above Web sites is physically present in the following directories:

- /chroot/httpd/www/vhosts/www.ebank.lab
- /chroot/httpd/www/vhosts/www.test.lab

Each Web site has its own log files, which are present in the following directories:

- /chroot/httpd/usr/local/apache/logs/www.ebank.lab
- /chroot/httpd/usr/local/apache/logs/www.test.lab

The above directories must be created before running the Apache for the first time - otherwise the Apache will not run correctly. The owner of the above directories should be root:sys, and the rights should be set to 0755.

Compared with the default Apache configuration file, the following changes have been made:

- the number of enabled modules has been significantly reduced
- Apache doesn't disclose information about its version number (directives: ServerTokens, ServerSignature)
- Apache's processes (except the root process) are set to be executed with unique regular user's/group's privileges (directives: User, Group)
- Apache will allow access only to the directories, subdirectories and files, which are explicitly specified in the configuration file (directives: Directory, Allow); all other requests will be denied by default
- Apache will log more information about HTTP requests

## Final steps

At the end we should create a start-up script "apache.sh", the content of which will be similar to the following:

```
#!/bin/sh
```

```
CHROOT=/chroot/httpd/
```

```
HTTPD=/usr/local/apache/bin/httpd
```

```
PIDFILE=/usr/local/apache/logs/httpd.pid
```

```
echo -n " apache"
```

```
case "$1" in
start)
    /usr/sbin/chroot $CHROOT $HTTPD
    ;;
stop)
    kill `cat ${CHROOT}/${PIDFILE}`
    ;;
*)
    echo ""
    echo "Usage: `basename $0` {start|stop}" >&2
    exit 64
    ;;
esac

exit 0
```

The above script should be copied to the proper directory (depends on particular UNIX system), where by default startup scripts are held. In case of FreeBSD it is the /usr/local/etc/rc.d directory.

## Summary

The above method allows achieving a higher security level of the Apache server than the one, offered in the default installation.

Thanks to enabling only the absolutely necessary Apache modules, finding a new vulnerability in one of them doesn't have to indicate that our server is vulnerable. Hiding the Apache's version number, turning off the directory indexing service, chrooting and restricted configuration make a successful break-in very difficult. A chrooted environment has also one more important advantage - immunity to the large number of exploits, mainly because of lack of the shell (/bin/sh, /bin/csh etc.). Even if an intruder will success in executing system commands, escaping the chrooted environment could turn out to be quite a problem.

## Relevant Links

[Apache HTTP Server Project](#)

[Sample httpd.conf](#)

[Sample apache.sh](#)

## About the author

[Artur Maj](#) works as a Principal Software Engineer for Oracle Corporation, in the EMEA Mobile, Wireless & Voice Center of Expertise. He is experienced in designing computer systems, performing security audits as well as providing security training. He is also author of many articles and publications devoted to securing computer systems and software against intruders.

View [all articles](#) by Artur Maj on SecurityFocus.

*Comments or reprint requests can be sent to the [editor](#).*

[Privacy Statement](#)

Copyright 2006, SecurityFocus