Using OpenBSDs chrooted httpd

Marc Balmer

micro systems

ABSTRACT

OpenBSD some time ago changed the mode of operation for the Apache webserver from the normal non-chrooted operation to chrooted operation. This enhances the security of the server on which Apache is run but it imposes a few challenges to the system administrator.

In this article I will discuss selected aspects of running a chrooted HTTP daemon and present strategies on how to set up a chrooted environment for more complex applications like database access or using CGI-scripts.

(First published on April 6, 2003, revised on March 20, 2004.)

1. Introduction and Background

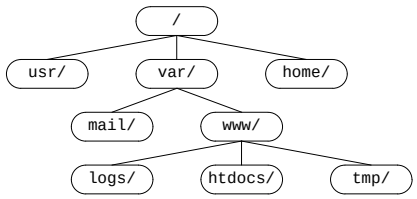
The OpenBSD HTTP daemon is based on the Apache 1.3 series webserver. Whereas the Apache webserver is normally run non-chrooted on other operating systems, it runs chrooted by default on recent OpenBSD releases.

The chrooted mode of operation confines the Apache webserver to a user defined directory, usually /var/www, by changing the filesystem root to this location. The operating system does this by a special system call, chroot(), which effectively shifts the filesystem root for the calling process to the directory given as the argument to the chroot() system call. This system call can not be reversed, i.e. once a process is chrooted it will stay there forever. The chrooted filesystem becomes the new filesystem hierarchy not only for the chrooted process but also for all of its child processes. If a webserver is chrooted this means that all programs forked by it are confined to the so called chroot jail, be it a CGI-program or a shell executed by some remote exploitable code in httpd.

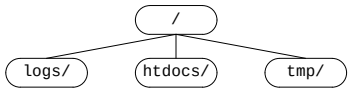
2. chroot

A UNIX filesystems always starts a the root directory, /. Every process has a pointer to its root directory within in the kernel. Whenever a new process is started it inherits this information from its parent process. The chroot() system call can change this information to any directory below the current root, making this directory the new filesystem root for the calling process and all its child processes. Calling chroot() is restricted to the super-user.

The following figure shows a simple partial UNIX filesystem hierarchy:



The next figure shows the same filesystem hierarchy chrooted to /var/www:



The chroot system call only shifts the file system root, it does not isolate the calling process from other system ressources like the process table, memory or sockets. It is a common misconception that chroot completely isolates the calling process from the system. When you are in a chrooted environment you still have access to all these other system ressources.

chroot is also available as a program which lets the super-user specify a directoy location and a command. The `chroot` utility will then chroot to the specified directory and execute the command. If the Apache webserver is chrooted to /var/www (the default location) then you can explore the chroot jail by issuing the command

```
# chroot /var/www /bin/ksh
```

You will notice that this command does not work. Why? The filesystem root has effectively changed to /var/www which means that /var/www becomes your new / and then there is no /bin/ksh (which would have to be installed in /var/www/bin/ksh for this example to work). This is the first rule of thumb when using chroot environments: They must be populated with all files and programs necessary for their operation.

3. Chrooting httpd

Chrooting httpd can lead to the following symptoms:

1) Shared libraries that are loaded at runtime are no longer found (this can affect some libraries loaded by the PHP module, e.g.).
2) CGI-programs won't run anymore.

This is all due to the fact that the webserver can no longer locate shared libraries or binaries like perl that are needed to execute CGI-programs.

4. Pre-loading Shared Libraries

To extend the functionality of the webserver it can dynamically load shared libraries, e.g. a database access library. Shared libraries for a binary program are normally loaded by the runtime linker when the program is invoked (and thus before it can call the chroot system call). Thus shared libraries like the mod_php PHP4 module, which is linked as a shared library to the httpd program when it is started, impose no problem. PHP4 will be available whether your httpd is started chrooted or not because the shared library is loaded before the chroot() system call is invoked. PHP4 itself, however, does dynamically load additional functionality at runtime and as needed. If you try to access a PostgreSQL function in PHP4 e.g. then it will fail in a chrooted httpd because only the PHP4 module is dynamically linked to httpd but not the PostgreSQL client library. The latter is loaded (mapped) to the running httpd executable by PHP4.

There are two possible solutions to get this code into the running httpd executable:

1) Install the shared library in a location so that after the chroot call it is found by the executable.
2) Pre-load the shared library when the main program is started.

The first solution leads to a new problem: The runtime linker uses a hint file when it tries to locate a shared library. This hint file must be installed in the mirrored hierarchy as well. Besides of having to install the complete runtime linker infrastructure in the mirrored hierarchy you also duplicate the binaries and you have to take care of both copies (e.g. when you upgrade to a more recent version of let's say PostgreSQL). The first solution is not the way to go.

The second solution means to instruct the runtime loader to load additional shared libraries into the program's address space although there seems to be no need for it (no need for it means that the program was not linked against the library). Loading additional shared libraries fortunately is very easy: The environment variable LD_PRELOAD can contain a list of shared libraries that are mapped in a process' address space before the executable (and its other shared libraries) is loaded. The pre-loaded code can then be used by the executable, whether chrooted or not. Most shells have a special syntax to specify environment variables for a single process only by prepending the variable definition to the command itself. The following command will start httpd with a pre-loaded PostgreSQL library:

```
# LD_PRELOAD=/usr/local/lib/libpq.so.2.2 \
     /usr/sbin/httpd -DPHP4
```

A httpd process wich is started in this way can now chroot to /var/www, use PHP4 and use the PostgreSQL client library. PHP4 will not try to dynamically load the library from the filesystem because it is already in memory.

5. Populating the chroot-Environment with Binaries

Populating the chroot-Environment with binaries is only necessary when you want (or need) to use CGI-programs. If you need to use CGI-programs then you will have to install a mirror filesystem hierarchy under /var/www which reflects what is found under / in a non-chrooted environment.

Things are simple when your CGI-programs are statically linked programs. You can directly install them somewhere under /var/www. If your programs require shared libraries, the shared libraries need to be installed as well.

Things can get more complicated if you install programs or program interpreters like Perl or Python. Not only do you have to install the binary but you also must install all shared libraries that they require to run. This means you have to find out which shared library a program needs. The `ldd` program gives you exactly this information. So you first define which binaries go into your chroot environment and secondly which shared libraries they use.

Please keep in mind that with every binary you install in the chroot environment you not only potentially lower the security of your web content (e.g. if there is no shell installed, an intruder will not be able to execute a possibly existing remote exploit that forks a shell) but you also duplicate part of the system and you have to take measures that your mirrored filesystem hierarchy stays in sync with the rest of your system. Your httpd and the rest of the system still use the same kernel, so libraries need to be in sync with the kernel. This procedure can be automated with an appropriate Makefile.

6. An Example: httpd with PHP4 and PostgreSQL

For the following example let's assume that you run a website that uses PHP4 to access a PostgreSQL database. I have already shown how httpd can be started so that PHP4 has access to the PostgreSQL client library. As the normal OpenBSD httpd starting mechanism knows nothing (yet?) about shared library preloading, we simply turn off httpd in /etc/rc.conf:

```
httpd_flags=NO
```

and we start it whith LD_PRELOAD set in /etc/rc.local

```
echo 'Starting local services'

# Clean up left-over httpd locks
rm -f /var/www/logs/ssl_mutex.*
rm -f /var/www/logs/httpd.lock.*
rm -f /var/www/logs/accept.lock.*

echo -n ' httpd'
LD_PRELOAD=/usr/local/lib/libpq.so.2.2 \
     /usr/sbin/httpd -DPHP4
```

There is another issue with PostgreSQL besides the loading of the shared library: For safety reasons you do not want it to use a TCP/IP socket but rather use a UNIX socket which is usually located in /tmp. The chrooted httpd has no access to this UNIX socket, of course. It will look for it in /var/www/tmp/. To solve this issue you must configure your PostgreSQL server to use the socket at /var/www/tmp/ with the following entry in the PostgreSQL configuration file:

```
unix_socket_directory = '/var/www/tmp'
```

The directory /var/www/tmp needs to be created first. It does not matter where your PostgreSQL server is installed but it should be outside the httpd chroot jail (I use /var/pgwww). Your PostgreSQL server will now create its UNIX socket in /var/www/tmp and the PHP4/PostgreSQL client after the chroot call will find it in its standard location /tmp.

The last thing to do is to indicate the location of the PostgreSQL UNIX socket to programs like psql. psql will by default look for the UNIX socket in /tmp and, in this case, fail. Fortunately the psql program allows to specify a different location with the -h option (which is normally used to indicate a TCP/IP hostname when using the TCP/IP method to connect to the server). psql needs to be called with -h /var/www/tmp in the chrooted environment.

7. Conclusion

OpenBSD's choice to run Apache chrooted by default is logical for an operating system that puts its main focus on security. Together with many other features to secure the operating system, the chrooted Apache provides a secure platform for serving content on the web. If you understand the chroot mechanism and the mechanisms of loading shared libraries then you will have no problems in setting-up and running a chrooted webserver.

8. Biography

Marc Balmer writes software and articles, consults and teaches UNIX related subjects. He runs his own consulting and development firm, micro systems, and is a former lecturor and member of the board of HyperWerk, a school for interactive media design at the University of Applied Sciences (FHBB), Basel, Switzerland. You can contact him via email at `<marc@msys.ch>`.

(Many thanks go to Rod. Whitworth for editing this text.)