Securing A Shared Web Server

chary_c, ⟨chary_c@epitech.net⟩
khamly_c, ⟨khamly_c@epitech.net⟩

March 20, 2009

Abstract

Typical users use shared hosting for their blog, etc ... On this kind of offer, the hoster have multiple clients on the same hardware. As it is impossible to be sure that clients applications are not vulnerable, we should at least be sure that if one is vulnerable, it won't bother the others. We will show you how a bad web server configuration could lead to disaster, and how to secure such a server.

Contents

1 Indroduction

Typical users don't need a dedicated web server to host apps like a blog, a forum, or other small web application. A solution is "Shared Hosting", ie putting multiple website on a single server. But how to secure such a server ? A vulnerability in one should not compromise the others. This article will mainly focuses on Linux with Apache HTTPd Webserver. We also assume that you have basic unix knowledge (mainly file permissions)

Shared Hosting Bases

On a shared web server, each user have an user account (sometime it's just an ftp account), and an home directory to put there files.
Each account belongs to different customers, so users must not access to other's files. This problem is easy to solve with tradiotional unix rights.

For example with users iksaif and thomas:

```
drwxr-x--- iksaif:iksaif /home/iksaif
drwxr-xr-x iksaif:iksaif /home/iksaif/public_html
drwxr-x--- thomas:thomas /home/thomas
drwxr-xr-x thomas:thomas /home/thomas/public_html
```

Here, only thomas or users in his group can access to /home/thomas.

The problem

The problem is that Apache (or another webserver) must have access to your files to serves them to the web. The webserver generaly run as an unprivileged user, let's say "apache" user. With the above configuration, it's impossible. An easy solution could be:

```
chown iksaif:apache /home/iksaif
chown thomas:apache /home/thomas
```

Or we could also add "apache" user to thomas and iksaif groups.
Theses solutions are ok for static files (html, images, etc...), but with scripts (php, ptyhon) come two problems:

- All users will be able access other's directories if the server offers scripting capabilities like php or python because theses scripts will be launched as "apache" user.
- To make a directory writable for scripts, a chmod 77x must be done on directories, and so, others can now write into your directories.

The solutions
Here is a list of solutions this article will describe:

- Apache mpm-peruser (and why worker, perchild, metuxmpm and itk won't work)
- suPHP and suExec (performance killers)

- Reverse Proxy with pound
- Virtual servers

There is not an unique solution, and this will depend of the number of user you want to put on each server. The easiest solution is using mpm-peruser. Reverse proxy and virtual servers are much more secure, but also take more time to configure.

2 The Graal

The best solution is a solution where there is a minimal amount of code running as root. We could use multiple hardware and the problem would be solved, but that's not the point, we wan't to secure a **shared** web server ! We need:

- An operating system
- A de-multiplexer to redirect the request to the right backend
- A backend to send the responses

As the backend will access user files and execute scripts, it needs to run as the user. A backend is here a webserver like apache or lighttpd.

3 Proof of concept

Ok, here is a proof of concept. Let's say you have an hosting service with ftp and http access. This hosting service provides also scripting capabilities (PHP, Python, Perl ...). If you have an ssh access it's a lot easier, but some people believe that giving access to a chroot'ed ftp is more secure. But as soon as you can execute scripts, you can do whatever you want.

Here is a PHP script to list all directories and files that you can access.

```
<?php
function list_dir($path) {
  if (!($handle = @opendir($path)))
    return false;
  echo "<dl>\n";
  while (false !== ($file = readdir($handle)))
    if ($file != "." && $file != "..") {
      $npath = realpath($path) . "/" . $file;
      if (is_dir($npath)) $file .= "/";
      $st = @stat($npath);
      printf("<dt>%o %d %d %s</dt>", $st['mode'] & 0777,
     $st['uid'], $st['gid'], htmlentities($file, ENT_QUOTES));
      if (!is_link($npath) && is_dir($npath)) {
echo "<dd>";
list_dir($npath, $recur);
echo "</dd>\n";
      }
```

```
    }
  echo "</dl>\n";
  closedir($handle);
}
list_dir("/home");
list_dir("/var/www");
?>
```

For example you may wan't to list the /home directory. And if your hosting service is not secure, you'll be able to browse (and sometime write into !) other's directory.

4 Let's skip bad solutions

4.1 PHP Safe Mode

> The PHP safe mode is an attempt to solve the shared-server security problem. It is architecturally incorrect to try to solve this problem at the PHP level, but since the alternatives at the web server and OS levels aren't very realistic, many people, especially ISP's, use safe mode for now.

[2] "Safe Mode was removed in PHP 6.0.0."
It have been removed in PHP 6, it's clearly unusable. Anyway, safe mode has never realy work, and you won't want to use that.

4.2 suPHP, suExec

These two solution are very similar:

4.2.1 suPHP

> suPHP is a tool for executing PHP scripts with the permissions of their owners. It consists of an Apache module (mod_suphp) and a setuid root binary (suphp) that is called by the Apache module to change the uid of the process executing the PHP interpreter. [6]

suPHP is used by hosters like OVH.

4.2.2 suExec

> The suEXEC feature – introduced in Apache 1.2 – provides Apache users the ability to run CGI and SSI programs under user IDs different from the user ID of the calling web-server. Normally, when a CGI or SSI program executes, it runs as the same user who is running the web server.
>
> Used properly, this feature can reduce considerably the security risks involved with allowing users to develop and run private CGI or SSI programs. However, if suEXEC is improperly configured, it can cause any number of problems and possibly create new holes in

> your computer's security. If you aren't familiar with managing setuid root programs and the security issues they present, we highly recommend that you not consider using suEXEC. [7]

4.2.3 Why are they bad ?

First, these two projects use root setuid program, it's really something we should avoid. There is also a performance problem, for each executed cgi/script, you'll need a fork. Fork may be fast on unix, it's still slow when you have one fork for each hit. Moreover, the configuration is not easy. To finish, just search "suPHP vulnerabilities" on any search engine, and you won't want to use that.

But the fastcgi + suexec couple is something that can work in a decent way with some configuration. For example Alwaysdata [11] use that for their services (Python, RoR, PHP, ..).

4.2.4 Critical Parts

- operating system
- web server
- root setuid binaries

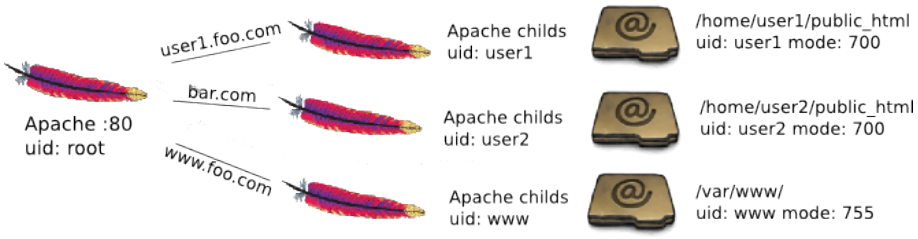
5 Apache mpm

> Apache 2.0 extends this modular design to the most basic functions of a web server. The server ships with a selection of Multi-Processing Modules (MPMs) which are responsible for binding to network ports on the machine, accepting requests, and dispatching children to handle the requests. [8]

And one of this modules is mpm-peruser [9]. There have been other mpm to try to solve the shared-hosting problem. But peruser is the only which work with mod_php, because peruser use process pool and not thread. In fact, mod_php is not thread-safe because it's linked to some libs, like GD, wich are not. So metuxmpm or mpm-perchild were not usable with mod_php because they use threads.

You can found some more information to configure mpm-peruser in "Using mpm-peruser To Secure A Shared Server" [10] article from Stuart Herbet.

Here is a schema to explain how it works with mpm-peruser



The typical process tree look like that

```
`- root - Main process listenning on port 80
|   `- nobody - Multiplexer, redirect to other childs
|   `- user1  - child for user1.foo.com
|   `- user1  - child for user1.foo.com
|   `- user1  - child for user1.foo.com
|   `- user2  - child for bar.com
|   `- user2  - child for bar.com
|   `- www    - child for www.foo.com
```

5.1 Critical Parts

- Operating System
- Apache
- chroot() must be secure (you may use grsecurity/selinux etc ..)

5.2 Pros and Con

+ Easy to setup

- Apache only

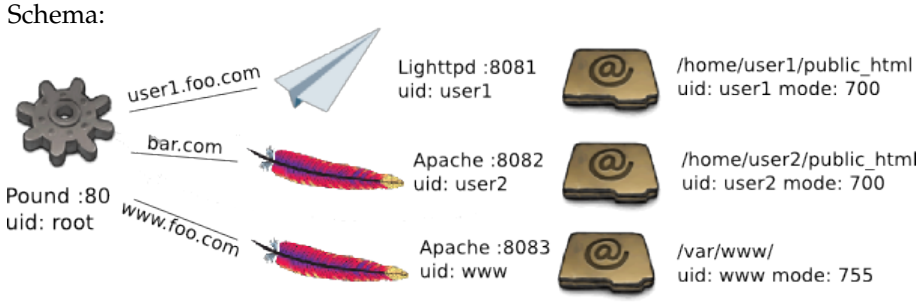
6 Reverse Proxy

A reverse proxy dispatches in-bound network traffic to a set of servers, presenting a single interface to the caller. For example there is pound [3]

> The Pound program is a reverse proxy, load balancer and HTTPS front-end for Web server(s). Pound was developed to enable distributing the load among several Web-servers and to allow for a convenient SSL wrapper for those Web servers that do not offer it natively. Pound is distributed under the GPL - no warranty, it's free to use, copy and give away.

Lighttpd [4] can also be used as a reverse proxy, but I think pound is a better solution, because it's have been designed from the begining to do that job.

Schema:



6.1 Critical Parts

- Operating System
- Reverse Proxy

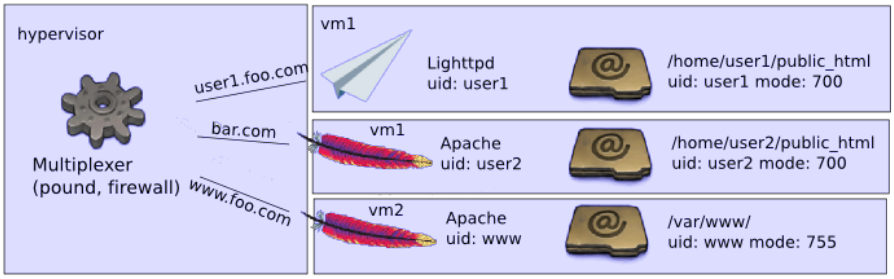
6.2 Pros and Con

+ There is less code running than using apache mpm
+ The reverse proxy is supposed to be secure and minimal
- Need some time to setup

7 Virtualization

A third solution is to install a virtualized server for each clients. As they don't won't to have a dedicated server to administrate, it's possible to give them only a user account on the virtualized server.

Here is a schema to explain how it works:



Here you have a lot of option to redirect HTTP request to the right guest. You can use a reverse proxy, a firewall, assign unique ip to the guests, etc...

Here we only get traditional virtualization vulnerabilities depending on the product you'll use.

7.1 Critical Parts

- The host operating system
- Virtualization software
- Reverse Proxy (if used)

7.2 Pros and Con

+ Easy to setup if you use libvirt
+ Scalable, you can use different virtualization products and setup with a read-only common system, etc...
- A little overkill

8 Conclusion

For a small installation (= 10 Website), mpm-peruser seems to be the right solution, because it's easy to setup, and it just works. For more websites, the reverse proxy will probably give you better performance and virtual servers better security. For that, you'll have to chosse depending of your needs.

9 References

References

[1] http://php.net/ *PHP Homepage*

[2] http://php.net/features.safe-mode *PHP Safe Mode*

[3] http://www.apsis.ch/pound/ *pound reverse proxy*

[4] http://www.lighttpd.net/ *lighttpd fly light*

[5] http://httpd.apache.org/ *Apache HTTP Server Project*

[6] http://www.suphp.org/Home.html *suPHP Home page*

[7] http://httpd.apache.org/docs/1.3/suexec.html *What is suEXEC?*

[8] http://httpd.apache.org/docs/2.0/mpm.html *Describes what a Multi-Processing Module is and how they are used by the Apache HTTP Server.*

[9] http://www.telana.com/peruser.php *Peruser MPM for Apache 2*

[10] http://blog.stuartherbert.com/php/2008/03/20/using-mpm-peruser-to-secure-a-shared-server/ *Using mpm-peruser To Secure A Shared Server*

[11] http://www.alwaysdata.net *Alwaysdata*