Apache + Chroot + FastCGI + PHP FAQ

Contents

I : Introduction

A : Summary and Scope

This faq intends to cover the general setup of Apache with mod_fastcgi with chroot user environments. Both Apache and mod_fastcgi are extremely well documented, as is using them together. This is mainly intended to show how chroot + Apache + FastCGI can be used in concert, not to show how each can be used individually. It is also not intended to be the authority on methods of doing this. It merely attempts to show how it can be done, and illustrate the various necessities of a functional configuration.

In this faq the FastCGI application we will focus on is PHP, partly due to popularity, partly due to that's what I am using it for now. Also, this is because other CGI's should be applicable in a similar manner.

B : What is a chroot?

From Wikipedia :

```
"A chroot on Unix operating systems is an operation which changes the root
 directory. It affects only the current process and its children. "chroot" itself
can refer to the chroot(2) system call or the chroot(8) wrapper program.

A program that is re-rooted to another directory cannot name files outside that
 directory. This provides a  convenient way to sandbox an untrusted, untested or
 otherwise dangerous program. It is also a simple kind of jail mechanism."

-- http://en.wikipedia.org/wiki/Chroot
```

C : Why use chroot?

Mainly security. In a couple different ways, first it segments your users from one another. If user X "chmod 777"'s all his files, he is still protected from user Y. User Y's scripts are chroot()'d when they run, his ssh or telnet is chroot()'d when he logs in. He can only see and interact with his own mini-environment.

This segmentation is not just a positive for user to user problems. It can prevent a single breach from becoming larger. For instance, our troublesome user Y writes a script with a security flaw that allows upload and execution of files. Or perhaps he just installs a faulty script. A malicious client visits user Y's site and uploads files and executes them, he is running as user Y, and he may be able to damage user Y's files. However, user X, from the previous example, is still safe, so are our system files that might have allowed privilege escalation.

Chroot is **not** the absolute in security, but I have seen one site get defaced and exploits uploaded that were never able to gain privileges because of the chroot environment.

D : Why not CGI?

The way I initially was using Apache with chroot environments was with CGI. Using the binfmt_misc module, we could set all .php scripts to execute with /usr/bin/php. Thus, inside each chroot existed a /usr/bin/php. For instance, /www/host.com/usr/bin/php or however you have defined your chroot VirtualHosts.

Apache would call suexec, which would chroot to the user passed to it from Apache's VirtualHost directive; it would then execute the PHP script, which would in turn call the interpreter /usr/bin/php. Hashbang (#! /usr/bin/php) at the top of the script achieves the same effect, binfmt_misc just saves your users a step).

There are several drawbacks to this scenario. The main being that, every time a script is called, PHP must load itself up into memory, process the script, output the result, and tear down. This is a lot of overhead for one script. Try Apache benchmark with 10 concurrent requests, and watch the CPU load as each PHP process executes. It is quite a load on the system, and it is also very slow compared to mod_php or FastCGI.

This leads us to several of the "PHP accelerators" out there, such as APC and eAccelerator. These will not work in CGI mode, because they try to cache compiled scripts in memory, however, as each CGI starts and stops for every script, there is no way to retain storage in the system's memory. This is another major drawback because these accelerators can offer quite an improvement in speed, through less disk access and less cpu cycles needed for each script.

Another speed improvement can be gained with persistent MySQL connections. However, as with the accelerators, nothing can be retained in memory, so persistent connections are not possible when using PHP as CGI.

All these are major drawbacks to running PHP, Perl, Python or anything as a CGI. In almost every category they are slower, require more CPU and memory, and make serving a large number of requests nearly untenable. The next section will cover how FastCGI eliminates all these problems, greatly improves performance, yet still maintains the advantages of the chroot environment.

E : Why FastCGI + Chroot?

First of all, mod_fastcgi spawns an application called "fcgi-pm" (FastCGI Process Manager), it in turn spawns the persistent FastCGI enabled application. Fcgi-pm pipes information back and forth between Apache and the FastCGI enabled application.

This way, only one running process is needed to execute PHP scripts, where previously we needed one CGI application running for each script that was processing. This saves on CPU, speed, and memory, since the binary is preloaded, and ready to process scripts.

Also, as discussed above, since it is persistent in memory, eAccelerator or APC will work with it, caching compiled scripts in memory. Thus, Eliminating compile time and disk access. Persistent MySQL connections will also function correctly with this setup.

These are all major improvements in CPU usage and speed of application of scripts. Speed improvement is dependent on the script, but the faster it executes and gets out of the way, the faster the next request can be processed.

As to the chroot, before we had the advantages of the security of segmenting users from users, as well as users and clients from other users and their clients. With FastCGI the process is very similar to chroot'ing with Apache. FastCGI provides for a wrapper that will execute the interpreter, in this case PHP.

Depending on your wrapper, it will run the FastCGI PHP process as the user specified in the VirtualHost directive as well as chroot'ing the FastCGI enabled process to their web space. There are chroot suexec wrappers out there that you can use if you do not already have one.

II : Apache + Chroot + FastCGI Configuration

A : Apache with chroot'd webspaces

As mentioned above, Apache allows for a wrapper when it runs CGI applications. You can specify the UID and GID of the user for the CGI to run as in the VirtualHost directive. Finding the proper location to chroot to is the wrapper's job. The suexec wrapper always runs as root, no matter who calls it. When called it chroot's itself to the calling user's directory, then drops privileges to that user, closes the file descriptor for the log file, and then runs the CGI in the chroot environment. Again, the user is passed from the VirtualHost Directive, which looks like this :

```
<VirtualHost x.x.x.x:80>
       ServerName www.host.com
       ServerAdmin user@host.com
       SuexecUserGroup "user" "group"
       ...
</VirtualHost>
```

I use chrootssh (**http://chrootssh.sf.net**) which allows for ssh sessions to be chroot'ed as well. It looks for "/./" in the user's home dir path specified in /etc/passwd.

For instance, in /etc/passwd it looks like this :

```
domain_admin:x:UID:GID:USER NAME:/www/h/o/host.com/./:/bin/bash
```

The Suexec wrapper then gets the home directory and uses "/./" to find where it should chroot() to. (Basically, it finds "/./" in the user's path, terminates the string there, and uses it as the chroot path. This should be a simple modification to your Apache and FastCGI wrappers if you know basic C.)

This way you can have :

```
domain_user:x:UID:GID:USER NAME:/www/h/o/host.com/./users/domain_user:/bin/bash
```

And domain_user exists within the same chroot, so you can have multiple users for each VirtualHost, with their own home directories, yet separated from other domain's users. This way a website can have several real users, giving the domain administrator the ability to add other users. They can share the same GID as well, and quotas can be applied to the user, and over the whole domain.

But this all just depends on how you set up your system.

I don't want to go too in-depth into suexec and Apache, because they have very detailed documentation already :

Suexec information for Apache 1.3 : **http://httpd.apache.org/docs/1.3/suexec.html**
Suexec information for Apache 2.0 : **http://httpd.apache.org/docs/2.0/suexec.html**

B : FastCgiWrapper

Just like Apache, FastCGI allows for a wrapper for execution of the FastCGI enabled applications. It will probably be very similar to Apache's wrapper, but there are two main differences between Apache's wrapper and FastCGI's wrapper.

First, the FastCGI application, in this case PHP, will most likely not be owned by the user. It will be owned by root, but does not have to be. By default, Apache's suexec wrapper checks for the proper ownership on a file before it is executed. This check will have to be disabled to allow a non-user owned FastCGI version of PHP to run. Also, since it will be owned by someone else, that means it will be writable by someone else, most likely root. So the checks for ownership, group writable and user writable files will have to be disabled. This will allow suexec to execute the wrapper even though it is owned by root and writable by root.

Secondly, the wrapper is only executed when an fcgi-pm (FastCGI process manager) needs to start a process, not for every CGI script that runs. This won't make a large amount of difference to you, but it is good to keep this in mind when troubleshooting your wrapper. Once it has started a FastCGI process to handle requests for scripts, it won't run again until you need a new FastCGI process, or until Apache is restarted, which necessitates the creation of a new process, since they are all terminated when Apache is terminated.

(For install information, **http://www.fastcgi.com**)

Note : Some people have requested I provide my wrapper. The main problem with this, is that I have a pretty unique wrapper and I had modified it heavily to allow specific things about my setup to work. The changes that need to be made have been enumerated above.

But for those who are interested, Gasior has made **his wrapper available here.** It is a modification of sbox (**http://stein.cshl.org/software/sbox/**).

Hopefully this and the information this FAQ provides will allow you to adjust an existing chroot wrapper to your FastCGI configuration. I strongly suggest that you get your setup working as CGI (Apache + Suexec + Chroot) first. **Then**, install mod_fastcgi and work on getting a working wrapper and functional setup. It's much easier to do it in this order, and much less overwhelming. Also, refer to section **IV-B for FastCGI wrapper troubleshooting tips**.

C : Apache + FastCGI

The specific VirtualHost directive for Apache and Fastcgi is covered below. In general, all you need in httpd.conf is to specify the wrapper :

```
<IfModule mod_fastcgi.c>
    FastCgiWrapper  /usr/local/apache2/bin/suexec-fcgi
</IfModule>
```

As mentioned above, this instructs fcgi-pm to call this wrapper when it executes the FastCGI-compiled PHP binary to process scripts.

More detailed information about FastCGI configuration parameters can be found here :
http://www.fastcgi.com/mod_fastcgi/docs/mod_fastcgi.html

D : How does it all work?

Before we get deeper into the specifics of the configuration, let us look at how all these pieces will work together. The best way I can describe this process is to show a simple version of the flow between applications. It's far more detailed, and really beyond me, to give a technical examination of how requests flow back and forth over multiplexed pipes. However, if you require such information, it is available at FastCGI's site (**http://www.fastcgi.com**).

These illustrations assume we are running Apache + PHP + mod_fastcgi and a FastCGI wrapper which chroots users to their own chroot environments.

First, an incoming request for a PHP script with no running FastCGI process to process it (i.e. fcgi-pm will have to start it) :

```
[web client] -> [Apache] -> [mod_fastcgi] -> [fcgi-pm] -> [FastCgiWrapper] -> [PHP wrapper] -> [PHP'
```

Note : The [PHP wrapper] and [PHP's process manager] steps are optional, and are discussed more in **section III-A**. This wrapper sets environment variables that instruct PHP to spawn and manage its own worker applications, instead of letting fcgi-pm manage the workers directly.

Letting FastCGI manage applications directly would look more like this :

```
[web client] -> [Apache] -> [mod_fastcgi] -> [fcgi-pm] -> [FastCgiWrapper] -> [PHP worker]
```

Both methods of configuration basically send data back to the client the same way, back to mod_fastcgi and back to Apache out to the client.

If the [fastcgi-compiled PHP] or [PHP wrapper] process is already running, fcgi-pm determines which running process is the correct one to handle this request, and forwards it there, it then reads back the response and sends it through Apache, back to the client.

III : Adding PHP to the mix

A : The PHP Wrapper and What it Does

The php-wrapper is a bash script that sits inside the VirtualHost's chroot directory. It allows us to set environment variables before FastCGI enabled PHP is run. There can be two if you plan to use PHP4 and PHP5. Php-wrapper and php5-wrapper. The "exec /usr/bin/php-fcgi" line is the only one that needs to change between the two, for instance :

```
====== php-wrapper ======
#!/bin/sh
PHP_FCGI_CHILDREN=3
export PHP_FCGI_CHILDREN
PHP_FCGI_MAX_REQUESTS=2000
export PHP_FCGI_MAX_REQUESTS
exec /usr/bin/php-fcgi
====== END php-wrapper ======
```

And for PHP5 :

```
====== php5-wrapper ======
#!/bin/sh
PHP_FCGI_CHILDREN=3
export PHP_FCGI_CHILDREN
PHP_FCGI_MAX_REQUESTS=2000
export PHP_FCGI_MAX_REQUESTS
exec /usr/bin/php5-fcgi
====== END php5-wrapper ======
```

This way you can specify how many "workers" you need for each VirtualHost and how many requests each should handle. If you have a wrapper that looks like this (or no wrapper at all and you call php-fcgi directly from the VirtualHost directive) :

```
====== php-wrapper ======
#!/bin/sh
PHP_FCGI_MAX_REQUESTS=2000
export PHP_FCGI_MAX_REQUESTS
exec /usr/bin/php-fcgi
====== END php-wrapper ======
```

Then, fcgi-pm will only spawn a PHP process to be a "worker." There will not be a PHP process manager that handles spawning multiple workers. This does work, but it is not ideal, because PHP manages it's workers better than fcgi-pm does, at least in my limited experience. The workers are set to "die" after so many requests, this requires fcgi-pm to restart them, but fcgi-pm is not expecting them to die too quickly, or at all until it signals termination. It's much more stable to let PHP be the process manager of its own children.

Note : In older versions of PHP (before 4.3.0) PHP is **always** the process manager, and does not start directly as a worker from fcgi-pm. In PHP 4.3.0 and above, if there is no PHP_FCGI_CHILDREN environment variable, it defaults to 0. Therefore, PHP always starts as a "worker," letting whatever process that called it handle the process management. Which in this case is FastCGI's process manager, fcgi-pm. This is not a very stable way to manage PHP processes, and it is much wiser to set the PHP_FCGI_CHILDREN variable!

The choice is really yours. On my low traffic VirtualHosts, I just let fcgi-pm handle php-fcgi processes. On some of the higher traffic ones, I let PHP manage its own processes and assign several "workers." It appears that each worker takes requests in a round-robin fashion (so far as I could tell strace'ing them.) Thus, if you have three workers, every third request will hit the same worker.

It all depends on your system resources and what you want to do. But, if you use the wrapper, you can have it both ways. Each VirtualHost can be individually configured. The wrapper allows for more flexibility, you only gain configuration ability and don't lose anything.

B : Configuring the VirtualHost

Really, this is part of setting up Apache, but most likely your VirtualHost is working the way it is now. So, really all the additions to it will be FastCGI related, and in this case, PHP related.

I found this to be one of the most confusing parts of setting up FastCGI, there appears to be numerous ways to do this.

But the way I have settled on at the moment is this. In the chroot, the FastCGI-compiled PHP4 process is located at /www/h/o/host.com/usr/bin/php-fcgi. PHP5 is here /www/h/o/host.com /usr/bin/php5-fcgi. The wrapper for PHP4 and PHP5 is, respectively, php-wrapper and php5-wrapper.

Of course, you will need the SuexecUserGroup line mentioned above, but since I was using that already, I just left it there. Basically, I took my existing VirtualHost config, and added this section to it :

```
######### FASTCGI ########
<IfModule mod_fastcgi.c>
    ScriptAlias /cgi/ /www/h/o/host.com/usr/bin/

    AddHandler php-fastcgi .php
    AddHandler php5-fastcgi .php5

    <Location /cgi/php-wrapper>
        SetHandler fastcgi-script
    </Location>
    <Location /cgi/php5-wrapper>
        SetHandler fastcgi-script
    </Location>

    Action php-fastcgi /cgi/php-wrapper
    Action php5-fastcgi /cgi/php5-wrapper

    AddType application/x-httpd-php .php
    AddType application/x-httpd-php .php5

</IfModule>
######### END FASTCGI ########
```

Note : It is possible to change php-wrapper to php-fcgi, which is what I call the PHP FastCGI-enabled binary, and run it directly. However, in the newer versions of PHP, it will only be a "worker" and fcgi-pm will be it's process manager. Refer to **section III-A** above for more information, it's recommended to use a wrapper for php-fcgi.

This way, if I ever decide to comment out LoadModule for FastCGI, the VirtualHosts just default back to CGI mode, because of the <IfModule mod_fastcgi.c> </IfModole> check.

But it sets "fastcgi-script" as the handler for the PHP binaries, this means when *.php is called, the handler is set to php-fastcgi, it's action is to load /cgi/php-fcgi which is an alias of /www/h /o/host.com/usr/bin/php-fcgi. (You need the full path, because at this point we have not chroot'ed yet.) This causes fcgi-pm to execute the php-wrapper, which sets environment variables and executes php-fcgi. It opens the pipes and uses php-fcgi to process .php scripts.

The FastCGI FAQ also covers this in greater detail : **http://www.fastcgi.com/docs/faq.html**

C : Enabling FastCGI in PHP

This is just about the easiest part. Basically, with all your current configuration options, just add "--enable-fastcgi". This builds the sapi CGI PHP binary with FastCGI capabilities. It should be located in <php_source_dir>/sapi/cgi/php after build. Type "./php -v" in the ./sapi/cgi directory to verify that it is built with FastCGI, you should see :

```
PHP 4.4.2 (cgi-fcgi) (built: Aug 24 2006 11:43:01)
```

The text "cgi-fcgi" is what we are looking for. Now when it's called by fcgi-pm, it will be ready to process FastCGI requests!

Note : Getting a little off topic, but it's useful to "strip" binaries that are going to be placed into many chroots. It can reduce their size significantly. Please 'man strip' for further information. Generally just 'strip <binary>' is all you need. Check size before and after and you will see how much of a difference it can make when multiplied over many chroot environments.

I renamed mine php-fcgi, so that I could continue to keep the standard "php" binary. This way, if I turn off FastCGI, requests are still served using "/usr/bin/php" because of the binfmt_misc kernel module. However, this may differ in configuration, but no matter how you do it, it would probably be best to leave a copy of the regular CGI PHP. This way you can have some VirtualHosts on FastCGI and some on regular CGI. Or you can move each site over to FastCGI one by one by simply changing the VirtualHost directive as specified above.

Of course, if you are new to chroot environment's this will be considerably more complex, this URL gives an overview as well as links to information on how to build a chroot : **http://en.wikipedia.org/wiki/Chroot**

D : Configuring php.ini

There are two things that **MUST** be set in php.ini in order for this configuration to work. They are vital to chroot working correctly with FastCGI :

First is "doc_root." In our current setup, our chroot is at /www/h/o/host.com/ our web space is at /www/h/o/host.com**/html**.

So, we must set :

```
doc_root = "/html"
```

We do this because FastCGI does not know our wrapper is chroot'ing, it sends all the full paths through the pipe to the php-fcgi process. Setting doc_root to the path inside the chroot allows PHP to find the scripts in the correct location : /html/file.php

Otherwise, php-fcgi will try to open /www/h/o/host.com/html/file.php, and since it is running in a chroot environment, that path does not exist. The "/www/h/o/host.com" does basically not exist any longer for this process.

The second is :

```
cgi.fix_pathinfo = 0
```

This keeps the correct paths we need for the scripts to be found under the chroot environment.

E : eAccelerator

As mentioned earlier in this FAQ, one of the advantages of FastCGI + chroot is that it allows you to use eAccelerator (**http://www.eaccelerator.net/**), which will provide a nice speed improvement to our scripts.

It stores compiled PHP scripts in memory, so it saves on CPU cycles due to repeated compiling of PHP scripts at every page load. It's a little off topic for this FAQ, but it's so simple to set up, that I will include it, since chroot + CGI PHP usually prevents it from working.

From here on, we will assume you have gone to **http://www.eaccelerator.net** and downloaded and compiled eAccelerator. They have detailed instructions and more options than I demonstrate here, so please read their documentation.

It's very simple to setup, first in /www/h/o/host.com/lib/ or wherever you have decided to store the libraries for the chroot :

```
[user@host /www/h/o/host.com/lib]$ ls -al eaccelerator*
-rwxr-xr-x    1 root     root       136232 Jul 28 11:46 eaccelerator5.so
-rwxr-xr-x    1 root     root       119912 Jul 28 11:46 eaccelerator.so
```

I have two separate php.ini's. One for PHP5 and one for PHP4, hence two different libraries for each to be included, in this case, we'll just look at the php.ini for PHP4 :

```
===== snippet from php.ini =====
[eAccelerator]
extension="eaccelerator.so"
eaccelerator.enable = "1"
;eaccelerator.log_file="/tmp/eaccelerator.log"
eaccelerator.shm_size = "5"
eaccelerator.debug = "0"
===== end php.ini snippet =====
```

In the php.ini for PHP5 of course you would just set extension="eaccelerator5.so". Now load <? phpinfo();?> in a test PHP page and look for the "eAccelerator" section. You should see that it is now caching scripts in memory!

F : Configuring Other Interpreters

I have only attempted PHP thus far. However, the system will stay the same. Basically, whatever interpreter or application you have just needs to be FastCGI compatible, and it will work in this same setup. It will need VirtualHost entries and have to exist inside the chroot, but the principles are the same.

IV : Troubleshooting

A : Troubleshooting Apache

As always check Apache's error_log, this is where mod_fastcgi's messages will be. Where it is located depends on how you set it up. If you are thinking of installing Apache + chroot FastCGI, we should hope you already know where to find your log file. =)

Common issues with Apache could be caused by the VirtualHost configuration. However, whatever the problem with the configuration, the relevant error should be logged to error_log.

If you are seeing any kind of segmentation fault, or any error that appears to be coming from Apache, this guide will definitely help : **http://httpd.apache.org/dev/debugging.html**

Sometimes when installing mod_fastcgi, you may discover a totally unrelated problem that went unnoticed. Like a bad module that causes Apache children to segfault. Try to track down where the error is coming from before assuming FastCGI or PHP is causing it. It may just be that you discover an intermittent issue while spending a lot of time with your log files during the FastCGI install!

B : Troubleshooting suexec (or your own FastCGI wrapper)

First, be sure you have read the **FastCgiWrapper section** as it details the major differences between Apache's default suexec wrapper and the wrapper that you will use to initiate FastCGI enabled applications, like PHP. It is very similar to Apache's wrapper, but there are a handful of changes you must make in order for it to work.

This can be difficult because you can't simply run it from the console and test. I attempted to just use the same wrapper for FastCGI that I used with CGI, this resulted in problems. It can be difficult to trace.

You can strace the fcgi-pm process with a command like :

```
strace -s 2000 -ff -o /tmp/fastcgi -p <PID of fcgi-pm>.
```

This will follow all the child processes of fcgi-pm and write their strace to /tmp/fastcgi.<PID of process>.

Sometimes that can be useful, but what helped me the most was compiling a separate binary for FastCGI's wrapper, I called it suexec-fcgi. Then, I changed its log file to log to a separate file from your normal suexec binary, suexec-fcgi.log for instance.

Then, I used the mechanism it already had for logging (log_err()) and put many log statements in it, before and after all the functions, etc. It takes a few minutes, but it's worth it when you look in suexec-fcgi's log file and see exactly where it is dying.

C : Troubleshooting FastCGI

As above, you can strace the fcgi-pm process, and that should tell you what it is trying to execute. But overall, I didn't have any problems with mod_fastcgi itself, only with the wrapper and PHP.

D : Troubleshooting FastCGI enabled PHP

Again, strace works wonders, if your wrapper is working, but your script is not executing, try strace'ing it :

```
strace -s 2000 -p <PID of FastCGI-compiled PHP worker>.
```

You can see what data it is receiving and what script it is trying to execute, what environment variables are being sent to it. This can be very helpful in some cases.

E : Troubleshooting the PHP Wrapper

Is PHP running as a single worker? Or is it running as a process manager spawning multiple workers?

Here is the snipped output of "ps afxu". This should be what you see on your own system (I will explain below what we are seeing) :

```
# ps axfu

--snip--
root     21955  0.0  0.2  7884 2840 ?        S    Aug29   0:00 /usr/local/apache/bin/httpd
nobody     764  0.0  0.2  7880 2768 ?        S    12:15   0:00  \_ /usr/local/apache/bin/fcgi-
1007       775  0.0  0.2 12592 2320 ?        SN   12:15   0:00  |   \_ /usr/bin/php-fastcgi
1007       776  0.0  0.5 13480 5564 ?        SN   12:15   0:06  |   |   \_ /usr/bin/php-fastcgi
1007       777  0.0  0.8 15764 8644 ?        SN   12:15   0:07  |   |   \_ /usr/bin/php-fastcgi
1007       778  0.0  0.5 13452 5668 ?        SN   12:15   0:06  |   |   \_ /usr/bin/php-fastcgi
617       2109  0.0  0.9 15104 9480 ?        SN   12:46   0:01  |   \_ /usr/bin/php-fastcgi
608       2902  0.0  0.7 16348 7712 ?        SN   13:11   0:03  |   \_ /usr/bin/php-fastcgi
--snip--
```

Starting from the top, we can see Apache spawning fcgi-pm on PID 764.

For user 1007 (which corresponds to a VirtualHost), we see fcgi-pm spawning a PHP process on PID 775. This is the PHP process manager. The php-wrapper has set the environment variable PHP_FCGI_CHILDREN and thereby told it to spawn 3 workers, and we see them below it, in the tree, on PIDs 776,777 and 778.

The other users, UIDs 617 and 608, have not set PHP_FCGI_CHILDREN, therefore, when fcgi-pm executes PHP, it takes on the role of a single worker process, managed by fcgi-pm (FastCGI Process Manager).

F : Troubleshooting FastCGI PHP Scripts

I had relatively few problems with the scripts themselves. One issue I had involved how variables are passed to scripts. With regular CGI I was able to call URLs like http://www.host.com/index.php/page4. This seems to give me "No input specified." But that may be a problem specific to my server. Also, because FastCGI does not know we are chroot'ed and sends full paths to the PHP worker processes, sometimes it can throw off the environment variables. This may just be something you need to fix within your scripts. In this case, it would be much better to pass "page4" as a normal GET variable.

A similar issue I had was that on some sites I could call http://www.host.com/01, where "01" became a variable parsed out of the REQUEST_URI and used inside index.php to load a file like "/html/content/01". However, since "/html/01" does not exist, the server gives a 404 error. The page still displays fine however, and you wouldn't know that there was an issue unless you viewed the headers.

The fix for this was sort of a workaround, there may be a better way, but if you create a mod_rewrite rule for the non-existent file ("/01") as follows :

```
RewriteCond %{REQUEST_URI} ^/01$
RewriteRule ^(.*)$ /%1 [QSA,L]
```

Then Apache sees the file as existing, and returns a 200 status code. Even though the file does not exist and is just a variable used inside index.php to signify a file name to be included into the template.

The above example is really just poor design when working with PHP, but it may have applications where the design is not so poor, so I thought I would mention it.

But It couldn't hurt to make sure the headers and status codes are correct on your site. The browser can get a 404 error and still display the page properly and you wouldn't be able to tell in some cases. Firefox has a nice plugin called livehttpheaders, which you can get here : **http://livehttpheaders.mozdev.org/**. This is a great tool for making sure everything is working correctly. You can easily see everything the server is telling the browser and vice versa.

Another big thing to watch out for is mod_rewrite rules. If you have a lot of them, make sure they are correct, they really need to be. I had some incorrect mod_rewrite rules that appeared to work under CGI, but did not under FastCGI. Mostly because of the way scripts were called, such as the example above, calling /index.php/variable. This can all be fixed though, and definitely was not any flaw in anything except my mod_rewrite rules. =)

The best news is that some of the most popular PHP scripts (phpBB, Drupal, Invision Power Board) really had no issues at all. The vast majority of scripts will work perfectly, without any modifications.

V : Comments, Caveats, Questions

Feel free to E-mail questions or additions, and we will be glad to include them in the FAQ or in this section. I can be reached at : fcgi-faq[at]seaoffire[dot]net. For most inquiries, you should go to **www.fastcgi.com** and join their mailing list.

VI : Credits

The authors and developers of FastCGI - Thanks! It really is an ingenious idea.

Maciej Gasiorowski (Gasior) - Helping me set this up and get it working on my system. I probably asked him 10,000 questions, and he graciously answered them all.

David Birnbaum - For providing tons of information and answering many questions.

Chris Lightfoot - For enlightening me to the function of the PHP_FCGI_CHILDREN environment variable.