APPENDIX B

Troubleshooting

The Apache Web server is a very complex beast. In the vanilla package, it includes over 30 functional modules and more than 12 dozen configuration directives. This means that there are significant opportunities for interactions that produce unexpected or undesirable results. This Appendix covers some of the more common issues that cause problems, as culled from various support forums.

Troubleshooting Methodology

In the Error Log

The Apache software does quite a reasonable job of reporting the details when it encounters problems. The reports are recorded in the server's error log, which is usually stored in one of the following places:

- */usr/local/apache/logs/error_log*
- */var/log/apache/error_log*
- */var/log/httpd-error.log*
- */var/log/httpd/error_log*
- *C:\Program Files\Apache Group\error.log*

Where the error log is put depends on how you installed and configured the server; the wealth of possible locations in the list above is because popular prepackaged installation kits (from Red Hat, SuSE, etc.) each has its own preferred location. Of course, the definitive location can be determined by examining your *httpd.conf* file for the *Error-Log* directive(s).

So the very first thing you should do when Apache appears to be misbehaving is see if the server has any comments to make.

If the messages in the error log don't make the cause of the problem immediately clear, or if there aren't any messages that seem to relate to the problem, it's a good idea to crank the logging level up by changing the *LogLevel* setting in the *httpd.conf* file:

```
LogLevel debug
```

The `debug` setting enables all possible error messages and makes the server extremely verbose, so it's a good idea to set it back to `warning` or `error` after it has helped you locate the cause of your problem.

Characterize the Problem

When you're trying to diagnose a problem, here is a question you should ask yourself: "What is the current behavior, and in what ways is it different from the expected or desired behavior?"

If you ask this question, a natural successive question is: "What could cause the current behavior?"

Between the answers to these two questions often lies a "Eureka!" moment. At the very least, they narrow your area of research.

Debugging the Configuration

When diagnosing a problem by examining your server's configuration, be sure to examine all of the files involved. In particular, look for files identified in Include directives, as well as those in the main *httpd.conf* file and in *.htaccess* files.

If you're editing the server-wide configuration files, be sure to restart the server afterward to make the changes take effect!

If editing a configuration or *.htaccess* file seems to have no effect, test that it's actually being processed by putting a line of gibberish into the file and trying again.

If it seems that an *.htaccess* file is being ignored, even when you insert gibberish, it indicates that it's within the scope of an *AllowOverride None* directive.

Debugging Premature End of Script Headers

When you're working with CGI scripts, certain messages can quickly become extremely familiar and tiresome; typically the output in the browser window will be either a blank page or an Internal Server Error page.

This message has several different possible causes. These include, but are not necessarily limited to:

- The CGI script is either not emitting any output at all, or it is emitting content before the required header lines, or it's neglecting to emit the obligatory blank line between the header and the content.
- The script encountered an error and emitted the error message instead of its expected output.

- You're using *suexec* and one or more of the *suexec* constraints has been violated.

To test to see if the problem is an error condition or improper CGI response formatting, run the script interactively from the command line to verify that it is emitting content in compliance with the CGI rules.

If you're using *suexec*, check the *suexec* logfile to see if there are security constraints being violated.

You can tell if you're using *suexec* with the following command:

```
% httpd -l
Compiled-in modules:
  http_core.c
  mod_so.c
suexec: disabled; invalid wrapper /var/www/apache/bin/suexec
```

If you get a message that says that *suexec* is disabled, you can ignore that as a possible cause of the script's execution problems.

If *suexec* is enabled, though, you should look at its logfile to get more details about the problem. You can find the logfile with:

```
# suexec -V
 -D DOC_ROOT="/usr/local/apache/htdocs"
 -D GID_MIN=100
 -D HTTPD_USER="www"
 -D LOG_EXEC="/usr/local/apache/logs/suexec.log"
 -D SAFE_PATH="/usr/local/bin:/usr/bin:/bin"
 -D UID_MIN=100
 -D USERDIR_SUFFIX="public_html"
```

The important line is `-D LOG_EXEC="/usr/local/apache/logs/suexec.log`"; it tells you *exactly* where *suexec* is recording its errors.

You can find out more about CGI and *suexec* here:

- The CGI specification, *http://CGI-Spec.Golux.Com/*
- Recipe 8.13
- The *suexec* manpage

Common Problems on Windows

Windows has its own distinct set of problem areas that don't apply to Unixish environments.

Cannot Determine Hostname

When trying to start Apache from a DOS window, you receive a message like "Cannot determine hostname. Use ServerName directive to set it manually."

If you don't explicitly supply Apache with a name for your system, it tries to figure it out. This message is the result of that process failing.

The cure for this is really quite simple: edit your *conf\httpd.conf* file, look for the string `ServerName`, and make sure there's an uncommented directive such as:

```
ServerName localhost
```

or:

```
ServerName www.foo.com
```

in the file. Correct it if there is one there with wrong information, or add one if you don't already have one.

Also, make sure that your Windows system has DNS enabled. See the TCP/IP setup component of the Networking or Internet Options control panel.

After verifying that DNS is enabled and that you have a valid hostname in your *ServerName* directive, try to start the server again.

Finding WS2_32.DLL on Windows

When trying to start Apache on Windows 95, a message like Unable To Locate WS2_32.DLL... appears. This file is necessary for Apache to function properly.

Prior to Version 1.3.9, Apache for Windows used Winsock 1.1. Beginning with Version 1.3.9, Apache began using Winsock 2 features (specifically, `WSADuplicateSocket()`). *WS2_32.DLL* implements the Winsock 2 API. Winsock 2 ships with Windows NT 4.0 and Windows 98. Some of the earlier releases of Windows 95 did not include Winsock 2.

To fix it, install Winsock 2, available at *http://www.microsoft.com/windows95/down loads/*. Then restart your server, and the problem should be gone.

Fixing WSADuplicateSocket Errors

If, when trying to start Apache on Windows, it fails and the Apache error log contains this message:

```
[crit] (10045) The attempted operation is not supported for the type of object
    referenced: Parent: WSADuplicateSocket failed for socket ###
```

it indicates that your system is using a firewall product that has inserted itself into the network software but doesn't fully provide all the functionality of the native network calls.

To get rid of the problem, you need to reconfigure, disable, or remove the firewall product that is running on the same box as the Apache server.

This problem has been seen when Apache is run on systems along with Virtual Private Networking (VPN) clients such as *Aventail Connect*. *Aventail Connect* is a Layered

Service Provider (LSP) that inserts itself, as a *shim*, between the Winsock 2 API and Windows' native Winsock 2 implementation. The *Aventail Connect* shim does not implement `WSADuplicateSocket`, which is the cause of the failure.

The shim is not unloaded when *Aventail Connect* is shut down. Once observed, the problem persists until the shim is either explicitly unloaded or the machine is rebooted.

Another potential solution (not tested) is to add *apache.exe* to the *Aventail Connect* exclusion list (see below).

Apache is affected in a similar way by any firewall program that isn't correctly configured. Assure you exclude your Apache server ports (usually port 80) from the list of ports to block. Refer to your firewall program's documentation for the how-to.

Relevant information specific to *Aventail Connect* can be found at *How to Add an Application to Aventail Connect's Application Exclusion List* at *http://support.aventail.com/akb/article00586.html*.

Handling System Error 1067

monospace

Sometimes, when starting Apache on Windows, you might get a message like "System error 1067 has occurred. The process terminated unexpectedly." This unfortunately uninformative message means that the Web server was unable to start correctly as a service for one reason or another.

As with any error, the first step should be to check your Apache error log. If that doesn't reveal anything useful, try checking the Windows application event log to find out why Apache won't start. If that doesn't help, try:

```
D:\>c:
C:\>cd "\Program Files\Apache Group\Apache"
C:\Program Files\Apache Group\Apache>apache
```

(If you don't get the prompt back, hit Ctrl-C to cause Apache to exit.)

This will run Apache interactively rather than as a service; any error messages should show up on your screen rather than being concealed behind a System Error 1067 alert box.

monospace

Fixing Build-Time Error Messages

__inet Symbols

If you have installed BIND-8, then this is normally because of a conflict between your include files and your libraries. BIND-8 installs its include files and libraries in */usr/local/include/* and */usr/local/lib/*, whereas the resolver that comes with your system is probably installed in */usr/include/* and */usr/lib/*.

If your system uses the header files in */usr/local/include/* before those in */usr/include/* but you do not use the new resolver library, then the two versions will conflict. To resolve this, you can either make sure you use the include files and libraries that came with your system, or make sure to use the new include files and libraries.

If you're using Apache 2.0 or later, or Apache 1.3 with the APACI build script, you can make changes to the library search lists by defining them on the *./configure* command line:

```
% LIBS=-lbind ./configure ...
```

If you're using Apache 1.3 or earlier and controlling the build process by editing the *Configuration* file directly, just add `-lbind` to the EXTRA_LDFLAGS line in the file.

After making the appropriate change to your build configuration process, Apache should build with the correct library.

Apache Versions 1.2 and earlier use EXTRA_LFLAGS in the *Configuration* file instead.

As of BIND 8.1.1, the *bind* libraries and files are installed under */usr/local/bind* by default, so you should not run into this problem. Should you want to use the bind resolvers, you'll have to add the following to the respective lines:

- For Apache 1.3 with APACI, or 2.0 and later:

  ```
  % CFLAGS=-I/usr/local/bin/include \
  > LDFLAGS=/usr/local/bind/lib LIBS=-lbind \
  > ./configure ...
  ```

- For Apache 1.2 or 1.3 with direct editing of *Configuration*, add/change the following lines in the file:

  ```
  EXTRA_CFLAGS=-I/usr/local/bind/include
  EXTRA_LDFLAGS=-L/usr/local/bind/lib
  EXTRA_LIBS=-lbind
  ```

Getting Server-Side Includes to Work

The solution is to make sure that *Options Includes* is turned on and that either *XBitHack* is turned *On*, or that you have the appropriate *AddHandler* directives set on the file type that you are using.

As discussed in Recipe 8.9, there are a number of ways to enable SSI. If the unparsed SSI directives are appearing in the HTML when the page is loaded, this is a clear indication that SSI execution is not enabled for the document in question.

If the server has difficulty parsing an SSI directive, it will substitute the phrases "An error occurred while processing this directive" in its place in the response. If this happens, the cause of the problem should be listed in the server's error log. See also Recipe 8.13.

Debugging Rewrites That Result in "Not Found" Errors

If your *RewriteRule* directives keep resulting in 404 Not Found error pages, add the PT (PassThrough) flag to the *RewriteRule* line. Without this flag, Apache won't process a lot of other factors that might apply, such as *Alias* settings.

You can verify that this is the cause of your problem by cranking the *mod_rewrite* logging level up to 9 and seeing that the entries relating to the RewriteRule mention something about prefixes with document_root:

```
RewriteLog logs/rewrite-log
RewriteLogLevel 9

% tail logs/rewrite_log
ip-address - - [date] [reqid] (2) prefixed with document_root to
/usr/local/apache/htdocs/robots.text
ip-address - - [date] [reqid] (1) go-ahead with
/usr/local/apache/htdocs/robots.text [OK]
```

Don't forget to turn off the *RewriteLog* directive, or possibly just turn down the logging level, after you've done your checking! Otherwise your disk space may disappear like the snows of yesteryear.

Without the PT flag, *mod_rewrite* assumes that any rewriting it does will be the last URL manipulation the server needs to do for the request. Because *mod_rewrite* directives are handled very early in request processing, this can mean that *Alias*, *ScriptAlias*, and other URL manipulations may not get executed. Specifying the flag tells *mod_rewrite* to not short-circuit processing but to let it continue as usual.

.htaccess Files Having No Effect

Make sure that *AllowOverride* is set to an appropriate value. Then, to make sure that the *.htaccess* file is being parsed at all, put the following line in the file and ensure that it causes a server error page to show up in your browser:

```
Garbage Goes Here
```

.htaccess files override the settings in the main server configuration file. Because this is frequently an undesired thing, *.htaccess* files are frequently disabled, which will cause your *.htaccess* file to be ignored.

.htaccess files are enabled using the *AllowOverride* directive, which lists categories of directives that may appear in an *.htaccess* file. For example, if you wish to put authentication-related directives in an *.htaccess* file, you will need to put the following line in the main server configuration file:

```
AllowOverride AuthConfig
```

AllowOverride All permits any directive to be put in the *.htaccess* file, while the directive *AllowOverride None* means, "Please ignore my *.htaccess* files."

Thus, the most common cause of an *.htaccess* file being ignored is simply that your configuration file tells Apache to ignore it.

If you put garbage in your *.htaccess* file, this should generate a Server Error message in the browser, which will verify that Apache is indeed looking at the contents of your file. However, if such a message is not displayed, this is a sure sign that your *.htaccess* file is being completely ignored.

Address Already in Use

If, when attempting to start your Apache server, you get the following error message:

```
[Thu May 15 01:23:40 2003] [crit] (98)Address already in use: make_sock: could not
    bind to port 80
```

One of three things is happening:

- You are attempting to start the server as a nonroot user. Become the root user and try again.
- There is already some process running (perhaps another Apache server) using port 80. Run *netstat*, or perhaps look at the process list and kill any process that seems to fill this role.
- You have more than one *Listen* directive in your configuration file pointing to the same port number. Find the offending duplicate directive and remove it.

In the case of the first condition, you will need to become the root user in order to start Apache. By long tradition, only the root user may bind to any port lower than 1025. Because Apache typically runs on port 80, this requires root privileges.

The second condition can be a little trickier. Sometimes a child process will refuse to die and will remain running after Apache has been shut down. There are numerous reasons this might happen. Most of the time, you can kill this process forcibly using *kill* or *kill -9* while logged in as root. As long as this process is running and has the port occupied, you will be unable to start anything else wanting to bind to that same port.

In the case of the third condition, the second *Listen* directive attempts to bind to port 80, which has already been taken by the first *Listen* directive. Simply removing one of the *Listen* directives will clear up this problem.