CHAPTER 10

Proxies

Proxy means to act on behalf of another. In the context of a Web server, this means one server fetching content from another server, then returning it to the client. For example, you may have several Web servers that hide behind a proxy server. The proxy server is responsible for having requests end up going to the right backend server.

mod_proxy, which comes with Apache, handles proxying behavior. The recipes in this chapter cover various techniques that can be used to take advantage of this capability. We discuss securing your proxy server, caching content proxied through your server, and ways to use *mod_proxy* to map requests to services running on alternate ports.

Additional information about *mod_proxy* can be found at *http://httpd.apache.org/docs/mod/mod_proxy.html* for Apache 1.3, or *http://httpd.apache.org/docs/2.0/mod/mod_proxy.html* for Apache 2.0.

Apache 2.2 introduces a number of sub-modules, such as *mod_proxy_balancer*, which give additional functionality to *mod_proxy*. These will be touched on in this chapter, too.

Please make sure that you don't enable proxying until you understand the security concerns involved and have taken steps to secure your proxy server. (See Recipe 6.20 for details.)

You also may wish to consider a dedicated proxy server, such as Squid (*http://www.squid-cache.org/*), which is focused entirely on one task, and thus has more options related to this task.

10.1 Securing Your Proxy Server

Problem

You want to enable proxying, but you don't want an open proxy that can be used by just anyone at all.

Solution

For Apache 1.3:

```
<Directory proxy:*>
    Order deny,allow
    Deny from all
    Allow from .yourdomain.com
</Directory>
```

For Apache 2.0:

```
<Proxy *>
    Order Deny,Allow
    Deny from all
    Allow from .yourdomain.com
</Proxy>
```

Discussion

Running an open proxy is a concern because it permits users from the Internet to use your proxy server to cover their tracks as they visit Web sites. This can be a problem for a variety of reasons. The user is effectively stealing your bandwidth and is certainly part of the problem. However, perhaps more concerning is the fact that you are probably enabling people to circumvent restrictions that have been put in place by their network administrators, or perhaps you are providing users with anonymity while they visit a Web site, and as a consequence, these visits appear to come from your network.

In these recipes, `.yourdomain.com` should be replaced by the name of your particular domain, or, better yet, the network address(es) that are on your network. (IP addresses are harder to fake than host and domain names.) For example, rather than the line appearing in the recipe, you might use a line such as:

```
Allow from 192.168.1
```

Note that every request for resources that goes through your proxy server generates a logfile entry, containing the address of the client and the resource that they requested through your proxy server. For example, one such request might look like:

```
192.168.1.5 - - [26/Feb/2003:21:26:13 -0500] "GET http://httpd.apache.org/docs/mod/
     mod_proxy.html HTTP/1.1" 200 49890
```

Your users, if made aware of this fact, will no doubt find it invasive, because this will show all HTTP traffic through the proxy server.

It is possible to configure your server not to log these requests. The technique for doing this is to set an environment variable for proxied requests:

```
<Directory proxy:*>
    SetEnv PROXIED 1
</Directory>
```

Then, in your log directive, specify that these requests are not to be logged:

```
CustomLog /www/logs/access_log common env=!PROXIED
```

See Also

- *http://httpd.apache.org/docs/mod/mod_proxy.html*
- *http://httpd.apache.org/docs/mod/mod_log_config.html*

10.2 Preventing Your Proxy Server from Being Used as an Open Mail Relay

Problem

If your Apache server is set up to operate as a proxy, it is possible for it to be used as a mail relay unless precautions are taken. This means that your system may be functioning as an "open relay" even though your mail server software is actually securely configured.

Solution

Use *mod_rewrite* to forbid proxy requests to port 25 (SMTP):

```
<Directory proxy:*>
    RewriteEngine On
    RewriteRule "^proxy:[a-z]*://[^/]*:25(/|$)" "-" [F,NC,L]
</Directory>
```

Discussion

To use the Apache proxy as an SMTP relay is fairly trivial, but then so is preventing it. The solution simply tells the server to respond with a `403 Forbidden` to any attempts to use it to proxy to a remote mail server (port 25). Other ports, such as HTTP (port 80), HTTPS (port 443), and FTP (ports 20 and 21), which are commonly permitted proxy access, will not be affected.

See Also

- *http://httpd.apache.org/docs/mod/mod_proxy.html*
- *http://httpd.apache.org/docs/mod/core.html#directory*
- *http://httpd.apache.org/docs/mod/mod_rewrite.html*

10.3 Forwarding Requests to Another Server

Problem

You want requests for particular URLs to be transparently forwarded to another server.

Solution

Use *ProxyPass* and *ProxyPassReverse* directives in your *httpd.conf*:

```
ProxyPass /other/ http://other.server.com/
ProxyPassReverse /other/ http://other.server.com/
```

Discussion

These directives will cause requests to URLs starting with */other/* to be forwarded to the server *other.server.com*, with the path information preserved. That is to say, a request for *http://www.server.com/other/something.html* will be translated into a request for *http://other.server.com/something.html*. Content obtained from this other server will be returned to the client, which will be unable to determine that any such technique was employed. The *ProxyPassReverse* directive ensures that any redirect headers sent from the backend server (in this case, *other.server.com*) will be modified so that they appear to come from the main server.

This method is often used to have the dynamic portion of the site served by a server running *mod_perl*—often even on the same machine, but on a different port—while the static portions of the site are served from the main server, which can be lighter weight, and so run faster.

Note that URLs contained within documents are not rewritten as they pass through the proxy, and links within documents should be relative, rather than absolute, so that they work correctly.

Use this recipe when you have a frontend server and one or more backend servers, inaccessible from the Internet, and you wish to serve content from them. In the example given, when a request is made for a URL starting with `/other/`, Apache makes a request for the URL *http://other.server.com/*, and returns the content obtained by the client. For example, a request for the URL `/other/example.html` results in a request for the URL *http://other.server.com/example.html*.

The *ProxyPassReverse* directive ensures that any header fields returned by the secondary server (which contain the name of the server, such as `Location` headers) will be rewritten to contain the URL that the end user will actually be using, ensuring that the redirect actually functions as desired.

Note that links within HTML documents on the secondary site should all be relative, rather than absolute, so that these links work for users using the content *via* the proxy server. In the recipe given, for example, a link to */index.html* removes the `/other/` portion of the URL, causing the request to no longer hit the proxied portion of the server.

Using this technique, you can have content for one Web site actually served by multiple Web server machines. This can be used as a means to traverse the border of your network, or it can be used as a load-sharing technique to lessen the burden on your primary Web server.

See Also

- *http://httpd.apache.org/docs/mod/mod_proxy.html*
- Recipe 11.12

10.4 Blocking Proxied Requests to Certain Places

Problem

You want to use your proxy server as a content filter, forbidding requests to certain places.

Solution

Use *ProxyBlock* in the *httpd.conf* to deny access to particular sites:

```
ProxyBlock forbiddensite.com www.competitor.com monster.com
```

Discussion

This example forbids proxied requests to the sites listed. These arguments are substring matches; `example.com` will also match `www.example.com`, and an argument of `example` would match both.

If you want more fine-grained control of what content is requested through your proxy server, you may want to use something more sophisticated, such as Squid, which is more full-featured in that area.

See Also

- The Squid proxy server, found at *http://www.squid-cache.org/*

10.5 Proxying mod_perl Content to Another Server

Problem

You want to run a second HTTP server for dynamically generated content and have Apache transparently map requests for this content to the other server.

Solution

First, install Apache, running on an alternate port, such as port 90, on which you will generate this dynamic content. Then, on your main server:

```
ProxyPass /dynamic/ http://localhost:90/
ProxyPassReverse /dynamic/ http://localhost:90/
```

Discussion

Most dynamic content generation techniques use a great deal more system resources than serving static content. This can slow down the process of serving static content from the same server, because child processes will be consumed with producing this dynamic content, and thus unable to serve the static files.

By giving the dynamic content its own dedicated server, you allow the static content to be served much more rapidly, and the dynamic content has a dedicated server. Each of the servers can have a smaller set of modules installed than they would otherwise require, because they will be performing a smaller subset of the functionality needed to do both tasks.

This technique can be used for a *mod_perl* server, a PHP server, or any other dynamic content method. Or you could reverse the technique and have, for example, a dedicated machine for serving image files using *mod_mmap_static* to serve the files very rapidly out of an in-memory cache.

In the example given, all URLs starting with */dynamic/* will be forwarded on to the other server, which will, presumably, handle only requests for dynamic content. URLs that do not match this URL, however, will fall through and be handled by the frontend server.

10.6 Configuring a Caching Proxy Server

Problem

You want to run a caching proxy server.

Solution

Configure your server to proxy requests, and provide a location for the cached files to be placed:

```
ProxyRequests on
CacheRoot /var/spool/httpd/proxy
```

Discussion

Running a caching proxy server allows users on your network to have more rapid access to content that others have already requested. They will, perhaps, not be getting the most recent version of the document in question, but, because they are retrieving the content from a local copy rather than from the remote Web server, they will get it much more quickly.

With the contents of the WWW growing ever more dynamic, running a caching proxy server perhaps makes less sense than it once did, when most of the Web was composed of static content. However, because *mod_proxy* is fairly smart about what it caches and what it does not cache, this sort of setup will still speed things up by caching the static portions of documents, such as the image files, while retrieving the freshest version of those documents that change over time.

The directory specified in the *CacheRoot* directive specifies where cached content will be stored. This directory must be writable by the user that Apache is running as (typically nobody), so that it is able to store these files there.

Finally, note that, while in Apache 1.3, the functions discussed here are provided by *mod_proxy*; in Apache 2.0, the proxying and caching functionality have been split into the modules *mod_proxy* and *mod_cache*, respectively. In either case, these modules are not enabled by default.

See Also

- *http://httpd.apache.org/docs/mod/mod_proxy.html*

10.7 Filtering Proxied Content

Problem

You want to apply some filter to proxied content, such as altering certain words.

Solution

In Apache 2.0 and later, you can use *mod_ext_filter* to create output filters to apply to content before it is sent to the user:

```
ExtFilterDefine naughtywords mode=output intype=text/html
    cmd="/bin/sed s/darned/blasted/g"

<Proxy *>
    SetOutputFilter naughtywords
</Proxy>
```

Discussion

The recipe offered is a very simple-minded "naughty word" filter, replacing the naughty word "darned" with the sanitized alternate "blasted." This could be expanded to a variety of more sophisticated content modification, because the *cmd* argument can be any command line, such as a Perl script, or arbitrary program, which can filter the content in any way you want. All proxied content will be passed through this filter before it is delivered to the client.

Note that this recipe will work only in Apache 2.0, as the module *mod_ext_filter*, the *SetOutputFilter* directive, and the *<Proxy>* directive are available only in Apache 2.0.

Note also that there are ethical and legal issues surrounding techniques like this, which you may need to deal with. We don't presume to take a position on any of them. In particular, modifying proxied content that does not belong to you may be a violation of the owner's copyright and may be considered by some to be unethical. Thankfully, this is just a technical book, not a philosophical one. We can tell you how to do it, but whether you should is left to your conscience and your lawyers.

See Also

- *http://httpd.apache.org/docs-2.0/mod/mod_proxy.html*
- *http://httpd.apache.org/docs-2.0/mod/mod_ext_filter.html*

10.8 Requiring Authentication for a Proxied Server

Problem

You wish to proxy content from a server, but it requires a login and password before content may be served from this proxied site.

Solution

Use standard authentication techniques to require logins for proxied content:

```
ProxyPass "/secretserver/" "http://127.0.0.1:8080"
<Directory "proxy:http://127.0.0.1:8080/">
    AuthName SecretServer
    AuthType Basic
    AuthUserFile /path/to/secretserver.htpasswd
    Require valid-user
</Directory>
```

Discussion

This technique can be useful if you are running some sort of special-purpose or limited-function Web server on your system, but you want to apply Apache's rich set of access control and its other features to access it. This is done by using the *ProxyPass* directive to make the special-purpose server's URI space part of your main server, and using the special `proxy:path` *<Directory>* container syntax to apply Apache settings only to the mapped URIs.

See Also

- Recipe 6.7

10.9 Load Balancing with *mod_proxy_balancer*

Problem

You want to balance the load between several backend servers.

Solution

Use *mod_proxy_balancer* to create a load-balanced cluster:

```
<Proxy balancer://mycluster>
    BalancerMember http://192.168.1.50:80
    BalancerMember http://192.168.1.51:80
</Proxy>
ProxyPass /application balancer://mycluster/
```

Discussion

mod_proxy_balancer is an exciting new module that provides load balancing between multiple backend servers. This kind of functionality has traditionally been associated with expensive and complex commercial solutions. This module makes this simple to configure, and it's included in the standard installation of the Apache web server.

The example given above sets up a two-member balanced cluster, and proxies the URL `/application` to that cluster.

mod_proxy_balancer offers a wide variety of options, which you can find in detail in the documentation, at *http://httpd.apache.org/docs/2.2/mod/mod_proxy_balanc er.html*.

For example, you can indicate that a particular server is more powerful than another, and so should be allowed to assume more of the load than other machines in the cluster. In the following configuration line, we indicate that one particular machine should receive twice as much traffic as other machines.

```
BalancerMember http://192.168.1.51:80 loadfactor=2
```

Traffic may be balanced by traffic (bytes transferred) or by request (number of requests made per host) by putting additional arguments on the *ProxyPass* directive:

```
ProxyPass /application balancer://mycluster/ lbmethod=bytraffic
```

See the *mod_proxy* documentation for more information on this point.

And there is a web-based balancer manager tool, which can be configured as follows:

```
<Location /balancer-manager>
    SetHandler balancer-manager
</Location>
```

The balancer manager lets you set servers available or unavailable, and change their load factor, without restarting the server. This allows you to take servers offline for maintenance, do whatever needs to be done, and bring them back up, without ever affecting the end user in any way.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html*

10.10 Proxied Virtual Host

Problem

You want to have an entire virtual host proxied to a different server.

Solution

Place a *ProxyPass* directive in your *VirtualHost* configuration block:

```
<VirtualaHost *:80>
    ServerName server2.example.com
    ProxyPass / http://192.168.1.52:80
    ProxyPassReverse / http://192.168.1.52:80
</VirtualHost>
```

Discussion

The recipe shown above will pass all requests to this virtual host to the specified backend server, and serve the content from there. The *ProxyPassReverse* directive ensures that redirects issues from the backend server will be correctly rewritten to the frontend server, rather than having clients try to request content directly from the backend server.

It will usually be useful to collect log files on the frontend server, rather than on the backend server. Requests to the backend server will all appear to come from the proxy server, rather than from the original client address. However, log files collected on the proxy (frontend) server will have the original client address.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_proxy.html*

10.11 Refusing to Proxy FTP

Problem

You want to make sure that FTP (or, perhaps, other protocols) are not proxied through your server.

Solution

Make sure that *mod_proxy_ftp* isn't loaded:

```
# LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
```

Discussion

mod_proxy has several helper modules that provide the protocol-specific proxying functionality. These modules are *mod_proxy_http*, for proxying HTTP requests, *mod_proxy_ftp*, for proxying FTP requests, and *mod_proxy_connect*, for support for the `CONNECT` HTTP method, used primarily for tunneling SSL requests through proxy servers.

If you with for *mod_proxy* to never proxy FTP requests, you need merely to ensure that the *LoadModule* directive for *mod_proxy_ftp* is commented out, as shown above.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_proxy_ftp.html*
- *http://httpd.apache.org/docs/2.2/mod/mod_proxy.html*