Membership lists PROTECT YOUR WEB PAGES

Some Web pages restrict access to authorised members. If you've ever wondered how this is done then wonder no more, David Tansley shows us how

If you're lucky enough to have been to a Formula One Grand Prix, then you'll know that there is a members' area and a VIP area. By entering these areas you'll have immediate privileges and access not afforded to the general public. A similar thing can be said about Web pages. Most Web pages you will want everyone to see, after all that's the whole point of the World Wide Web. On some pages, however, you may want to restrict the viewing to members or special users – VIPs, if you will.

You can protect Web pages based upon the calling browser, IP address, domain name or simply via password protection. We will look at the latter, which is more commonly known as basic authentication. We will also look at how to personalise those nasty error messages that get thrown in your face when you try to go to a page that is missing (see Figure 1.) or to an unauthorised area.

The challenge/response process

First look at the process involved. Here's how it goes: you point your browser to a Web page protected by a username and password. The Web server then looks for a file in that directory called .htaccess, if that file is present it reads the directives (configuration) to obtain the type of authentication (if any) and what files to protect with this information; authentication begins. What happens now is commonly called the challenge/response cycle. The Web server sends an authentication request to your browser, the browser will prompt you for a user name and password within a dialog box. The user enters their username/password then clicks on OK and the information is sent back to the Web server. The Web server then validates the username and password against the information held in a password file. If the user passes the authentication (there is a username/password match) the page is displayed, if not then the Web server throws up a 401 error page in the browser, see Figure 2.

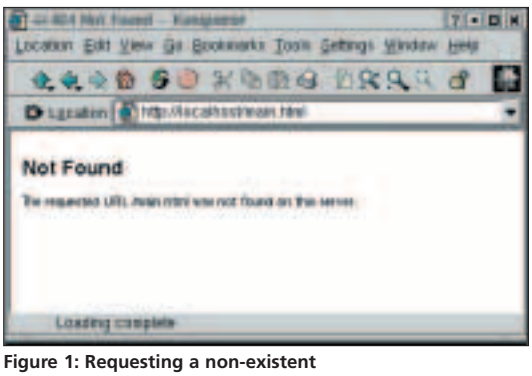

Figure 1: Requesting a non-existent document. A 404 error page

Setting up your password file

To enable access to certain users, you first have to create a password file. We will call our password file, ".ht_users", though you can chose your own meaningful name if you wish. Now this is not the password file that gets read when you login to your Linux machine, this is a totally different file.

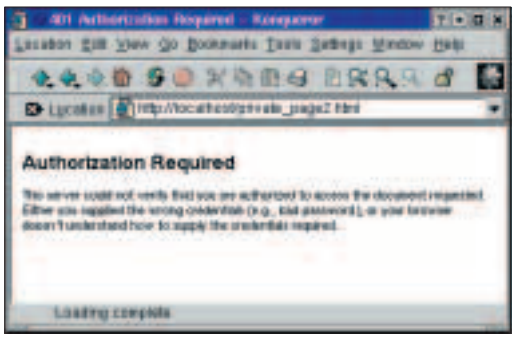

Figure 2: Authentication Failed. A 401 error page

To create this file we use the *htpasswd* utility, which enables you to add users and their passwords to an encrypted flat password file. As this file will hold users' names and passwords it is best to stick this file (at least) off the main Web root directory. For goodness sake, DO NOT put it in your HTML, CGI-BIN or ICONS directory. Create a new directory, called "private" say, off the www directory. (All Apache installs now stick your HTML (or HTDOCS), CGI-BIN and ICONS directory within this www directory layout structure by default.)

Next, lets create a couple of users, say "davetan" and "paulinetan".:

```
$ mkdir private
$ pwd
/var/www/private
$ htpasswd -c /var/www/private/.ht_users
davetan
New password:
Re-type new password:
Adding password for user davetan
$ htpasswd /var/www/private/.ht_users
paulinetan
New password:
Re-type new password:
Adding password for user paulinetan
```

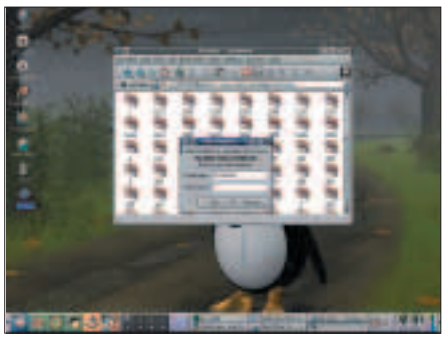

Figure 3: A challenge/response dialogue box.

Notice that when adding user davetan we have used the *-c* after *htpasswd*. The *-c* option tells htpasswd that this is a new file and thus a new file should be created. We give the full pathname to the location of the *passwd* file(*.ht_users*). In this case we are sticking the file in */var/www/private* – you may want to use a different directory structure. After a space, the username we are adding is given. Finally htpasswd prompts for password confirmations for that user. Adding user paulinetan, there is no need to specify the *-c* option as we do not want to create a new file, only append to it. If you use the *-c*, guess what, the file contents previously held will be wiped.

Here's how the file we just created looks:

```
$ more .ht_users
davetan:ETEkRxqtoentY
paulinetan:C.ePHk1ASFlIs
```

Notice the user names and password are colon separated and the passwords are encrypted.

Informing the Apache Web server

By default, Apache comes pretty much secure. Locate the httpd.conf file and do a bit of editing. To find out where your httpd.conf resides use the find utility to do all the work for you:

```
$ find / -name "httpd.conf" -print
/etc/httpd/conf/httpd.conf
```

Next using *vi*, *vim* or some other text editor, edit http.conf and locate the directory directive: <Directory>. Make sure you have the correct AllowOveride entry within this directive, it will probably have:

```
AllowOverride None
```

Change this to "AllowOverride All", so you have an entry like so:

```
<Directory />
Options None
AllowOverride All
</Directory>
```

If you have made changes to your configuration file, you must restart the Apache Web server. On a Red Hat box with Apache put in place at installation, you can use the *rc* script to stop/start the Apache Web server:

```
$ /etc/rc.d/init.d/httpd restart
```

You can use different realms to protect different parts of your Web page directory

Setting up the .htaccess file

Now for the meaty part. Change into the directory where the HTML files you wish to protect are located and create a .htaccess file. For example, to protect all pages that start with the word "private" at the beginning of the file, the following pattern match will do it for us:

```
private*.*
```

So the above pattern would match all of these files: private_main.html, privatepage1.html, private_page2.html and private.php.

Create a file called .htaccess with the following contents:

```
AuthUserFile /var/www/private/.ht_users
AuthNAME "Hey! Restricted Directory"
AuthTYPE "Basic"

<Files private*.*>
require valid-user
</Files>
```

In the first line, AuthUserFile instructs Apache where the file we created to hold the usernames and password is located. In the second line, AuthName is the Realm Name – you can use different realms to protect different parts of your Web page directory structure. For the basics, just use it as a header line that will be displayed on the dialog box when a browser tries to access a protected page. You must enclose this with double quotes if you have more than one word, as above. In the third line, AuthTYPE is basic; this means we are only using Basic Authentication, as mentioned at the beginning of the article.

The Files directive specifies that we are protecting the files "private*.*", which will protect all files that match this pattern. The require valid-user, means the HTML page(s) matched will not be loaded unless the user first gets successfully authenticated.

Now load up the browser and point to a file that is protected and you will get a challenge sent from the server to your Web browser, similar to Figure 3. If you hit cancel your browser will throw up a 401 error page, as in Figure 2. Assuming you enter a correct username/password, the protected page you requested will be displayed.

Other examples

To limit access to a page to a single user:

```
<Files top_secret.html>
require user davetan
</Files>
```

The above only allows the user davetan to access the

page top_secret.html

You may be thinking, what if somebody points their browser to an HTML directory and specifically tries to load a .htaccess file. No problem, just deny viewing from everybody:

```
<Files .htaccess>
deny from all
</Files>
```

The above file directive will set the state to deny from everybody. Your .htaccess file is safe. If some one tries to access it directly, a 403 forbidden error page will be thrown up in their browser, saying it does not have access to this file. Neat, eh?

Personalising error pages

Ever gone to a broken link and had a totally unfriendly "Not Found" document thrown in your face? It is possible to make these pages friendlier to the calling browser, however. There are quite a few error code pages on a Web server. The most common ones are:

204 No content
401 Authorisation Required
403 Forbidden
404 Not Found
500 Internal Server Error

Lets see how to create a "404 Not Found" error page; the principles are the same for other error pages you wish to personalise. All you need to do is put an entry in your .htaccess file (that you created earlier). Like so:

```
ErrorDocument 404 /icons/not_found404.html
```

Each ErrorDocument for a different error code must go on a new line. The format of the entry is:

```
ErrorDocument <error code> <path to error page>
```

In the example shown above, I have put my error document in */icons*, which is off the Web root directory. You are not restricted where you put these HTML pages; some like to create a separate directory and stick them in there – it's up to you. Also notice the name I have given to the HTML page is a meaningful one that corresponds to the actual error code page. In my example I have used not_fouund404.html, so I know it is concerned with the 404 error code page

When throwing up personalised error pages it is considered good practise to always put a link back to your homepage, or at least to some main Web site (like *http://www.netscape.com*). There should also be a way for the user to complain that some thing is wrong. Listing 1, shows my very sparse, but more friendly HTML code for a 404 error page.

Listing 1: not_found404.html

```
<HTML>
I am sorry, but the file you requested could not be found,<BR> it may
have been
moved, deleted or simply just does not exist.<BR>

Back to <A HREF=/index.php>Home</A></strong><BR>
If you have a query or something we should know about email the
administrator
at <a href="mailto:webadmin@localhost">webadmin@localhost</a><BR>
<BR><BR>
<HR>
<CENTER><IMG SRC="/icons/apache_pb.gif" border=0></CENTER>
<HR>
</HTML>
```

Please note that you do not have to create usernames/passwords if you only wish to personalise your error pages, simply create a .htaccess file and insert the entries for the error pages you are personalising, as shown above.

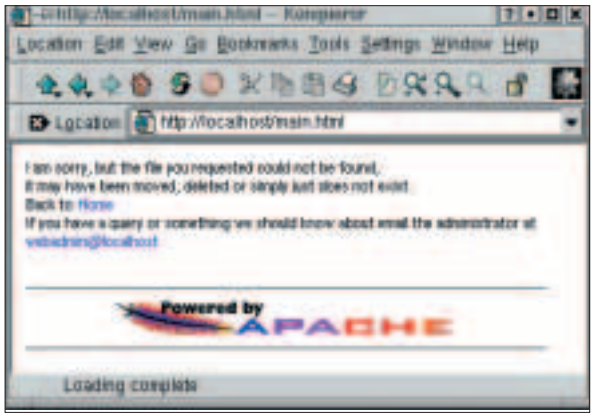

Figure 4: Personalised 404 error page

Conclusion

I have demonstrated how to carry out basic authentication on a Web server protecting individual or many Web pages based via basic pattern matching. There are many more directives that you can specify, however space does not allow me to go through all of them. When testing your .htaccess configuration directives it is always a good idea to open up a new shell window and continuously page the end of your error log file, so you can pick up any mis-configurations you may have in the .htaccess file straight away and fix them. Like so:

```
$ tail -f error_log
```

When a user has been validated, they remain validated, even if they go off to another site then come back to view the same protected page again so long as they have not closed down their browser. To re-set the authentication the calling browser must re-start their browser. Bear this in mind when testing your authentication procedures.

Being able to personalise your error pages, makes your Web site friendlier and more professional to a user visiting your site. When these types of hiccups do happen, it shows you care about your Web site.

Info

Apache homepage:
www.apache.com