Adding authentication to your website

PAGE PROTECTION

Apache offers several options for adding a password-protected area to a website. **BY FRANK WILES**

If you want to offer login access to restricted web pages, you don't need a MySpace account or a big corporate website. Apache provides several convenient alternatives for supporting user authentication. Although these login options require a few extra configuration steps, you can easily protect your pages without the need for add-on, proprietary applications. In this article, I will describe some techniques for password-protecting your pages.

The Apache web server goes through three phases to determine whether the current user is allowed to view the requested resource. The *Access* phase checks to see whether the requesting IP address is allowed to view the resource. The *Authentication* phase verifies that the username provided matches the password associated with the user. The *Authorization* phase is usually used to support user groups for easier administration. With a bit of custom coding, I will explain how you can extend any of these phases to do whatever you want.

Unless your connection is over SSL, all of these methods will send your password in the clear. Using SSL for all pages that require authentication is highly recommended.

Examples in this article were tested with Apache 2.2.8. If you are using an earlier version, some of these options might not be available or the configuration could differ slightly.

File-Based Authentication

File-based authentication, which is sometimes referred to as *Basic Auth* or *htpasswd auth*, is the most common technique for supporting user login in Apache. For example, if you want to protect all of the administration pages of our site, you can invoke file-based authentication with the following code in your Apache configuration file. (The Apache configuration file is located in different places on different systems. On Red Hat/Fedora systems, it is */etc/httpd/conf/httpd.conf*, and it's */etc/apache2/apache2.conf* on Ubuntu.)

```
<Location /admin>
AuthType Basic
AuthName "Admin Pages"
AuthUserFile /path/to/our/password-file
AuthBasicProvider file
Require valid-user
</Location>
```

If you're using Apache 2.0.x, you will need to remove the *AuthBasicProvider* directive.

After you set the *AuthType* to *Basic*, name this area (e.g., Admin Pages) so users know what they are logging into. Also, you must tell Apache where to find the password file for this area.

The final step is the *Require* directive, which tells Apache to allow any valid user in the file.

Apache comes with a program called *htpasswd* that helps you create and update the password file. Initially, you create the file with:

```
htpasswd -c ⮯
/path/to/our/password-file ⮯
<username>
```

Listing 1: Restricting User Access

```
<Location />
    AuthType Basic
    Authname "My Site"
    AuthBasicProvider file
    AuthUserFile /path/to/our/password-file
</Location>

<Location /admin>
    Require user steve bob
</Location>

<Location /content>
    Require valid-user
</Location>
```

After the file is created and you just want to add more users or change their passwords, use the same *htpasswd* command, but without the *-c* option. If you need to remove a user, you can either edit the file by hand in a text editor or use *htpasswd* with the *-D* option to delete the user.

Also, you could use this method (and the other methods I'll discuss later) with directives such as *<Directory>*, *<DirectoryMatch>*, *<LocationMatch>*, or *<FilesMatch>*. For example, you could require a password to view any *.gif* files on your site with:

```
<FilesMatch "\.gif$">
AuthType Basic
AuthName "Images Require a Password"
AuthUserFile /path/to/our/password-file
AuthBasicProvider file
Require valid-user
</FilesMatch>
```

In addition to allowing all users in the password file into an area, you can also restrict access down to specific users.

Listing 2: Working with Groups

```
<Location />
  AuthType Basic
  AuthName "My Site"
  AuthBasicProvider file
  AuthUserFile /path/to/our/password-file
  AuthGroupFile /path/to/our/group-file
</Location>

<Location /admin>
  Require group admin
</Location>

<Location /content>
  Require valid-user
</Location>
```

The configuration in Listing 1 grants *steve* and *bob* access to the */admin* section with the *Require user* entry, followed by the space-separated list of users. However, any authenticated user who logs in succesfully can access */content*.

Another useful feature is that you can grant access to specific groups rather than individual users. To do this, create a group file. Listing 2 shows the preceding example using groups. The format of the group file is very simple:

```
<group name>: <user1>... <userN>
```

For this example, the contents of the file would be:

```
admin: steve bob
```

In this way, you can create as many groups as you like – one per line. The use of groups is a great way to keep your maintenance time down and make your configuration easier to read and understand.

Note that it is important to make sure your password and group files aren't inside your Apache's *DocumentRoot*; otherwise, anyone could download them.

Authenticating with an SQL Database

Maybe your site already uses software that requires a password – a forum perhaps – and after you have several users in your forum, you decide to add another section to your website that is only for registered forum users.

Although you could have all of your users re-register with some form of file-based authentication (as described previously), this could turn into a maintenance nightmare, creating two places to add and modify users. Furthermore, some users might end up with different usernames and passwords for different parts of the site.

If your forum uses an SQL database and stores the password, you can configure Apache to access the password information from the database. The existing software will have to store its passwords with the same one-way hash as Apache – in this case, the *crypt()* function. Also, you'll need to enable two Apache modules: *mod_dbd.so* and *mod_authn_dbd.so*. To enable these modules, add these lines to your Apache configuration file:

```
LoadModule dbd_module modules/mod_dbd.so
LoadModule authn_dbd_module modules/mod_authn_dbd.so
```

Note that the second parameter is either a relative or a full path to the module itself; the exact value might differ on your system, depending on your distribution or how you compile Apache.

After those modules are loaded, you need to configure Apache to reference your SQL database to retrieve passwords. If you're using a PostgreSQL database, your configuration would look something like Listing 3. First, you must define the database driver – in this case, *pgsql* – for the PostgreSQL database. If you're using a MySQL database, set this driver to *mysql*. Apache also has built-in support for Oracle, SQLite2, and SQLite3 databases.

After defining the driver, you must pass the driver some parameters about how to connect to the database. In the case of MySQL or PostgreSQL, you need

Listing 3: Authenticating with SQL

```
DBDriver pgsql
DBDParams "host=localhost dbname=forum user=apache password=secret"

<Location /forum-users-only>
  AuthType Basic
  AuthName "Forum Users Only"
  AuthBasicProvider dbd
  Require valid-user

  AuthDBDUserPWQuery "SELECT password FROM users WHERE user = %s"
</Location>
```

to pass the database host, database name, user, and password.

The next few lines should be familiar by now; the only thing different is that *AuthBasicProvider* is set to use the *dbd* module. Setting up the actual user password query comes last. Because every database differs in table and column names, you must instruct Apache on how to go about retrieving a password from the database.

Authenticating with LDAP

Another of the more popular authentication modules is *mod_authnz_ldap*, which allows you to authenticate your website against an LDAP server. LDAP authentication can be very useful in corporate environments in which a central LDAP server handles all authentication across the company.

Enable Apache's LDAP module with:

```
LoadModule authnz_ldap_module ↩
modules/mod_authnz_ldap.so
```

To configure your Apache server to authenticate users with LDAP, you need to set the LDAP URL:

```
<Location /content>
AuthLDAPURL ↩
"ldap://ldap1.company.com/ou=↩
People, o=Company"
Require valid-user
</Location>
```

Also, you can define redundant LDAP servers for fault tolerance by just adding another server to the URL:

```
<Location /content> AuthLDAPURL ↩
 "ldap://ldap1.company.com ↩
 ldap2.company.com/ou=↩
 People, o=Company"
 Require valid-user
</Location>
```

The LDAP modules offer several options. By ensuring that the user has certain LDAP attributes, you can include group information and restrict access for a variety of situations.

Custom Authentication

Apache is a very extensible system – so flexible that you don't even have to use it for http. By writing Apache modules, you can extend Apache. Modules can be written in C, like the modules discussed in this article, but the C language is time consuming and cumbersome for web admins who aren't experienced software developers.

Thanks to modules such as *mod_perl* and *mod_python*, you can build your own custom Apache modules in more agile languages.

Apache provides great flexibility when dealing with access to content, but one thing it doesn't take into account is the time of day. As an example of how to customize Apache authentication, assume your company has a first and second shift and you want to ensure that first-shift employees can only access

Listing 4: No Weekend Work!

```perl
package ByShiftAuth;

use strict;
use warnings;

use Apache2::Access ();
use Apache2::RequestUtil ();
use Apache2::Const -compile => qw(OK HTTP_UNAUTHORIZED AUTH_REQUIRED);

use DBI;
use Digest::MD5 qw( md5_hex );

sub handler {
    my $r = shift;

    # See if this is the initial request or not, if it isn't
    # they are already authentication and we just need to reset
    # the username
    if( !$r->is_initial_req ) {

        if( defined $r->prev ) {
            $r->user( $r->prev->user );
        }

        return Apache2::Const::OK;

    }

    # Check to see if it's a weekend
    my $day_of_week = (localtime(time))[6];
    if( $day_of_week == 0 or $day_of_week == 6 ) {
        return Apache2::Const::HTTP_UNAUTHORIZED;
    }

    # Get the username and password
    my ($rc, $password) = $r->get_basic_auth_pw();
    my $user = $r->user;

    unless ( $user and $password ) {
        $r->note_basic_auth_failure;
        return( Apache2::Const::AUTH_REQUIRED );
    }

    # Now let's connect to our database and compare things in
    # our database we're going to store passwords as MD5 digests
    my $dbhv = DBI->connect('dbi:Pg:dbname=admin', 'apache', 'secret')
        or die "Cannot connect to database: $!";

    my $sth = $dbh->prepare( qq{
        SELECT password FROM users WHERE user = ? AND
            current_time BETWEEN shift_begin AND shift_end });

    $sth->execute( $user );
    my $db_password = $sth->fetchrow;
    $sth->finish;

    # Make sure we found a password for this user, if we don't
    # it means they don't exist or their shift isn't in progress
    if( !$db_password ) {
        $r->note_basic_auth_failure;
        return( Apache2::Const::AUTH_REQUIRED );
    }

    # Check to make sure the passwords match
    if( md5_hex( $password ) ne $db_passwd ) {
        $r->note_basic_auth_failure;
        return( Apache2::Const::AUTH_REQUIRED );
    }

    return( Apache2::Const::OK );

}
```

pages or applications during their work hours. First, enable *mod_perl* with:

```
LoadModule perl_module ↩
modules/mod_perl.so
```

mod_perl lets you override a particular phase of the Apache life cycle by writing a Perl module. In this case, the module compares the username and password but also makes sure the employee is authorized to work at the current time (Listing 4). Also, reject everyone who tries to work on Saturdays and Sundays.

This code can be placed anywhere in Perl's *@INC* path in a file named *ByShiftAuth.pm*. If you want to put it in another location, add the following to your Apache configuration:

```
<Perl>
use lib qw( /path/to/directory );
</Perl>
```

After loading in the necessary *mod_perl* modules, DBI, and the MD5 libraries, Listing 4 defines the handler.

First, check to see whether this is the initial request or some sort of internal redirect. If it's not the initial request, the system has already authenticated this request and can just pass along the information.

Next, check to make sure the current day isn't a weekend, and if it is, simply reject everyone. The script then obtains the username and password from the user.

If both a username and password aren't available, the script bails out and tells the browser to re-prompt with the *AUTH_REQUIRED* return value.

Next, the script must connect to the database and look for the user's password. The query assumes the start and end times of the user's shift are in the *users* table.

If the user doesn't exist or is attempting to login outside of their shift hours, this query won't return any data. In production, you would probably want to tell the user that the reason they can't log in is that it isn't their shift so that they don't keep retrying.

Finally, compare the MD5 digest of the password with the password retrieved from the database.

If the test fails, the user is prompted again. If all goes well, return the *OK* constant, which tells Apache that the user is authenticated.

Now that you see how the code works, the next step is to tell Apache to use it, which you can do by overriding the authentication phase with the custom code:

```
<Location /admin>
AuthType Basic
AuthName "Admin Pages"
PerlAuthenHandler ByShiftAuth
Require valid-user
</Location>
```

Conclusion

These short examples show some of the ways you can protect your web pages using more advanced Apache configurations. For more options, I encourage you to check the Apache website (*http://httpd.apache.org*). ■

THE MATHEMATICS OF HUMOUR

TWELVE Quirky Humans,
TWO Lovecraftian Horrors,
ONE Acerbic A.I.,
ONE Fluffy Ball of Innocence and
TEN Years of Archives
EQUALS
ONE Daily Cartoon that Covers the
Geek Gestalt from zero to infinity!

Over Two Million Geeks around the world can't be wrong!
COME JOIN THE INSANITY!

UserFriendly.Org