How to Protect apache directories with a password in Ubuntu

Question : How to Protect apache directories with a password in Ubuntu?

Answer:

To enable this option we have to use 2 terms .htaccess and htpasswd

Warning: On at least some versions of Ubuntu, .htaccess files will not work by default. See EnablingUseOfApacheHtaccessFiles for help on enabling them.

- Create a file called .htaccess in the directory you want to password-protect with the follwing content:

```
AuthUserFile /your/path/.htpasswd
AuthName "Authorization Required"
AuthType Basic
require valid-user
```

Instead of valid-user, you can also add the users you want directly

- If you want to password protect just a single file in a folder add the following lines to the .htaccess file:

```
<Files "mypage.html">
  Require valid-user
</Files>
```

Then create the file /your/path/.htpasswd which contains the users that are allowed to login and their passwords. We do that with the htpasswd command:

```
htpasswd -c /path/to/your/.htpasswd user1
```

Example:

```
pirat9@unixmen-laptop:/var/www$ sudo htpasswd-c /var/www/web/.htpasswd pirat9

Output

New password:
Re-type new password:
Adding password for user pirat9
pirat9@unixmen-laptop:/var/www$
```

You can see your crypted password with

```
pirat9@unixmen-laptop:/var/www$ sudo more /var/www/web/.htpasswd
pirat9:Amt1ZMf.BqDjC
```

- Finally we need to add the following lines to **`/etc/apache2/apache2.conf`**:

<Directory /your/path>
AllowOverride All
</Directory>

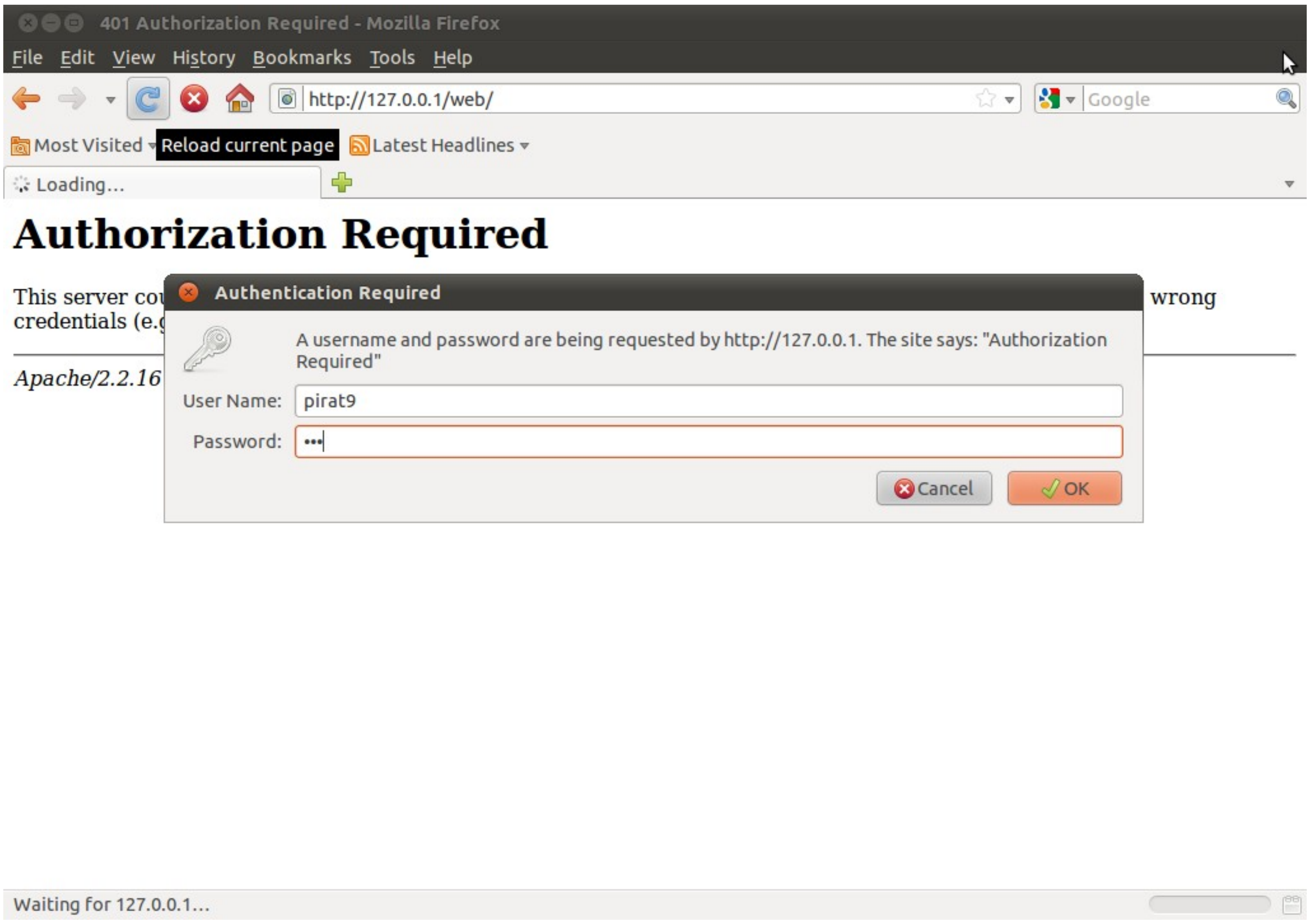
You have to adjust `/your/path/.htpasswd`

Restart your webserver:

```
sudo /etc/init.d/apache2 restart
```

Now restart apache and check

```
http://ip/path
```


401 Authorization Required - Mozilla Firefox

http://127.0.0.1/web/

Authorization Required

This server co wrong credentials (e.g

Apache/2.2.16

Authentication Required

A username and password are being requested by http://127.0.0.1. The site says: "Authorization Required"

User Name: pirat9

Password: •••

Waiting for 127.0.0.1...

create the `.htaccess` file.

```
sudo nano /var/www/html/.htaccess
```

Add this first line at the top of the new file to activate the `RewriteEngine`.

```
/var/www/html/.htaccess
RewriteEngine on
```

Save and exit the file.

To ensure that other users may only *read* your `.htaccess`, run the following command to update permissions.

```
sudo chmod 644 /var/www/html/.htaccess
```

You now have an operational `.htaccess` file, to govern your web application's routing rules.

Step 4 — Setting Up Files

In this section, we will set up a basic URL rewrite, which converts pretty URLs into actual paths to code. Specifically, we will allow users to access `example.com/about`.

We will begin by creating a file named `about.html`.

```
sudo nano /var/www/html/about.html
```

Copy the following code into the HTML page.

```
/var/www/html/about.html
<html>
    <head>
        <title>About Us</title>
    </head>
    <body>
        <h1>About Us</h1>
    </body>
</html>
```

You may access your web application at `your_server_ip/about.html` or `example.com/about.html`. Now notice that only `about.html` is accessible; if you try to access `your_server_ip/about`, you will get a **Not Found** error. We would like users to access `about` instead. Our rewrite rules will allow this very functionality.

Open up the `.htaccess` file.

```
sudo nano /var/www/html/.htaccess
```

After the first line, add the following.

```
/var/www/html/.htaccess
RewriteRule ^about$ about.html [NC]
```

Your file should now be identical to the following.

```
/var/www/html/.htaccess
RewriteEngine on
RewriteRule ^about$ about.html [NC]
```

Congratulations. You can now access `example.com/about` in your browser!

This is a good simple example that shows the general syntax that all Rewrite Rules follow.

`^about$` is the string that gets matched from the URL. That is, it's what the viewer types in her browser. Our example uses a few *metacharacters*.

- `^` indicates the start of the URL, after `example.com/` is stripped away.
- `$` indicates the end of the URL
- `about` matches the string "about"

`about.html` is the actual path that the user accesses; that is, Apache will still serve the `about.html` file.

`[NC]` is a *flag* that ignores capitalization in the URL.

With the rule shown above, the following URLs will point to `about.html`:

- `example.com/about`
- `example.com/About`
- `example.com/about.html`

The following will not:

- `example.com/about/`
- `example.com/contact`

Common Patterns

In this section, we will show some commonly-used directives.

Your web application is now running and is governed by a protected `.htaccess` file. The simplest example was included above. We will explore an additional two examples in this section.

You can set up example files at the result paths if you would like, but this tutorial does not include creating the HTML and PHP files; just the rules for rewriting.

Example 1: Simplifying Query Strings with RewriteRule

All `RewriteRule`s abide by the following format:

```
RewriteRule pattern substitution [flags]
```

- **RewriteRule**: specifies the directive `RewriteRule`
- **pattern**: a regular expression that matches the desired string
- **substitution**: path to the actual URL

- **flags**: optional parameters that can modify the rule

Web applications often make use of *query strings*, which are appended to a URL using the `?` question mark and delimited using the & ampersand. These are ignored when matching rewrite rules. However, sometimes query strings may be required for passing data between pages. For example, a search result page written in PHP may utilize something akin to the following:

```
http://example.com/results.php?item=shirt&season=summer
```

In this example, we would like to simplify this to become:

```
http://example.com/shirt/summer
```

Example 1A: Simple Replacement

Using a rewrite rule, we could use the following:

```
/var/www/html/.htaccess
RewriteRule ^shirt/summer$ results.php?item=shirt&season=summer
```

The above is fairly self-explanatory, as it actually maps `shirt/summer` to `results.php?item=shirt&season=summer`. This achieves our desired effect.

Example 1B: Matching Options

However, we would like to generalize this to include all seasons. So, we will do the following:

- Specify a series of options using the `|` boolean, meaning "OR"
- Group the match using `()`, then reference the group using `$1`, with `1` for the first matched group

The Rewrite Rule now becomes:

```
/var/www/html/.htaccess
RewriteRule ^shirt/(summer|winter|fall|spring) results.php?item=shirt&season=$1
```

The rule shown above matches a URL of `shirt/` followed by a specified season. That season is grouped using `()` and then referenced with the `$1` in the subsequent path. This means that, for example, that:

```
http://example.com/shirt/winter
```

becomes:

```
http://example.com/results.php?item=shirt&season=winter
```

This also achieves the desired effect.

Example 1C: Matching Character Sets

However, we would also like to specify any type of item, not just URLs at `/shirt`. So, we will do the following:

- Write a *regular expression* that matches all alphanumeric characters. The bracket expression `[]` matches any character inside of it, and the `+` matches any number of characters specified

in the brackets
- Group the match, and reference it with `$2` as the second variable in the file

```
/var/www/html/.htaccess
RewriteRule ^([A-Za-z0-9]+)/(summer|winter|fall|spring) results.php?
item=$1&season=$2
```

The above will convert, for example:

```
http://example.com/pants/summer
```

to:

```
http://example.com/results.php?item=pants&season=summer
```

Example 1D: Passing Query Strings

This section doesn't introduce any new concepts but addresses an issue that may come up. Using the above example, say we would like to redirect `http://example.com/pants/summer` but will pass an additional query string `?page=2`. We would like the following:

```
http://example.com/pants/summer?page=2
```

to map to:

```
http://example.com/results.php?item=pants&season=summer&page=2
```

If you were to attempt to access the above URL with our current settings, you would find that the query string `page=2` got lost. This is easily fixed using an additional `QSA` flag. Modify the rewrite rule to match the following, and the desired behavior will be achieved.

```
/var/www/html/.htaccess
RewriteRule ^([A-Za-z0-9]+)/(summer|winter|fall|spring) results.php?
item=$1&season=$2 [QSA]
```

Example 2: Adding Conditions with Logic

`RewriteCond` lets us add conditions to our rewrite rules. All `RewriteCond`s abide by the following format:

```
RewriteCond TestString Condition [Flags]
```

- **RewriteCond**: specifies the `RewriteCond` directive
- **TestString**: the string to test against
- **Condition**: the pattern to match
- **Flags**: optional parameters that may modify the condition

If a `RewriteCond` evaluates to true, the `RewriteRule` immediately following will be considered.

Example 2A: Default Page

In an imaginary administration panel, we may want to direct all malformed URLs back to the home page, instead of greeting users with a 404. Using a condition, we can check to see if the requested file exists.

```
/var/www/html/.htaccess
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^admin/(.*)$ /admin/home
```

This will redirect something like `/admin/blargh` to `/admin/home`.

With the above:

- `%{REQUEST_FILENAME}` is the string to check
- `!-f` uses the `!` not operator on the filename
- `RewriteRule` redirects all requests back to `/admin/home`

Note that a more syntactically and technically correct approach would be to define the 404 `ErrorDocument`.

```
/var/www/html/.htaccess
ErrorDocument 404 /error.html
```

Example 2B: IP Access Restriction

Although this can also achieved using other methods, a `RewriteCond` can be used to restrict access to one IP or a collection of IP addresses.

This example blocks traffic from everywhere **except** 12.34.56.789.

```
/var/www/html/.htaccess
RewriteCond %{REMOTE_ADDR} !^(12\.34\.56\.789)$
RewriteRule (.*) - [F,L]
```

This example is simply the negation of Example 3 from the old mod_rewrite article. The entire statement reads "if the address is *not* 12.34.56.789, do not allow access."

In short:

- `%{REMOTE_ADDR}` is the address string
- `!^(12\.34\.56\.789)$` escapes all `.` periods with a `\` backslash and negates the IP address using `!`
- The `F` flag forbids access, and the `L` flag indicates that this is the last rule to run, if executed

If you'd rather **block** 12.34.56.789, use this instead:

```
/var/www/html/.htaccess
RewriteCond %{REMOTE_ADDR} ^(12\.34\.56\.789)$
RewriteRule (.*) - [F,L]
```