How to securely upload files with PHP?

5 min read · Last change October 26, 2017

Uploading files in PHP is achieved with the move_uploaded_file() function.

The HTML form for uploading single or multiple files must include the `enctype="multipart/form-data"` attribute. Use the POST method:

```
<form method="post" enctype="multipart/form-data" action="upload.php"> File: <input type="file"
name="pictures[]" multiple="true"> <input type="submit"> </form>
```

And the PHP `upload.php` script looks like the following:

```
<?php foreach ($_FILES['pictures']['error'] as $key => $error) { if ($error == UPLOAD_ERR_OK) {
$tmpName = $_FILES['pictures']['tmp_name'][$key]; // basename() may prevent directory traversal
attacks, but further // validations are required $name = basename($_FILES['pictures']['name']
[$key]); move_uploaded_file($tmpName, "/var/www/project/uploads/$name"); } }
```

Don't stop here just yet and **continue reading**! The uploaded files must be validated for security purposes. A lot of hacks can occur when uploading hasn't been properly secured. Imagine a malicious attacker uploads `evil.php` which is publicly accessible over `https://example.com/uploads/evil.php`!

Validation

Always make sure that you implement server-side validation in order to be able to upload securely, and make sure that you understand the reasons for this, and the security vulnerabilities that you would otherwise be exposed to.

Directory traversal

To avoid directory traversal (a.k.a. path traversal) attacks, use `basename()` like shown above, or even better, rename the file completely like in the next step.

Rename uploaded files

Renaming uploaded files avoids duplicate names in your upload destination, and also helps to prevent directory traversal attacks. If you need to keep the original filename, you can it in a database for retrieval in the future. As an example, renaming a file with `microtime()` and some random number:

```
$uploadedName = $_FILES['upload']['name']; $ext = strtolower(substr($uploadedName,
strripos($uploadedName, '.')+1)); $filename = round(microtime(true)).mt_rand().'.'.$ext;
```

You can also use hashing functions like `hash_file()` and `sha1_file()` to build filenames. This method can save some storage space when different users upload the same file.

```
$uploadedName = $_FILES['upload']['name']; $ext = strtolower(substr($uploadedName,
strripos($uploadedName, '.')+1)); $filename = hash_file('sha256', $uploadedName) . '.' . $ext;
```

Check file type

Instead of relying on file extensions, you can get the mime-type of a file with finfo_file():

```
$finfo = finfo_open(FILEINFO_MIME_TYPE); // return mime-type extension echo finfo_file($finfo,
$filename); finfo_close($finfo);
```

For images, a check that's more reliable, but still not really good enough is using the `getimagesize()` function:

```
$size = @getimagesize($filename); if (empty($size) || ($size[0] === 0) || ($size[1] === 0)) {
throw new \Exception('Image size is not set.'); }
```

Check file size

To limit or check the size of the uploaded file, you can check `$_FILES['files']['size']` and the errors `UPLOAD_ERR_INI_SIZE` and `UPLOAD_ERR_FORM_SIZE`:

```
if ($_FILES['pictures']['size'] > 1000000) { throw new RuntimeException('Exceeded filesize
limit.'); }
```

Storing uploads to a private location

Instead of saving uploaded files to a public location available at `https://example.com/uploads`, storing them in a publicly inaccessible folder is a good practice. To deliver these files, so called proxy scripts are used.

Client-side validation

For better user experience, HTML offers the accept attribute to limit filetypes by the extension or mime-type in the HTML, so users can see the validation errors on the fly and select only allowed filetypes in their browser. However, browser support is limited at the time of writing this. Keep in mind that client-side validation can be easily bypassed by hackers. The server-side validation steps explained above are more important forms of validation to use.

Full example

Let's take all of the above into consideration and look at a very simple example:

```
// Check if we've uploaded a file if (!empty($_FILES['upload']) && $_FILES['upload']['error'] ==
UPLOAD_ERR_OK) { // Be sure we're dealing with an upload if (is_uploaded_file($_FILES['upload']
['tmp_file']) === false) { throw new \Exception('Error on upload: Invalid file definition'); } //
Rename the uploaded file $uploadName = $_FILES['upload']['name']; $ext =
strtolower(substr($uploadName, strripos($uploadName, '.')+1)); $filename =
round(microtime(true)).mt_rand().'.'.$ext; move_uploaded_file($_FILES['upload']['tmp_file'],
__DIR__.'../uploads/'.$filename); // Insert it into our tracking along with the original name }
```

Server configuration

The server-side validation mentioned above can be still bypassed by embedding custom code inside the image itself with tools like jhead, and the file might be ran and interpreted as PHP.

That's why enforcing filetypes should also be done at the server level.

Apache

Make sure Apache is not configured to interpret multiple files as the same (e.g., images being interpreted as PHP files). Use the ForceType directive to force the type on the uploaded files.

```
<FilesMatch "\.(?i:pdf)$"> ForceType application/octet-stream Header set Content-Disposition
attachment </FilesMatch>
```

Or in the case of images:

```
ForceType application/octet-stream <FilesMatch "(?i).jpe?g$"> ForceType image/jpeg </FilesMatch>
```

```
<FilesMatch "(?i).gif$"> ForceType image/gif </FilesMatch> <FilesMatch "(?i).png$"> ForceType
image/png </FilesMatch>
```

Nginx

On Nginx, you can use the rewrite rules, or use the `mime.types` configuration file provided by default.

```
location ~* (.*\.pdf) { types { application/octet-stream .pdf; } default_type application/octet-
stream; }
```

See also

- How to securely allow users to upload files
- PHP image upload security: How not to do it
- Related FAQ: How to increase the file upload size in PHP?
- PHP Manual: Handling file uploads
- brandonsavage/Upload - standalone PHP upload component with validation and storage strategies.
- Uploading files with Laravel framework
- Uploading files with Symfony framework
- ralouphie/mimey - PHP package for converting file extensions to MIME types and vice versa.
- phpMussel/phpMussel - PHP-based anti-virus anti-trojan anti-malware solution capable of scanning file uploads and with some simple upload controls included.