



# PHP – Ficheros

---



**Ciclo:** DAW  
**Módulo:** DWES  
**Curso:** 2020-2021  
**Autor:** César Guijarro Rosaleny



<b>Introducción.....</b>	<b>3</b>
<b>Ficheros.....</b>	<b>4</b>
Abrir ficheros.....	4
Cerrar ficheros.....	7
Leer ficheros.....	8
Escribir en ficheros.....	11
Borrar ficheros.....	14
<b>Directorios.....</b>	<b>15</b>
Listar un directorio.....	15
<b>Subir ficheros al servidor.....</b>	<b>16</b>
<b>Seguridad en la subida de ficheros.....</b>	<b>21</b>
Medidas de seguridad.....	22
Comprobaciones básicas.....	23
Comprobación MIME Type.....	24
Comprobar archivos por extensión.....	25
Comprobar el header de las imágenes.....	27
Restringir la carpeta de subidas.....	29
<b>Bibliografía.....</b>	<b>30</b>

## Introducción

---

Las aplicaciones web normalmente trabajan con datos que están almacenados en un sistema gestor de bases de datos. Pero a veces, también trabajan con otras fuentes de datos, como por ejemplo ficheros. Por eso es importante conocer cómo se manejan los ficheros (como se crea, se escribe o se lee un fichero) y cómo se accede al sistema de ficheros (para copiar o borrar un fichero).

El lenguaje PHP dispone de una gran cantidad de funciones para realizar todo tipo de operaciones con archivos y carpetas, tanto básicas (crear, modificar, eliminar... archivos y carpetas) como otras más avanzadas (obtener y asignar permisos, crear enlaces simbólicos...).

En este tema, veremos como realizar algunas de las tareas que más usualmente se realizan sobre archivos y carpetas en PHP.

# Ficheros

---

PHP permite guardar y recuperar información a través de los ficheros, para lo cual dispone de funciones y procedimientos que crean, destruyen o modifican su contenido.

## Abrir ficheros

Para abrir un fichero en PHP usaremos la función **fopen()**. Esta función admite cuatro argumentos, de los cuales sólo los dos primeros (**filename** y **mode**) son obligatorios. Los otros dos (**\$use\_include\_path** y **\$context**) son opcionales.

El parámetro *filename* indica la ruta y el nombre del archivo que se quiere abrir. Si *filename* está en la forma “esquema:...”, se asume que será una URL y PHP buscará un gestor de protocolos (también conocido como envoltura) para ese protocolo.

Si PHP ha decidido que *filename* especifica un fichero local, intentará abrir un flujo para ese fichero. El fichero debe ser accesible para PHP, por lo que es necesario asegurarse que el fichero tiene los permisos adecuados.

El segundo parámetro indica el tipo de apertura del fichero. Puede ser cualquiera de los siguientes:

Modo de acceso	Descripción
'r'	Apertura para sólo lectura. Coloca el puntero al principio del fichero.
'r+'	Apertura para lectura y escritura. Coloca el puntero al principio del fichero.
'w'	Apertura para sólo escritura. Coloca el puntero al principio del fichero y trunca el fichero a longitud 0. Si no existe el archivo, intenta crearlo.
'w+'	Apertura para lectura y escritura. Coloca el puntero al principio del fichero y trunca el fichero a longitud 0. Si no existe el archivo, intenta crearlo.
'a'	Apertura para sólo escritura. Coloca el puntero al final del fichero. Si no existe el archivo, intenta crearlo.
'a+'	Apertura para lectura y escritura. Coloca el puntero al final del fichero. Si no existe el archivo, intenta crearlo.
'x'	Creación y apertura para sólo escritura. Coloca el puntero al principio del fichero. Si el fichero ya existe, la llamada fallará devolviendo <i>false</i> y generando un error. Si el fichero no existe, se intentará crear.
'x+'	Creación y apertura para escritura y lectura. De otra forma tiene el mismo comportamiento que 'x'.
'c'	Apertura para sólo escritura. Coloca el puntero al principio del fichero. Si el fichero no existe, se intentará crear. Si existe, no es truncado, ni la llamada a esta función falla.
'c+'	Apertura para lectura y escritura. De otra forma tiene el mismo comportamiento que 'c'.

Por ejemplo, si queremos abrir el fichero *fichero1.txt* que está en la carpeta *dws* en modo sólo lectura, tendríamos que escribir la siguiente sentencia:

```
fopen("/dws/fichero1.txt", "r");
```

Por el contrario, si lo queremos abrir para lectura y escritura:

```
fopen("/dws/fichero1.txt", "w+");
```

Si dejamos las sentencias anteriores como están, sin asignarlas a ninguna variable, no podríamos operar con los ficheros que hemos abierto. Para hacerlo, necesitamos asignar el resultado de la función **fopen()** a una variable que se convertirá en el **manejador del fichero**.

```
$archivo1 = fopen("/dws/fichero1.txt", "r");
```

De esta manera, *\$archivo1* se convierte en una referencia al fichero físico, con lo que ya podemos realizar operaciones sobre él (escribir en el fichero, borrar entradas, cerrarlo...)

## Cerrar ficheros

Para cerrar un fichero, usaremos la función **fclose()**, pasándole como parámetro la variable que contiene el manejador del archivo. Devuelve *true* en caso de éxito o *false* en caso de error.

```
fclose($archivo1);
```

Desde ese momento, ya no tendremos acceso al fichero *archivo1.txt*, con lo que si queremos volver a operar sobre él tendremos que abrirlo de nuevo y asignarlo a una variable para que se convierta en el nuevo manejador.

No es obligatorio cerrar un fichero cuando hemos acabado con él, aunque es recomendable, ya que si no mantendremos una variable apuntando al fichero que no vamos a utilizar nunca.

## Leer ficheros

PHP ofrece muchas formas de leer un archivo. Una de las más sencillas es la función **fread()**, que lee un fichero en modo binario seguro. Esta función admite dos parámetros. El primero es el manejador del fichero, y el segundo es el tamaño en bytes que queremos leer del archivo.

Para saber el tamaño en bytes que tiene el fichero, podemos utilizar la función **filesize()**, así podemos leer el fichero completo.

```
$nombreFichero = "/dws/fichero1.txt";  
$archivo1 = fopen($nombreFichero, "r");  
fread($archivo1, filesize($nombreFichero));
```

Vamos a ver un ejemplo. Crea un fichero en la carpeta *apache\_htdocs/DWS/ ejemplos/T2/ficheros* llamado *fichero1.txt* con el siguiente contenido:

```
Al Pacino  
Robert De Niro  
Robert Duvall  
Susan Sarandon  
Paul Newman  
Katharine Hepburn
```

Ahora crea un archivo llamado *leerFichero.php* y copia el siguiente código:

```
<?php  
  
$fileName = "./fichero1.txt";  
$file1 = fopen($fileName, "r");  
$fileContent = fread($file1, filesize($fileName));  
fclose($file1);  
echo $fileContent;
```



Lo que hacemos es abrir el fichero en modo lectura, leer su contenido y mostrarlo por pantalla.



Al Pacino Robert De Niro Robert Duvall Susan Sarandon Paul Newman Katharine Hepburn

Como puedes observar, **fread()** lee el fichero completo sin tener en cuenta los saltos de línea (EOL).

¿Y si queremos leer el fichero línea a línea? Para eso, tenemos otra función muy útil: **fgets()**. La diferencia es que **fgets()** lee el fichero hasta que encuentra un EOL.

Vamos a crear otro archivo llamado *leerFichero2.php* con el siguiente código:

```
<?php
$fileName = "./fichero1.txt";
$file1 = fopen($fileName, "r");
$fileContent = fgets($file1);
fclose($file1);
echo $fileContent;
```

Fíjate que a **fgets()** no le tenemos que pasar la longitud que queremos leer. Cuando encuentre un EOL, parará de leer el fichero. Si ejecutamos el archivo en un navegador, observamos que cambia la salida:



Ahora sólo ha leído la primera línea de nuestro fichero, ya que, como hemos comentado, para de leer el contenido cuando encuentra el primer EOL.

Si no encuentra más EOL, `fgets()` devolverá `FALSE`, con lo que combinándolo con un bucle `while`, por ejemplo, podemos leer el fichero completo línea a línea.

Modifica `leerFichero2.php` para incorporar el bucle y poder leer el fichero completo de la siguiente forma:

```
<?php

$fileName = "./fichero1.txt";
$file1 = fopen($fileName, "r");
while ($content = fgets($file1)) {
    echo $content;
}
fclose($file1);
```

Ahora ya podemos sacar todo el contenido del fichero por pantalla. Incluso podríamos añadir `<br>` para separar las líneas del fichero original:

```
<?php

$fileName = "./fichero1.txt";
$file1 = fopen($fileName, "r");
while ($content = fgets($file1)) {
    echo "$content<br>";
}
fclose($file1);
```



Al Pacino  
Robert De Niro  
Robert Duvall  
Susan Sarandon  
Paul Newman  
Katharine Hepburn

## Escribir en ficheros

Para escribir en un fichero, podemos usar la función **fwrite()**. El primer parámetro es el manejador del fichero donde queremos escribir. El segundo es el contenido que queremos añadir. Si se añade un tercer argumento (*length*), la escritura se detendrá después de que *length* bytes hayan sido escritos o se alcance el final de *string*, lo que suceda primero.

Por ejemplo, crea un fichero en la misma carpeta llamado *escribirFichero.php* y añade este código:

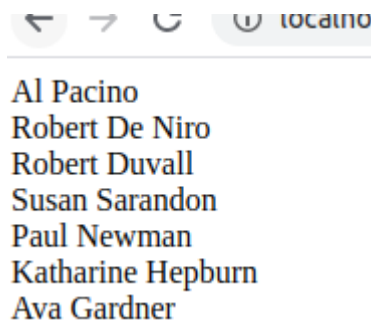
```
<?php
$fileName = "./fichero1.txt";
$file1 = fopen($fileName, "a+");
$content = "\nAva Gardner";
fwrite($file1, $content);
rewind($file1);
while ($content = fgets($file1)) {
    echo "$content<br>";
}
fclose($file1);
```

Hay varias cosas a comentar sobre el código anterior. Lo primero es que, si te das cuenta, hemos abierto el fichero con la opción **a+**, ya que lo queremos abrir para escritura y lectura. Si lo abriéramos con la opción **w+**, el fichero se borraría entero y después añadiría “Ava Gardner”.

En el contenido a añadir empezamos con **\n**, carácter que significa EOL. Si dejáramos “Ava Gardner” a secas, lo escribiría en la misma línea que la última cadena de texto de nuestro fichero.

La función **rewind()** rebobina la posición de un puntero a un fichero. Si no la utilizáramos, no veríamos nada por pantalla (aunque nuestra cadena se hubiera añadido correctamente al fichero), ya que el puntero después de escribir el nuevo contenido estará apuntando al final del fichero, con lo **fgets()** que no encontraría ningún texto.

Si ejecutamos el archivo, el navegador nos muestra el contenido del fichero con la nueva línea añadida:

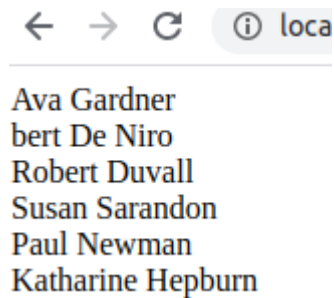


Al Pacino  
Robert De Niro  
Robert Duvall  
Susan Sarandon  
Paul Newman  
Katharine Hepburn  
Ava Gardner

¿Y si queremos añadir la línea al principio?. En principio la solución es sencilla: abrimos el fichero con la opción **c+** y cambiamos el EOL del nuevo texto al final.

```
<?php
$fileName = "./fichero1.txt";
$file1 = fopen($fileName, "c+");
$content = "Ava Gardner\n";
fwrite($file1, $content);
rewind($file1);
while ($content = fgets($file1)) {
    echo "$content<br>";
}
fclose($file1);
```

Pero si miramos la salida, vemos que no es lo que esperamos:



Ava Gardner  
bert De Niro  
Robert Duvall  
Susan Sarandon  
Paul Newman  
Katharine Hepburn

Lo que ha pasado es que el nuevo texto ha reemplazado el texto del fichero anterior. Si sabemos que cada carácter ocupa un byte, la cadena “Ava Gardner\n” ocupa 12 bytes, por lo tanto habrá reemplazado los 12 primeros caracteres de nuestro fichero “Al Pacino\nRo”.

Para solucionarlo tenemos varias alternativas. Por ejemplo, podemos crear un fichero intermedio e ir añadiendo las líneas que queramos insertando la nueva línea al principio (o en la posición que queramos).

```
<?php

$fileName1 = "./fichero1.txt";
$fileName2 = "./fichero2.txt";
$file1 = fopen($fileName1, "r");
$file2 = fopen($fileName2, "w+");
$content = "Ava Gardner\n";
fwrite($file2, $content);
while ($content = fgets($file1)) {
    fwrite($file2, $content);
}
fclose($file1);
rewind($file2);
while ($content = fgets($file2)) {
    echo "$content<br>";
}
fclose($file2);
```

## Borrar ficheros

Para borrar un fichero, usaremos la función **unlink()**. Si el fichero no existe o no se tiene permisos para borrar el archivo, **unlink()** devolverá *false*.

Por ejemplo, después de añadir la línea al principio de nuestro archivo, podemos usar la función para borrar el primer fichero.

```
<?php

$fileName1 = "./fichero1.txt";
$fileName2 = "./fichero2.txt";
$file1 = fopen($fileName1, "r");
$file2 = fopen($fileName2, "w+");
$content = "Ava Gardner\n";
fwrite($file2, $content);
while ($content = fgets($file1)) {
    fwrite($file2, $content);
}
fclose($file1);
rewind($file2);
while ($content = fgets($file2)) {
    echo "$content<br>";
}
fclose($file2);
unlink($fileName1);
```

En este caso, **unlink()** espera una ruta válida de fichero, no el manejador, por lo que tenemos que usar *\$fileName1* en lugar de *\$file1*.

## Directorios

---

PHP ofrece muchas funciones para manejar directorios, pero lo más probable es que en el mayoría de casos lo único que nos interese de un directorio es conocer los archivos que tiene. Una vez conocido ese dato, podremos construir rutas relativas a sus subdirectorios y listarlos, y así de forma sucesiva.

### Listar un directorio

Para listas los archivos de un directorio podemos usar la función **scandir()**. Esta función enumera los ficheros y directorios ubicados en la ruta especificada.

Por defecto el listado se ordena alfabéticamente en sentido ascendente. Si se usa el parámetro opcional **sorting\_order** con el valor **SCANDIR\_SORT\_DESCENDING** se ordenará en orden descendente. Si queremos que el directorio no esté ordenado, podemos usar el valor **SCANDIR\_SORT\_NONE**.

A partir de PHP 5.4.0 se puede usar 0 para el orden ascendente y 1 para el descendente.

```
<?php
$directorio = "/var/www/html";
$ficheros = scandir($directorio);
var_dump($ficheros);
```

## Subir ficheros al servidor

Para subir ficheros a un servidor, podemos utilizar un formulario que apunte a una página PHP para que ésta lo procese.

Para que ésto sea posible, debemos tener establecidas las siguientes directivas de configuración en el archivo **php.ini**:

- **file\_uploads**: (On/Off), permite que haya o no carga de archivos.
- **upload\_max\_filesize**: tamaño máximo del archivo que se puede subir.
- **upload\_tmp\_dir**: directorio temporal donde se guardarán los archivos subidos. Si no se especifica, PHP usará el directorio por defecto del sistema.
- **post\_max\_size**: tamaño máximo de los datos enviados por el método POST.

Podemos saber los valores que tenemos configurados en nuestra servidor Apache viendo la salida de la función **phpinfo()**, en la sección *Core*:

extension_dir	/usr/local/lib/php/extensions/no-debug-non-zts-20190902	/usr/local/lib/php/extensions/no-debug-non-zts-20190902
file_uploads	On	On
hard_timeout	2	2
unserialize_callback_func	no value	no value
upload_max_filesize	2M	2M
upload_tmp_dir	/tmp/	/tmp/
output_handler	no value	no value
post_max_size	8M	8M
precision	14	14



**NOTA:** Una vez configures la carpeta temporal donde se almacenarán los archivos subidos, revisa el tema de los permisos. Tienes que tener en cuenta que se tiene que poder escribir y leer archivos en ella.

Vamos a ver un ejemplo de como subir un archivo a nuestro servidor.

Crearemos dos archivos, el primero *formulario.html* tendrá el código HTML de nuestro formulario, y el segundo *subida.php* será el encargado de leer el fichero elegido y guardarlo en nuestra carpeta de subidas.

Guarda los dos ficheros en *apache\_htdocs/DWS/ejemplos/T2/ficheros2*.

El código de *formulario.html* puede ser algo parecido a ésto:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Subida de ficheros</title>
</head>
<body>
    <form enctype="multipart/form-data" action="/subida.php"
        method="POST">
        <input type="hidden" name="MAX_FILE_SIZE"
value="2000000">
        <label for="fichero">Elige un archivo</label>
        <input type="file" name="fichero">
        <input type="submit" value="Subir fichero">
    </form>
</body>
</html>
```

Añadimos el campo oculto **MAX\_FILE\_SIZE** (medido en bytes), que debe preceder al campo de entrada del fichero, siendo su valor el tamaño máximo aceptado por PHP. Hay que tener en cuenta que engañar a esta configuración en el lado del navegador es muy fácil, por lo que es conveniente fijar el tamaño máximo también en el servidor (mediante `upload_max_filesize`). En nuestro caso lo ponemos a 2 MB (2.000.000 Bytes).

El atributo **method** del formulario debe ser **POST**, y hay que asegurarse que el formulario de subida de ficheros tiene el atributo **enctype=multipart/form-data** o de lo contrario la subida de ficheros no funcionará.

El siguiente paso será crear nuestro fichero *subida.php*. Por ahora mostraremos el valor de la variable `$_FILES`:

```
<?php  
var_dump($_FILES);
```

El array global `$_FILES` contendrá toda la información de los ficheros subidos. Su contenido en el formulario del ejemplo anterior será el siguiente:

- **`$_FILES['fichero_usuario']['name']`**: Nombre original del fichero en la máquina del cliente.
- **`$_FILES['fichero_usuario']['type']`**: El tipo MIME del fichero, si el navegador proporcionó esta información. Un ejemplo sería *"image/gif"*. Este tipo MIME, sin embargo, no se comprueba en el lado de PHP, y por lo tanto no se garantiza su valor.

- `$_FILES['fichero_usuario']['size']`: El tamaño, en bytes, del fichero subido.
- `$_FILES['fichero_usuario']['tmp_name']`: El nombre temporal del fichero en el cual se almacena el fichero subido en el servidor.
- `$_FILES['fichero_usuario']['error']`: El código de error asociado a esta subida.

Ejecuta el ejemplo en un navegador y comprueba que se muestran los valores correctos en la pantalla:

```
/var/www/html/DWS/ejemplos/T2/ficheros2/subida.php:3:  
array (size=1)  
  'fichero' =>  
    array (size=5)  
      'name' => string 'jack.jpg' (length=8)  
      'type' => string 'image/jpeg' (length=10)  
      'tmp_name' => string '/tmp/phpm0qgrm' (length=14)  
      'error' => int 0  
      'size' => int 81537
```

Si los campos *type*, *tmp\_name* y *size* están vacíos, quiere decir que algo ha fallado. Comprueba los permisos de la carpeta temporal. También puede ser que el tamaño de la imagen sea demasiado grande.

Una vez subido el fichero, éste **será borrado del directorio temporal al final de la solicitud**, con lo que deberemos renombrarlo o moverlo a otra ubicación si no queremos perderlo.

Para moverlo a una nueva ubicación, podemos usar la función **`move_upload_file()`**. Esta función moverá el archivo a la carpeta que queramos con el nombre que elijamos. La función devolverá FALSE si algo falla.

Vamos a modificar *subida.php* para guardar la imagen en una carpeta de imágenes en nuestro servidor.

Crea una carpeta llamada *imagenes* en el directorio donde están nuestros dos archivos (acuérdate de darle los permisos adecuados para poder escribir en ella).

Ahora sustituye la línea `var_dump($_FILES)` por:

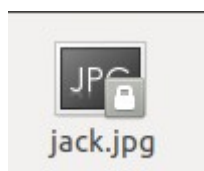
```
<?php

$name = $_FILES["fichero"]["name"];
if (move_uploaded_file($_FILES["fichero"]["tmp_name"],
"./imagenes/$name")) {
    echo "El archivo se ha subido con éxito";
} else {
    echo "Algo ha fallado en la subida del archivo";
}
```

Lo que hacemos es mover el archivo de la carpeta temporal a nuestra carpeta de imágenes con el nombre original del archivo.

**NOTA:** Ten en cuenta que si el archivo ya existe, `move_uploaded_file()` lo machacará, con lo que sería buena idea cambiar el nombre para asegurarnos que es único. Por ejemplo, si estamos subiendo imágenes para los avatar de los usuarios de nuestra web, podríamos renombrar esas imágenes a `avatarj#idUsuario.extension_archivo`.

Si todo ha ido bien, veremos el archivo en nuestra carpeta *imagenes*.



## Seguridad en la subida de ficheros

---

Incluir medidas de **seguridad al subir archivos a un servidor desde PHP** es muy importante, ya que puede ser usado por atacantes para subir archivos maliciosos.

Subir archivos a un servidor es algo que está presente en casi todas las aplicaciones web y apps que se usan hoy en día. Cuando se permite **subir archivos** a usuarios, se abre una **puerta de hackeo para posibles atacantes**. En muchos casos subir archivos es un requisito indispensable para el funcionamiento ciertas plataformas como por ejemplo redes sociales, blogs, foros, etc. En dichas plataformas los usuarios pueden subir imágenes, videos u otros tipos de documentos. Cuantos más tipos de archivos se permiten subir, mayor es el riesgo ante la posible subida de un archivo malicioso.

Tal y como está construido nuestro formulario, un atacante puede subir archivos PHP (o de cualquier otro lenguaje interpretable por el servidor) con código malicioso.

## Medidas de seguridad

La primera comprobación la podríamos hacer en el lado del cliente. Podríamos usar el atributo **accept** de la etiqueta `<input type=file>` para especificar el tipo de archivos que acepta el formulario (en nuestro caso imágenes).

De esta forma, al abrir la ventana de *seleccionar archivo* nos mostraría sólo los archivos de tipo imagen:

En realidad esta comprobación valdría de poco, ya que el atacante podría seleccionar como tipo de archivo *todos los archivos*:

También podríamos comprobar el tipo de archivo (o la extensión) mediante **javascript** (o cualquier otro lenguaje del lado del cliente). El problema es que esta validación es muy fácil de saltar. El atacante podría cambiar el nombre del fichero (*virus.jpg*, por ejemplo), capturar el paquete que se envía desde el navegador al servidor mediante alguna herramienta como *Burp Suite*, y volver a cambiarlo a *virus.php* antes de enviarlo al servidor.

Aunque las validaciones del lado del cliente son poco efectivas para alguien con mínimos conocimientos informáticos, no está de más utilizarlas, teniendo en cuenta que **la validación final la deberemos hacer siempre en nuestro servidor**.

## Comprobaciones básicas

Primero sería recomendable agregar al script PHP una comprobación de que el archivo ha sido subido antes de moverlo del directorio temporal al definitivo, esto se puede comprobar con la función de php **is\_uploaded\_file()**.

Si no se agrega esta verificación podría llegar a engañarse a la aplicación con lo que el archivo subido no se movería del directorio temporal al definitivo y podría permitir (dependiendo de la configuración del servidor) acceder al directorio temporal.

```
<?php

if (is_uploaded_file($_FILES["fichero"]["tmp_name"])) {
    $name = $_FILES["fichero"]["name"];
    if (move_uploaded_file($_FILES["fichero"]["tmp_name"],
"./imagenes/$name")) {
        echo "El archivo se ha subido con éxito";
    } else {
        echo "Algo ha fallado en la subida del archivo";
    }
} else {
    echo "Posible ataque";
}
```

## Comprobación MIME Type

Cuando un archivo es subido al servidor, PHP establece el elemento del array `$_FILES['uploadfile']['type']` en función del MIME Type proporcionado por el navegador web que el cliente este usando.

Es un error validar un archivo dependiendo sólo de este valor. **Un atacante puede enviar peticiones HTTP de forma automática adjuntando un archivo con su MIME Type falsificado.**

```
<?php

$imagenesPermitidas = ["image/jpeg", "image/png"];

if (in_array($_FILES["fichero"]["type"], $imagenesPermitidas)) {
    $name = $_FILES["fichero"]["name"];
    move_uploaded_file($_FILES["fichero"]["tmp_name"],
    "./imagenes/$name");
    echo "Imagen subida";
} else {
    echo "Archivo no permitido";
}
```



## Comprobar archivos por extensión

Un paso adicional de seguridad sería **crear una lista negra con las extensiones** de archivos que no se quieran permitir subir. De esta forma si un archivo contiene una extensión de la lista negra denegar su subida.

Un inconveniente de esta comprobación es que es casi imposible tener una lista completa de todas las posibles extensiones que un atacante puede usar. Si la aplicación funciona en un entorno de hosting, generalmente interpretan una gran cantidad de lenguajes (Ruby, Python, Perl, etc) y la lista de extensiones a denegar se puede extender demasiado.

**Apache, por ejemplo, puede manejar archivos de más de una extensión,** siendo el orden de las extensiones irrelevante. Si existe más de una extensión e indican el mismo tipo de meta-información, se usará el de la derecha, excepto para lenguajes y codificación de contenidos. Por ejemplo, si *.jpg* indica *image/jpeg* y *.html* indica que es *text/html*, entonces el archivo *ejemplo.jpg.html* se asocia con el MIME Type *text/html*.

Por lo tanto, un archivo con nombre *ejemplo.php.abcd* sería interpretado como PHP pudiendo llegar a ejecutarse. Esto solo funciona si la segunda extensión (*abcd*) no corresponde a ninguna especificada en la lista de mime-types conocidos por el servidor. Mediante esta técnica el atacante podría subir un *virus.php.abcd* sin que este archivo fuera denegado por la black list creada en la medida de seguridad anterior al no estar incluida la extensión “*abcd*” en ella, y por lo tanto ser ejecutado por el servidor.

El número de **extensiones dobles puede llegar a ser infinito**, lo que hace imposible poder crear una black list con todas las extensiones a denegar. Por ello, **lo mejor es realizar el proceso contrario y crear una lista blanca** en la que se define una lista de extensiones permitidas.

Para sacar la extensión, podemos utilizar la función **pathinfo()**, la cual nos devuelve información acerca del fichero en forma de array.

```
<?php

$imagenesPermitidas = ["jpeg", "jpg", "png"];

$partes_ruta = pathinfo($_FILES["fichero"]["name"]);
$extension = $partes_ruta['extension'];

if (in_array($extension, $imagenesPermitidas)) {
    $name = $_FILES["fichero"]["name"];
    move_uploaded_file($_FILES["fichero"]["tmp_name"],
    "./imagenes/$name");
    echo "Imagen subida";
} else {
    echo "Archivo no permitido";
}
```

A pesar de contar con esta validación, puede no servir de nada según como este la configuración de Apache.

Si se usa la directiva **AddHandler**, todos los archivos con la extensión *.php*, incluidas las dobles como *.php.jpg*, se ejecutarán como script PHP. Un atacante puede subir el fichero *virus.php.jpg*, saltarse la validación y ejecutar el código.

## Comprobar el header de las imágenes

Si solamente se permite la subida de imágenes se suele comprobar el header de la imagen mediante la función **getimagesize()**, la cual devuelve un array con las dimensiones y el tipo de la imagen, entre otras cosas. Si el header es incorrecto, la función devuelve false. Por lo tanto, si un atacante intenta subir un script PHP camuflado en un archivo *.jpg* la función devolverá false.

```
<?php

$tamanoImagen = getimagesize($_FILES["fichero"]["tmp_name"]);

if(!$tamanoImagen) {
    echo "Archivo no permitido";
} else {
    $name = $_FILES["fichero"]["name"];
    move_uploaded_file($_FILES["fichero"]["tmp_name"],
        "./imagenes/$name");
    echo "Imagen subida";
}
```

Otra posible función de PHP a usar es **exif\_imagetype()**. Se le pasa el nombre del archivo por parámetro y devuelve una constante que puede ser:

- IMAGETYPE\_GIF
- IMAGETYPE\_JPEG
- IMAGETYPE\_PNG
- IMAGETYPE\_SWF
- IMAGETYPE\_PSD
- IMAGETYPE\_BMP
- IMAGETYPE\_TIFF\_II (intel byte order)
- IMAGETYPE\_TIFF\_MM (motorola byte order)
- IMAGETYPE\_JPC
- IMAGETYPE\_JP2
- IMAGETYPE\_JPX
- IMAGETYPE\_JB2
- IMAGETYPE\_SWC
- IMAGETYPE\_IFF
- IMAGETYPE\_WBMP

Se puede comparar el resultado devuelto por la función con la constante que se espera recibir. De esta forma es muy fácil comprobar si el archivo subido es una imagen o no.

Esta medida puede ser fácilmente saltada editando el archivo mediante un **editor de imágenes como por ejemplo Gimp**, y **editando el comentario de la imagen donde se puede insertar el código PHP**. La aplicación verificará que efectivamente el archivo es una imagen pero al tener extensión PHP, el servidor interpretará el código del comentario de la imagen y tomará el resto del archivo como código basura sin llegar a tenerlo en cuenta. De esta forma la imagen mantendrá el header.

## **Restringir la carpeta de subidas**

Otra medida adicional sería proteger el directorio en el que se suben los archivos. La finalidad es restringir la ejecución de archivos desde dicha carpeta.

## Bibliografía

---

<http://php.net>