Foro de elhacker.net

Seguridad Informática => Nivel Web => Mensaje iniciado por: SH4V en 11 Diciembre 2009, 07:30

Título: **PHP upload security**
Publicado por: **SH4V** en **11 Diciembre 2009, 07:30**

En este artículo vamos a ver como programar aplicaciones seguras en PHP para la subida de archivos. Este paper está basado en el publicado por *Alla Bezroutchko* de *"SCANIT the Security Company"* en el año 2007. Como siempre, trataré de adaptarlo para que los más neófitos en la programación en PHP lo entiendan sin problemas. No obstante, cualquier problema podéis postearlo y será atendido.

La subida de archivos en la web es algo que está a la orden del día, no sólo en aplicaciones webmail para adjuntar archivos, sino también en servidores de alojamiento de imágenes, documentos de texto, pdf's, videos, etc. Para que una aplicación PHP sea segura, tenemos que atender siempre a las variables de entrada (y en ocasiones, también a las de salida!). A lo largo del paper iremos viendo diferentes aplicaciones PHP con filtros para evitar el upload de archivos dañinos y sus correspondientes bypasses. Finalmente se mostrará una forma segura de programar una aplicación no vulnerable al upload de archivos así como distintas medidas de seguridad alternativas. Si a tí, estimado lector, se te ocurre alguna forma mejor ó consideras que las expuestas aquí no son lo suficiente seguras, me encantaría que compartieras con todos nosotros tu forma de securizar las aplicaciones.

0.- Breves consideraciones:

Cuando utilizamos una aplicación PHP para subir un archivo, se crea un array superglobal que se almacena en la variable global _FILES. Antíguamente se utilizaba HTTP_POST_VARS, que aunque sigue estando disponible, ha ha caído para muchos programadores en el desuso en pro de la anterior. El array que se crea es similar a este:

Código:
```
Array
(
    [attachment] => Array
        (
            [name] => imagen.jpeg //Nombre del archivo
            [type] => image/jpeg //Tipo de archivo
            [tmp_name] => /tmp //Directorio temporal en el que se almacenará el archivo
            [error] => 0 //tipo de error
            [size] => 1234 //tamaño del archivo
        )
)
```

Y es que cuando subimos un archivo a un servidor, primero se almacena en un directorio temporal y posteriormente la aplicación lo mueve al directorio que había previsto el programador para ser almacenado. De todos estos valores, tenemos que tener cuidado con name y type porque son los únicos que tienen como origen el propio usuario ya que los demás son asignados por el servidor. Estos valores son por tanto variables de entrada que deben ser atendidas con especial atención si no queremos que nos cuelen una shell.php.

1.- Aplicación básica:

Vamos a ver una aplicación muy elemental que carece de medidas de seguridad. A partir de ahora, todas estas pruebas las puedes hacer tú en tu servidor local. Tan sólo crea una carpeta llamada *"images"* en el directorio raíz del servidor y asignale permisos necesarios para el usuario *Nobody*, que es el ocupado por Apache.

Código:
```
chmod -R 777
```

Veréis que utilizo la función *is_uploaded_file()* para verificar si el archivo ha sido subido. La explicación es que es necesario saber si se ha subido el archivo antes de moverlo a su directorio final porque si no se verifica y el archivo ha sido subido, podría engañarse a la aplicación spoofeando el formulario y cambiando el name='button' por name='foo' por ejemplo. Esto no movería el archivo de su directorio temporal a su directorio final, lo que nos podría permitir (dependiendo de la configuración del *httpd.conf*) acceder al directorio temporal. Nuestro servidor estará seguro si para los parámetros **<directory />** tenemos *Options FollowSymLinks* y *AllowOverride None*. Importante no tener la opción *Indexes* habilitada para evitar que se expongan directorios que puedan resultar sensibles por unas razones u otras.

Código:
```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

Para probar, sólo tenéis que guardar el código que tenemos a continuación como **upload.php** y subirlo al directorio raíz de vuestro servidor.

Código

```
1.  <html>
2.  <head>
3.  <meta http-equiv='Content-type' Content='charset UTF-8'>
4.  <title>Aplicación básica</title>
5.  <body>
6.  <center>
7.  <h2>Uploader de imágenes</h2>
8.  <FORM method="POST" ENCTYPE="multipart/form-data" action='upload.php'>
9.  <INPUT type='file' name='archivo'>
10. <INPUT type='submit' name='button' value='Enviar'>
11. </FORM>
12. <?php
13. $dir= 'images/';
14. if (isset ($_POST['button']) && is_uploaded_file($_FILES['archivo']['tmp_name'])){
15. $file= $dir.basename($_FILES['archivo']['name']);
16. move_uploaded_file($_FILES['archivo']['tmp_name'], $file);
17. }
18. ?>
19. </body>
20. </html>
```

Veamos la comunicación que ha tenido lugar entre nuestro navegador y el servidor:

Código:
```
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux i686; es-ES; rv:1.9.1.5) Gecko/20091105 Fedora/3.5.5-1.fc11 Firefox/3.5.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
```

```
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost/upload.php
Content-Type: multipart/form-data; boundary=-----------------------------102279127385402989103647552 5
Content-Length: 381

-----------------------------102279127385402989103647552 5\r\n
Content-Disposition: form-data; name="archivo"; filename="shell.php"\r\n
Content-Type: application/octet-stream\r\n
\r\n
<?php\n
system($_GET['cmd']);\n
?>\n
\r\n
-----------------------------102279127385402989103647552 5\r\n
Content-Disposition: form-data; name="button"\r\n
\r\n
Enviar\r\n
-----------------------------102279127385402989103647552 5--\r\n
```

Bypassing básico:

¿Bypassing? No hay nada qué bypassear. Sólo sube tu shell.php y listo.

2.- Verificación por Content-Type:

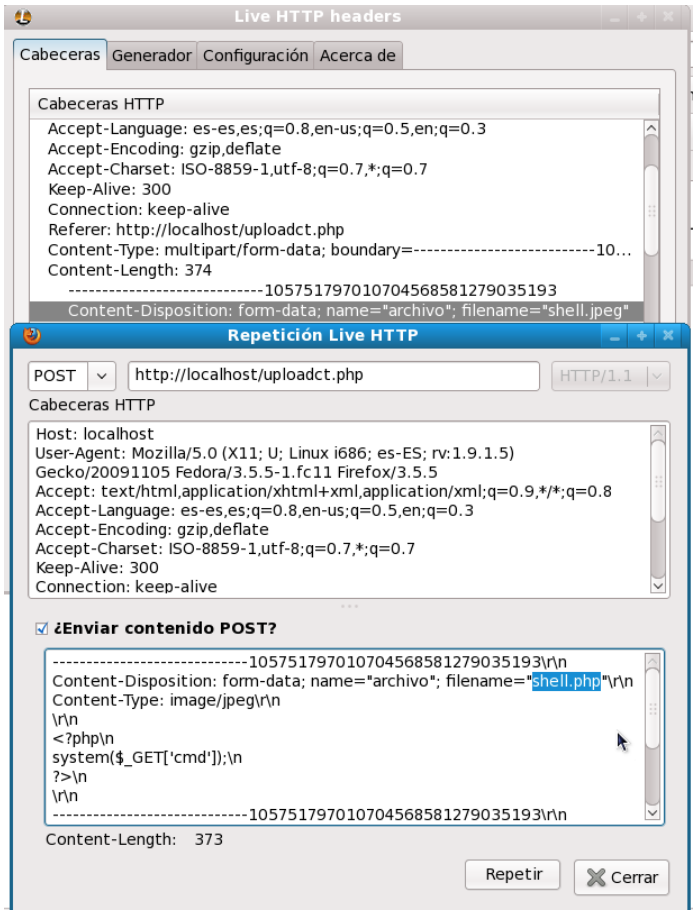
El protocolo HTTP, como muchos otros protocolos de red, utiliza una serie de variables en sus cabeceras a las que asigna un valor. Content-Type es una de ellas e indica el tipo de contenido del recurso. Una medida de seguridad podría ser verificar el Content-Type de la cabecera del paquete para cerciorarnos de que efectivamente se está tratando de subir una imagen. Supongamos una aplicación **uploadct.php** que sólo permita subir imágenes con extensión *.gif y *.jpeg:

Código

```
1. <html>
2. <head>
3. <meta http-equiv='Content-type' Content='charset UTF-8'>
4. <title>Aplicación por Content-Type</title>
5. <body>
6. <center>
7. <h2>Uploader de imágenes</h2>
8. <FORM method="POST" ENCTYPE="multipart/form-data" action='uploadct.php'>
9. <INPUT type='file' name='archivo'>
10. <INPUT type='submit' name='button' value='Enviar'>
11. </FORM>
12. <?php
13. $dir= 'images/';
14. if (isset ($_POST['button']) && is_uploaded_file($_FILES['archivo']['tmp_name'])){
15. if ($_FILES['archivo']['type'] != "image/gif" && $_FILES['archivo']['type'] != "image/jpeg"){
16. echo "<script>alert('Archivo no permitido!')</script>";
17. }else{
18. $file= $dir.basename($_FILES['archivo']['name']);
19. move_uploaded_file($_FILES['archivo']['tmp_name'], $file);
20. }
21. }
22. ?>
23. </body>
24. </html>
```

Bypassing Content-Type:

El bypassing de este mecanismo de seguridad es muy sencillo. Basta con coger nuestra shell.php y renombrarla a shell.jpg. De esta forma el servidor verá que _FILES['archivo']['type']=='image/jpeg' por lo que dejará subir la aplicación. Pero ¿qué hacemos con esto? ¿de qué nos sirve subir una shell con extensión de imagen?. Ahí está la gracia del bypass. Este es uno de los ataques más comunes a formularios de subida de imágenes ó al menos, es del que he visto más videotutos por la red. La ídea es modificar la cabecera del paquete, dejando tal cual el Content-Type pero modificando la extensión del archivo que estamos subiendo. Podéis ver un ejemplo de cómo hacerlo con Live-HTTP-Headers:



De tal manera, la extensión de nuestro archivo es .php, por lo que nuestro script podrá ser ejecutado sin ayuda de ningún LFI. Esto lo habremos conseguido como ya he dicho, bypasseando el filtro _FILES['archivo']['type'] vía modificación del apartado (Content-Type) de la cabecera del paquete HTTP.

2.- Verificación por getimagesize() ó exif_imagetype():

PHP incluye una función llamada **getimagesize()** que nos revela datos informativos sobre un archivo de imagen. Por el contrario de lo que pueda parecer a simple vista, no sólo nos revela el tamaño de la imagen, también nos revela infomación sobre el tipo de imagen vía parámetro: *imageinfo*. Al utilizar esta función, se genera el siguiente array:

Código:

```
Array[0] = Width
Array[1] = Height
Array[2] = Image Type Flag
Array[3] = width="xxx" height="xxx"
Array['bits'] = bits
Array['channels'] = channels
Array['mime'] = mime-type
```

De aquí nos interesa ['mime'], que nos dirá el tipo de imagen que es. ¿Qué ocurre si intentamos subir una shell.jpg? Sencíllamente, verificando con getimagesize() el índice del array 'mime' no nos dejará subir ningún archivo **que no tenga la estructura propia de una imagen**. Un ejemplo de uploading sería el siguiente:

Código

```
1. <html>
2. <head>
3. <meta http-equiv='Content-type' Content='charset UTF-8'>
4. <title>Aplicación getimagesize</title>
5. <body>
6. <center>
7. <h2>Uploader de imágenes</h2>
8. <FORM method="POST" ENCTYPE="multipart/form-data" action='uploadgis.php'>
9. <INPUT type='file' name='archivo'>
10. <INPUT type='submit' name='button' value='Enviar'>
11. </FORM>
12. <?php
13. $dir= 'images/';
14. if (isset ($_POST['button']) && is_uploaded_file($_FILES['archivo']['tmp_name'])){
15. $imaginf= getimagesize($_FILES['archivo']['tmp_name']);
16. if ($imaginf['mime'] != "image/gif" && $imaginf['mime'] != "image/jpeg"){
17. echo "<script>alert('Archivo no permitido!')</script>";
18. }else{
19. $file= $dir.basename($_FILES['archivo']['name']);
20. move_uploaded_file($_FILES['archivo']['tmp_name'], $file);
21. }
22. }
23. ?>
24. </body>
25. </html>
```

Otra función que PHP incorpora es **exif_imagetype()**. Al pasarle un input, genera una constante que puede ser del siguiente tipo:

Código:

```
1  IMAGETYPE_GIF
2  IMAGETYPE_JPEG
3  IMAGETYPE_PNG
4  IMAGETYPE_SWF
5  IMAGETYPE_PSD
6  IMAGETYPE_BMP
7  IMAGETYPE_TIFF_II (intel byte order)
8  IMAGETYPE_TIFF_MM (motorola byte order)
9  IMAGETYPE_JPC
10 IMAGETYPE_JP2
11 IMAGETYPE_JPX
12 IMAGETYPE_JB2
13 IMAGETYPE_SWC
14 IMAGETYPE_IFF
15 IMAGETYPE_WBMP
16 IMAGETYPE_XBM
```

Comparando el resultado con la constante, es muy fácil saber si el archivo que se está subiendo es una imagen o por el contrario no:

Código

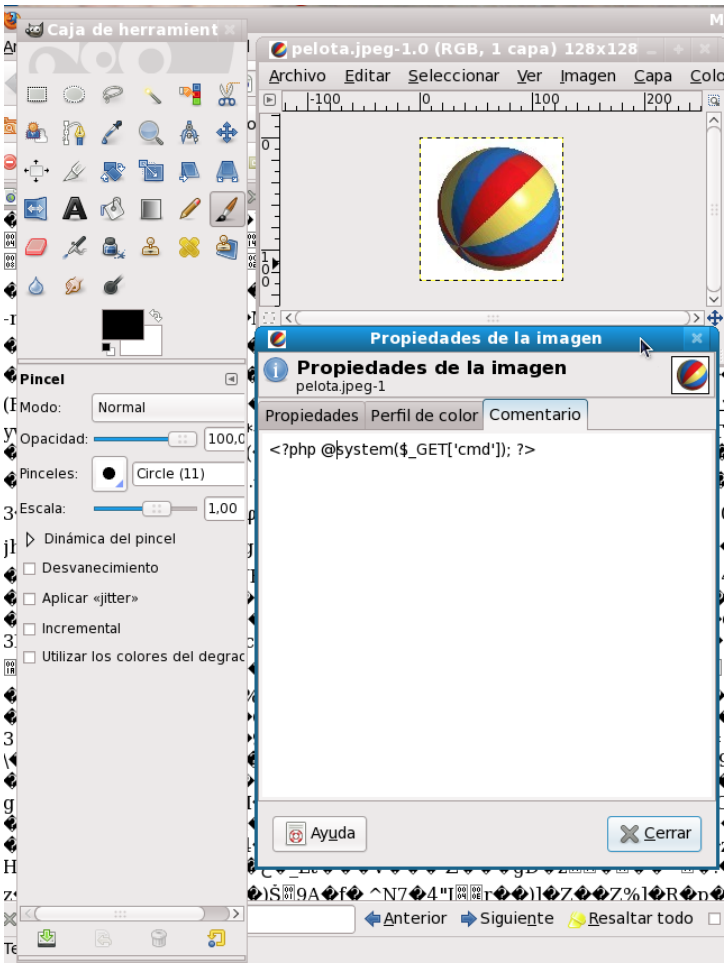
```
<html>
<head>
<meta http-equiv='Content-type' Content='charset UTF-8'>
<title>Aplicación getimagesize</title>
<body>
<center>
<h2>Uploader de imágenes</h2>
<FORM method="POST" ENCTYPE="multipart/form-data" action='uploadeit.php'>
<INPUT type='file' name='archivo'>
<INPUT type='submit' name='button' value='Enviar'>
</FORM>
<?php
$dir= 'images/';
if (isset ($_POST['button']) && is_uploaded_file($_FILES['archivo']['tmp_name'])){
$imaginf= getimagesize($_FILES['archivo']['tmp_name']);
if (exif_imagetype($_FILES['archivo']['tmp_name']) != IMAGETYPE_GIF && exif_imagetype($_FILES['archivo']['tmp_name']) != IMAGETYPE_JPEG){
echo "<script>alert('Archivo no permitido!')</script>";
}else{
$file= $dir.basename($_FILES['archivo']['name']);
move_uploaded_file($_FILES['archivo']['tmp_name'], $file);
}
}
?>
</body>
</html>
```

Para hacer las pruebas en vuestro entorno local, tan sólo tenéis que renombrar los ejemplos como **uploadgis.php** y **uploadeit.php** respectivamente.

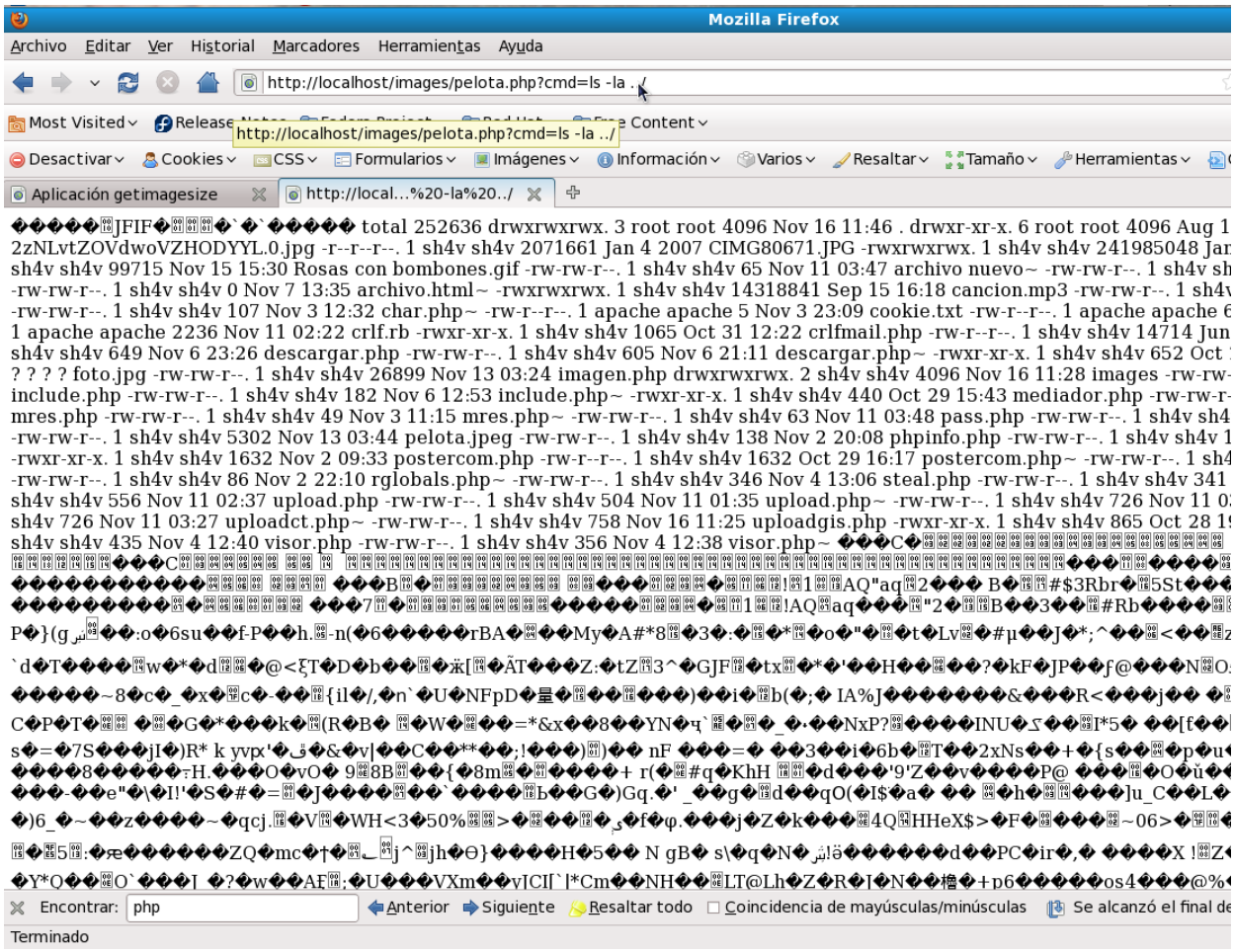
Bypassing getimagesize() y exif_imagetype():

Por el contrario de lo que pueda parecer, cualquiera de estas aplicaciones son fáciles de bypassear. Hace tiempo, Codebreak mostró una forma de infectar mediante LFI y la inserción de código PHP en los comentarios de las imágenes. De esta forma, con un simple LFI y un upload de imágenes en el servidor se podía ejecutar código PHP. Esto es perfectamente aplicable a este caso. La aplicación verificará que efectivamente se trata de una imagen pero al tener extension PHP, el servidor interpretará el código del comentario de la imagen y tomará la demás parte del archivo como basura, por lo que no lo tendrá en cuenta. Insertar comentarios en imágenes se puede hacer de muchas maneras, por comando type de windows, con un editor de imágenes como GIMP, etc. Veremos como hacerlo desde GIMP:

Abrimos la imagen, vamos a Imagen-->Propiedades de la imagen y en el apartado comentario insertamos el código PHP:



En el ejemplo del GIMP he utilizado una imagen *pelota.jpeg*. Para que se ejecute habrá que renombrarla a *pelota.jpeg* después de haber insertado el comentario. Una vez subida, podremos hacer cosas como esta:



Verificación por extensión:

La verificación por extension no es otra cosa que una verificación de tipo de archivo por extension del nombre del mismo. Existen diversas formas de verificar por archivo. Hace tiempo ya creó OzX un post en Undersecurity.net en el que varios usuarios fuimos poniendo nuestra propia forma de reconocer un archivo. Pongo a continuación los ejemplos y el autor:

Ozx:
Código

```
<?php

$testcase = array("sample.txt", "sample.jpg", "sample.case.txt");

function extension($filename){
    return substr(strrchr($filename, '.'), 1);
}

foreach($testcase as $test) {
    echo "Extension from $test is " . extension($test) . "\n";
}

?>
```

SH4V:
Código

```
<?php
$archivo=array("archivo.txt","archivo.jpeg","archivo.rb.txt",
"archivo.html","archivo.php.txt");

foreach ($archivo as $name){
$ext=explode(".",$name);
$num=count($ext)-1;
echo "Extensión para $name: $ext[$num]<br>";
}
?>
```

Seth:
Código

```
<?php

$testcase = array("sample.txt", "sample.jpg", "sample.case.txt.jpg");

function extension($filename){
```

```
    $path_parts = pathinfo($filename);
    return $path_parts['extension'];
}

foreach($testcase as $test) {
    echo "Extension from $test is " . extension($test) . "\n";
}

?>
```

C1c4Tr1Z:
Código

```
<?php
$files=array("example.php", "new_example.log", "third.txt", "something.asp", "another_thing.js");
foreach($files as $file){
        $extention;
        if((preg_match('/(?:[\.]+([a-z0-9]{2,3}))$/i', $file, $extention))){
                echo "$file -> {$extention[1]}\n";
        }
}
?>
```

WHK (elhacker.net):
Código

```
<?php
$files=array("example.php", "new_example.log", "third.txt", "something.asp", "another_thing.js", 'x..jpeg');
foreach($files as $file){
        $extention;
        if((preg_match('/(?:[\.]+([a-z0-9]{2,3}))$/i', $file, $extention))){
                echo "$file -> {$extention[1]}\n";
        }
}
?>
```

Bypassing por extensión de archivo:

Una verificación por extension evitaría subir en teoría subir código de riesgo a la carpeta de uploads. Sin embargo existen metodos para poder burlarla como los LFI (Local File Inclusion) que aprovechan las funciones:

Código

```
include()
include_once()
require()
require_once()
```

No entraré a explicar los LFI. Puedes encontrar un número elevado de textos si buscas por la red un poco.

Solución:

La mejor solución bajo mi punto de vista sería utilizar una mezcla de todas. Veamos un ejemplo de uploader seguro en el que se deberan de pasar varios filtros. Si se pasa el primer filtro, deberá pasarse el siguiente y así sucesivamente. Finalmente si se pasan todos los filtros la imagen se renombrará con un nombre aleatorio de 15 caracteres y su extensión correspondiente. Esta aplicación no nos protegería de una imagen con código PHP incrustado en los comentarios y extensión de imagen. Así que ¡¡**CUIDADO CON LOS LFI**!!

Código

```
<html>
<head>
<meta http-equiv='Content-type' Content='charset UTF-8'>
<title>Aplicación segura</title>
<body>
<center>
<h2>Uploader de imágenes</h2>
<FORM method="POST" ENCTYPE="multipart/form-data" action='uploadsec.php'>
<INPUT type='file' name='archivo'>
<INPUT type='submit' name='button' value='Enviar'>
</FORM>
<?php

$dir= 'images/';

if (isset ($_POST['button']) && is_uploaded_file($_FILES['archivo']['tmp_name'])){
$file= basename($_FILES['archivo']['name']);
        if ($_FILES['archivo']['type'] != "image/gif" && $_FILES['archivo']['type'] != "image/jpeg"){
        echo "<script>alert('Archivo no permitido! El archivo no es una imagen.')</script>";
        }elseif(exif_imagetype($_FILES['archivo']['tmp_name']) != IMAGETYPE_GIF && exif_imagetype($_FILES['archivo']['tmp_name']) != IMAGETYPE_JPEG){
        echo "<script>alert('Archivo no permitido! Se intentó subir un archivo envenenado.')</script>";
        }else{
        $name=$dir.substr(md5(rand()), 15, 30).extension($file);
        move_uploaded_file($_FILES['archivo']['tmp_name'], $name);
        }
}
?>
</body>
</html>
```

Un método propuesto por WHK, de elhacker.net, para la validación del nombre del archivo es este:

Código

```
<?php
$nombrearchivo = 'test../../&$%;,(xx)-_-(xx)[]"\\n.jpg';
$nombrearchivo = str_replace(array(
  "\x00", '*', '\\', '/', ':', '?', '¿', '<', '>', '|', '&', ',', ';'
), '', $nombrearchivo);
echo $nombrearchivo;
exit;
```

```
8. ?>
```

También sería oportuno asignar unos valores en el php.ini a las variables:

Código:

```
- upload_max_filesize
- post_max_size
- upload_tmp_dir
```

Para validar otros archivos que no sean imágenes, podemos usar fileinfo del repositorio PECL. Para ello tendremos que instalar *Pear*.Una vez instalado Pear, podemos instalar la librería de la siguiente manera:

Código:

```
pear install fileinfo
```

Después podemos hacerla correr añadiéndola como extensión al php.ini:

Código:

```
extension=fileinfo.so
```

ó en el propio script:

Código

```
1. <?php dl("fileinfo." . PHP_SHLIB_SUFFIX);?>
```

Para verificar el tipo de archivo se puede utilizar de dos maneras. La primera nos mostraría una descripción del tipo de archivo y la segunda sólo el tipo de archivo:

Código

```
1. <?php
2. finfo_file(finfo_open(), "documento.pdf");//muestra una descripción del tipo de archivo
3. finfo_file(finfo_open(FILEINFO_MIME), "documento.pdf");//muestra el tipo de archivo
4. ?>
```

Si alguien considera que esta aplicación no es segura del todo ó que hay alguna errata ó conoce alguna forma mejor de programarla, que postee en los comentarios su aporte y será añadio y se harán las correcciones oportunas. Toda crítica constructiva es bienvenida.

Gr33tz:

Agradecimientos a Pr0x, Protos, Lix, OzX, Yasión, Seth, Nork, S[e]C, N0b0dy, 1995, C1c4Tr1Z, WHK y a todos los miembros de Undersecurity y N3t-Datagrams.

Ejemplos:

http://n3t-datagrams.net/downloads/phpuploadsexamples.tar.gz

FUENTE:

http://n3t-datagrams.net/docs/?/=15

Saludos!

Título: **Re: PHP upload security**
Publicado por: **Azielito** en **11 Diciembre 2009, 17:46**

mas post de esos! :D

ñ_ñ'

Para los comentarios se me ocurre....

abrir el JPG como archivo de texto, y si se encuentran las cadenas

"<?" , "<?php" , "system(", "excec(" [y todas sus variantes xD] ",?>"
(aun que seguro basta con "<?" y "?>")
si encuentra alguna de esas cadenas no pasa el archivo y/o se eliminan :D

Por cierto, usa las etiquetas **[code=php]** para colorear el codigo :D

Título: **Re: PHP upload security**
Publicado por: **yasión** en **11 Diciembre 2009, 18:03**

Nice job SH4V! ;-) ;-)

Pues no se me ocurre nada más a mí... Cómo dices hay que vigilar los LFIs.

Título: **Re: PHP upload security**
Publicado por: **WHK** en **11 Diciembre 2009, 18:55**

Hola se ve super bueno.

En la función de C1c4Tr1Z te limita de dos a tres carácteres de extensión, yo le pondría 2 a 4 ya que dejas afuera las extensiones con 4 carácteres como el .jpeg o el tiff.

Código

```
1. <?php
2. $files=array("example.php", "new_example.log", "third.txt", "something.asp", "another_thing.js", 'x..jpeg');
3. foreach($files as $file){
```

```
4.      $extention;
5.      if((preg_match('/(?:[\.]+([a-z0-9]{2,3}))$/i', $file, $extention))){
6.          echo "$file -> {$extention[1]}\n";
7.      }
8.  }
9.  ?>
```

Yo preferiría utilizar preg_replace() para la eliminación de carácteres no soportados por el sistema operativo en el nombre de archivos y directorios tales como /*?<>%00 etc.

La función pathinfo() que utilizó Seth me gustó mas porque utilizas una función nativa de php ahorrando el recurso de utilizar dos o mas funciones como lo hizo ozx para verificar una extensión o haciendo explode que al final da casi lo mismo en rendimiento.

Yo pienso que si vas a renombrar el nombre del archivo de la imagen para guardarla entonces está demás la verificación de la extensión ya que las misma función de exif_imagetype() es el corazón encargado de verificar que el archivo sea realmente de tipo imagen y php elimina de forma automática todos los archivos subidos de esta manera (hablo de los temporales) asi que veo innecesario la eliminación.

Como dijeron en el documento, si puedes hacer bypass a exif_imagetype() con el comentario del jpg tal como lo hacían antiguamente con el editjpgcom entonces buscaría otra solución como por ejemplo la utilización de GD para obtener la imagen desde el archivo sin importar su extensión con imagecreatefromstring(file_get_contents($file)) y luego hacer un imagepng() de salida como nuevo archivo evitando la inserción de comentarios y strings que puedan servir de inyección a nuestro servidor web.

Con esto te evitas utiliza exif_imagetype() y el case o loop para verificar el tipo de archivo y le das una @ al comienzo de la función de GD por si la imagen cargada no es una imagen válida.
También hay formas de procesar transparencias y animaciones con GD.

Para procesar el nombre del archivo yo creo que es otro tema, puedes mantener el mismo o puedes cambiarselo por uno al azar, yo en lo personal preferiría mantenerselo filtrando posibles incompatibilidades de carácteres y posibles ataques como por ejemplo
Código

```
1.  <?php
2.  $nombrearchivo = 'test../../&$%;,(xx)-_-(xx)[]"\\n.jpg';
3.  $nombrearchivo = str_replace(array(
4.    "\x00", '"', '\\', '/', ':', '?', '¿', '<', '>', '|', '&', ',', ';'
5.  ), '', $nombrearchivo);
6.  echo $nombrearchivo;
7.  exit;
8.  ?>
```

Y con respecto a la extensión preferiría utilizar pathinfo() como dijo seth. De todas formas si tu servidor tiene LFI eso ya es otro tema porque ni si quiera necesitas subir un archivo para ejecutar código arbitrario, basta con infectar el log de acceso o cualqueir tipo de log al cual se tengaa acceso aunque sea de solo lectura.

Ahora, para las descargas es otro tema también super interesante para evitar la descarga arbitraria de archivos locales, etc.

Este es mi comentario y crítica constructiva solamente xD el documento está super bueno de todas formas, lo agregaré al post de recopilatorios.

saludos.

Título: **Re: PHP upload security**
Publicado por: **SH4V** en **12 Diciembre 2009, 09:02**

WoW! gracias WHKi

Ahora mismo edito el post. Sobre el unlink()del archivo después de terminarse el request tienes toda la razón, es absurdo ya que PHP lo elimina del directorio temporal y de la tabla de hashes.

Te añado a los greetz por tus aportes =)

Título: **Re: PHP upload security**
Publicado por: **wisehacks** en **12 Diciembre 2009, 09:43**

Muy bueno sh4van3. ¿Sabes si los framework de ASP.NET te implementan directamente este tipo de protecciones? Cuando tenga tiempo me pondré a jugar con ASP.NET y ya que te veo puesto en uploads pues te pregunto ::)

Título: **Re: PHP upload security**
Publicado por: **SH4V** en **12 Diciembre 2009, 09:53**

Actualizado!

Cita de: wisehacks en 12 Diciembre 2009, 09:43

> Muy bueno sh4van3. ¿Sabes si los framework de ASP.NET te implementan directamente este tipo de protecciones? Cuando tenga tiempo me pondré a jugar con ASP.NET y ya que te veo puesto en uploads pues te pregunto ::)

Pues no lo sé wisehacksi para serte sincero, a día de hoy no tengo ni idea de ASP-net. Lo siento! Sé en frameworks como RoR (ahí sí que me manejo algo más) sí que te brindan este tipo de protecciones aunque siempre quedan agujeros de seguridad.

Saludos!

Título: **Re: PHP upload security**
Publicado por: **ChElAnO** en **20 Diciembre 2009, 04:04**

muy bueno el post,

espero sigas realizando este tipo de material,

saludos!

ChElAnO

Título: **Re: PHP upload security**
Publicado por: **TRICKY** en **20 Diciembre 2009, 18:36**

Ala!

Que casualidad!

http://www.acunetix.com/websitesecurity/upload-forms-threat.htm

;D

/*** MOD ***/

imagino que los de Acunetix se habran basado en el mismo paper que el tuyo ?!

Muy buen aporte por cierto :_)

Título: **Re: PHP upload security**
Publicado por: **temporal12345** en **21 Diciembre 2009, 01:07**

bueno el post, creo que solo faltaría quitar permisos de ejecución al directorio dónde se suben los archivos, y error_reporting(0).

Título: **Re: PHP upload security**
Publicado por: **WHK** en **21 Diciembre 2009, 13:29**

Cita de: temporal12345 en 21 Diciembre 2009, 01:07

> bueno el post, creo que solo faltaría quitar permisos de ejecución al directorio dónde se suben los archivos, y error_reporting(0).

Y si por esas casualidades de la vida tienes un sistema de uploads donde tus archivos se guardan en un directorio igual a tu nick o te da la posibilidad de renombrar archivos? que pasaría si alguien le pone de nombre de archivo "../../../../shell.php" ? ya no estaría en el directorio donde no tienes permisos de ejecución y no necesitas un error_reporting a cero para evitar la subida de la shell tal como pasaba con el mod de my_uploads de phpnuke, inluso a veces para hacerles bypass podías subir archivos php con el nombre de x.php. o x.php5 o x.php... y se ejecutaban igual xD o si no le subias un .htaccess para forzar una extensión .ext a ejecutable php

Título: **Re: PHP upload security**
Publicado por: **O_G_T** en **23 Enero 2010, 07:13**

hola que tal a todos ;D
he pasado por este hilos que he encontrado googleando... he leido muchos temas publicados por sh4van3, algunos buenos otros no tantos debo suponer que se trata del mismo user SH4V de undersecurity.net ya que todos los paper de sh4van3 son una copia exapta de SH4V....

Primero sh4van3 como tu dices Alla Bezroutchko publico esto en el año 2007 y hoy es obsoleto... exiten muchas cosas que han cambiado desde esa fecha con relacion a php .... hay muchas funciones antiguas y nuevas de php para el tratamiento de imagenes; como asi tambien existen muchos trucos que pocos revelan, yo no voy a revelar los mios, porque de esta manera mantendre seguras mis aplicaciones.
Pero voy a mostrar algunas funciones que has descartado en tu codigo y que muchos dejan de lado a la hora de programar un upload de imagenes...

VAMOS AL HECHO:
exif_imagetype ---> determina el tipo de imagen
se puede utilizar en versiones PHP mayores a 4.3.0
ejemplo de uso
Código:

```
<?php
if (exif_imagetype('imagen_subida.gif') != IMAGETYPE_GIF) {
    echo 'Hey! la imagen no es GIF!!! estas usando el tuto de sh4van3';
}else{
///codigo normal
}
?>
```

exif_read_data --> en versiones superiores 4.3.0 (recomendado)
lee los encabezados EXIF de un archivo de imagen JPEG o TIFF. De esta manera se puede obtenet los metadatos generados por cámaras digitales utilizadas.

testeo
Código:

```
<?php
echo "test1.jpg:<br />\n";
$exif = exif_read_data('tests/test1.jpg', 'IFD0');
echo $exif===false ? "No header data found.<br />\n" : "Image contains headers<br />\n";

$exif = exif_read_data('tests/test2.jpg', 0, true);
echo "test2.jpg:<br />\n";
foreach ($exif as $key => $section) {
    foreach ($section as $name => $val) {
        echo "$key.$name: $val<br />\n";
    }
}
?>
```

exif_thumbnail
Código:

```
<?php
if (array_key_exists('file', $_REQUEST)) {
    $image = exif_thumbnail($_REQUEST['file'], $width, $height, $type);
} else {
    $image = false;
}
if ($image!==false) {
    header('Content-type: ' .image_type_to_mime_type($type));
    echo $image;
    exit;
} else {
    // no thumbnail available, handle the error here
    echo 'No thumbnail available';
}
?>
```

LOS SOBRESALIENTES:
imagecopyresampled() copia una porción rectangular de una imagen sobre otra, suavizando los valores de los píxeles mediante interpolación, de forma que al reducir

el tamaño de una imagen aún mantiene una buena claridad. img_dst es la imagen de destino, img_org es el identificador de la imagen de origen. Si las coordenadas de origen y destino y ancho y alto son diferentes, se encogerá o agrandará el fragmento de imagen según sea necesario. Las coordenadas son relativas a la esquina superior izquierda. Esta función se puede usar para copiar regiones dentro de la misma imagen (si img_dst es la misma que img_org) pero si las regiones se superponen, los resultados serán impredecibles.

imagecopyresized() copia una porción rectangular de una imagen hacia otra imagen. dst_im es la imagen de destino, src_im es el identificador de la imagen origen. Si la altura y anchura de las coordenadas de origen y destino difieren se realizará un estrechamiento o un estiramiento apropiado del fragmento de la imagen. Las coordenadas van localizadas sobre la esquina superior izquierda. Esta función se puede usar para copiar regiones dentro de la misma imagen (si dst_im es igual que src_im) pero si las regiones se solapan los resultados seran impredecibles.

Nota: Existe un problema debido a las limitaciones de las imagenes con paleta de colores (255+1 colores). La acción de volver a muestrear o filtrar una imagen, normalmente requiere de más de 255 colores, con los que se usa una aproximación para calcular el nuevo pixel muestreado y su color.
En las imagenes con paleta de color, se trata de obtener el nuevo color, o el color calculado más próximo en caso de error. Sin embargo, este nuevo color no siempre es el que visualmente es mas semejante. De esta forma, se pueden producir resultados extraños como imagenes vacías.
Para solventar este problema, se recomienda usar imagenes de color real como imagen destino, que se pueden obtener por ejemplo mediante la función **imagecreatetruecolor()**.

imagecreate Crea una nueva imagen con una paleta de colores

imagecreatefromgif Crear una nueva imagen a partir de un archivo o URL

imagecreatefromjpeg Crea una imagen nueva desde un archivo o URL
devuelve un identificador de imagen que representa a la imagen obtenida a partir del nombre de archivo indicado.
devuelve una cadena vacía si ha fallado. También escribe un mensaje de error, que desafortunadamente se muestra en el navegador como un enlace roto...

imagecreatefrompng() idem anteriores :-)

imagecreatetruecolor Crea una imagen nueva en color real (true color)
devuelve un identificador de imagen representando una imagen en blanco de tamaño anchura por altura

image_type_to_mime_type envia Mime-Type para image-type utiles para getimagesize, exif_read_data, exif_thumbnail, exif_imagetype.

ahora veamos un ejemplo utilizando todo esto :rolleyes:

EL SIGUIENTE CODIGO ES PARTE DE UNA APLICACION JQUERY, QUE CONSIDERO BUENA ;D
Código:

```
function resizeImage($image,$width,$height,$scale) {
	$image_data = getimagesize($image);
	$imageType = image_type_to_mime_type($image_data[2]);
	$newImageWidth = ceil($width * $scale);
	$newImageHeight = ceil($height * $scale);
	$newImage = imagecreatetruecolor($newImageWidth,$newImageHeight);
	switch($imageType) {
		case "image/gif":
			$source=imagecreatefromgif($image);
			break;
	    case "image/pjpeg":
		case "image/jpeg":
		case "image/jpg":
			$source=imagecreatefromjpeg($image);
			break;
	    case "image/png":
		case "image/x-png":
			$source=imagecreatefrompng($image);
			break;
	}
	imagecopyresampled($newImage,$source,0,0,0,0,$newImageWidth,$newImageHeight,$width,$height);

	switch($imageType) {
		case "image/gif":
			imagegif($newImage,$image);
			break;
	case "image/pjpeg":
		case "image/jpeg":
		case "image/jpg":
			imagejpeg($newImage,$image,90);
			break;
		case "image/png":
		case "image/x-png":
			imagepng($newImage,$image);
			break;
  }

	chmod($image, 0777);
	return $image;
}
function resizeThumbnailImage($thumb_image_name, $image, $width, $height, $start_width, $start_height, $scale){
	list($imagewidth, $imageheight, $imageType) = getimagesize($image);
	$imageType = image_type_to_mime_type($imageType);

	$newImageWidth = ceil($width * $scale);
	$newImageHeight = ceil($height * $scale);
	$newImage = imagecreatetruecolor($newImageWidth,$newImageHeight);
	switch($imageType) {
		case "image/gif":
			$source=imagecreatefromgif($image);
			break;
	    case "image/pjpeg":
		case "image/jpeg":
		case "image/jpg":
			$source=imagecreatefromjpeg($image);
			break;
	    case "image/png":
		case "image/x-png":
			$source=imagecreatefrompng($image);
			break;
	}
	imagecopyresampled($newImage,$source,0,0,$start_width,$start_height,$newImageWidth,$newImageHeight,$width,$height);
	switch($imageType) {
		case "image/gif":
			imagegif($newImage,$thumb_image_name);
			break;
	case "image/pjpeg":
		case "image/jpeg":
		case "image/jpg":
			imagejpeg($newImage,$thumb_image_name,90);
			break;
		case "image/png":
		case "image/x-png":
			imagepng($newImage,$thumb_image_name);
			break;
  }
```

```
        chmod($thumb_image_name, 0777);
        return $thumb_image_name;
}
        chmod($thumb_image_name, 0755);
        return $thumb_image_name;
}

function getHeight($image) {
        $size = getimagesize($image);
        $height = $size[1];
        return $height;
}

function getWidth($image) {
        $size = getimagesize($image);
        $width = $size[0];
        return $width;
}


$large_image_location = $upload_path.$large_image_name;
$thumb_image_location = $upload_path.$thumb_image_name;


if(!is_dir($upload_dir)){
        mkdir($upload_dir, 0755);
        chmod($upload_dir, 0755);
}
```

CODIGO DEL ARCHIVO QUE PROCESARA LOS DATOS
Código:

```
<?php

include'ARCHIVO-CON-EL-CODIGO-ANTERIOR.php';

if ($_POST["upload"]=="Upload") {
        //RECIBE DATOS
        $userfile_name = $_FILES['image']['name'];
        $userfile_tmp = $_FILES['image']['tmp_name'];
        $userfile_size = $_FILES['image']['size'];
        $userfile_type = $_FILES['image']['type'];
        $filename = basename($_FILES['image']['name']);
        $file_ext = strtolower(substr($filename, strrpos($filename, '.') + 1));


        if((!empty($_FILES["image"])) && ($_FILES['image']['error'] == 0)) {

                foreach ($allowed_image_types as $mime_type => $ext) {

                        if($file_ext==$ext && $userfile_type==$mime_type){
                                $error = "";
                                break;
                        }else{
                                $error = "Only <strong>".$image_ext."</strong> images accepted for upload<br />";
                        }
                }
                //VERIFICA EL TAMAÑO
                if ($userfile_size > ($max_file*1048576)) {
                        $error.= "Images must be under ".$max_file."MB in size";
                }

        }else{
                $error= "Please select an image for upload";
        }
        //SI TODO ESTA BIEN SUBE LA IMAGEN
        if (strlen($error)==0){

                if (isset($_FILES['image']['name'])){
/*VERIFICA LA EXTENSION QUE ES TOMADA POR UNA SESION ESTA PARTE DE LA APLICACION NO HA SIDO PUBLICADA EN ESTE EJEMPLO ESTE ES MERO EJEMPLO DE EMPLEO DE LAS FUNCIONES CITADAS EN ESTE PAPER*/
                        $large_image_location = $large_image_location."."$file_ext;
                        $thumb_image_location = $thumb_image_location."."$file_ext;

                        //UNA VEZ CARGADO EL ARCHIVO SE AGREGA LA EXTENSION TOMANDO DESDE SESION
                        if($_SESSION['user_file_ext']!=$file_ext){
                                $_SESSION['user_file_ext']="";
                                $_SESSION['user_file_ext']="."$file_ext;
                        }

                        move_uploaded_file($userfile_tmp, $large_image_location);
                        chmod($large_image_location, 0777);

                        $width = getWidth($large_image_location);
                        $height = getHeight($large_image_location);
                        //REDUCE LA IMAGEN SI ES MAYOR EN ANCHO QUE EL PERMITIDO
                        if ($width > $max_width){
                                $scale = $max_width/$width;
                                $uploaded = resizeImage($large_image_location,$width,$height,$scale);
                        }else{
                                $scale = 1;
                                $uploaded = resizeImage($large_image_location,$width,$height,$scale);
                        } ?>
```

YO CREO QUE ES BUEN CODIGO, PODEIS PROBAR METIENDO TODO EL CODIGO QUE QUERAIS EN LA PELOTITA DEL EJEMPLO DE sh4van3 **OS ASEGURO QUE PARA LO UNICO QUE SERVIRA LA PELOTITA ESA, ES PARA METERSELA EN EL<? echo $mi_nick; ?>** ;D ;D ;D ;D ;D
SALUDOS!!!

Título: **Re: PHP upload security**
Publicado por: **WHK** en **23 Enero 2010, 15:29**

Hola, creo que debiste haber puesto tu post en el foro de php y no acá porque lo único que veeo es como utilizar funciones descritas exactamente igual que en php.net

http://cl.php.net/exif
http://cl.php.net/image

Lo digo porque de seguridad no le veo mucho, incluso ese código está mal hecho porque en el caso que hagas multiupload el array de imagenes subidas va a pasar de ser dimensional a multidimensional y tu código va a fallar.
Para solucionar el problema de las transparencias y la cantidad de colores hay alternativas mucho mejores.
Por ejemplo este pequeño código que hize tomando la documentación desde php.net puede hacer lo mismo y mejor:

Código

```
1. <?php
2. $original = 'test.jpg';
3. $destino = 'test.png'; // La salida siempre será en PNG para no perder calidad
4.
```

```
$width_d = 100; // ancho de salida en pixeles
$height_d = 100; // alto de salida en pixeles
/* obtengo información del archivo */
list($width_s, $height_s, $type, $attr) = getimagesize($original, $info2);
/* crea el recurso gd para el origen */
if(!$gd_s = imagecreatefromstring(file_get_contents($original)))
 die('El archivo no es una imagen.'); // el archivo no es una imagen
/* crea el recurso gd para la salida */
if(!$gd_d = imagecreatetruecolor($width_d, $height_d))
 die('El archivo no es una imagen.'); // No maneja GD o escala fuera del limite

magealphablending($gd_d, false); // desactivo el procesamiento automatico de alpha
imagesavealpha($gd_d, true); // Alpha original se graba en el archivo destino
/* Redimensiona */
imagecopyresampled($gd_d, $gd_s, 0, 0, 0, 0, $width_d, $height_d, $width_s, $height_s);
unlink($original); // Elimina el archivo original
if(!imagepng($gd_d, $destino)) // Graba la imagen
 die('No tiene permisos de escritura sobre el directorio de imagenes.');
imagedestroy($gd_s); // Libera memoria
imagedestroy($gd_d); // Libera memoria
?>
```

Ahora, que alguien haya copiado el tuto de alguien mas lo dudo porque es lo mismo que puedes ver en php.net y por algo fueron hechas esas funciones y no solo el o tu lo utilizan sino miles de desarrolladores y no necesariamente lo vieron desde un tutorial.

Si quieres solucionar el problema de la subida de imagenes yo hize una función que puede solucionar eso, talves te sirva. En php.net también sale similar.

Código

```
function guardar_subidos($directorio_almacenamiento, $extensiones_permitidas = false, $crear_directorio = false, $un_solo_archivo = false){

 /* Verifica si hay archivos subidos para ser recibidos */
 if(is_array($_FILES)){
  if(count($_FILES) < 1){
   return false;
  }
 }else{
  return false;
 }

 if($un_solo_archivo){
  if(count($_FILES) > 1){
   return array(
    'estado' => 'error',
    'descripcion_error' => 'Solo es permitido subir un solo archivo y se ha detectado mas de uno'
   );
  }
 }

 /* Verifica si el directorio de guardado es válido o no */
 if(!is_dir($directorio_almacenamiento)){
  if($crear_directorio){
   if(!mkdir($directorio_almacenamiento, 0755)){
    return array(
     'estado' => 'error',
     'descripcion_error' => 'El directorio a guardar los archivos subidos no existe y fue imposible crear'
    );
   }
  }else{
   return array(
    'estado' => 'error',
    'descripcion_error' => 'El directorio a guardar los archivos subidos no existe'
   );
  }
 }

 /* Limpia la ruta del directorio evitando doble slashses y null bytes */

 $directorio_almacenamiento = dirname($directorio_almacenamiento.'/archivo.ext').'/';

 /* Si no existe el directorio entonces lo creará */

 if(!is_dir($directorio_almacenamiento)){

  if(!mkdir($directorio_almacenamiento, 0755))

   return array(
    'estado' => 'error',
    'descripcion_error' => 'No se pudo crear el nuevo directorio. Verifique los permisos de escritura sobre el directorio raiz'
   );

 }


 /* Procesa cada archivo subido para pasar de un array con uno a dos dimensiones a una sola dimensión */
 foreach($_FILES as $nombre_array => $archivo){
  /* Verifica si se ha subido un solo archivo o varios */
  if(is_array($_FILES[$nombre_array]['name'])){ /* Múltiples archivos */
   /* Procesa cada archivo subido */
   foreach($_FILES[$nombre_array]['name'] as $id => $nombre){
    /* Verifica que no sea un input vacio */
    if($_FILES[$nombre_array]['name'][$id]){
     $subidos[] = array(
      'name' => $_FILES[$nombre_array]['name'][$id],
      'type' => $_FILES[$nombre_array]['type'][$id],
      'tmp_name' => $_FILES[$nombre_array]['tmp_name'][$id],
      'error' => $_FILES[$nombre_array]['error'][$id],
      'size' => $_FILES[$nombre_array]['size'][$id]
     );
    }
   }
  }else{ /* Un solo archivo */
   $subidos[] = $archivo;
  }
 }

 /* Verifica si se ha subido algún archivo */
```

```
79.   if(is_array($subidos)){
80.    /* Procesa cada archivo subido previamente filtrado en un solo array de una dimensión */
81.    foreach($subidos as $subido){
82.     if(is_array($extensiones_permitidas)){
83.      if(archivos::coincide_extension($subido['name'], $extensiones_permitidas)){
84.       if(move_uploaded_file($subido['tmp_name'], $directorio_almacenamiento.$subido['name'])){
85.        $resultados[] = array(
86.         'estado' => 'ok',
87.         'name' => $subido['name'],
88.         'type' => $subido['type'],
89.         'error' => $subido['error'],
90.         'size' => $subido['size']
91.        );
92.       }else{
93.        $resultados[] = array(
94.         'estado' => 'error',
95.         'descripcion_error' => 'El archivo "'.$subido['name'].'" no pudo ser movido. Verifique los permisos de escritura sobre el directorio raiz',
96.         'name' => $subido['name'],
97.         'type' => $subido['type'],
98.         'error' => $subido['error'],
99.         'size' => $subido['size']
100.        );
101.       }
102.      }else{
103.       $resultados[] = array(
104.        'estado' => 'error',
105.        'descripcion_error' => 'El archivo "'.$subido['name'].'" contiene una extensión no válida',
106.        'name' => $subido['name'],
107.        'type' => $subido['type'],
108.        'error' => $subido['error'],
109.        'size' => $subido['size']
110.       );
111.      }
112.     }else{
113.      if(move_uploaded_file($subido['tmp_name'], $directorio_almacenamiento.$subido['name'])){
114.       $resultados[] = array(
115.        'estado' => 'ok',
116.        'name' => $subido['name'],
117.        'type' => $subido['type'],
118.        'error' => $subido['error'],
119.        'size' => $subido['size']
120.       );
121.      }else{
122.       $resultados[] = array(
123.        'estado' => 'error',
124.        'descripcion_error' => 'El archivo "'.$subido['name'].'" no pudo ser movido',
125.        'name' => $subido['name'],
126.        'type' => $subido['type'],
127.        'error' => $subido['error'],
128.        'size' => $subido['size']
129.       );
130.      }
131.     }
132.    }
133.   }else{
134.    return array(
135.     'estado' => 'error',
136.     'descripcion_error' => 'No hay archivos para ser subidos'
137.    );
138.   }
139.
140.   /* Retorna el resultado */
141.   return array(
142.    'estado' => 'ok',
143.    'datos' => $resultados
144.   );
145.
146.  }
147.
```

Lo usas así:

Código

```
1.  $data = guardar_subidos('/archivos/imagenes/', array('doc','xls','pdf','ppt'), true); print_r($data);
```

En ves de hacer criticas destructivas podrias aportar algo que realmente valga la pena, si quieres criticarle algo a alguien mandale un privado mostrando tu molestia pero no vengas al foro lanzando piedras porque alguien le copió a otro que copió funciones desde php.net.