
# Sécurité PHP

*FaillesUpload*

# Table des matières

# Introduction

Toutes les fonctions WEB d'upload de fichiers sont potentiellement vulnérables. L'expérience nous montre que régulièrement ces fonctions ne sont pas correctement sécurisées, particulièrement dans certains espaces d'administration ou les développeurs sont plus laxistes avec la sécurité.

Ces failles sont pourtant très dangereuses et systématiquement testées par les attaquants désireux de prendre le contrôle du site Internet cible.

Dans cet article nous aborderons plusieurs exemples de vulnérabilités afin de comprendre comment un auditeur ou pirate pourra les contourner, puis nous terminerons par un exemple de code propre et sécurisé.

## Les risques liés auxfaillesupload

- Upload d'une backdoor (porte dérobée)
- Ecrasement de fichiers

# L'upload de fichiers

Voici un exemple de code HTML et PHP d'upload de fichierssans restrictions pour que les néophytes comprennent le principe. Notez qu'il faudra les droits en écriture pour que l'on puisse importer des fichiers dans le dossier « images ».

upload.php

```
<html>
    <h1>Upload image</h1>
    <formenctype="multipart/form-data" method="POST" action="upload.php">
    <p>
          <label>Image</label>
          <input type="file" name="image">
          <br /><br />
          <input class="button" type="submit">
    </p>
</form>
```

```
<?php

if (isset($_FILES["image"]) == true)
{
    $dir = "./images/";
    $file = $dir.basename($_FILES["image"]["name"]);

    if (file_exists($file) == true)
    {
          print "Erreur! Fichier existant.";
          exit;
    }

    if (move_uploaded_file($_FILES["image"]["tmp_name"], $file))
          print "Fichier upload : <a href='" . $file . "'>ici</a>";
    else
          print "Erreur! Fichier non upload";
}

?>
```

# Protections coté client

Les restrictions coté client sont les plus facile à contourner, on peut les trouver en HTML ou en JavaScript, par exemple :

```
<input type="hidden" name="MAX_FILE_SIZE" value="1000" />
```

Cette ligne ajoutée dans le formulaireHTMLest un champ caché (*hidden*) qui limitera la taille du fichier à uploader à 1000 octets.

Pour contourner les protections coté client il ya plusieurs méthodes possible.Par exemple nous pouvons enregistrer le formulaire sur notre machine, supprimer les protections, mettre l'adresse du site et le chemin de la page PHP dans les champs « action » de « form » créant ainsi notre formulaire sans les restrictions. Il nous restera plus qu'à l'utiliser.

# Protections coté serveur

Etudions quelques protections PHP afin de voir leurs faiblesses. Dans notre démonstration l'objectif étant d'envoyer le fichier PHP suivant.

test.php

```
<?php
print "Je suis interprété par PHP";
?>
```

## 1. Vérification du type de contenu

Le code suivant prend en considération le type de contenu du fichier et n'autorise que le contenu de type image/png.

upload1.php

```
<?php

if (isset($_FILES["image"]) == true)
{
    $dir = "./images/";
    $file = $dir.basename($_FILES["image"]["name"]);

    if ($_FILES['image']['type'] != 'image/png')
    {
          print "Erreur! seul les png sont autorisés.";
          exit;
    }

    if (file_exists($file) == true)
    {
          print "Erreur! Fichier existant.";
          exit;
    }

    if (move_uploaded_file($_FILES["image"]["tmp_name"], $file))
          print "Fichier upload : <a href='" . $file . "'>ici</a>";
    else
          print "Erreur! Fichier non upload";
}

?>
```

Si l'on envoie un fichier PHP nous verrons un message d'erreur. Si l'on envoie un fichier PNG il sera correctement uploadé. La faille dans cette protection est que le type de contenu est envoyé depuis le client. Par conséquent si nous modifions le type de contenu de notre fichier PHP celui-ci sera uploadé.

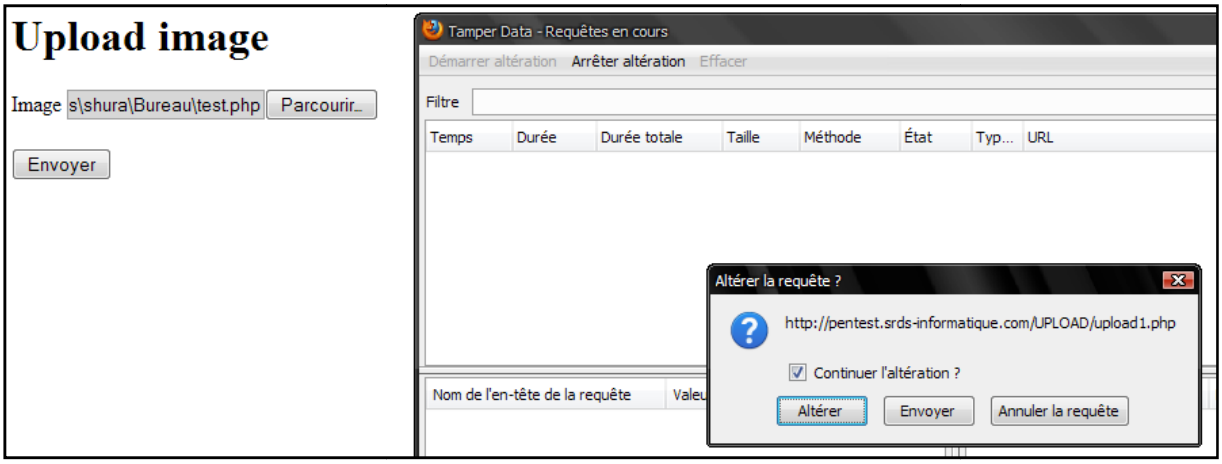
Pour la démonstration nous allons utiliser le navigateur Firefox avec l'add-on *Tamper Data* que vous pourrez trouver à l'adresse suivante :

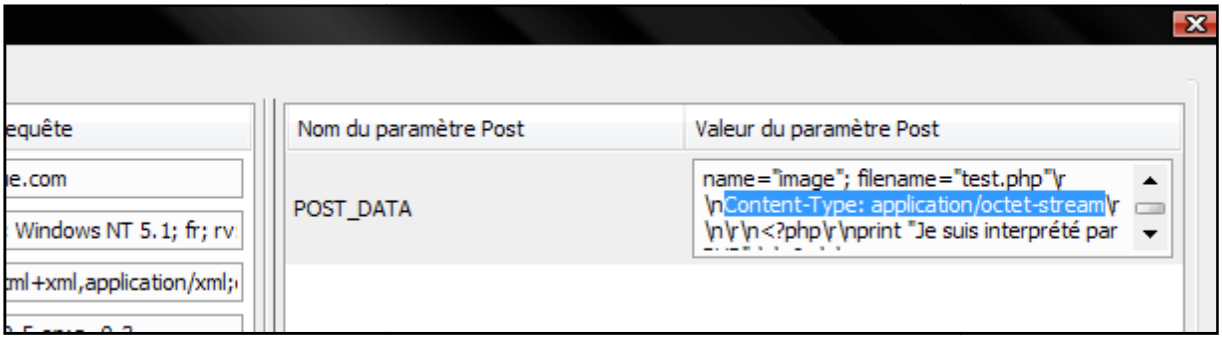
⇨ https://addons.mozilla.org/fr/firefox/addon/966/

*Tamper Data* est un outil très pratique lors d'un audit, il permet de modifier les données qui sont envoyées du navigateur vers le serveur WEB.

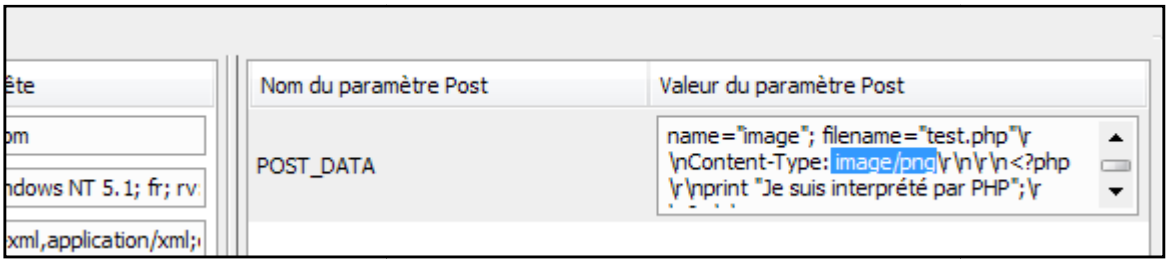
Considérons que votre navigateur est sur la page du formulaire d'upload et que *Tamper Data* est lancé et que l'altération est démarrée.

Choisissonsd'envoyernotre fichier test.php et regardons les données qui transitent lorsque nous altérons la requête.

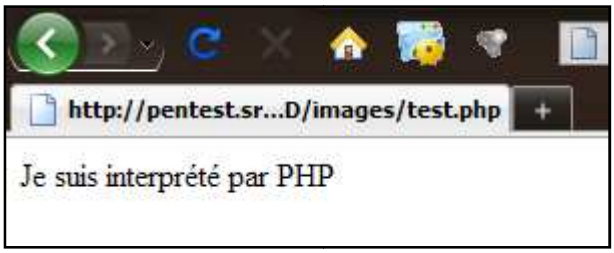



Dans POST_DATA nous voyons notre fichier test.php et son *Content-Type*. Il nous suffit de le remplacer par image/png pour pouvoir uploader notre fichier PHP.




On valide « OK ».


Coté serveur le fichier a été accepté.

## 2. Filtrage par extension

Imaginons un code PHP qui vérifie que l'extension du fichier uploadé soit parmi les extensions valides (exemple : .jpg, .gif, .png).

upload2.php

```
<?php

if (isset($_FILES["image"]) == true)
{
    $dir = "./images/";
    $file = $dir.basename($_FILES["image"]["name"]);
    $whitelist = array (".jpg", ".gif", ".png");
    $key = false;

    foreach ($whitelist as $item)
    {
          if (preg_match("/$item$/i", $_FILES['image']['name']))
          {
                $key = true;
                break;
          }
    }
```

```
    if ($key == false)
    {
          print "Erreur! seul les images sont autorisés.";
          exit;
    }

    if (file_exists($file) == true)
    {
          print "Erreur! Fichier existant.";
          exit;
    }

    if (move_uploaded_file($_FILES["image"]["tmp_name"], $file))
          print "Fichier upload : <a href='" . $file . "'>ici</a>";
    else
          print "Erreur! Fichier non upload";
}

?>
```

La faille cette fois ne vient pas que de notre script, celui se contente de vérifier les extensions et si elles correspondent avec celles de la *white list*, le fichier est autorisé.

Il arrive que la configuration d'Apache avec l'utilisation d'un code comme celui ci-dessus provoque une faille de sécurité. Quand Apache est configuré pour exécuter un script il y a deux façons de le configurer : en utilisant *AddHandler* ou *AddType*. Dans les deux cas si la configuration suivante est utilisée l'image sera interprétée.

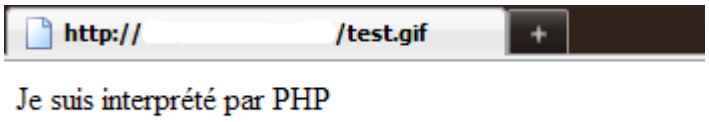
mime.conf

```
AddHandler application/x-httpd-php .gif
```

Ou

```
AddType application/x-httpd-php .gif
```

Ce genre de configuration d'Apache est utilisé plus fréquemment que l'on pourrait le penser, on peut les trouver par exemple dans le cas de figure où un code PHP génère dynamiquement un graphique. Il ne reste plus qu'à modifier l'extension de notre fichier *test.php* en *test.gif* et le tour est joué.



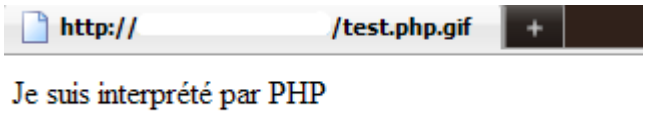

Je suis interprété par PHP

Dans la même catégorie, si le serveur Apache utilise la configuration ci-dessous nous sommes en présence d'une autre faille.

```
AddHandlerapplication/x-httpd-php.php
```

Le problème est que tous les fichiers ayant dans leurs noms « *.php* » seront interprétés par PHP, aussi bien *test.php* que *test.php.gif*.

Cette fois en renommant notre *test.php* en *test.php.gif* notre fichier passera le script de restriction des extensions et sera interprété avec succès.




## 3. Filtrage avec getimagesize()

La fonction *getimagesize()* détermine la taille de l'image fournie et en retourner les dimensions, elle peut également retourner plus d'informations comme le type de fichier.

Utilisons un script qui vérifie le type de l'image avec cette fonction et n'autorise que les JPG.

upload3.php

```
<?php

if (isset($_FILES["image"]) == true)
{
    $dir = "./images/";
    $file = $dir.basename($_FILES["image"]["name"]);
    $imageinfo = getimagesize($_FILES['image']['tmp_name']);

    if ($imageinfo["mime"] != "image/jpeg")
    {
          print "Erreur! seul les JPG sont autorisés.";
          exit;
    }

    if (file_exists($file) == true)
    {
          print "Erreur! Fichier existant.";
          exit;
    }

    if (move_uploaded_file($_FILES["image"]["tmp_name"], $file))
```

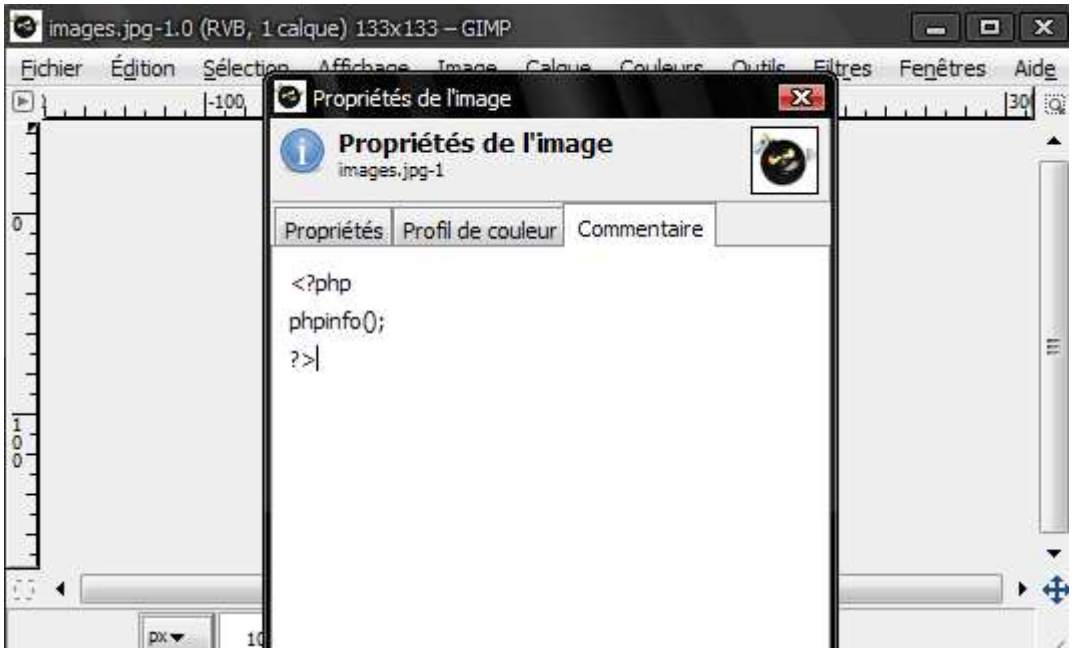
```
            print "Fichier upload : <a href='" . $file . "'>ici</a>";
    else
            print "Erreur! Fichier non upload";
}

?>
```

Il ne suffira pas de changer le *content-type* comme on l'a vu précédemment, car cette fonction travaille directement sur le fichier uploadé dans le dossier temporaire.

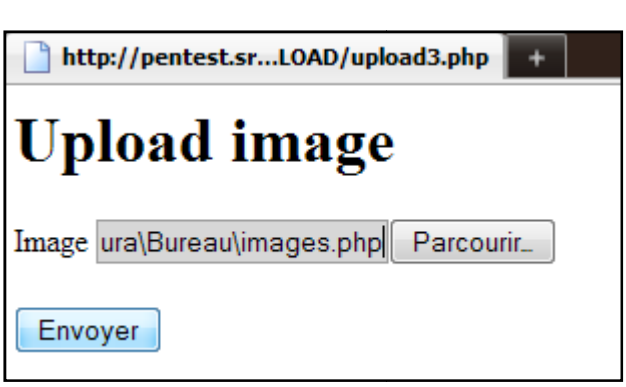
Il va falloir ruser un peu et lui fournir une vraie image pour qu'il passe le filtre,cette image aura l'extension *.php* pour que Apache demande a PHP de l'interpréter et qu'il y ait du code PHP dedans.

Procédure :

- Prenons une image quelconque JPG (nommée *images.jpg*)
- Utilisons GIMP pour lui insérer du code PHP (Image > Propriétés de l'image > Commentaire)



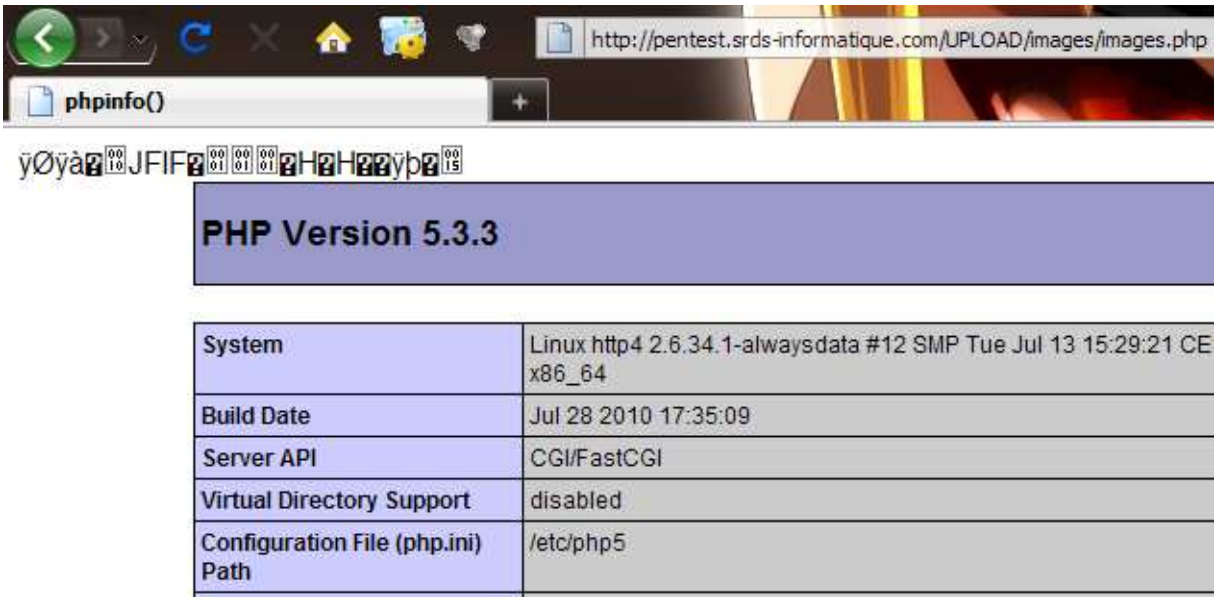

- Enregistrons
- Changeons l'extension en .php : *images.php*





Le fichier a passé le filtre et est sur le serveur.

Voyons le résultat :




## 4. .htaccess

Une autre façon de sécuriser l'envoi de fichiers et de mettre un *.htaccess* dans le répertoire d'upload qui interdit les extensions de fichiers black listés et n'autorise que les extensions de fichiers images white listés.

Pour notre démonstration nous utiliserons le *.htaccess* ci-dessous dans le répertoire images.

.htaccess

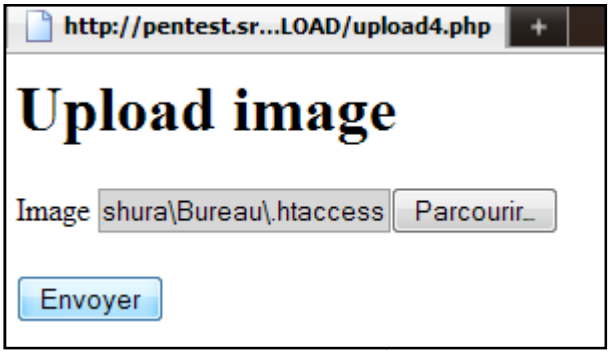
```
Options -Indexes
Options -ExecCGI
AddHandlercgi-script  .php  .php3  .php4  .phtml  .pl  .py  .jsp  .asp  .htm
.shtml .sh .cgi

<Files ^(*.jpeg|*.jpg|*.png|*.gif)>
orderdeny,allow
deny from all
</Files>
```

Ainsi que ce script d'upload.

upload4.php

```
<?php

if (isset($_FILES["image"]) == true)
{
    $dir = "./images/";
    $file = $dir.basename($_FILES["image"]["name"]);

    if (move_uploaded_file($_FILES["image"]["tmp_name"], $file))
          print "Fichier upload : <a href='" . $file . "'>ici</a>";
    else
          print "Erreur! Fichier non upload";
}

?>
```

Nous remarquons qu'il n'y a plus la vérification qui permet de déterminer si un fichier du même nom que celui à uploader existe déjà. La faille réside dans ce manque de vérification. Il devient alors possible d'envoyer un fichier *.htaccess* qui va écraser celui déjà présent.




En créant un .htaccess vierge et en l'uploadant celui écrasera l'ancien, supprimant les restrictions.

Il est maintenant possible d'exécuter du PHP dans le répertoire images.

# Contre-mesures

## Quelques suggestions

Voici quelques conseils, qui vous permettront de sécuriser ces failles :

- Ne jamais se fier à ce que peut envoyer le client.
- Vérifier la configuration d'Apache afin d'agir en conséquence.
- Ne pas placer le *.htaccess* dans le répertoire d'upload.
- Ne pas permettre l'écrasement de fichiers.
- Générer un nom aléatoire pour le fichier uploadé et enregistrer le nom dans une base de données.
- Ne pas permettre de voir l'*index of* du répertoire d'upload.
- Assigner les bonnes permissions au répertoire.
- Vérifier le *mime-type* avec *getimagesize()* et l'extension du fichier.

# Conclusion

Vous avez pu constater que de mauvaises implémentations de sécurité d'upload permettent l'exécution de code. Dans le cadre d'un audit ou d'une attaque une backdoor aurait pu être envoyée. Il est vraiment vital pour la sécurité des données que cette fonctionnalité soit correctement protégée.

Guillaume LAUMAILLET