

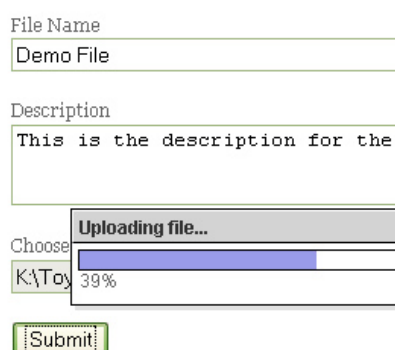
[shoxx-website.com](http://shoxx-website.com)

# Failles Upload ! | ShoxX Security Blog

*Rémi Martin*

4-6 minutes

---



De nos jours, de nombreux sites permettent d'envoyer des fichiers sur leurs serveurs, pour les partager, les montrer a tout le monde. Ces services peuvent être très dangereux si ils ne sont pas bien protégés, car on propose à l'utilisateur d'envoyer des données sur le serveur. Et ils pourraient bien envoyer par exemple du code PHP contenant une Backdoor. Voyons comment exploiter cette faille.

## Le But de l'attaque :

Prendre la décision d'attaquer un serveur est déjà une bonne chose. Mais pour y faire quoi, ça c'est une autre question ! Et bien la plus part du temps, c'est pour y envoyer une Backdoor. En français, une porte dérobée, un accès a la machine et qui

grâce aux fonctions de PHP peut être une source d'informations et de possibilités incalculables ! Imaginez, PHP permet d'exécuter des fonctions systèmes ! Vous pouvez faire ce que vous voulez si le serveur est mal paramétré et que PHP a des droits d'admins ..

Avant de tenter une telle attaque il faut s'assurer de pouvoir accéder aux fichiers une fois uploadés, si non l'intérêt est vraiment moindre.

Il existe plusieurs grands types de failles d'upload, chacune utilise une technique qui lui est propre.

Pour chacune d'entre elles j'utiliserais [Burp](#) pour modifier mes données.

### **Attaque Old School :**

Imaginons tout d'abord un serveur qui vérifierait que l'on envoie pas de fichiers contenant l'extension .php

Et exactement cette extension là, il est alors possible d'envoyer des fichiers « .php2 » « .phtml » « .phtml ».

Bref toutes les extensions que PHP sait interpréter auxquelles le programmeur n'aurait pas pensé puisque pas vraiment connues de tous !

### **Double Extensions :**

On a déjà tous vu des mentions du genre : » seuls les fichiers : .jpeg, .gif, .png sont acceptés ».

Ceci peut se traduire par : » une fois envoyé, le serveur va vérifier que votre fichier a bien pour extension un de ces formats. »

Donc, si j'envoie un « lalalalala.jpeg » ça passe, un « lalala.php » non.

Et un « lalala.php.jpeg » ?

Ce dernier respecte bien l'extension finale, et me permet d'exécuter mon code !

Voyons comment faire :

On commence par se rendre sur la page d'upload, jusque là rien de nouveau, puis on choisit un fichier .php

  
  
-----WebKitFormBoundaryjjkIABLfy6t6IW3n  
Content-Disposition: form-data; name="file"; filename="test.php"  
Content-Type: application/octet-stream  
  
<?php

Dans Burp, on bloque la communication, on peut voir ici que le fichier a bien pour nom, test et pour extension .php . Ce que le serveur cible ne va pas apprécier.

-----WebKitFormBoundaryjjkIABLfy6t6IW3n  
Content-Disposition: form-data; name="file"; filename="test.php.jpeg"  
Content-Type: application/octet-stream  
  
<?php

On modifie alors le nom de ce fichier en lui rajoutant une extension que le serveur accepte, comme par exemple .jpeg. On envoie la requête et la miracle :

**File uploaded.**

Notre fichier est bien présent sur le serveur !

**Type MIME :**

Pour lutter contre l'envoi de fichiers non souhaités, par exemple du PHP, il est possible de vérifier le type de fichiers envoyés. Pas l'extension ( car maintenant vous savez bypasser ce problème ) mais bien le type de fichiers en lui même.

```
-----WebKitFormBoundarySOLEPwaHwFtF0q5j  
Content-Disposition: form-data; name="file"; filename="test.php"  
Content-Type: application/octet-stream
```

Comme on peut le voir ici, quand je tente d'envoyer mon fichier PHP il est détecté sous le type « application/octet-stream ». Pas très bon ça, le serveur saura directement que ce n'est pas une image, et bien soit ! Changeons le type!

```
-----WebKitFormBoundarySOLEPwaHwFtF0q5j  
Content-Disposition: form-data; name="file"; filename="test.php"  
Content-Type: image/jpeg
```

On envoie, et résultat :

**File uploaded.**

Aucun problème d'envoi, le serveur pense avoir reçu un fichier de type image, alors qu'il a notre code PHP !

## Null Byte :

Pour cette dernière version de la faille d'upload, nous allons agir directement sur le fichier que nous allons envoyer. Pour cela il est utile de savoir une chose, PHP provient du C, ce qui est très important pour cette dernière faille. En effet sa provenance du C fait que PHP est sensible aux Null Bytes. Le Null Byte est utilisé par C pour signaler la fin d'une chaîne de caractères. Si vous écrivez « lalacoucou » C lira : « lala ». Nous allons donc créer un fichier » test.php.jpeg », ainsi le

contrôleur d'extension pensera que nous envoyons bien un fichier image, alors que quand PHP le liera il s'arrêtera à .php et notre code sera une fois de plus exécuté !

```
1 <?php
2     echo "Je suis un code mechant !";
3 ?>
```

### *Notre code PHP*

Une fois votre code prêt, enregistrez le comme suit :

|                  |  |  |
|------------------|--|--|
| Nom du fichier : | <input type="text" value="fichier.php%00.jpeg"/> | <input type="button" value="Enregistrer"/> |
|------------------|--|--|

Envoyez le fichier le null byte fera effet vous aurez une fois de plus réussi a uploader votre fichier !

**File uploaded.**

### **Conclusion :**

C'est tout pour notre petit tour des failles uploads, a vous d'être imaginatifs pour exploiter ces failles. Maintenant que vous les connaissez, vous pouvez choisir entre les deux cotés de la force !

Protéger votre serveur a maximum, ou exploiter ces failles et développer une super Backdoor en PHP ! Je ne vous influencerai pas, mais je suis a votre écoute si vous voulez opter pour le développement !