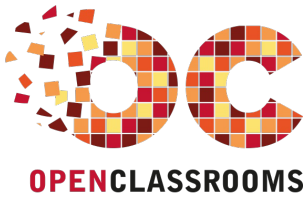
Empêcher le téléchargement direct de fichiers

Par Noxalus



www.openclassrooms.com

Dernière mise à jour le 19/12/2011

Sommaire

Empêcher le téléchargement direct de fichiers

Par  Noxalus

Mise à jour : 19/12/2011
Difficulté : Facile Durée d'étude : 15 minutes

Bonjour à toutes et à tous !

Si vous êtes ici, c'est que vous avez un site internet sur lequel se trouvent des fichiers dont vous voulez maîtriser le téléchargement. Beaucoup de sites disposent d'une page qui vous donne accès aux fichiers que vous souhaitez télécharger ; le plus souvent, elle ne fait que vous rediriger vers le fichier ciblé à coups de :

Code : PHP

```
<?php header('Location:
http://www.monsite.com/fichiers/le_fichier_a_telecharger'); ?>
```

et permet à l'administrateur, par exemple, de compter le nombre de fois qu'il a été téléchargé *via* une base de données ou un fichier texte. Une telle redirection présente un inconvénient majeur : le lien direct vers le fichier est révélé à l'utilisateur, qui peut donc le télécharger sans passer par la page de votre site prévue à cet effet. Ce qui fausse toutes vos statistiques et conduit même au « vol de fichiers », c'est-à-dire que d'autres administrateurs peuvent rendre publics les liens directs de vos fichiers sans que leurs utilisateurs ne sachent qu'ils sont hébergés sur votre site ! Malgré le trafic ainsi généré, ces personnes ne visitent pas votre site.

Heureusement, ce tutoriel vous apprendra une méthode efficace pour résoudre ce problème et vous épargnera, je l'espère, de mauvaises surprises.
Sommaire du tutoriel :

La page de téléchargement

Avant de commencer à sécuriser quoi que ce soit, nous allons voir à quoi ressemble notre page de téléchargement — ou devrais-je dire nos pages —, car nous allons prendre l'habitude de séparer les différents fichiers afin de ne pas mélanger PHP et HTML.

Voici donc notre page `telecharger.php` :

Code : PHP - telecharger.php

```
<?php
// Si l'utilisateur a demandé le téléchargement d'un fichier
if(!empty($_GET['fichier']))
 // On lance le téléchargement du fichier
else
```

```
        require('erreur.php');
    ?>
```

Ici, rien de bien compliqué : nous vérifions seulement si le téléchargement a été demandé *via* une variable `$_GET['fichier']`. Si oui, nous le lançons (nous verrons comment par la suite) ; sinon, nous afficherons un message prévenant l'utilisateur qu'il n'a demandé le téléchargement d'aucun fichier.

Afin de faire quelque chose d'à peu près propre, ledit message sera inclus dans une page, que nous pourrions nommer `erreur.php`, grâce à la fonction require() :

Code : PHP

```
<!DOCTYPE html>
<html>
 <head>
  <title>Téléchargement d'un fichier</title>
 </head>
 <body>
  <p>
   Désolé, ce fichier n'existe pas.
  </p>
 </body>
</html>
```

La partie qui nous intéresse est celle dans le cas où l'on souhaite télécharger un fichier, donc quand la variable `$_GET['fichier']` n'est pas nulle, dans le fichier `telecharger.php`.

Avant, la méthode consistait à rediriger directement l'utilisateur sur le fichier qu'il avait demandé. En imaginant que le dossier qui contient tous vos fichiers à télécharger ait pour nom fichiers et qu'il se trouve à la racine, nous aurions fait quelque chose de ce type :

Code : PHP - telecharger.php

```
<?php
if(!empty($_GET['fichier']))
{
 $chemin = 'fichiers/' . $_GET['fichier'];
 if(file_exists($chemin))
  header('Location: http://monsite.com/' . $chemin);
 else
  require('erreur.php');
}
else
 require('erreur.php');
?>
```

Nous venons d'utiliser la fonction file_exists(), qui vérifie que le fichier existe bien avant d'en faire quoi que ce soit.

Ainsi, dans le cas où un fichier `test.txt`, par exemple, se trouverait dans le dossier des fichiers (fichiers/), nous aurions seulement à donner à l'utilisateur le lien http://www.monsite.com/telecharger.php?fichier=test.txt pour qu'il puisse le télécharger (ou du moins l'ouvrir, dans le cas de notre fichier texte). Nous pourrions donc, pour reprendre l'exemple du début, rajouter un compteur afin de connaître le nombre de fois qu'il a été téléchargé.

Code : PHP - telecharger.php

```
<?php
```

```php
if(!empty($_GET['fichier']))
{
 $chemin = 'fichiers/' . $_GET['fichier'];
 if(file_exists($chemin))
 {
  // On incrémente le nombre de fois que le fichier a été téléchargé !
  incrementer_compteur($chemin); // C'est cette fonction qui va s'en occuper !
  header('Location: http://monsite.com/' . $chemin);
 }
 else
  require('erreur.php');
}
else
 require('erreur.php');
?>
```

Cependant, cette méthode ne résout pas le problème du vol de fichiers puisqu'elle communique à l'utilisateur le lien direct vers le fichier qu'il souhaite télécharger. Il lui sera non seulement possible d'y accéder de nouveau sans passer par la page de téléchargement, mais votre compteur ne vous sera plus d'aucune utilité. Nous allons voir tout de suite comment empêcher cela.

Protégeons nos fichiers

Il est temps de vous expliquer comment protéger nos fichiers. Et là, le PHP ne pourra malheureusement rien pour nous : la protection des fichiers concerne directement le serveur. Pour lui donner des instructions, il faut modifier le **.htaccess**.

Je ne vous expliquerai pas le fonctionnement d'un fichier .htaccess, ni même ne rentrerai dans les détails de ce qu'il est possible de faire grâce à lui, puisque d'autres l'ont fait avant moi. Je vais simplement vous dire en quoi il nous sera utile. Pour faire court, une des possibilités du .htaccess est d'empêcher l'accès à des dossiers entiers avec ce petit bout de texte :

Code : Apache

```
Deny From All
```

En ajoutant ce code à un fichier texte nommé .htaccess et en plaçant ce dernier dans le dossier de votre choix, vous tomberez sur une jolie erreur 403 : *Forbidden* (ce qui signifie « Interdit », pour les anglophobes) si vous essayez d'atteindre un élément présent dans ce dossier ou dans un de ses sous-dossiers.

Code : Console

```
Forbidden

You don't have permission to access [chemin du fichier] on this server.
```

Youhouhou, c'est exactement ce que nous voulions ! Maintenant, si ces vilains fraudeurs veulent accéder directement aux fichiers, ils ne le pourront plus.

C'est génial ! Mais… Euh… La page de téléchargement ne fonctionne plus… C'est normal ?!

Bien sûr, notre page ne fait que rediriger l'utilisateur vers le fichier qui va bien. Donc, si nous empêchons l'accès à ces fichiers, il ne pourra pas non plus les télécharger en passant par notre page. Ce qui n'est pas sans poser problème, n'est-ce pas ? 
Heureusement, le but de ce tutoriel est de vous expliquer comment contourner cette protection.

Je préfère que les choses soient claires. TOUS les fichiers qui se trouvent dans le même dossier que le .htaccess, ainsi

que les fichiers situés dans d'éventuels sous-dossiers, seront inaccessibles.
Par conséquent, il est primordial de n'y placer aucun fichier PHP. Ces dossiers doivent uniquement contenir les fichiers que vos utilisateurs sont autorisés à télécharger.

Passer outre la protection

Tel est notre but : faire en sorte de garder la protection .htaccess tout en autorisant le téléchargement grâce à notre page. Pour cela, vous devez savoir que les fonctions PHP se servant de fichiers ne prennent pas en compte les protections appliquées par un .htaccess. Ainsi, il est possible d'ouvrir, de modifier, voire de supprimer un fichier censé être protégé par ce même .htaccess.

Là où ça nous arrange, c'est qu'il existe une fonction readfile() qui permet, comme le dit la documentation, de lire un fichier et de l'envoyer dans le *buffer* de sortie. On ne redirige donc plus vers le fichier, on prend carrément son contenu !

En reprenant le code précédent, nous pouvons donc remplacer le header() (la redirection) par ceci :

Code : PHP

```php
<?php
if(!empty($_GET['fichier']))
{
 $chemin = 'fichiers/' . $_GET['fichier'];
 if(file_exists($chemin))
  readfile($chemin);
 else
  require('erreur.php');
}
?>
```

Super ! Mais… Euh… Quand j'essaye de télécharger une image, de nombreux symboles bizarres s'affichent. C'est normal ?!

Encore une fois, oui, c'est tout à fait normal. Vous avez déjà tenté d'ouvrir une image avec le bloc-notes ? Eh bien, cela devrait donner la même chose ! En effet, readfile() ne fait que lire le contenu du fichier et ne cherche pas à l'afficher en fonction de son type.

Cela dit, malgré le .htaccess, le contenu de notre fichier est bien lu et affiché. La solution que nous allons employer pour ne plus afficher le contenu du fichier est d'en forcer le téléchargement.

Forcer le téléchargement des fichiers

Voici une petite astuce très pratique pour forcer le téléchargement de n'importe quel fichier, astuce qui pourra d'ailleurs résoudre le problème précédent.

Vous avez dû remarquer à plusieurs reprises que, quelle que soit la technique utilisée, sécurisée ou non, tous les fichiers qui peuvent être ouverts par le navigateur le sont effectivement — à condition, bien entendu, de ne pas avoir modifié la configuration du navigateur. Donc, en temps normal, plutôt que de télécharger un fichier au format texte ou XML, une image, un PDF et beaucoup d'autres, votre navigateur l'ouvrira. Vous pouvez ensuite télécharger le fichier en question grâce à un clic droit puis à « Enregistrer sous », mais ce n'est vraiment pas pratique.

Néanmoins, il est possible, avec les en-têtes HTTP, de forcer la main aux navigateurs et de les obliger à télécharger ledit fichier avec ces quelques lignes :

Code : PHP

```php
<?php
header('Content-Description: File Transfer');
header('Content-Type: application/octet-stream');
header('Content-Disposition: attachment; filename=' .
basename($chemin));
header('Content-Transfer-Encoding: binary');
```

```
header('Expires: 0');
header('Cache-Control: must-revalidate, post-check=0, pre-check=0');
header('Pragma: public');
header('Content-Length: ' . filesize($chemin));
readfile($chemin);
exit;
?>
```

Ce code est extrait de la documentation PHP de la fonction readfile() et n'a subi qu'une simple modification. En effet, la gestion de la tamporisation de sortie est devenue inutile dans notre cas.

Ne vous inquiétez pas, je ne vais pas vous laisser avec ce code sans aucune explication.

Pour commencer, notons qu'il y a beaucoup d'appels à la fonction header(). Nous l'avons vu au début de ce cours, cette fonction assure une redirection (vers un fichier par exemple) si elle est utilisée avec `'Location: [adresse de redirection]'`. Elle permet en réalité bien plus que cela, notamment de modifier l'en-tête HTTP reçu par le navigateur. Ainsi, vous pourrez facilement lui dire qu'il s'agit d'un fichier à télécharger et l'obliger à l'interpréter comme tel.

Voici maintenant le détail de chacun de ces header() :

- **Ligne 2** : spécifie au navigateur que les données qu'il va recevoir doivent être considérées comme un fichier à télécharger.
- **Ligne 3** : indique que le flux de données qui va suivre est de type « flux d'octet ». Comme n'importe quel fichier peut être considéré ainsi, il n'est pas nécessaire d'en connaître le type MIME exact (qui diffère en fonction de l'extension).
- **Ligne 4** : attribue un nom au fichier. Par conséquent, le nom qui apparaîtra dans la *popup* de téléchargement sera celui indiqué après `filename=`.
- **Ligne 5** : précise que le fichier à traiter devra être envoyé en binaire. En d'autres termes, les données seront conservées telles quelles afin d'éviter les problèmes d'encodage et de « transformation » non voulus.
- **Lignes 6 à 8** : ces lignes ordonnent au navigateur de ne pas mettre les fichiers en cache pour que le téléchargement soit déclenché à chaque fois.
- **Ligne 9** : donne au navigateur la taille du fichier, sans laquelle il ne pourrait afficher correctement la barre de progression ni donner le pourcentage déjà téléchargé — et encore moins estimer le temps restant.

Pour aller un peu plus loin avec les en-têtes HTTP, vous trouverez d'autres informations sur ce site.

En ce qui concerne les fonctions PHP maintenant : basename() permet, à partir d'un chemin, de récupérer uniquement le nom d'un fichier ; filesize() (les anglophones l'auront déjà deviné) en indique la taille. Quant à la fonction exit(), qui clôt cette série d'appels, elle sert à stopper l'exécution du *script* pour ne pas rajouter de données à la suite du fichier sous peine de le corrompre. De plus, il est inutile de charger le reste d'une page qui ne sera, en théorie, pas visible.

Pour continuer à garder un code clair et propre, nous allons créer une fonction, appelée readfile(), à la seule fin d'envoyer le *header* personnalisé et même de faire la vérification.

Vous devez certainement déjà posséder un fichier `fonctions.php` dans lequel vous mettez toutes les fonctions de votre site. Il vous suffit de lui ajouter la fonction suivante :

Code : PHP - fonctions.php

```
<?php
function telecharger_fichier($fichier)
{
 $chemin = 'fichiers/' . $fichier;
 if(file_exists($chemin))
 {
  header('Content-Description: File Transfer');
  header('Content-Type: application/octet-stream');
  header('Content-Disposition: attachment; filename=' .
basename($chemin));
```

```
  header('Content-Transfer-Encoding: binary');
  header('Expires: 0');
  header('Cache-Control: must-revalidate, post-check=0, pre-
check=0');
  header('Pragma: public');
  header('Content-Length: ' . filesize($chemin));
  readfile($chemin);
  exit;
 }
 else
  require('erreur.php');
}
?>
```

Si vous voulez faire en sorte d'afficher les images plutôt que d'obliger l'utilisateur à les télécharger, vous pouvez modifier cette fonction comme ceci :

Secret (cliquez pour afficher)

Code : PHP - fonctions.php

```
<?php
function telecharger_fichier($fichier)
{
 $chemin = 'fichiers/' . $fichier;
$images_ext = array('jpg', 'jpeg', 'png', 'bmp', 'gif');
 if(file_exists($chemin))
 {
$image format = substr(strrchr($chemin, '.'), 1);
if(in array($image format, $images ext))
header('Content-Type: image/' . $image_format);
  else
  {
   header('Content-Description: File Transfer');
   header('Content-Type: application/octet-stream');
   header('Content-Disposition: attachment; filename=' .
basename($chemin));
   header('Content-Transfer-Encoding: binary');
   header('Expires: 0');
   header('Cache-Control: must-revalidate, post-check=0,
pre-check=0');
   header('Pragma: public');
   header('Content-Length: ' . filesize($chemin));
  }
  readfile($chemin);
  exit;
 }
 else
  require('erreur.php');
}
?>
```

Le principe de cette méthode est d'interdire l'accès direct à vos images (clic droit → « Afficher l'image ») tout en permettant leur affichage. Par contre, vu que c'est du bonus, je n'expliquerai pas en détail les rajouts. S'il y a quelque chose que vous ne comprenez pas, faites-le-moi savoir par MP ou dans les commentaires. Sinon, la documentation est toujours là pour vous aider.

Ce qui nous amène à la modification suivante de notre fichier `telecharger.php` :

Code : PHP - telecharger.php

```php
<?php
if(!empty($_GET['fichier']))
{
 // N'oubliez pas d'inclure le fichier qui contient notre fonction
!
 include('fonctions.php');
 // On appelle la fonction qu'on a créée juste avant !
 telecharger_fichier($_GET['fichier']);
}
else
 require('erreur.php');
?>
```

Une faille particulièrement dangereuse

Nous avons désormais un *script* qui fonctionne très bien. Malheureusement, et vous ne vous en êtes peut-être pas rendu compte, il possède une énorme faille…

Oui, imaginez qu'un utilisateur malveillant décide de se rendre sur cette page : http://www.monsite.com/telecharger.php [...] =../index.php, il sera alors en mesure de télécharger directement le contenu de votre page d'accueil. Pire, avec cette technique, l'ensemble des fichiers de votre site sera à sa disposition et, pour peu qu'il connaisse sa structure, il pourra faire une copie complète de votre site comme s'il avait accès à votre FTP. Il n'aurait plus qu'à récupérer les identifiants de votre base de données (que vous avez forcément stockés dans l'un de vos fichiers PHP) pour finir de voler votre site en entier !

Vous ne pensiez pas qu'un petit *script* comme celui-là se révélerait aussi dangereux, n'est-ce pas ? C'est pourquoi il est important de se prémunir contre cette faille. Heureusement, je vous propose une solution permettant de pallier ce problème.

La première chose à faire est d'empêcher le visiteur d'utiliser le *slash* (/). Ainsi, il ne pourra pas retourner en arrière ou naviguer dans les dossiers à sa guise.

Oui, mais attends ! Si je fais ça, je ne pourrais plus classer les fichiers téléchargeables qui se trouvent dans le dossier fichiers/ à l'intérieur de sous-dossiers. Tous les fichiers téléchargeables ne doivent-ils pas se trouver à la racine du dossier fichiers/ ?!

Pas nécessairement. C'est à vous et à vous seul de définir les règles, et vous pouvez très bien choisir de rajouter des variables GET qui détermineront dans quel sous-dossier le fichier demandé se trouve.

Un petit exemple avec ce lien : http://www.monsite.com/telecharger.php [...] txt&dossier=1. Il suffit de faire en sorte que l'identifiant **1** corresponde à un dossier particulier pour la variable `$_GET['dossier']`. Nous pourrions organiser les dossiers de la façon suivante :

- **1** = *fichiers/jeux/* ;
- **2** = *fichiers/musiques/* ;
- **3** = *fichiers/videos/* ;
- **4** = *fichiers/images/* ;
- etc.

Ce qui, en programmation, donnerait ceci :

Code : PHP

```php
<?php
function id_dossier($id)
{
 $dossier_base = 'fichiers/';
 switch($id)
 {
  case 1:
   $dossier_base .= 'jeux/';
```

```
        break;
      case 2:
        $dossier_base .= 'musiques/';
        break;
      case 3:
        $dossier_base .= 'videos/';
        break;
      case 4:
        $dossier_base .= 'images/';
        break;
      default:
        break;
    }
    return $dossier_base;
}
?>
```

Mais ce n'est pas tout… Même si l'on bouche cette faille, il en reste une : quoi qu'il arrive, il y aura toujours dans votre dossier des fichiers qui ne seront pas destinés au téléchargement. C'est par exemple le cas pour notre .htaccess, qui devra forcément se trouver à cet endroit, et pour tous les fichiers cachés dont les noms commencent par un point. Il faut donc vérifier également qu'il n'y a pas de point au début du nom du fichier.

Pour ce faire, nous allons utiliser la fonction strpos() qui permet de récupérer la position d'un caractère (ou de plusieurs caractères dans notre cas) dans une chaîne.

La position d'un caractère dans une chaîne ? À quoi cela va-t-il nous servir ?

À une seule chose : vérifier qu'il y a ou non des caractères que nous voulons interdire. Si les caractères en question ne sont pas trouvés, la fonction renverra *FALSE*.

Ça, c'était pour le *slash*. Pour le point, en revanche, récupérer la position du caractère va nous être utile. Non, il n'est pas interdit de mettre un point dans le nom d'un fichier, surtout si vous décidez de laisser l'extension. Par contre, nous ne voulons surtout pas que ce point se trouve au début du nom du fichier ; nous vérifierons donc la position du point.

En sachant tout cela, notre fichier de fonctions sera modifié de cette façon :

Code : PHP - fonctions.php

```
<?php
function id_dossier($id)
{
    $dossier_base = 'fichiers/';
    switch($id)
  {
    case 1:
      $dossier_base .= 'jeux/';
      break;
    case 2:
      $dossier_base .= 'musiques/';
      break;
    case 3:
      $dossier_base .= 'videos/';
      break;
    case 4:
      $dossier_base .= 'images/';
      break;
    default:
      break;
  }
  return $dossier_base;
}
```

```
function telecharger_fichier($fichier, $id = 0)
{
$chemin = id dossier($id) . $fichier;
if(file exists($chemin) && strpos($fichier, '/') === FALSE &&
strpos($fichier, '.') !== 0)
 {
  header('Content-Description: File Transfer');
  header('Content-Type: application/octet-stream');
  header('Content-Disposition: attachment; filename='.
basename($chemin));
  header('Content-Transfer-Encoding: binary');
  header('Expires: 0');
  header('Cache-Control: must-revalidate, post-check=0, pre-
check=0');
  header('Pragma: public');
  header('Content-Length: ' . filesize($chemin));
  readfile($chemin);
  exit;
 }
 else
  require('erreur.php');
}
?>
```

Ne vous inquiétez pas, je ne me suis pas trompé en mettant trois signes « égal ». L'opérateur === signifie « égal et de même type » et, ici, la condition est nécessaire à notre vérification, car *0 == FALSE* mais *0 !== FALSE* !

Nous nous assurons ainsi que l'utilisateur ne pourra pas naviguer à l'extérieur du dossier dans lequel se trouvent tous les fichiers à télécharger.

Vous ne devez donc pas — j'insiste encore sur ce point — placer dans ce dossier des fichiers qui ne sont pas censés être téléchargés, comme des fichiers PHP. Une bonne fois pour toutes : le dossier fichiers/ ne doit contenir que les fichiers que vos utilisateurs sont autorisés à télécharger !

De ces changements découle une modification minime de notre fichier `telecharger.php` afin de prendre en compte l'identifiant du dossier :

Code : PHP - telecharger.php

```
<?php
if(!empty($_GET['fichier']))
{
 // N'oubliez pas d'inclure le fichier qui contient notre fonction
!
 include('fonctions.php');
 // On appelle la fonction qu'on a créée juste avant !
telecharger_fichier($_GET['fichier'], $_GET['dossier']);
}
else
 require('erreur.php');
?>
```

La suite va vous présenter une autre solution un peu plus aboutie (à noter qu'elle reste néanmoins complémentaire de la première). Cela veut dire qu'il est hautement préférable d'appliquer cette solution en plus, et vous allez tout de suite comprendre pourquoi.

Comme je l'ai dit au début de ce cours, sur de nombreux sites, les fichiers à télécharger sont gérés par une base de données. Par conséquent, chaque fichier est lié à des informations comme un nom (plus joli que le nom du fichier), une description, un compteur, etc. Ainsi, il vous suffirait de vérifier que le fichier que veut télécharger l'utilisateur figure bien dans la base de

données. Dans le cas contraire, le téléchargement n'est pas lancé.

Cette solution peut ne pas suffire lorsque vous permettez le téléchargement de fichiers PHP (pour des tutoriels par exemple) et que ces derniers portent, parfois, le même nom que ceux qui se trouvent sur votre site. Dans ce cas-là, il est important de vérifier que le fichier que le visiteur souhaite télécharger se trouve dans le bon dossier. C'est précisément ce que nous venons de faire.

Ce qui donne sous forme de code :

Code : PHP

```php
<?php
/*
On imagine que la connexion se fait avec PDO et que l'objet
PDO se trouve dans la variable du nom de $pdo
*/
$nom_fichier = $pdo->quote($_GET['fichier']);
$sql = $pdo->query("SELECT COUNT(nom_fichier) FROM fichiers WHERE
nom_fichier = '$nom_fichier'");
if(file_exists($chemin) && strpos($fichier, '/') === FALSE &&
strpos($fichier, '.') !== 0 &&
$sql->fetchColumn() > 0)
{
 // Appel de la fonction pour télécharger le fichier
}
?>
```

Il ne s'agit là que d'un exemple qui ne fonctionnera qu'à deux conditions. Premièrement, que vous vous soyez connecté avec PDO en stockant l'objet PDO dans la variable qui répond au doux nom de $pdo ; deuxièmement, que vous possédiez une table de votre base de données, nommée fichiers, qui comporte une colonne *nom_fichier* et où sont stockés tous les fichiers téléchargeables.

Pfiou, ça en fait des conditions !

Quoi qu'il en soit, je pense que vous avez compris que cet exemple vous permet uniquement de comprendre le concept. À vous de l'adapter au fonctionnement de votre site.
Ouf, vos fichiers sont maintenant à l'abri de ces méchants voleurs !

Vous pouvez désormais vous concentrer sur des applications concrètes se servant de cette protection. Nous avons vu le cas du compteur de téléchargement, mais d'autres horizons s'offrent à vous : instaurer un système de téléchargement payant, avec un nombre limité de téléchargements ; ou encore empêcher tout téléchargement pendant une plage d'heures définie pour éviter une surcharge du serveur. Et encore beaucoup d'autres choses que je vous laisse imaginer !

En tout cas, voilà, ce tutoriel est fini. Il était court (même très court), mais j'espère qu'il vous aura été aussi utile qu'à moi. D'ailleurs, je souhaite faire quelques remerciements. En effet, ce cours a été fait après que j'ai posé une question sur le forum et à laquelle deux personnes ont apporté de très bonnes réponses : maxima et orklah. Merci à eux, donc, sans qui je n'aurais certainement rien fait et serait resté, tout comme vous, dans l'ignorance. Ah, c'est beau le partage !

J'aimerais remercier une autre personne : Kyle Katarn, qui, en plus de m'avoir mis au courant de l'énorme faille, m'a donné beaucoup de conseils tant sur le fond que sur la forme pour faire de ce tutoriel ce qu'il est aujourd'hui. Merci encore !

Ce tutoriel a été corrigé par les zCorrecteurs.