

Tutoriel pour la conception de systèmes anti "Brute Force" simples.

par M Alexandre Joly ([autres articles](#))

Date de publication : 22/07/2007

Dernière mise à jour : 22/07/20073

Cet article a pour but de vous aidez a concevoir un système anti-brute force simple mais efficace, de deux manière une par fichier et l'autre avec une base de donnée.

- 0 - Avant-propos
- 1 - Brute Force c'est quoi ?
 - 1.1 - Définition
 - 1.2 - Fonctionnement
 - 1.3 - Quotas
- 2 - Les logiques de protection
 - 2.1 - Protection par nombre d'attaques/jour
 - 2.2 - Protection sur une base attaques/temps
 - 2.3 - Protection par nombre d'attaques/compte
- 3 - Anti "Brute Force" par fichier
 - 3.1 - Conception des besoins et choix de la logique
 - 3.2 - Réalisation de l'anti "Brute Force"
 - 3.3 - Explications sur le code
 - 3.4 - Conclusion et évolution possible
- 4 - Anti "Brute Force" par base de données
 - 4.1 - Conception des besoins et choix de la logique
 - 4.2 - Réalisation de l'anti "Brute Force"
 - 4.3 - Explication sur le code source et les modifications apportées.
 - 4.4 - Conclusion et évolution possible
- 5 - Bibliographie
- 6 - Conclusion & Remerciements
 - 6.1 - Remerciement
 - 6.2 - Conclusion

0 - Avant-propos

Le but de cet article est de vous montrer, le principe de base d'une protection contre les attaques dites de "Brute Force" ou "force brute". J'ai décidé de vous expliquer deux des très nombreuses manières de procéder et de les mettre en pratique à travers deux exemples concrets. Ces exemples seront utilisables sur vos sites. Il y aura tout d'abord un système de gestion par fichier et un autre avec une base de données MySql. De plus il vous sera présenté les différentes logiques anti "Brute Force" que je connais, il est même possible de combiner ces logiques de protection pour en avoir une meilleure encore.

Il va de soit que seul un anti "Brute Force" n'est pas suffisant, des mots de passe complexes doivent être utilisés ainsi que d'autres protections.

1 - Brute Force c'est quoi ?

Nous allons voir ici le principe d'une attaque par "Brute force", et son fonctionnement.

1.1 - Définition

Méthode d'analyse de chiffrement dans laquelle toutes les clés possibles sont systématiquement essayées. Le terme de "brute" qualifie parfaitement cette démarche, aussi appelée "recherche exhaustive", effectuée sans grande subtilité. La méthode est autant employée par les pirates que ceux qui cherchent à s'en protéger.

Auteur : Alain Ceccato // **Source** : Dictionnaire de developpez.com

1.2 - Fonctionnement

Pour réaliser une attaque par "force brute" sur un formulaire de connexion, il faut valider le formulaire un grand nombre de fois le plus rapidement possible avec des valeurs aléatoires. Généralement se sont donc des "bots", programmes informatique agissant seuls, qui effectuent ce genre de manoeuvre. Il y a deux cas de figure classiques du déroulement d'une attaque par "Brute force". La première, l'attaquant ne connaît ni le pseudonyme ni le mot de passe, il va donc devoir valider le formulaire en essayant des pseudonymes et des mots de passe aléatoirement. Cette attaque découragera bon nombre d'attaquants tant les chances de réussites sont minces.

Ensuite le cas le plus fréquent est que l'attaquant connaît un pseudonyme valide. Chose qui se trouve généralement très facilement sur un forum ou des sites de news. Par exemple on aurait aucun mal à connaître un pseudonyme sur le forum de developpez.com, nous prendrons donc le mien comme exemple à savoir Inazo. Dans ce cas l'attaquant n'a plus qu'à valider le formulaire avec comme valeur pour le pseudonyme "Inazo". Si aucune protection n'est mise en place, même si mon mot de passe est plutôt coton, cela lui prendra un peu de temps. Mais si vous avez mis un mot de passe tel que le nom de votre femme, par exemple Cindy il mettra moins d'une journée pour le trouver.

1.3 - Quotas

Le dernier point qui est plutôt obscur est la notion de quotas. Les quotas ne sont incrémentés qu'en cas d'échec de connexion. Donc si nous avons un quotas de 20 tentatives, et que l'on se connecte 40 fois sans la moindre erreur le quotas n'aura pas été utilisé. Par contre si on se trompe 20 fois, la 21 fois même si c'est le bon mot de passe avec le bon login l'accès vous sera refusé.

2 - Les logiques de protection

2.1 - Protection par nombre d'attaques/jour

Cette première logique de protection fonctionne de la manière suivante, on autorise un quota d'essai par jour pour la connexion d'une personne à son compte. Sur certains sites par exemple, avec mon pseudonyme Inazo j'aurais le droit à 40 tentatives de connexion dans la journée.

Le but est de faire en sorte qu'au bout de 40 tentatives manquées, par exemple, le compte soit bloqué pour le reste de la journée. Le lendemain il sera à nouveau possible de faire 40 tentatives. Ce qui en terme de temps pour trouver un mot de passe devient énorme surtout si on a un mot de passe complexe.

Il faut faire cependant attention à ne pas mettre un taux trop faible je pense pour ma part qu'entre 20 et 40 tentatives est une bonne moyenne. Cependant cela à fait débat au sein des forums réservé aux rédacteurs de developpez.com. En effet la majorité, étant le seul à penser le contraire, estime que cinq ou dix tentatives devraient être un maximum, il est vrai toutefois que cela peut être plus juste de ne mettre que cinq tentative raté au maximum. Mais pour ce type de protection je pense que cela est trop faible. Pour une logique de base attaques/temps oui il faut des quotas très petit cinq est une très bonne solution. Mais quant on est par jours un quotas plus long devrait être utilisé je pense que 20 est un bon compromis.

2.2 - Protection sur une base attaques/temps

La logique que j'appelle attaques/temps. Prenons un exemple : Vous disposez d'un crédit de cinq tentatives et en cas d'échec vous devrez patienter 15 minutes.

Avec cette technique l'attaquant devra prendre son mal en patience certes il aura la possibilité de faire aussi un grand nombre de tentative mais cela lui prendra plus de temps qu'avec la technique attaques/jour. Attention cependant lorsque l'on utilise cette méthode à garder une chose en tête : "un être humain ne peut valider une page qu'en une seconde pour les plus rapides".

En sachant bien sûr que si la personne doit taper uniquement son mot de passe de quatre caractères cela prendra au moins plus d'une seconde voir deux si la taille minimum est au moins de six caractères. Donc si on valide un formulaire de connexion plus de soixante fois en une minute il y a anguille sous roche.

2.3 - Protection par nombre d'attaques/compte

La logique consiste donc au bout, par exemple, de 10 tentatives de bloquer une fois pour toute le compte de la personne visée. Cette dernière logique est celle que j'affectionne le moins, car elle pénalise beaucoup trop les membres d'un site. Et dans le cas ou un petit malin voudrait juste s'amuser à bloquer tous les comptes d'administrateur ou de modérateur cela serait gênant.

Pour que le compte soit débloqué il est nécessaire que l'utilisateur contacte l'assistance du site pour accéder de nouveau à son compte. Une autre raison pour laquelle je n'aime pas cette logique j'en ai été victime sur un site très connu je me suis trompé seulement cinq fois de mot de passe et mon compte a été bloqué. Néanmoins vous pouvez l'utiliser mais en mettant un fort quota dessus ce qui rend vulnérable votre connexion aux espaces restreints. Préférez donc l'une des techniques citée ci-dessus.

3 - Anti "Brute Force" par fichier

3.1 - Conception des besoins et choix de la logique

Dans notre exemple nous allons nous placer dans le contexte d'un site de news communautaire. L'objectif est de pouvoir permettre aux membres de ne pas se faire voler leur mot de passe, mais étant donné que c'est un site de news communautaire les pseudos seront connus de qui se donne la peine de visiter le site. De plus étant un site de news les membres auront tendance à venir se connecter plusieurs fois par jour pour certain.

Je vais utiliser pour cet exemple la logique des "attaques/jour" car elle me semble relativement bien adaptée à ce type de site internet. Il est donc d'abord indispensable de définir un quota journalier pour les tentatives. Ici nous prendrons un quota de 20 tentatives infructueuses avant le blocage du compte, car comme dit plus haut les personnes se connectent souvent plusieurs fois par jours.

Ensuite nous allons donc créer notre arborescence pour cette protection, à savoir que les logins et mots de passe seront stockés dans une base de données et nos informations sur les tentatives dans un fichier. Ces fichiers, nous allons les ranger à la racine de notre site dans le répertoire : "BruteForce". Nous allons mettre un .htaccess dans ce répertoire pour en interdire l'accès complet à quiconque.

Et pour finir ce que nous allons mettre dans ces fichiers seront la date de tentative des attaques, le nombre d'attaques et un séparateur. Pour nommer les fichiers, nous allons simplement utiliser l'identifiant de la base de données des membres. Pourquoi pas le pseudonyme essayé me direz-vous ? Tous simplement pour ne pas avoir n'importe quoi comme nom de fichier et aussi pour ne pas surcharger le serveur de fichiers si le pseudo n'existe pas dans la base. Je reviendrais sur ce dernier point par la suite.

3.2 - Réalisation de l'anti "Brute Force"

J'ai décidé de faire cette anti "Brute Force", sous la forme d'une fonction simple pour ne pas trop embrouiller les personnes qui ne sont pas familières avec les objets. L'essentiel étant de bien comprendre la logique même de l'anti "Brute Force". Elle sera nommée "VerificationQuotas" et comportera un argument obligatoire pour donner le chemin vers le répertoire de stockage des fichiers de contrôle des quotas. Ensuite nous aurons pour cette fonction un second argument qui lui sera facultatif, qui définira le quota en lui même, qui par défaut sera de 20.

Nous allons travailler avec la base de données suivante :

Table Membres	
IdMembre	INT(11)
Login	VARCHAR(40)
LoginHasher	VARCHAR(32)
MotDePasse	VARCHAR(32)

Les champs LoginHasher et Mot_de_Passe sont hashés en md5 ceci pour nous permettre de les vérifier facilement et en toute sécurité. C'est la fonction "VerificationQuotas" qui me dira si mon utilisateur est identifié correctement ou non, c'est donc dans cette fonction que seront vérifiées les informations fournies par l'internaute.

Attention cependant, après une discussion avec d'autres rédacteurs, il n'est pas obligatoire et un peu futile pour ce tutoriel d'utiliser un md5 pour le login. Mais si vous n'utilisez pas cette méthode n'oubliez, dans le cas d'une base de données MySql, d'utiliser les fonctions PHP prévu pour sécuriser les données pour éviter les injections de requêtes SQL; tel que `mysql_real_escape_string()`. De plus pour les très parano il faut savoir que le md5 n'est pas un hashage

sûr; personnellement j'utilise le sha256, que l'on utilise comme ceci en PHP : `hash('sha256', $Password)`, et pour les plus parano il y a le sha512. Cette remarque est valable pour le second exemple avec la base de données.

Réalisation de la page d'index pour la connexion dans notre exemple :

Page d'index

```
<?php

/* Ici on inclue notre Script où se trouve la fonction */
require_once('AntiBruteForceFichier.php');

/* Ce sera notre page d'index pour notre exemple.
 * Dans cette page se trouve le formulaire de connexion et les traitements de ce dernier y compris
 * l'anti brute force
 *
 */

if(VerificationQuotas('C:/wamp/www/TutoBruteForce/ParFichier/BruteForce')){

    echo 'Vous êtes bien identifié.';

}
else{
    /*
    * Formulaire de connexion
    * Avec juste un champ pour le login et un autre pour le mot de passe
    *
    */

    echo '
        <form action="" method="post">
        <label for="Log">Login : </label><input type="text" name="Login" value="" size="40"
id="Log"/><br />
        <label for="Mdp">Mot de passe : </label><input type="password" name="Mdp" value=""
size="40" id="Mdp"/><br />
        <input type="submit" name="Send" value="Connexion" />
        </form>';
}
?>
```

Réalisation de la fonction pour nous permettre de gérer les quotas de tentatives :

Function VerificationQuotas

```
<?php

/* Ce script ne comportera que la fonction pour la vérification
 * Cette fonction servira aussi pour faire l'authentification
 */

function VerificationQuotas($CheminFichierQuotas, $Quotas = 20){

    /* On va vérifier qu'aucun des deux champs n'est vide */
    if (!empty($_POST['Login']) && !empty($_POST['Mdp'])){

        $LoginH = md5(trim($_POST['Login'])); //On Hash notre Login en ayant enlevé les espaces et
les caractères bizarres en début et fin de chaîne
        $MdpH   = md5(trim($_POST['Mdp'])); // On lui applique le même traitement

        /* Connexion à la base de données */
        $Conex = mysql_connect('localhost', 'root', '');
        mysql_select_db('BruteForceFichier', $Conex);

        /* On va pouvoir faire une requête pour vérifier la correspondance de nos informations */
        $Sql    = mysql_query('SELECT IdMembre, MotDePasse FROM Membres WHERE
LoginHasher="'. $LoginH. "'");
```

Function VerificationQuotas

```
/* Si on a qu'un seul résultat et pas d'erreur Sql on peut donc continuer */
if(mysql_num_rows($Sql) == 1 && !mysql_error()){

    $LesInformations = mysql_fetch_array($Sql);
    $SqlResultMdp     = $LesInformations['MotDePasse'];
    $SqlResultId      = $LesInformations['IdMembre'];

    /* Tout d'abord on va ouvrir notre fichier de quotas pour voir si la personne ne l'a
    pas dépassé
    * SI c'est le membre d'id 1 et que $CheminFichierQuotas = "BruteForce"
    * $CheminFichierQuotas.'/'.$SqlResultId.'.Bf' vaudra :
    * 'BruteForce/1.Bf'
    */

    /* D'abord on va s'assurer que le fichier existe si ce n'est pas le cas
    * on va le créer avec la valeur 0 pour les tentatives
    */
    if(!file_exists($CheminFichierQuotas.'/'.$SqlResultId.'.Bf')){
        WriteQuotasFile($CheminFichierQuotas,0,$SqlResultId);
    }
    /* Si il existe on va vérifier que la personne n'est pas hors quotas */
    else{

        /* On récupère nos informations */
        $FichierQuotas = fopen($CheminFichierQuotas.'/'.$SqlResultId.'.Bf', 'r+');
        $InformationQuotas = fgets($FichierQuotas,4096);
        fclose($FichierQuotas);

        /* On sépare nos valeurs */
        $InformationQuotas = explode('-SEPARATEUR-', $InformationQuotas);

        /* Si ce n'est pas la date d'aujourd'hui on remet le compteur à 0 et on affecte
        nous même les valeurs */
        if($InformationQuotas[1] != date('y-m-d')){
            WriteQuotasFile($CheminFichierQuotas,0,$SqlResultId);
            $InformationQuotas[0] = 0;
        }

        /* On regarde si le quotas n'est pas atteint on pourra donc vérifier nos
        informations */
        if($Quotas > $InformationQuotas[0]){
            /* Donc maintenant on peut vérifier si c'est le bon passe ou non */
            if($SqlResultMdp == $MdpH)
                return TRUE;
            else{
                /* On va donc ajouter +1 notre valeur de quotas et réécrire notre fichier
                */
                WriteQuotasFile($CheminFichierQuotas,($InformationQuotas[0]+1),$SqlResultId);
            }
            else
                return FALSE;
        }
    }
    else
        return FALSE;
}

/* Cette fonction nous permettra d'écrire dans notre fichier de log pour les quotas
```

Function VerificationQuotas

```
* Le seul argument correspond au quota atteint par la personne
*/
function WriteQuotasFile($CheminFichierQuotas,$QuotasAtteint,$SqlResultId){
    $FichierQuotas = fopen($CheminFichierQuotas.'/'.$SqlResultId.'.Bf', 'w+');
    /* On écrit donc dans notre fichier */
    fwrite($FichierQuotas,$QuotasAtteint.'-SEPARATEUR-'.date('y-m-d'));
    fclose($FichierQuotas);
}
?>
```

Ensuite il va nous falloir réaliser un fichier en .htaccess pour protéger le répertoire où seront stockés les fichiers pour l'anti "Brute Force". Ce fichier est à placer dans le répertoire BruteForce lui même.

Le fichier .htaccess

```
# Interdiction d'accéder au contenu du répertoire sauf via le serveur et par FTP
Deny from all
```

3.3 - Explications sur le code

Je vais éclaircir plusieurs points qui pourraient paraître un peu obscurs dans les sources ci-dessus. Tout d'abord pourquoi utiliser la fonction date() et non pas un time() ? Tout simplement parce que la fonction date() dans notre cas sera plus efficace et moins gourmande en ressource, et que dans quelques temps la fonction time() connaîtra des soucis car elle arrivera en fin de course. Ensuite, il faut utiliser un séparateur dans le fichier pour nous permettre de retrouver nos éléments simplement, vous pouvez utiliser un séparateur qui vous est propre sauf le caractère "-" à cause de la date (Bien sûr si vous avez séparés vos éléments de date par un "-").

J'ai aussi utilisé une fonction pour créer le fichier et le modifier si besoin mais pourquoi ? Car il serait maladroit de réécrire tous le temps le même bout de code alors qu'en faisant une fonction on allège notre code et sa relecture. Note importante lors du hashage des informations transmises par l'internaute, il faut obligatoirement faire votre hashage comme il a été opéré lors de l'enregistrement, c'est à dire que si vous avez par exemple mis les chaînes de mot de passe et de login en minuscule n'oubliez pas de le faire car le md5 est sensible à la casse.

3.4 - Conclusion et évolution possible


Il est possible de faire évoluer cet anti "Brute Force", de plusieurs manières, d'abord en en faisant un objet, ce que j'ai évité de faire pour ne pas embrouiller certaines personnes. Il est aussi possible de le modifier pour lui permettre d'afficher le nombre d'essai qu'il reste avant que le compte soit bloqué. Cependant je ne suis pas familier de ce genre de message. Car le but est de ne pas dévoiler à l'attaquant que le quotas est atteint; ainsi il continuera à valider le formulaire. Mais ayant atteint le quotas, même si il entre le bon mot de passe, il aura une réponse négative. De ce fait l'attaquant peut essayer un mot de passe valide sans le savoir et continuera des siècles durant.

Pour conclure comme tout en informatique cette méthode est appelée à évoluer mais sera très pratique pour des sites de petites envergures, pour des sites de plus grande taille je conseille d'utiliser la seconde méthode que je vais aborder juste après.

4 - Anti "Brute Force" par base de données

4.1 - Conception des besoins et choix de la logique

Nous allons utiliser la même structure de base de données que l'anti "Brute Force" par fichier en ajoutant plusieurs champs à la table "Membres". Le contexte et la logique de protection seront les mêmes pour bien vous montrer que la seule différence qui existe entre ces deux méthodes est la façon de gérer le contrôle. En aucun cas il n'est préférable de toujours utiliser l'une ou l'autre des méthodes. Pour des sites à forte audience un système par fichier risquerait d'engorger le serveur avec un grand nombre de fichiers.

Car avec un nombre trop important de fichiers il se peut que votre serveur fragmente ces fichiers et se ralentisse. De plus vous risquez un  "Denial of Service" par des hackers qui créeraient une multitude de compte juste pour pouvoir engendrer ce genre de fichier et rendre le serveur inactif.

4.2 - Réalisation de l'anti "Brute Force"

Il va falloir modifier quelque peu notre table "Membres" pour pouvoir nous permettre de gérer, ce système. Nous allons rajouter un champ "DateBf" qui nous servira de repère temporel et nous permettra ainsi de faire des contrôles journaliers. Ensuite nous allons ajouter le champ "NombreAttaques", qui aura pour but de recenser le nombre d'attaques de la journée. Ces deux nouveaux champs auront des valeurs par défaut pour les initialiser.

Table Membres	
IdMembre	INT(11)
Login	VARCHAR(40)
LoginHasher	VARCHAR(32)
MotDePasse	VARCHAR(32)
DateBf	VARCHAR(8) Valeur de défaut : 00-00-00
NombreAttaques	INT(2) Valeur de défaut : 0

Une fois les modifications effectuées nous allons modifier certaines choses sur le code en lui même car une très grande partie sera réutilisée, par rapport à l'anti "Brute Force" par fichier.

Page index.php

```
<?php

/* Ici on inclue notre Script ou se trouve notre fonction */
require_once('AntiBruteForceFichier.php');

/* Ce sera notre page d'index pour notre exemple.
 * Dans cette page se trouve le formulaire de connexion et les traitements de ce dernier y compris
 * l'anti brute force
 *
 */
/* J'ai juste supprimé le chemin vers le répertoire BruteForce car ici il ne servira à rien.*/

if(VerificationQuotas()){

    echo 'Vous êtes bien identifié.';

}
else{
    /*
    * Formulaire de connexion
    * Avec juste un champ pour le login et un autre pour le mot de passe
    *
    */
}
```

Page index.php

```
echo '  
    <form action="" method="post">  
    <label for="Log">Login : </label><input type="text" name="Login" value="" size="40"  
id="Log"/><br />  
    <label for="Mdp">Mot de passe : </label><input type="password" name="Mdp" value=""  
size="40" id="Mdp"/><br />  
    <input type="submit" name="Send" value="Connexion" />  
    </form>';  
}  
?>
```

Page de la fonction VerificationQuotas

```
<?php  
  
/* Ce script ne comportera que la fonction pour la vérification  
 * Cette fonction servira aussi pour faire l'authentification  
 */  
  
function VerificationQuotas($Quotas = 20){  
  
    /* On va vérifier qu'aucun des deux champs n'est vide */  
    if (!empty($_POST['Login']) && !empty($_POST['Mdp'])) {  
  
        $LoginH = md5(trim($_POST['Login'])); //On Hash notre Login en ayant enlevé les espaces et  
les caractères bizarres en début et fin de chaîne  
        $MdpH = md5(trim($_POST['Mdp'])); // On lui applique le même traitement  
  
        /* Connexion à la base de données */  
        $Conex = mysql_connect('localhost', 'root', '');  
        mysql_select_db('BruteForcebdd', $Conex);  
  
        /* Ici le code change légèrement au niveau de la requête nous allons récupérer les  
informations concernant la date et nombre d'attaques */  
        $Sql = mysql_query('SELECT IdMembre, MotDePasse, DateBf, NombreAttaques FROM Membres WHERE  
LoginHasher="'.$LoginH.'");  
  
        /* Si on a qu'un seul résultat et pas d'erreur Sql on peut donc continuer */  
        if(mysql_num_rows($Sql) == 1 && !mysql_error()){  
  
            $LesInformations = mysql_fetch_array($Sql);  
  
            /* On va récupérer les informations concernant le membre son id la date de la dernière  
tentative ratée d'authentification  
 * ainsi que son Id et le nombre d'attaques.  
 */  
            $SqlResultMdp = $LesInformations['MotDePasse'];  
            $SqlResultId = $LesInformations['IdMembre'];  
            $SqlResultDateBf = $LesInformations['DateBf'];  
            $SqlResultNombreAttaques = $LesInformations['NombreAttaques'];  
  
            $DateDuJour = date('y-m-d');  
  
            /* La première chose à faire après avoir récupérer ces informations est de vérifier si  
le quota n'est pas atteint pour aujourd'hui  
 * SI Le nombre d'attaques enregistré est inférieur pour la date d'aujourd'hui OU si la  
date est différente de la date d'aujourd'hui ALORS  
 * En regarde la correspondance entre les mots de passe  
 * SINON  
 * Le quota est atteint pour la journée.  
 */  
  
            if ( ( $SqlResultNombreAttaques < $Quotas && $SqlResultDateBf == $DateDuJour ) || (  
$SqlResultDateBf != $DateDuJour ) ) {  
                if($SqlResultMdp == $MdpH)
```

Page de la fonction VerificationQuotas

```
        return TRUE;// La personne a fourni les bonnes informations on laisse donc
passer.
        else //Ce n'est pas bon on va donc modifier la valeur dans la base de données
            @mysql_query('UPDATE Membres SET
NombreAttaques='.($SqlResultNombreAttaques+1).', DateBf="'.$DateDuJour.'" WHERE
Id_Membre='. $SqlResultId);
            }
            else // Cela signifie que l'on a atteint le quota
                return FALSE;
        }
        else
            return FALSE;
    }
    else
        return FALSE;
}
?>
```

4.3 - Explication sur le code source et les modifications apportées.

Vous aurez pu le constater la fonction `VerificationQuotas()`; est bien plus courte qu'avec une gestion par fichier. Pour plusieurs raisons, tout d'abord nous récupérons toutes les informations dès la première requête et donc en cas de succès de cette dernière nous pouvons directement, sans autre traitement, nous occuper de la vérification des quotas pour la date du jour. Ensuite en cas d'échec de connexion on fait une simple requête de mise à jour mais uniquement si on a pas dépassé le nombre de quotas. Sinon on aurait un risque d'engorgement de requêtes SQL.

Cet exemple est de loin celui qui parait le plus simple à comprendre mais il arrivera que nous ne puissions pas mettre en place ce genre de système, et que nous devons donc nous tourner vers un système par fichiers.

4.4 - Conclusion et évolution possible

Comme pour le système anti "Brute Force" par fichier il est possible d'améliorer ce système. On pourrait donc facilement ajouter un quota de Date où le compte aurait été bloqué. Si par exemple le compte s'est retrouvé bloqué trois fois, il est préférable de le bloquer et d'avertir le possesseur du compte ainsi que l'administrateur du site. Il est aussi possible bien entendu d'utiliser les autres logiques de protection ou de les combiner.

5 - Bibliographie

Voici deux tutoriels que je vous conseil pour optimiser vos protections, mais surtout il ne faut pas hésiter à en lire de nombreux autres.

Bibliographie

-  [Faq Captcha](#) (Developpez.com)
-  [Tutoriel de sécurité général](#) (Developpez.com)

Vous pouvez aussi utiliser des noms de champs dynamique pour votre connexion, en stockant leurs nom dans des variables de sessions. Explication fournis par Yogui (qui a soumis l'idée ASHA - il comprendra -) :

Par exemple à l'affichage de la page, tu donnes un nom aléatoire à chacun des "input" du formulaire de login (login + mdp). Tu gardes ces valeurs en session afin de pouvoir les retrouver lors de la réception des données de l'internaute, et l'avantage est que le nom des champs change à chaque chargement de la page, une attaque force brut est donc bien plus coûteuse (il faut télécharger le formulaire avant de l'envoyer, ce qui complique considérablement la tâche voire augmente la quantité de données transférées, et donc le temps passé à essayer de pirater le site).

6 - Conclusion & Remerciements

6.1 - Remerciement

Je tiens à remercier en premier lieu la personne qui m'a autorisé à écrire un tutoriel pour Developpez.com : M Guillaume Rossolini. Ensuite je voudrais remercier les personnes qui ont regardé et corrigé la rédaction, mise en page, orthographe, critique du code et autres éléments Merci à eux car sans eux ce tutoriel n'aurait pas cette qualité.

Les correcteurs

- Anthony.Desvernois
- Yoshio
- Yogui
- Xave
- Et ma correctrice attiré ;)

6.2 - Conclusion

Pour conclure avec ce tutoriel je n'ajouterai rien de spécial, simplement le rappel que seul un anti "Brute Force" n'est pas une sécurité. Il ne faut négliger aucun de vos aspects sur cette sécurité notamment sur l'accès à des répertoires ou autres. Ce tutoriel est une des briques qui constitue un faible mur contre les attaques de toutes sortes.

