Comment créer un script de connexion sécurisée avec PHP et MySQL

Alors qu'on entend parler de plus en plus aux informations de pirates informatiques, les développeurs essayent de trouver les meilleures façons de sécuriser leurs sites internet. Si votre site possède une zone membre, il existe un risque de piratage et les données de vos membres pourraient être volées. Ce guide vous expliquera comment créer un script PHP sécurisé d'ouverture de session. Le code n'est pas parfait, mais est aussi bon que possible, la sécurité sur le net et le cryptage sont des sujets complexes qui changent tout le temps, et personne ne peut se vanter de tout connaître sur le sujet. C'est pourquoi il est possible que certaines astuces nous aient échappées dans le code présenté ci-dessous. Si tel est le cas, nous vous remercions de nous le faire savoir et nous essayerons d'améliorer cet article. En suivant les étapes de ce guide, vous préviendrez nombre d'attaques dont les hackeurs se servent pour prendre le contrôle des comptes de vos membres, pour effacer des comptes et/ou pour en changer des données. Voici une liste des attaques que le code de cet article essaye d'empêcher :

- Les injections SQL
- Les session hijacking
- Les network eavesdropping
- Les cross site scripting
- Les attaques par brute force

Le code utilisé dans ce guide est un mélange de filtrage de données, de cryptage et d'autres méthodes qui vont rendre la tâche plus difficile au petit malin qui veut pirater votre site.

Nous essayons en permanence d'améliorer ce script. La dernière version mise à jour du code est disponible sur github. Il est possible que vous trouviez des différences entre le code que vous pouvez télécharger depuis ce site et le code présenté dans cet article. Prenez aussi en compte le fait que nous n'avons fait aucun effort pour rendre le code HTML plus joli sur la page web.

Vous remarquerez aussi que nous ne fermons pas les balises PHP dans les morceaux de code qui ne contiennent que du PHP, la raison étant que nous suivons les recommandations générales de mise en forme.

Pour finir, vous devez savoir qu'il vous faut créer tous les fichiers de l'application qui ne sont pas du HTML dans plusieurs dossiers à l'intérieur de la racine de l'application. La façon la plus simple d'avoir une structure de dossier correcte est de télécharger la dernière mise à jour du code en cliquant sur l'un des liens ci-dessus.

N'hésitez pas à utiliser cette application comme base de votre propre code, mais ne vous en servez pas comme d'un exemple de bonnes pratiques de codage !

Éléments nécessaires

Puisque nous allons nous servir du set de classes PHP mysqli_* pour accéder à la base de données MySQL, vous allez avoir besoin des versions suivantes de PHP et MySQL.

- PHP version 5.3 ou plus
- MySQL version 4.1.3 ou plus

Bien entendu, vous allez aussi avoir besoin d'un serveur web configuré pour faire tourner PHP, afin d'y héberger vos pages internet. Il est fort probable que vous vous serviez du serveur web de votre hébergeur, à moins que vous n'hébergiez votre site internet vous-même.

Si vous souhaitez vérifier la version de PHP sur votre serveur, utilisez la fonction phpinfo();.

Partie 1 sur 8: Configurez votre serveur

1 Installez un serveur web avec PHP et MySQL.

La plupart des services d'hébergement auront déjà PHP et MySQL installés. Il vous faudra simplement vérifier que les versions installées sont bien les versions dont vous avez besoin pour ce guide. Si vous n'avez pas au moins PHP5.3 et MySQL5, vous pourriez vous poser des questions sur le souci que votre hébergeur porte à votre sécurité. Mettre vos logiciels à jour fait aussi partie des mesures de sécurité que vous devez prendre.

Si vous possédez votre propre serveur ou ordinateur, vous devez installer les logiciels en suivant les étapes nécessaires pour votre système d'exploitation. En règle générale, si vous n'allez pas configurer le logiciel pour le mettre en ligne et que vous développez sous Windows, Mac ou Linux, vous pourriez essayer d'installer XAMPP. Trouvez la version nécessaire pour votre système sur :

http://www.apachefriends.org/en/xampp.html

Mais souvenez-vous que vous ne devez jamais vous servir de XAMPP pour mettre votre site internet en ligne.

Si vous utilisez Linux, ouvrez le package manager pour télécharger et installer les packages nécessaires. Pour certaines distributions, comme Ubuntu, les packages sont regroupés en un seul paquet. Tapez le code suivant dans la console d'Ubuntu :

```
sudo apt-get install lamp-server^ phpmyadmin
```

Quelle que soit la façon dont vous installez les éléments nécessaires, assurez-vous de configurer MySQL avec un mot de passe sécurisé.

Partie 2 sur 8: Configurez la base de données MySQL

Créez la base de données MySQL.

1 Connectez-vous à votre base de données en tant qu'administrateur (en général à la racine).

Pour ce guide, nous allons créer une base de données qui s'appellera "secure_login".

Pour créer la base de données, vous pouvez vous servir du code ci-dessous ou faire la même chose manuellement dans phpMyAdmin ou votre client MySQL préféré, comme vous préférez.

```
CREATE DATABASE `secure_login`;
```

Notez que certains hébergeurs ne vous laissent pas créer de base de données au travers de phpMyAdmin, apprenez à le faire au travers de cPanel.

2 Créez un utilisateur avec des privilèges pour SELECT, UPDATE et INSERT.

Lorsque vous restreignez les privilèges d'un utilisateur, vous empêchez un hackeur qui aurait trouvé une faille de sécurité dans votre script d'effacer quoi que ce soit de votre base de données. En vous servant de ces privilèges, vous pouvez arriver à faire à peu près tout ce que vous voulez avec votre application. Si vous êtes vraiment parano, créez un utilisateur différent pour chaque fonction.

Il va de soi que vous devez être connecté dans MySQL en tant qu'utilisateur avec les privilèges requis pour créer un utilisateur. Cet utilisateur sera en général le premier.

Voici les détails de l'utilisateur que nous avons créé :

- **Utilisateur :** "sec_user"
- **Mot de passe :** "eKcGZr59zAa2BEWU"

Notez que vous devriez changer le mot de passe et ne pas utiliser celui donné ci-dessus lorsque vous mettez votre script en ligne. Si vous choisissez cette solution, assurez-vous de changer aussi le code ci-dessous, et le code de connexion à la base de données que nous avons créé.

Souvenez-vous que vous n'avez pas besoin de mettre un mot de passe dont vous pouvez vous souvenir, alors rendez-le aussi compliqué que possible. Voici un générateur de mots de passe. Ci-dessous vous trouverez le code SQL qui crée l'utilisateur dans la base de données et lui donne les privilèges nécessaires. Vous pouvez aussi faire la même chose dans votre client GUI de base de données comme phpMyAdmin :

```
CREATE USER 'sec_user'@'localhost' IDENTIFIED BY 'eKcGZr59zAa2BEWU';
GRANT SELECT, INSERT, UPDATE ON `secure_login`.* TO 'sec_user'@'localhost';
```

Si votre code donne la possibilité d'effacer des données de la base de données, vous devez ajouter DELETE à la liste des privilèges, ou alors vous pourriez créer un autre utilisateur avec DELETE comme seul privilège, et seulement sur la table où il sera autorisé à effacer des données si vous ne voulez pas qu'il en efface des deux. Vous n'avez pas besoin de donner la permission d'effacer quoi que ce soit pour l'exemple ci-dessous.

3 Créez une table MySQL que vous appellerez "members".

Le code ci-dessous crée une table avec cinq champs (id, username, email, password, salt). Nous utilisons la valeur CHAR pour les champs dont nous savons la longueur, les champs "password" et "salt" auront toujours une longueur de 128 caractères. Le type CHAR permet d'économiser de la puissance de calcul :

```
CREATE TABLE `secure_login`.`members` (
    `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `username` VARCHAR(30) NOT NULL,
    `email` VARCHAR(50) NOT NULL,
    `password` CHAR(128) NOT NULL,
    `salt` CHAR(128) NOT NULL
) ENGINE = InnoDB;
```

Comme cité précédemment, vous pouvez effectuer la même opération avec n'importe quel client de votre choix.

4 Créez une table pour y enregistrer les essais d'ouverture de session.

Nous allons nous servir de cette table pour enregistrer les essais d'ouverture de session effectués par l'utilisateur. De cette façon nous rendrons les attaques par brute force plus difficile :

```
CREATE TABLE `secure_login`.`login_attempts` (
    `user_id` INT(11) NOT NULL,
    `time` VARCHAR(30) NOT NULL
) ENGINE=InnoDB
```

5 Créez une ligne de test dans la table "members".

Il est important de pouvoir tester votre script de connexion, voici donc ci-dessous le script pour créer un tel utilisateur :

- **Username:** test_user
- **Email:** test@example.com
- **Password:** 6ZaxN2Vzm9NUJT2y

Et le code dont vous avez besoin pour vous connecter en tant qu'utilisateur :

```
INSERT INTO `secure_login`.`members` VALUES(1, 'test_user', 'test@example.com',
'00807432eae173f652f2064bdca1b61b290b52d40e429a7d295d76a71084aa96c0233b82f1feac
'f9aab579fc1b41ed0c44fe4ecdbfcdb4cb99b9023abb241a6db833288f4eea3c02f76e0d35204a
```

Partie 3 sur 8: Créez la page de connexion à la base de données

1 Créez la page de configuration globale

Créez un dossier que vous appellerez "includes" dans la racine de l'application, puis créez un fichier PHP dans ce dossier. Appelez-le psl-config.php. Si vous souhaitez mettre votre script en ligne, vous pourriez placer ce fichier ainsi que tous les autres fichiers include en dehors de la racine du serveur. Si vous le faites, et nous vous conseillons vivement de le faire, il vous faudra modifier vos appels aux instructions include et require pour que l'application puisse localiser les fichiers.

Un emplacement différent de vos fichiers include en dehors de la racine du serveur signifie que vous ne pouvez pas les afficher en vous servant de leur URL. De cette façon, si quelqu'un oublie l'extension php, ou trifouille les permissions de fichiers, il ne pourra tout de même pas l'afficher en tant que fichier texte.

Ce fichier contient des variables de configuration globales. Vous pouvez y mettre des choses comme savoir qui peut s'enregistrer, si c'est une connexion sécurisée (HTTPS), et plein d'autres choses...

```
<?php
/**
 * Voici les détails de connexion à la base de données
 */
define("HOST", "localhost");     // L'hébergeur où vous voulez vous connecter.
define("USER", "sec_user");    // Le nom d'utilisateur de la base de données.
define("PASSWORD", "4Fa98xkHVd2XmnfK");    // Le mot de passe de la base de don
define("DATABASE", "secure_login");    // Le nom de la base de données.

define("CAN_REGISTER", "any");
define("DEFAULT_ROLE", "member");

define("SECURE", FALSE);    // SEULEMENT DANS LE CADRE DE CE GUIDE!!!!
```

2 Créez la page de connexion à la base de données

Voici le code PHP dont nous allons nous servir pour nous connecter à la base de données MySQL. Créez un nouveau fichier php que vous appellerez db_connect.php dans le dossier où se trouvent les includes de l'application et ajoutez-y le code ci-dessous. Vous pouvez ensuite inclure le fichier dans n'importe quel page où vous avez besoin d'un accès à la base de données.

```
<?php
include_once 'psl-config.php';   // Car functions.php n'est pas inclus
$mysqli = new mysqli(HOST, USER, PASSWORD, DATABASE);
```

Partie 4 sur 8: Créez les fonctions en PHP

Ce sont les fonctions qui vont faire tourner votre script d'ouverture de session. Ajoutez toutes ces fonctions dans un fichier que vous appellerez functions.php dans le dossier des includes de l'application.

1 Démarrez une session PHP en toute sécurité.

Les sessions PHP ont la réputation de ne pas être sécurisées, c'est pourquoi il est important de ne pas seulement ajouter "session_start();" en haut de chaque page si vous voulez créer une session PHP. Nous allons créer une fonction que nous allons appeler "sec_session_start()", avec pour but de créer une session PHP sécurisée. Vous devez appeler cette fonction en haut de chaque page si vous souhaitez accéder à une variable de session PHP. Si vous vous faites beaucoup de souci à propos de la sécurité et de l'accessibilité de vos cookies, faites des recherches pour trouver des systèmes de gestion de session sécurisée en PHP et MySQL.

Cette fonction va grandement améliorer la sécurité de votre script. Elle permet d'éviter que les hackeurs aient accès au cookie qui stocke l'id de session avec JavaScript (par exemple lors d'une attaque XSS). Tout comme la fonction "session_regenerate_id()" qui génère un nouveau mot de passe chaque fois que la page se recharge et permet d'éviter les session hijackings. Notez que si vous souhaitez utiliser le HTTPS dans votre script de connexion, vous devez déclarer la variable "$secure" true. Lorsque vous mettez le site en ligne, il est vital d'utiliser le HTTPS.

Créez un nouveau fichier que vous appellerez functions.php dans le dossier includes de votre application et copiez-y le code suivant :

```
<?php
include_once 'psl-config.php';

function sec_session_start() {
    $session_name = 'sec_session_id';   // Attribue un nom de session
```

```
    $secure = SECURE;
    // Cette variable empêche Javascript d'accéder à l'id de session
    $httponly = true;
    // Force la session à n'utiliser que les cookies
    if (ini_set('session.use_only_cookies', 1) === FALSE) {
        header("Location: ../error.php?err=Could not initiate a safe session (i
        exit();
    }
    // Récupère les paramètres actuels de cookies
    $cookieParams = session_get_cookie_params();
    session_set_cookie_params($cookieParams["lifetime"],
        $cookieParams["path"],
        $cookieParams["domain"],
        $secure,
        $httponly);
    // Donne à la session le nom configuré plus haut
    session_name($session_name);
    session_start();            // Démarre la session PHP
    session_regenerate_id();    // Génère une nouvelle session et efface la pré
}
```

2 Créez la fonction d'ouverture de session.

Cette fonction va comparer l'adresse email et le mot de passe avec ceux présents dans la base de données. Si ce sont les mêmes, true sera renvoyé. Ajoutez cette fonction à votre fichier functions.php :

```
function login($email, $password, $mysqli) {
    // L'utilisation de déclarations empêche les injections SQL
    if ($stmt = $mysqli->prepare("SELECT id, username, password, salt
        FROM members
       WHERE email = ?
        LIMIT 1")) {
        $stmt->bind_param('s', $email);  // Lie "$email" aux paramètres.
        $stmt->execute();    // Exécute la déclaration.
        $stmt->store_result();

        // Récupère les variables dans le résultat
        $stmt->bind_result($user_id, $username, $db_password, $salt);
        $stmt->fetch();

        // Hashe le mot de passe avec le salt unique
        $password = hash('sha512', $password . $salt);
        if ($stmt->num_rows == 1) {
            // Si l'utilisateur existe, le script vérifie qu'il n'est pas verro
            // à cause d'essais de connexion trop répétés

            if (checkbrute($user_id, $mysqli) == true) {
                // Le compte est verrouillé
                // Envoie un email à l'utilisateur l'informant que son compte e
                return false;
            } else {
                // Vérifie si les deux mots de passe sont les mêmes
                // Le mot de passe que l'utilisateur a donné.
                if ($db_password == $password) {
                    // Le mot de passe est correct!
                    // Récupère la chaîne user-agent de l'utilisateur
                    $user_browser = $_SERVER['HTTP_USER_AGENT'];
                    // Protection XSS car nous pourrions conserver cette valeur
                    $user_id = preg_replace("/[^0-9]+/", "", $user_id);
                    $_SESSION['user_id'] = $user_id;
                    // Protection XSS car nous pourrions conserver cette valeur
                    $username = preg_replace("/[^a-zA-Z0-9_\-]+/",
                                                        "",
                                                        $username);
                    $_SESSION['username'] = $username;
                    $_SESSION['login_string'] = hash('sha512',
                              $password . $user_browser);
                    // Ouverture de session réussie.
                    return true;
                } else {
                    // Le mot de passe n'est pas correct
                    // Nous enregistrons cet essai dans la base de données
                    $now = time();
                    $mysqli->query("INSERT INTO login_attempts(user_id, time)
                                    VALUES ('$user_id', '$now')");
                    return false;
                }
            }
        } else {
```

```
            // L'utilisateur n'existe pas.
            return false;
        }
    }
}
```

3 La fonction de brute force.

Une attaque de brute force se produit lorsqu'un hackeur essaye des milliers de mots de passe différents sur votre compte, soit générés à la volée ou tirés d'un dictionnaire. Dans le script que nous créons, le compte se verrouille si l'utilisateur essaye de se connecter sans succès plus de 5 fois.

Il est difficile de se prémunir face aux attaques de brute force. Une des façons dont vous pouvez vous protéger est en utilisant un test de CAPTCHA, en verrouillant les comptes d'utilisateurs et en ajoutant un délai entre chaque échec de connexion, par exemple en n'autorisant pas d'autres essais d'ouverture de session avant 30 secondes.

Nous vous conseillons vivement d'utiliser les CAPTCHA. Pour l'instant nous n'avons pas implémenté cette fonction dans le script, mais nous espérons pouvoir le faire dans un futur proche, en nous servant d'une image sécurisée, puisque vous n'avez pas besoin de vous enregistrer. Vous pourriez aussi essayer quelque chose de plus connu comme le php reCAPTCHA de Google.

Quelle que soit la solution que vous choisissiez, nous vous conseillons de ne faire apparaître l'image CAPTCHA qu'après deux essais infructueux pour éviter de déranger l'utilisateur inutilement.

Lorsqu'ils se retrouvent confrontés au problème des attaques par brute force, la plupart des développeurs PHP bloquent l'adresse IP après un certain nombre d'essais de connexion. Mais il existe de nombreux outils qui permettent d'automatiser ce genre d'attaques, et ces mêmes outils peuvent utiliser différents proxis et même changer d'adresse IP à chaque tentative de connexion. Le blocage d'adresses IP mène au blocage d'utilisateurs légitimes. Dans notre script, nous enregistrons les essais de connexions et verrouillons le compte après cinq essais. Cela déclenchera aussi l'envoi d'un email à l'utilisateur avec un lien de déverrouillage, mais nous n'avons pas présenté ce code ici. Voici le code de la fonction checkbrute(), ajoutez-la à votre fichier functions.php :

```
function checkbrute($user_id, $mysqli) {
    // Récupère le timestamp actuel
    $now = time();

    // Tous les essais de connexion sont répertoriés pour les 2 dernières heures
    $valid_attempts = $now - (2 * 60 * 60);

    if ($stmt = $mysqli->prepare("SELECT time
                             FROM login_attempts <code><pre>
                             WHERE user_id = ?
                            AND time > '$valid_attempts'")) {
        $stmt->bind_param('i', $user_id);

        // Exécute la déclaration.
        $stmt->execute();
        $stmt->store_result();

        // S'il y a eu plus de 5 essais de connexion
        if ($stmt->num_rows > 5) {
            return true;
        } else {
            return false;
        }
    }
}
```

Vérifiez le statut de la session.

4 Nous procédons en vérifiant les variables de session "user_id" et "login_string". La variable de session "login_string" contient les informations du navigateur de l'utilisateur hashées avec le mot de passe. Nous utilisons les informations du navigateur, car il y a peu de chances que l'utilisateur change de navigateur en plein milieu de la session. Cette mesure aide à éviter les session hijackings. Ajoutez cette fonction à votre fichier functions.php :

```
function login_check($mysqli) {
    // Vérifie que toutes les variables de session sont mises en place
    if (isset($_SESSION['user_id'],
                        $_SESSION['username'],
                        $_SESSION['login_string'])) {

        $user_id = $_SESSION['user_id'];
        $login_string = $_SESSION['login_string'];
        $username = $_SESSION['username'];

        // Récupère la chaîne user-agent de l'utilisateur
        $user_browser = $_SERVER['HTTP_USER_AGENT'];

        if ($stmt = $mysqli->prepare("SELECT password
                                      FROM members
                                      WHERE id = ? LIMIT 1")) {
            // Lie "$user_id" aux paramètres.
            $stmt->bind_param('i', $user_id);
            $stmt->execute();   // Exécute la déclaration.
            $stmt->store_result();

            if ($stmt->num_rows == 1) {
                // Si l'utilisateur existe, récupère les variables dans le résu
                $stmt->bind_result($password);
                $stmt->fetch();
                $login_check = hash('sha512', $password . $user_browser);

                if ($login_check == $login_string) {
                    // Connecté!!!!
                    return true;
                } else {
                    // Pas connecté
                    return false;
                }
            } else {
                // Pas connecté
                return false;
            }
        } else {
            // Pas connecté
            return false;
        }
    } else {
        // Pas connecté
        return false;
    }
}
```

5 Nettoyez l'URL de PHP_SELF

La fonction suivante nettoie le résultat obtenu par la variable d'environnement PHP_SELF.

C'est en fait la modification d'une fonction du même nom utilisée par le système de gestion de contenu de WordPress :

```
function esc_url($url) {

    if ('' == $url) {
        return $url;
    }

    $url = preg_replace('|[^a-z0-9-~+_.?#=!&;,/:%@$\|*\'()\\x80-\\xff]|i', '', $

    $strip = array('%0d', '%0a', '%0D', '%0A');
    $url = (string) $url;

    $count = 1;
```

```
    while ($count) {
        $url = str_replace($strip, '', $url, $count);
    }

    $url = str_replace(';//', '://', $url);

    $url = htmlentities($url);

    $url = str_replace('&amp;', '&#038;', $url);
    $url = str_replace("'", '&#039;', $url);

    if ($url[0] !== '/') {
        // Nous ne voulons que les liens relatifs de $_SERVER['PHP_SELF']
        return '';
    } else {
        return $url;
    }
}
```

Le souci lorsque vous utilisez une variable de serveur sans la filtrer est qu'elle peut être utilisée lors d'une attaque XSS. La plupart des tutoriels vous diront de la filtrer avec htmlentities(), cependant, même cette approche pourrait ne pas suffire, c'est pourquoi nous avons recours à plusieurs méthodes pour cette fonction.

D'autres développeurs conseillent de laisser l'attribut action du formulaire vide, ou avec une valeur NULL. Mais en faisant cela, vous laissez le formulaire ouvert à une attaque d'iframe clickjacking.

Partie 5 sur 8: Créez les pages de traitement

1 Créez la page de traitement de la connexion (process_login.php)

Créez un fichier qui va s'occuper de traiter les connexions, et que vous appellerez process_login.php, dans le dossier includes de l'application. Vous le mettrez dans ce dossier, car il ne contient pas de code HTML.

Nous allons nous servir des fonctions PHP du set de mysqli_* car elles font partie des extensions MySQL qui sont mises à jour le plus souvent.

```
<?php
include_once 'db_connect.php';
include_once 'functions.php';

sec_session_start(); // Notre façon personnalisée de démarrer la session PHP

if (isset($_POST['email'], $_POST['p'])) {
    $email = $_POST['email'];
    $password = $_POST['p']; // Le mot de passe hashé.

    if (login($email, $password, $mysqli) == true) {
        // Connecté
        header('Location: ../protected_page.php');
    } else {
        // Pas connecté
        header('Location: ../index.php?error=1');
    }
} else {
    // Les variables POST correctes n'ont pas été envoyées à cette page
    echo 'Invalid Request';
}
```

2 Créez le script de déconnexion.

Votre script de déconnexion doit pouvoir démarrer la session, la détruire et rediriger vers

une autre page. Notez que vous pourriez aussi ajouter une protection CSRF au cas où quelqu'un trouverait une façon d'envoyer un lien sur cette page. Si vous souhaitez plus d'informations sur le CSRF, rendez-vous sur Coding Horror.

Voici le script de déconnexion de l'utilisateur, que vous devez ajouter au fichier logout.php placé dans le dossier includes de l'application :

```
<?php
include_once 'includes/functions.php';
sec_session_start();

// Détruisez les variables de session
$_SESSION = array();

// Retournez les paramètres de session
$params = session_get_cookie_params();

// Effacez le cookie.
setcookie(session_name(),
        '', time() - 42000,
        $params["path"],
        $params["domain"],
        $params["secure"],
        $params["httponly"]);

// Détruisez la session
session_destroy();
header('Location: ../ index.php');
```

3 La page d'enregistrement.

Le code pour la page d'enregistrement se retrouvera dans deux fichiers, un que vous appellerez register.php dans la racine de l'application et un autre que vous appellerez register.inc. php dans le dossier includes. Il prend en charge les actions suivantes :

- Récupère et valide le nom d'utilisateur que l'utilisateur a choisi.
- Récupère et valide l'adresse email que l'utilisateur a entrée.
- Hashe le mot de passe et le renvoie à la page register.php (c'est-à-dire qu'elle va se le poster à elle-même)

La plupart du processus de validation se fera avec du JavaScript, côté client. Simplement car l'utilisateur n'a aucune raison de contourner ces vérifications. Pourquoi un utilisateur voudrait créer un compte qui serait moins sécurisé ? Nous parlerons du JavaScript dans la partie suivante.

Pour l'instant, créez simplement un fichier register.php et ajoutez-y le code suivant :

```
<?php
include_once 'includes/register.inc.php';
include_once 'includes/functions.php';
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Connexion sécurisée, formulaire d'enregistrement</title>
        <script type="text/JavaScript" src="js/sha512.js"></script>
        <script type="text/JavaScript" src="js/forms.js"></script>
        <link rel="stylesheet" href="styles/main.css" />
    </head>
    <body>
        <!-- S'il n'y a pas de variables POST ou si l'enregistrement a retourné
        <h1>Enregistrez-vous</h1>
        <?php
        if (!empty($error_msg)) {
            echo $error_msg;
        }
        ?>
        <ul>
            <li>Le nom d'utilisateur ne peut contenir que des nombres, des lettr
```

```
            <li>Les emails doivent avoir un format valide</li>
            <li>Le mot de passe doit avoir au moins 6 caractères</li>
            <li>Le mot de passe doit contenir
                <ul>
                    <li>Au moins une lettre majuscule (A..Z)</li>
                    <li>Au moins une lettre minuscule (a..z)</li>
                    <li>Au moins un chiffre (0..9)</li>
                </ul>
            </li>
            <li>Les deux mots de passe entrés doivent être exactement les mêmes
        </ul>
        <form action="<?php echo esc_url($_SERVER['PHP_SELF']); ?>"
                method="post"
                name="registration_form">
            Username: <input type='text'
                name='username'
                id='username' /><br>
            Adresse email: <input type="text" name="email" id="email" /><br>
            Mot de passe: <input type="password"
                            name="password"
                            id="password"/><br>
            Confirmez le mot de passe: <input type="password"
                                    name="confirmpwd"
                                    id="confirmpwd" /><br>
            <input type="button"
                   value="S'enregistrer"
                   onclick="return regformhash(this.form,
                                    this.form.username,
                                    this.form.email,
                                    this.form.password,
                                    this.form.confirmpwd);" />
        </form>
        <p>Retournez à la page d'<a href="index.php">inscription</a>.</p>
    </body>
</html>
```

Vous devez mettre le code suivant dans le fichier register.inc.php dans le dossier includes :

```
<?php
include_once 'db_connect.php';
include_once 'psl-config.php';

$error_msg = "";

if (isset($_POST['username'], $_POST['email'], $_POST['p'])) {
    // Nettoyez et validez les données transmises au script
    $username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
    $email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
    $email = filter_var($email, FILTER_VALIDATE_EMAIL);
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        // L'adresse email n'est pas valide
        $error_msg .= '<p class="error">L'adress email que vous avez entrée n'es
    }

    $password = filter_input(INPUT_POST, 'p', FILTER_SANITIZE_STRING);
    if (strlen($password) != 128) {
        // Le mot de passe hashé ne doit pas dépasser les 128 caractères
        // Si ce n'est aps le cas, quelque chose de vraiment bizarre s'est prod
        $error_msg .= '<p class="error">Mot de passe invalide.</p>';
    }

    // La forme du nom d'utilisateur et du mot de passe a été vérifiée côté cli
    // Cela devrait suffire, car personne ne tire avantage
    // à briser ce genre de règles.
    //

    $prep_stmt = "SELECT id FROM members WHERE email = ? LIMIT 1";
    $stmt = $mysqli->prepare($prep_stmt);

    if ($stmt) {
        $stmt->bind_param('s', $email);
        $stmt->execute();
        $stmt->store_result();

        if ($stmt->num_rows == 1) {
            // Il y a déjà un utilisateur avec ce nom-là
            $error_msg .= '<p class="error">Il existe déjà un utilisateur avec
        }
```

```
    } else {
        $error_msg .= '<p class="error">Erreur de base de données</p>';
    }

    // CE QUE VOUS DEVEZ FAIRE:
    // Nous devons aussi penser à la situation où l'utilisateur n'a pas
    // le droit de s'enregistrer, en vérifiant quel type d'utilisateur essaye de
    // s'enregistrer.

    if (empty($error_msg)) {
        // Crée un salt au hasard
        $random_salt = hash('sha512', uniqid(openssl_random_pseudo_bytes(16), TR

        // Crée le mot de passe en se servant du salt généré ci-dessus
        $password = hash('sha512', $password . $random_salt);

        // Enregistre le nouvel utilisateur dans la base de données
        if ($insert_stmt = $mysqli->prepare("INSERT INTO members (username, emai
            $insert_stmt->bind_param('ssss', $username, $email, $password, $rand
            // Exécute la déclaration.
            if (! $insert_stmt->execute()) {
                header('Location: ../error.php?err=Registration failure: INSERT
            }
        }
        header('Location: ./register_success.php');
    }
}
```

Si aucune donnée n'a été passée par la méthode POST, la page affichera le formulaire. Le bouton submit du formulaire appelle la fonction JavaScript regformhash(). Cette fonction vérifie les données et transmet le formulaire si tout est bon. La partie suivant aborde le sujet des fonctions JavaScript.

Veuillez noter que ces vérifications n'ont pas été terminées au moment de l'écriture de cet article. Certains des problèmes sont abordés dans les commentaires du code. Pour l'instant, nous vérifions simplement que l'adresse email est sous un format correct, que le mot de passe hashé a une longueur correcte et que l'utilisateur n'essaye pas d'utiliser une adresse email qui a déjà été enregistrée.

Si tout se passe bien, le nouvel utilisateur est enregistré et une nouvelle entrée a été ajoutée dans la table des membres.

Partie 6 sur 8: Créez les fichiers JavaScript

1 Créez le fichier sha512.js

Ce fichier est une implémentation en JavaScript de l'algorithme de hashage sha512. Nous allons nous en servir pour éviter que le mot de passe ne soit envoyé tel quel.

Vous pouvez télécharger le fichier sur pajhome.org.uk

(Vous pouvez aussi le trouver sur github)

Sauvegardez une copie de ce fichier dans un dossier que vous appellerez "js", en dehors de la racine de l'application.

2 Créez le fichier forms.js

Créez ce fichier dans le dossier js de l'application, il vous servira à hasher les mots de passe pour le formulaire de connexion (formhash()) et d'inscription (regformhash()) :

```
function formhash(form, password) {
    // Créez un nouvel élément input, il nous servira de champ pour le mot de pa
    var p = document.createElement("input");

    // Ajoutez le nouvel élément au formulaire.
    form.appendChild(p);
    p.name = "p";
    p.type = "hidden";
    p.value = hex_sha512(password.value);

    // Assurez-vous que le mot de passe sous sa forme de texte brut n'est pas en
    password.value = "";

    // Finissez en envoyant le formulaire.
    form.submit();
}

function regformhash(form, uid, email, password, conf) {
     // Vérifiez qu'aucun champ n'a été laissé vide
    if (uid.value == ''          ||
          email.value == ''      ||
          password.value == ''   ||
          conf.value == '') {

        alert('Vous devez fournir tous les détails nécessaires. Veuillez essaye
        return false;
    }

    // Vérifiez le nom d'utilisateur

    re = /^\w+$/;
    if(!re.test(form.username.value)) {
        alert("Le nom d'utilisateur ne doit contenir que des lettres, des chiff
        form.username.focus();
        return false;
    }

    // Vérifiez que le mot de passe a la longueur nécessaire (au moins 6 caracte
    // La même vérification est répétée ci-dessous, mais nous l'avons laissée pa
    // Guide pour l'utilisateur
    if (password.value.length < 6) {
        alert('Votre mot de passe doit avoir au moins 6 caractères.  Essayez de
        form.password.focus();
        return false;
    }

    // Au moins un chiffre, une lettre en minuscule et une lettre en majuscule
    // Au moins six caractères

    var re = /(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}/;
    if (!re.test(password.value)) {
        alert('Le mot de passe doit contenir au moins un chiffre, une lettre en
        return false;
    }

    // Vérifiez que les deux mots de passe sont les mêmes
    if (password.value != conf.value) {
        alert('Votre mot de passe et sa confirmation ne sont pas les mêmes. Essa
        form.password.focus();
        return false;
    }

    // Crée un nouvel élément input qui nous servira de champs pour notre mot de
    var p = document.createElement("input");

    // Ajoute le nouvel élément au formulaire.
    form.appendChild(p);
    p.name = "p";
    p.type = "hidden";
    p.value = hex_sha512(password.value);

    // Assurez-vous que le mot de passe sous sa forme de texte brut n'est pas en
    password.value = "";
    conf.value = "";

    // Finalement envoie le formulaire.
    form.submit();
    return true;
}
```

Dans les deux cas, le JavaScript hashe le mot de passe et l'envoie par la méthode POST en créant et en remplissant un champ du formulaire.

Partie 7 sur 8: Créez les pages HTML

1 Créez le formulaire de connexion (login.php).

Nous allons créer un formulaire avec deux champs, l'un d'eux appelé "adresse email" et l'autre "mot de passe". Le bouton submit appellera la fonction JavaScript formhash(), qui génèrera un hash du mot de passe, et enverra les valeurs des champs "adresse email" et "p" (le mot de passe hashé) au serveur. Vous devez créer ce fichier dans la racine de l'application.

Lors de la connexion, il est préférable de ne pas utiliser quelque chose de public, dans le cadre de ce guide nous utilisons l'adresse email en tant qu'id de connexion, le nom d'utilisateur peut être utilisé pour identifier l'utilisateur. Si l'adresse email n'est jamais affichée sur aucune des pages de l'application, cela ajoute un peu plus de sécurité pour se prémunir de quelqu'un qui essaiera de cracker le compte.

Même si nous avons crypté le mot de passe pour qu'il n'apparaisse pas sous forme de texte brut, nous vous conseillons toujours d'utiliser le protocole HTTPS (TLS/SSL) lorsque vous envoyez un mot de passe à un script. Nous ne pouvons qu'insister sur le fait qu'hasher le mot de passe n'est pas suffisant. Un hackeur pour monter une attaque de type attaque de l'homme du milieu pour lire le mot de passe hashé et s'en servir pour se connecter au compte.

```
<?php
include_once 'includes/db_connect.php';
include_once 'includes/functions.php';

sec_session_start();

if (login_check($mysqli) == true) {
    $logged = 'in';
} else {
    $logged = 'out';
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Connexion sécurisée, page de connexion</title>
        <link rel="stylesheet" href="styles/main.css" />
        <script type="text/JavaScript" src="js/sha512.js"></script>
        <script type="text/JavaScript" src="js/forms.js"></script>
    </head>
    <body>
        <?php
        if (isset($_GET['error'])) {
            echo '<p class="error">Une erreur s'est produite lors de votre conne
        }
        ?>
        <form action="includes/process_login.php" method="post" name="login_forr
            Email: <input type="text" name="email" />
            Password: <input type="password"
                             name="password"
                             id="password"/>
            <input type="button"
                   value="Connexion"
                   onclick="formhash(this.form, this.form.password);" />
        </form>
        <p>Si vous n'avez pas de compte, veuillez vous <a href="register.php">er
        <p>Si vous avez terminé, veuillez vous <a href="includes/logout.php">déc
        <p>Vous êtes connecté <?php echo $logged ?>.</p>
    </body>
</html>
```

2 Créez la page register_success.php

Créez une nouvelle page PHP que vous appellerez register_success.php. dans la racine de l'application. C'est la page vers laquelle l'utilisateur sera redirigé une fois qu'il

se sera connecté. Il est évident que vous pouvez faire ce que vous voulez sur cette page, vous pouvez même le rediriger vers une tout autre page (ou même ne pas le rediriger du tout). C'est vous qui choisissez. Cette page doit être placée dans la racine de l'application. La page register_success.php que nous avez écrite devrait ressembler à cela :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Connexion sécurisée réalisée avec succès</title>
        <link rel="stylesheet" href="styles/main.css" />
    </head>
    <body>
        <h1>Enregistrement terminé!</h1>
        <p>Vous pouvez maintenant retourner sur la <a href="index.php">page de
    </body>
</html>
```

3 Créez la page d'erreurs

Créez une nouvelle page HTML dans la racine de l'application.
Donnez-lui le nom de error.php. C'est la page vers laquelle l'utilisateur sera redirigé au cas où une erreur se produirait pendant le processus d'enregistrement ou de connexion, ou lorsque vous essayez d'établir une connexion sécurisée. Le code que vous trouverez ci-dessous vous fournit le squelette d'une page d'erreur. Vous aurez sûrement besoin de quelque chose de plus sophistiqué. Notez cependant que vous devez correctement filtrer l'input placé dans la page pour prévenir toute attaque XSS. Voici ce à quoi la page doit ressembler :

```
<?php
$error = filter_input(INPUT_GET, 'err', $filter = FILTER_SANITIZE_STRING);

if (! $error) {
    $error = 'Oups! Une erreur s'est produite.';
}
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Erreur lors de la connexion sécurisée</title>
        <link rel="stylesheet" href="styles/main.css" />
    </head>
    <body>
        <h1>Une erreur s'est produite</h1>
        <p class="error"><?php echo $error; ?></p>
    </body>
</html>
```

Partie 8 sur 8: Protégez vos pages

1 Le script de protection de pages.

Un des problèmes les plus fréquents lors des essais d'authentification est que le développeur a oublié de vérifier si l'utilisateur était déjà connecté ou non. Il est très important d'utiliser le code ci-dessous sur chaque page que vous voulez protéger pour vérifier que l'utilisateur est connecté. Assurez-vous de bien utiliser cette fonction pour savoir si l'utilisateur est connecté.

```
// Mettez les includes pour la connexion à la base de données et les autres fonc
```

```
// Reportez-vous à la section 3.1
sec_session_start();
if(login_check($mysqli) == true) {
        // Ajoutez le contenu de vos pages protégées ici!
} else {
        echo 'Vous n'êtes pas autorisé à accéder à ce contenu, veuillez vous co
}
```

Pour vous montrer une exemple de ce que vous devez faire, nous vous avons inclus un petit morceau de page. Créez un ficher que vous appellerez protected_page.php dans la racine de l'application. Le fichier devrait contenir un code qui ressemble au suivant :

```
<?php
include_once 'includes/db_connect.php';
include_once 'includes/functions.php';

sec_session_start();
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Connexion sécurisée sur page protégée</title>
        <link rel="stylesheet" href="styles/main.css" />
    </head>
    <body>
        <?php if (login_check($mysqli) == true) : ?>
            <p>Bienvenue <?php echo htmlentities($_SESSION['username']); ?>!</p
            <p>
                Ceci est un exemple de page protégée. Pour avoir accès à cette p
être connecté. À un certain point, nous vérifions aussi le rôle de l'utilisateur
 pages puissent connaître le type d'utilisateur autorisé à accéder à la page.
            </p>
            <p>Revenir à la <a href="index.php">page de connexion</a></p>
        <?php else : ?>
            <p>
                <span class="error">Vous n'avez pas les autorisations nécessaire
            </p>
        <?php endif; ?>
    </body>
</html>
```

Notre application va rediriger l'utilisateur vers cette page si l'identification a fonctionné correctement. Votre propre code n'a pas besoin de le faire, bien sûr.

Conseils

- En les modifiant un tout petit peu, ces scripts peuvent aussi marcher avec d'autres systèmes utilisant SQL comme SQLite ou PostgreSQL.
- Si vous voulez utiliser un autre algorithme de hashage que sha512, essayez Whirlpool. Essayez de ne pas utiliser Gost, sha1 (à moins que vous choisissiez un bon salt et que vous utilisiez de nombreuses itérations), et comme mentionné plus tôt, md5. Encouragez vos utilisateurs à créer des mots de passe uniques et variés avec des lettres en minuscule et en majuscule, des chiffres et des symboles. Envisagez aussi de demander à vos utilisateurs de créer un nom de connexion différent de leur nom d'utilisateur pour créer encore plus de sécurité.
- Utilisez HTML et CSS pour mettre en forme le formulaire de connexion et le rendu des pages selon vos préférences.
- N'utilisez pas la fonction md5() dans les scripts de connexion, l'algorithme de hashage md5 est devenu désuet et peu sûr.

Avertissements

- Il n'existe aucun script sûr à 100%. Souvenez-vous de vous tenir au courant des dernières mises à jour de sécurité pour continuer à améliorer la qualité de protection de vos scripts.
- Vos utilisateurs pourraient facilement faire un mauvais emploi du script anti brute force, qui verrouille un compte. Nous vous conseillons vivement d'utiliser un système anti brute force comme le CAPTCHA.
- Assurez-vous que vos utilisateurs n'aient jamais accès à votre code PHP, cela pourrait se produire si le serveur n'est pas configuré correctement. Vos utilisateurs pourraient récolter des informations à propos des noms et des mots de passe de votre base de données si le script PHP devenait visible. L'idéal serait d'enregistrer les scripts que vous incluez dans d'autres scripts ou d'autres pages dans un dossier en dehors de la racine de votre site, en les appelant avec un chemin relatif, par exemple '../../includes/myscript.inc.php'.
- Vous devez utiliser le protocole HTTPS pour votre page de connexion et d'inscription. Le script présenté dans cet article ne vous force pas à le faire, et il est vrai que vous pourriez trouver cela plus simple de ne pas le faire pendant la phase de développement. Mais vous ne devez pas vous servir de ces scripts lorsque vous mettez votre site en ligne, sauf si vous utilisez le HTTPS.
- Vous pourriez trouver de meilleures solutions que ce code si vous utilisez un framework tel que Zend 2, Symfony et CakePHP. Tous ces frameworks incluent des modules de sessions sécurisées, ainsi que d'autres qui vous aident avec le processus de connexion. Vous allez peut-être vous rendre compte que vous écrivez de meilleures applications en vous servant d'un framework.
- Il est vivement recommandé d'utiliser des CAPTCHA sur la page de connexion afin de vous prémunir contre les attaques de brute force et par déni de service. Cette solution n'est pas encore implémentée dans notre code.

Sources et citations

- http://crackstation.net/hashing-security.htm Password Hashing
- https://www.owasp.org/index.php/SQL_Injection Info on SQL Injection

Détails de l'article

Catégories: Ordinateurs et l'électronique | Informatique

Autres langues:

English: Create a Secure Login Script in PHP and MySQL, Português: Criar um Script de Login Seguro em PHP e MySQL, Español: crear un script de inicio de sesión segura en php y MySQL, Italiano: Creare uno Script Sicuro per il Login Usando PHP e MySQL, Deutsch: Ein sicheres Login Skript mit PHP und MySQL erstellen, Русский: создать безопасный логин скрипт в PHP и MySQL