SLAM – TP Login (version 2)

1re Étape

T1.1 | Récupérez sur le site SLAM l'archive `TP_Login_v2.zip` et décompressez-la dans le répertoire des pages Web de votre serveur HTTP.

T1.2 | A l'aide du script login `TP_Login_v2/core/sql/login.sql`, créez une base de donnée nommée `edudock`, la table et un utilisateur.

Les explications des différents scripts sont données en cours.

2ème Étape

Voir sur le site OWASP le top 10 des risques de sécurité applicatifs Web :
https://owasp.org/Top10/

- ***Le principe de l'injection SQL***

Qu'est-ce qu'une injection SQL ?

Une injection SQL utilise une faille de sécurité d'une application Web dans le but de connaître les informations essentielles de la base de données utilisée et/ou d'y introduire des scripts malveillants.

Sites de référence :

http://www.securite-informatique.gouv.fr/
http://www.securiteinfo.com/divers/documentation_securite_informatique.shtml

Des « outils » :

Exemples d'outils d'injection SQL : **The Mole**, **Havij** , **SqlCake**… Ces outils présupposent une faille de sécurité révélée par des sites qui utilisent une fourniture de page Web par numéro et sans contrôle (recherche Google : `inurl:index.php/id=`). Le nombre de ce type d'exploit baisse car de plus en plus de sites utilisent un CMS moderne et donc éliminant cette faille.

- ***Injection SQL par un formulaire de mot de passe***

Beaucoup de sites disposent d'un formulaire pour l'entrée d'une identification par login et mot de passe. La requête SQL dans le script PHP de validation peut être de la forme :

```
$rqut = "SELECT * FROM `users` WHERE login='".$login."' AND passe='".$passe."';";
```

Dès lors, si l'on met dans le champ du login ceci : `a' or 1=1 -- '`

La requête devient :

```
SELECT * FROM `users` WHERE login='a' or 1=1 -- '' AND passe='';
```

On constate que la requête comporte une clause toujours vraie `(1=1)` et ne s'occupe plus du reste (après – signe de commentaire).

- ***Injection JavaScript par un formulaire et une table SQL***

Le principe se base ici sur un champ de type `textearea` pour inscrire un commentaire dans une table. Une inscription de ce genre de commentaire, donnera en lecture autre chose... :

```
<script>alert(\" Coucou!\");</script>
<script>window.location=\"http://www.google.fr\"</script>
```

Ceci sera revu lors de l'étude du langage JavaScript.

- ***Les remèdes***

Une fonction PHP `addslashes()` permet de filtrer les variables. Comme son nom l'indique, elle ajoute des slashes : par exemple la chaîne `'--` devient `\' --`.

Mais cette fonction ne suffit pas pour stopper toutes les injections ; celles via les variables numériques ne sont pas encadrées d'apostrophes ou de guillemets dans les requêtes SQL…

On pourrait utiliser la directive PHP `magic_quotes` (qui en fait n'en était pas une) et qui protège automatiquement les données introduites dans un script PHP. Or, comme le manuel de PHP l'indique, cette fonctionnalité est devenue OBSOLETE depuis PHP 5.3.0 et a été SUPPRIMEE depuis PHP 5.4.0 :

http://php.net/manual/fr/security.magicquotes.php

Alors que faire ?

Une solution de remplacement existe avec la fonction `mysqli_real_escape_string()` (style procédural) ou `$mysqli->real_escape_string()` (style objet) qui ajoute un slash aux caractères suivants : `NULL` , `\x00`, `\n`, `\r`, `\`, `'`, `"` et `\x1a`.

http://php.net/manual/fr/mysqli.real-escape-string.php

Les variantes existent pour PDO.

T2.1 | Remédiez à l'injection SQL dans ce TP.