Sécurité des Applications Web

Sécurité de la base de données (Partie I)

Mehdi BENZINE

benzine.mehdi@gmail.com

Département d'Informatique
Faculté des Sciences
Université FERHAT ABBAS Sétif I

2019/2020

Introduction

Les données constituent la principale richesse d'une application web.

L'accès aux données doit être rigoureusement contrôlé.

Tous les moyens d'accéder aux données doivent être pris en compte et sécurisés

- accès à travers l'application web
- accès avec un client base de données quelconque
- accès via le système de fichier aux fichiers de données
- accès physique au support de stockage
- accès aux fichiers de log et aux sauvegardes

Le SGBD peut stocker d'autres données que celles de l'application web. Le(s) compte(s) utilisé(s) par une application web doit(vent) être limité(s), en terme de droits d'accès, aux données nécessaires à l'application.

Injection SQL

L'injection de SQL consiste à utiliser des champs destinés à saisir des information textuelles pour entrer du code SQL.

Les attaques par injection SQL exploitent la construction de requêtes SQL par concaténation de chaînes de caractères.

Les requêtes construites par concaténation avec des chaînes de caractères contenant des injections SQL fournissent un résultat différent de celui prévu par le développeur de l'application.

Injection SQL(suite)

Une application web qui affiche à chaque étudiant ses notes d'examens, peut fonctionner la manière suivante:

1. identification de l'étudiant et création d'une session pour cet étudiant
2. renseignement de la matière dont l'étudiant souhaite consulter la note
3. recherche de la note en construisant une requête incluant l'identifiant de l'étudiant (variable de session) et la matière recherchée (variable de formulaire)

Un étudiant ne doit pas avoir accès aux notes des autres étudiants.

Exemple:

```
select matiere, note
from notes
where identifiant = '" . $_SESSION['identifiant'] ."'
      and matiere like '" . $_POST['matiere'] . "%'
```

Injection SQL(suite)

La saisie par un utilisateur malicieux (Farid) de la chaîne suivante comme "matière" `GL' or 1=1 #` la requête envoyée au SGBD et exécutée sera:

```
select matiere, note
from notes
where identifiant = 'Farid'
      and matiere like 'GL' or 1=1 #'
```

Le résultat renvoyé par le SGBD sera l'ensemble des notes de tous les étudiants!!!

Injection SQL(suite)

MySQL offre la possibilité de stocker le résultat d'une requête SQL dans un fichier en définissant dans la requête le nom et l'emplacement du fichier.

Cette fonctionnalité permet de réaliser des attaques XSS exportant des données vers un emplacement accessible à l'attaquant.

Exemple:

```
select matiere, note
from notes
where identifiant = 'Farid'
      and matiere like 'GL'
union
select num, note  into outfile /tmp/export.txt
from notes #'
```

Injection SQL(suite)

Cette fonctionnalité permet également de générer des scripts PHP et les placer dans un répertoire publié pour les exécuter.

Exemple:

```
select matiere, note
from notes
where identifiant = 'Farid'
      and matiere like 'GL' union
select <?php echo "Attention"; ?>
       into outfile /var/www/export.php #'
```

Injection SQL(suite)

Pour se prémunir des injection SQL, il faut neutraliser les caractères d'échappement pouvant être inclus dans la chaîne de caractères saisie par l'utilisateur.

La fonction PHP `mysqli_real_escape_string($connexion, $chaine)` prend en paramètre une chaîne de caractères et en neutralise les caractères d'échappement.

Exmeple:

```php
<?php
//$_POST["matiere"] contient "GL' or 1=1 #"
$VALIDE["matiere"] = mysqli_real_escape_string($connect, $_POST["matiere"]);
//$VALIDE["matiere"] contient "GL\' or 1=1 #"
?>
```

L'apostrophe sera considéré par MySQL comme un littéral et non plus comme un caractère de délimitation de chaîne de caractères.

Injection SQL(suite)

Une autre solution consiste à utiliser des requêtes **paramétrées**.
La requêtes est envoyée au serveur avec des paramètres à la place des valeurs constantes.
Les valeurs sont ensuite envoyées au serveur au moment de l'exécution de la requête.
La requête est envoyée au serveur grâce à la fonction `mysqli_prepare($connexion , $requete)` prend en paramètres un `identifiant de connexion` et une `requête SQL` et retourne un `statement`.

Exmeple:

```php
<?php
    $requete = "select * from employe where num >= ?";
    $statement = mysqli_prepare($connexion, $requete);
    ...
```

Injection SQL(suite)

Il faut ensuite faire correspondre des valeurs réelles aux paramètres de la requête (binding).
La fonction qui réalise cela se nomme `mysqli_stmt_bind_param($statement, $types, $valeur1, $valeur2...)` et prend en paramètres un `statement`, un `une chaîne de caractères contenant les types des paramètres` et les `valeurs des paramètres` dans l'ordre où ils apparaissent dans la requête préparée.

La fonction retourne un booléen indiquant le succès ou l'échec de l'opération.

La chaîne `$types` contient les types des paramètres dans l'ordre où ils apparaissent dans la requête préparée.

Injection SQL(suite)

Exmeple:

```php
<?php
    ...
    $nb = 1003;
    mysqli_stmt_bind_param($statement, "i", $nb);
    ...
```

La chaîne `$types` contient les types des paramètres dans l'ordre où ils apparaissent dans la requête préparée.

Les types possibles sont:

`i:` nombre entier

`d:` nombre décimal

`s:` chaîne de caractères

`b:` BLOB (objet binaire)

Injection SQL(suite)

L'exécution de la requête se fait grâce à l'appel de la fonction `mysqli_stmt_execute($statement)` qui prend en paramètre un `statement` et retourne un booléen indiquant le succès ou l'échec de l'opération.
Pour être accessible, le résultat de l'exécution doit être associé à des variables PHP (binding) avec la fonction `mysqli_stmt_bind_result($statement, $var1, $var2...);` retourne un booléen indiquant le succès ou l'échec de l'opération

Exmeple:

```php
<?php
    ...
    mysqli_stmt_execute($statement);
    mysqli_stmt_bind_result($statement, $num, $nom, $prenom);
    ...
```

Injection SQL(suite)

Le résultat est enfin parcouru avec la fonction `mysqli_stmt_fetch($statement` qui va à chaque appel recopier les valeurs des attributs d'un des tuples résultat, dans les variables associées et retourne un booléen indiquant le succès ou l'échec de l'opération.

Exmeple:

```php
<?php
...
while(mysqli_stmt_fetch($statement)){
    echo "<tr>";
    echo "<td>" . $num . "</td>\n";
    echo "<td>" . $nom . "</td>\n";
    echo "<td>" . $prenom . "</td>\n";
    echo "</tr>";
}
...
```

Injection SQL(fin)

Une requête préparée peut être exécutée plusieurs fois en changeant à, chaque fois les valeurs des paramètres associés. Enfin, le statement doit être fermé en appelant la fonction `mysqli_stmt_close($statement)` qui prend en paramètre le `statement` et retourne un booléen indiquant le succès ou l'échec de l'opération.

Exmeple:

```php
<?php
    ...
    mysqli_stmt_close($statement);
    ...
```

Sécurité des identifiants de connexion

Les identifiants de connexion (identifiant, mots de passe, nom du serveur) utilisés doivent être protégés pour empêcher qu'un attaquant puisse y avoir accès et puisse accéder de manière illicite à la base de données.

Les identifiants peuvent être stockés à différents endroits:

- dans les scripts PHP
- dans un fichier de configuration PHP
- comme variables sur le serveur web
- comme variables dans le fichier de configuration de PHP

Sécurité des identifiants de connexion(suite)

dans les scripts PHP

Le stockage des identifiants de connexion dans les sripts PHP (sous forme de variables ou de constantes dans l'appel à la fonction de connexion) est dangereux car si un utilisateur peut accéder au code source des scripts, il peut récupérer les identifiants de connexion.

Exemple:

```php
<?php
    ...
    $user = "mehdi";
    $passwd = "1234";
    $host  = "192.168.56.101";
    $bd = "scolarite";
    $cnx = mysqli_connect($host, $user, $passwd, $bd);
    $requete = "...";
    ...
?>
```

Sécurité des identifiants de connexion(suite)

dans les scripts PHP

Le stockage des identifiants de connexion dans des fichiers de configuration PHP présente les même risques.

Exemple:

```
//Fichier config.inc
<?php
    ...
    $user = "mehdi";
    $passwd = "1234";
    $host  = "192.168.56.101";
    $bd = "scolarite";
?>
//Autre fichier *.php
<?php
    include "config.inc";
    $cnx = mysqli_connect($host, $user, $passwd, $bd);
    $requete = "...";
    ...
?>
```

Sécurité des identifiants de connexion(suite)

comme variables sur le serveur web

Le stockage des identifiants de connexion comme variables du serveur web offre plus de sécurité. Les variables sont stockées comme variables d'environnement du serveur et sont accessibles aux scripts PHP via le tableau super global $_SERVER.

Exemple:

```
Dans le fichier configuration Apache (ex: httpd.conf)
...
SetEnv mysql_host "192.168.56.101"
SetEnv mysql_user "mehdi"
SetEnv mysql_passwd "1234"
SetEnv mysql_bd "scolarite"
...
```

```php
<?php
    ...
    $cnx = mysqli_connect($_SERVER["mysql_host"], $_SERVER["mysql_user"],
    $_SERVER["mysql_passwd"], $_SERVER["mysql_bd"]);
    $requete = "...";
    ...
?>
```

Sécurité des identifiants de connexion(suite)

comme variables dans le fichier de configuration de PHP

Le stockage des identifiants de connexion par défaut dans le fichier `php.ini`.

Exemple:

```
Dans le fichier configuration php.ini
...
mysql.default_host = "192.168.56.101";
mysql.default_user = "mehdi";
mysql.default_password = "1234";
...
```

```php
<?php
    ...
    $cnx = mysqli_connect();
    mysqli_select_db($cnx, "scolarite");
    ...
?>
```

Ces deux dernières manières de stocker les identifiants de connexion étant les plus fiables.

Connexion au SGBD

Les utilisateurs disposant d'un identifiant (login) et d'un mot de passe doivent être le seuls habilités à se connecter au SGBD. Il faut interdire les accès anonymes.

L'application web doit disposer de son propre compte MySQL et ses accès doivent être limités:

- si le serveur web et le serveur MySQL sont sur la même machine, seules les connexions locales doivent être autorisées
- s'ils sont sur deux machines du même réseau local seules les connexion depuis le réseau local doivent être possibles
- dans les autres cas, seule l'accès depuis l'adresse IP fixe du serveur web doivent être autorisés

Connexion au SGBD(suite)

Sur MySQL, à chaque utilisateur est associé un hôte à partir du quel il est autorisé à se connecter au SGBD.

- `user@host:` l'utilisateur `user` ne peu se connecter qu'à partir de l'hôte `host`.
- `user@127.0.0.1:` l'utilisateur `user` peu se connecter à partir de la machine sur laquelle est installé MySQL.
- `user@192.168.56.101:` l'utilisateur `user` peu se connecter à partir de la machine ayant pour adresse IP `192.168.56.101`.
- `user@192.168.56.%:` l'utilisateur `user` peu se connecter à partir d'une machine du sous-réseau `192.168.56.0`.
- `user@%:` l'utilisateur `user` peu se connecter à partir de n'importe quel emplacement.

La création d'un utilisateur se fait grâce à la commande SQL `CREATE USER`.

Exemple:

```
CREATE USER 'mehdi'@'127.0.0.1' IDENTIFIED BY '1234';
```

La suppression d'un utilisateur se fait grâce à la commande SQL `DROP USER`.

Exemple:

```
DROP USER 'mehdi'@'127.0.0.1';
```

L'attribution de droits à un utilisateur se fait grâce à la commande SQL GRANT.

```
GRANT droit1, droit2, ...
ON objet
TO utilisateur
[WITH GRANT OPTION];
```

Les droits d'accès pouvant être attribués un utilisateur sont:

- `SELECT`
- `INSERT`
- `UPDATE(liste attributs)`
- `DELETE`
- `CREATE TABLE`
- `CREATE VIEW`
- `ALTER TABLE`
- `DROP`
- `...`

Les objets auxquels s'appliquent les droits sont les tables, les vues, les procédures stockées, les bases de données ...

- `*.*:` tous les objets de toutes les bases de données
- `scolarite.*:` tous les objets de la base de donnée scolarite
- `scolarite.notes:` l'objet notes(table) de la base de donnée scolarite

L'option `WITH GRANT OPTION` permet à l'utilisateur qui en dispose en recevant des droits, de transmettre ces droits à d'autres utilisateurs.

La révocation de droits à un utilisateur se fait grâce à la commande SQL REVOKE.

```
REVOKE droit1, droit2, ...
ON objet
FROM utilisateur;
```

Les fichiers contenant les données (et méta-données) doivent être protégés contre les accès directes via le système de fichiers.

Les droits d'accès définis sur ces fichiers doivent être minimaux et ne permettre que leur utilisation par le SGBD.

Les fichiers de sauvegarde de la base de données peuvent être générés en utilisant des outils du SGBD(`mysqldump`, `SELECT ... INTO OUTFILE`) ou en faisant directement une copie des fichiers de données.

Ces fichiers de sauvegarde doivent être protégés de la même manière que les fichiers utilisés par le SGBD.

Si ces fichiers sont stockés sur des supports amovibles (DVD,disque dur externe ...), leur entreposage et leur utilisation doivent être fait de manière sûre de manière à éviter leur vol, destruction ...

`mysqldump` permet de créer un `dump` d'une ou plusieurs bases de données MySQL.

L'utilisation de la réplication des fichiers de données du serveur maître vers des serveurs esclaves imposent d'appliquer la même politique de sécurité sur les fichiers stockés sur les serveurs esclaves.

Les fichiers de log SQL contiennent des informations sur les requêtes soumises au SGBD. Un pirate ayant accès à ces fichiers peut en extraire de l'information sensible et l'exploiter.

Ces fichiers doivent également être protégés et leur accès doit être restreint.

Sur Ubuntu le fichier de log des connexions et des requêtes traitées par MySQL se trouve dans le répertoire `/var/lib/mysql/[nom machine].log`

Par défaut, le log des connexions et des requêtes traitées n'est pas activé.

Conclusion

Les données étant la principale richesse d'une application web, leur confidentialité et leur intégrité est un problème majeur à prendre en compte dans toute réflexion sur la sécurité d'une application web.

La sécurité des données doit être envisagée par rapport aux utilisateurs, mais aussi par rapport aux développeurs d'applications.