Cours de Développement Web
Un peu de sécurité

Frédéric Flouvat

Université de la Nouvelle-Calédonie

frederic.flouvat@univ-nc.nc

Et la sécurité ?

- Comment pirater un site web ?
  - exploiter ses failles de sécurité
    - injection SQL
    - faille XSS
    - …
  - exploiter les failles de ses utilisateurs

- Qui ?
  - un "pirate" informatique
  - un bot, i.e. un programme informatique

- Comment se protéger ?
  - sécuriser l'accès à la base de données et masquer son fonctionnement
  - surveiller et sécuriser les données saisies par les utilisateurs ainsi que leur comportement
  - utiliser des protocoles de communication sécurisés lorsque cela est nécessaire

Les injections SQL

- Principe: se servir des champs de saisie (ou plus généralement des variables transmises) pour "injecter" du code SQL "malicieux"

- Permet de changer le résultat d'une requête, d'accéder au contenu de la base de données, voire de supprimer des tables

login inexistant dans la base de données!

Saisir:
" or 1 = 1#

$query= 'SELECT login FROM Users WHERE login="'.$_POST['login'].'" and password="'.$_POST['password'].""';

Exécute la requête suivante:

SELECT login FROM Users WHERE login="truc" and password="" or 1 = 1#"

symbole MySQL permettant de mettre la suite en commentaire

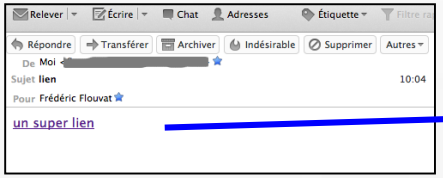
- retourne toujours un résultat

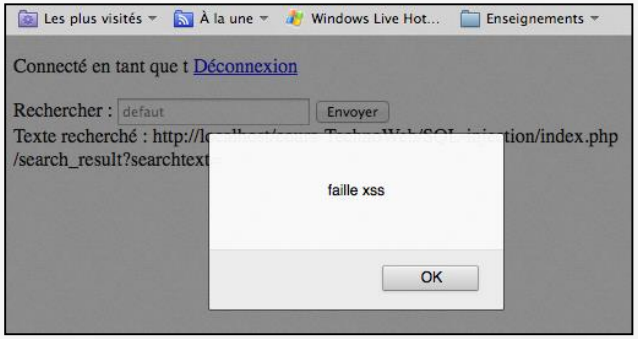
La faille XSS (*cross-site scripting*)

- Principe: injecter un script (exécuté côté client) dans des pages vues par d'autres utilisateurs
  - du code JavaScript, HTML, …

- Permet de rediriger vers un faux site (hameçonnage, *phishing*), de voler des informations enregistrées dans les cookies, de faire exécuter du code malicieux
  - failles non permanentes (par réflexion)

un mail avec un lien vers un site de confiance





http://sitesur.com/index.php/search_result?searchtext=<script language="javascript" src="http:/monsite.com/xss.js" </script>

  - failles permanentes
    - le lien ou le code malicieux est enregistré par le site dans un fichier ou dans la base de données, et est affiché/exécuté à chaque ouverture du site (p.ex. dans un post)

Les autres failles

- D'autres failles peuvent être exploitées *
  - la faille include
    - exploite une mauvaise utilisation de la fonction **include**

```
<?php include($_GET['fichier']); ?>
```

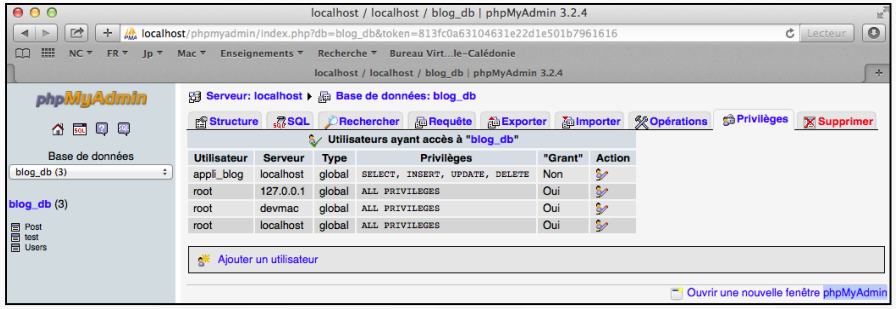
      - ➢ utilisée pour exécuter du code PHP situé sur une autre page, parcourir les répertoires du site, …
  - la faille upload
    - uploader du code PHP malicieux à la place d'une image, d'un fichier, …
    - vérifier l'extension du fichier ne suffit pas !
      p.ex. upload de "**backdoor.php\0.jpg**" fonctionnera
  - la faille CRLF (Carriage Return Line Feed)
    - insérer un retour à la ligne (%0A) dans un champ de saisie de texte
    - ➢ permet de récupérer le mot de passe d'un utilisateur en se mettant en copie (dans les champs de type "mot de passe oublié")
  - …

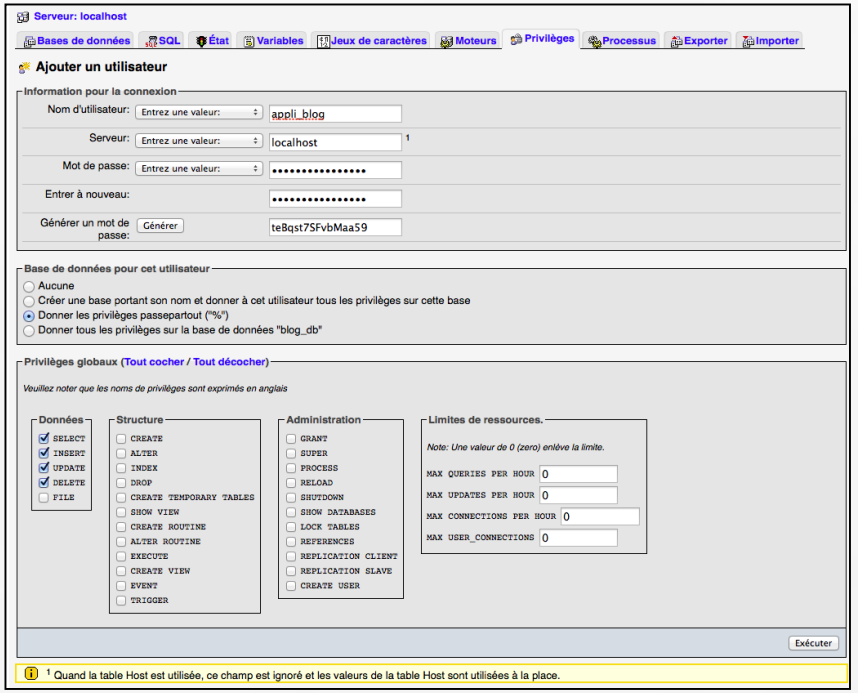
victime@service.com%0Apirate@service.com

* https://openclassrooms.com/courses/protegez-vous-efficacement-contre-les-failles-web

Sécuriser la BD: son accès et son fonctionnement

- Sécuriser l'accès à la base de données
  - créer des utilisateurs avec des droits limités dans la base de données
  - ➢ ne jamais connecter l'application web en "root" !





- Masquer son fonctionnement
  - ne pas afficher directement les erreurs SQL car cela peut donner des renseignements sur la base de données

Sécuriser la BD: son interrogation

- En PHP, utiliser les requêtes préparées et les paramètres liés de Mysqli (ou des Php Data Objects, i.e. PDO)
  - La fonction **mysqli_query** interdit l'exécution de plusieurs requêtes en même temps, mais n'empêche pas de modifier la requête
    - p.ex. évite juste l'injection de ***DROP TABLE Users;*** si l'utilisateur saisi ***" or 1; DROP TABLE Users;*** # mais accepte ***" or 1;***

➢ Intérêt des requêtes préparées

1. la requête est créée et envoyée au SGBD sans ses paramètres utilisateurs
   - p.ex. SELECT login FROM Users WHERE login=? and password=?
2. le SGBD compile et prépare le plan d'exécution de la requête
3. à l'exécution, l'application lie les valeurs des paramètres à la requête et le SGBD l'exécute

➢ impossible d'avoir des requêtes dans les paramètres

Sécuriser la BD: son interrogation

- A la place de **mysqli_query** et de **mysqli_fetch_assoc** faire:

1. créer une requête préparée avec **mysqli_prepare**(*$link*, *$query*)
   - les paramètres issus de données utilisateurs sont remplacées par des **?** dans la requête
2. lier les paramètres de la requête avec **mysqli_stmt_bind_param(***$stmt,$types,$var1,$var2,…* **)**
2. exécuter la requête avec **mysqli_stmt_execute(***$stmt***)**
4. lier les résultats (colonnes) à des variables avec **mysqli_stmt_bind_result(**$col1, $col2,…**)**
4. parcourir les résultats avec **mysqli_stmt_fetch(***$stmt***)**
6. libérer le résultat et/ou la requête avec **mysqli_stmt_free_result(** $stmt **)** **mysqli_stmt_close(** $stmt **)**

```
$link = mysqli_connect('localhost', 'user', 'pwd', 'blog_db');

$stmt = mysqli_prepare($link,'SELECT id, title FROM
Post WHERE login =?');


mysqli_stmt_bind_param($stmt, 's', $_POST['login']);



mysqli_stmt_execute($stmt);


mysqli_stmt_bind_result($stmt, $id, $title);


$userposts = array();
while ( mysqli_stmt_fetch($stmt) ) {
        $userposts[] = array( 'id' => $id, 'title' => $title ) ;
}
mysqli_stmt_free_result( $stmt );
mysqli_stmt_close( $stmt);
```

Vérifier toutes les données provenant de l'utilisateur

- Utiliser la fonction PHP **htmlspecialchars(**$string**)** pour filtrer les symboles du type <, & ou "
  - la fonction les remplace par leur "code" HTML (p.ex. **& "** ..)

- Vérifier le type des données passées en paramètres par l'utilisateur
  - p.ex. utilisation de la fonction PHP **intval(** $input **)** pour s'assurer qu'une valeur est un entier
  - ➢ http://www.php.net/manual/en/book.ctype.php

- Interdire les retours à la ligne dans certains champs de saisie (p.ex. pour les mails)
  - utiliser p.ex. la fonction PHP **str_replace**

```
$secure_str = str_replace(array('\n','\r',PHP_EOL),' ',$user_str);
```

- Renommer tous les fichiers uploader (du nom à l'extension)*

* https://openclassrooms.com/courses/protegez-vous-efficacement-contre-les-failles-web

Vérifier toutes les données provenant de l'utilisateur

- Mettre en place un système de jetons afin de garantir que les actions en cours sont bien faites dans le cadre de la session en cours *
  - générer aléatoirement un identifiant de session,
  - enregistrer cet identifiant dans la session et dans un champ **hidden** dans les formulaires
  - vérifier la correspondance à chaque modification

- Vérifier la durée de la "session"
  - stocker l'heure dans la session et dans champ **hidden** dans les formulaires
  - vérifier que la delta entre les deux ne dépasse pas une certaine limite

- Vérifier d'où viennent les données postées via la variable **$_SERVER[‘HTTP_REFERER’]**

- **…**

* https://openclassrooms.com/courses/protegez-vous-efficacement-contre-les-failles-web

Renforcer l'authentification des utilisateurs

- Sécuriser les mots de passe et ne pas les enregistrer en clair dans la base de données
  - au moins 10 caractères (lettres, chiffres, casse, symboles)

11 min. pour craquer e87a5de7
3 heures pour e8!a5d&7
3 jours pour A8!a5d&7
58 ans pour A8!a5d&712

(avec un PC "lambda")

  - le faire à la place de l'utilisateur si nécessaire …

```
$hash= password_hash($pwdGet, PASSWORD_DEFAULT);
INSERT INTO users VALUES ($login, $hash)
```

enregistrement

```
…
if( password_verify( $pwdGet, $hashInDB) ){
…
}
```

vérification

- Utiliser des captcha pour limiter les attaques des bots
  - mais les programmes de reconnaissances sont de plus en plus efficaces, et possibilité d'acheter des captcha (p.ex. 1000 pour 1$)

- Utiliser un protocole sécurisé (p.ex. HTTPS) lors de la phase d'authentification