Cours 06 - Sécurité et nettoyage des saisies utilisateurs

https://github.com/heig-vd-progserv1-course

Support de cours · Présentation (web) · Présentation (PDF)

L. Delafontaine, avec l'aide de GitHub Copilot.

Retrouvez plus de détails dans le support de cours

Cette présentation est un résumé du support de cours. Pour plus de détails, consultez le support de cours.

Objectifs (1/2)

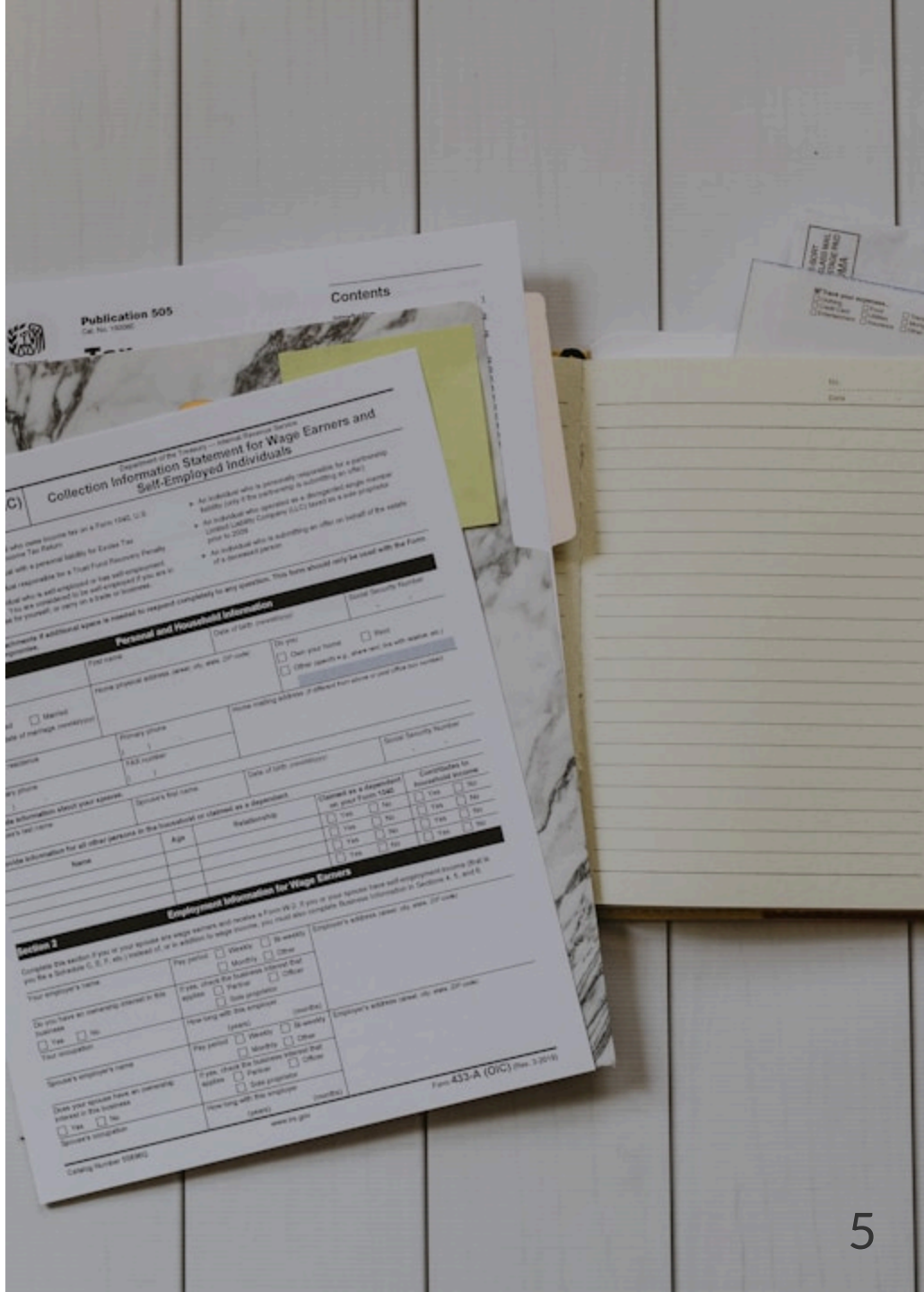
- Lister les différences entre la validation et le nettoyage des saisies utilisateurs
- Lister les implications de sécurité des saisies utilisateurs
- Décrire les attaques par injection SQL et XSS
- Se prémunir contre les injections SQL et les attaques XSS

Objectifs (2/2)

- Utiliser des requêtes préparées avec PDO
- Échapper les données avant de les afficher dans une page web

Validation et nettoyage des saisies utilisateurs

- La validation consiste à vérifier que les données saisies par l'utilisateur respectent un certain format ou des règles spécifiques.
- Le nettoyage consiste à supprimer ou échapper (= modifier) les caractères spéciaux ou malveillants des données saisies par l'utilisateur.



Nettoyage des saisies utilisateurs (1/3)

- *"Échapper"* signifie remplacer les caractères spéciaux par des séquences de caractères interprétées comme des caractères littéraux plutôt que comme des caractères spéciaux.
- La fonction `htmlspecialchars` est utilisée pour cela.

Nettoyage des saisies utilisateurs (2/3)

Par exemple, la table suivante présente des caractères échappés.

Caractère	Séquence d'échappement
`<`	`<`
`>`	`>`
`&`	`&`
`"`	`&quot;`
`'`	`&apos;`

Nettoyage des saisies utilisateurs (3/3)

```php
// On définit une chaîne de caractères HTML avec des caractères spéciaux
$string = "<a href='index.php'>Accueil</a>";

// Affiche un lien cliquable - à éviter à tout prix
echo $string;

// On échappe les caractères spéciaux
// La chaîne échappée sera : &lt;a href='index.php'&gt;Accueil&lt;/a&gt;
$escapedString = htmlspecialchars($string);

// On affiche la chaîne échappée, qui sera littéralement
// <a href='index.php'>Accueil</a>
// et non un lien cliquable
echo $escapedString;
```

Implications de sécurité

- Les saisies utilisateurs peuvent contenir des données malveillantes.
- Ces données peuvent être utilisées pour réaliser des attaques avec des conséquences graves.
- Les attaques les plus courantes sont :
  - Les injections SQL
  - Les attaques XSS

Injections SQL (1/3)

- Permettent d'injecter du code SQL dans une requête déjà existante.
- Permettent d'accéder à des données sensibles ou de modifier la base de données.
- Souvent réalisées en utilisant des formulaires web lorsque les saisies utilisateurs ne sont pas correctement échappées ou validées.

Injections SQL (2/3)

```php
<?php
// Constante pour le fichier de base de données SQLite
const DATABASE_FILE = "./users.db";

// Connexion à la base de données
$pdo = new PDO("sqlite:" . DATABASE_FILE);

// Création d'une table `users`
$sql = "CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL
)";
```

```php
// On exécute la requête SQL pour créer la table
$pdo->exec($sql);

// Gère la soumission du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // On récupère les données du formulaire
    $email = $_POST["email"];
    $password = $_POST["password"];

    // On prépare la requête SQL pour ajouter un utilisateur
    $sql = "INSERT INTO users (email, password) VALUES ('$email', '$password')";

    // On exécute la requête SQL pour ajouter l'utilisateur
    $pdo->exec($sql);
}
?>
```

```
<!-- Gère l'affichage du formulaire -->
<!DOCTYPE html>
<html>

<head>
    <title>Création d'un compte</title>
</head>

<body>
    <h1>Création d'un compte</h1>
    <a href="view-accounts.php"><button>Voir les comptes</button></a>
```

```
<form action="create-account.php" method="POST">
    <label for="email">E-mail :</label><br>
    <input
        type="text"
        id="email"
        name="email" />

    <br>

    <label for="password">Mot de passe :</label><br>
    <input
        type="password"
        id="password"
        name="password" />
```

```
        <br>

        <button type="submit">Envoyer</button>
    </form>

    <?php if ($_SERVER["REQUEST_METHOD"] == "POST") { ?>
        <p>Le formulaire a été soumis avec succès.</p>
    <?php } ?>
</body>

</html>
```

Injections SQL (3/3)

- Si un utilisateur saisit `'); DROP TABLE users; --` dans le champ password, la requête SQL devient :

```sql
INSERT INTO users (email, password) VALUES ('me@example.com', ''); DROP TABLE users; --'
```

- La requête SQL est alors exécutée et la table `users` est supprimée.
- Les traits d'union (`--`) sont un commentaire SQL ; le reste de la requête est ignoré.
- En injectant notre propre code SQL, nous pouvons manipuler la base de données. Les conséquences sont infinies. 🙃

Attaques XSS (1/3)

- Permettent d'injecter et d'exécuter du code JavaScript dans une page web.
- Souvent réalisées lors de l'affichage de données non échappées dans une page web.
- Comme vous n'avez pas encore étudié JavaScript, nous n'allons pas aller dans les détails mais illustrer le principe.

Attaques XSS (2/3)

```php
<?php
// Constante pour le fichier de base de données SQLite
const DATABASE_FILE = "./users.db";

// Connexion à la base de données
$pdo = new PDO("sqlite:" . DATABASE_FILE);

// On prépare la requête SQL pour récupérer tous les utilisateurs
$sql = "SELECT * FROM users";

// On exécute la requête SQL pour récupérer les utilisateurs
$users = $pdo->query($sql);
```

```
// On transforme le résultat en tableau
$users = $users->fetchAll();
?>

<!-- Gère l'affichage du formulaire -->
<!DOCTYPE html>
<html>

<head>
    <title>Comptes utilisateurs</title>
</head>

<body>
    <h1>Comptes utilisateurs</h1>

    <a href="create-account.php"><button>Créer un compte</button></a>
```

```
    <ul>
        <?php foreach ($users as $user) : ?>
            <li><?= $user["email"] ?></li>
        <?php endforeach; ?>
    </ul>
</body>

</html>
```

Attaques XSS (3/3)

- Si un utilisateur saisit `<script>alert("Vous avez été piraté !");</script>` dans le champ email, la page affichera une alerte JavaScript avec le message `Vous avez été piraté !`.
- Bien que l'exemple paraisse inoffensif, du code JavaScript a été injecté et exécuté dans la page web avec succès.
- En injectant notre propre code JavaScript, nous pouvons manipuler la page web. Les conséquences sont infinies. 🙃

Se prémunir conte les injections SQL et les attaques XSS

- Les bonnes pratiques à suivre :
  - Ne jamais faire confiance aux saisies utilisateurs.
  - Toujours valider, nettoyer et échapper les saisies utilisateurs avant de les utiliser.

Requêtes préparées avec PDO (1/4)

- Fonctionnalité de PDO pour éviter les injections SQL.
- Permet de préparer une requête SQL avec des paramètres.
- Les paramètres sont remplacés et échappés automatiquement par des valeurs lors de l'exécution de la requête.

Requêtes préparées avec PDO (2/4)

```php
// On prépare la requête SQL pour ajouter un utilisateur
$sql = "INSERT INTO users (email, password) VALUES (:email, :password)";

// On prépare la requête SQL
$stmt = $pdo->prepare($sql);

// On lie les paramètres aux valeurs réelles
$stmt->bindValue(':email', $email);
$stmt->bindValue(':password', $password);

// On exécute la requête
$stmt->execute();
```

Requêtes préparées avec PDO (3/4)

```
      // On prépare la requête SQL pour ajouter un utilisateur
-     $sql = "INSERT INTO users (email, password) VALUES ('$email', '$password')";
+     $sql = "INSERT INTO users (email, password) VALUES (:email, :password)";
+
+     // On prépare la requête
+     $stmt = $pdo->prepare($sql);
+
+     // On lie les paramètres
+     $stmt->bindValue(':email', $email);
+     $stmt->bindValue(':password', $password);

      // On exécute la requête SQL pour ajouter l'utilisateur
-     $pdo->exec($sql);
+     $stmt->execute();
```

Requêtes préparées avec PDO (4/4)

- Les paramètres sont automatiquement échappés et remplacés par PDO.
- Les injections SQL sont donc impossibles.
- Recommandé d'utiliser des requêtes préparées pour toutes les requêtes SQL, même celles qui ne contiennent pas de saisies utilisateurs.

Échapper les données (1/4)

- Permet de transformer les caractères spéciaux en séquences d'échappement.
- Permet d'éviter les attaques XSS.
- Utiliser la fonction `htmlspecialchars` offerte par PHP pour échapper les caractères spéciaux.

Échapper les données (2/4)

```
<ul>
    <?php foreach ($users as $user) : ?>
        <li><?= htmlspecialchars($user["email"]) ?></li>
    <?php endforeach; ?>
</ul>
```

Échapper les données (3/4)

```
      <ul>
          <?php foreach ($users as $user) : ?>
-             <li><?= $user["email"] ?></li>
+             <li><?= htmlspecialchars($user["email"]) ?></li>
          <?php endforeach; ?>
      </ul>
```

Échapper les données (4/4)

- Les caractères spéciaux sont transformés en séquences d'échappement.
- Les attaques XSS sont donc impossibles.
- Recommandé d'utiliser `htmlspecialchars` pour toutes les données affichées issues de saisies utilisateurs.

Conclusion (1/2)

- **Ne jamais faire confiance aux saisies utilisateurs !**
- La validation et le nettoyage des saisies utilisateurs sont essentiels pour éviter les injections SQL et les attaques XSS.
- Utiliser des requêtes préparées avec PDO pour éviter les injections SQL.

Conclusion (2/2)

- Utiliser `htmlspecialchars` pour échapper les données avant de les afficher dans une page web.
- Dans la vie réelle, il est souvent nécessaire de tâtonner pour trouver les vulnérabilités.
- **Il peut y avoir des conséquences graves !** Ne pas reproduire ces attaques sur des applications web sans autorisation explicite !

Questions

Est-ce que vous avez des questions ?

À vous de jouer !

- (Re)lire le support de cours.
- Réaliser le mini-projet.
- Faire les exercices.
- Poser des questions si nécessaire.

Pour le mini-projet ou les exercices, n'hésitez pas à vous entraidez si vous avez des difficultés !

Sources

- Illustration principale par Richard Jacobs sur Unsplash
- Illustration par Aline de Nadai sur Unsplash
- Illustration par Kelly Sikkema sur Unsplash
- Illustration par Towfiqu barbhuiya sur Unsplash
- Illustration par Sunder Muthukumaran sur Unsplash
- Illustration par Jen Theodore sur Unsplash
- Illustration par John Salvino sur Unsplash
- Illustration par Nikita Kachanovsky sur Unsplash