

La sécurité

Table des matières

I. Contexte	3
II. Attaques par injections	3
A. Attaques par injections	3
B. Exercice : Quiz	9
III. Les attaques XSS, CSRF et le Hijacking	10
A. Les attaques XSS, CSRF et le Hijacking	10
B. Exercice : Quiz	14
IV. Essentiel	15
V. Auto-évaluation	15
A. Exercice	15
B. Test	16
Solutions des exercices	17

I. Contexte

Durée : 1 h

Prérequis : Bases de PHP - SQL

Environnement de travail : IDE - WampServer - PHP 8.1.xx -phpmyadmin5.2.1

Contexte Les failles de sécurité sur PHP

La sécurité est une préoccupation majeure pour tout développeur de logiciels et de sites web. PHP est un des langages de programmation les plus populaires, et malheureusement, il n'est pas exempt de failles. Ces faiblesses peuvent permettre à des attaquants de prendre le contrôle d'un serveur web et d'en soutirer des données ou causer d'autres types de dommages. C'est pourquoi il est nécessaire de se prémunir contre celles-ci à tous les niveaux d'un projet web. En commençant par être conscient des différentes failles et par prendre des mesures de sécurité. Ces failles peuvent provenir de différentes sources, telles que des erreurs de codage, des problèmes de configuration ou de diverses vulnérabilités.

Dans ce cours, nous allons voir comment se protéger de ces risques. En général, ils sont catégorisés en plusieurs types : attaques par injection de code, XSS, et CSRF, avec leurs propres caractéristiques et exigences pour les corriger.

II. Attaques par injections

A. Attaques par injections

Attaque par inclusion de fichier

La première faille que nous allons aborder est l'attaque par inclusion de fichiers. En PHP, l'inclusion de fichiers est une fonctionnalité qui permet d'inclure le contenu d'un fichier dans un autre fichier. Cela peut être très utile pour réutiliser du code commun à plusieurs pages, mais cela peut aussi être exploité par des pirates informatiques pour exécuter du code malveillant sur notre site web.

La méthode la plus courante utilisée par les attaquants pour exploiter cette faille est appelée « *l'inclusion de fichiers locaux* ». Ils utilisent une variable non filtrée pour inclure un fichier local. Par exemple, si un site web utilise une URL de la forme `http://example.com/?page=about.php`, un pirate informatique peut remplacer le nom du fichier **about.php** par un chemin de fichier absolu sur le serveur (**`http://example.com/?page=/etc/passwd`**), ce qui inclura le contenu du fichier **/etc/passwd**. Pour éviter cela, il est important de valider et filtrer toutes les entrées utilisateur, comme les paramètres d'URL, les formulaires et les cookies, pour s'assurer qu'elles sont conformes à ce qui est attendu.

Exemple

Voici un exemple de code pour filtrer les entrées utilisateur et éviter les injections de code en utilisant la fonction `htmlspecialchars()` de PHP :

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="UTF-8">
5     <title>Formulaire</title>
6 </head>
7 <body>
8     <h1>Formulaire</h1>
9     <form action="traitement.php" method="post">
10         <label for="user_input">Entrez une valeur :</label>
11         <input type="text" id="user_input" name="user_input" required><br><br>
```

```

12     <input type="submit" value="Envoyer">
13     </form>
14 </body>
15 </html>
16
17 <?php
18 // On récupère la valeur du champ "user_input" envoyé en POST depuis le formulaire HTML
19 $user_input = $_POST['user_input'];
20 // On filtre la valeur du champ "user_input" avec la fonction htmlspecialchars() pour
    supprimer les caractères indésirables
21 $sanitized_input = htmlspecialchars($user_input, ENT_QUOTES | ENT_HTML5, 'UTF-8');
22 ?>

```

Dans cet exemple, la variable `$user_input` contient la valeur saisie par l'utilisateur via un formulaire POST. La fonction `htmlspecialchars()` prend trois arguments :

- Le type d'entrée (**INPUT_POST** pour un formulaire POST),
- Le nom de l'entrée (**user_input** dans notre exemple).

Le résultat de la fonction `htmlspecialchars()` est stocké dans la variable **\$sanitized_input**. Cette variable contient la même valeur que `$user_input`, mais avec les caractères indésirables supprimés. En utilisant la fonction `htmlspecialchars()`, nous pouvons donc filtrer les entrées utilisateur pour éviter les injections de code malveillant dans notre application. Il est recommandé d'appliquer ce filtre à toutes les entrées utilisateur avant de les utiliser dans notre code.

Attaque par inclusion SQL

Un autre type d'attaque par inclusion est l'attaque d'inclusion SQL, aussi connue sous le nom d'injection SQL. Cette faille de sécurité permet à un attaquant d'exécuter du code SQL malveillant sur le serveur de base de données en exploitant une vulnérabilité dans l'application qui n'a pas correctement validé les entrées utilisateur. L'attaque d'injection SQL peut être très dangereuse, car elle permet à l'attaquant de manipuler la base de données à sa guise. Les attaquants peuvent utiliser cette faille pour extraire des données confidentielles, modifier ou supprimer des données, ou même prendre le contrôle du serveur.

Pour éviter l'injection SQL, il est crucial de toujours valider et filtrer les entrées utilisateur avant de les utiliser dans des requêtes SQL. Les requêtes préparées sont une méthode courante pour prévenir les injections SQL en PHP. Elles utilisent des marqueurs de paramètres qui sont remplacés par les valeurs appropriées au moment de l'exécution de la requête, ce qui empêche l'attaquant de modifier le code SQL.

Exemple

Voici un exemple de code utilisant une requête préparée pour prévenir l'injection SQL :

```

1 // On définit la source de données (DSN) qui permettra de se connecter à la base de données
2 $dsn = "mysql:host=localhost;dbname=mydb;charset=utf8mb4";
3 // On définit les options pour la connexion PDO, notamment le mode d'erreur à utiliser en cas
    de problème
4 $options = [
5     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
6 ];
7 // On crée une nouvelle instance de PDO en utilisant les informations de connexion
    précédemment définies
8 $pdo = new PDO($dsn, "username", "password", $options);

```

Autre exemple : La configuration d'une connexion à la base de données en utilisant `dbname` comme nom de base de données, avec un encodage de caractères UTF-8 qui permet un plus grand nombre de caractères. La variable **\$options** est un tableau qui contient un mode de d'erreur pour le PDO. Ici on utilise le mode **PDO::ERRMODE_EXCEPTION** qui lève une exception en cas d'erreur, ce qui permet de faciliter le débogage.

```

1 <form action="votrelienduscript.php" method="post">
2   <label for="username">Nom d'utilisateur :</label>
3   <input type="text" name="username" id="username" required>
4   <input type="submit" value="Rechercher">
5 </form>
6 // On récupère la valeur du champ "username" envoyé en POST depuis le formulaire HTML
7 $user_input = $_POST['username'];
8 // On prépare une requête SQL pour sélectionner tous les utilisateurs ayant pour nom
  d'utilisateur celui envoyé depuis le formulaire
9 $stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username");
10 // On exécute la requête en passant le nom d'utilisateur comme paramètre
11 $stmt->execute(['username' => $user_input]);
12 // On récupère le résultat de la requête sous forme d'un tableau associatif
13 $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

```

Ce code permet de filtrer une entrée utilisateur pour éviter les attaques par injection SQL.

- **\$user_input** récupère la valeur de l'entrée utilisateur à partir de `$_POST['username']`.
- `$stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username");` cette requête utilise une variable de liaison `:username` pour représenter la valeur de l'entrée utilisateur dans la clause `WHERE` de notre requête `SELECT`.
- `->execute` permet d'exécuter la requête en remplaçant les paramètres liés (dans ce cas-ci, `:username`) par les valeurs spécifiées dans le tableau associatif passé en argument.
- **fetchAll()** est appelée sur l'objet de requête préparée **\$stmt** pour récupérer tous les résultats de la requête sous la forme d'un tableau associatif.

Dans la vidéo suivante, nous allons paramétrer une connexion à une base de données avec la méthode PDO, et éviter les injections de commande grâce à l'utilisation de requêtes préparées.

Exemple

Voici un exemple de code qui filtre les entrées d'utilisateurs en faisant appel à la fonction `htmlspecialchars()` permettant de convertir certains caractères envoyés par l'utilisateur, avec leur équivalence en HTML. Pratique pour filtrer les caractères qui peuvent être utilisés comme les guillemets, les apostrophes et cela afin d'éviter les injections de code malveillant. Par exemple les doubles quotes deviennent `"'"`.

```

1 <?php
2 // On utilise la fonction htmlspecialchars() pour convertir les caractères spéciaux en entités
  HTML dans les champs "username" et "password"
3 //les options ENT_QUOTES et UTF-8 prennent en compte tous les types de caractères et les
  guillemets simples et doubles.
4 $username = htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8');
5 $password = htmlspecialchars($_POST['password'], ENT_QUOTES, 'UTF-8');

```

Voici un exemple de configuration d'une requête préparée en intégrant des marqueurs de position pour ajouter une couche de sécurité supplémentaire.

```

1 $dsn = "mysql:host=localhost;dbname=faille;charset=utf8mb4";
2 $options = [
3     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
4 ];
5 $pdo = new PDO ($dsn, 'admin', 'admin', $options);
6 // On prépare la requête SQL avec des marqueurs de position
7 $pdo_prep = $pdo->prepare("SELECT * from user WHERE username = ? AND password = ? ");
8 // On exécute la requête SQL avec les valeurs des variables $username et $password comme
  arguments
9 $pdo_prep->execute([$username,$password]);
10 // On récupère le premier enregistrement de la requête SQL sous forme de tableau associatif
11 $user = $pdo_prep->fetch();

```

Voici un exemple de récupération des valeurs d'une requête dans une variable `$user` en vérifiant qu'elle soit à l'état logique `true`. En renvoyant "true", elle nous informe de la présence d'informations dans la base de données. Ainsi les données seront accessibles en faisant appel à cette variable.

```
1 //code final
2 if ($user) {
3     echo "<br>";
4     echo " Bienvenue " . $user['username'] . "!";
5 } else {
6     echo " Nom d'utilisateur ou mot de passe incorrect";
7 }
8 <?php
9 $username = htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8');
10 $password = htmlspecialchars($_POST['password'], ENT_QUOTES, 'UTF-8');
11 // Connexion a la bdd
12 $dsn = "mysql:host=localhost;dbname=faille;charset=utf8mb4";
13 $options = [
14     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
15 ];
16 // requete prepare
17 $pdo = new PDO ($dsn, 'admin', 'admin', $options);
18 $pdo_prep = $pdo->prepare("SELECT * from user WHERE username = ? AND password = ? ");
19 $pdo_prep->execute([$username,$password]);
20 $user = $pdo_prep->fetch();
21 if ($user) {
22     echo "<br>";
23     echo " Bienvenue " . $user['username'] . "!";
24 } else {
25     echo " Nom d'utilisateur ou mot de passe incorrect";
26 }
```

Afin d'éviter les actions malveillantes comme des injections SQL, il est important de sécuriser l'accès aux informations stockées dans une base de données. C'est la raison pour laquelle il est nécessaire d'utiliser les requêtes dites préparées.

\$requête-> préparée

Une requête préparée est une fonctionnalité des bibliothèques de base de données qui permet de créer une requête SQL qui peut être exécutée plusieurs fois avec des paramètres différents. La requête est préparée une seule fois, puis les paramètres sont liés à la requête avant chaque exécution. Cela signifie que les valeurs de paramètres fournies par l'utilisateur sont passées à la requête via un mécanisme sécurisé qui empêche l'utilisateur d'insérer des commandes malveillantes dans la requête SQL.

Lorsqu'on conçoit une application qui utilise une base de données, il est crucial de comprendre les risques de sécurité liés aux injections de code malveillant. Ces failles peuvent être utilisées sous différentes formes d'attaques, comme l'injection SQL et l'injection de commande.

Attaque par injection de commande

L'injection de commande est une technique d'attaque par injection qui permet à un attaquant d'exécuter des commandes système sur le serveur distant. Cette technique est souvent utilisée par les attaquants pour exécuter des commandes malveillantes sur un serveur web vulnérable, afin de prendre le contrôle du système ou de voler des données sensibles. L'injection de commande se produit généralement lorsqu'un utilisateur malveillant parvient à injecter des commandes système dans des entrées utilisateur non filtrées, telles que les paramètres de formulaire ou les champs de recherche.

Méthode Éviter les injections de commande

Nous allons voir comment éviter ces injections de commande dans l'exemple suivant :

On commence par initialiser des variables d'erreurs de type string pour le message d'erreur, la valeur de sortie et le champ commande.

```
1 // On initialise les variables $commandErr, $output et $command à des chaînes vides
2 $commandErr = "";
3 $output = "";
4 $command = " ";
```

Exemple

Les données utilisateur contenant des caractères spéciaux peuvent être interprétées par le serveur web comme des commandes à exécuter ou du code HTML à afficher. Ceci peut permettre à un attaquant de compromettre l'application et d'exécuter des commandes malveillantes, telles que la suppression de fichiers, l'exécution de scripts, la prise de contrôle du système et l'accès à des informations sensibles. Pour éviter ces attaques, il est essentiel de traiter correctement les données utilisateur.

```
1 <!-- Formulaire qui permet d'envoyer une commande en méthode POST vers la page courante -->
2 <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
3 <!-- Champ de texte qui permet de saisir la commande, sa valeur est récupérée dans la variable
   $command -->
4   Commande:
5   <input type="text" name="command" value="<?php echo htmlspecialchars($command);?>"
6 <!-- Affichage d'un message d'erreur s'il y a une erreur dans la commande -->
7   <span class="error"><?php echo $commandErr;?></span>
8   <br><br>
9 <!-- Bouton qui permet de soumettre le formulaire -->
10  <input type="submit" name="submit" value="Exécuter la commande">
11 </form>
```

Ce code apporte des règles de sécurité à mettre en place pour traiter efficacement des entrées utilisateur en appliquant des mesures du début à la fin dans le but de récupérer une information sécurisée.

```
1 // Vérifie si la méthode de la requête est POST et si le champ "command" a été soumis
2 if ($_SERVER["REQUEST_METHOD"] == "POST") {
3 // On vérifie si le champ "command" a été rempli
4 if (!isset($_POST["command"])) {
5 // Si le champ est vide, on envoie un message d'erreur
6 $commandErr = "La commande est requise";
7 // Sinon, on filtre la valeur du champ avec la fonction filter_form()
8 } else {
9 $command = filter_form($_POST["command"]);
```

Voici un exemple de fonction qui nettoie et sécurise des données en traitant la valeur d'entrée par plusieurs fonctions internes. En faisant appel à une fonction, il est possible de cibler un élément appliquant un script spécifique. Ici, il est question de validation de la valeur **\$data** en utilisant des sous fonctions de validation de l'entrée utilisateur, qui après traitement devient plus sûre et sécurisée.

```
1 // Cette fonction filtre les données d'un formulaire en enlevant les espaces inutiles en début
   et fin de chaîne, en supprimant les antislashes ajoutés pour échapper les caractères spéciaux
   et en convertissant les caractères spéciaux en entités HTML. Elle renvoie les données
   filtrées.
2 function filter_form($data) {
3   $data = trim($data);
4   $data = stripslashes($data);
5   $data = htmlspecialchars($data);
6   return $data;
7 }
```

- trim() supprime les espaces en début et fin de chaîne,
- stripslashes() supprime les antislashes,
- htmlspecialchars() convertit les caractères spéciaux en entités HTML.

Voici un exemple de code qui fait usage des expressions régulières pour vérifier que l'entrée **\$command** ne contient que des caractères alphanumériques ainsi que des espaces. Si c'est le cas, elle renverra un message d'erreur.

- ^ : début de la chaîne.
- [a-zA-Z0-9\s] : correspond à n'importe quel caractère alphanumérique (lettre ou chiffre) ou espace.
- + : signifie que le motif précédent doit être présent au moins une fois.
- \$: fin de la chaîne.

```

1 if (!preg_match('/^[a-zA-Z0-9\s]+$/', $command)) {
2     $commandErr = "L'entrée utilisateur n'est pas valide";
3 }
4 }
5 }

1 <?php
2 // Initialiser les variables d'erreur
3 $commandErr = "";
4 $output = "";
5 $command = " ";
6 // Vérifier si le formulaire a été soumis
7 if ($_SERVER["REQUEST_METHOD"] == "POST") {
8     // Valider et filtrer l'entrée utilisateur
9     if (!isset($_POST["command"])) {
10         $commandErr = "La commande est requise";
11     } else {
12         $command = test_input($_POST["command"]);
13         // Vérifier que l'entrée utilisateur ne contient que des caractères alphanumériques et des
        espaces
14         if (!preg_match('/^[a-zA-Z0-9\s]+$/', $command)) {
15             $commandErr = "L'entrée utilisateur n'est pas valide";
16         }
17     }
18     // Exécuter la commande si l'entrée utilisateur est valide
19     if ($commandErr == "") {
20         // Exécuter la commande en utilisant la fonction shell_exec()
21         $output = shell_exec($command);
22     }
23 }
24 // Fonction pour filtrer les données d'entrée utilisateur
25 function test_input($data) {
26     $data = trim($data);
27     $data = stripslashes($data);
28     $data = htmlspecialchars($data);
29     return $data;
30 }
31 ?>
32 <!DOCTYPE html>
33 <html>
34 <head>
35     <title>Exemple de formulaire pour éviter l'injection de commande</title>
36 </head>
37 <body>

```



```
38 <h2>Exemple de formulaire pour éviter l'injection de commande</h2>
39 <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
40   Commande: <input type="text" name="command" value="<?php echo htmlspecialchars($command);?
41   >">
42   <span class="error"><?php echo $commandErr;?></span>
43   <br><br>
44   <input type="submit" name="submit" value="Exécuter la commande">
45 </form>
46 <div>
47   <?php echo $output; ?>
48 </div>
49 </body>
50 </html>
```

Voici un exemple de formulaire qui permet d'éviter les attaques malveillantes qui peuvent se produire lorsque l'utilisateur saisit une commande dans un champ de saisie. **shell_exec()** est utilisé pour exécuter une commande via un terminal et afficher le résultat. Afin de prévenir les attaques malveillantes, l'entrée utilisateur est filtrée à l'aide de la fonction **filter_form()**.

En plus de la fonction de filtrage, le code prend également en compte la validation de l'entrée utilisateur avant l'exécution de la commande en vérifiant que seuls les caractères alphanumériques et les espaces sont présents. Si l'entrée utilisateur contient d'autres caractères, une erreur est renvoyée à l'utilisateur. En stockant les messages d'erreur et le résultat de la commande dans des variables, le code assure une présentation claire et efficace des informations à l'utilisateur. Cela contribue à renforcer la sécurité de l'application en réduisant les risques d'exploitation des failles de sécurité du formulaire. En adoptant ces pratiques, nous réduisons les risques d'attaques par injection de commande et garantissons la sécurité d'un projet web.

B. Exercice : Quiz

[solution n°1 p.19]

Question 1

Qu'est-ce que l'injection de code en PHP ?

- ☐ Une attaque permettant l'attraction de données
- ☐ Une attaque permettant l'exécution de commandes sur le système d'exploitation
- ☐ Une attaque permettant l'insertion de code dans une application

Question 2

Comment les injections peuvent-elles être évitées ?

- ☐ En utilisant des variables aléatoires dans le PHP
- ☐ En validant les entrées utilisateur
- ☐ En désactivant le pare-feu

Question 3

Que fait la fonction `htmlspecialchars` en plus de filtrer les variables ?

- ☐ Elle ajoute une entrée de formulaire
- ☐ Elle propose des types de filtres pour les entrées de formulaire
- ☐ Elle ajoute un champ dans un formulaire

Question 4

Qu'est l'injection la plus courante en PHP ?

- ☐ Injection de code JavaScript
- ☐ Injection SQL
- ☐ Injection PHP

Question 5

Comment les injections SQL peuvent-elles être évitées ?

- ☐ En renforçant les mots de passe
- ☐ En limitant l'accès à des fichiers système
- ☐ En utilisant des requêtes préparées

III. Les attaques XSS, CSRF et le Hijacking

A. Les attaques XSS, CSRF et le Hijacking

Si vous êtes un débutant en PHP, il est crucial de comprendre les vulnérabilités de sécurité courantes liées à ce langage de programmation. Cette fois-ci, nous verrons les attaques XSS (Cross Site Scripting).

Il en existe plusieurs types, mais elles impliquent généralement l'injection de code JavaScript dans une page web. Les attaquants peuvent utiliser cette technique pour voler des informations sensibles telles que des mots de passe ou pour rediriger les utilisateurs vers des sites web malveillants. En tant que débutant en PHP, la validation des entrées utilisateur est la première arme pour éviter des attaques. Nous avons déjà vu l'usage de `htmlspecialchars()` qui convertit les caractères spéciaux en entités HTML, empêchant le navigateur de les interpréter comme du JavaScript, mais aussi `htmlspecialchars()`, `addslashes()` et encore bien d'autres fonctions de sécurité qui, si bien configurées, permettent de réduire les risques d'actes malveillants. Cette fois-ci, nous allons observer le comportement de l'une de ces fonctions de sécurité, **`strip_tags()`**, qui permet de supprimer toutes les balises HTML et PHP potentiellement malveillantes dans le cas d'un téléchargement de fichier.

Dans cette vidéo, nous avons vu des mesures pour éviter les attaques XSS qui pourraient permettre à des personnes malveillantes de prendre le contrôle du navigateur d'un utilisateur. Ces mesures incluent la suppression de toutes les balises HTML et tous les caractères spéciaux qui pourraient être utilisés pour injecter des scripts malicieux.

La fonction "**`strip_tags()`**" est utilisée pour supprimer toutes les balises HTML et PHP du nom du fichier téléchargé avant de l'utiliser dans le script. Cela permet d'éviter les attaques XSS qui pourraient être incluses dans le nom du fichier. En utilisant "`strip_tags`", on s'assure que le nom de fichier téléchargé est composé uniquement de texte, ce qui le rend plus sûr à utiliser dans le script. Cela réduit également le risque d'erreurs lors de l'utilisation du nom de fichier dans d'autres parties du script.

```
1 // Vérifie si le formulaire a été soumis en vérifiant si le bouton submit a été utilisé
2 if (isset($_POST['submit'])){
3 // Récupère le nom du fichier en supprimant les balises HTML et PHP potentiellement
  dangereuses
4 $file_name = strip_tags($_FILES['file']['name']);
5 // Récupère la taille du fichier envoyé
6 $file_size = $_FILES['file']['size'];
7 // Récupère l'emplacement temporaire du fichier envoyé sur le serveur
8 $file_tmp = $_FILES ['file']['tmp_name'];
9 // Récupère le type MIME du fichier envoyé
10 $file_type = $_FILES['file']['type'];
11 // Sépare le nom de fichier et son extension
12 $file_ext = explode('.', $file_name);
13 // Récupère l'extension du fichier en transformant la chaîne en minuscule
14 $file_end = end($file_ext);
15 $file_end = strtolower($file_end);
```

```

16 // Définit la liste des extensions de fichiers autorisées
17 $extensions = [ 'doc', 'jpg', 'docx', 'xls'];}

```

Cet exemple de code commence par vérifier si l'élément "submit" a été défini dans la méthode POST. Ensuite, il récupère les informations sur le fichier téléchargé, y compris le nom, la taille, le type et l'emplacement temporaire du fichier. Il utilise ensuite la fonction "explode" pour extraire l'extension du fichier et stocke cette extension dans la variable "file_end". La variable "file_end" est ensuite convertie en minuscules pour faciliter la comparaison avec les extensions de fichier autorisées stockées dans un tableau.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Télécharger un fichier</title>
5 </head>
6 <body>
7   <h1>Télécharger un fichier</h1>
8   <form method="post" enctype="multipart/form-data">
9     <input type="file" name="file" required>
10    <button type="submit" name="submit">Télécharger</button>
11  </form>
12 </body>
13 </html>
14 <?php
15 if (isset($_POST['submit'])){
16 // récupération des informations de notre fichier
17 $file_name = strip_tags($_FILES['file']['name']);
18 $file_size = $_FILES['file']['size'];
19 $file_tmp = $_FILES ['file']['tmp_name'];
20 $file_type = $_FILES['file']['type'];
21
22 $file_ext = explode('.', $file_name); // cahier.jpg => ['cahier' , 'jpg']
23 $file_end = end($file_ext); // jpg $
24 $file_end = strtolower($file_end);
25 $extensions = [ 'doc', 'jpg', 'docx', 'xls'];
26 if(in_array($file_end, $extensions) === false) {
27 // Si l'extension du fichier n'est pas autorisée, on affiche un message d'erreur
28   echo " Veuillez utiliser les extensions suivantes : DOC, JPG , DOCX , XLS";
29 }
30 // Si la taille du fichier dépasse la limite autorisée, on affiche un message d'erreur
31 elseif($file_size > 300000) {
32   echo " le fichier est trop volumineux";
33 }else {
34 // On nettoie le nom de fichier en supprimant les caractères spéciaux
35   $file_name = preg_replace('/[^A-Za-z0-9.\-]/', '', $file_name);
36 // On déplace le fichier uploadé vers le répertoire "uploads" avec son nom d'origine
37   move_uploaded_file($file_tmp, "uploads/".$file_name);
38 // On affiche un message de succès pour indiquer que le téléchargement a réussi
39   echo " Le fichier ".$file_name." a été téléchargé avec succès";
40 }
41 }

```

Ce code utilise une structure conditionnelle pour vérifier si l'extension du fichier téléchargé est valide et si sa taille est inférieure à 300 ko. Si l'extension n'est pas valide, un message d'erreur est affiché pour demander à l'utilisateur d'utiliser une extension de fichier différente. Si la taille est supérieure à 300 ko, un autre message d'erreur est affiché. Si le fichier est valide, le code utilise la fonction "preg_replace" pour nettoyer le nom de fichier de tout caractère spécial, puis utilise la fonction "move_uploaded_file" pour déplacer le fichier téléchargé vers un dossier nommé "uploads". Enfin, un message de confirmation est affiché pour informer l'utilisateur que le fichier a été téléchargé avec succès.

Maintenant, nous allons voir un autre type d'attaque malveillante, les attaques CSRF.

Attaque CSRF et Hijacking

Ce sont des attaques qui exploitent l'utilisation de cookies pour l'authentification et les sessions. Les attaquants se font passer pour un utilisateur légitime. Ils exploitent le processus d'envoi de formulaire en incitant l'utilisateur à soumettre un formulaire piégé sur un site malveillant. Si l'utilisateur est connecté à un site légitime, la requête POST est authentifiée avec les cookies du site, permettant à l'attaquant de réaliser une action malveillante. Pour se protéger contre les attaques de session, nous allons utiliser des jetons CSRF qui sont des chaînes aléatoires incluses dans les formulaires HTML et vérifiées par le serveur. Si le jeton n'est pas valide, la requête sera rejetée.

Méthode	Jeton CSRF
1	<?php
2	// On démarre la session et on vérifie si la méthode de requête est POST.
3	session_start();
4	if (\$_SERVER['REQUEST_METHOD'] === 'POST') {
5	// Si le jeton CSRF n'est pas présent ou s'il ne correspond pas au jeton stocké en session, on
6	arrête le processus et affiche une erreur.
7	if (!isset(\$_POST['csrf_token']) \$_POST['csrf_token'] !== \$_SESSION['csrf_token']) {
8	die('Erreur CSRF !');
9	}
10	?>

Cet exemple de condition vérifie si la variable "csrf_token" dans la méthode POST est bien définie et si elle correspond à la variable "csrf_token" stockée dans la session.

```
1 // Génère un token aléatoire de 32 octets à partir de la fonction random_bytes() de PHP, puis
  le convertit en une chaîne hexadécimale et le stocke dans la variable de session 'csrf_token'.
2 $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
```

Cette ligne de code génère un nouveau jeton CSRF de 32 octets en utilisant la fonction **random_bytes()**. Ensuite, ce jeton est converti en une chaîne de caractères hexadécimaux à l'aide de la fonction **bin2hex()**. Ce jeton sera ensuite utilisé pour vérifier si les requêtes soumises à partir du formulaire de connexion proviennent bien de la page actuelle et non d'un attaquant.

Lorsque la condition `if ($_SESSION['csrf_token'] === $_POST['csrf_token'])` est vraie, cela signifie que le jeton CSRF envoyé avec le formulaire soumis correspond au jeton CSRF stocké dans la session utilisateur. Cela garantit que le formulaire a été soumis par un utilisateur légitime qui a chargé la page précédente. À la suite de cela, nous paramétrons nos informations de connexion à la base de données.

```
1 <?php
2 session_start(); // Démarrer la session
3 // Vérification CSRF
4 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
5     if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
6         die('Erreur CSRF !');
7     } else {
8         // Jeton CSRF défini ou celui-ci =
9         echo "<br>";
10        echo " Jeton CSRF transmis";
11        echo "<br>";
12    }
13 }
14 // Générer un nouveau jeton CSRF
15 $_SESSION['csrf_token'] = bin2hex(random_bytes(32))
16 // Connexion à la base de données
17 $host = 'localhost';
18 $username = 'admin';
```

```

19 $password = 'admin';
20 $dbname = 'faillie';
21 try {
22     $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
23 } catch (PDOException $e) {
24     echo "Erreur : " . $e->getMessage();
25 }
26 // Vérification des informations d'identification
27 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
28     $username = $_POST['username'];
29     $password = $_POST['password'];
30     $stmt = $pdo->prepare('SELECT * FROM user WHERE username = ? AND password = ?');
31     $stmt->execute([$username, $password]);
32     $user = $stmt->fetch();
33     if ($user) {
34         // Authentification réussie
35         $_SESSION['user_id'] = $user['id'];
36         echo " Connexion réussie";
37         var_dump( $_SESSION);
38         exit;
39     } else {
40         // Authentification échouée
41         $error = 'Adresse e-mail ou mot de passe incorrect.';
42     }
43 }
44 ?>
45
46 <!DOCTYPE html>
47 <html>
48 <head>
49     <meta charset="UTF-8">
50     <title>Formulaire de connexion</title>
51 </head>
52 <body>
53     <?php if (isset($error)) { ?>
54         <p><?php echo $error; ?></p>
55     <?php } ?>
56     <form method="post">
57         <input type="text" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
58         <label for="username">username</label>
59         <input type="text" id="username" name="username" required>
60         <label for="password">Mot de passe :</label>
61         <input type="password" id="password" name="password" required>
62         <button type="submit">Se connecter</button>
63     </form>
64 </body>
65 </html>

```

Voici un exemple de vérification d'authentification pour un formulaire de connexion. Il vérifie si la méthode de requête HTTP est POST, récupère les informations de nom d'utilisateur et de mot de passe soumises par le formulaire, puis exécute une requête SQL **SELECT** sur une table utilisateur pour vérifier si les informations sont valides. Si les informations sont valides, la session utilisateur est démarrée en stockant l'ID utilisateur dans la variable de session "user_id". Un message "Connexion réussie" est affiché et le contenu de la variable de session est affiché à l'aide de la fonction `var_dump()`. Si les informations d'identification sont invalides, un message d'erreur "Adresse e-mail ou mot de passe incorrect." est stocké dans la variable `$error`.

En conclusion, l'utilisation d'un jeton CSRF est une mesure de sécurité importante pour prévenir ces attaques. En ajoutant un jeton CSRF dans les formulaires, le serveur peut vérifier que la requête est légitime et éviter que des attaquants ne puissent envoyer des requêtes malveillantes en usurpant l'identité de l'utilisateur. Cette vérification permet d'assurer que les requêtes proviennent bien de l'utilisateur authentifié et évite les risques d'injection de code malveillant. Vous devez donc prendre en compte la sécurité dès la conception de vos applications et inclure des mesures de protection telles que l'utilisation de jetons CSRF pour assurer la sécurité des utilisateurs et des données sensibles.

Hijacking

Le détournement de session (session hijacking) est une attaque où un attaquant intercepte la session d'un utilisateur authentifié et l'utilise pour accéder à l'application web.

Pour prévenir le détournement de session, il est important d'utiliser des mesures de sécurité appropriées :

- Utiliser HTTPS pour chiffrer la communication entre le navigateur et le serveur.
- Stocker les informations de session dans un emplacement sécurisé, tel qu'une base de données, pour empêcher l'accès non autorisé aux sessions.
- Renouveler les sessions après une période définie pour limiter la durée de la fenêtre d'attaque.

En outre, les utilisateurs peuvent également prendre des mesures pour protéger leurs sessions, comme utiliser des mots de passe forts et différents pour chaque compte ou se déconnecter de l'application web après utilisation. Mais aussi éviter d'utiliser des réseaux Wi-Fi publics non sécurisés pour accéder à des applications web sensibles. En suivant ces mesures de sécurité, les développeurs et les utilisateurs peuvent contribuer à prévenir le détournement de session et à renforcer la sécurité de leurs applications web.

B. Exercice : Quiz

[solution n°2 p.20]

Question 1

Qu'est-ce qu'une attaque XSS ?

- ☐ Une attaque qui permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur
- ☐ Une attaque qui permet à un attaquant de voler les informations d'identification d'un utilisateur
- ☐ Une attaque qui permet à un attaquant d'accéder à la base de données d'un site web

Question 2

Comment se prémunir du hijacking dans le contexte informatique ?

- ☐ En utilisant des mots de passe complexes et en les changeant régulièrement
- ☐ En n'acceptant pas les cookies
- ☐ En utilisant un logiciel antivirus et un pare-feu pour protéger votre système

Question 3

Comment un attaquant peut-il tirer profit d'une vulnérabilité XSS ?

- ☐ En volant des informations d'identification (username, password)
- ☐ En redirigeant l'utilisateur vers un site web malveillant
- ☐ En exécutant du code JavaScript malveillant sur le navigateur de l'utilisateur

Question 4

Comment peut-on prévenir les attaques CSRF dans une application web ?

- ☐ En validant côté serveur les données saisies par l'utilisateur
- ☐ En utilisant un token CSRF pour chaque requête
- ☐ En nettoyant les données saisies par l'utilisateur avant de les afficher sur la page

Question 5

Quelle est la méthode de prévention recommandée pour les attaques XSS ?

- ☐ Validation côté serveur des données saisies par l'utilisateur
- ☐ Utilisation d'un token CSRF pour chaque requête
- ☐ Nettoyage des données saisies par l'utilisateur avant de les afficher sur la page

IV. Essentiel

Pour conclure, la sécurité est un aspect essentiel lors du développement et de la maintenance d'une application web. Les vulnérabilités telles que l'injection de code, le Cross-Site Scripting (XSS) et la falsification de requêtes entre sites (CSRF) sont courantes et peuvent causer de graves problèmes en termes de sécurité de l'application et de données des utilisateurs. Il est crucial de prendre en compte la sécurité dès le début du processus de développement. La mise en place de mesures préventives telles que la validation stricte des entrées utilisateur, la gestion sécurisée des sessions, la mise à jour et l'utilisation de HTTPS est indispensable pour renforcer la sécurité de votre application web. Il est également important de souligner que la sécurité ne doit pas être traitée comme un problème ponctuel, mais plutôt comme un processus continu. Il est important de surveiller en permanence votre application web afin de détecter rapidement les vulnérabilités et de les corriger. En fin de compte, la sécurité est une responsabilité partagée entre les développeurs, les administrateurs système et les utilisateurs finaux. En collaborant, nous pouvons tous contribuer à renforcer la sécurité de nos applications web et protéger les données des utilisateurs contre les attaques malveillantes.

Voici un récapitulatif des mesures de prévention pour éviter les failles de sécurité en PHP :

- Les entrées utilisateur doivent être validées de manière stricte pour éviter les injections de code malveillant. Il est recommandé d'utiliser des fonctions de validation intégrées dans PHP.
- Stocker les informations de session dans un emplacement sécurisé tel qu'une base de données, plutôt que dans des fichiers sur le serveur. Utiliser des identifiants de session robustes et renouveler les sessions après une période définie.
- Mettre à jour régulièrement les packages et les bibliothèques tiers utilisés dans une application web PHP pour éviter les vulnérabilités connues et obtenir les dernières fonctionnalités et améliorations de sécurité.
- Mettre en place le protocole HTTPS pour protéger les données sensibles transmises entre le client et le serveur.
- Éviter d'exécuter ses requêtes de base de données avec l'utilisateur ROOT pour éviter les injections SQL.

Il est important de combiner ces mesures de prévention pour réduire le risque d'exécution de code à distance sur un serveur PHP et renforcer la sécurité de votre application web. La sécurité est une responsabilité partagée entre les développeurs, les administrateurs système et les utilisateurs finaux, il est donc important de travailler ensemble pour améliorer la sécurité et protéger les données des utilisateurs contre les attaques malveillantes.

V. Auto-évaluation

A. Exercice

Vous êtes un développeur web travaillant sur une application qui affiche des messages sur une page. Les utilisateurs peuvent soumettre des messages en utilisant un formulaire. Cependant, vous avez récemment découvert que certains utilisateurs malveillants ont commencé à soumettre des messages contenant des scripts XSS, ce qui permet aux attaquants de voler des informations utilisateur.

Question 1

[solution n°3 p.21]

Que pouvez-vous faire pour protéger votre application contre les attaques XSS ?

Question 2

[solution n°4 p.21]

Vous devez également vous assurer que les utilisateurs soumettant des messages ne peuvent pas soumettre des liens. Comment pouvez-vous implémenter cette sécurité supplémentaire pour protéger les utilisateurs de votre application ? Fournissez un exemple de code pour illustrer comment valider les entrées utilisateur pour les liens.

B. Test

Exercice 1 : Quiz

[solution n°5 p.22]

Question 1

Quelle est l'une des façons les plus courantes de se protéger contre les attaques de hijacking de session ?

- ☐ Utiliser SSL / TLS pour chiffrer les communications entre le serveur et le navigateur
- ☐ Définir une durée d'expiration de session courte
- ☐ Utiliser une méthode d'authentification forte

Question 2

Qu'est-ce qu'une attaque de hameçonnage (phishing) ?

- ☐ Une attaque dans laquelle un attaquant utilise une application tierce pour accéder à un compte utilisateur
- ☐ Une attaque dans laquelle un attaquant utilise une technique de piratage pour accéder à un réseau informatique
- ☐ Une attaque dans laquelle un attaquant utilise des e-mails ou des messages instantanés pour inciter les utilisateurs à divulguer leurs informations d'identification

Question 3

Considérez le code PHP suivant qui interagit avec une base de données MySQL. Quelle est la principale vulnérabilité de ce code et comment pouvez-vous la corriger ?

```
1 $username = $_POST['username'];
2 $password = $_POST['password'];
3 $conn = mysqli_connect('localhost', 'user', 'password', 'mydb');
4 $query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
5 $result = mysqli_query($conn, $query);
6 if(mysqli_num_rows($result) == 1) {
7     echo "Bienvenue " . $username;
8 } else {
9     echo "Nom d'utilisateur ou mot de passe incorrect";
10 }
```

- ☐ La principale vulnérabilité de ce code est le stockage de mots de passe en texte clair dans la base de données. Pour corriger cela, on peut utiliser une fonction de hachage comme password_hash() pour stocker les mots de passe de manière sécurisée
- ☐ La principale vulnérabilité de ce code est le Cross-Site Scripting (XSS), car les entrées de l'utilisateur ne sont ni échappées ni validées avant d'être affichées à l'écran. Pour corriger cela, on peut utiliser htmlspecialchars() pour échapper les caractères spéciaux dans les entrées de l'utilisateur
- ☐ La principale vulnérabilité de ce code est l'injection SQL, car les entrées de l'utilisateur ne sont ni échappées ni validées avant d'être utilisées dans la requête SQL. Pour corriger cela, on peut utiliser PDO (PHP Data Objects) avec des requêtes préparées

Question 4

Quelle est la vulnérabilité présente dans ce code PHP ?

```
1 <form action="transfer_funds.php" method="post">
2   <input type="text" name="amount" placeholder="Amount">
3   <input type="text" name="account_number" placeholder="Account Number">
4   <button type="submit">Transfer Funds</button>
5 </form>
```

- ☐ Injection SQL
- ☐ Attaque XSS
- ☐ Attaque CSRF

Question 5


Quelle est la principale différence entre XSS et CSRF ?

- ☐ XSS permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur tandis que CSRF permet à un attaquant d'effectuer des actions non autorisées au nom de l'utilisateur
- ☐ XSS permet à un attaquant d'effectuer des actions non autorisées au nom de l'utilisateur tandis que CSRF permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur
- ☐ Il n'y a pas de différence entre XSS et CSRF, ce sont des termes interchangeables pour la même vulnérabilité de sécurité

Solutions des exercices


Exercice p. 9 Solution n°1**Question 1**

Qu'est-ce que l'injection de code en PHP ?

- ☐ Une attaque permettant l'attraction de données
- ☐ Une attaque permettant l'exécution de commandes sur le système d'exploitation
- ☒ Une attaque permettant l'insertion de code dans une application
-  Une injection de code en PHP est une attaque où l'attaquant insère du code malveillant dans les entrées utilisateur.


Question 2

Comment les injections peuvent-elles être évitées ?

- ☐ En utilisant des variables aléatoires dans le PHP
- ☒ En validant les entrées utilisateur
- ☐ En désactivant le pare-feu
-  La validation d'entrée est la première étape primordiale pour sécuriser son code PHP.


Question 3

Que fait la fonction htmlspecialchars en plus de filtrer les variables ?

- ☐ Elle ajoute une entrée de formulaire
- ☒ Elle propose des types de filtres pour les entrées de formulaire
- ☐ Elle ajoute un champ dans un formulaire
-  La fonction htmlspecialchars est couramment utilisée pour proposer des filtres afin de valider les entrées utilisateur notamment en encodant les caractères spéciaux


Question 4

Qu'est l'injection la plus courante en PHP ?

- ☐ Injection de code JavaScript
- ☒ Injection SQL
- ☐ Injection PHP
-  L'injection SQL est l'attaque la plus courante en PHP pour exécuter du code malveillant.

Question 5


Comment les injections SQL peuvent-elles être évitées ?

- ☐ En renforçant les mots de passe
- ☐ En limitant l'accès à des fichiers système
- ☒ En utilisant des requêtes préparées
-  La meilleure façon d'éviter les injections SQL est d'utiliser des requêtes préparées et des paramètres liés.

Exercice p. 14 Solution n°2


Question 1

Qu'est-ce qu'une attaque XSS ?

- ☒ Une attaque qui permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur
- ☐ Une attaque qui permet à un attaquant de voler les informations d'identification d'un utilisateur
- ☐ Une attaque qui permet à un attaquant d'accéder à la base de données d'un site web
-  XSS (Cross-Site Scripting) est une attaque qui permet à un attaquant d'injecter du code malveillant dans une page web, qui sera exécuté sur le navigateur d'un utilisateur qui consulte cette page.


Question 2

Comment se prémunir du hijacking dans le contexte informatique ?

- ☒ En utilisant des mots de passe complexes et en les changeant régulièrement
- ☐ En n'acceptant pas les cookies
- ☒ En utilisant un logiciel antivirus et un pare-feu pour protéger votre système
-  Pour prévenir ce type d'attaque, les mots de passe forts et la vigilance sont des mesures à respecter pour éviter de se faire subtiliser son identité sur internet. L'utilisation d'un antivirus et d'un pare-feu est vivement conseillée.

Question 3

Comment un attaquant peut-il tirer profit d'une vulnérabilité XSS ?

- ☐ En volant des informations d'identification (username, password)
- ☐ En redirigeant l'utilisateur vers un site web malveillant
- ☒ En exécutant du code JavaScript malveillant sur le navigateur de l'utilisateur
-  Une vulnérabilité XSS permet à un attaquant d'injecter du code JavaScript malveillant sur un site web légitime, qui sera alors exécuté sur le navigateur de l'utilisateur qui accède à ce site. L'attaquant peut voler des informations confidentielles, telles que des cookies ou des données de formulaires, ou rediriger l'utilisateur vers un site web malveillant.

Question 4

Comment peut-on prévenir les attaques CSRF dans une application web ?

- ☐ En validant côté serveur les données saisies par l'utilisateur
- ☒ En utilisant un token CSRF pour chaque requête
- ☐ En nettoyant les données saisies par l'utilisateur avant de les afficher sur la page

- Q Recourir à un jeton CSRF pour chaque requête est une technique très recommandée pour prévenir les attaques CSRF.

Question 5

Quelle est la méthode de prévention recommandée pour les attaques XSS ?

- ☐ Validation côté serveur des données saisies par l'utilisateur
- ☐ Utilisation d'un token CSRF pour chaque requête
- ☒ Nettoyage des données saisies par l'utilisateur avant de les afficher sur la page

- Q Le nettoyage (sanitization) des données saisies par l'utilisateur avant de les afficher sur la page est une méthode efficace pour prévenir ce type d'attaque.

p. 16 Solution n°3

```
1 if(isset($_POST['submit'])){
2     $message = $_POST['message'];
3     $message = htmlspecialchars($message, ENT_QUOTES, 'UTF-8');}
```

Avec ce code, le message soumis par l'utilisateur est récupéré dans la variable `$message`. Ensuite, la fonction **htmlspecialchars()** est utilisée pour échapper les caractères spéciaux dans le message avant de l'afficher sur la page pour éviter les attaques XSS. Cela permet de protéger les utilisateurs contre les attaques XSS en empêchant l'exécution de code malveillant provenant d'entrées utilisateur dans les pages de l'application web.

p. 16 Solution n°4

```
1 <?php
2 If (isset($_POST['submit']) && isset($_POST['message'])) {
3     $message = $_POST['message'];
4     $message = htmlspecialchars($message, ENT_QUOTES, 'UTF-8');
5     $url = filter_var($message, FILTER_VALIDATE_URL);
6 }
7 ??
8 <!DOCTYPE html>
9 <html>
10 <head>
11     <title>Cas pratique</title>
12 </head>
13 <body>
14     <h1>Formulaire</h1>
15     <form method="post">
16         <label for="message">Message:</label><br>
17         <textarea name="message" id="message" rows="5" cols="40"><?php if($url !== false ){echo
18 "Lien interdit";} else{ echo $message; } ?></textarea><br>
19         <input type="submit" name="submit" value="Soumettre">
20     </form>
21 </body>
22 </html>
```

Afin de sécuriser davantage une application, il est possible d'empêcher les utilisateurs de soumettre des liens dans les messages. Pour cela, on peut utiliser la fonction `filter_var()` avec l'option `FILTER_VALIDATE_URL` pour valider si le contenu de la variable `$message` est un lien valide. Si la validation réussit, cela indique que l'utilisateur a soumis un lien, et un message d'erreur peut être affiché à la place du lien pour avertir l'utilisateur qu'il n'est pas autorisé à soumettre des liens.

Exercice p. 16 Solution n°5

Question 1

Quelle est l'une des façons les plus courantes de se protéger contre les attaques de hijacking de session ?

- ☐ Utiliser SSL / TLS pour chiffrer les communications entre le serveur et le navigateur
- ☐ Définir une durée d'expiration de session courte
- ☒ Utiliser une méthode d'authentification forte

Q L'authentification à deux facteurs est une méthode couramment utilisée pour se protéger contre les attaques de hijacking de session. Cette méthode exige une vérification supplémentaire de l'identité de l'utilisateur, rendant l'accès au compte plus difficile pour un attaquant, même s'il a volé les informations d'identification.

Question 2

Qu'est-ce qu'une attaque de hameçonnage (phishing) ?

- ☐ Une attaque dans laquelle un attaquant utilise une application tierce pour accéder à un compte utilisateur
- ☐ Une attaque dans laquelle un attaquant utilise une technique de piratage pour accéder à un réseau informatique
- ☒ Une attaque dans laquelle un attaquant utilise des e-mails ou des messages instantanés pour inciter les utilisateurs à divulguer leurs informations d'identification

Q Le hameçonnage est une technique d'attaque qui exploite la confiance des utilisateurs pour obtenir leurs informations d'identification en se faisant passer pour une source de confiance. Cette attaque peut être combinée avec d'autres vulnérabilités de sécurité, telles que les vulnérabilités XSS, pour obtenir des résultats plus efficaces.

Question 3

Considérez le code PHP suivant qui interagit avec une base de données MySQL. Quelle est la principale vulnérabilité de ce code et comment pouvez-vous la corriger ?

```
1 $username = $_POST['username'];
2 $password = $_POST['password'];
3 $conn = mysqli_connect('localhost', 'user', 'password', 'mydb');
4 $query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
5 $result = mysqli_query($conn, $query);
6 if(mysqli_num_rows($result) == 1) {
7     echo "Bienvenue " . $username;
8 } else {
9     echo "Nom d'utilisateur ou mot de passe incorrect";
10 }
```

- ☐ La principale vulnérabilité de ce code est le stockage de mots de passe en texte clair dans la base de données. Pour corriger cela, on peut utiliser une fonction de hachage comme password_hash() pour stocker les mots de passe de manière sécurisée
- ☐ La principale vulnérabilité de ce code est le Cross-Site Scripting (XSS), car les entrées de l'utilisateur ne sont ni échappées ni validées avant d'être affichées à l'écran. Pour corriger cela, on peut utiliser htmlspecialchars() pour échapper les caractères spéciaux dans les entrées de l'utilisateur
- ☒ La principale vulnérabilité de ce code est l'injection SQL, car les entrées de l'utilisateur ne sont ni échappées ni validées avant d'être utilisées dans la requête SQL. Pour corriger cela, on peut utiliser PDO (PHP Data Objects) avec des requêtes préparées

- Q La vulnérabilité principale de ce code est l'injection SQL. Utiliser PDO avec des requêtes préparées est une solution efficace pour se protéger contre cette attaque. Les requêtes préparées permettent de séparer la requête SQL de ses paramètres en utilisant des marqueurs de paramètres et de les lier ultérieurement avec les valeurs appropriées, ce qui empêche les attaquants d'injecter du code SQL malveillant dans la requête.

Question 4

Quelle est la vulnérabilité présente dans ce code PHP ?

```
1 <form action="transfer_funds.php" method="post">
2   <input type="text" name="amount" placeholder="Amount">
3   <input type="text" name="account_number" placeholder="Account Number">
4   <button type="submit">Transfer Funds</button>
5 </form>
```

- ☐ Injection SQL
- ☐ Attaque XSS
- ☒ Attaque CSRF

- Q La principale vulnérabilité de ce code est l'attaque CSRF, car il n'y a aucun mécanisme de protection contre les requêtes forgées. Les jetons CSRF protègent contre les attaques d'injection de code en générant des chaînes de caractères uniques pour chaque formulaire. Vérifiant l'authenticité de la requête lors de la soumission, empêchant les attaquants d'usurper l'identité de l'utilisateur.

Question 5

Quelle est la principale différence entre XSS et CSRF ?

- ☒ XSS permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur tandis que CSRF permet à un attaquant d'effectuer des actions non autorisées au nom de l'utilisateur
 - ☐ XSS permet à un attaquant d'effectuer des actions non autorisées au nom de l'utilisateur tandis que CSRF permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur
 - ☐ Il n'y a pas de différence entre XSS et CSRF, ce sont des termes interchangeables pour la même vulnérabilité de sécurité
- Q La vulnérabilité XSS permet à un attaquant d'exécuter du code malveillant sur le navigateur d'un utilisateur, tandis que la vulnérabilité CSRF permet à un attaquant d'effectuer des actions non autorisées au nom de l'utilisateur. Les deux vulnérabilités exploitent la confiance accordée entre l'utilisateur et le site web, mais de manière différente. XSS profite de la confiance que l'utilisateur accorde au site web en injectant du code malveillant dans les pages web, tandis que CSRF profite de la confiance que le site web accorde à l'utilisateur en lui permettant d'effectuer des actions en son nom sans vérification adéquate.