

1. NOTIONS DE SÉCURITÉ

1.1. QUELQUES PRINCIPES ESSENTIELS (LISTE NON-EXHAUSTIVE)

- Veiller à limiter au **strict nécessaire** les **droits et permissions** de l'utilisateur.
- **Ne jamais faire confiance** aux données transmises par l'utilisateur : un contrôle des données **côté client** est bien pratique, mais **illusoire** quant à la sécurité : on peut désactiver Javascript. **Seul un contrôle côté serveur peut prétendre être efficace.**
- **Préférer**, quand cela est possible, la **méthode POST** à la méthode GET (les variables n'apparaissent pas dans l'url)
- **Remplacer** les noms de variables par des clés (moins facile à décoder pour un utilisateur malveillant) ainsi que les valeurs lorsque cela est possible. **(non étudié ici).**
- **Tenir à jour** système serveur et interpréteur PHP avec les dernières versions stables.
- A terme, utiliser des Framework de développement qui intègrent des éléments de sécurité et accélèrent le développement (Symfony, Zend, Code Igniter ...
https://fr.wikipedia.org/wiki/Liste_de_frameworks_PHP)

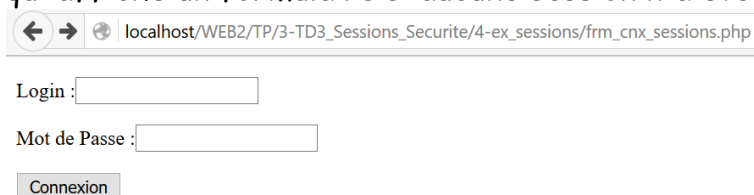
2. LES SESSIONS (35 MINUTES)

L'usage des sessions est un moyen sûr, simple, efficace et largement utilisé de "tracer" ses visiteurs (sites de commerce en ligne, forums, etc). Le principe en est simple : il s'agit d'attribuer un identifiant unique à chaque visiteur (par défaut PHPSESSID), ce qui créera un fichier dans un répertoire temporaire sur le serveur, fichier dans lequel seront stockées les variables générées par la session.

2.1. ECRIRE UNE PAGE FRM CNX SESSIONS.PHP

(prévoir deux comptes : un avec le statut de prof, l'autre avec le statut d'étudiant)

1. qui affiche un formulaire si aucune session n'a été ouverte



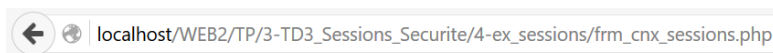
← → localhost/WEB2/TP/3-TD3_Sessions_Securite/4-ex_sessions/frm_cnx_sessions.php

Login :

Mot de Passe :

Connexion

2. qui affiche les informations suivantes si la connexion a réussi
 - un lien vers une **page page_session.php**
 - les variables de session
 - un lien pour se déconnecter



← localhost/WEB2/TP/3-TD3_Sessions_Securite/4-ex_sessions/frm_cnx_sessions.php

Accueil depuis la page initiale

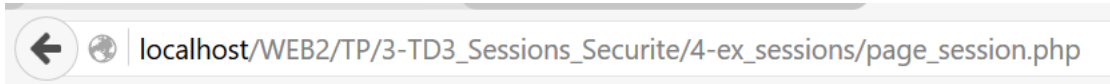
[Lien vers la section membre](#)

- login : durand
- password : philippe
- statut : Prof

[Se déconnecter](#)

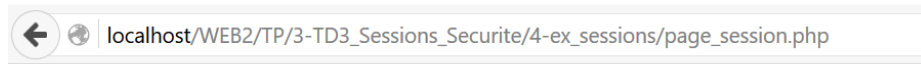
2.2. ECRIRE UNE PAGE PAGE_SESSION.PHP

- qui est protégé en accès lorsqu'il n'y a pas de session d'ouverte



[Vous êtes déconnecté, cliquer pour retourner à la page accueil](#)

- qui affiche le login et le statut de la personne connectée



Bienvenue sur la page accueil des membres

Bonjour toto. Vous êtes Etudiant

[Se déconnecter et retourner à la page accueil](#)

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Comme toutes les applications, les applications web sont sujettes à des vulnérabilités. Nous allons en voir deux d'entre elles :

- une faiblesse basée sur les cookies ;
 - Ce qui permet – par exemple – à un attaquant de contourner un mécanisme d'authentification.
- une faiblesse basée sur un code source mal développé.
 - Ce qui permet – par exemple – à un attaquant de contourner un mécanisme d'authentification, d'accéder à des données pour les divulguer ou les corrompre.

Extrait de :

https://www.ssi.gouv.fr/uploads/2016/05/cyberedu_module_3_reseau_et_applicatifs_02_2017.pdf

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Les cookies sont des fichiers gérés par les navigateurs web afin de stocker (et réutiliser) des informations concernant l'utilisateur, par exemple :

- son identifiant ;
- ses préférences d'affichage et de disposition de la page web.

Les cookies sont nécessaires pour toutes les pages web dynamiques qui nécessitent d'identifier ou d'authentifier l'utilisateur, en permettant notamment la mise en œuvre de sessions :

- les sites marchand (afin d'afficher le panier de l'utilisateur connecté) ;
- les sites bancaires (afin d'afficher le solde du compte de l'utilisateur connecté et non pas celui d'un autre client) ;
- les sites « en général » (afin d'afficher des publicités ciblées sur notre navigation).

Il est possible – sous certaines conditions – d'usurper l'identité d'un utilisateur sur un site web si on arrive à récupérer son cookie d'identification.

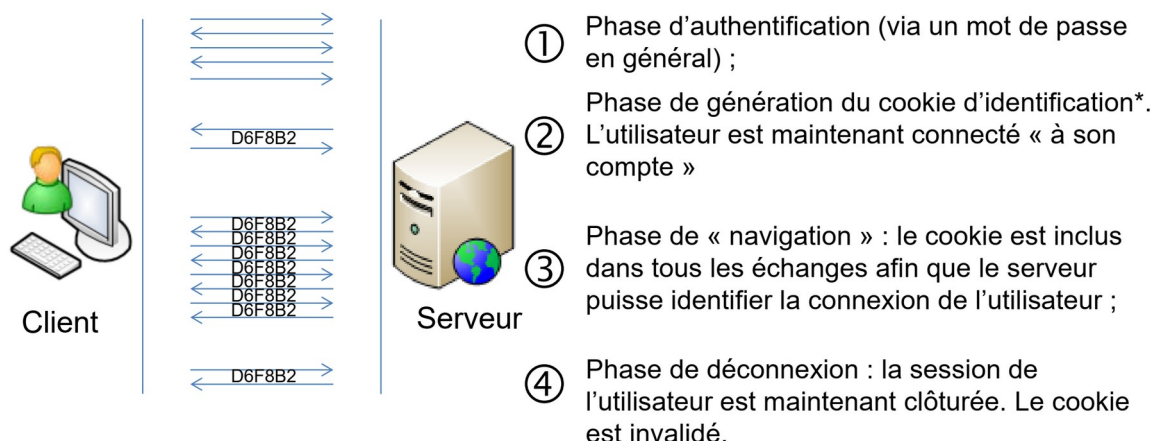
03/11/2017

Sensibilisation et initiation à la cybersécurité

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Fonctionnement habituel d'une connexion sur un site web nécessitant une authentification (site marchand, site bancaire, etc.) :



- Un cookie d'identification est en fait une chaîne de caractères aléatoire et **unique**, suffisamment longue pour qu'elle ne puisse pas être générée deux fois par erreur.
Exemple d'un cookie d'identification : D6F8B2BE3ED3040D9A3C10-D6F8B2A305D048B9

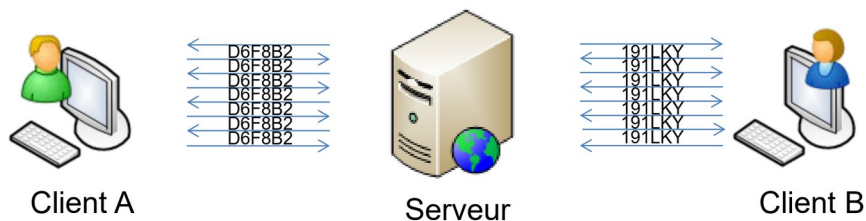
03/11/2017

Sensibilisation et initiation à la cybersécurité

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

A tout moment d'une connexion, chaque utilisateur du site web possède donc son propre cookie, unique à lui. Le serveur est donc en mesure d'identifier à qui appartient chaque connexion, et donc d'afficher les pages web qui lui sont propres.



03/11/2017

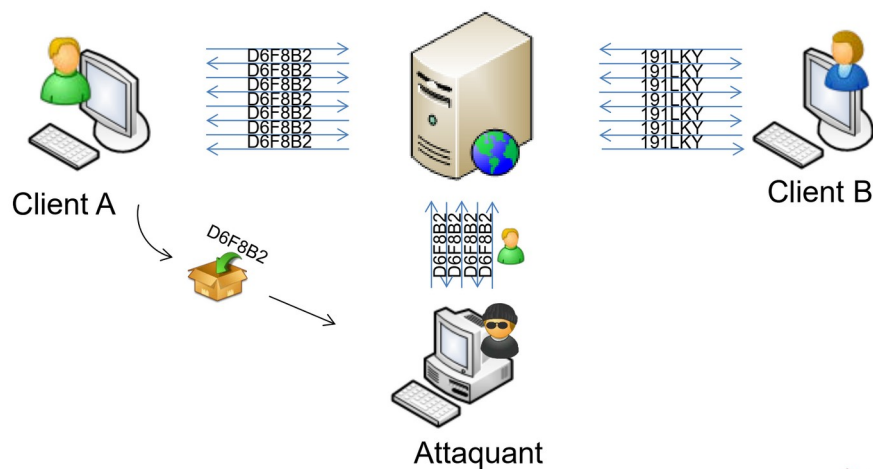
Sensibilisation et initiation à la cybersécurité

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

Mais que se passe-t-il si un attaquant arrive à dérober le cookie d'un utilisateur et se connecte au même serveur ?

- Il se fait passer pour l'utilisateur dont il a dérobé le cookie au près du serveur applicatif ! Il usurpe donc l'identité de la victime et accède à son compte.



03/11/2017

Sensibilisation et initiation à la cybersécurité

4. La sécurité des applications web

a. Usurpation d'identité via les cookies

L'attaquant peut dérober un cookie d'identification par différents moyens :



- soit en écoutant le trafic réseau HTTP et en interceptant les données applicatives, dont le cookie ;
 - Moyen de protection : l'utilisateur doit **s'assurer que le site auquel il est connecté utilise du HTTPS** (le cookie est donc chiffré pendant le transport).



- soit en dérobant le cookie sur le poste de travail en utilisant une vulnérabilité du système ;
 - Moyen de protection : l'utilisateur doit **sécuriser son système d'exploitation et ses logiciels** correctement (services inutiles désactivés, installation des mises à jours de sécurité, anti-virus, etc. voir le module 2 pour plus d'informations).



- soit en dérobant le cookie sur le poste de travail via des méthodes d'ingénierie sociale ciblées sur l'utilisateur ;
 - Moyen de protection : l'utilisateur doit **être sensibilisé aux méthodes d'ingénierie sociale** (phishing, spam, etc.) afin de « ne pas tomber dans le panneau »



- soit en dérobant le cookie via une faille sur le serveur ;
 - Moyen de protection : l'exploitant du serveur doit **suivre les bonnes pratiques de sécurisation et du maintien en condition de sécurité** du serveur, ainsi que les **bonnes pratiques de développement applicatif**.

3. PROTÉGEZ VOS REQUÊTES SQL (25 MINUTES)

4. La sécurité des applications web

b. Injection SQL

- Une attaque par injection SQL permet à un **attaquant d'interagir directement avec la base de données** d'un site web (alors que l'accès à cette base est bien entendu interdite) ;
- L'objectif de ce type d'attaque est en général de **contourner le mécanisme d'authentification, d'accéder ou de modifier frauduleusement les données** confidentielles de la base (mots de passe, téléphones, numéro de carte bancaire, etc.) ;
- Il existe de multiples variantes possibles, la diapositive suivante présente un exemple de contournement d'authentification d'une page web.

03/11/2017

Sensibilisation et initiation à la cybersécurité

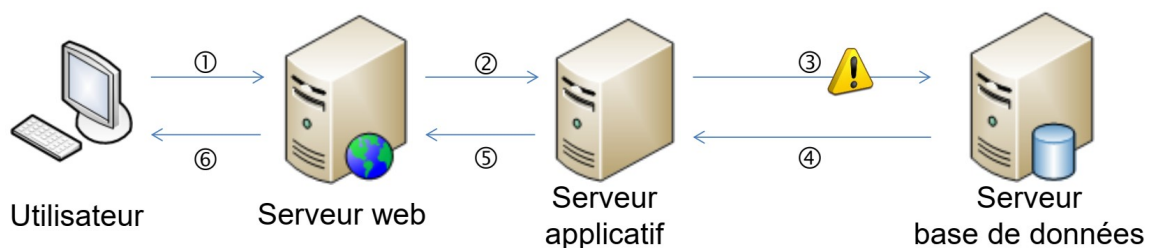


71

4. La sécurité des applications web

b. Injection SQL

Architecture standard logicielle d'un site web faisant appel à une base de données



- ① Le navigateur client demande l'affichage d'une page ;
- ② Le serveur web transfère la demande au serveur applicatif ;
- ③ Le serveur applicatif génère une requête SQL afin de récupérer les informations nécessaires ;
- ④ Le serveur base de données retourne le résultat de la requête au serveur applicatif ;
- ⑤ Le serveur applicatif transmet au serveur web les informations nécessaires à la création de la page à afficher ;
- ⑥ Le serveur web envoie les pages HTML au navigateur client.

03/11/2017

Sensibilisation et initiation à la cybersécurité



72

4. La sécurité des applications web

b. Injection SQL

- L'objectif d'une attaque de type injection SQL consiste à détourner la requête SQL de l'étape 3 (diapositive précédente), et – en fonction du contexte – créer sa propre requête SQL malveillante ;
- La diapositive suivante illustre comment une telle attaque peut être menée à partir d'un navigateur client.

03/11/2017

Sensibilisation et initiation à la cybersécurité



73

4. La sécurité des applications web

b. Injection SQL

Formulaire WEB :

Entrez votre identifiant et votre mot de passe puis cliquez sur Connexion :

Login

Mot de passe

Connexion

\$user contient le login renseigné dans le formulaire par l'utilisateur.

\$mdp contient le mot de passe.

La requête SQL permettant de vérifier le login et le mot est la suivante :

```
select count(*) from user where user='$user' and mdp='$mdp'
```

Ainsi, une requête légitime serait la suivante :

```
select count(*) from user where user='thomas' and mdp='cykUf19an'
```

03/11/2017

Sensibilisation et initiation à la cybersécurité



74

4. La sécurité des applications web

b. Injection SQL

Formulaire WEB :

Entrez votre identifiant et votre mot de passe puis cliquez sur Connexion.

Login	Mot de passe
Connexion	

Mais que se passe-t-il si un attaquant rentre précisément les chaînes de caractères suivantes ?

Login : azerty

Mot de passe : **abcd' or 1=1/***

La requête SQL `select count(*) from user where user='$user' and mdp='$mdp'`

devient donc :

```
select count(*) from user where user='azerty' and mdp='abcd' or 1=1/*'
```

Cette condition est toujours vraie !

03/11/2017

Sensibilisation et initiation à la cybersécurité



CyberEdu

75

4. La sécurité des applications web

b. Injection SQL

- La condition étant toujours vraie, la requête est donc toujours valide, quel que soit le mot de passe renseigné par l'attaquant !
 - Les caractères /* sont utilisés pour ignorer la fin de la requête légitime.
- La faiblesse réside ici dans le code applicatif : les **données** renseignées par l'utilisateur (i.e. un attaquant dans notre scénario) **ne sont pas vérifiées/validées** ; elles sont au contraire utilisées telles quelles sans aucune vérification préalable qu'elles sont « inoffensives »
- Comment s'en protéger ?
 - **Valider systématiquement chaque donnée** extérieure avant de l'utiliser ;
 - Recourir à des requêtes préparées (connues sous le nom de « **prepared statements** »), qui ont l'avantage d'être plus résistantes aux injections ;
 - D'une façon générale, **respecter les bonnes pratiques de développement** recommandées par l'industrie concernant le code PHP, Java, etc.

03/11/2017

Sensibilisation et initiation à la cybersécurité



CyberEdu

76

Extrait de :

https://www.ssi.gouv.fr/uploads/2016/05/cyberedu_module_3_reseau_et_applicatifs_02_2017.pdf

Lors de la réception des données, si vous faites des appels à votre base de données, en fonction des données envoyées, il faut se protéger des injections SQL. Le principe de ce type d'attaque est de passer des requêtes "imprévues" au SGBD, le plus souvent via les champs d'un formulaire, et grâce à l'emploi astucieux de caractères spéciaux ("'", "%", "*", "#", etc). Un exemple très connu d'injection sql repose sur l'idée de "couper", à l'aide de commentaires, une partie de la requête à l'exécution par exemple avec un login **admin'/***.

Il faut ajouter des apostrophes autour de vos champs. Avec PDO, il suffit d'utiliser la fonction **quote()** :

```
$madb = new PDO('sqlite:bdd/IUT.sqlite');  
$login = $madb->quote($param);
```

1. Créer un utilisateur admin / admin dans votre BDD
2. Reprendre le script (fourni) du TD2 EX4_BDD.php associé à son fichier de fonctions EX4_BDD_Fonctions.php et tester le avec le login **admin'/*** (on injecte un commentaire)

Tester cet exemple en utilisant **quote()** pour régler le problème.

Par la suite, il faudra systématiquement protéger les données reçues par formulaire HTML avant de les utiliser dans une requête vers un SGDB
<http://php.net/manual/fr/pdo.quote.php>

4. RÉCEPTION DES DONNÉES (TD 45 MINUTES À FINIR EN TP)

4.1. FILTRE LES FORMULAIRES CÔTÉ PHP

Le formulaire contrôlera les données saisies et indiquera les erreurs en rouge avec d'une classe CSS:

<div><h4>Validation Formulaire en PHP</h4><p>* Champs obligatoires</p><p>Nom: <input type="text"/> *</p><p>E-mail: <input type="text"/> *</p><p>SiteWeb: <input type="text"/></p><p>Commentaire: <input type="text"/></p><p>Genre: <input type="radio"/> Féminin <input type="radio"/> Masculin *</p><p><input type="submit" value="Submit"/></p><h4>Votre saisie</h4><ol style="list-style-type: none">1. Quelle page appelle ce formulaires ?2. Avec quelle méthode HTTP ?3. Combien y-a-t-il de balises input dans le formulaire ?4. Que permet l'instruction PHP suivante : <code><?php echo \$nomErr;?></code><p>Exemple d'affichage avec deux erreurs de saisie :</p><h4>Validation Formulaire en PHP</h4><p>* Champs obligatoires</p><p>Nom: <input type="text"/> *</p><p>E-mail: <input type="text"/> * Il faut saisir une adresse email</p><p>SiteWeb: <input type="text"/></p><p>Commentaire: <input type="text"/></p><p>Genre: <input type="radio"/> Féminin <input type="radio"/> Masculin * Il faut saisir le genre</p><p><input type="submit" value="Submit"/></p><h4>Votre saisie</h4><p>DURAND</p></div>	<pre><!DOCTYPE HTML> <html> <head> <meta charset="utf-8"/> <style> .error {color: #FF0000;} </style> </head> <body> <?php //variables affectées à des valeurs vides \$nomErr = \$emailErr = \$genreErr = \$sitewebErr = ""; \$nom = \$email = \$genre = \$comment = \$siteweb = ""; ?> <h2>Validation Formulaire en PHP</h2> <p>* Champs obligatoires</p> <form method="post" action="<?php echo htmlspecialchars(\$_SERVER["PHP_SELF"]);?>"> Nom: <input type="text" name="nom"> * <?php echo \$nomErr;?>

 E-mail: <input type="text" name="email"> * <?php echo \$emailErr;?>

 SiteWeb: <input type="text" name="siteweb"> <?php echo \$sitewebErr;?>

 Commentaire: <textarea name="comment" rows="5" cols="40"></textarea>

 Genre: <input type="radio" name="genre" value="femme">Féminin <input type="radio" name="genre" value="homme">Masculin * <?php echo \$genreErr;?>

 <input type="submit" name="submit" value="Submit"> </form> <?php echo "<h2>Votre saisie</h2>"; echo \$nom; echo "
"; echo \$email; echo "
"; echo \$siteweb; echo "
"; echo \$comment; echo "
"; echo \$genre; ?> </body> </html></pre>
---	--

4.1.1. MÉTHODE

Il faut commencer par tester pour chaque input si le contenu est vide, si tel est le cas il faut afficher le message d'erreur. Sinon, il faut filtrer chaque entrée avec les fonctions PHP suivantes :

1. `trim($data);` Supprime les espaces (ou d'autres caractères) en début / fin de chaîne
2. `stripslashes($data);` Supprime les antislashes d'une chaîne
3. `htmlspecialchars($data);` Convertit les caractères spéciaux en entités HTML

On peut utiliser la fonction suivante :

```
<?php function formater_saisie($data) {  
    $data = trim($data);  
    $data = stripslashes($data);  
    $data = htmlspecialchars($data);  
    return $data;  
}>>
```

On doit ensuite pousser le test en vérifiant le formatage de l'entrée en utilisant par exemple les expressions régulières :

// Test si le nom contient seulement des lettres et des espaces

```
<?php if (!preg_match("/^[a-zA-Z ]*$/", $nom)) {  
    $nomErr = "Seules les lettres et les espaces sont autorisés.";  
}>>
```

La fonction `preg_match` effectue une recherche de correspondance avec une expression régulière standard.

Pour le nom, on peut programmer cela ainsi :

```
<?php if (empty($_POST)) {  
    if (empty($_POST["nom"])) { $nomErr = "Il faut saisir un nom";  
    } else { $nom = formater_saisie($_POST["nom"]);  
        // Test si le nom contient seulement des lettres et des espaces  
        if (!preg_match("/^[a-zA-Z ]*$/", $nom)) {  
            $nomErr = "Seules les lettres et les espaces sont autorisées.";  
        }  
    }  
}>>
```

1. Compléter le script fourni pour tester le nom

4.1.2. TESTER UN MAIL

On peut écrire une expression régulière de la même manière que précédemment ou alors utiliser une fonction PHP : `filter_var($chaîne, FILTER_VALIDATE_EMAIL)` ; qui retourne les données filtrées si la chaîne respecte le filtre passé en second paramètre ou false.

<http://php.net/manual/fr/filter.filters.validate.php>

2. Compléter le script fourni pour tester l'email

4.1.3. TESTER L'URL

De la même manière, on peut utiliser une des deux méthodes précédentes.

Exemple d'expression régulière :

```
"/\b(?:(:?https?|ftp):\/\/|www\.|)[-a-z0-9+&@#\/%?~_!|:,;]*[-a-z0-9+&@#\/%?~_!|]/i"
```

ou

filtre : `FILTER_VALIDATE_URL`

3. Compléter le script fourni pour tester l'url

4.1.4. TESTER LE COMMENTAIRE

4. Compléter le script fourni pour tester le commentaire

4.1.5. TESTER LE GENRE

5. Compléter le script fourni pour tester le genre

(1) Faire valider par l'enseignant

5. FAIBLE XSS (TP)

Il faut restreindre au strict minimum la saisie possible pour chaque champ afin de ne pas polluer les données destinées à un script qui accédera à une BDD avec des caractères non prévus. (Exemple : pas de guillemets dans un mot de passe...). **Le contrôle doit aussi se faire côté serveur.**

Par principe, il ne faut pas faire confiance aux données transmises par un utilisateur. La fonction `htmlspecialchars()` permet d'éviter que vos données contiennent des bouts de codes html, ou autre caractères pouvant poser problèmes tels que `<>&"'`. Sans cette fonction, il est possible de placer du code malicieux, des effets javascript, polluer votre site, effectuer des requêtes sql non voulues...

Soit la page <code>xss.php</code> <code><?php \$var=\$_GET["var"]; echo "Variable : ".\$var; ?></code>	Appeler la page « normalement » : <code>http:// @IPSERVEUR/ xss.php?var=essai</code>	Appeler la page en introduisant un code javascript <code>xss.php?var=J'ai hacké ton site!!!<script type='text/javascript'>window.location = 'http://www.iut-lannion.fr/'</script></code>
---	--	---

(2) Faire valider par l'enseignant en montrant le problème et la solution

6. CRÉER UN FICHIER DE LOG (TP)

Vous n'avez pas forcément accès au fichier de log du serveur web hébergeant votre application, vous pouvez enregistrer dans un fichier ou une base de données différents logs (user, url demandée, IP source, date ...)

<http://php.net/manual/fr/reserved.variables.server.php>

<http://php.net/manual/fr/function.fopen.php>

<http://php.net/manual/fr/function.fputs.php>

```
<?php // 1 : on ouvre le fichier
    $monfichier = fopen('acces.log', 'a+');
    // 2 : on fera ici nos opérations sur le fichier...
    fputs($monfichier, $_POST['login']." de ".$_SERVER['REMOTE_ADDR']." à ".date('l
jS \of F Y h:i:s A'));
    fputs($monfichier, "\n");
    // 3 : quand on a fini de l'utiliser, on ferme le fichier
    fclose($monfichier);?>
```

(3) Reprendre l'exercice 1 (Sessions) et ajoutez y un fichier de log pour toutes les tentatives de connexions

7. ANTI-FLOOD AVEC CAPCHA (TP)

Si vous avez choisi de recevoir un email lorsqu'un utilisateur envoie un formulaire de votre application, pensez alors à installer un système qui évitera la répétition d'envoi de données provenant de la même IP par exemple, ou vous risqueriez d'être inondé (flood) de mails. (CAPCHA par exemple)

Il existe plusieurs solutions pour lutter contre ces robots spammeurs, l'une d'entre elle consiste à soumettre l'utilisateur à un test auquel seul un être humain est sensé être capable de répondre (on appelle ça un test de Turing).

En pratique, ça se fait tout simplement en ajoutant un champ antispam appelé Captcha dans un formulaire HTML qu'on souhaite protéger contre les robots.

Il existe plusieurs types de Captcha et pour mettre en place un système de captcha sur votre site avec PHP, il y a 2 solutions :

- Gérer soi-même le captcha de A à Z
- Utiliser un service de captcha comme reCaptcha

7.1. CAPCHA AVEC TEXTE DÉFORMÉ

Le fonctionnement est très simple : on présente à l'utilisateur une image qui contient un texte déformé ou brouillé, et pour passer le test avec succès il faut que l'utilisateur saisisse le texte qu'il voit dans un champ texte.

Ce type de captcha est le plus répandu, cependant il commence à montrer ses limites car les robots sont de plus en plus élaborés et arrivent à lire ces images tout comme le font les humains. Du coup on est obligé d'utiliser des captchas de plus en plus déformés, si bien qu'ils en deviennent difficiles à déchiffrer pour les êtres humains !



Ex: reCaptcha

reCaptcha est un service gratuit proposé par Google pour intégrer facilement un captcha sur un site.

7.2. CAPCHA AVEC QUESTION / RÉPONSE

Le principe est très similaire au texte déformé : on pose une question très simple à l'utilisateur, du style "Combien font deux plus vingt ?", et il faut écrire le résultat (la réponse) dans un champ texte.

Les captcha Q/R sont de plus en plus utilisés car ils sont beaucoup plus efficaces pour stopper les robots, beaucoup moins frustrants pour l'utilisateur car ils ne nécessitent pas d'effort, contrairement à la lecture d'une image déformée, et aussi très facile à mettre en place techniquement.

7.3. CAPCHA AVEC ACTIONS SIMPLES À EFFECTUER

Ex : clic sur des images : chat, voiture, moto...

Encore assez rare pour le moment, on commence à voir certains sites qui utilisent des sortes de tests de logique très simple.

Par exemple, on présente une série de 9 photos à l'utilisateur (divers objets : une voiture, un chat, une moto, le ciel, une ville...), et on demande à l'utilisateur de cliquer sur un animal.

Autre exemple, encore plus simple : on crée un bouton on/off, par défaut positionné sur Off, et on demande à l'utilisateur de le glisser sur On.

7.4. EXEMPLE D'UN CAPCHA AVEC UNE IMAGE


7.4.1. PAGE CONTACT.PHP

```
<?php session_start();  
<!DOCTYPE html>  
<html lang="fr" >  
<head>  
<meta charset="utf-8">  
<title>Formulaire de contact avec Capcha</title>  
</head>  
<body>  
<h1>Formulaire de contact avec Capcha</h1>  
<form action="contact.php" method="post">  
  <input type="text" name="captcha"/>  
  <input type="submit"/>
```

Formulaire de contact avec Capcha

Envoyer 

Formulaire de contact avec Capcha

Envoyer 
Code correct

Formulaire de contact avec Capcha

Envoyer 
Code incorrect

```
    
</form>  
<?php if(isset($_POST['captcha']))){  
  if($_POST['captcha']==$_SESSION['code']){  
    echo "Code correct";  
    //ici vous traitez les autres éléments du formulaire  
  
  } else {  
    echo "Code incorrect"; //ici vous faites un "echo" pour avertir qu'il y a une erreur  
  }  
}  
>>  
</body>
```

7.4.2. PAGE IMAGE.PHP (POUR INFO)

Le formulaire charge une image générée par ce script **image.php** avec la balise **** suivante :

```

```

Math.random() nous permet de demander une nouvelle image car elle fait en sorte de recharger l'image par une autre en disant que c'est pas la même, on rajoute juste du code du style **image.php?0.9610655198268976**

```
<?php  
session_start();//on enregistre la session, pour le code, pour la vérification du formulaire  
//le fichier se nomme: image.php  
//on indique au header qu'il faut afficher le code en tant qu'image  
header('Content-Type: image/png');  
$largeur=80;//largeur de l'image  
$hauteur=25;//hauteur de l'image  
$lignes=10;//nombre de lignes multicolore qui seront affichées avec le code (10 est bien)  
$caracteres="ABCDEF123456789";//type de caractère du code qui sera affiché dans l'image  
//on crée le rectangle  
$image = imagecreatetruecolor($largeur, $hauteur);
```

```

//on met un fond en blanc (255,255,255)
imagefilledrectangle($image, 0, 0, $largeur, $hauteur, imagecolorallocate($image, 255, 255, 255));
//on ajoute les lignes
function hexargb($hex) { //fonction qui permet de retourner la valeur en RGB d'une couleur
hexadécimale
    return array("r"=>hexdec(substr($hex,0,2)), "g"=>hexdec(substr($hex,2,2)), "b"=>hexdec(substr($hex,4,2)));
}
//on retourne la valeur sous forme d'array
}
for($i=0;$i<=$lignes;$i++){
    $rgb=hexargb(substr(str_shuffle("ABCDEF0123456789"),0,6)); //choisi une couleur aléatoirement
    (str_shuffle), de 6 caractères (substr(chaine,0,6)) avec la sélection alphanumérique
    imageline($image, rand(1, $largeur-25), rand(1, $hauteur), rand(1, $largeur+25), rand(1, $hauteur),
    imagecolorallocate($image, $rgb['r'], $rgb['g'], $rgb['b']));
}
$code1=substr(str_shuffle($caracteres),0,4);
$_SESSION['code']=$code1; //on enregistre le code dans une session pour vérifier ensuite si qu'à
entré le visiteur est identique
$code=""; //on initialise le code
for($i=0;$i<=strlen($code1);$i++){
    $code .=substr($code1,$i,1)." "; //on rajoute des espace entre chaque lettre ou chiffre pour faire
plus aéré (notez le . devant = qui permettra d'ajouter un caractère après l'autre à $code)
}
//on écrit le code dans le rectangle
imagestring($image, 5, 10, 5, $code, imagecolorallocate($image, 0, 0, 0));
//on affiche l'image
imagepng($image);
//puis on détruit l'image pour libérer de l'espace
imagedestroy($image);
?>

```

7.4.3. TRAVAIL À RÉALISER

(4) Mettre en place un captcha sur le formulaire de l'exercice 4. Il ne doit y avoir qu'un seul bouton sur le formulaire !!!

(5) Finir le TP 2 si vous n'avez pas eu le temps de le terminer