

Sites web et bases de données

Sylvain Tenier, Romain Vallee, Vincent Derrien

Département TIC - Esigelec

Semestre 7 - 2016

Objectifs de la séance

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Principe de l'authentification HTTP

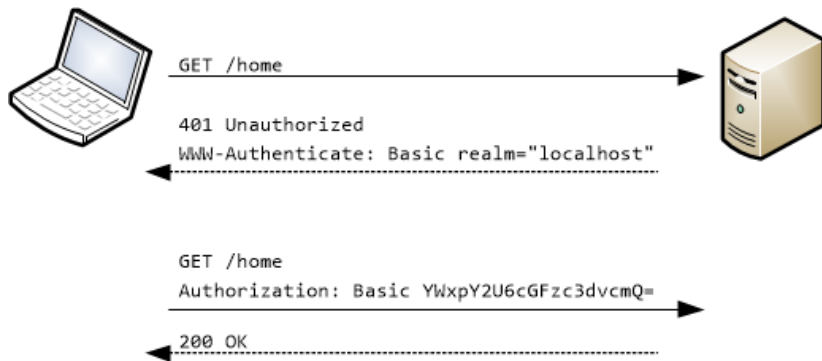


FIGURE : Transactions HTTP d'authentification

Mise en place de l'authentification HTTP

Objectifs

- simplicité de mise en place
- sécurisation de *groupes de pages* (le "realm")
- support large (tous les navigateurs compatibles HTTP 1.1)

Configuration

- depuis le serveur web (Apache, Nginx, IIS, ...)
- depuis le langage serveur pour les sites dynamiques

Exemple de configuration par serveur

Configuration du serveur web *Apache*

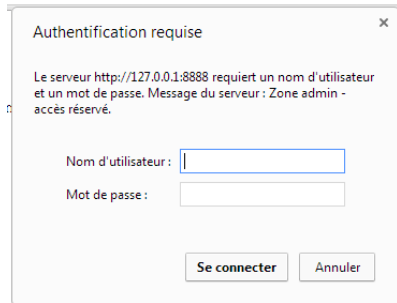
Configuration

- Fichier `.htaccess` à la racine des pages

```
1 AuthUserFile .htpasswd
2 AuthName "Zone admin - accès
  réservé"
3 AuthType Basic require
  valid-user
4
```

- Fichier `.htpasswd` dans le dossier *apache*

```
1 admin:admin
2
```



Authentification requise

Le serveur `http://127.0.0.1:8888` requiert un nom d'utilisateur et un mot de passe. Message du serveur : Zone admin - accès réservé.

Nom d'utilisateur :

Mot de passe :

FIGURE : Demande d'authentification

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Reconnaître et suivre un utilisateur

Problématique

- HTTP est un protocole *déconnecté*
- Chaque requête est *indépendante*
- Le serveur ne sait pas avec qui il échange

Personnalisation des URL

- Chaque ressource est accessible par son URL
- L'adresse peut être suivie d'une *chaîne de requête*
- Exemple : `https://www.google.fr/search?q=chaine+de+requete`

Principe des URL étendues

- Ajout d'une chaîne personnalisée dans l'URL
 - Exemple : `www.example.com/?refid=f4d5t64kjhfk575`
- Génération par PHP au premier accès de l'utilisateur
- Ajout dynamique dans chaque lien du site
 - Exemple :
`la page 2`
- Récupération dans chaque page par `$_GET['refid']`

Sécurité des URL étendues

- L'utilisateur peut modifier l'URL
- La chaîne doit être suffisamment longue et aléatoire
- Prévoir l'expiration des chaînes

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Cookies de sessions PHP

- Cookie : fichier texte stocké par le navigateur
 - Valeurs transmises dans la requête HTTP avec l'en-tête *Cookie*
- Utilisation transparente en PHP
 - `<?php session_start()?>` au début de chaque page
 - active le tableau associatif `$_SESSION`
 - `$_SESSION` conserve son contenu de page en page
- <http://www.php.net/manual/fr/intro.session.php>

Cas d'utilisation des sessions PHP

Persistance entre pages

- Chaque nouvel arrivant obtient son propre cookie
- Toute information peut être stockée dans le tableau de session
- Exemple :
\$_SESSION['msg'] permet de transmettre un message personnalisé d'une page à l'autre

```
1 | <p><?php echo '  
   Attention : ' . $  
   _SESSION['msg'] ?></p>
```

Authentification

- Un formulaire est présenté demandant un id et un mot de passe
- Si les éléments sont corrects, l'id est enregistré dans le tableau de session
- Exemple :
\$_SESSION['id']=\$id, où \$id est issu de \$_POST['id'] (après validation et filtrage)

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Choix d'une extension PHP/MySQL

extension mysql

- Obsolète (depuis PHP 5.5)
- Interface procédurale
- *Non disponible* sur le serveur de production

extension mysqli

- Évolution de l'extension mysql
- Interface procédurale ET objet
- API spécifique à MySQL

extension PDO

- Interface objet uniquement
- Reprend le principe de JDBC pour JAVA
- API commune à tous les SGBD

<http://php.net/manual/fr/mysqliinfo.api.choosing.php>

Comparaison mysqli / PDO

mysqli

```
1 $mysqli = new mysqli("example.com", "user", "password", "madb");
2 $res = $mysqli->query("SELECT attribut FROM matable");
3 $row = $res->fetch_assoc(); //première ligne du résultat
4 echo htmlentities($row['attribut']); //affichage d'un attribut
5
```

PDO

```
1 $pdo= new PDO("mysql:host=example.com;dbname=madb","user","password");
2 $res= $pdo->query("SELECT attribut FROM matable");
3 $row = $res->fetch(PDO::FETCH_ASSOC); //première ligne du résultat
4 echo htmlentities($row['attribut']); //affichage d'un attribut
5
```

Exemple de configuration

configuration du serveur de développement EasyPHP

- Création de 4 variables PHP dans un fichier d'inclusion (voir séance 3)
- Récupération des variables dans chaque script utilisant la BDD
`<?php require_once("param.inc.php"); ?>`
- Le déploiement sur le serveur de production sera facilité
 - Attention, la valeur des variables devra être modifiée sur le serveur de production !

param.inc.php

```
1 $host='127.0.0.1';  
2 $login='root';  
3 $password='';  
4 $dbname='madb';
```

Exemple de page générée depuis une base de données

Affichage généré depuis la table *clients* d'une base *commercial*

```
1 <!DOCTYPE html>
2 <html><head><title>Clients à spammer</title>
3 <meta charset="utf-8"></head>
4 <body><h1>Liste des clients à spammer</h1>
5 <?php
6     require_once('param.inc.php');
7     $mysqli = new mysqli($host,$login,$password,$dbname);
8     if ($mysqli->connect_errno) {
9         echo "Echec lors de la connexion à MySQL : (" . $mysqli->
10             connect_errno . ") " . $mysqli->connect_error;
11     }else{
12         $res=$mysqli->query("SELECT * FROM clients");
13         if(!$res){
14             die('Echec de la requête SQL : '.$mysqli->error);
15         }elseif($res->num_rows == 0){
16             echo '<p>Aucun resultat :</p>';
17         }else{
18             while($tuple=$res->fetch_assoc()){
19                 echo '<p>'.htmlentities($tuple['nom']).'</p>';
20             }
21         }
22     }
23 <?></body></html>
```

Points clés de la récupération des données

① Connexion à la base de données

- `$mysqli = new mysqli($host,$login,$password,$dbname);`
- `frPeut` échouer : toujours tester si `$mysqli` vaut **FALSE** Can fail : always check whether `$mysqli` equals **FALSE**

② Envoi de la requête

- `$res=$mysqli->query("SELECT * FROM clients");`
- `$res` contient le résultat de la requête ou **FALSE**
 - En cas d'échec, afficher `$mysqli->error` pour identifier le problème

③ Récupération du résultat de la requête dans un tableau

- Il faut appeler `$res->fetch_assoc()` autant de fois qu'il y a de ligne retournée par la requête SQL
- `fetch_assoc()` retourne un tableau associatif dont les clés sont chaque attribut demandé dans le **SELECT**

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Règles de sécurité

- ❶ Toujours valider tous les éléments saisis par l'utilisateur
 - (re)voir séance précédente
- ❷ Échapper les caractères avec
 - `mysqli_real_escape_string()`
 - `PDO::quote()`
- ❸ Privilégier les requêtes préparées
- ❹ Réduire les privilèges de la connexion SQL
 - Exemple : le serveur de production ne vous donne pas accès aux commandes de création/suppression de bases

Exemple d'attaque

Attaque par injection SQL

- Insertion par l'utilisateur de code SQL dans un champ de formulaire
- Objectifs
 - ① Accéder à des données
 - ② Modifier des données
 - ③ Se connecter sans mot de passe
- Exemple de code vulnérable

```
1 $input=$_POST['nom']; //aucun filtrage ni validation!!  
2 $password=$_POST['password'];  
3 $res=$mysqli->query("SELECT userid FROM users WHERE username='  
    $input' AND password='$password'");
```

- L'injection est réalisée en saisissant 'OR 1=1 --' dans le champ de formulaire ayant l'attribut `name="nom"`

Protection par utilisation des requêtes préparées

2 étapes : préparation et exécution

Préparation d'une requête

```
1 if (!($stmt = $mysqli->prepare("SELECT userid FROM users WHERE username
   =? AND password=?"))) {
2     echo "Echec de la préparation : (" . $mysqli->errno . ") " . $mysqli->
       error;
3 }
4 $stmt->bind_param('ss', $input, $password);
```

Exécution de la requête

```
1 //affectation
2 $input=$_POST['nom'];
3 $password=$_POST['password'];
4 //execution
5 if (!$stmt->execute()) {
6     echo "Echec lors de l'exécution de la requête : (" . $stmt->
       errno . ") " . $stmt->error;
7 }
```

Bilan sécurité

- Tout élément saisi par l'utilisateur :
 - ① doit être sécurisé contre les injections SQL avant d'interagir avec la BDD,
 - ② doit être sécurisé contre les attaques XSS avant d'être affiché.
- Pour sécuriser la BDD :
 - valider/filtrer les saisies en PHP,
 - utiliser les fonctions d'échappement des caractères dangereux,
 - privilégier les requêtes préparées.
- Pour sécuriser l'affichage :
 - utiliser une fonction d'échappement des caractères spéciaux : `htmlspecialchars` / `htmlspecialchars`,
 - y compris si une requête préparée a été utilisée.

Plan

- 1 Identifier et authentifier ses utilisateurs
 - Authentification par HTTP
 - URL étendues (Fat URL's)
 - Sessions PHP
- 2 Interagir avec une base de données
 - Connexion à une base de données MySQL
 - Se protéger des injections SQL
- 3 Application à la connexion d'un utilisateur

Principes

- ❶ Afficher un formulaire de connexion *si l'utilisateur n'est pas déjà connecté*
 - Si l'utilisateur est déjà connecté, afficher son nom
- ❷ Comparer le *Hash* du mot de passe reçu avec celui stocké en base de données
- ❸ Définir une variable dans le tableau de session si les identifiants correspondent ou pour indiquer l'échec
- ❹ Une fois l'utilisateur connecté, le rediriger vers la page souhaitée
 - Si les identifiants ne sont pas corrects, renvoyer l'utilisateur au formulaire en affichant un message d'erreur

Documentation

- ❶ Déterminer si une variable est définie
 - secure.php.net/manual/en/function.isset.php
 - Exemple : `if(isset($_SESSION['nom']))...`
- ❷ Stocker et vérifier le hash d'un mot de passe dans la BDD
 - <https://secure.php.net/manual/fr/function.password-hash.php>
 - <https://secure.php.net/manual/fr/function.password-verify.php>
- ❸ Ecrire l'en-tête HTTP pour rediriger l'utilisateur
 - <https://secure.php.net/manual/en/function.header.php>
 - Exemple : `header('Location: ' . $_SERVER['HTTP_REFERER']);`

Application 1 : Connexion et déconnexion

- ❶ Mettre en place la table nécessaire dans la BDD
 - Pensez à remplir la table avec un ou deux utilisateurs. Utilisez la fonction `password_hash` pour sécuriser le mot de passe.
- ❷ Écrire le script de connexion en suivant l'algorithme proposé
 - Ce script effectue un *traitement* et *redirige vers une autre page*
 - Il ne contient donc pas de HTML
- ❸ Implémenter la déconnexion

Références

Bonnes pratiques en SQL SQL Antipatterns, Bill Karwin,
Pragmatic programmers, 2010, ISBN-10 :
1934356557

Manuel PHP <http://php.net/manual/fr/index.php>