Gérer l’authentification dans une application Web

Copyright © 2020-2021 Pier-Luc Brault.
Cette présentation est mise à disposition selon les termes de la
Licence Creative Commons Paternité - Partage des Conditions Initiales à l'Identique 4.0 International.

Contenu

- **Question d’ouverture**
- **Bonnes pratiques**
- **Gestion des droits d’accès**
- **Gérer l’authentification avec PHP**
  - Variables de session
- **Pour aller plus loin**

Question d’ouverture

Question d’ouverture

Que doit-on implanter dans une application Web pour avoir une gestion complète de l’authentification?

Image par Everyday basics sur Unsplash

Question d’ouverture

- **Un endroit où conserver les identifiants des utilisateurs**
  - Ex: table « users »
- **Un formulaire de login**
- **Un bouton « Se déconnecter »**
- **Du code côté serveur pour répondre au formulaire de login et au bouton de déconnexion**
- **Une gestion de l’état (authentifié ou non) de l’utilisateur**
- **Un contrôle d’accès (contenu privé)**
- **Idéalement, une façon de gérer l’oubli d’identifiants**
  - Procédure « J’ai oublié mon mot de passe »

Bonnes pratiques

Stockage des mots de passe

- **Les mots de passe en BD devraient être hachés**
  - Ex: Hash de « monMotDePasse » avec l'algorithme de hachage SHA1: 797350ADCB79E42B6795C62FC5FEFABA80E7D418
  - Pourquoi ne pas conserver les mots de passe en clair?
  - Différence entre hachage et encryption?
- **Lors de l'authentification:**
  - On produit un *hash* à partir du mot de passe saisi par l'utilisateur
  - On compare ce *hash* avec celui en BD

Stockage des mots de passe

- **L'algorithme de hachage utilisé doit être sécuritaire**
  - Ex: Bcrypt
  - ***PAS*** MD5 ou SHA

Stockage des mots de passe

- **Bcrypt est moins vulnérable aux attaques par force brute**
  - L'algorithme de hachage est conçu pour être lent
  - Les attaques par force brute sont donc coûteuses en temps de calcul
  - Un paramètre appelé « key factor » permet de rendre l'algorithme encore plus lent pour suivre l'évolution de la puissance des ordinateurs

Stockage des mots de passe

- **Bcrypt n'est pas vulnérable aux attaques par *rainbow table***
  - *Rainbow table* = liste de hashs avec les mots de passe correspondants
  - Bcrypt ajoute une valeur aléatoire (appelée *salt*) à chaque mot de passe avant de le hacher
    - Le *salt* est stocké avec le hash pour pouvoir vérifier le mot de passe saisi par l'utilisateur lors de l'authentification

Format d’un hash Bcrypt

- **Exemple de hash Bcrypt:**
  - **$2y$12$LRRQ78HlcaXCBfAbaDNPruyeEMbp1jXPHCSYEsk6qQ7gbt6C/jok2**
- **Format:**
  - **Norme Bcrypt utilisée**
  - **Key factor**
  - **Salt (22 caractères)**
  - **Mot de passe haché**

Utilisation de HTTPS

- **En HTTP, les données sont transmises en clair sur le réseau**
  - Incluant les données de formulaire (ex: nom d'utilisateur et mot de passe)
- **Il est donc impératif qu'une application Web soit en HTTPS pour sécuriser l'authentification!**

Utilisation de HTTPS

- **HTTPS nous donne deux garanties:**
  - Le contenu des pages que nous consultons n'a pas été altéré
  - Les données échangées, si elles sont interceptées, ne peuvent pas être lues (car elles sont encryptées)

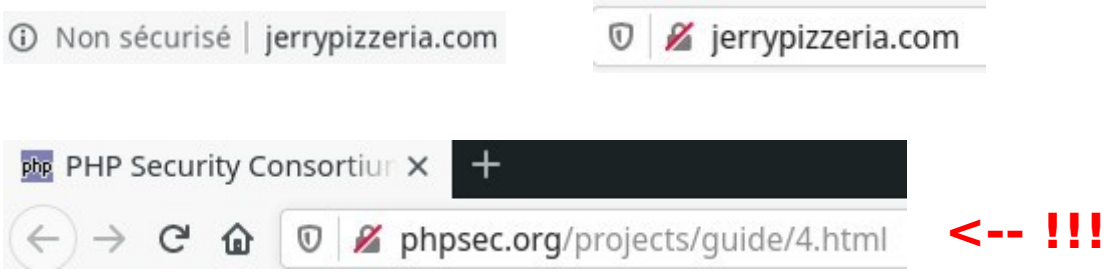
Utilisation de HTTPS

- **Pour activer HTTPS sur un serveur Web, il faut y installer un certificat SSL**
  - Délivré par une autorité de certification
  - Prouve qu'on est bien qui on prétend être (un peu comme un passeport)
- **Autorité de certification gratuite: Let's Encrypt (https://letsencrypt.org)**
  - Intégré directement dans plusieurs services d'hébergement
  - Les certificats expirent après 3 mois
    - Objectif: Pousser l'industrie à automatiser le processus de renouvellement de certificats (qui était manuel auparavant)

Let's Encrypt

Utilisation de HTTPS

- **D’autres bonnes raisons d’activer HTTPS sur son site:**
  - Meilleur référencement
  - Éviter que le navigateur affiche quelque chose comme ça:



  - C’est désormais plus facile et gratuit, alors pourquoi s’en passer?

Détecter les tentatives d’attaque par force brute

- **Enregistrer les tentatives de connexion échouées**
  - Ne PAS conserver les mots de passe entrés
- **Bloquer le compte si un certain nombre de tentatives de connexion ont été détectées dans un court laps de temps**

Gestion des droits d’accès

Gestion des droits d’accès

- **S’assurer que les utilisateurs ne peuvent accéder qu’aux données et fonctionnalités auxquelles ils ont droit**

Gestion des droits d’accès

- **Sous sa forme la plus simple:**
  - Obliger l’utilisateur à être authentifié pour accéder à certaines sections de l’application
- **Ne JAMAIS transmettre de données sensibles au client si l’utilisateur n’y a pas accès**
  - Même si le client ne les affiche pas

Gestion des droits d’accès

- **Même authentifiés, les utilisateurs ne sont pas tous égaux**
- **Certains peuvent avoir accès à des données ou fonctionnalités auxquelles d’autres n’ont pas accès**

Permissions

- **Une façon simple de gérer les droits d'accès est de définir des permissions dans le code de notre application**
  - Exemples de permissions (application de type blogue):
    - WRITE_POSTS
    - EDIT_POSTS
    - DELETE_POSTS
    - DELETE_COMMENTS

Attribution des permissions

- **Plusieurs approches existent pour attribuer des permissions à des utilisateurs**
  - En voici deux:
    - Role-Based Access Control (RBAC)
    - Access Control List (ACL)

Role-Based Access Control

- **On définit un petit nombre de rôles, ex:**
  - Writer
  - Moderator
  - Administrator
- **Chaque rôle est associé à un certain nombre de permissions**
- **Le rôle d’un utilisateur détermine donc les permissions qui lui sont attribués**

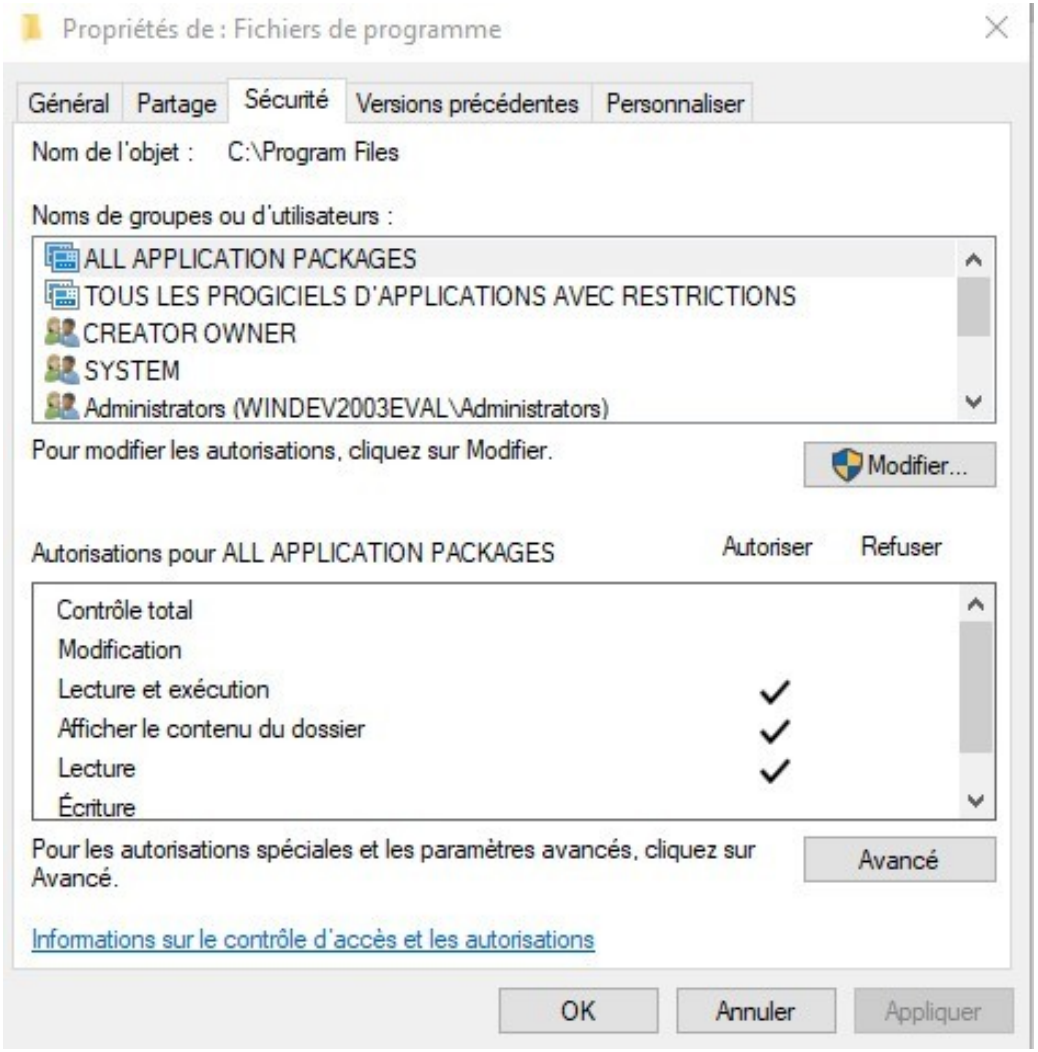
Role-Based Access Control – Exemple

Rôles / Permissions	WRITE_POSTS	EDIT_POSTS	DELETE_POSTS	DELETE_COMMENTS
WRITER	✓			
MODERATOR	✓			✓
ADMINISTRATOR	✓	✓	✓	✓

Access Control List

- **On attribue des permissions à des utilisateurs (ou des groupes d'utilisateurs) directement**
- **Il y a typiquement une liste d'accès par ressource**

Access Control List – Exemple



Gérer l’authentification en PHP

Hachage des mots de passe

- **Utiliser la fonction password_hash pour hacher un mot de passe avant de le stocker en BD**
  - Prend deux paramètres obligatoires:

    1) Le mot de passe à hacher

    2) Une constante indiquant l'algorithme de hachage à utiliser
      - Utiliser la constante PASSWORD_DEFAULT ou PASSWORD_BCRYPT pour utiliser Bcrypt
- **Exemple:**
  - `$hash = password_hash($password, PASSWORD_DEFAULT);`

Vérification des mots de passe

- **La fonction password_verify permet de vérifier si le mot de passe saisi par l'utilisateur correspond au hash conservé en BD**
  - Prend automatiquement en compte le *salt* et le *key factor* du hash (Bcrypt)
  - Retourne un booléen indiquant le résultat de la vérification
- **Exemple:**
  - `$isCorrect = password_verify($password, $hash)`

Variables de session

- **On veut éviter à l'utilisateur d'avoir à se ré-authentifier chaque fois qu'il change de page sur notre site**
- **Les variables de session permettent de conserver, côté serveur, des informations propres à l'utilisateur qui seront persistées d'une page à l'autre**
- **La session expire à la fermeture du navigateur**

Variables de session

- **Pour démarrer une session:**
  - `session_start();`
  - Doit être appelé au tout début de la page Web (avant la balise « body »)
- **Pour créer une variable de session:**
  - `$_SESSION["username"] = "Bob";`

Processus d'authentification en PHP

- **Passage des informations saisies dans le formulaire de login en POST (et non en GET)**
- **Récupérer depuis la BD le mot de passe haché correspondant au nom d'utilisateur saisi**
  - Utiliser une requête préparée (*prepared request*)
    - Pourquoi?
- **Valider le mot de passe saisi avec password_verify**
- **Si le mot de passe est bon, utiliser les variables de session pour indiquer qu'il est authentifié**
  - Ex: $_SESSION["isAuthenticated"], $_SESSION["username"]

Processus de déconnexion en PHP

- **Pour déconnecter un utilisateur, on détruit sa session, tout simplement**
  - session_unset(); // supprime toutes les variables de session
  - session_destroy(); // détruit la session

Sécurité des sessions

- **Quelques précautions doivent être prises dans la configuration du serveur PHP pour une utilisation sécuritaire des sessions**
  - Attaque la plus fréquente: vol de « session ID »
    - Le « session ID » est une valeur aléatoire générée par le serveur et retransmise à chaque requête par le navigateur pour que le serveur l'associe à la bonne session
- **Cette page décrit la configuration recommandée:**
  - https://www.php.net/manual/en/session.security.ini.php
- **Faits saillants**
  - Forcer le « session ID » à être passé au serveur via un cookie et non un paramètre dans l'URL
  - Rendre ce cookie inaccessible à JavaScript
  - Empêcher le navigateur de transmettre ce cookie autrement que via HTTPS

Rester connecté

- **Pour implanter une option « Se souvenir de moi »:**
  - Générer un « jeton » (*token*), c'est-à-dire une chaîne de caractères aléatoire associée à l'utilisateur
    - Doit être « cryptographiquement sûre » (ex: utiliser la fonction random_bytes)
  - Conserver cette valeur dans une table de tokens en l'associant à l'utilisateur
  - Transmettre le token au client
  - Le client conserve le token localement (ex: dans un cookie) et le repasse au serveur pour se ré-authentifier
    - Comme pour les cookies de session, attention à bien sécuriser le cookie de token

Pour aller plus loin

Authentification à deux facteurs

- **Qu’est-ce que l’authentification à deux facteurs, et pourquoi est-ce une bonne idée de l’offrir?**

Single sign-on

- **Autre approche d'authentification: utiliser un compte d'un service externe pour s'authentifier**
  - ex: Facebook, Google, GitHub, etc.
  - On laisse donc la gestion de la sécurité à quelqu'un d'autre, en plus d'éviter à l'utilisateur d'avoir à créer encore un autre mot de passe!

Fin de la présentation

Des questions?

Photo par Emily Morter sur Unsplash