Architecture d'un site web : authentification

Alexandre Niveau

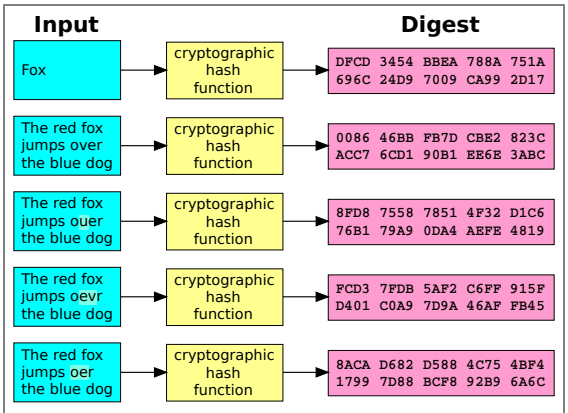
GREYC — Université de Caen

Gestion des comptes utilisateur

- Dans l'ensemble, les comptes des utilisateurs sont gérés de la même façon que les autres données
- Il y a cependant des différences cruciales, puisque ces objets sont utilisés pour l'*authentification* :
  - Stockage des mots de passe et vérification
  - Gestion de la connexion et de la déconnexion
  - Restriction des accès et des actions possibles

Stockage des mots de passe

- On ne stocke **jamais** les mots de passe en clair dans une BD : si notre serveur est compromis, l'attaquant récupère tout !
- Pas à prendre à la légère : des fuites de ce genre arrivent *tout le temps*, même à des grandes compagnies (infographie des plus grosses fuites des dix dernières années [http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/])
- On ne stocke que l'*empreinte* (*hash*) des mots de passe :
  - on leur applique une fonction mathématique non inversible avant de les stocker
  - la base ne contient que des empreintes : comme la fonction n'est pas inversible, avoir accès à la base ne permet pas de retrouver les mots de passe
  - pour vérifier qu'un mot de passe fourni est le bon, il suffit de lui appliquer la même fonction mathématique et de comparer le résultat à ce qui est stocké en BD



[Cryptographic_Hash_Function.svg
Illustration d'une fonction de hachage cryptographique
Wikimedia Commons (CC0) [https://commons.wikimedia.org/wiki/File:Cryptographic_Hash_Function.svg]

Choix du *hash*

- Il ne faut pas utiliser n'importe quelle fonction de *hash* : typiquement `md5`, `sha1` et `sha256` **ne sont pas adaptés** (en particulier `md5`, qui a été cassé il y a plus de 10 ans)
- Ces fonctions sont conçues pour être rapides à exécuter, or on veut une fonction lente (pour gêner l'attaquant au maximum)
- Un bien meilleur choix est l'algorithme `bcrypt`, qui est conçu exactement pour ce genre d'application. (Par exemple, le coût de calcul d'un empreinte est configurable)

Sel

- Stocker des empreintes ne suffit pas !
- Attaque classique dite des *rainbow tables* : l'attaquant peut utiliser un dictionnaire des empreintes de tous les mots de passe de moins de n caractères
- La parade est d'ajouter un *sel* au mot de passe, c'est-à-dire une chaîne de caractères aléatoire, qu'on stocke avec le mot de passe.
- Plus d'explications et d'autres conseils et bonnes pratiques dans cet article [https://codingkilledthecat.wordpress.com/2012/09/04/some-best-practices-for-web-app-authentication/]

Gestion des mots de passe en PHP

- Avec PHP (≥5.5), il est *très simple* de prendre ces quelques précautions pour manipuler des mots de passe
- Il suffit d'utiliser la fonction `password_hash()` [http://www.php.net/manual/fr/function.password-hash.php] :
  - permet d'utiliser `bcrypt`
  - ajoute automatiquement un sel de bonne qualité
- Par exemple, pour récupérer l'empreinte de 'abc1234' :

```
$hash = password_hash("abc1234", PASSWORD_BCRYPT);
```

- Le résultat sera une chaîne de 60 caractères (ou `false` en cas de problème) ressemblant à

```
$2y$10$UCsnhCjQZ4X7tj36DLSp0OPLzQYDnd7JEg/84DXLUBtHKhfHduG6y
```

  contenant différentes informations :
  - l'identifiant de l'algorithme utilisé
  - le coût utilisé pour le calcul (= le logarithme du nombre d'itérations effectuées)
  - le sel utilisé pour cette empreinte
  - l'empreinte elle-même.
- Il suffit de stocker cette chaîne dans la base de données pour disposer de toutes les informations nécessaires à la vérification ultérieure du mot de passe.

Vérification du mot de passe en PHP

- Pour vérifier qu'un mot de passe correspond à une empreinte donnée, `password_verify()` fait tout le travail pour vous, en utilisant les informations stockées dans la chaîne :

```
if (password_verify('toto', $hash)) {
            echo "Le mot de passe est 'toto'\n";
} else {
            echo "Le mot de passe n'est pas 'toto'\n";
}
```

- Voir si besoin ce post [http://stackoverflow.com/a/6337021/1749513] qui explique comment utiliser le *hash* `bcrypt` sur différentes versions de PHP.

Connexion et déconnexion

- Le fait que l'internaute est connecté·e ou non est simplement enregistré dans une variable de session.
- Souvent, on mettra un formulaire de connexion sur toutes les pages
- Presque toujours, on mettra un formulaire de déconnexion sur toutes les pages

  ➤ Il est alors recommandé de replacer l'internaute sur la page courante après sa connexion/déconnexion
- La meilleure solution est donc de gérer connexion et déconnexion comme des cas particuliers, dès le début du routeur
- Cependant pour simplifier il est parfaitement possible de se contenter d'une page de connexion et d'une page de déconnexion, qu'on peut gérer comme les autres pages

Restriction des accès

- Pour empêcher l'accès à certaines pages/actions, il suffit de vérifier pour chaque cas si l'internaute a le droit d'accès, et de générer une page spéciale de type « accès interdit » dans le cas contraire
- La vérification des droits dépend de l'application. Par exemple pour un site de petites annonces, on pourrait :
  - n'autoriser que l'accès à la liste des annonces et aux pages des annonces pour les internautes non authentifié·e·s

    ➤ vérification très simple
  - n'autoriser l'édition/suppression que de ses propres annonces
    - nécessite de savoir à qui appartient chaque annonce et de faire la vérification avant de générer les pages de modification/suppression…
    - … mais aussi de retirer les liens vers ces pages

      ➤ on peut envisager de faire une vue différente (qui pourrait hériter de la vue principale, par exemple)
  - faire un espace d'administration, pour gérer les utilisateurs

➤ zone potentiellement très différente : on utilisera certainement une autre vue et un autre contrôleur

Conseil pour la gestion des droits

- Principe de base de sécurité : une « liste blanche » est plus sûre qu'une « liste noire »
- L'idée est qu'il est difficile (voire impossible) de prévoir tous les cas de malveillance

  ➤ il vaut toujours mieux tout interdire par défaut et autoriser au cas par cas
- Dans le contexte de la gestion des accès sur un site, l'application de cette règle revient à refuser par défaut l'accès à tout le monde

  ➤ Les cas où l'accès est autorisés sont ceux explicitement listés.
- Conséquence : on ne *peut pas* oublier de sécuriser une page (ce qui serait grave, car on pourrait ne pas s'en rendre compte pendant très longtemps).
- Si l'inverse se produit, càd qu'on oublie d'autoriser un accès légitime, c'est moins grave, et on s'en rendra vite compte.

Tutoriels

- Some best practices for web app authentication [https://codingkilledthecat.wordpress.com/2012/09/04/some-best-practices-for-web-app-authentication/]

Lectures complémentaires

- Enough with the rainbow tables: what you need to know about secure password schemes [http://www.securityfocus.com/blogs/262]

Outils

- `password_compat` [https://github.com/ircmaxell/password_compat] : bibliothèque de compatibilité de `password_hash` pour PHP<5.5

 [http://creativecommons.org/licenses/by/4.0/]