Grzadziel Bartlomiej
Manon Dupré

Analyse de sécurité de notre site

Voici trois méthodes que nous avons implémenté dans notre site afin d'améliorer sa résistance aux attaques malveillantes :

I. Hachage des mots de passe avec bcrypt

Exemple explicatif :
Les mots de passe stockés en clair dans une base de données constituent une vulnérabilité critique. Un attaquant qui parvient à accéder à la base de données peut lire et utiliser ces mots de passe directement, ce qui peut compromettre non seulement l'application en question, mais aussi d'autres comptes de l'utilisateur si celui-ci réutilise le même mot de passe sur différents sites.

Prenons un exemple concret. Imaginons une base de données contenant les identifiants et mots de passe en clair :

```
| username | password   |
|----------|------------|
| user1    | password123|
| user2    | qwerty     |
```

Si un attaquant accède à cette base de données, il peut immédiatement voir et utiliser les mots de passe pour accéder aux comptes des utilisateurs. De plus, la base de données contient souvent le mail de l'utilisateur. Et sachant que la majorité des utilisateurs d'internet ne détiennent pas de mots de passe différents pour chacun de leur compte, avec ces deux informations, c'est toutes les données sécurisées des utilisateurs qui pourraient être en danger.

Solution : Hachage des mots de passe avec bcrypt
Pour protéger les mots de passe, une méthode efficace consiste à les hacher avant de les stocker. Le hachage est un processus unidirectionnel qui transforme le mot de passe en une chaîne de caractères unique. bcrypt est un algorithme de hachage spécialement conçu pour les mots de passe, car il est résistant aux attaques par force brute.

En appliquant bcrypt, la base de données précédente serait transformée comme suit :

```
| username | password                                          |
|----------|---------------------------------------------------|
| user1    | zahPqEX7fNZaFWoOBL5x8fgf3N1/bxF1fU9hD8tlvTjTe1W |
| user2    | qsdqsdX7fNZaFWoOBL5x8fgf3N1/bxF1fU9hD8tlvTjTe1W |
```

Application :
Pour implémenter bcrypt dans votre application, vous pouvez utiliser une bibliothèque comme bcrypt pour hacher les mots de passe avant de les stocker dans la base de données. Voici un exemple d'implémentation en JavaScript :

```
const bcrypt = require('bcryptjs');
// Hacher un mot de passe
const password = 'password123';
bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(password, salt, (err, hash) => {
        if (err) throw err;

        // Afficher le mot de passe haché
        console.log('Mot de passe haché:', hash);

        // Vérifier un mot de passe
        bcrypt.compare(password, hash, (err, isMatch) => {
            if (err) throw err;
            if (isMatch) {
                console.log("Le mot de passe correspond");
            } else {
                console.log("Le mot de passe ne correspond pas");
            }
        });
    });
});
```

L'impact de cette solution est significatif. Même si un attaquant parvient à accéder à la base de données, il ne pourra pas facilement récupérer les mots de passe originaux en raison de la robustesse de l'algorithme bcrypt.

II. Injonction de requête SQL de manière sécurisée

Exemple explicatif :
Les attaques par injection SQL surviennent lorsqu'un attaquant peut insérer ou manipuler des requêtes SQL pour exécuter des actions non autorisées sur une base de données. Un exemple courant est l'utilisation de champs de formulaire non sécurisés :

```
SELECT * FROM users WHERE username = 'admin' AND password = '';
```

Si un attaquant entre `' OR '1'='1'` comme mot de passe, la requête devient :

```
SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1';
```

Cette requête retourne tous les enregistrements de la table `users`, permettant à l'attaquant de se connecter sans connaître le mot de passe.

Solution : Préparation des requêtes SQL
Pour prévenir les injections SQL, il est essentiel d'utiliser des requêtes préparées avec des paramètres. Cela garantit que les entrées utilisateur sont traitées comme des valeurs de données et non comme des instructions SQL.

Application :

Voici en exemple comment nous avons incorporé cette méthode pour lire les informations d'un utilisateur :

```
module.exports = {
    // Fonction pour lire les informations d'un utilisateur spécifique par mail.
    read: function (mail, callback) {
        // Exécution d'une requête SQL pour sélectionner un utilisateur par son mail.
        db.query("SELECT * FROM Utilisateur WHERE mail = ?", mail, function (err, results) {
            // Gestion des erreurs lors de l'exécution de la requête.
            if (err) throw err;
            // Retour des résultats via la fonction de callback.
            callback(results);
        });
    },
```

En utilisant des requêtes préparées, même si un attaquant tente de manipuler les entrées, elles seront traitées comme des données et non comme des commandes SQL. Cela protège l'application contre les injections SQL.

III. Obligation de mots de passe conformes aux règles de la CNIL

Exemple explicatif :
Les mots de passe faibles ou faciles à deviner constituent une vulnérabilité majeure pour la sécurité des comptes utilisateurs. Un attaquant peut utiliser des techniques comme le "brute force" ou des attaques par dictionnaire pour deviner ces mots de passe. Par exemple, des mots de passe courants comme "123456" ou "password" sont facilement devinables.

Prenons un exemple : si un utilisateur utilise "password123" comme mot de passe, un attaquant peut rapidement le deviner en utilisant une liste de mots de passe courants. Cela peut conduire à une compromission de compte, permettant à l'attaquant d'accéder à des informations sensibles.

Solution : Imposition de règles strictes de mot de passe
Pour renforcer la sécurité des mots de passe, il est essentiel de suivre les recommandations de la CNIL, qui exigent que les mots de passe soient d'une longueur minimale de 12 caractères, et contiennent une combinaison de lettres majuscules et minuscules, de chiffres et de caractères spéciaux. Ces règles rendent les mots de passe beaucoup plus difficiles à deviner ou à casser par des attaques automatisées.

Voici une règle de validation de mot de passe conforme aux recommandations de la CNIL :

- Longueur minimale de 12 caractères.
- Au moins un caractère spécial (ex: !@#$%^&*).
- Au moins un chiffre (0-9)

Application :

```
function isPasswordValid(password) {
    const minLength = 12;
    const specialCharPattern = /[!@#$%^&*(),.?":{}|<>]/; // RegEx pour les caractères spéciaux
    const digitPattern = /\d/; // RegEx pour les chiffres

    if (password.length < minLength) {
        alert('Le mot de passe doit contenir au moins 12 caractères.');
        return false;
    }

    if (!specialCharPattern.test(password)) {
        alert('Le mot de passe doit contenir au moins un caractère spécial.');
        return false;
    }

    if (!digitPattern.test(password)) {
        alert('Le mot de passe doit contenir au moins un chiffre.');
        return false;
    }

    return true;
}
```

En utilisant cette validation, nous forçons les utilisateurs à choisir des mots de passe forts et conformes aux recommandations de la CNIL, réduisant ainsi considérablement le risque de compromission des comptes.