Recherche de site Web

Comment hacher et vérifier un mot de passe dans Node.js avec bcrypt

Si vous travaillez avec les mots de passe des utilisateurs, vous leur devez de le faire en toute sécurité. La bibliothèque bcrypt simplifie ce processus.

L’un des meilleurs moyens de stocker les mots de passe en toute sécurité est de les saler et de les hacher. Le salage et le hachage convertissent un mot de passe simple en une valeur unique difficile à inverser. La bibliothèque Bcrypt vous permet de hacher et de saler des mots de passe dans Node.js avec très peu d'effort.

Qu’est-ce que le hachage de mot de passe ?

Le hachage de mot de passe consiste à transmettre un mot de passe en texte brut via un algorithme de hachage pour générer une valeur unique. Cette valeur unique est appelée hachage. Quelques exemples d'algorithmes de hachage sont bcrypt, scrypt et SHA.

L’une des principales propriétés d’un bon algorithme de hachage est qu’il génère la même sortie pour la même entrée. Cette prévisibilité rend les hachages vulnérables aux attaques par force brute. Un pirate informatique peut précalculer les valeurs de hachage pour de nombreuses entrées couramment utilisées, puis les comparer aux valeurs de hachage des valeurs cibles. Vous pouvez atténuer cette vulnérabilité en utilisant le salage.

Qu’est-ce que le salage de mots de passe ?

Le salage de mot de passe ajoute une chaîne aléatoire (le sel) à un mot de passe avant de le hacher. De cette façon, le hachage généré sera toujours différent à chaque fois. Même si un pirate informatique obtient le mot de passe haché, il lui faudra un temps considérable pour découvrir le mot de passe d'origine qui l'a généré.

Comment utiliser Bcrypt pour hacher et vérifier un mot de passe

bcrypt est un module npm qui simplifie la façon dont vous hachez les mots de passe dans Node.js. Pour l'utiliser, suivez

les étapes ci-dessous :

Étape 1 : Installer Bcrypt

Installez bcrypt en exécutant les commandes de terminal suivantes.

Utilisation de npm :

```
npm install bcrypt
```

Utiliser du fil :

```
yarn add bcrypt
```

Étape 2 : Importer Bcrypt

En haut de votre fichier JavaScript, importez Bcrypt.

```
const bcrypt = require("bcrypt")
```

Étape 3 : générer un sel

Appelez la méthode **bcrypt.genSalt()** pour générer un sel. Cette méthode accepte une valeur entière qui est le facteur de coût qui détermine le temps nécessaire pour hacher un mot de passe. Plus le facteur coût est élevé, plus l'algorithme prend du temps et plus il est difficile d'inverser le mot de passe crypté par force brute.

Une bonne valeur doit être suffisamment élevée pour sécuriser le mot de passe mais également suffisamment basse pour ne pas ralentir le processus. Il se situe généralement entre 5 et 15. Dans ce tutoriel, nous utiliserons 10.

```
bcrypt.genSalt(10, (err, salt) => {
    // use salt to hash password
})
```

Étape 4 : hachez le mot de passe

Dans la fonction **bcrypt.genSalt**, transmettez le mot de passe brut et le sel généré à la méthode **bcrypt.hash()** pour hacher le mot de passe.

```
bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(plaintextPassword, salt, function(err, hash) {
        // Store hash in the database
    });
})
```

Une fois que vous avez généré le hachage, stockez-le dans la base de données. Vous l'utiliserez pour vérifier un mot de passe et authentifier un utilisateur essayant de se connecter.

Au lieu de générer le sel et le hachage séparément, vous pouvez également générer automatiquement le sel et le hachage à l'aide d'une seule fonction.

```
bcrypt.hash(plaintextPassword, 10, function(err, hash) {
    // store hash in the database
});
```

Étape 5 : Comparez les mots de passe à l'aide de bcrypt

Pour authentifier les utilisateurs, vous devrez comparer le mot de passe qu'ils fournissent avec celui de la base de données à l'aide de la fonction **bcrypt.compare()**. Cette fonction accepte le mot de passe en texte brut et le hachage que vous avez stocké, ainsi qu'une fonction de rappel. Ce rappel fournit un objet contenant toutes les erreurs survenues et le résultat global de la comparaison. Si le mot de passe correspond au hachage, le résultat est vrai.

```
bcrypt.compare(plaintextPassword, hash, function(err, result) {
    if (result) {
        // password is valid
    }
});
```

Utilisation d'Async/Await

Vous pouvez chiffrer les mots de passe dans Node.js avec Bcrypt en utilisant async/await comme suit.

```
async function hashPassword(plaintextPassword) {
    const hash = await bcrypt.hash(plaintextPassword, 10);
    // Store hash in the database
}

// compare password
async function comparePassword(plaintextPassword, hash) {
    const result = await bcrypt.compare(plaintextPassword, hash);
    return result;
}
```

Utiliser les promesses

La bibliothèque bcrypt prend également en charge l'utilisation de promesses. Par exemple, voici une fonction qui hache le mot de passe à l'aide du bloc then...catch.

```
function hashPassword(plaintextPassword) {
    bcrypt.hash(plaintextPassword, 10)
        .then(hash => {
            // Store hash in the database
        })
        .catch(err => {
            console.log(err)
        })
}
```

De même, cette fonction compare un mot de passe simple de l'utilisateur à un mot de passe haché à l'aide de promesses.

```
function comparePassword(plaintextPassword, hash) {
   bcrypt.compare(plaintextPassword, hash)
       .then(result => {
           return result
       })
       .catch(err => {
           console.log(err)
       })
}
```

Le hachage et le salage sont une victoire facile

Vous pouvez utiliser la bibliothèque Bcrypt pour hacher et vérifier les mots de passe dans Node.js. Le hachage des mots de passe minimise les chances que les cybercriminels accèdent à des mots de passe simples et les utilisent pour accéder à des données ou à des services sensibles.

Saler vos mots de passe hachés les rend encore plus sécurisés. Outre le hachage, validez toujours la force du mot de passe comme mesure de sécurité supplémentaire.
