

Compte rendu sécurité

Vulnérabilité n°1 : Contrôle des accès

Certains utilisateurs peuvent avoir accès à des pages réservées à un rôle plus haut hiérarchiquement. Un candidat peut par exemple avoir accès aux pages réservées à l'administrateur ce qui pose un gros problème car il pourrait par exemple modifier les profils des utilisateurs.

Pour remédier à cette vulnérabilité nous avons choisi de modifier nos routes pour y ajouter un contrôle du rôle du candidat.

```
✓ router.get('/admin_main', function (req, res, next) {  
  const session = req.session;  
  ✓ if(session.usermail && session.type_user === "admin") {  
    res.render('admin/admin_main');  
  ✓ }else {  
    res.redirect("/auth/login");  
  }  
});
```

Ici par exemple nous faisons une vérification du rôle de l'utilisateur, si c'est un admin il peut accéder à la page, sinon il est redirigé vers /auth/login qui va l'amener sur la page d'accueil correspondant à son rôle.

Vulnérabilité n°2 : Tentative de bruteforce

Sur un site web il est possible de tenter une connexion par la méthode bruteforce, c'est-à-dire tenter de se connecter en boucle en changeant le mail / mot de passe. Pour cela il est possible d'utiliser des bases de données d'identifiants / mot de passe communs qui sont disponibles sur internet

(exemple : <https://github.com/danielmiessler/SecLists/tree/master>) et d'utiliser des outils tels que Hydra.

Pour se prémunir de ces tentatives d'attaques, nous avons pris plusieurs mesures. Tout d'abord les mots de passe sont chiffrés à l'aide du module bcrypt.

```
create: async (email, nom, prenom, num_tel, password) => {
  return new Promise((resolve, reject) => {
    var ladata = new Date();
    var date_creation = ladata.getFullYear() + "-" + (ladata.getMonth() + 1) + "-" + ladata.getDate();

    bcrypt.hash(password, 10, (err, hash) => {
      if (err) {
        return reject(err);
      }

      db.query("INSERT INTO Utilisateur (id_utilisateur, email, nom, prenom, num_tel, date_creation, last_login, statut, password, role) \
VALUES(NULL, ?, ?, ?, ?, ?, 1, ?, 'candidat');", [email, nom, prenom, num_tel, date_creation, date_creation, hash], (err, results) => {
        if (err) {
          reject(err);
        } else {
          var id = results.insertId;
          resolve(id);
        }
      });
    });
  });
},
});
```


Ensuite, nous avons rajouté un captcha à la connexion à notre site. Ainsi il n'est pas possible de programmer un bot pour tenter des connexions en boucle étant donné qu'il ne peut pas remplir le captcha.

Connexion

Se connecter


Email

Mot de passe

☐ Je ne suis pas un robot 
Confidentialité - Conditions

Se connecter

C'est votre première visite ? [Créer un compte](#)



La connexion est bien empêchée lorsque le captcha n'est pas validé :

Connexion

Se connecter

Email


Mot de passe

CAPTCHA non valide

☐ Je ne suis pas un robot 
Confidentialité - Conditions

Se connecter

C'est votre première visite ? [Créer un compte](#)



Enfin nous avons rajouté une limite d'erreur lors de la tentative de connexion. Lorsqu'un utilisateur essaye de se connecter plus de 5 fois, la connexion sera empêchée pendant 15 minutes.

Vulnérabilité 3 : Injections SQL

Sur un site web sans sécurité, il est possible avec certaines entrées dans les champs de texte de modifier le contenu de la base de données.

Par exemple, sur le champ mot de passe de cette page, en supposant que la requête attendue soit :

```
select * from Utilisateur where email='quentin.fouinat-beal@etu.utc.fr' and password = 'password'
```


L'entrée xxx' or 'a' = 'a' aurait alors l'effet de sélectionner les informations de tous les utilisateurs de la base.

Pour lutter contre cela, nous avons mis en place des placeholders pour les requêtes de toutes nos fonctions.

Email

Mot de passe

Email ou mot de passe incorrect.

☐ Je ne suis pas un robot 
Confidentialité - Conditions

Se connecter

```
arevalid: async (email, password) => {  
  return new Promise((resolve, reject) => {  
    const sql = "SELECT password FROM Utilisateur WHERE email = ?";  
    db.query(sql, email, (err, results) => {  
      if (err) {  
        reject(err);  
      }  
      if (results.length == 1) {  
        bcrypt.compare(password, results[0]["password"], (err, result) => {  
          if (err) {
```

On constate que les informations sont ici insérées dans la chaîne de caractère de la requête SQL via des placeholders dans la méthode query et non directement "à la main". Cela permet d'empêcher la concaténation des chaînes de caractères.

Petit plus :

En plus des vulnérabilités présentées ci-dessus, nous avons obligé les utilisateurs à avoir des mots de passe sécurisés, soit au moins 12 caractères, dont au moins 1 majuscule, 1 minuscule, 1 chiffre et 1 caractère spécial.