LISTES DES VULNÉRABILITÉS ET DES CONTRE MESURE

1. Contre-mesures pour l’Injection NoSQL

Exemple 1 : Validation des entrées avec Joi

Dans **Node.js**, utilise **Joi** pour valider et filtrer les entrées utilisateur avant de les envoyer à MongoDB.

```
const Joi = require('joi');

const userSchema = Joi.object({
  username: Joi.string().alphanum().min(3).max(30).required(),
  password: Joi.string().min(8).required()
});

app.post('/register', async (req, res) => {
  const { error } = userSchema.validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  const user = new User(req.body);
  await user.save();
  res.send("User registered successfully.");
});
```

✅ **Pourquoi ?** Cela empêche un attaquant d'injecter du code malveillant via les entrées utilisateur.

Exemple 2 : Utilisation de paramètres sécurisés au lieu de `$regex`

⚠️ Mauvaise pratique (vulnérable) :

```
db.users.find({ username: { $regex: /" + req.body.username + "/" } });
```

✅ Bonne pratique (sécurisée)

```
const sanitizedUsername = req.body.username.replace(/[^\w]/g, '');
db.users.find({ username: sanitizedUsername });
```

✅ **Pourquoi ?** Empêche l’exploitation des **expressions régulières non sécurisées**.

2. Contre-mesures pour les Contrôles d’Accès Inadéquats

Exemple 3 : Configuration de RBAC dans MongoDB

⚠️ Mauvaise pratique : Un utilisateur ayant **tous les privilèges** sur la base de données :

```
db.createUser({
  user: "developer",
  pwd: "password123",
  roles: [{ role: "dbOwner", db: "appDB" }]
});
```

✅ Bonne pratique : Utiliser un **rôle spécifique avec des permissions limitées**

```
db.createUser({
  user: "readonlyUser",
  pwd: "securepassword",
  roles: [{ role: "read", db: "appDB" }]
});
```

✅ **Pourquoi ? Empêche un utilisateur malveillant d'obtenir plus de droits que nécessaire**

3. Contre-mesures pour l'Absence de Validation du Schéma

Exemple 4 : Définition d'un schéma strict dans MongoDB

Dans MongoDB, active la validation de schéma pour empêcher l'insertion de données non conformes.

```
db.createCollection("users", {
  validator: {
        $jsonSchema: {
        bsonType: "object",
        required: ["username", "password", "email"],
        properties: {
        username: {
        bsonType: "string",
        description: "must be a string and is required"
        },
        password: {
        bsonType: "string",
        minLength: 8,
        description: "must be at least 8 characters long"
        },
        email: {
        bsonType: "string",
```

```
        pattern: "^\\S+@\\S+\\.\\S+$",
        description: "must be a valid email format"
        }
        }
        }
 }
});
```

✅ **Pourquoi ? Cela bloque les données mal formatées ou injectées.**

4. Contre-mesures pour les Données en Transit Non Sécurisées

Exemple 5 : Activation de TLS/SSL pour MongoDB

Ajoute l'option SSL obligatoire dans la configuration de MongoDB.

```
mongod --sslMode requireSSL --sslPEMKeyFile /etc/ssl/mongodb.pem --sslCAFile /etc/ssl/ca.pem
```

✅ **Pourquoi ? Empêche les attaques MITM (Man-in-the-Middle) en chiffrant les échanges entre l'application et la base de données.**

Exemple 6 : Connexion sécurisée depuis l'application Node.js

⚠️ **Mauvaise pratique : Connexion non sécurisée à MongoDB.**

```
mongoose.connect("mongodb://localhost:27017/appDB");
```

✅ **Bonne pratique : Activer TLS dans la connexion :**

```
mongoose.connect("mongodb://yourserver:27017/appDB?ssl=true", {
  ssl: true,
  sslValidate: true,
  sslCA: "/etc/ssl/ca.pem"
});
```

✅ **Pourquoi ? Garantit que toutes les connexions sont chiffrées.**

5. Contre-mesures pour le Stockage Non Sécurisé des Mots de Passe

Exemple 7 : Hashage des mots de passe avec bcrypt

Ne jamais stocker des mots de passe en clair !

```
const bcrypt = require('bcrypt');
const saltRounds = 10;

async function registerUser(req, res) {
  const hashedPassword = await bcrypt.hash(req.body.password, saltRounds);
  const user = new User({ username: req.body.username, password: hashedPassword
});
  await user.save();
  res.send("User registered securely.");
}
```

✅ **Pourquoi ? Même si la base est compromise, les mots de passe restent protégés.**

Résumé des Contre-mesures et Exemples

Vulnérabilité	Contre-mesure	Exemple
Injection NoSQL	- Valider les entrées avec Joi. - Utiliser des paramètres sécurisés au lieu de `$regex`.	`Joi.validate()` + Suppression des caractères spéciaux
Contrôles d'accès inadéquats	- Configurer RBAC. - Donner des permissions minimales aux utilisateurs.	`db.createUser({ roles: ["read"] })`
Validation du schéma absente	- Définir un schéma strict dans MongoDB.	`$jsonSchema` pour forcer les types de données
Données en transit non sécurisées	- Activer TLS/SSL. - Utiliser des connexions sécurisées avec Mongoose.	`mongod --sslMode requireSSL`
Stockage des mots de passe non sécurisé	- Hashage avec bcrypt. - Ne jamais stocker de mot de passe en clair.	`bcrypt.hash(password, saltRounds)`

Conclusion

Ces exemples concrets permettent de renforcer la sécurité de MongoDB en :
✅ Bloquant les injections NoSQL avant qu'elles n'atteignent la base.
✅ Restreignant les accès utilisateurs pour limiter l’impact d’une attaque.
✅ Appliquant un schéma strict pour empêcher l’insertion de données malveillantes.
✅ Chiffrant les données en transit pour éviter les interceptions.
✅ Stockant les mots de passe de manière sécurisée pour prévenir les fuites.