NoSQL Injection Application

Index

1. Project Overview

This application demonstrates NoSQL Injection vulnerabilities in MongoDB-based applications when untrusted input is used directly in query objects.

2. Learning Outcomes

- Understand NoSQL Injection
- Identify unsafe MongoDB queries
- Exploit authentication bypass
- Apply secure query practices

3. Scope

In scope: NoSQL injection attack and mitigation.
Out of scope: SQL injection, ORM-level protection.

4. Tech Stack & Versions

• Node.js
• Express.js
• MongoDB
• Mongoose (optional)

5. Folder Structure

```
nosql_injection/
├── app.js
├── routes/auth.js
├── models/user.js
├── package.json
└── README.md
```

6. High-Level Architecture

```
Client → Express API → MongoDB
```

7. Vulnerability Demonstration Flow

```
1. Crafted JSON payload sent
2. MongoDB query logic altered
3. Authentication bypass
```

8. Attack Scenarios

```
{ "username": { "$ne": null }, "password": { "$ne": null } }
```

9. Secure Coding Fix

```
• Enforce input types
• Reject MongoDB operators
• Use allow-list validation
```

10. API Contract

```
POST /login
```

11. How to Run

```
npm install
node app.js
```

12. Common Mistakes

- Assuming NoSQL is immune to injection
- Using req.body directly in queries

13. Debugging Techniques

- Log request payloads
- Inspect generated MongoDB queries

Appendix A: Source Code

models/User.js

```
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  username: { type: String, required: true },
  password: { type: String, required: true }
}, { strict: true });

module.exports = mongoose.model("User", userSchema);
```

package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^5.2.1",
    "mongoose": "^9.2.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.11"
  }
}
```

public/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Injection Demo</title>
  <style>
    body { font-family: Arial; margin: 40px; }
    input { display: block; margin: 10px 0; padding: 6px; width: 250px; }
    button { margin-right: 10px; padding: 6px 12px; }
    pre { background: #f4f4f4; padding: 10px; }
  </style>
</head>
<body>
```

```
<h2>Login Demo (Injection)</h2>

<input id="username" placeholder="Username" />
<input id="password" placeholder="Password" />

<button onclick="login('login-vulnerable')">Login (Vulnerable)</button>
<button onclick="login('login-secure')">Login (Secure)</button>

<h3>Response:</h3>
<pre id="result"></pre>

<script src="script.js"></script>
</body>
</html>
```

public/script.js

```
async function login(type) {
  const username = document.getElementById("username").value;
  const password = document.getElementById("password").value;
  const BASE_URL = "http://localhost:3000";

  let payload;

  // Allow JSON injection input
  try {
    payload = {
      username: JSON.parse(username),
      password: JSON.parse(password)
    };
  } catch {
    payload = { username, password };
  }

  const res = await fetch(`${BASE_URL}/auth/${type}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(payload)
  });

  const text = await res.text();
  document.getElementById("result").innerText = text;
}
```

readme.txt

```
Step 1: Install and Start
npm i
npm start
```

```
Step 2: Insert a test user (mandatory)

Open Mongo shell:

mongosh

use injection_demo

db.users.insertOne({
  username: "admin",
  password: "1234"
})

Step 3: Demo scenarios

Open browser
http://localhost:5000

Step 3.1 : Normal login
Username: admin
Password: 1234

Button     Result
Vulnerable    Success
Secure   Success

Injection attack
Username: {"$ne": null}
Password: {"$ne": null}

Button     Result
Vulnerable    Login Successful
Secure     Invalid Input
```

routes/auth.js

```
const express = require("express");
const router = express.Router();
const User = require("../models/User");

// Vulnerable Login (Injection Possible)
router.post("/login-vulnerable", async (req, res) => {
  const user = await User.findOne(req.body);
  if (user) res.send("Login Successful (Vulnerable)");
  else res.status(401).send("Invalid Credentials");
});

// Secure Login (Prepared-Statement Equivalent)
router.post("/login-secure", async (req, res) => {
  const { username, password } = req.body;
```

```
  if (typeof username !== "string" || typeof password !== "string") {
    return res.status(400).send("Invalid Input Type");
  }

  const user = await User.findOne({ username, password }).lean();
  if (user) res.send("Login Successful (Secure)");
  else res.status(401).send("Invalid Credentials");
});

module.exports = router;
```

server.js

```
const express = require("express");
const mongoose = require("mongoose");
const authRoutes = require("./routes/auth");
const path = require("path");

const app = express();

// serve static files
app.use(express.static(path.join(__dirname, "public")));
app.use(express.json());

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

app.use("/auth", authRoutes);

app.listen(3000, () => console.log("Backend running on port 3000"));
```