**Task 3 :-** secure coding review of an application by reviewing its vulnerabilities and providing recommendations to solve the problem .

Here , I used a code with node.js with express, with vulnerabilities and solved it .

**Secure Coding Review for Node.js Application**

Vulnerable Code (Node.js with Express)

This application has a SQL injection vulnerability because it directly uses user input in SQL queries without proper sanitization.

Vulnerable Code:

```javascript

// app.js

const express = require('express');

const sqlite3 = require('sqlite3').verbose();

const app = express();

const port = 3000;

let db = new sqlite3.Database(':memory:');

db.serialize(() => {

db.run("CREATE TABLE users (id INT, name TEXT)");

let stmt = db.prepare("INSERT INTO users VALUES (?, ?)");

stmt.run(1, 'Alice');

stmt.run(2, 'Bob');

stmt.finalize();

});

app.get('/user', (req, res) => {

let id = req.query.id;

db.get(`SELECT name FROM users WHERE id = ${id}`, (err, row) => {

if (err) {

res.status(500).send("An error occurred");

return;

}

res.send(`User: ${row ? row.name : 'Not found'}`);

});

});

app.listen(port, () => {

console.log(`App running on port ${port}`);

});

```

Secure Code (Node.js with Express)

This version fixes the SQL injection vulnerability by using parameterized queries.

Secure Code:

```javascript

// app.js

const express = require('express');

const sqlite3 = require('sqlite3').verbose();

const app = express();

const port = 3000;

let db = new sqlite3.Database(':memory:');

db.serialize(() => {

db.run("CREATE TABLE users (id INT, name TEXT)");

let stmt = db.prepare("INSERT INTO users VALUES (?, ?)");

stmt.run(1, 'Alice');

stmt.run(2, 'Bob');

stmt.finalize();

});

app.get('/user', (req, res) => {

let id = req.query.id;

db.get(`SELECT name FROM users WHERE id = ?`, [id], (err, row) => {

if (err) {

res.status(500).send("An error occurred");

return;

}

res.send(`User: ${row ? row.name : 'Not found'}`);

});

});

app.listen(port, () => {

console.log(`App running on port ${port}`);

});

```

**Secure Coding Review**

Vulnerability Description

- SQL Injection : The vulnerable code directly interpolates user input into a SQL query string. This allows an attacker to manipulate the SQL query by injecting malicious SQL code via the `id` parameter.

Example Exploitation

- An attacker could access user data by visiting `http://localhost:3000/user?id=1 OR 1=1`, which would return all rows in the users table because the condition `1=1` is always true.

**Secure Coding Practices and Recommendations**

1. Use Parameterized Queries:

- Always use parameterized queries or prepared statements when interacting with a database. This ensures that user input is treated as data and not executable code.

Fix:

```javascript

db.get(`SELECT name FROM users WHERE id = ?`, [id], (err, row) => {

...

});

```

2. Input Validation and Sanitization:

- Validate and sanitize all user inputs. Ensure that inputs conform to the expected format and length before processing them.

Example:

```javascript

const id = parseInt(req.query.id, 10);

if (isNaN(id)) {

res.status(400).send("Invalid ID");

return;

}

```

3. Error Handling:

- Implement robust error handling to avoid exposing sensitive information through error messages.

Example:

```javascript

db.get(`SELECT name FROM users WHERE id = ?`, [id], (err, row) => {

if (err) {

res.status(500).send("An error occurred");

return;

}

res.send(`User: ${row ? row.name : 'Not found'}`);

});

```

4. Least Privilege Principle:

- Ensure that the application runs with the least privileges required to perform its tasks. This limits the impact of a successful attack.

Example :

- Use a database user with restricted permissions, only allowing necessary operations.

**Conclusion**

By following these secure coding practices and recommendations, you can significantly reduce the risk of SQL injection and other common vulnerabilities in your applications. Always validate user inputs, use parameterized queries, handle errors properly, and follow the principle of least privilege to enhance the security of your applications.