Week 2: Implementing Security Measures

1. Fixing Vulnerabilities

🔒 Input Validation & Password Hashing

To enhance application security, user inputs were validated and passwords were securely stored using hashing:

In index.js, the following logic was implemented for login authentication:

javascript

CopyEdit

```
const validator = require('validator');

const bcrypt = require('bcrypt');


app.post('/login', async function(req, res) {

  const { username, password } = req.body;


  if (!username || !password) {

    return res.status(400).send("Missing credentials");

  }


  pool.query('SELECT * FROM admin WHERE username = ?', [username], async function(err, rows) {

    if (err || rows.length === 0) {

      return res.status(400).send("User not found");

    }


    const isMatch = await bcrypt.compare(password, rows[0].password);

    if (isMatch) {
```

```
    const token = jwt.sign(
      { id: rows[0].id, username: rows[0].username },
      'your-secret-key',
      { expiresIn: '1h' }
    );
    console.log(token);
    res.status(200).send({
      message: "Authentication successful",
      token: token
    });
  } else {
    return res.status(401).send("Invalid credentials");
  }
 });
});
```

This ensures only validated users with correct credentials can access the system, and their passwords remain secure through hashing.

---

2. Enhancing Authentication

To protect sensitive operations (CRUD), token-based authentication was introduced using JWT and middleware.

📦 Token Authentication Middleware

File: middleware/authenticateToken.js

javascript

CopyEdit

```
const jwt = require('jsonwebtoken');
```

```
function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Bearer <token>

  if (!token) return res.status(401).send('Access Denied');

  jwt.verify(token, 'your-secret-key', (err, user) => {
    if (err) return res.status(403).send('Invalid Token');
    req.user = user;
    next();
  });
}

module.exports = authenticateToken;
```

This middleware is added to all important routes in index.js:

javascript

CopyEdit

```
const authenticateToken = require('./middleware/authenticateToken');
```

🔐 Protected CRUD Operations

Create Route:

javascript

CopyEdit

```
app.post('/create', authenticateToken, function(req, res) {
  const userData = {
    name: req.body.Name,
```

```
    studentID: req.body.StudentID,
    department: req.body.Department
  };

  pool.query('INSERT INTO user SET ?', userData, function(err) {
    if (err) return res.status(400).send("Unable to insert into database");
    res.status(200).send("User Added");
  });
});
```

Read/List Users:

javascript

CopyEdit

```
app.get('/list', authenticateToken, function(req, res) {
  pool.query('SELECT * FROM user', (err, result) => {
    if (err) return res.status(400).send("Error in Connection");
    res.status(200).send(result);
  });
});
```

Delete User:

javascript

CopyEdit

```
app.delete('/delete/:id', authenticateToken, function(req, res) {
  pool.query('DELETE FROM user WHERE studentID = ?', [req.params.id], (err) => {
    if (err) return res.status(400).send("User not found");

    pool.query('SELECT * FROM user', (err, result) => {
```

```
    if (err) return res.status(400).send("Error in Connection");
    res.status(200).send(result);
  });
 });
});
```

---

3. Securing Data Transmission

To defend against common web threats like XSS, clickjacking, and MIME sniffing, the helmet middleware was added to enforce secure HTTP headers.

🛡️ Helmet Integration in index.js

javascript

CopyEdit

```
const helmet = require('helmet');


app.use(helmet.contentSecurityPolicy({
 directives: {
  defaultSrc: ["'self'"],
  scriptSrc: ["'self'", "'unsafe-inline'"], // Update based on frontend requirements
  objectSrc: ["'none'"],
  upgradeInsecureRequests: []
 }
}));
```

This setup helps enforce a strong Content Security Policy (CSP), protecting the application against malicious script injections and content spoofing.

✅ Summary

By the end of Week 2, the following key security enhancements were successfully implemented:

- ✅ **Input validation and secure password hashing with bcrypt**
- ✅ **JWT-based authentication with token middleware to guard all sensitive routes**
- ✅ **CRUD operations protected using authorization headers**
- ✅ **HTTP headers enforced using Helmet for XSS, clickjacking, and MIME-type protection**

These changes significantly improved the overall security posture of the application, aligning it with industry-standard best practices.