

Final Report – Cybersecurity Internship Project

Name: SHEIKH ARSALAN

DHC-ID: DHC-191

Project Used: <https://github.com/Pranavk-Official/user-management>

Summary of All 3 Weeks

Week 1 – Security Assessment

Step 1: Set Up the Web Application

1.1 – Choose a Mock Web Application

1.2 – Download and Run the Application

Open Terminal or Command Prompt

Clone the project:

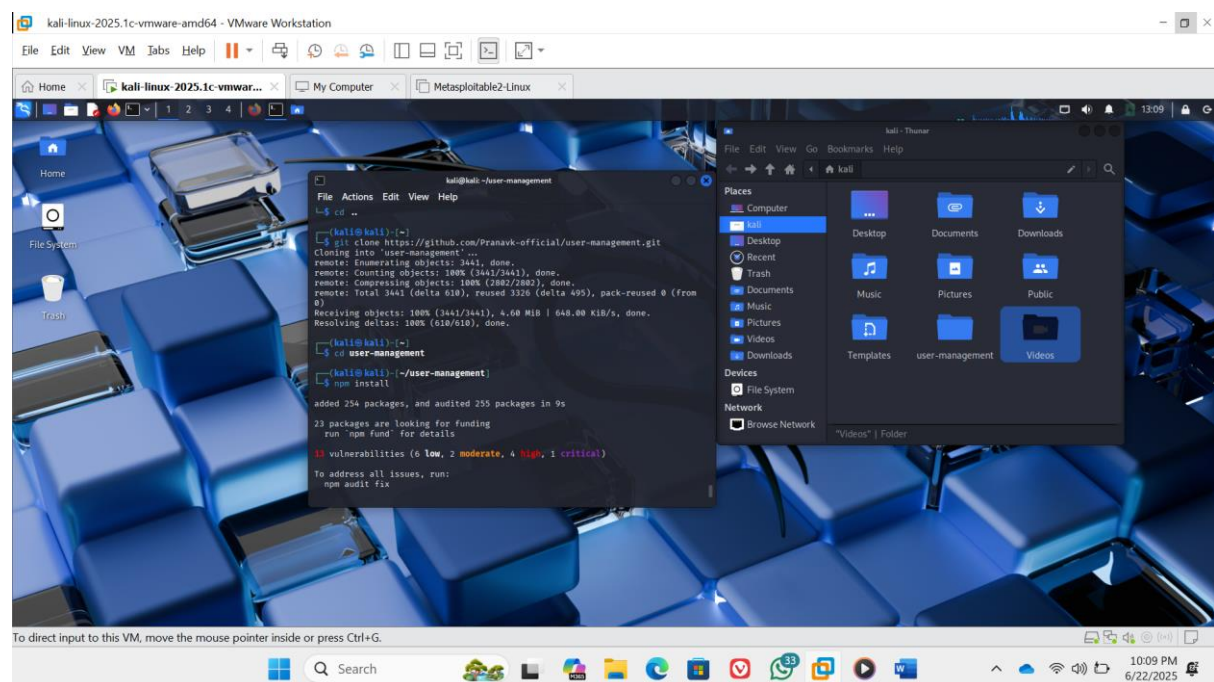
```
git clone <project-link>
```

Go to project folder:

```
cd <project-folder>
```

Install required packages:

```
npm install
```



```
npm start
```



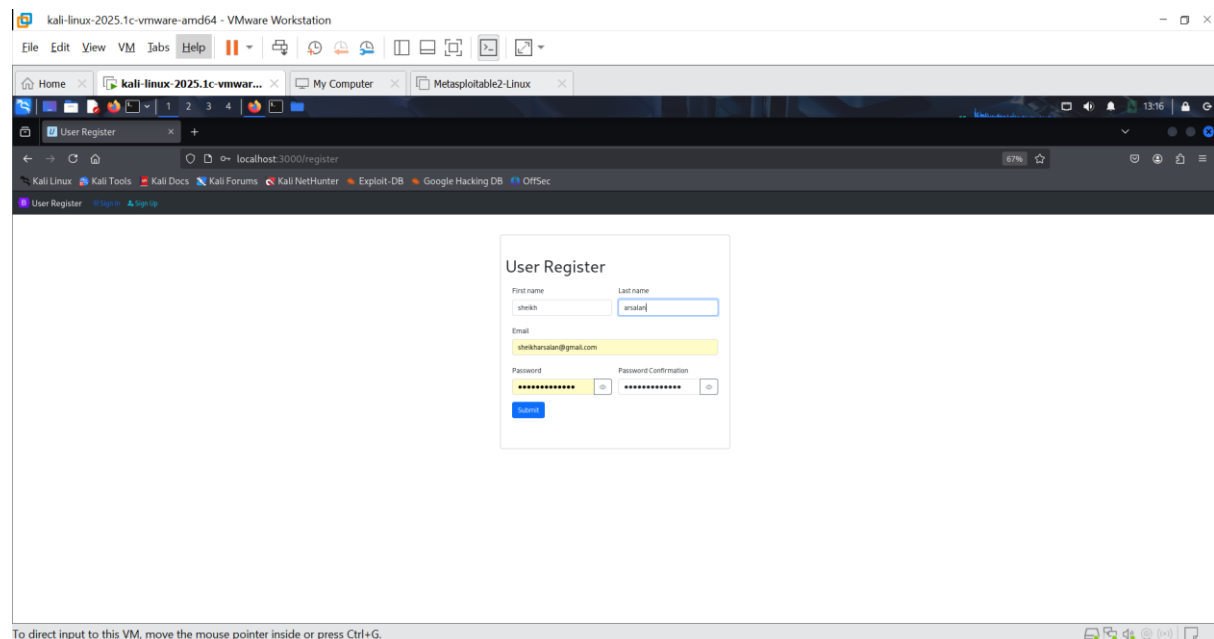
The screenshot shows a Kali Linux virtual machine environment. The browser window displays a web application at 'localhost:3000/login'. The page features a 'User Login' form with the following elements:

- Title:** User Login
- Email Field:** A text input field containing 'example@email.com'.
- Password Field:** A text input field with a toggle icon on the right.
- Submit Button:** A blue button labeled 'Submit'.

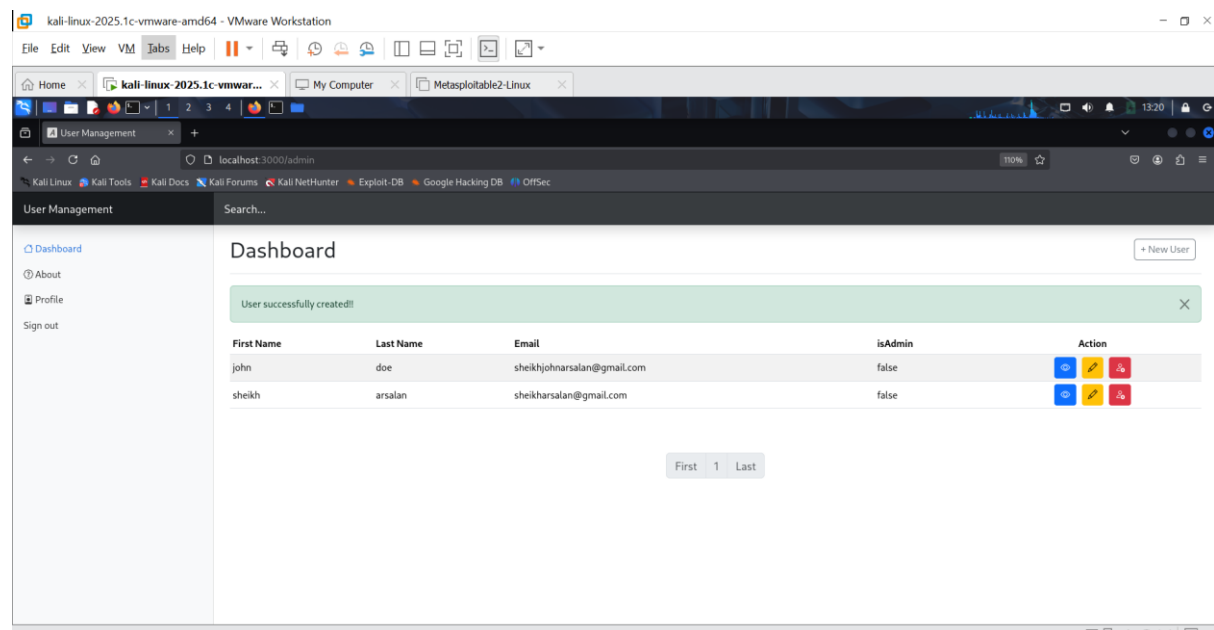
The VMware Workstation interface is visible at the top, showing the VM name 'kali-linux-2025.1c-vmware-amd64'. The bottom of the image shows the Windows taskbar with various application icons and the system clock indicating 10:12 PM on 6/22/2025.

Step 2: Explore the Application

Use the app like a normal user: - Sign up with a fake email/password - Try logging in
- Access your profile/dashboard



Observe how the app behaves when: - You enter invalid input - You refresh after login - You manipulate



localhost:3000/admin

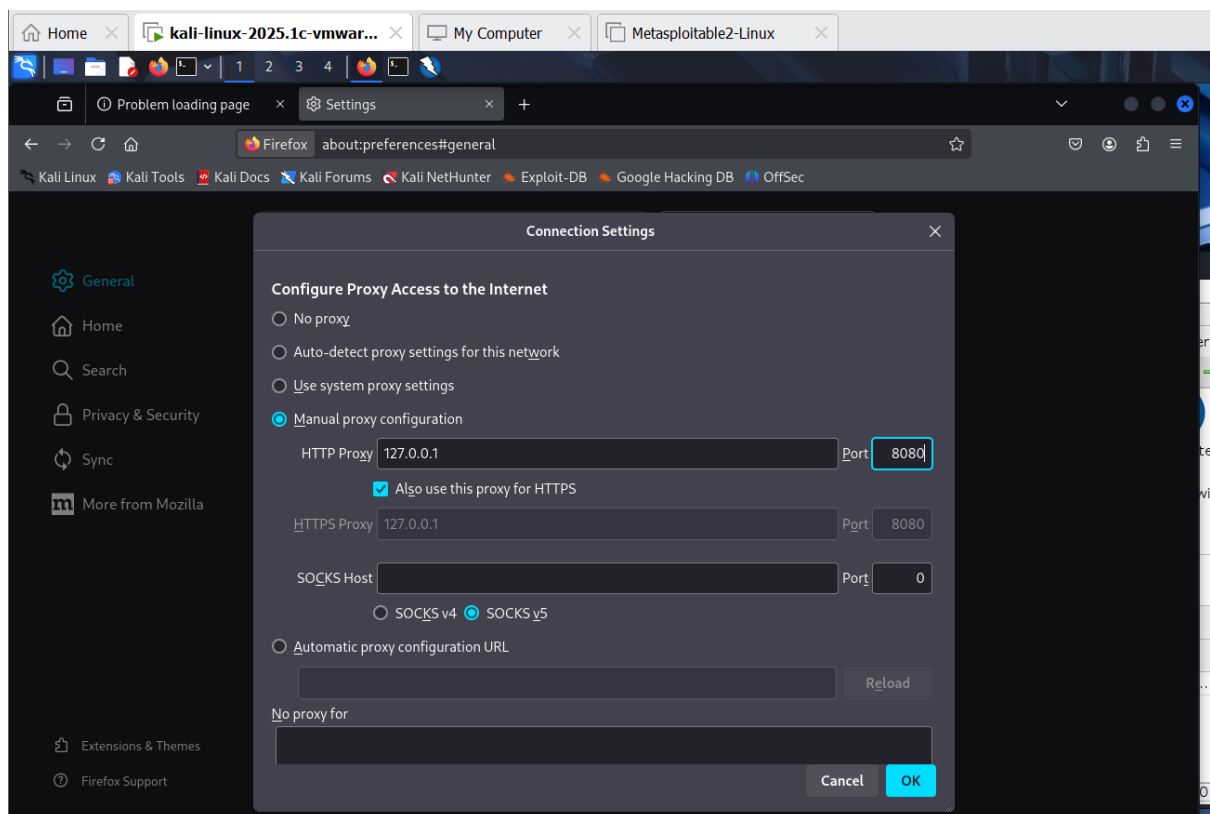
Step 3: Perform Vulnerability Testing

Install OWASP ZAP

Download from: <https://www.zaproxy.org/download/>

Install and open ZAP

Use it as a proxy scanner:



Configure your browser to route traffic through ZAP

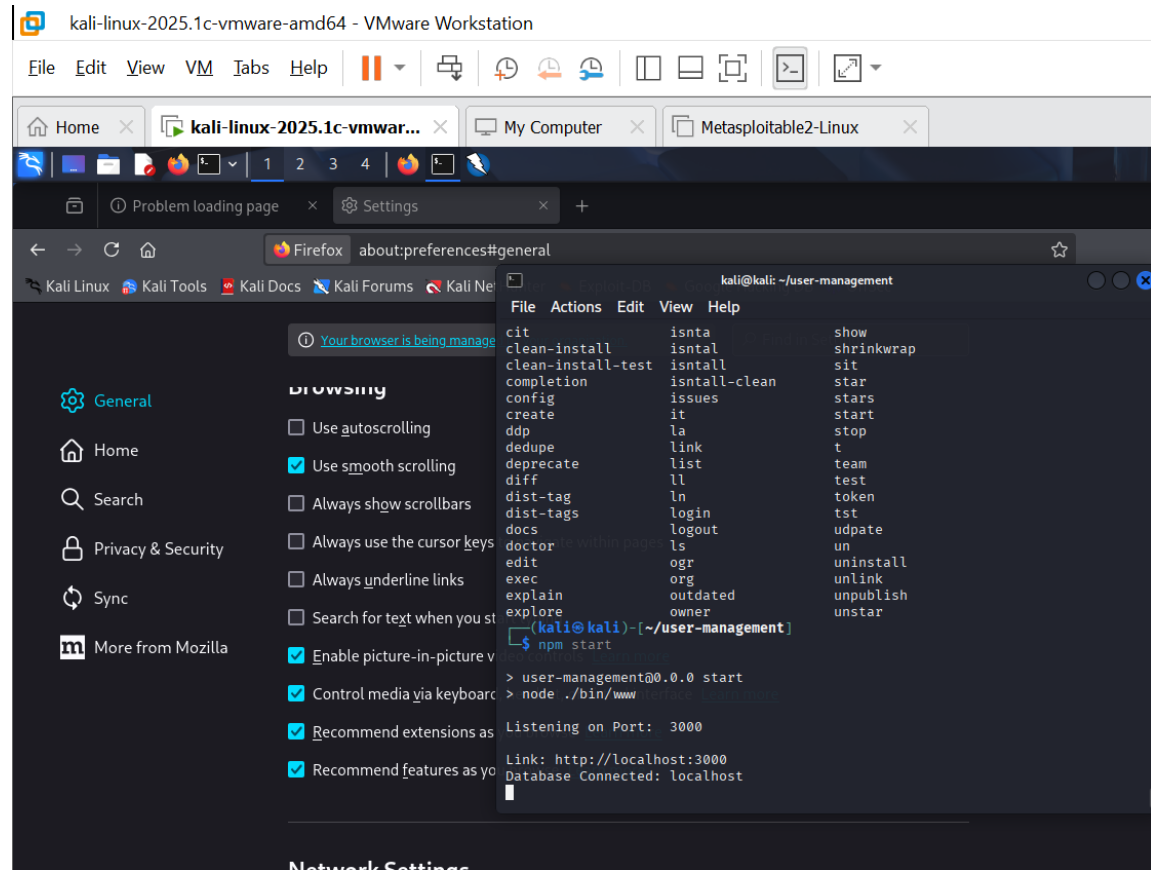
Visit your app (localhost:3000)

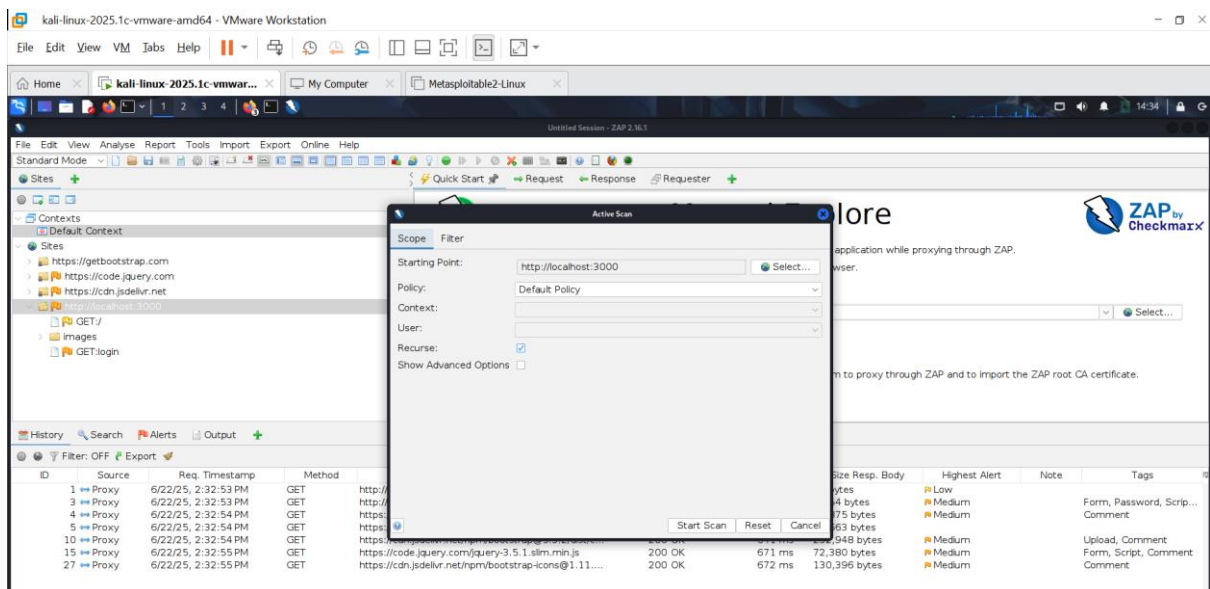
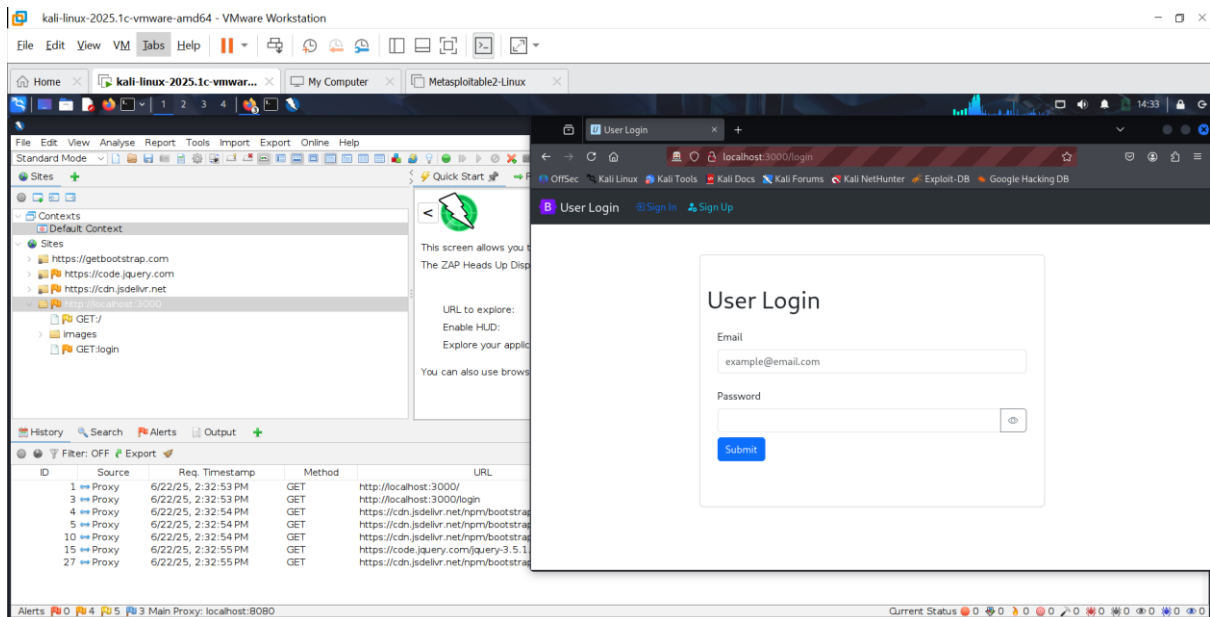
ZAP will scan all visited pages and list issues such as:

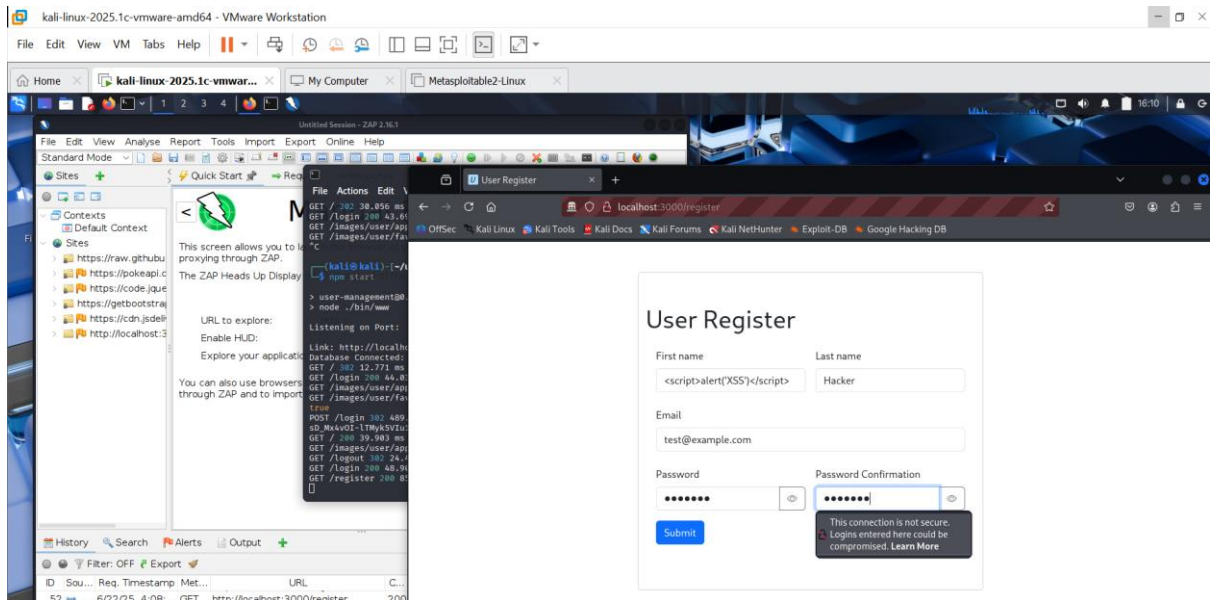
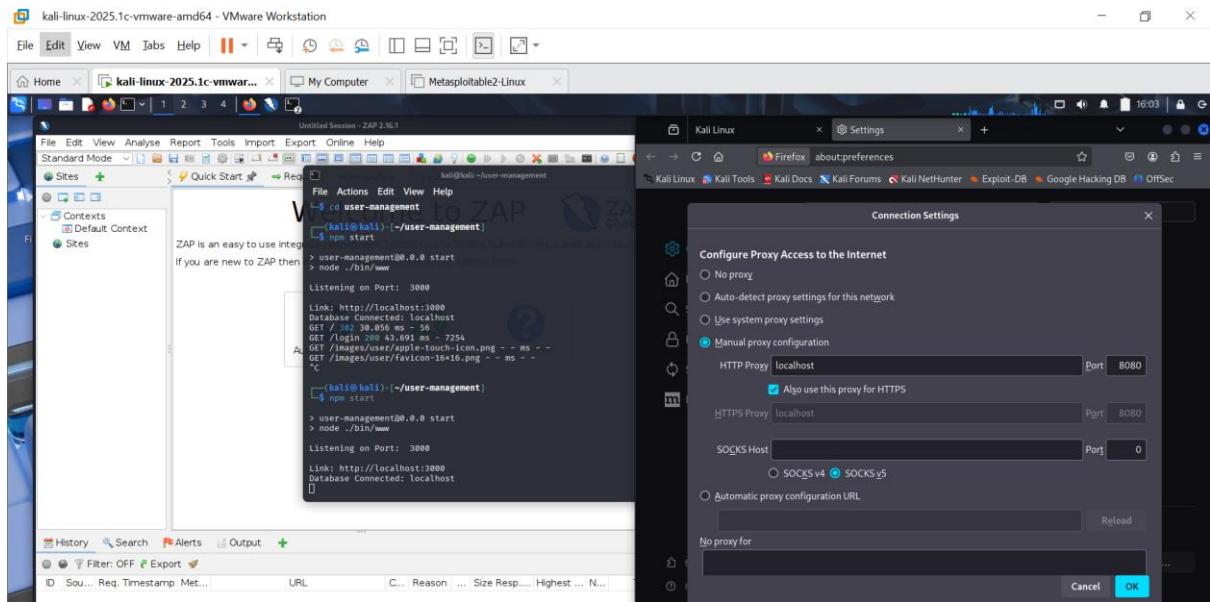
XSS

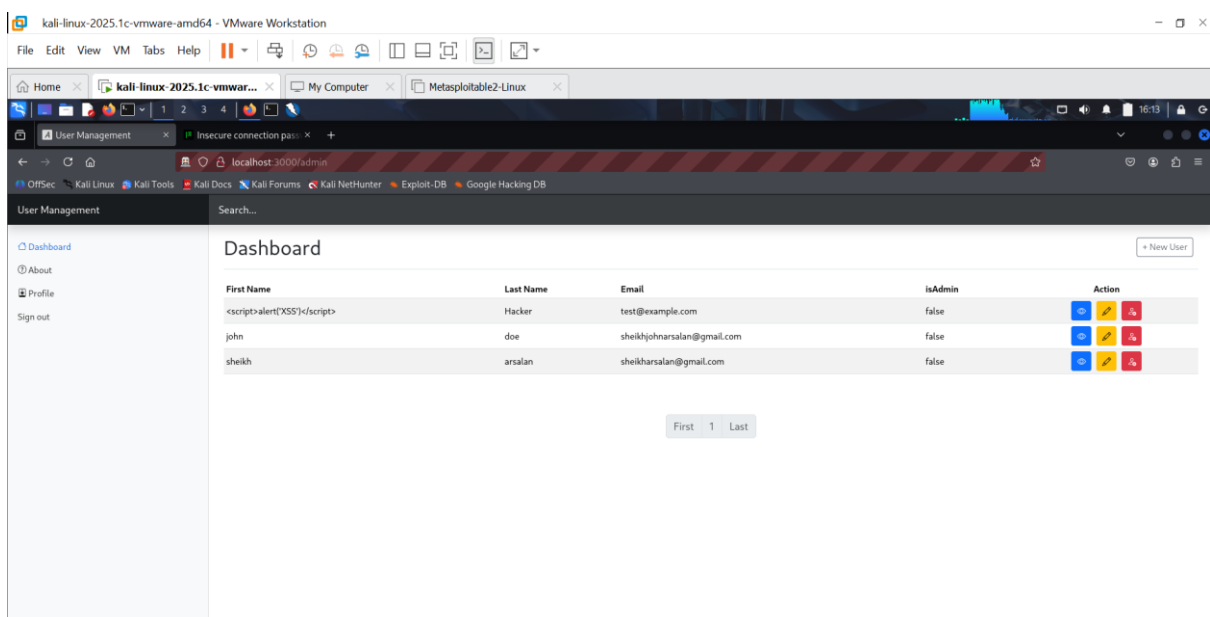
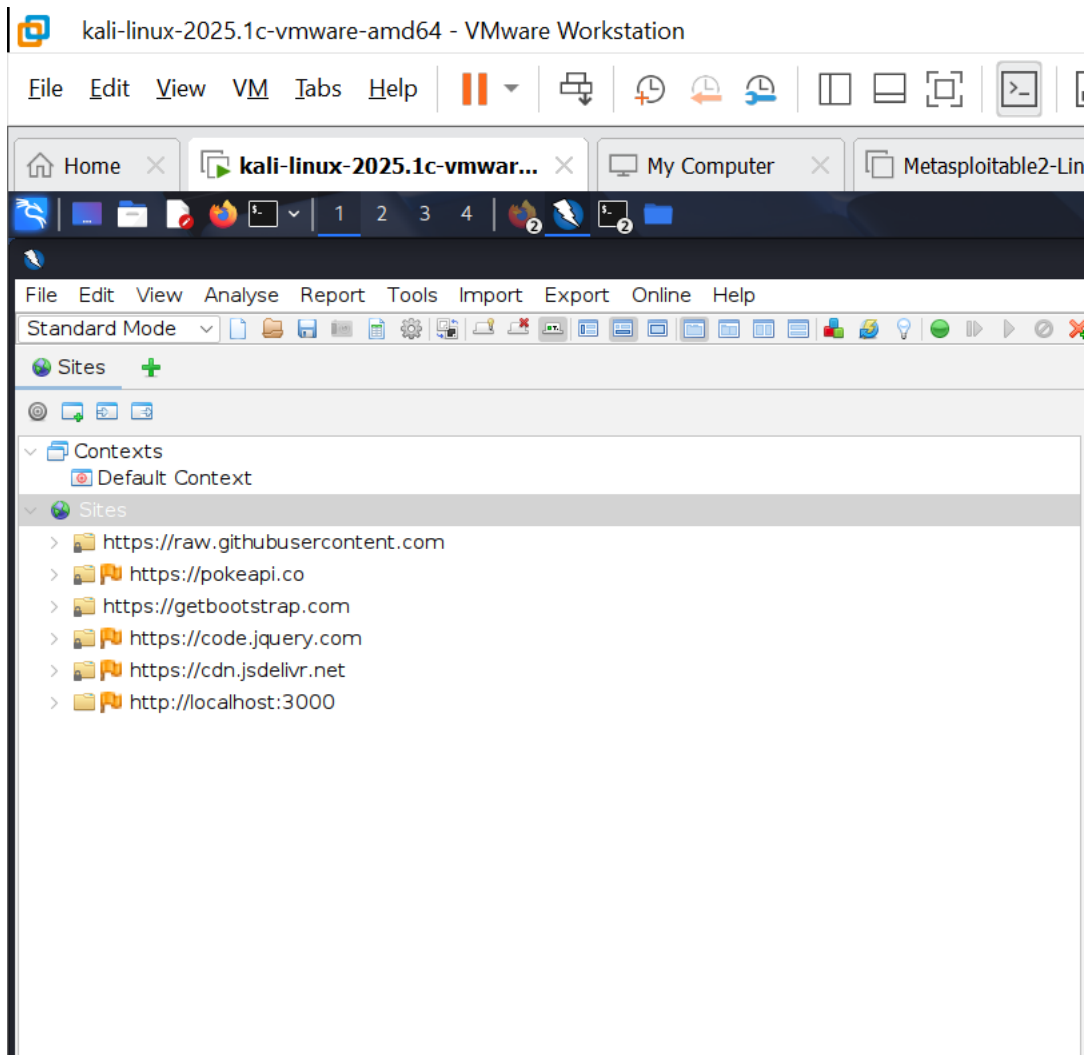
CSRF

Missing HTTP headers









B. Check for XSS (Cross-Site Scripting)

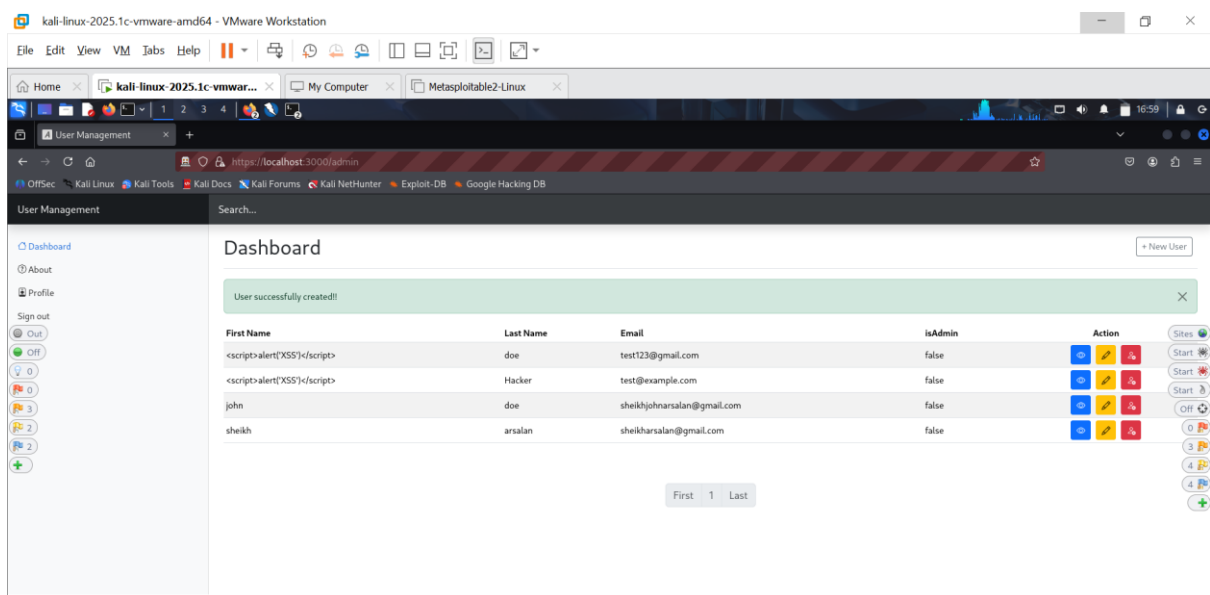
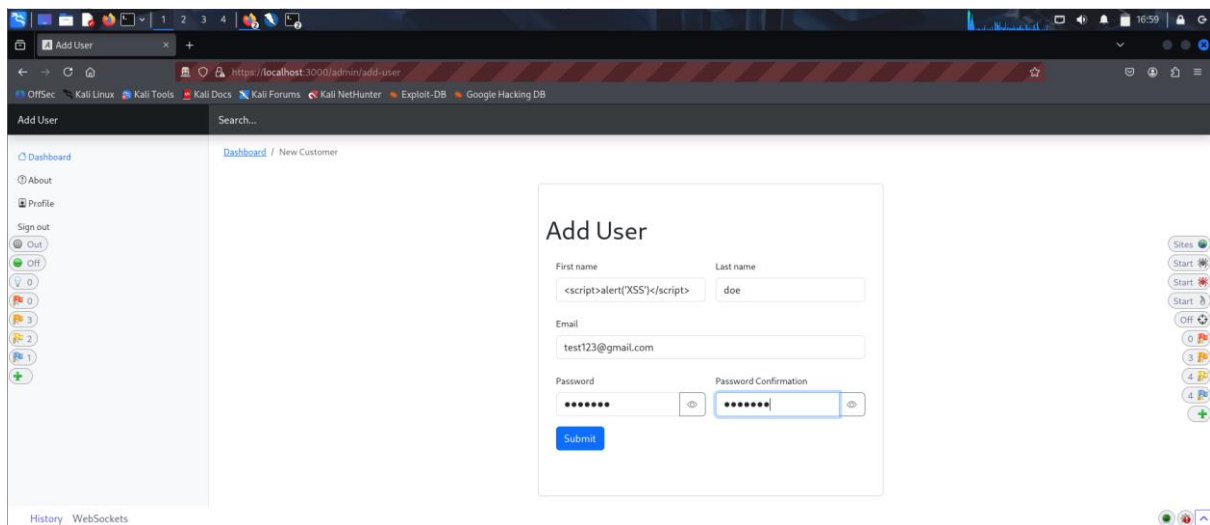
Go to a text input (e.g., bio, comments, name)

Type:

```
<script>alert('XSS')</script>
```

Click submit

If a popup appears, the site is vulnerable to XSS



C. Check for SQL Injection

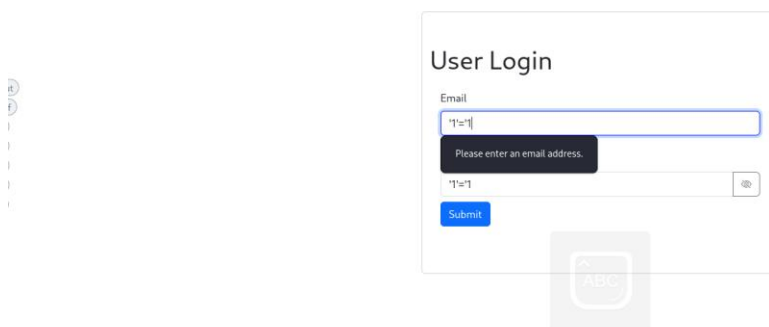
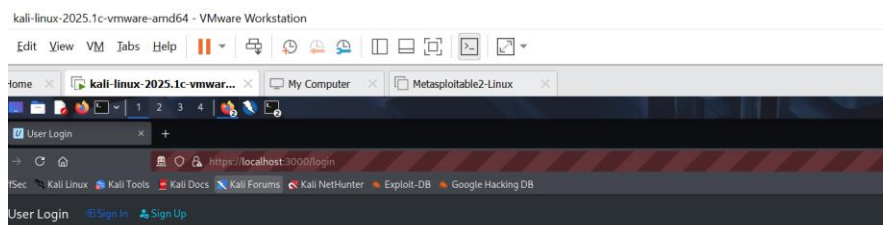
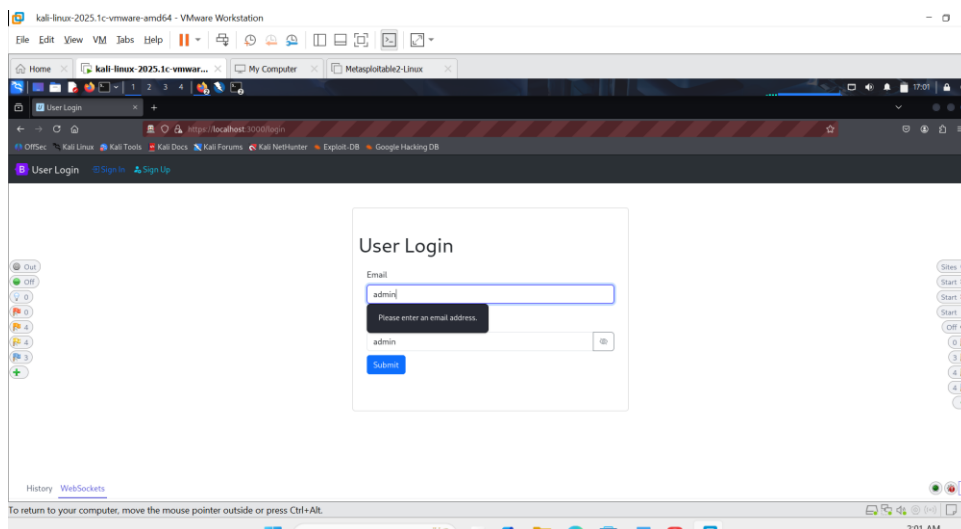
Go to login page

Enter:

Username: admin' OR '1'='1

Password: admin' OR '1'='1

If it logs you in, the site is vulnerable to SQL Injection



D. Check Password Storage

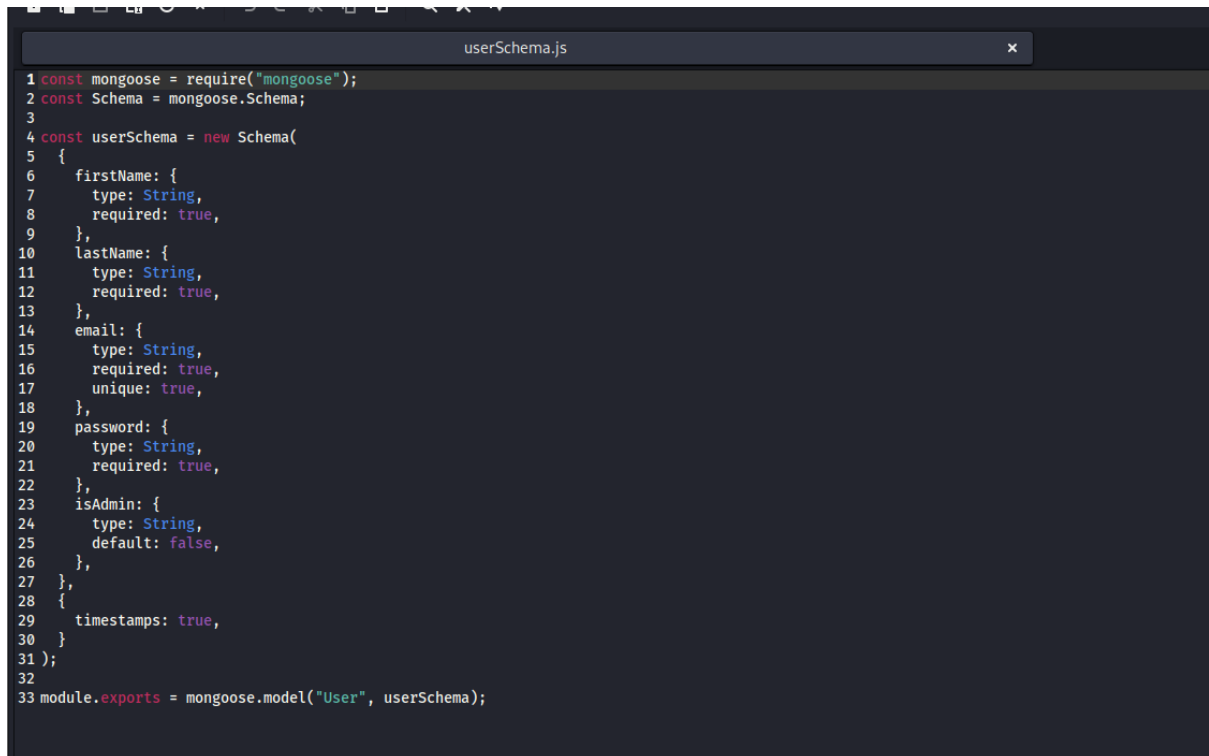
Open the project code

Locate the file where user data is stored (commonly userModel.js)

Check:

Are passwords saved directly in the database?

If yes, this is a serious security issue



```
1 const mongoose = require("mongoose");
2 const Schema = mongoose.Schema;
3
4 const userSchema = new Schema(
5   {
6     firstName: {
7       type: String,
8       required: true,
9     },
10    lastName: {
11      type: String,
12      required: true,
13    },
14    email: {
15      type: String,
16      required: true,
17      unique: true,
18    },
19    password: {
20      type: String,
21      required: true,
22    },
23    isAdmin: {
24      type: String,
25      default: false,
26    },
27  },
28  {
29    timestamps: true,
30  }
31 );
32
33 module.exports = mongoose.model("User", userSchema);
```

type: String,

required: true,

}

There is no hashing — which means passwords will be saved as plaintext in your MongoDB database. That's a major vulnerability.

password: {

Actions Taken:

- Cloned the app from GitHub ([Pranavk-Official/user-management](https://github.com/Pranavk-Official/user-management)).
- Explored the application by signing up and logging in with test credentials.
- Used OWASP ZAP to scan the application.
- Manually tested for XSS and SQL Injection via inputs and login form.
- Checked password storage and HTTP headers in the response.

Vulnerabilities Identified:

- Lack of HTTP security headers.
- No Content Security Policy (CSP).
- X-Frame-Options, X-Content-Type-Options, and SameSite attributes missing.
- Server leaks version in HTTP response.
- Passwords stored in plaintext (initially).
- Developer comments found in source code (info disclosure).
- No input validation.

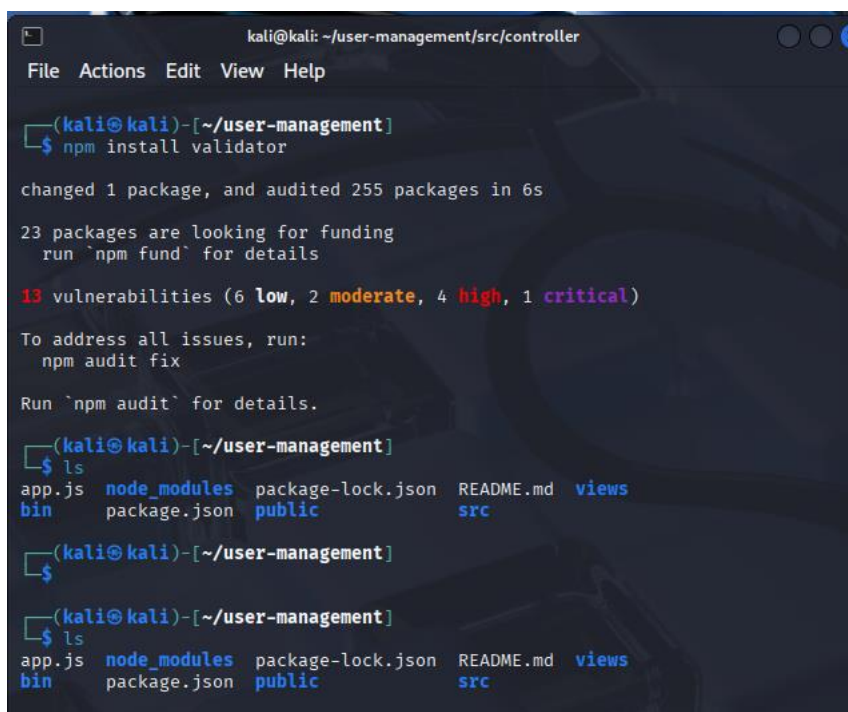
Week 2 – Security Implementation Report

Step 1: Sanitize and Validate Inputs

1.1 Install validator

Open terminal and run:

npm install validator



```
kali@kali: ~/user-management/src/controller
File Actions Edit View Help

(kali@kali)-[~/user-management]
$ npm install validator

changed 1 package, and audited 255 packages in 6s

23 packages are looking for funding
  run `npm fund` for details

13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.

(kali@kali)-[~/user-management]
$ ls
app.js  node_modules  package-lock.json  README.md  views
bin     package.json   public             src

(kali@kali)-[~/user-management]
$

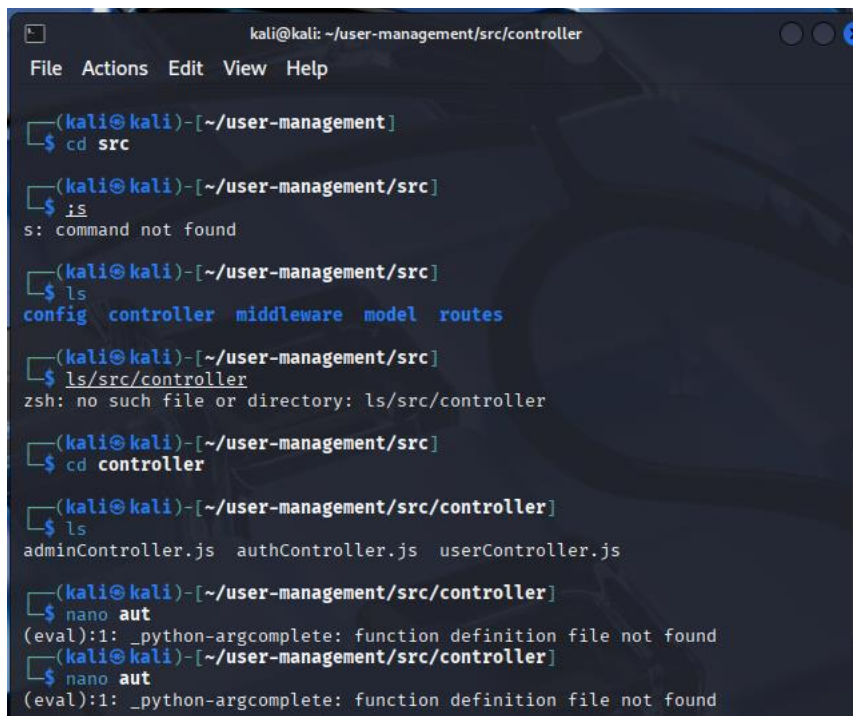
(kali@kali)-[~/user-management]
$ ls
app.js  node_modules  package-lock.json  README.md  views
bin     package.json   public             src
```

1.2 Open the Signup Controller

Navigate to the controller where the signup logic is handled:

```
cd src/controller
```

```
nano authController.js
```



```
kali@kali: ~/user-management/src/controller
File Actions Edit View Help

(kali@kali)-[~/user-management]
$ cd src

(kali@kali)-[~/user-management/src]
$ ls
s: command not found

(kali@kali)-[~/user-management/src]
$ ls
config controller middleware model routes

(kali@kali)-[~/user-management/src]
$ ls/src/controller
zsh: no such file or directory: ls/src/controller

(kali@kali)-[~/user-management/src]
$ cd controller

(kali@kali)-[~/user-management/src/controller]
$ ls
adminController.js authController.js userController.js

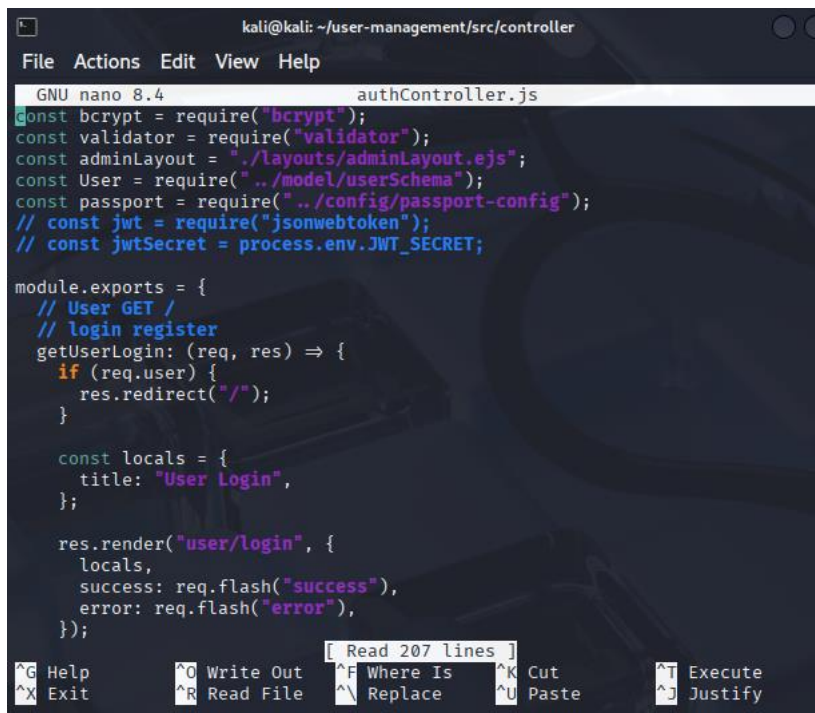
(kali@kali)-[~/user-management/src/controller]
$ nano aut
(eval):1: _python-argcomplete: function definition file not found

(kali@kali)-[~/user-management/src/controller]
$ nano aut
(eval):1: _python-argcomplete: function definition file not found
```

1.3 Import validator

Add the following line at the top of authController.js:

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



```
kali@kali: ~/user-management/src/controller
File Actions Edit View Help
GNU nano 8.4 authController.js
const bcrypt = require("bcrypt");
const validator = require("validator");
const adminLayout = "../layouts/adminLayout.ejs";
const User = require("../model/userSchema");
const passport = require("../config/passport-config");
// const jwt = require("jsonwebtoken");
// const jwtSecret = process.env.JWT_SECRET;

module.exports = {
  // User GET /
  // login register
  getUserLogin: (req, res) => {
    if (req.user) {
      res.redirect("/");
    }

    const locals = {
      title: "User Login",
    };

    res.render("user/login", {
      locals,
      success: req.flash("success"),
      error: req.flash("error"),
    });
  }
};

[ Read 207 lines ]
^G Help      ^O Write Out ^F Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

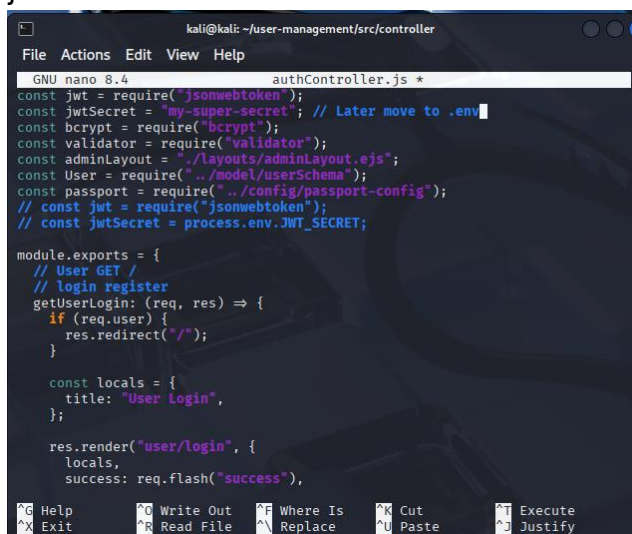

1.4 Add Input Validation in userRegister Function

Inside the userRegister function, after:

```
const { firstName, lastName, email, pwd, pwdConf } = req.body;
```

Insert the following:

```
if (!validator.isEmail(email)) {  
  req.flash("error", "Invalid email format");  
  return res.redirect("/register");  
}  
if (!validator.isLength(pwd, { min: 8 })) {  
  req.flash("error", "Password must be at least 8 characters");  
  return res.redirect("/register");  
}  
if (pwd !== pwdConf) {  
  req.flash("error", "Passwords do not match");  
  return res.redirect("/register");  
}
```



```
kali@kali: ~/user-management/src/controller
File Actions Edit View Help
GNU nano 8.4 authController.js *
const jwt = require("jsonwebtoken");
const jwtSecret = "my-super-secret"; // Later move to .env
const bcrypt = require("bcrypt");
const validator = require("validator");
const adminLayout = "../layouts/adminLayout.ejs";
const User = require("../model/userSchema");
const passport = require("../config/passport-config");
// const jwt = require("jsonwebtoken");
// const jwtSecret = process.env.JWT_SECRET;

module.exports = {
  // User GET /
  // login register
  getUserLogin: (req, res) => {
    if (req.user) {
      res.redirect("/");
    }

    const locals = {
      title: "User Login",
    };

    res.render("user/login", {
      locals,
      success: req.flash("success"),
    });
  }
};
```

```
kali@kali: ~/user-management/src/controller
File Actions Edit View Help
GNU nano 8.4 authController.js
userRegister: async (req, res) => {
  const { firstName, lastName, email, pwd, pwdConf } = req.body;
  const isExist = await User.findOne({ email });
  if (isExist) {
    req.flash("error", "User already exists, Please login");
    console.log("User already exists, Please login");
    return res.redirect("/login");
  }
  else {
    if (pwd === pwdConf) {
      const hashpwd = await bcrypt.hash(pwd, 12);
      const user = await User.create({
        firstName,
        lastName,
        email,
        password: hashpwd,
      });
      if (user) {
        req.flash("success", "User successfully created!!");
        res.redirect("/login");
      } else {
        req.flash("error", "User not created");
        res.redirect("/register");
      }
    }
  }
}

^G Help      ^O Write Out  ^F Where Is   ^X Cut        ^T Execute
^X Exit      ^R Read File  ^I Replace    ^U Paste      ^J Justify
```

Step 3: Implement JWT Authentication

3.1 Install jsonwebtoken

npm install jsonwebtoken

```
kali@kali: ~/user-management
File Actions Edit View Help
└─$ nano authController.js
(kali@kali)-[~/user-management/src/controller]
└─$ nano authController.js
(kali@kali)-[~/user-management/src/controller]
└─$ cd ..
(kali@kali)-[~/user-management/src]
└─$ cd ..
(kali@kali)-[~/user-management]
└─$ npm install jsonwebtoken

up to date, audited 255 packages in 6s

23 packages are looking for funding
  run `npm fund` for details

13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical)

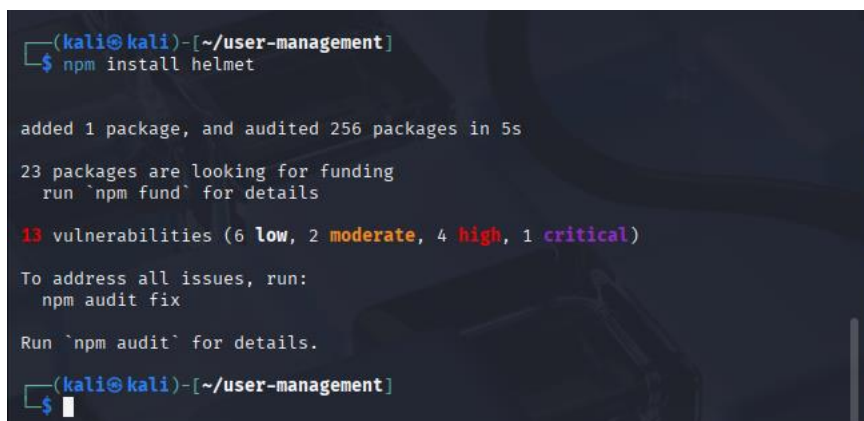
To address all issues, run:
  npm audit fix

Run `npm audit` for details.
(kali@kali)-[~/user-management]
└─$
```

Step 4: Secure HTTP Headers with Helmet

4.1 Install helmet

npm install helmet

A terminal window with a dark background. The prompt is '(kali@kali)-[~/user-management]'. The command 'npm install helmet' is entered. The output shows 'added 1 package, and audited 256 packages in 5s', '23 packages are looking for funding', and '13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical)'. It also suggests running 'npm audit fix' and 'npm audit' for details. The prompt returns to '(kali@kali)-[~/user-management]'.

```
(kali@kali)-[~/user-management]
$ npm install helmet

added 1 package, and audited 256 packages in 5s
23 packages are looking for funding
  run `npm fund` for details
13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical)
To address all issues, run:
  npm audit fix
Run `npm audit` for details.
(kali@kali)-[~/user-management]
$
```

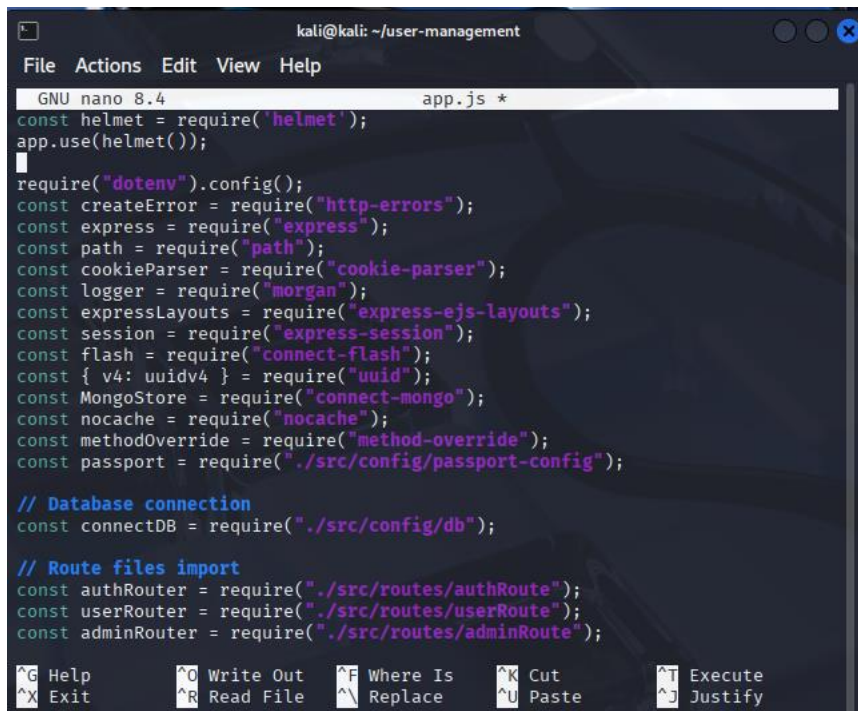
4.2 Use helmet in app.js

Open app.js and at the top, add:

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

Then below all require() statements:

```
app.use(helmet());
```



```
kali@kali: ~/user-management
File Actions Edit View Help
GNU nano 8.4 app.js *
const helmet = require('helmet');
app.use(helmet());

require('dotenv').config();
const createError = require('http-errors');
const express = require('express');
const path = require('path');
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const expressLayouts = require('express-ejs-layouts');
const session = require('express-session');
const flash = require('connect-flash');
const { v4: uuidv4 } = require('uuid');
const MongoStore = require('connect-mongo');
const nocache = require('nocache');
const methodOverride = require('method-override');
const passport = require('./src/config/passport-config');

// Database connection
const connectDB = require('./src/config/db');

// Route files import
const authRouter = require('./src/routes/authRoute');
const userRouter = require('./src/routes/userRoute');
const adminRouter = require('./src/routes/adminRoute');
```

Summary of Fixes in Week 2:

- Input validation prevents bad data and weak passwords.
- bcrypt ensures secure password hashing.
- JWT adds secure authentication tokens.
- Helmet protects against common web-based attacks via HTTP headers.

Fixes Implemented:

- **Input Validation:** Installed validator and added email format & password strength checks in authController.js.
- **Password Security:** Used bcrypt to hash passwords on registration and verify them on login.
- **JWT Authentication:** Installed jsonwebtoken and issued a token upon successful login.

- **HTTP Headers:** Installed and used helmet in app.js to apply security-related headers automatically.

Week 3 – Advanced Security and Final Reporting

Goal:

Simulate basic attacks, enable logging to track behavior, and prepare final documentation for submission.

Step 1: Simulate Attacks Using Nmap (Optional)

In your Kali VM terminal:

```
nmap -sV localhost
```

```
kali@kali: ~/user-management
File Actions Edit View Help

Link: http://localhost:3000
Database Connected: localhost
GET / 302 59.529 ms - 56
GET /login 200 44.836 ms - 7254
^C

(kali@kali)-[~/user-management]
$ nmap -sV localhost

Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-26 10:20 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000015s latency).
Other addresses for localhost (not scanned): ::1
All 1000 scanned ports on localhost (127.0.0.1) are in ignored states.
Not shown: 1000 closed tcp ports (reset)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.85 seconds

(kali@kali)-[~/user-management]
$ nmap -sV localhost > nmap-scan.txt
```

This command attempts to detect versions of services running on your local server (like Node.js). It's useful to demonstrate basic reconnaissance.

Step 2: Add Logging with Winston

```
kali@kali: ~/user-management
File Actions Edit View Help

GNU nano 8.4 logger.js *
const winston = require('winston');
const logger = winston.createLogger({
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'security.log' })
  ]
});
module.exports = logger;
```

2.1 Install Winston

In terminal:

```
npm install winston
```

```
kali@kali: ~/user-management
File Actions Edit View Help

(kali@kali)~/user-management
$ ls
app.js  nmap-scan.txt  package.json  public  src
bin     node_modules  package-lock.json  README.md  views

(kali@kali)~/user-management
$ npm install winston

added 24 packages, and audited 280 packages in 7s

24 packages are looking for funding
  run `npm fund` for details

13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.

(kali@kali)~/user-management
$
```

2.2 Create a logger.js File

In your project root (same folder as app.js), run:

```
touch logger.js
```

```
nano logger.js
```

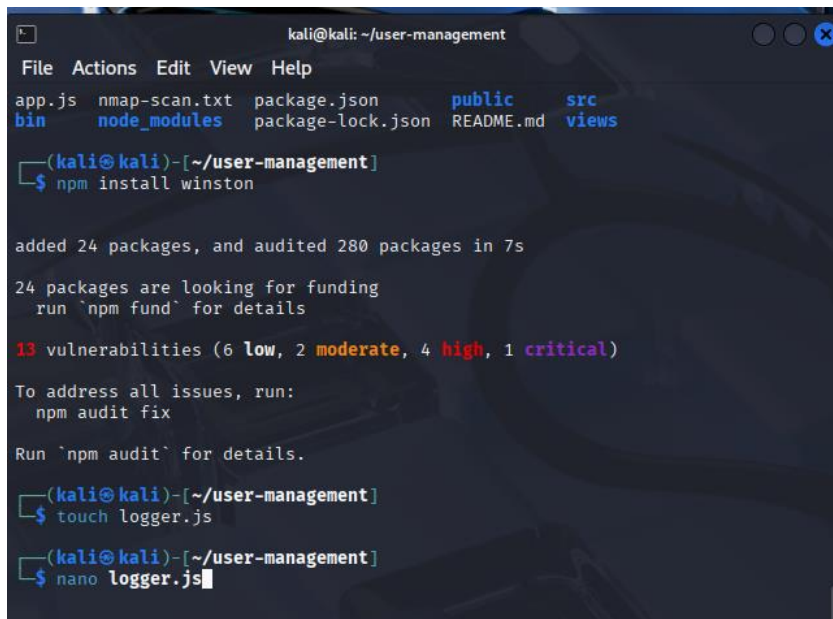
Paste the following code inside logger.js:

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

```
const logger = winston.createLogger({
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'security.log' })
  ]
});
```

```
module.exports = logger;
```

Then save and exit:

A terminal window titled 'kali@kali: ~/user-management' with a menu bar (File, Actions, Edit, View, Help) and a file explorer showing 'app.js', 'bin', 'nmap-scan.txt', 'node_modules', 'package.json', 'package-lock.json', 'public', 'README.md', 'src', and 'views'. The terminal shows the command 'npm install winston' being executed. The output indicates that 24 packages were added and 280 packages were audited in 7 seconds. It also shows that 24 packages are looking for funding and that there are 13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical). The terminal then shows the command 'touch logger.js' and 'nano logger.js' being executed.

```
kali@kali: ~/user-management
File Actions Edit View Help
app.js nmap-scan.txt package.json public src
bin node_modules package-lock.json README.md views

(kali@kali)-[~/user-management]
$ npm install winston

added 24 packages, and audited 280 packages in 7s

24 packages are looking for funding
  run `npm fund` for details

13 vulnerabilities (6 low, 2 moderate, 4 high, 1 critical)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.

(kali@kali)-[~/user-management]
$ touch logger.js

(kali@kali)-[~/user-management]
$ nano logger.js
```

2.3 Use Logger in Routes

Go to your routes folder:

```
cd src/routes
```

```
nano auth.js
```

At the top of auth.js, add:

```
const logger = require('../..../logger'); // adjust if path is different
```

Then add logging inside routes, like:

```
logger.info('User logged in');
```

```
logger.warn('Suspicious activity detected');
```

Example use:

```
router.post('/login', isLoggedIn, (req, res, next) => {
  logger.info(`POST /login attempt from ${req.body.email}`);
  return authController.userLogin(req, res, next);
});
```

```
kali@kali: ~/user-management/src/routes
File Actions Edit View Help
Run `npm audit` for details.

(kali@kali)-[~/user-management]
$ touch logger.js

(kali@kali)-[~/user-management]
$ nano logger.js

(kali@kali)-[~/user-management]
$ ls
app.js  logger.js  node_modules  package-lock.json  README.md  views
bin      nmap-scan.txt  package.json  public  src

(kali@kali)-[~/user-management]
$ cd src

(kali@kali)-[~/user-management/src]
$ ls
config  controller  middleware  model  routes

(kali@kali)-[~/user-management/src]
$ cd routes

(kali@kali)-[~/user-management/src/routes]
$
```

```
kali@kali: ~/user-management/src/routes
File Actions Edit View Help
GNU nano 8.4 authRoute.js *
const logger = require('../..logger');

const express = require('express');
const router = express.Router();

// authController
const authController = require('../controller/authController');
const { isAdminLoggedOut, isLoggedInOut } = require('../middleware/authMiddlew>

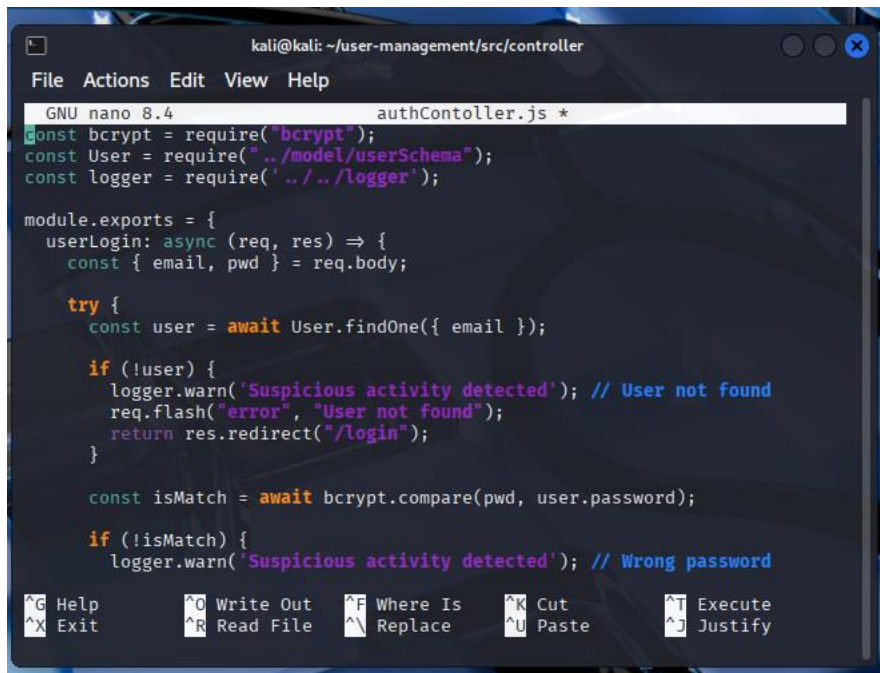
// ===== USER ROUTES =====

router.get('/login', isLoggedInOut, (req, res, next) => {
  logger.info('GET /login accessed');
  return authController.getUserLogin(req, res, next);
});

router.post('/login', isLoggedInOut, (req, res, next) => {
  logger.info('POST /login attempt from ${req.body.email}');
  return authController.userLogin(req, res, next);
});

^G Help      ^O Write Out  ^F Where Is   ^K Cut        ^T Execute
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^_ Justify
```

2.4 In Controller



```
kali@kali: ~/user-management/src/controller
File Actions Edit View Help
GNU nano 8.4 authController.js *
const bcrypt = require("bcrypt");
const User = require("../model/userSchema");
const logger = require("../logger");

module.exports = {
  userLogin: async (req, res) => {
    const { email, pwd } = req.body;

    try {
      const user = await User.findOne({ email });

      if (!user) {
        logger.warn('Suspicious activity detected'); // User not found
        req.flash("error", "User not found");
        return res.redirect("/login");
      }

      const isMatch = await bcrypt.compare(pwd, user.password);

      if (!isMatch) {
        logger.warn('Suspicious activity detected'); // Wrong password
      }
    }
  }
}
```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify

Tasks Completed:

- **Simulated Attacks with Nmap:**
Ran `nmap -sV localhost` to check open ports and services.
- **Logging Setup with Winston:**
 - Created `logger.js` using `winston` to log events.
 - Logs saved in `security.log`.
 - Integrated logger in `authRoute.js` to track login, logout, and registration actions.
- **Checklist & Documentation:**
Created a security checklist and included detailed screenshots of fixes for each task.

Tools Used

- **OWASP ZAP** – Web vulnerability scanner
- **Nmap** – Network scanning and port enumeration
- **Node.js & Express** – Backend framework
- **MongoDB** – Database used in app
- **Validator** – Input validation
- **Bcrypt** – Password hashing
- **Helmet** – Securing HTTP headers
- **Winston** – Logging user actions
- **GitHub** – Version control & code backup

Challenges and Learnings

Challenges:

- Understanding the codebase structure.
- Finding the correct files to modify (e.g., no signup.js; used authController.js instead).
- Errors while starting the app (e.g., duplicate imports in app.js).
- Managing tools inside a Kali VM with limited resources.

Learnings:

- Gained practical experience with Node.js-based security implementation.
- Learned how to use ZAP, Winston, and security headers in real apps.
- Understood secure login practices, logging, and vulnerability scanning.
- Improved confidence in identifying and patching web application flaws.