**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:

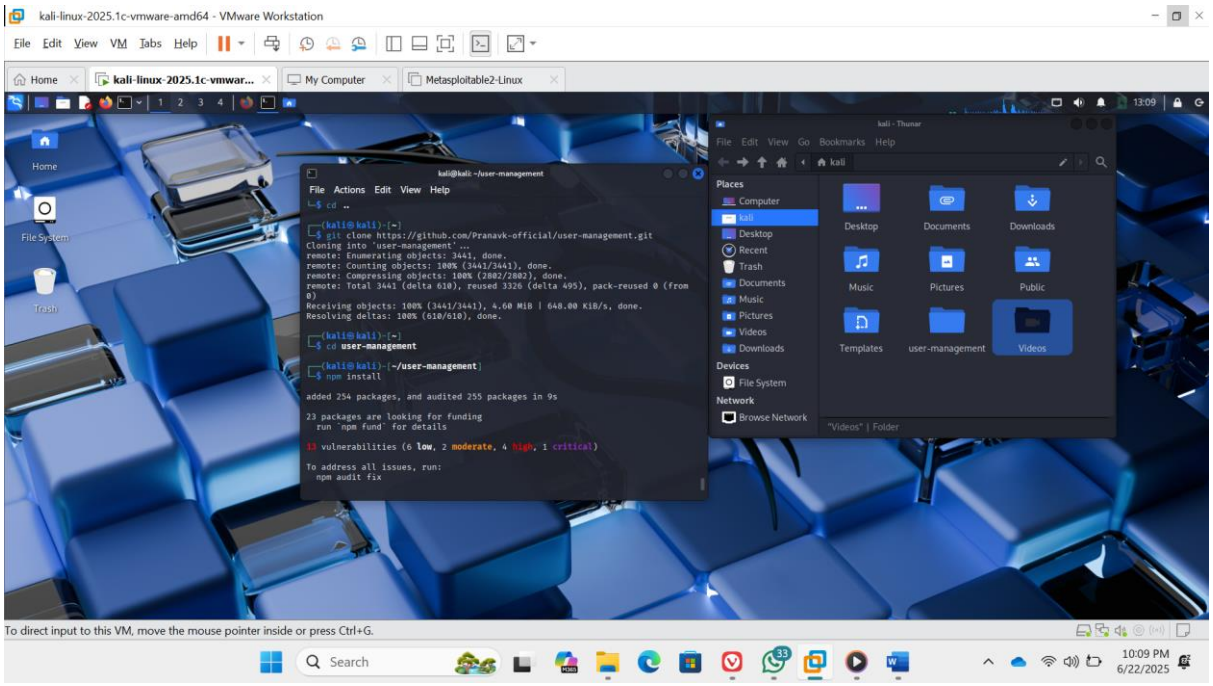
git clone <project-link>

Go to project folder:

cd <project-folder>

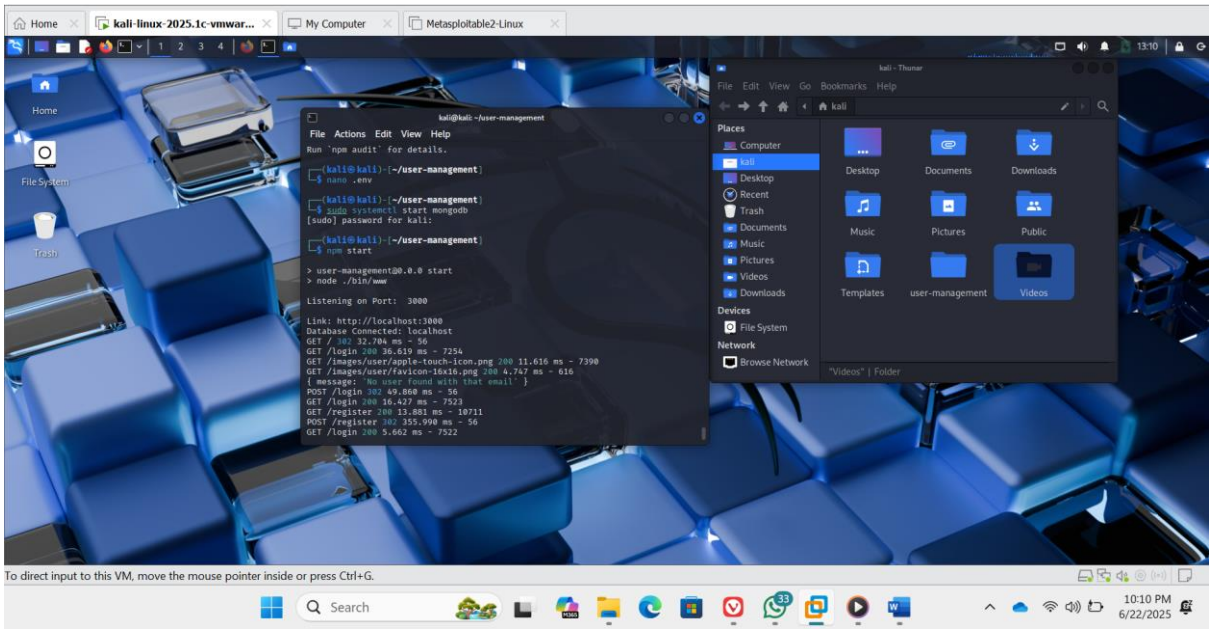
Install required packages:

npm install



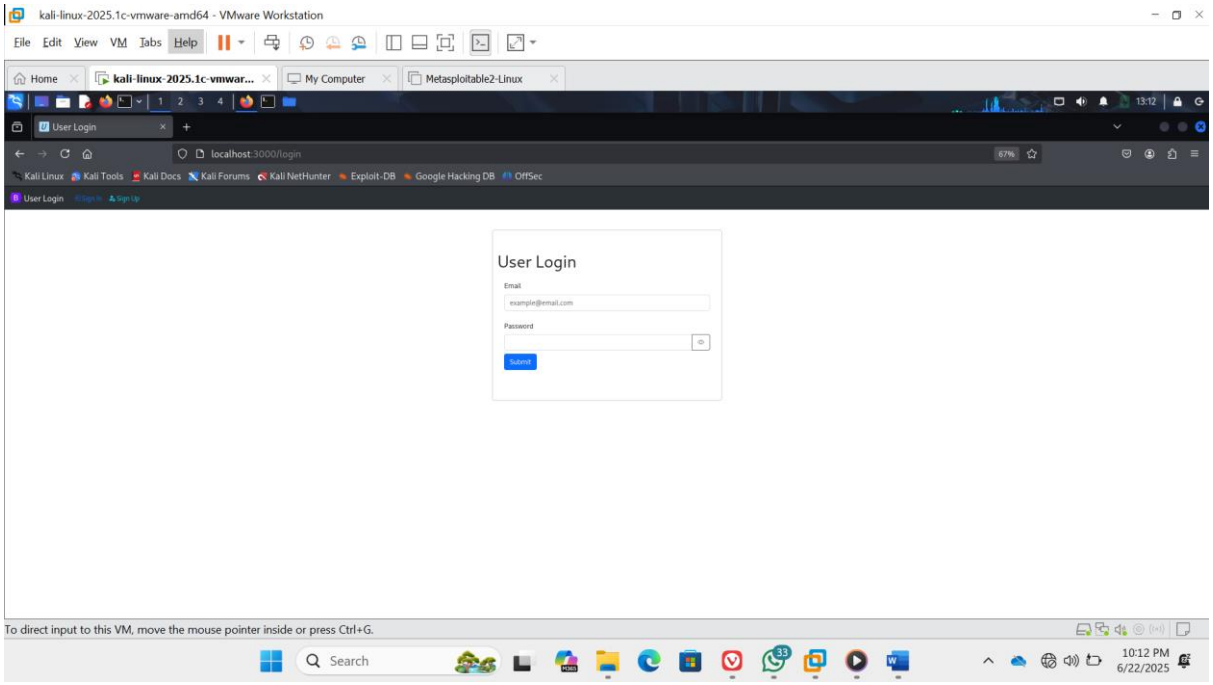
Start the application:

npm start




Open browser and visit:

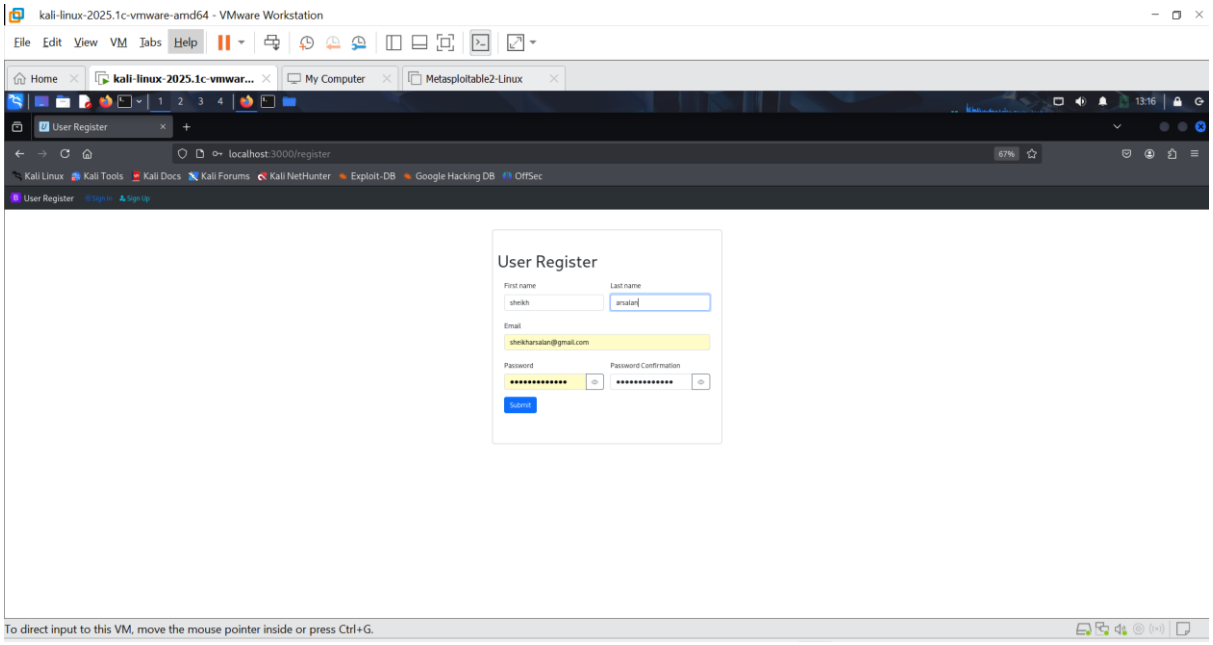
http://localhost:3000



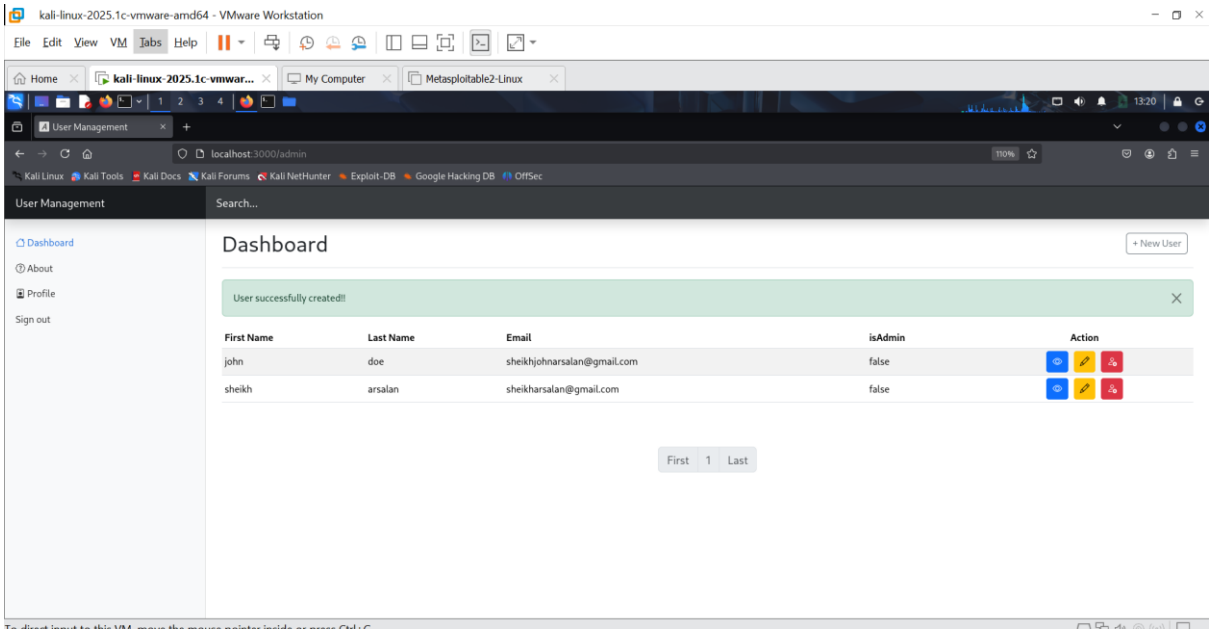
Step 2: Explore the Application

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

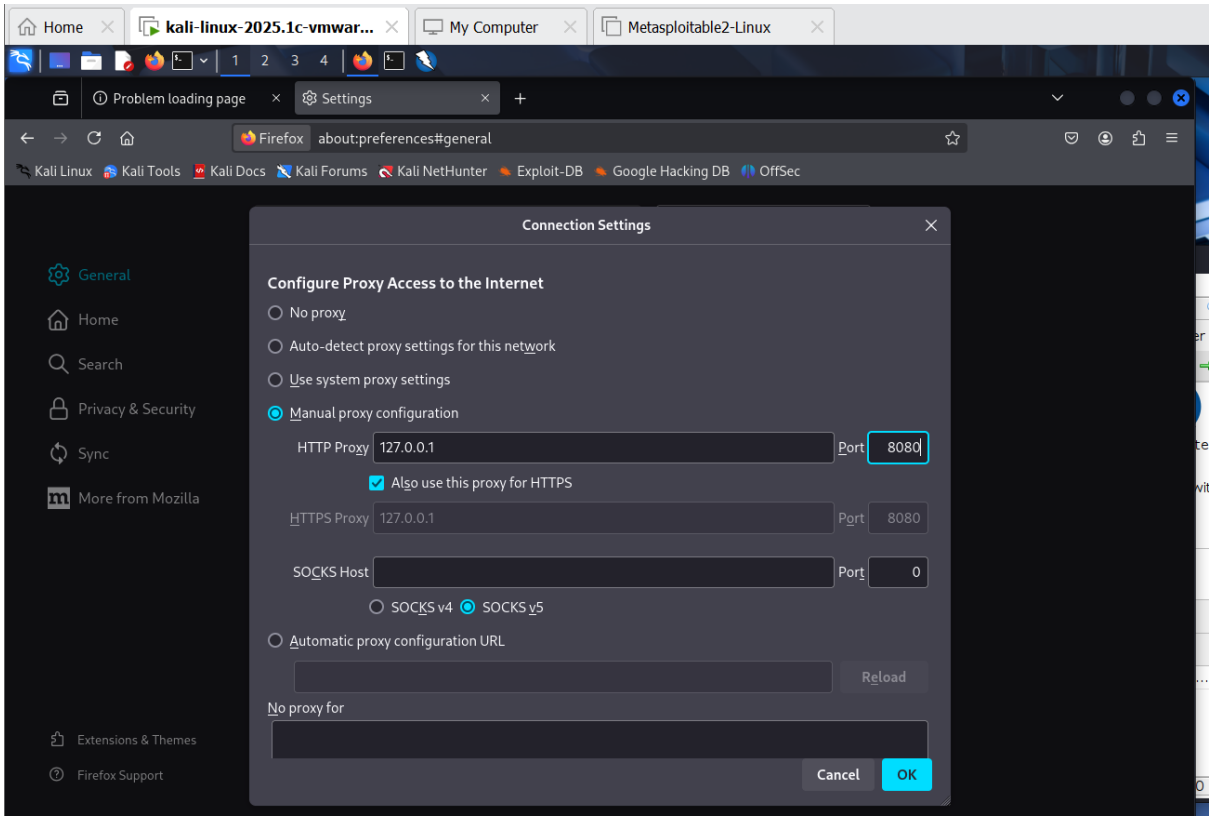
Step 3: Perform Vulnerability Testing

Install OWASP ZAP

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

Install and open ZAP

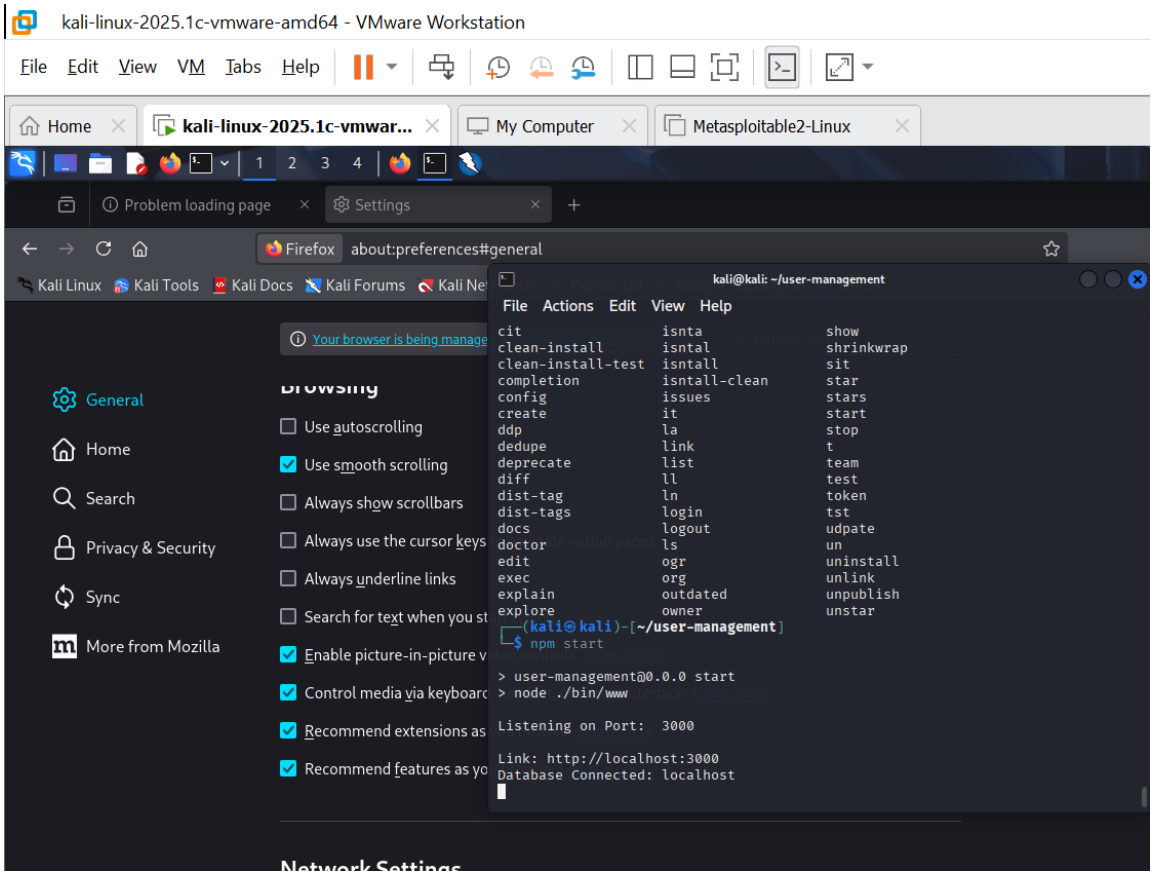
Use it as a proxy scanner:

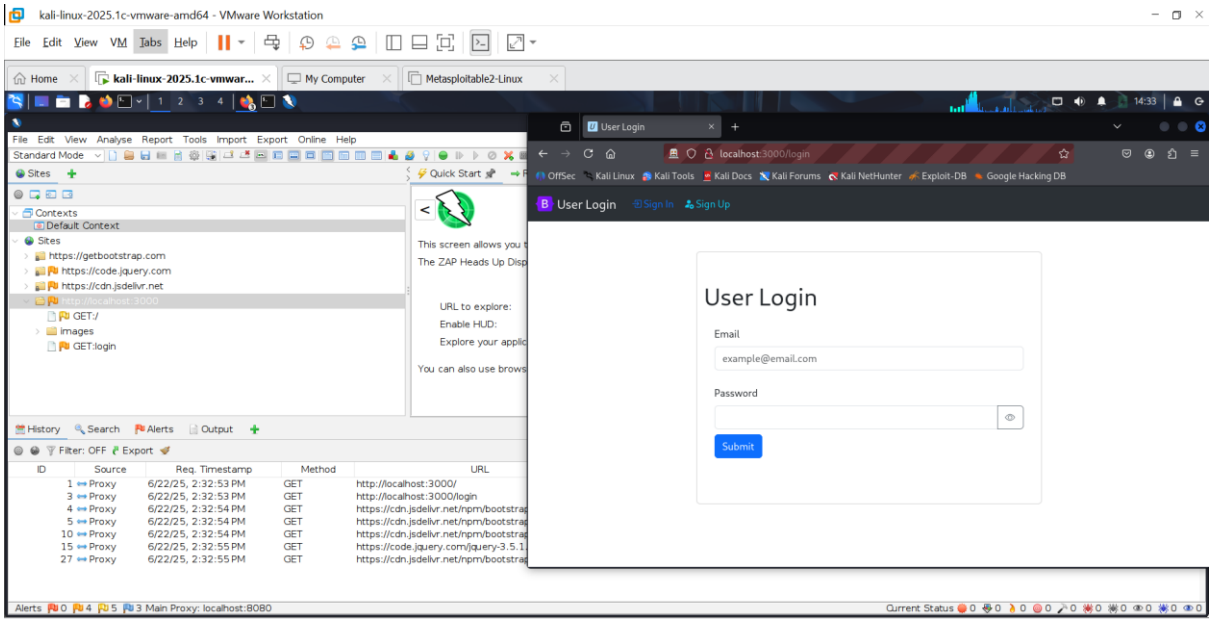


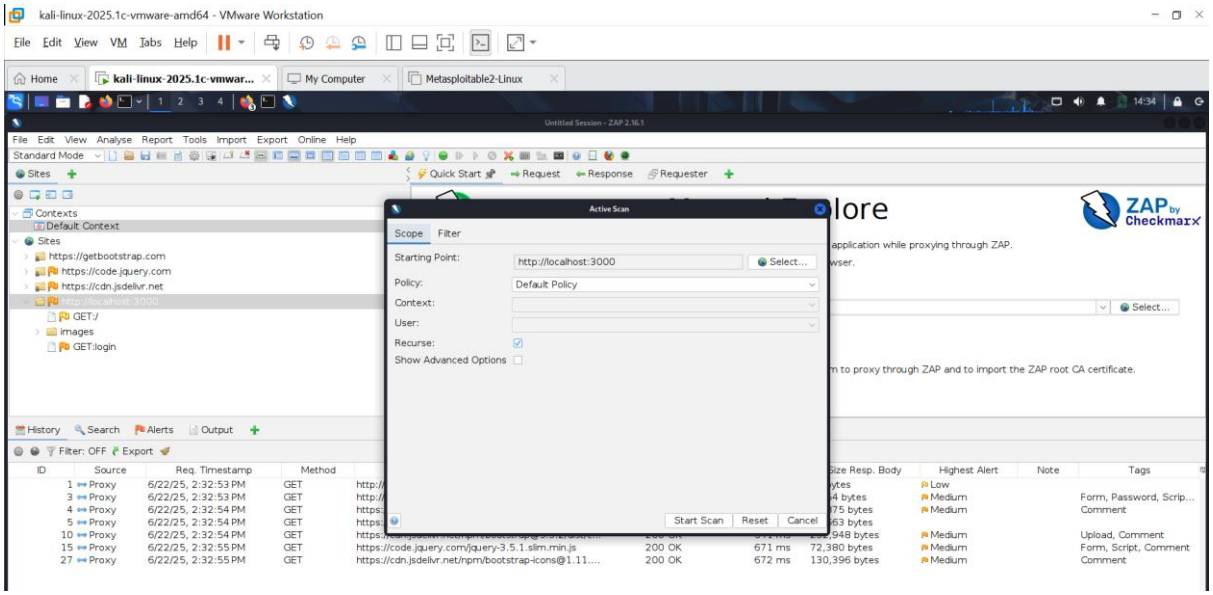
Configure your browser to route traffic through ZAP

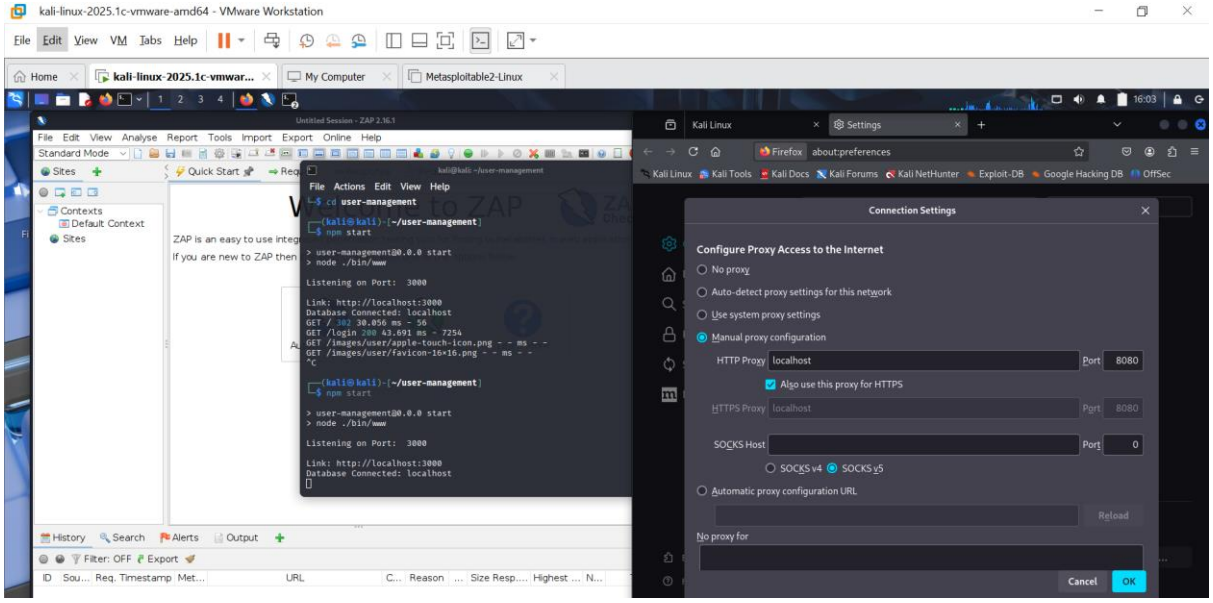
Visit your app ( localhost:3000 )

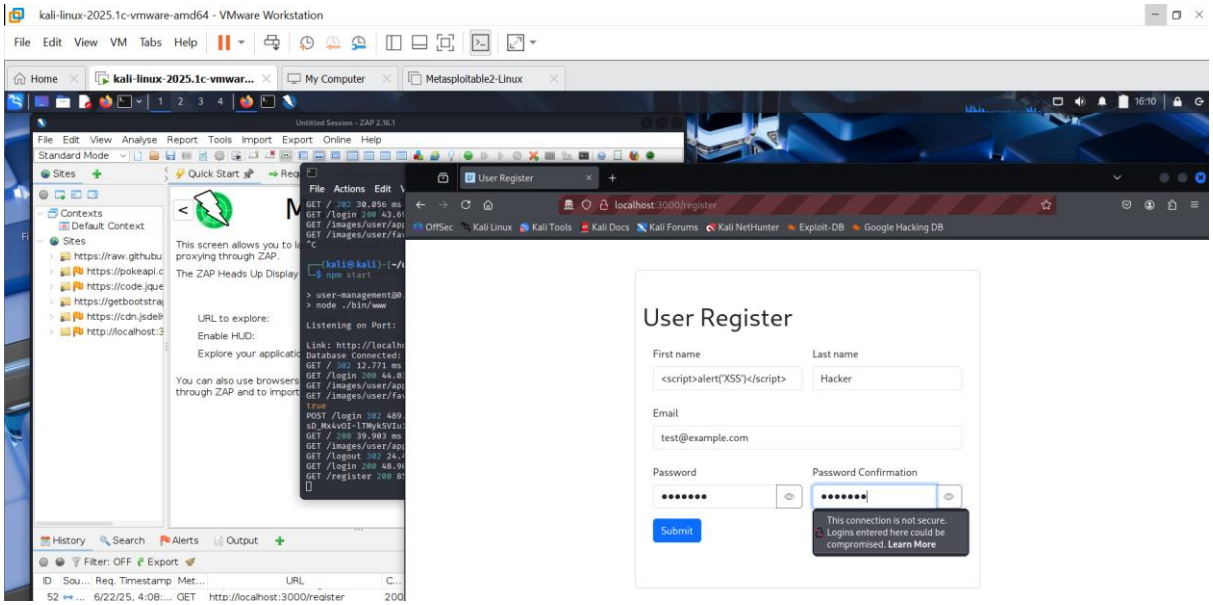
ZAP will scan all visited pages and list issues such as:
XSS
CSRF
Missing HTTP headers

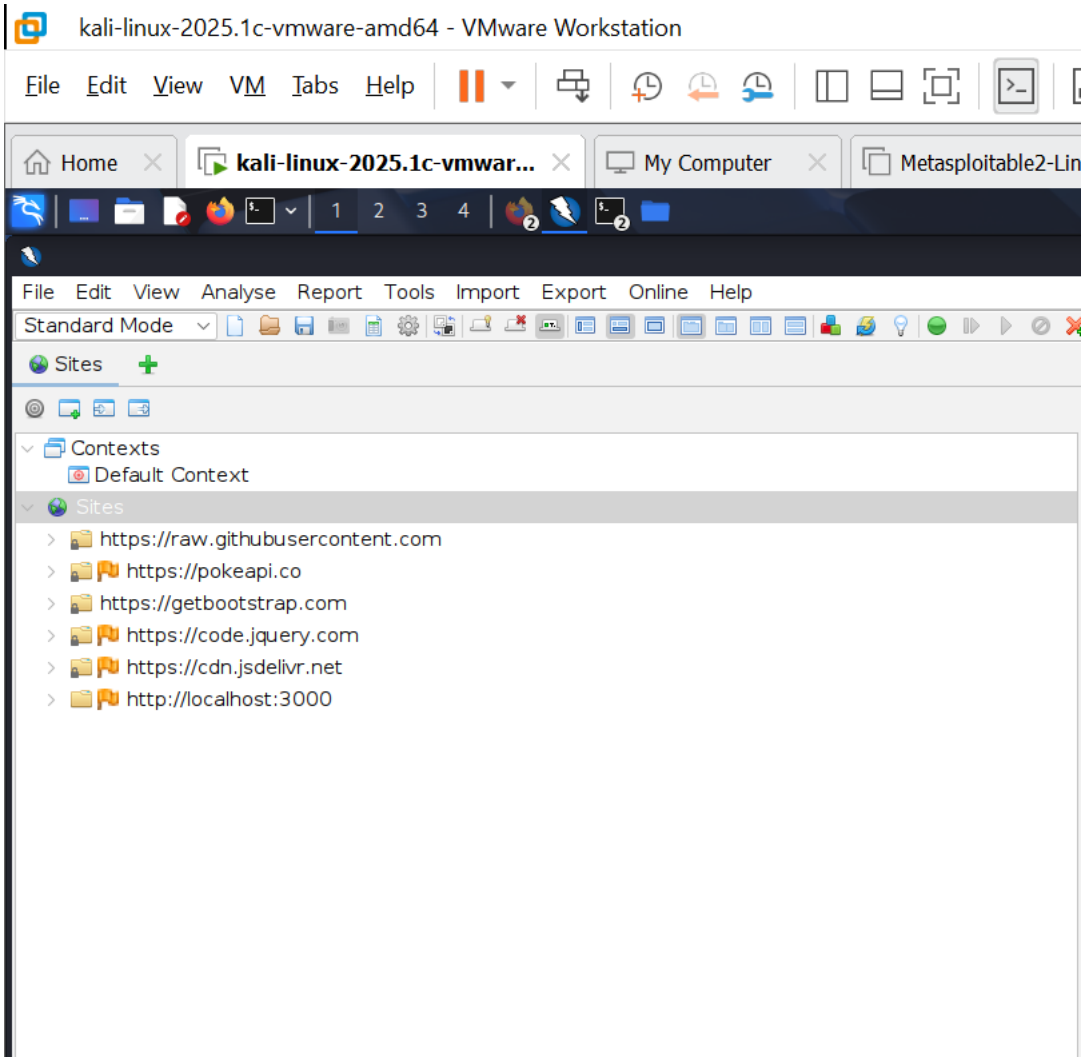


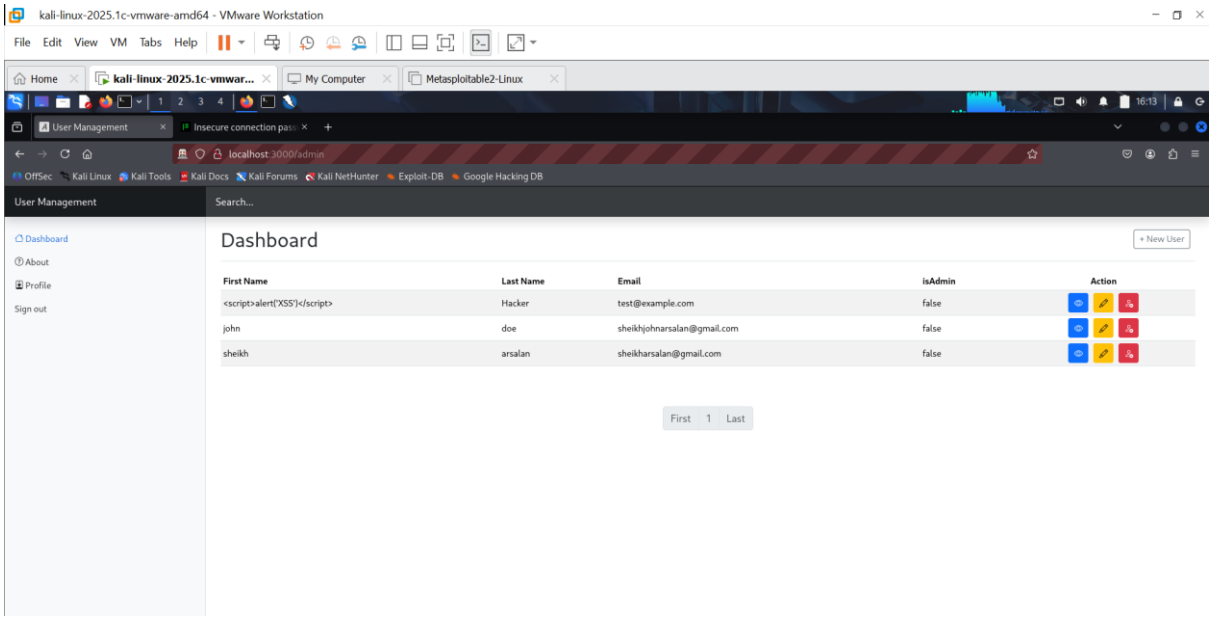

kali-linux-2025.1c-vmware-amd64 - VMware Workstation
User Login
User Login
Email
example@email.com
Password
Submit


kali-linux-2025.1c-vmware-amd64 - VMware Workstation
Active Scan
Scope
Filter
Starting Point:
http://localhost:3000
Policy:
Default Policy
Context:
User:
Recurse:
Show Advanced Options
Start Scan
Reset
Cancel


kali-linux-2025.1c-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2025.1c-vmwar...
My Computer
Metasploitable2-Linux
Connection Settings
Configure Proxy Access to the Internet
No proxy
Auto-detect proxy settings for this network
Use system proxy settings
Manual proxy configuration
HTTP Proxy localhost Port 8080
Also use this proxy for HTTPS
HTTPS Proxy localhost Port 8080
SOCKS Host Port 0
SOCKS v4 SOCKS v5
Automatic proxy configuration URL
Reload
No proxy for
Cancel
OK


kali-linux-2025.1c-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2025.1c-vmwar...
My Computer
Metasploitable2-Linux
User Register
localhost:3000/register
User Register
First name
Last name
<script>alert('XSS')</script>
Hacker
Email
test@example.com
Password
Password Confirmation
Submit
This connection is not secure.
Logins entered here could be compromised. Learn More


kali-linux-2025.1c-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2025.1c-vmwar...
My Computer
Metasploitable2-Lin
1 2 3 4
File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Contexts
Default Context
Sites
https://raw.githubusercontent.com
https://pokeapi.co
https://getbootstrap.com
https://code.jquery.com
https://cdn.jsdelivr.net
http://localhost:3000


kali-linux-2025.1c-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2025.1c-vmwar...
My Computer
Metasploitable2-Linux
User Management
Insecure connection pass
localhost:3000/admin
OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB
User Management
Search...
Dashboard
About
Profile
Sign out
Dashboard
+ New User
First Name
Last Name
Email
isAdmin
Action
<script>alert('XSS')</script>
Hacker
test@example.com
false
john
doe
sheikhjohnarsalan@gmail.com
false
sheikh
arsalan
sheikharsalan@gmail.com
false
First 1 Last

B. Check for XSS (Cross-Site Scripting)

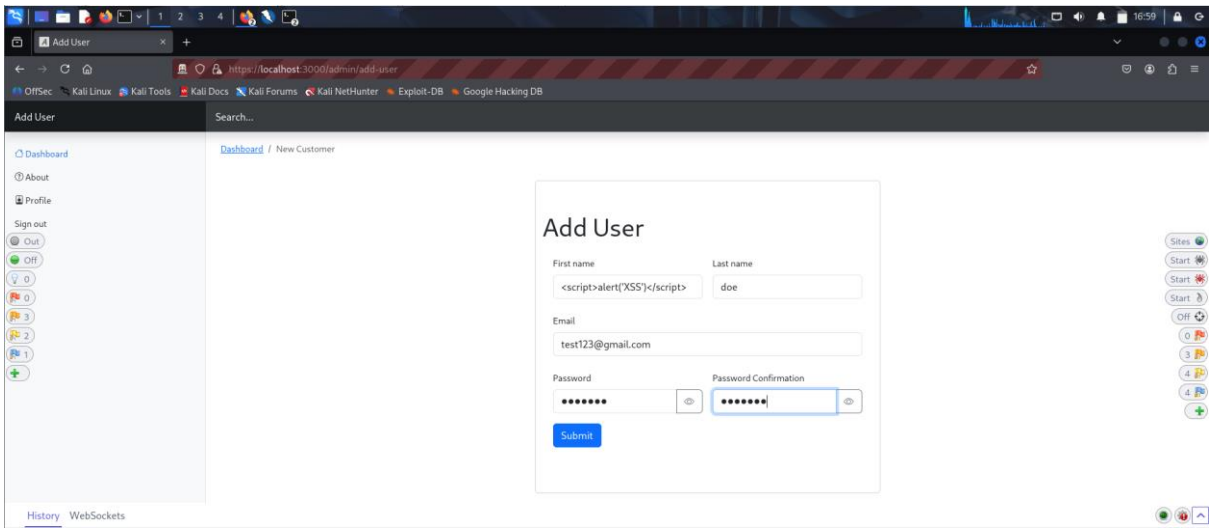
Go to a text input (e.g., bio, comments, name)

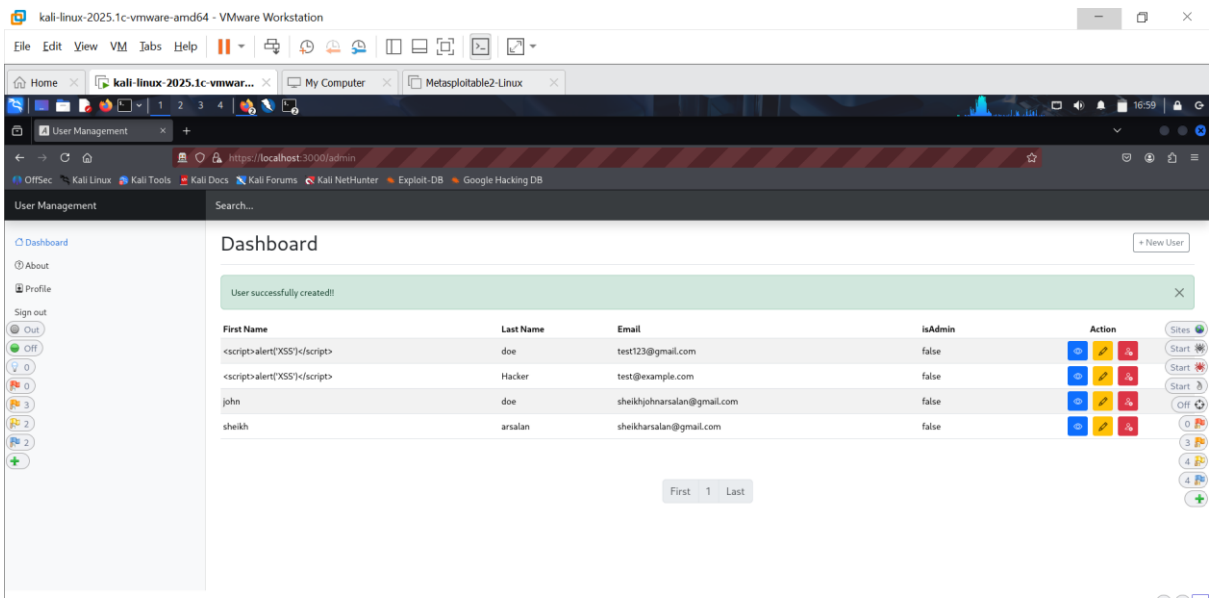
Type:

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

Click submit

If a popup appears, the site is vulnerable to XSS



C. Check for SQL Injection

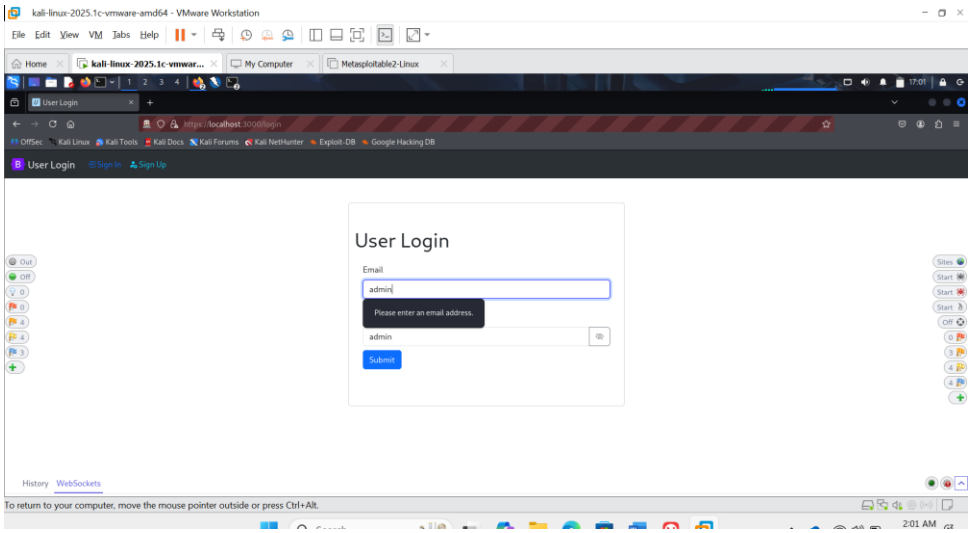
Go to login page

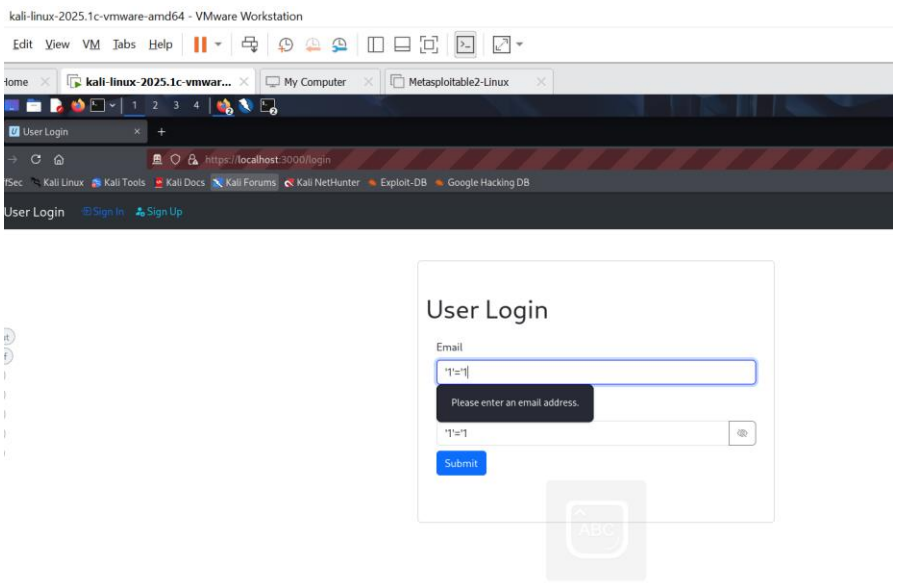
Enter:

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

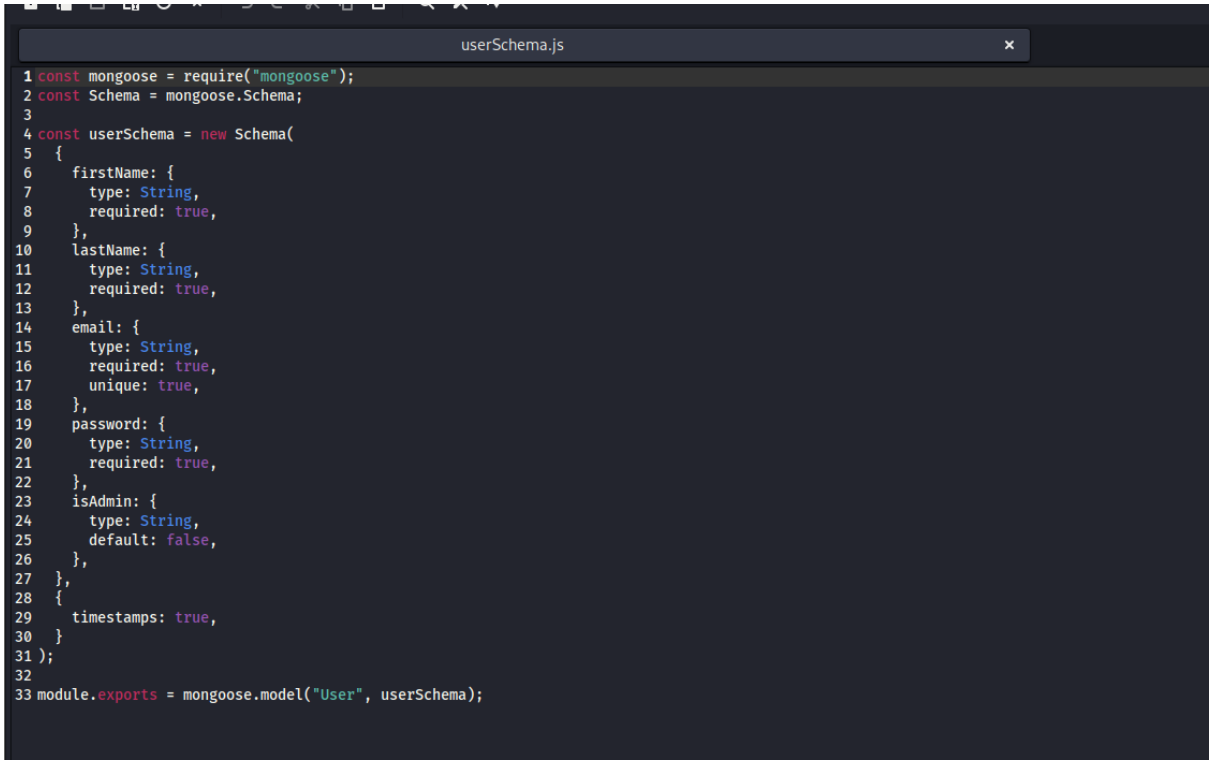
Open the project code

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




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).
- 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.

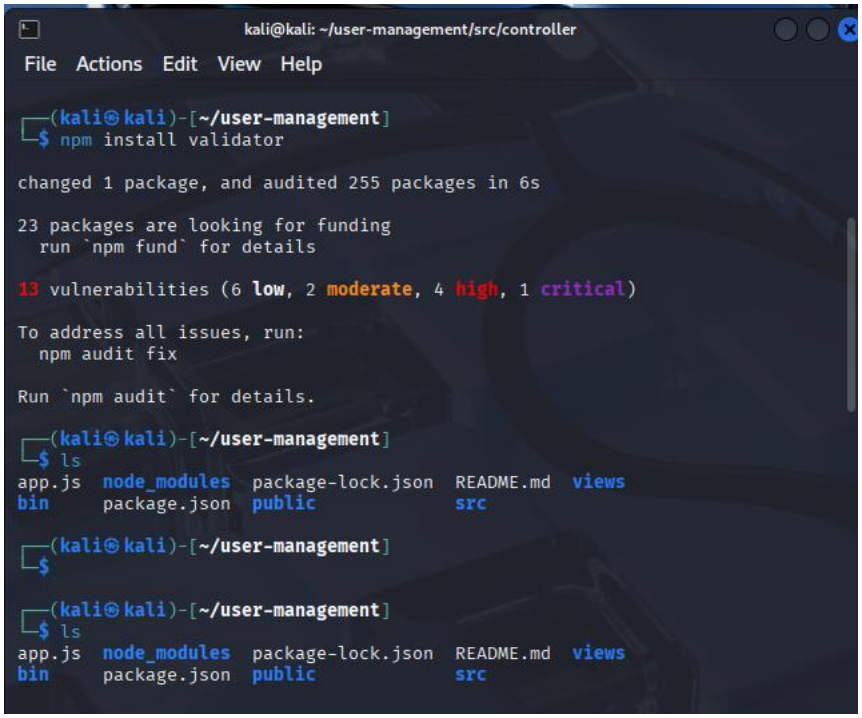
## Week 2 – Security Implementation Report

## Step 1: Sanitize and Validate Inputs

### 1.1 Install validator

Open terminal and run:

npm install validator

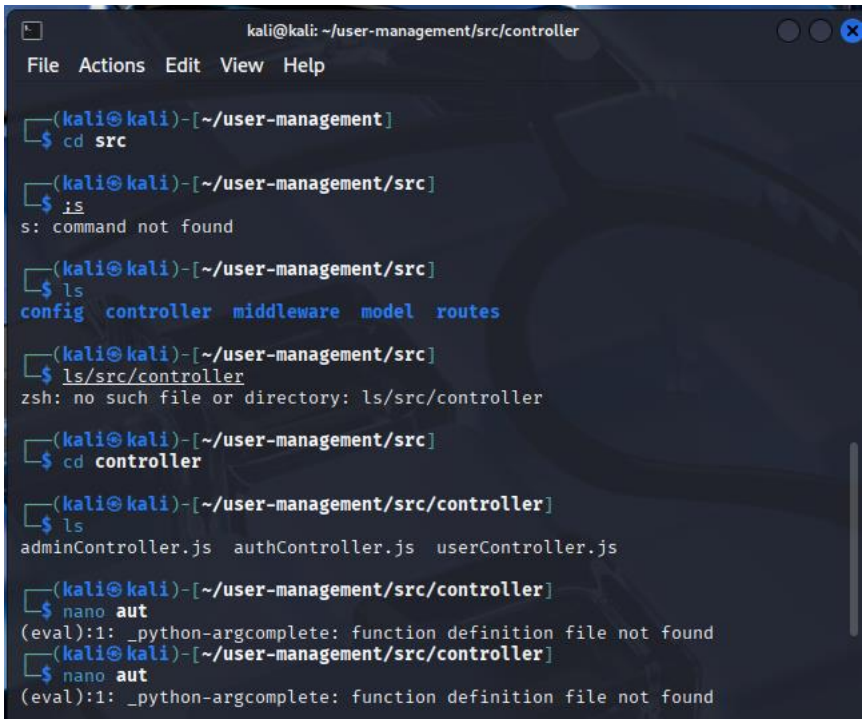


### 1.2 Open the Signup Controller

Navigate to the controller where the signup logic is handled:

cd src/controller

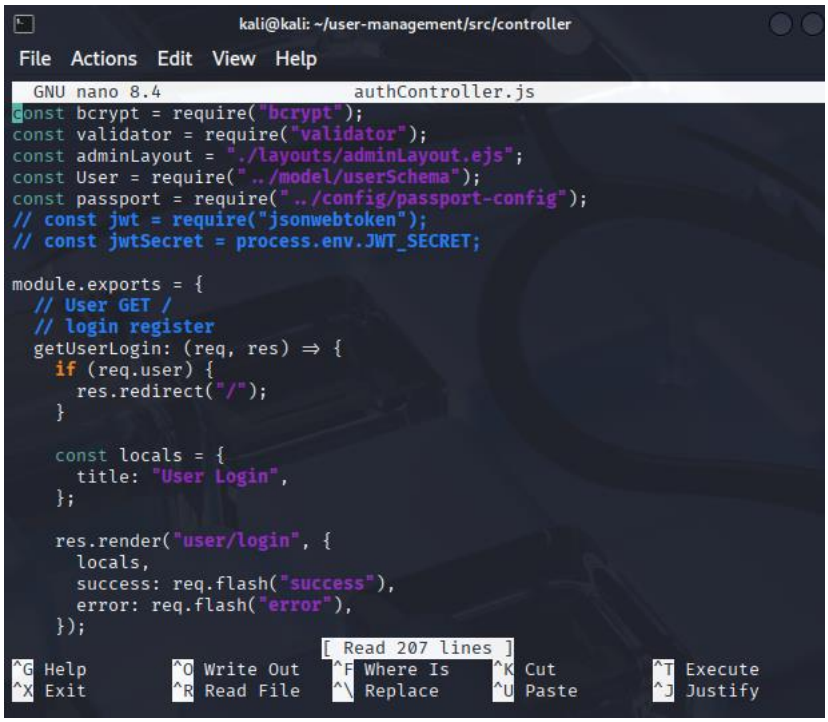
nano authController.js



### 1.3 Import validator

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

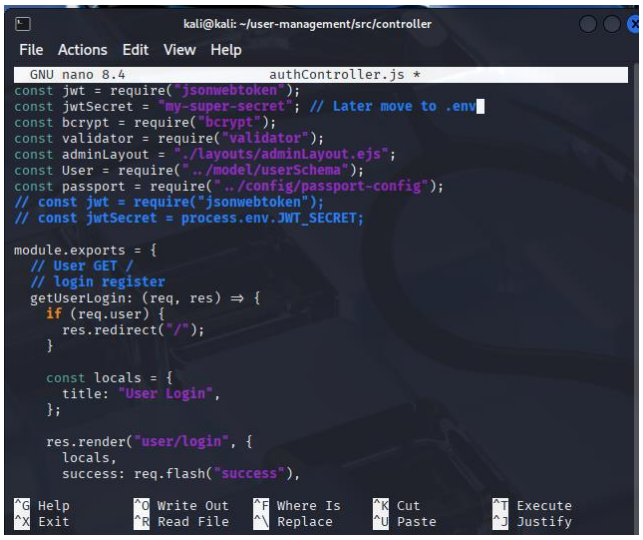
const validator = require('validator');

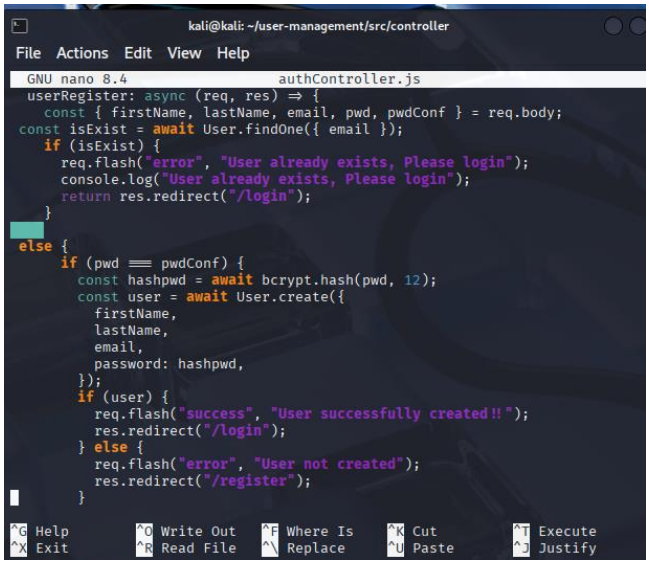


### 1.4 Add Input Validation in userRegister Function

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");
}
```

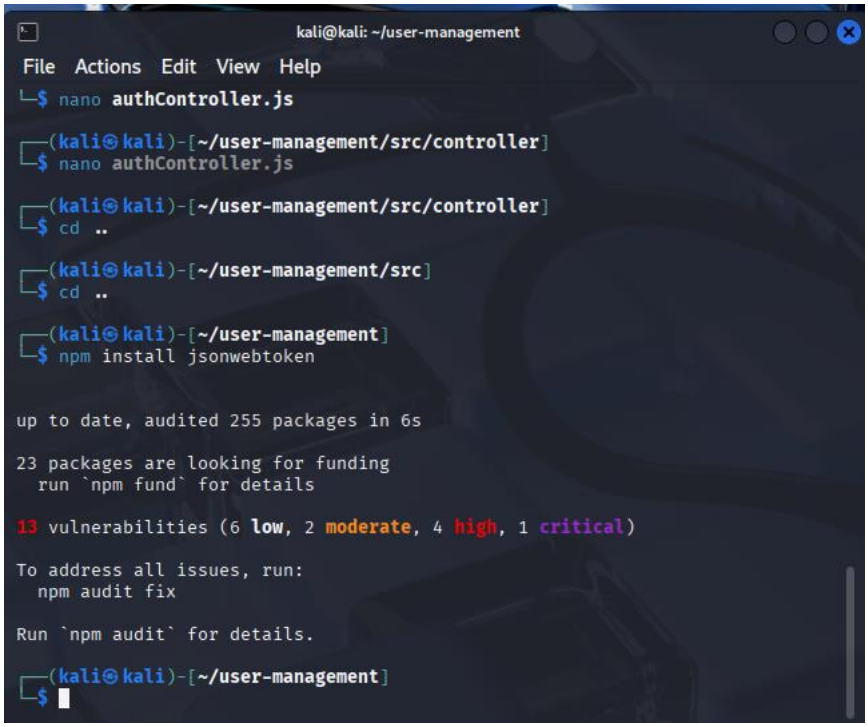





## Step 3: Implement JWT Authentication

### 3.1 Install jsonwebtoken

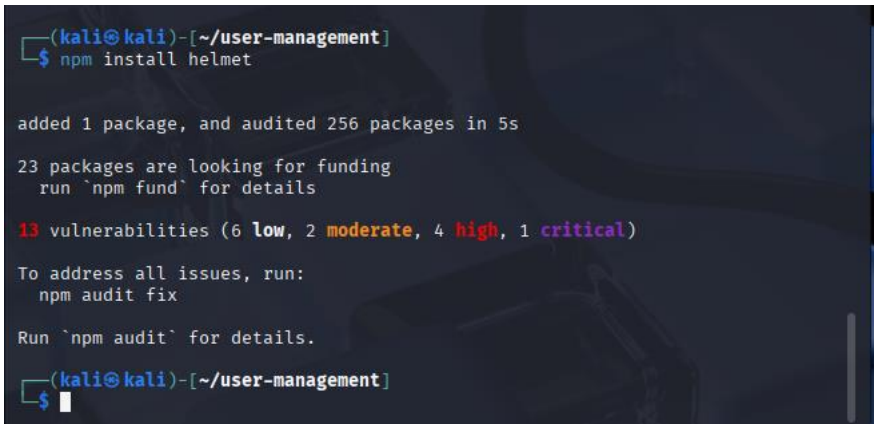
npm install jsonwebtoken



## Step 4: Secure HTTP Headers with Helmet

### 4.1 Install helmet

npm install helmet



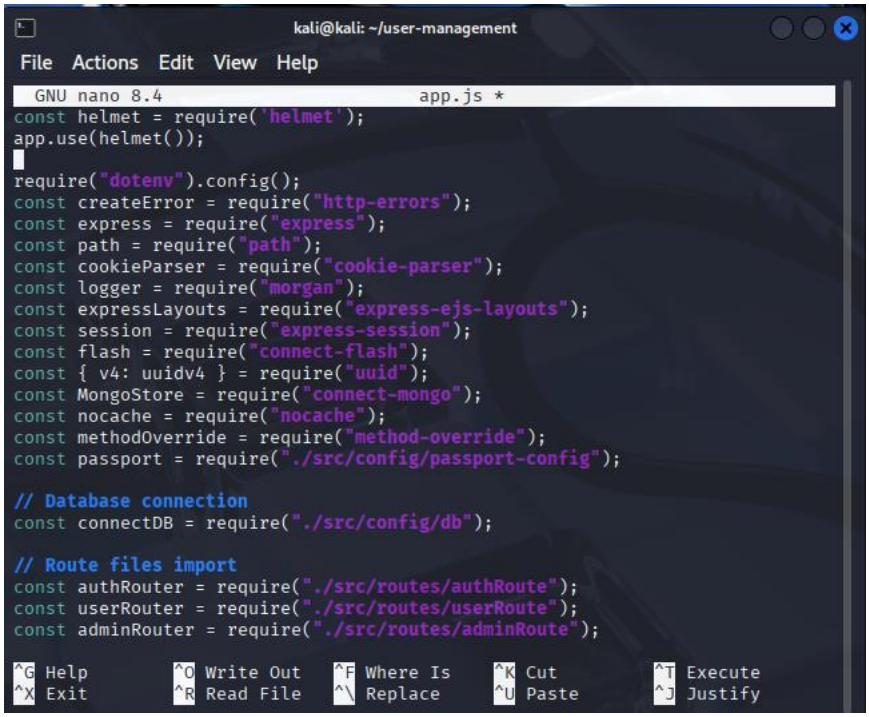

### 4.2 Use helmet in app.js

Open app.js and at the top, add:

const helmet = require('helmet');

Then below all require() statements:

app.use(helmet());




**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**

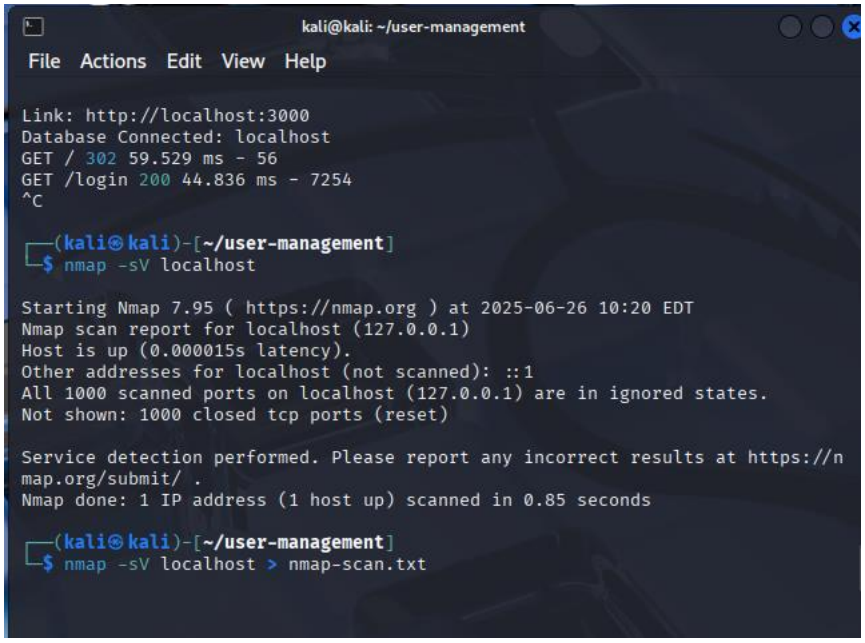
**Goal:**

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

**Step 1: Simulate Attacks Using Nmap (Optional)**

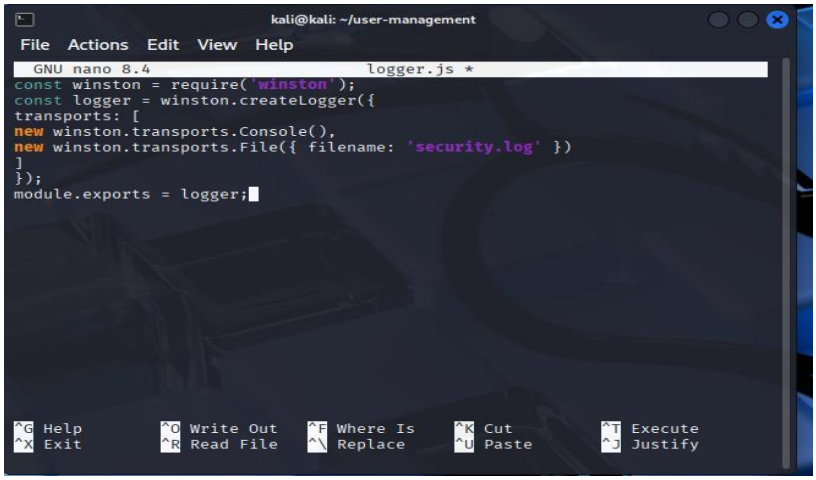
In your Kali VM terminal:

nmap -sV localhost




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

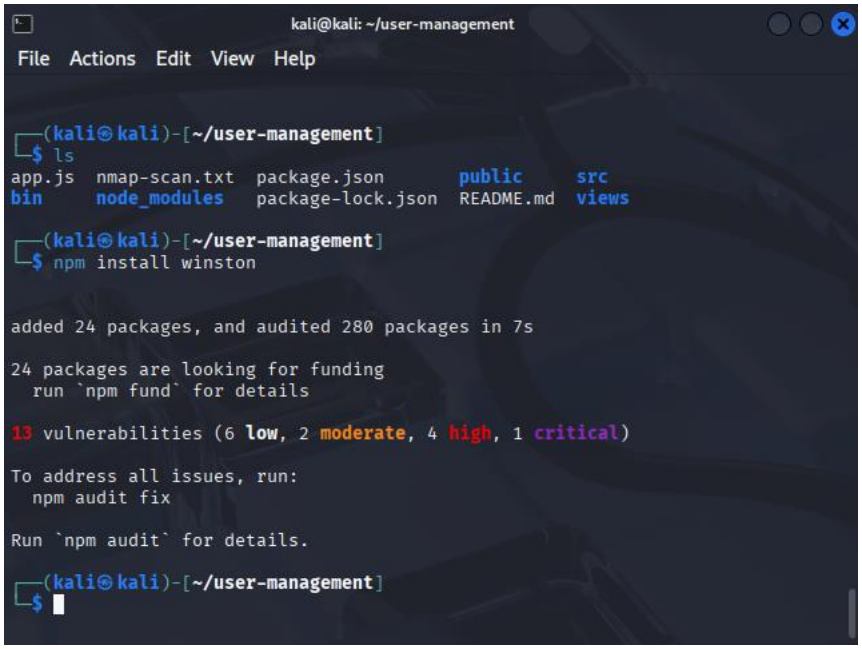
### Step 2: Add Logging with Winston




### 2.1 Install Winston

In terminal:

npm install winston




### 2.2 Create a logger.js File

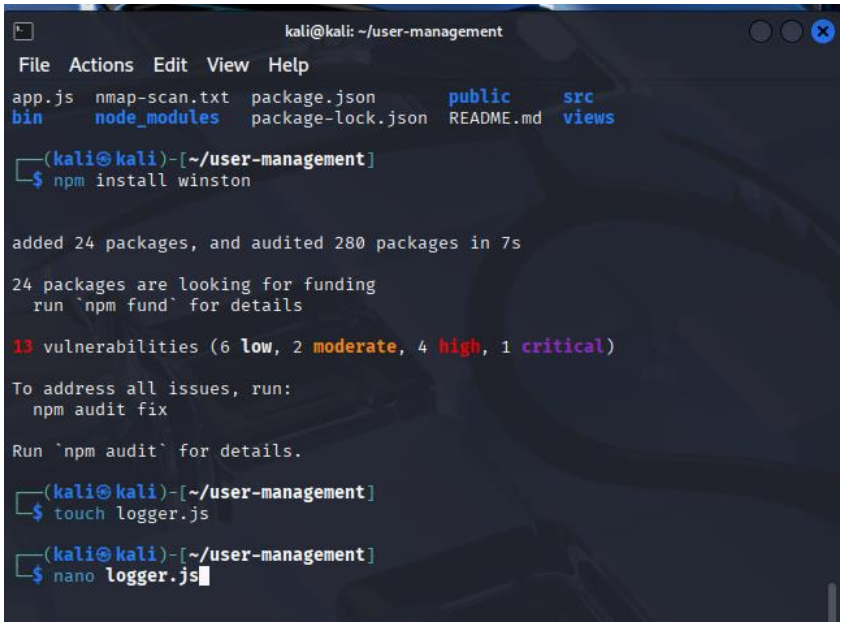
In your project root (same folder as app.js), run:
touch logger.js
nano logger.js
Paste the following code inside logger.js:

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

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

module.exports = logger;
```

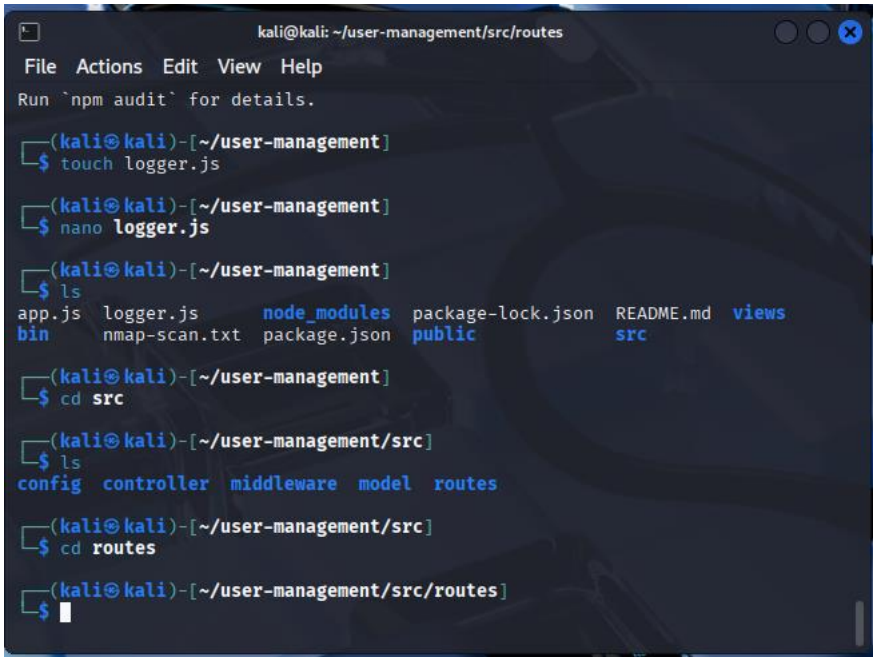
Then save and exit:

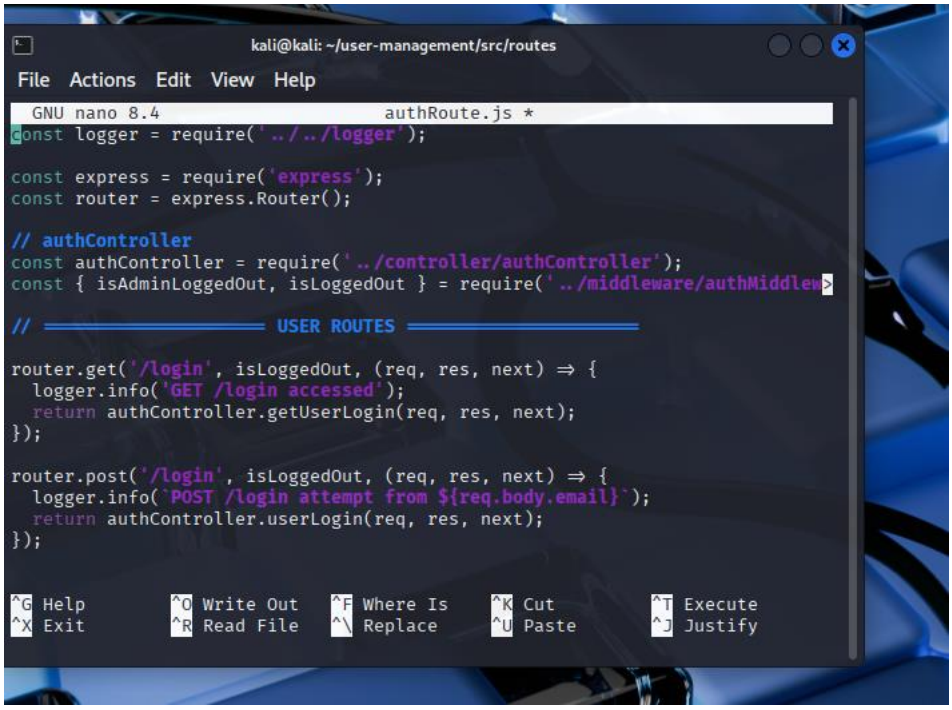



### 2.3 Use Logger in Routes

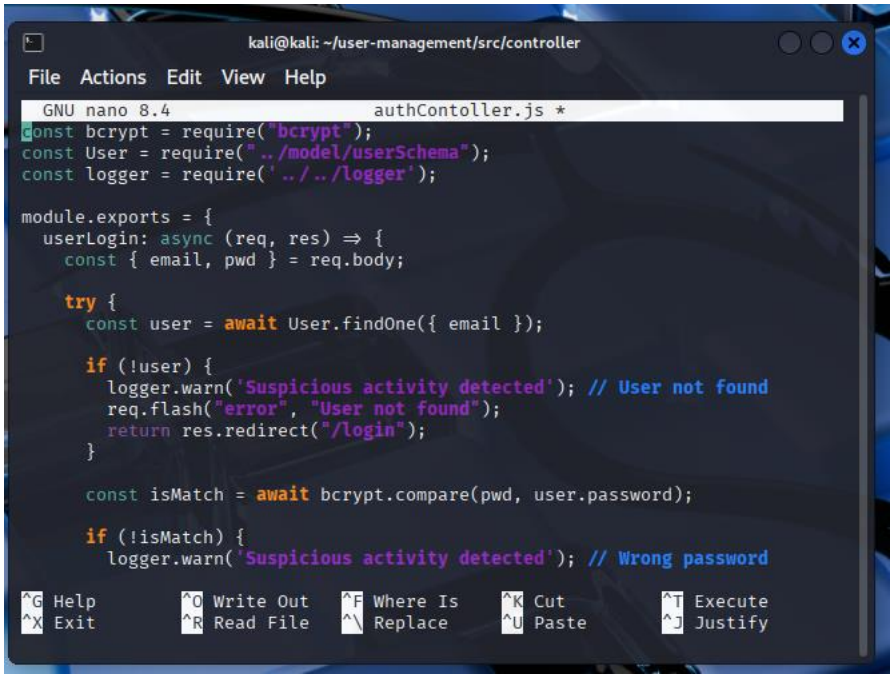
Go to your routes folder:
cd src/routes
nano auth.js
At the top of auth.js, add:
const logger = require('../../logger'); // adjust if path is different
Then add logging inside routes, like:
logger.info('User logged in');
logger.warn('Suspicious activity detected');
Example use:

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




## 2.4 In Controller




**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.