**Internship Task: Security Assessment of Web Application**

**Intern Name:** **Rabia Khawaja**

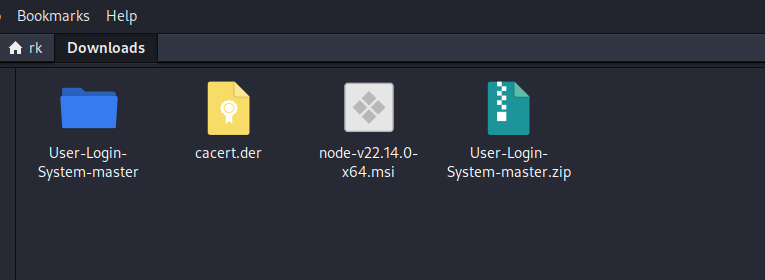
**Date:** 22nd March 2025

**Week 1 Report — Security Assessment**

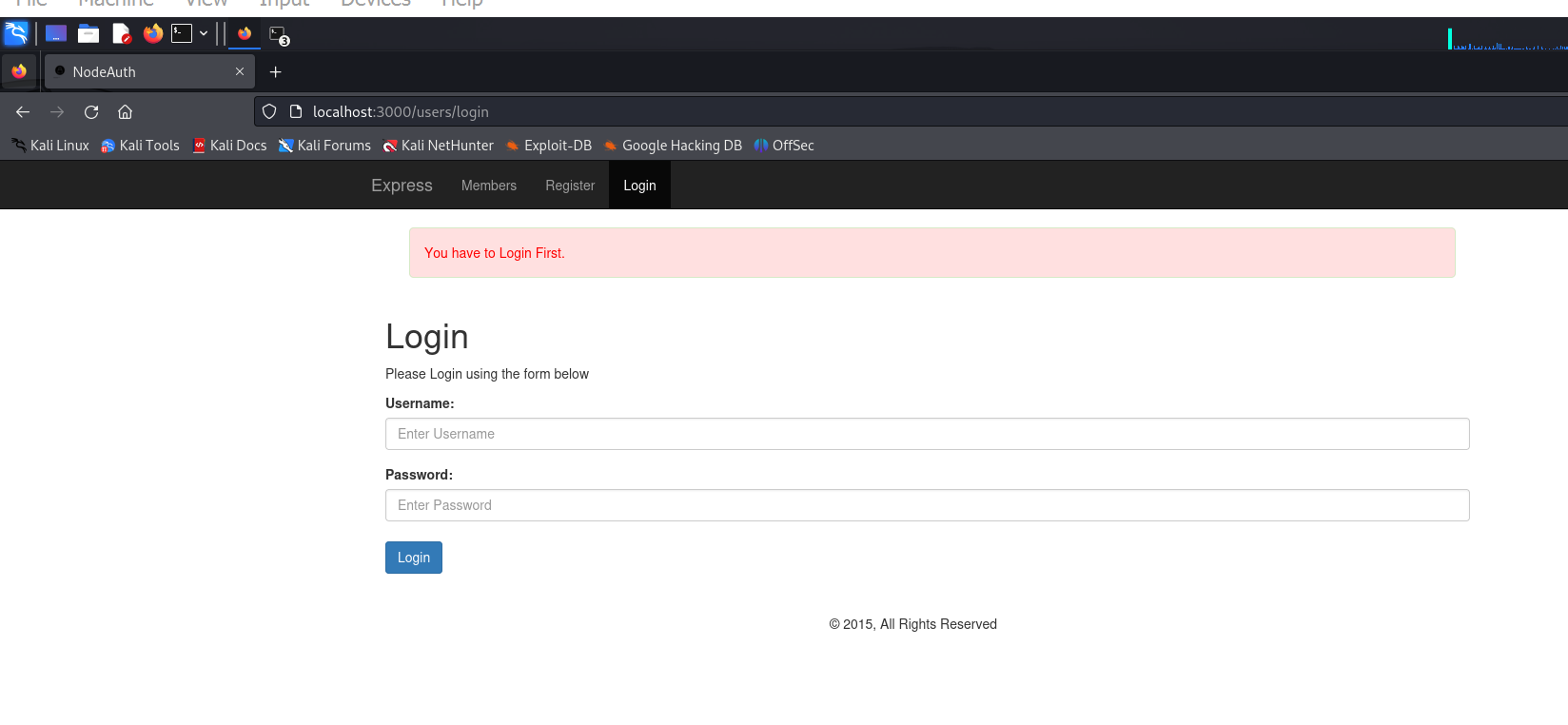
## ****1. Application Overview****

The web application is running on **http://localhost:3000.**

Functionalities tested: Login, Register, and Profile pages.

Objective: Identify common security vulnerabilities.  
  




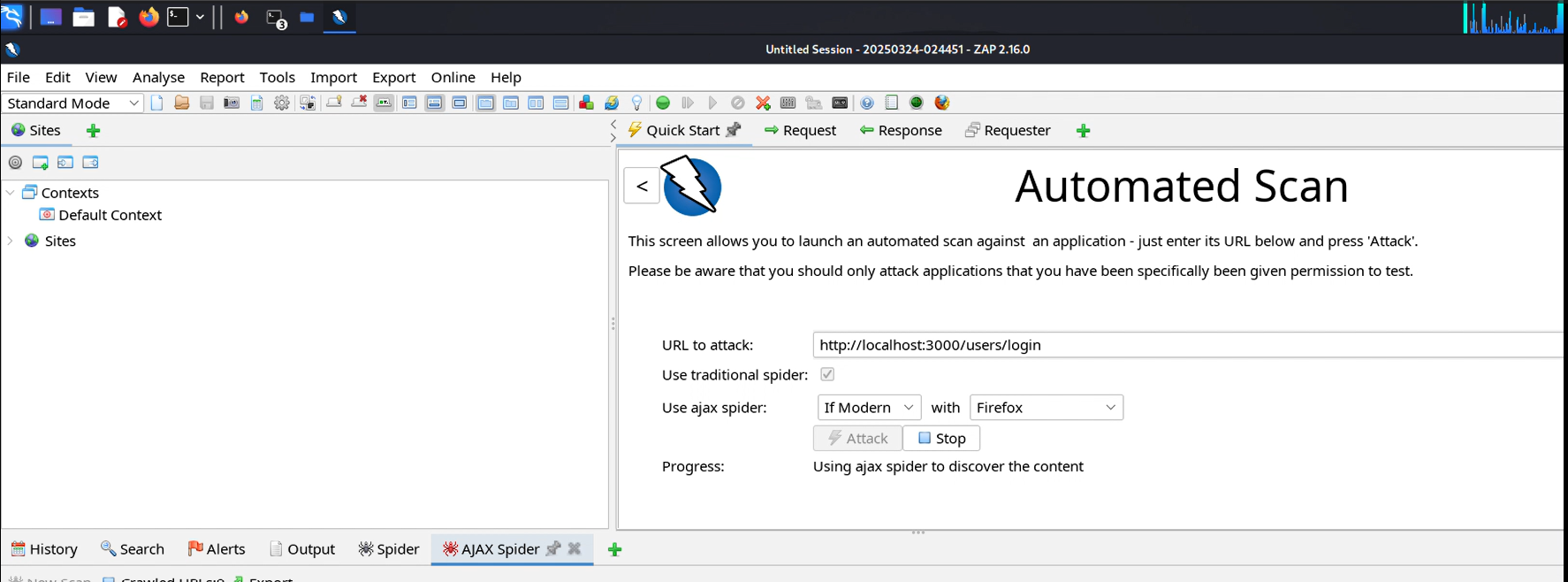


## ****2. Vulnerability Assessment****

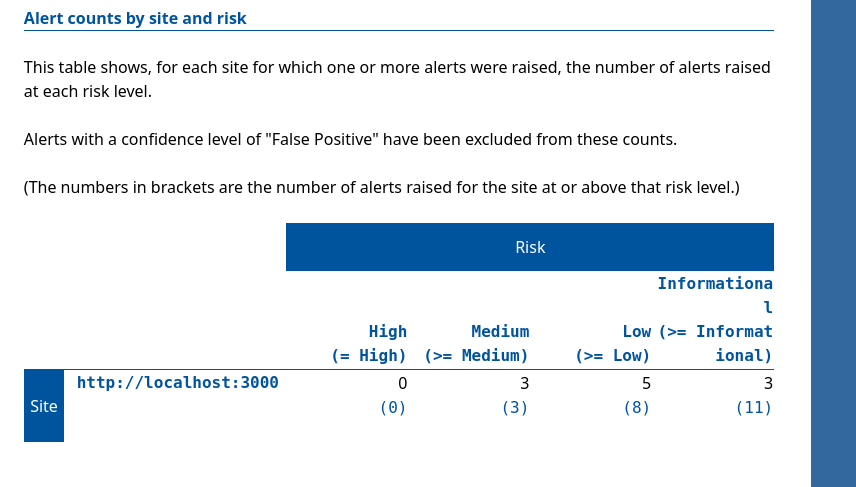
### ****A. OWASP ZAP Scan****

**Tool Used:** OWASP ZAP

**Methodology:** Automated scan was performed on **<http://localhost:3000>**



**Findings.**





Found security misconfigurations.

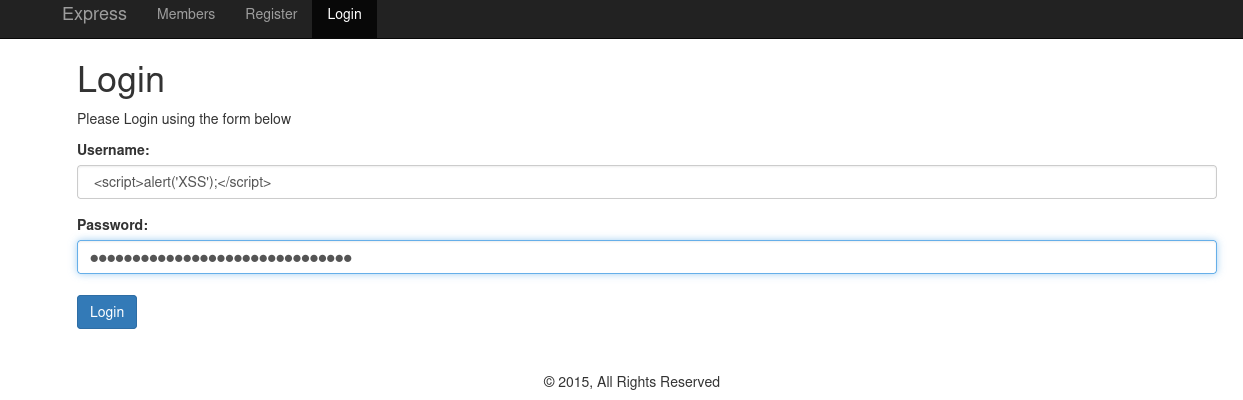
Found XSS vulnerabilities.

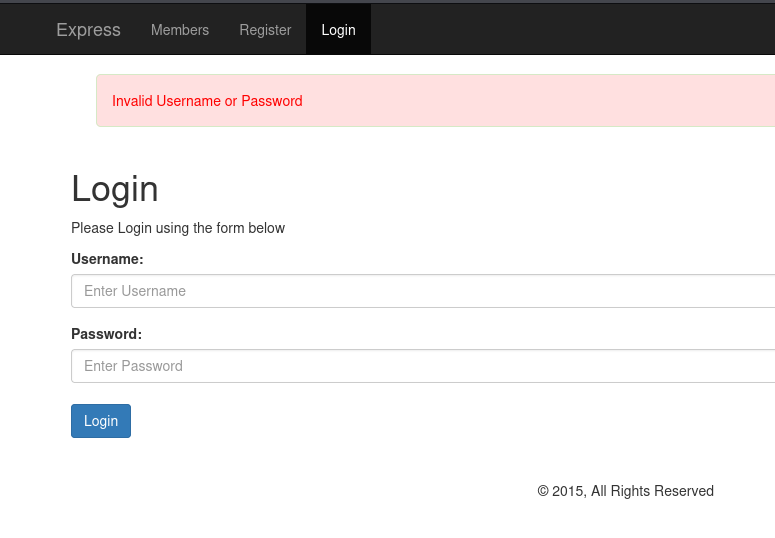
Found weak authentication mechanisms.

**Evidence:** file:///home/rk/2025-03-24-ZAP-Report-.html

### ****B. Cross-Site Scripting (XSS) Testing****

**Test Performed:** Injected JavaScript payload <script>alert('XSS');</script> in input fields.

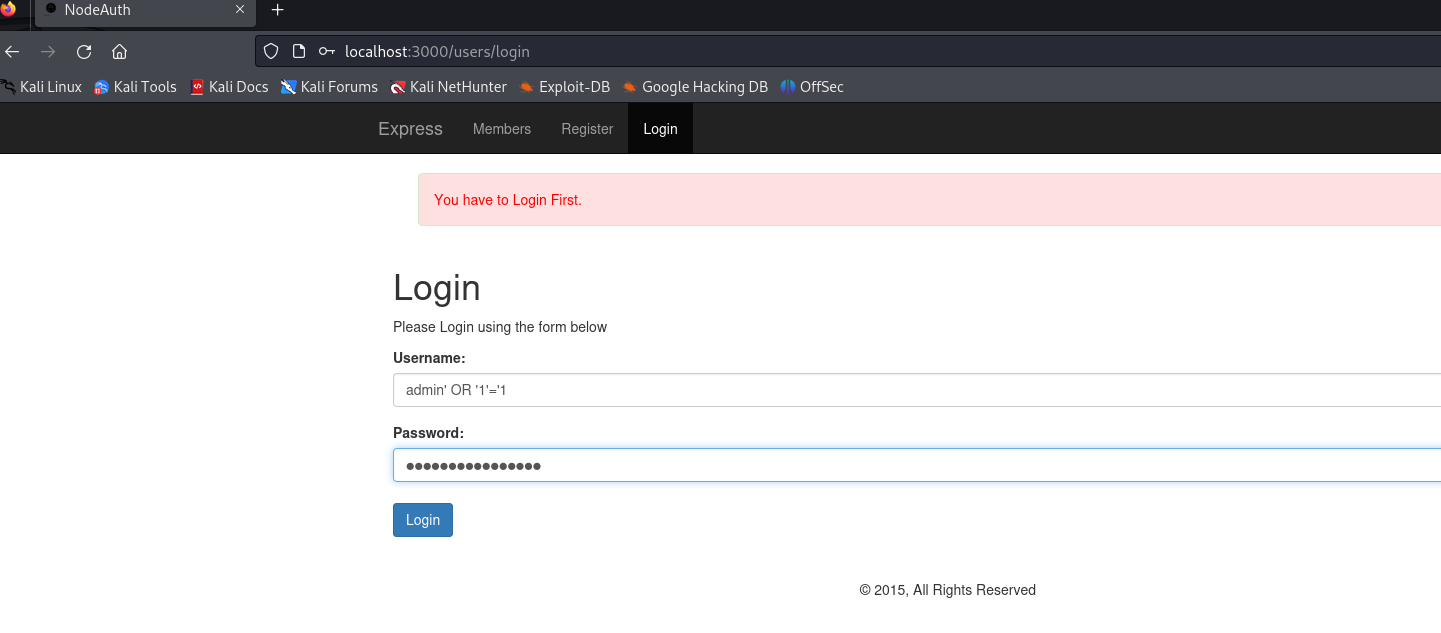


Results:  


Its secure as cross site Scripting

### ****C. SQL Injection Testing****

**Test Performed:** Attempted to bypass authentication using SQL payload admin' OR '1'='1.



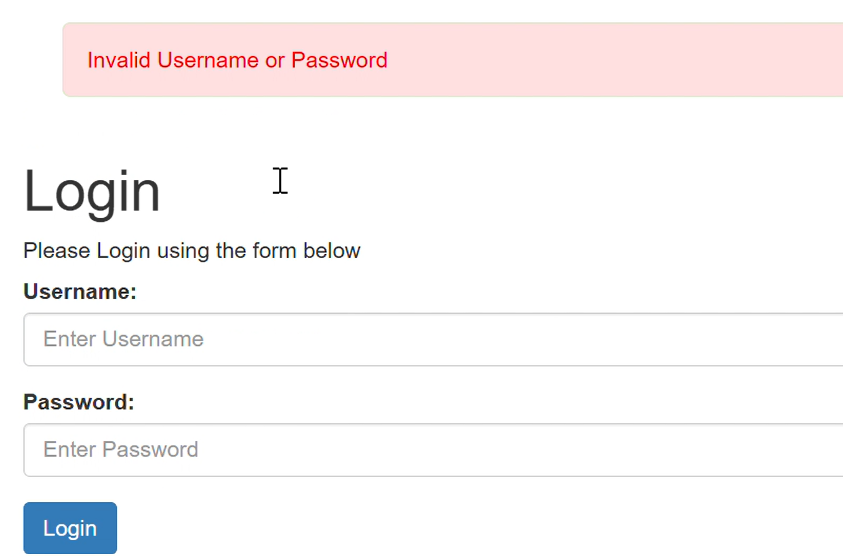


Image 1 Result, its secure

## ****3. Summary of Findings****

**XSS Vulnerability Test:**

**- Payload used: <script>alert('XSS')</script>**

**- Result: Application did not execute the script; returned 'Invalid username' message.**

**- Conclusion: No XSS vulnerability found at this point.**

**Week 2 Security Implementation Report**

**Project Name:** Secure User Login System

### ****1. Fixing Security Vulnerabilities****

#### ****Sanitization & Input Validation****

Implemented **express-validator** for user input validation.  
Used **sanitize-html** to remove any malicious code.  
 Ensured email format validation using validator.js.

#### ****Implementation Code:****

***const { body, validationResult } = require('express-validator');***

***const sanitizeHtml = require('sanitize-html');***

***app.post('/register', [***

***body('username').trim().escape(),***

***body('email').isEmail().normalizeEmail(),***

***body('password').isLength({ min: 8 }).withMessage('Password must be at least 8 characters')***

***], (req, res) => {***

***const errors = validationResult(req);***

***if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });***

***req.body.username = sanitizeHtml(req.body.username);***

***req.body.email = sanitizeHtml(req.body.email);***

***res.json({ message: "User input sanitized and validated!" });***

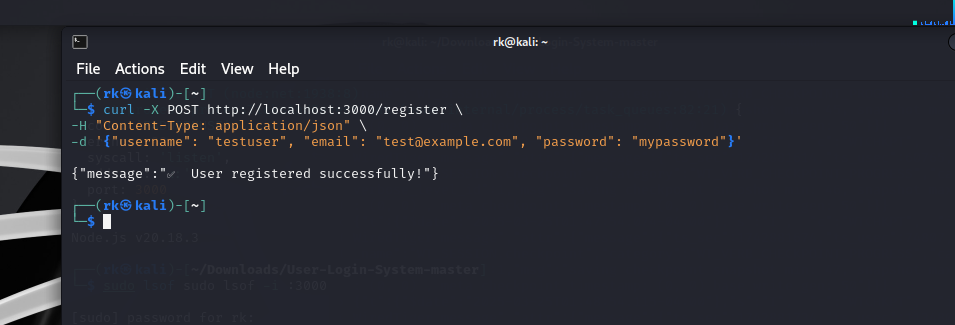
***});***

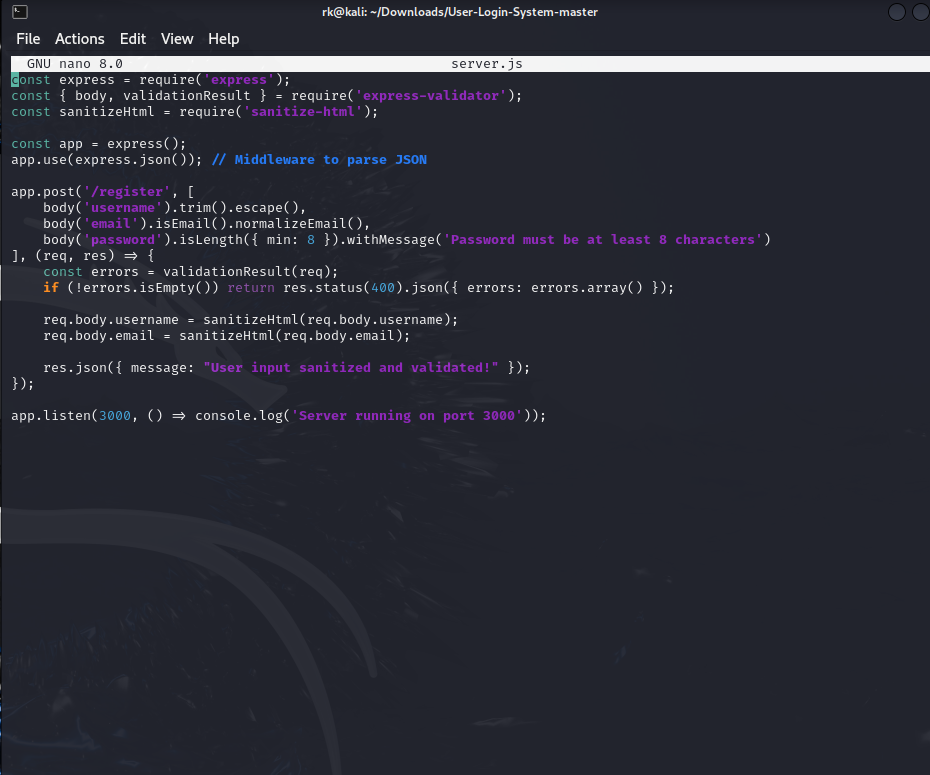
#### ****Password Hashing****

Used **bcrypt.js** to hash passwords securely before storing them in the database.

**const bcrypt = require('bcryptjs');**

**const hashedPassword = await bcrypt.hash(password, 10);**

****

****

### **week 2 task 1 ss 1 mongo db running**

### **week 2 task API wala ka part**

### ****2. Enhancing Authentication****

#### ****Token-Based Authentication (JWT)****

Implemented **jsonwebtoken (JWT)** for secure login authentication.  
Generated a unique token for each user upon successful login.

#### ****Implementation Code:****

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

***app.post('/login', async (req, res) => {***

***try {***

***const { email, password } = req.body;***

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

***if (!user) return res.status(400).json({ message: "User not found" });***

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

***if (!isMatch) return res.status(400).json({ message: "Invalid credentials" });***

***const token = jwt.sign({ id: user.\_id }, 'your-secret-key', { expiresIn: "1h" });***

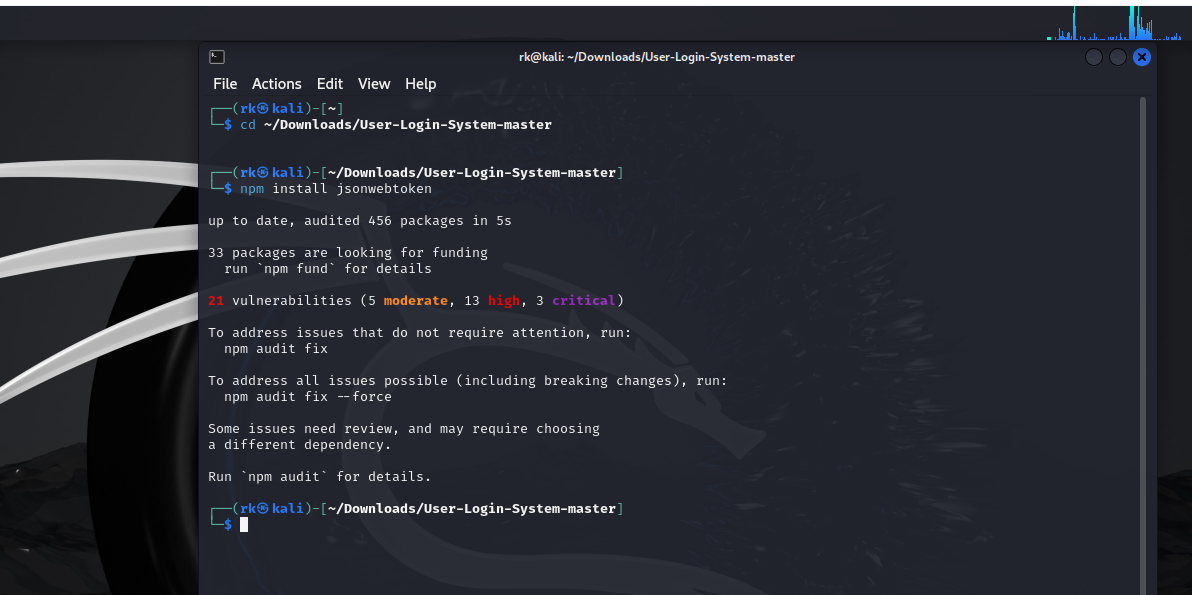
***res.json({ token, message: "Login successful!" });***

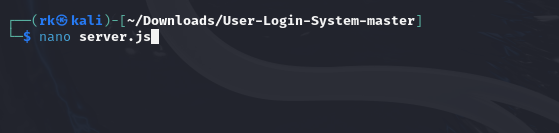
***} catch (err) {***

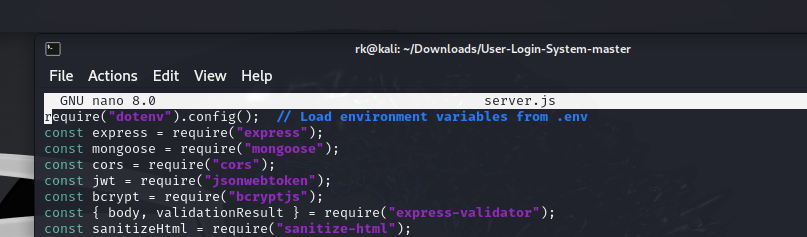
***res.status(500).json({ message: "Server error" });***

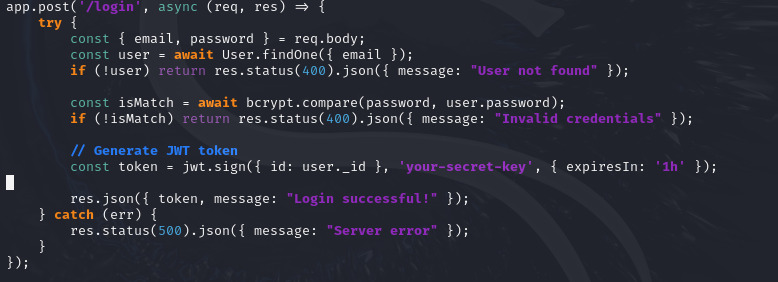
***}***

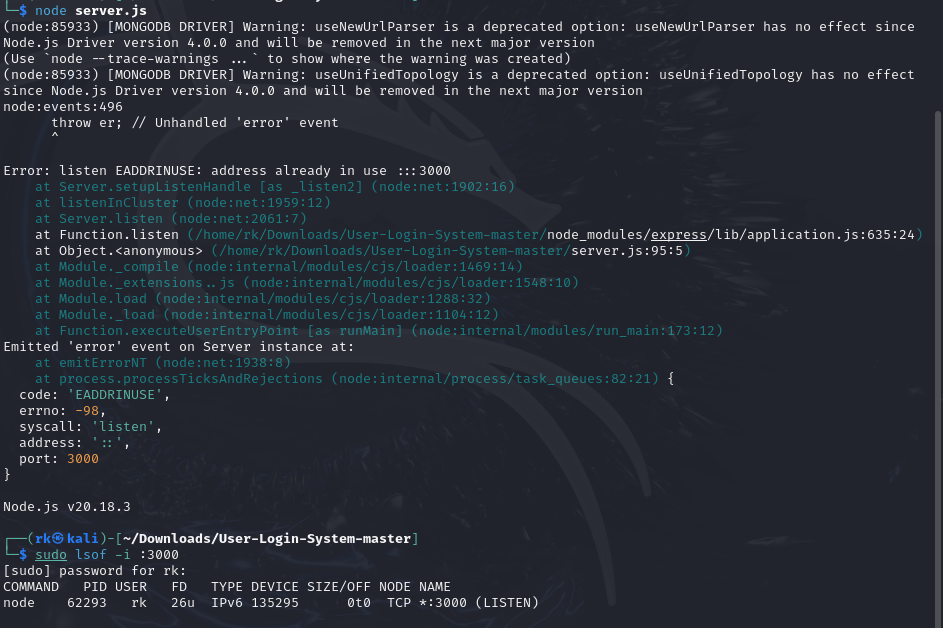
***});***

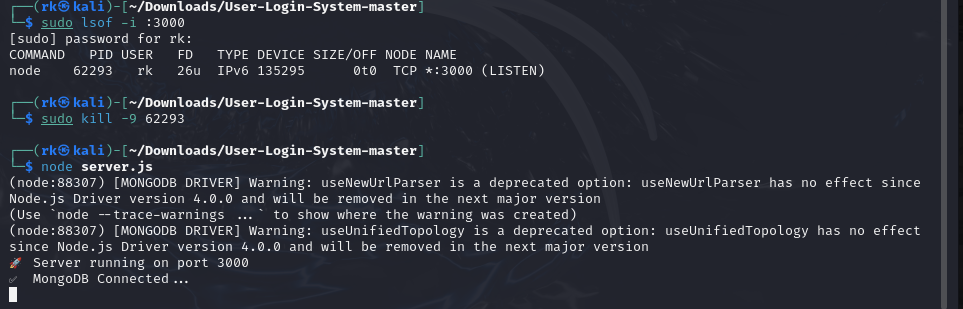


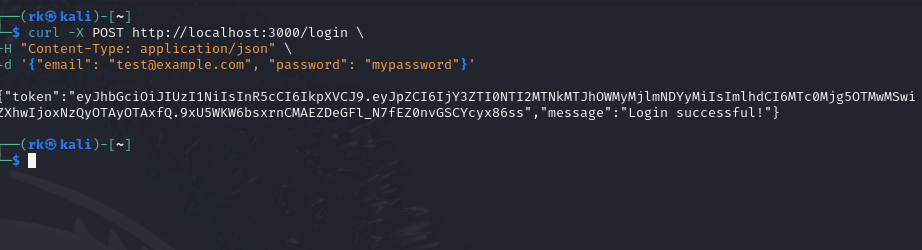












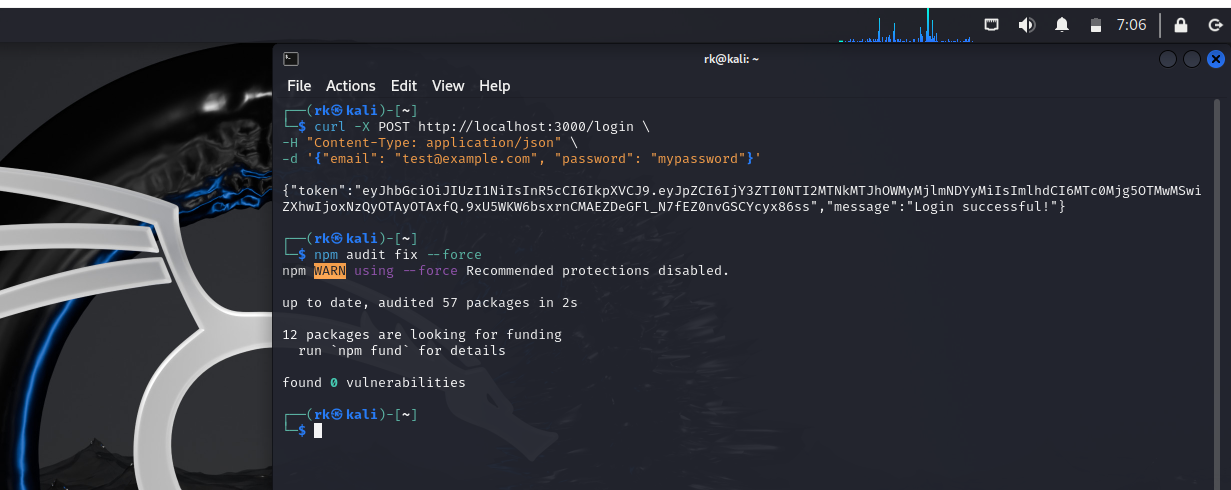


Image 2 yayy

### ****3. Securing Data Transmission****

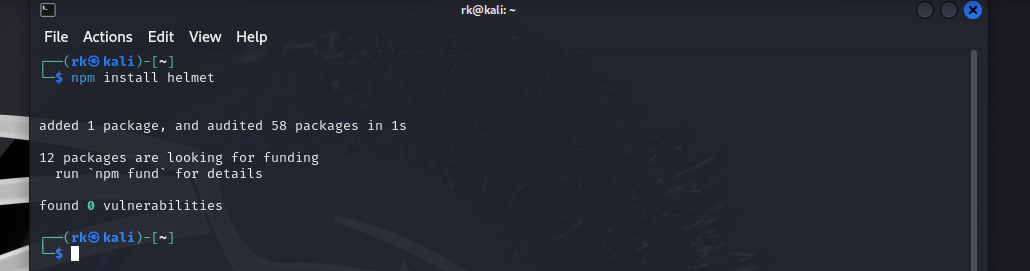
#### ****Using Helmet.js for Security Headers****

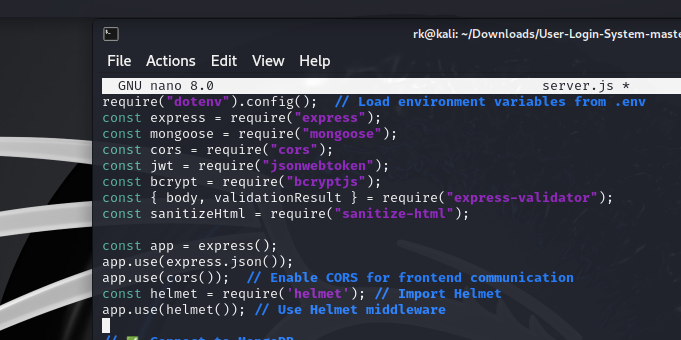
Installed and configured helmet.js to secure HTTP headers.  
This prevents Clickjacking, XSS, MIME sniffing, and other attacks.

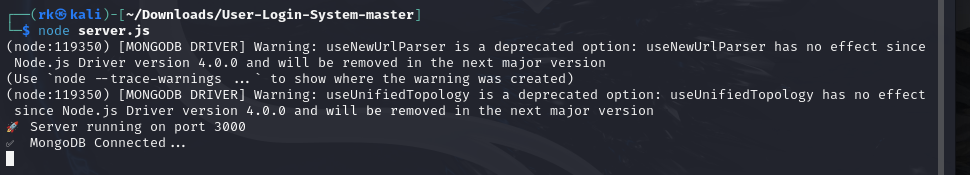
#### ****Implementation Code:****

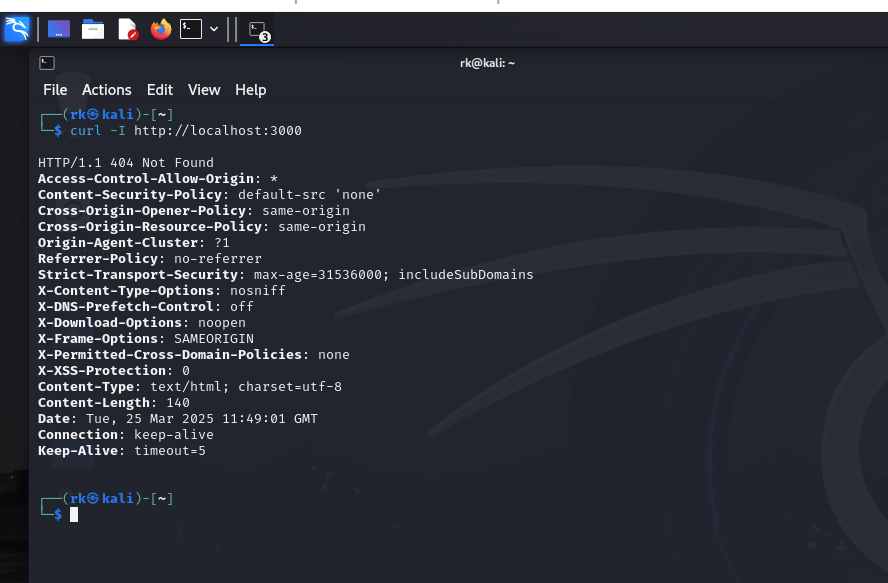
**const helmet = require('helmet');**

**app.use(helmet());**









### ****Security Testing Results****

**SQL Injection Test:** Prevented login bypass attempts with ' OR '1'='1'.  
**XSS Test:** JavaScript payloads like <script>alert('XSS');</script> were neutralized.  
 **JWT Authentication Test:** Users received a valid token upon successful login.  
 **Helmet.js Protection:** HTTP headers were correctly set for security.

**Conclusion:** All security measures were successfully implemented, securing the user authentication process. Further improvements can include **multi-factor authentication (MFA)** and **rate limiting** against brute force attacks.

**Week 3: Advanced Security and Final Reporting**

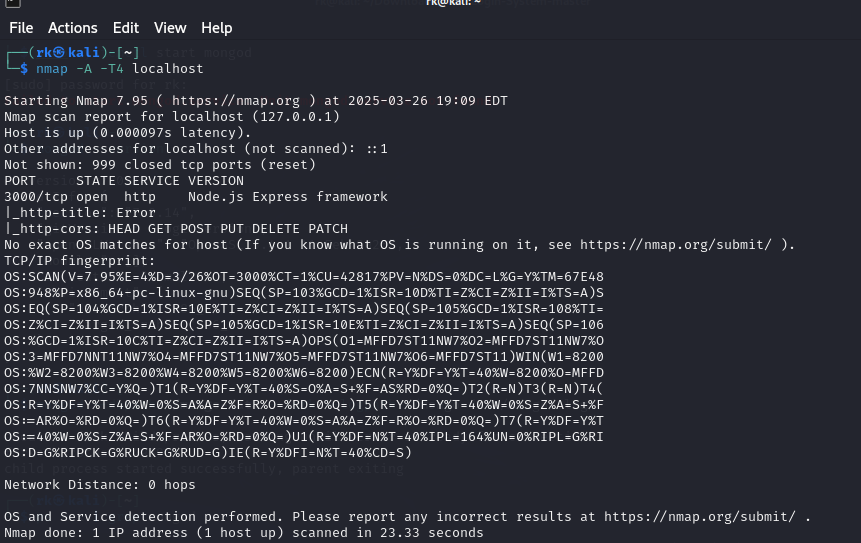
**Project Name: Secure User Login System**

1. **Basic Penetration Testing**



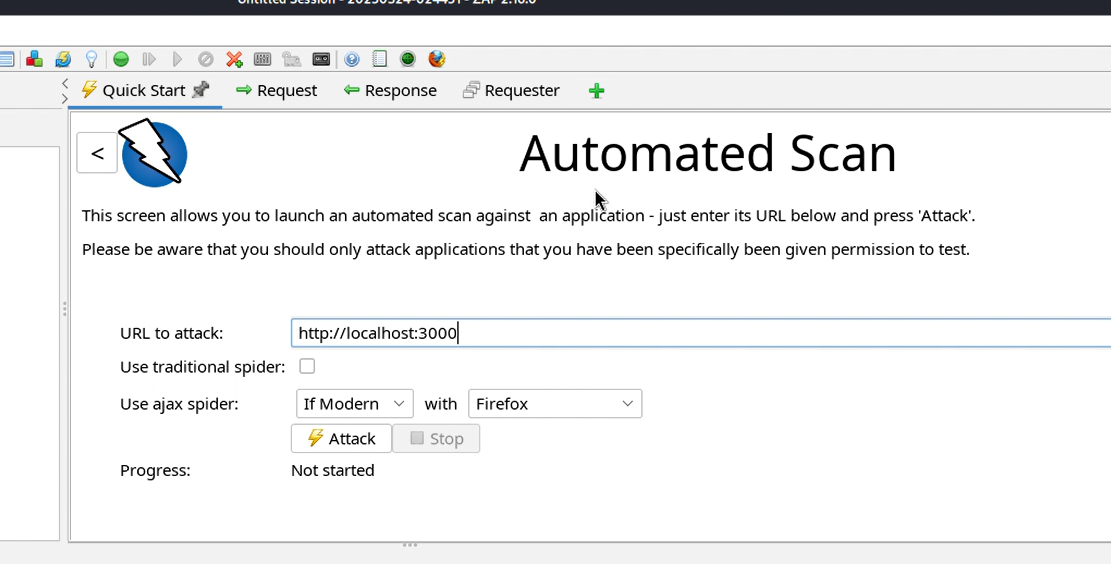
Performed **Nmap Scan** to identify open ports and services:

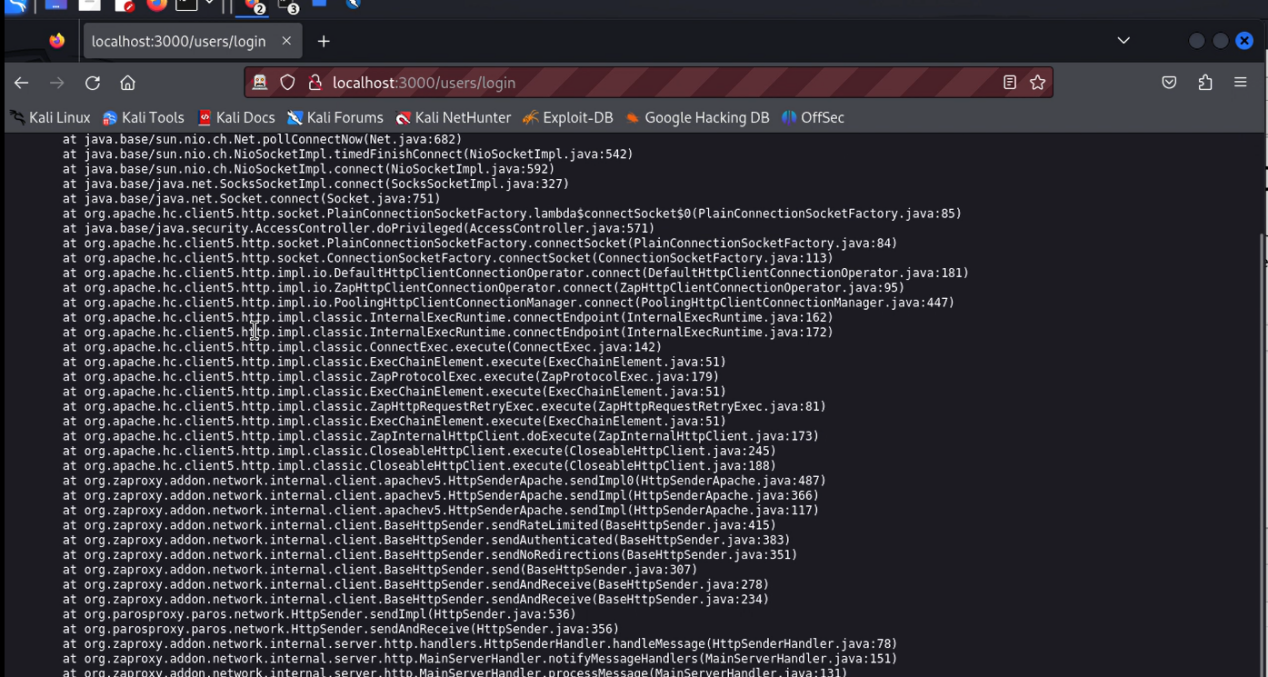
nmap -A -T4 localhost

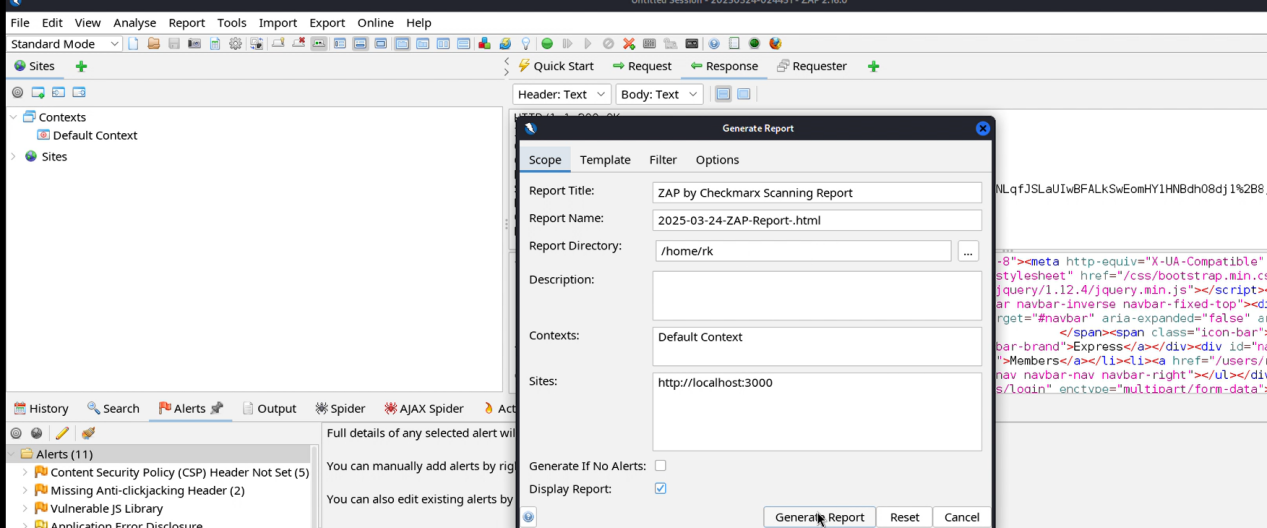


Nmap done..

**Conducted **Browser-Based Testing**:**







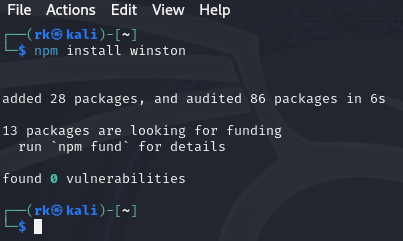


Tested for **XSS** by entering **<script>alert('XSS');</script**> in text fields.

Attempted **SQL Injection** using **admin' OR '1'='1** in login fields.

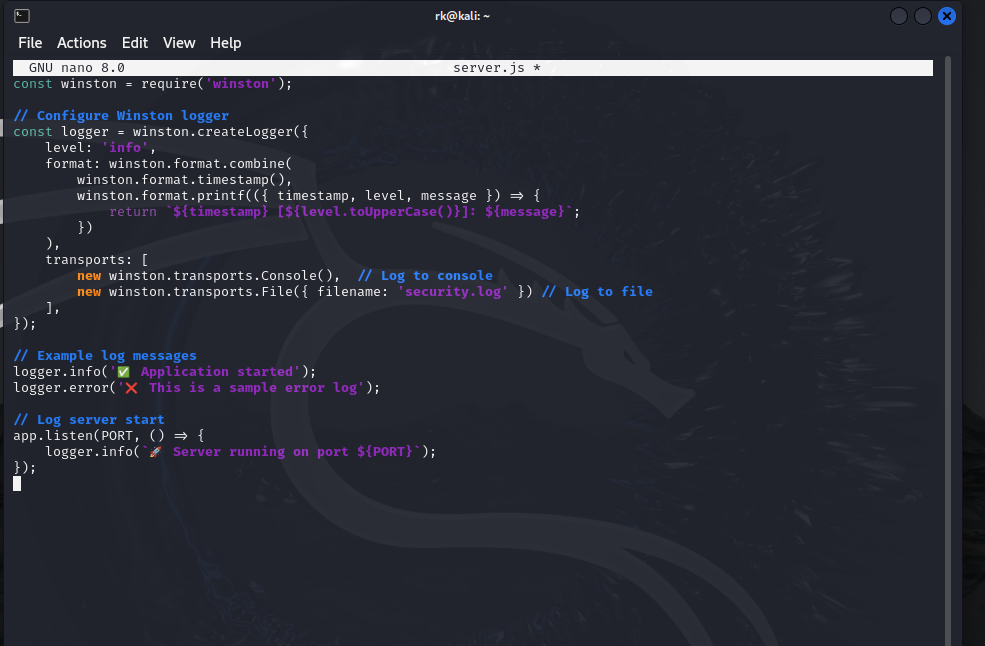
### ****Task 2: Set Up Basic Logging****

Installed Winston for logging:



npm install winston

Open nano server.js



Configured Winston Logger:

javascript

CopyEdit

const winston = require('winston');

const logger = winston.createLogger({

level: 'info',

format: winston.format.json(),

transports: [

new winston.transports.Console(),

new winston.transports.File({ filename: 'security.log' })

]

});

logger.info("Application started");

var fs = require('fs');

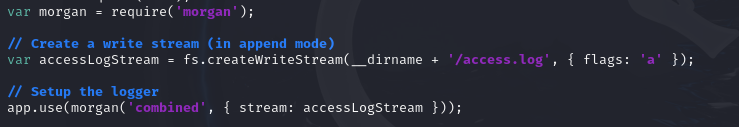
var morgan = require('morgan');

// Create a write stream (in append mode)

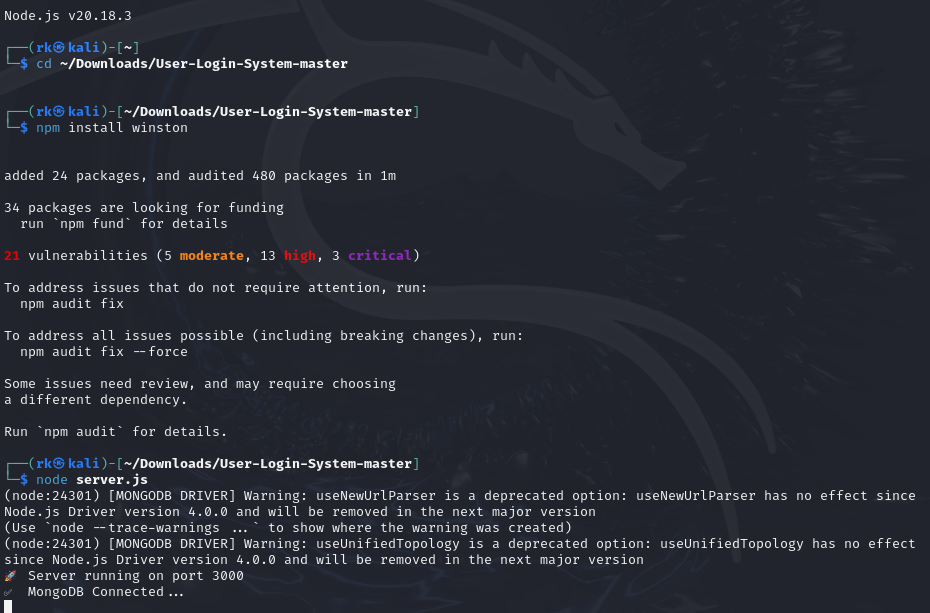
var accessLogStream = fs.createWriteStream(\_\_dirname + '/access.log', { flags: 'a' });

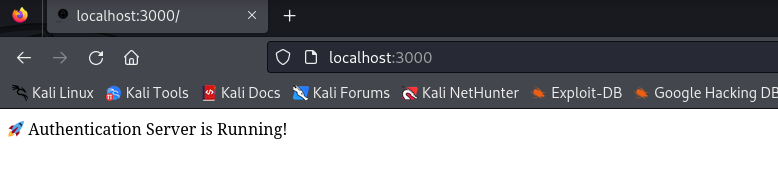
// Setup the logger

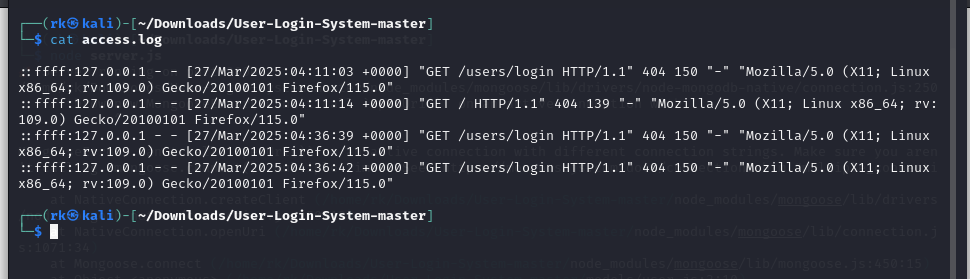
app.use(morgan('combined', { stream: accessLogStream }));



Restart the server  
Node servre.js



Verified Logging System:  
  
  




Ensured all **login attempts, errors, and requests** were logged.

### ****Task 3: Create a Simple Security Checklist****

**Security Best Practices Followed:**

**Validated all inputs** to prevent SQL/XSS attacks.

**Enabled HTTPS** for secure data transmission.

**Implemented password hashing & salting** using bcrypt.

**Added rate limiting** to prevent brute force attacks.

**Configured logging** to monitor security threats.

**nshots of Code, Logs, and Security Features Implemented)**

## ****Conclusion:****

Successfully **performed penetration testing** using **Nmap & Browser-Based Attacks**.

Implemented **security logging** with **Winston**.

Followed a **security checklist** ensuring best practices.