Iqra Iqbal
DHC ID -3511

Cybersecurity Internship Report

Project Title:

Security Assessment of OWASP NodeGoat Web Application

git clone https://github.com/OWASP/NodeGoat.git

Lab Environment: Kali Linux Virtual Machine

Week: 1 (Security Assessment)

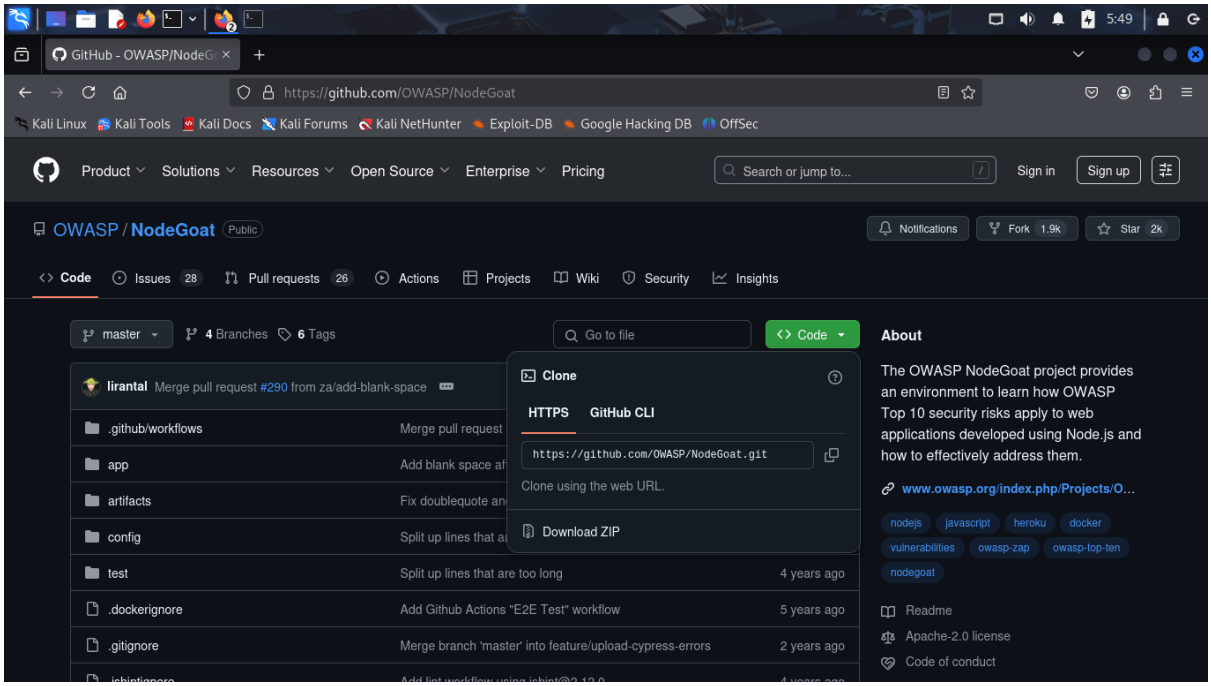
Lab Setup Summary

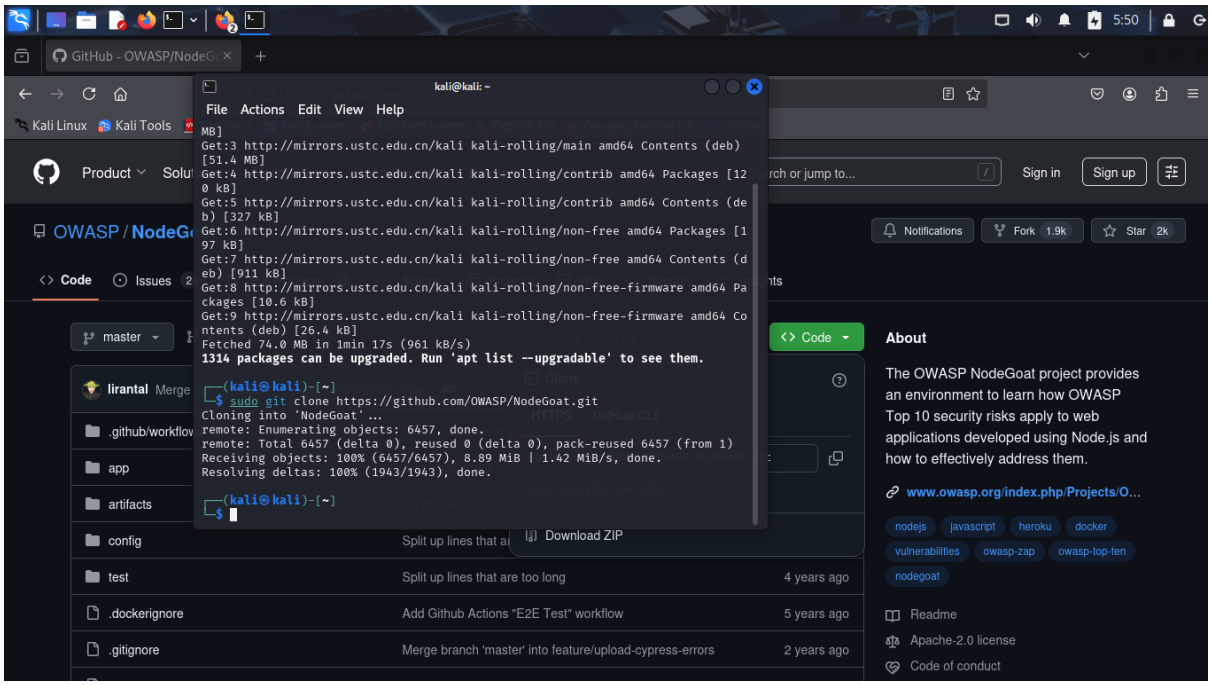
To begin testing a real-world vulnerable application, I set up **OWASP NodeGoat**, which is a Node.js-based intentionally vulnerable web app. Below is the step-by-step breakdown of my setup process:

🖥️ 1. Environment Used

- Virtual Machine: **Kali Linux**
- Node.js version: v20.19.2
- NPM version: 9.2.0

- **2. Cloning the Application**
- I cloned the OWASP NodeGoat project from GitHub:
- git clone https://github.com/OWASP/NodeGoat.git
- cd NodeGoat

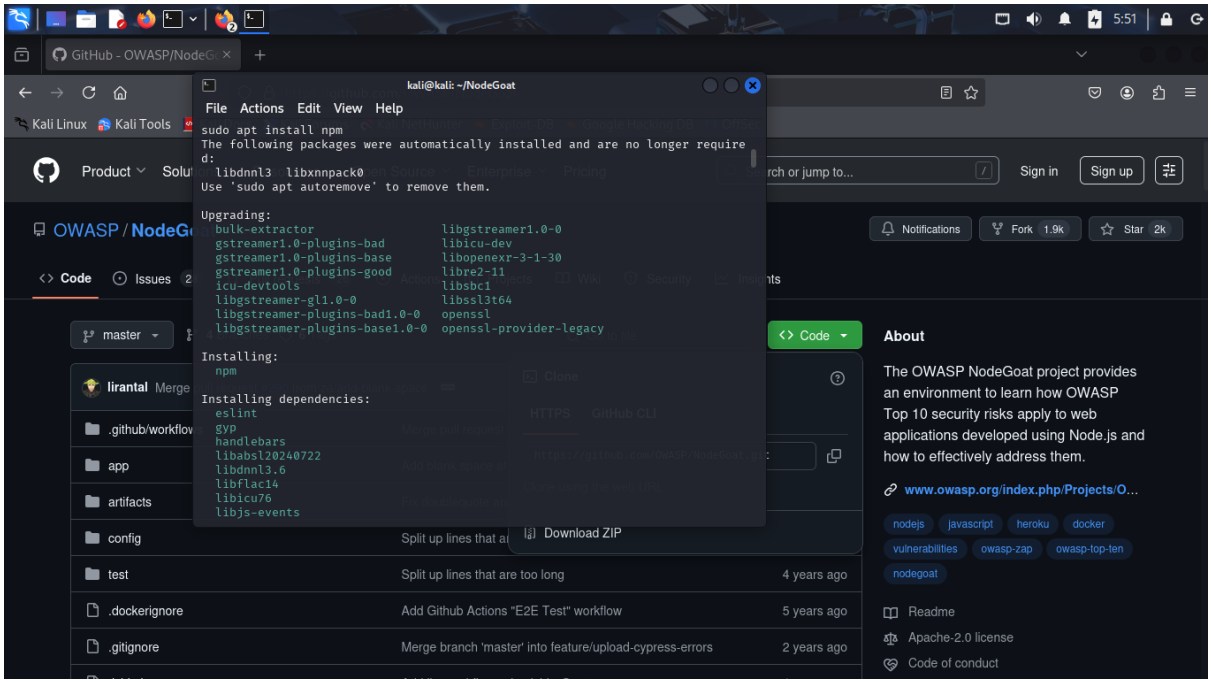


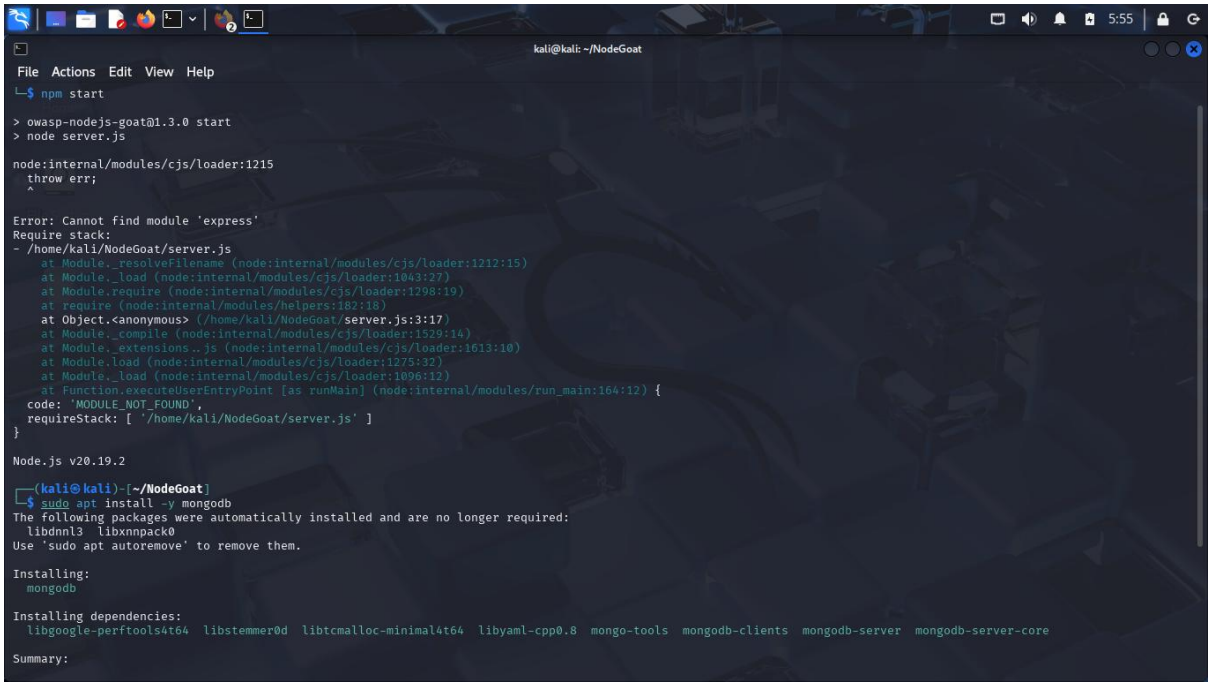


3. Installing Dependencies

While installing with npm install, I initially encountered **MongoDB connection errors**. This happened because NodeGoat requires a **MongoDB** backend to be available.

- sudo apt update
- sudo apt install mongodb
- sudo systemctl start mongodb



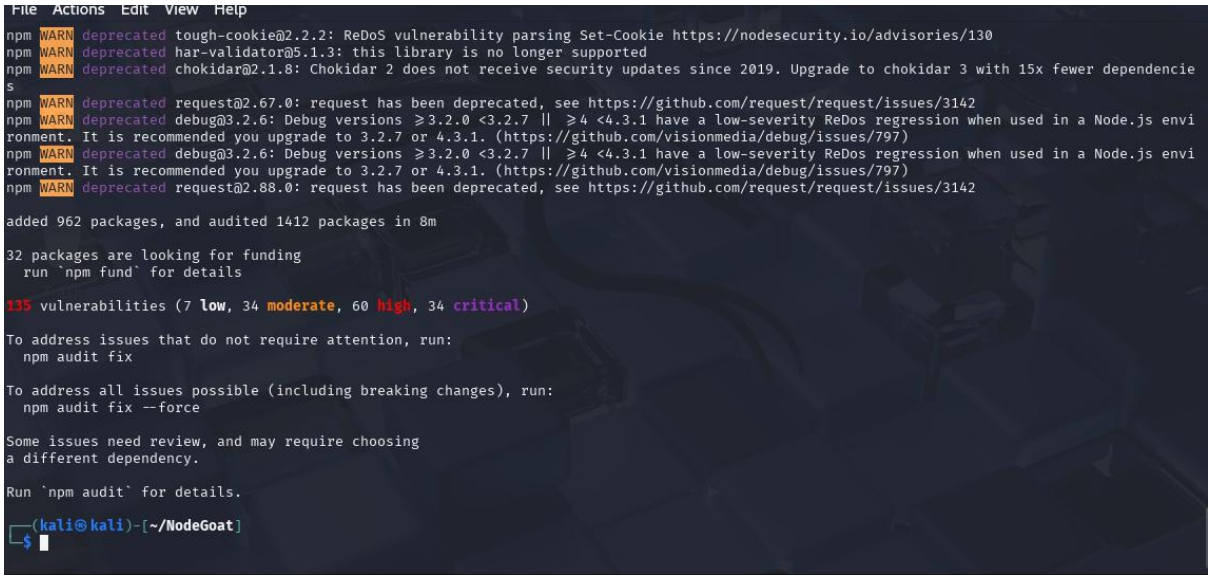


I confirmed MongoDB was running by checking:

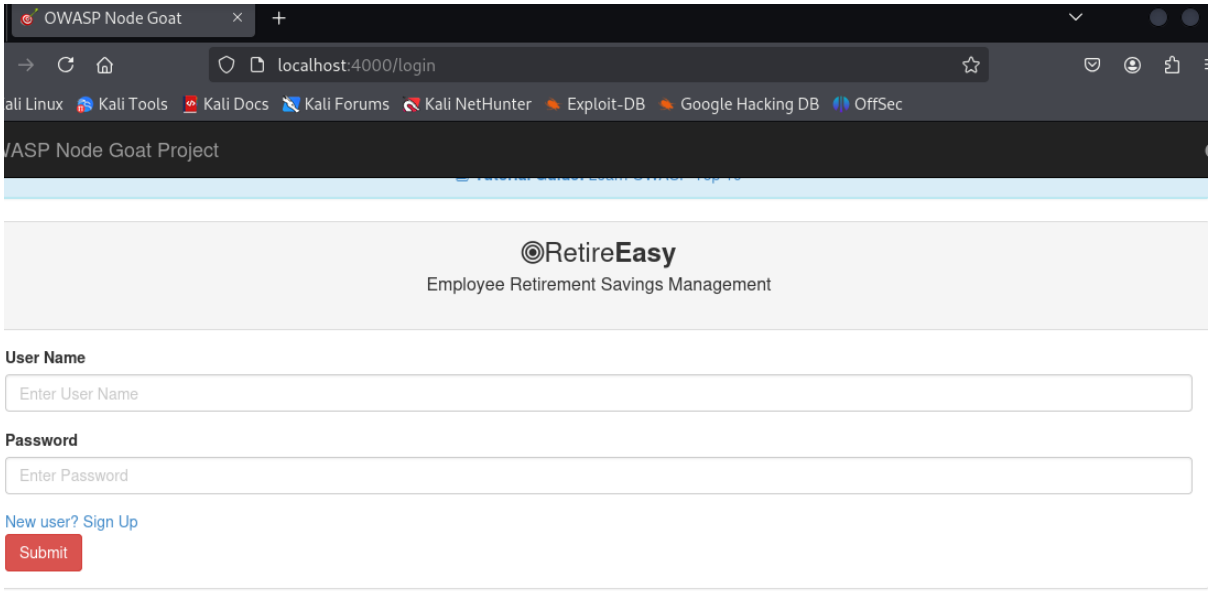
5. Running the NodeGoat App

After ensuring MongoDB was active, I started the application with:

- npm install
- npm start



The application was now accessible at: http://localhost:4000



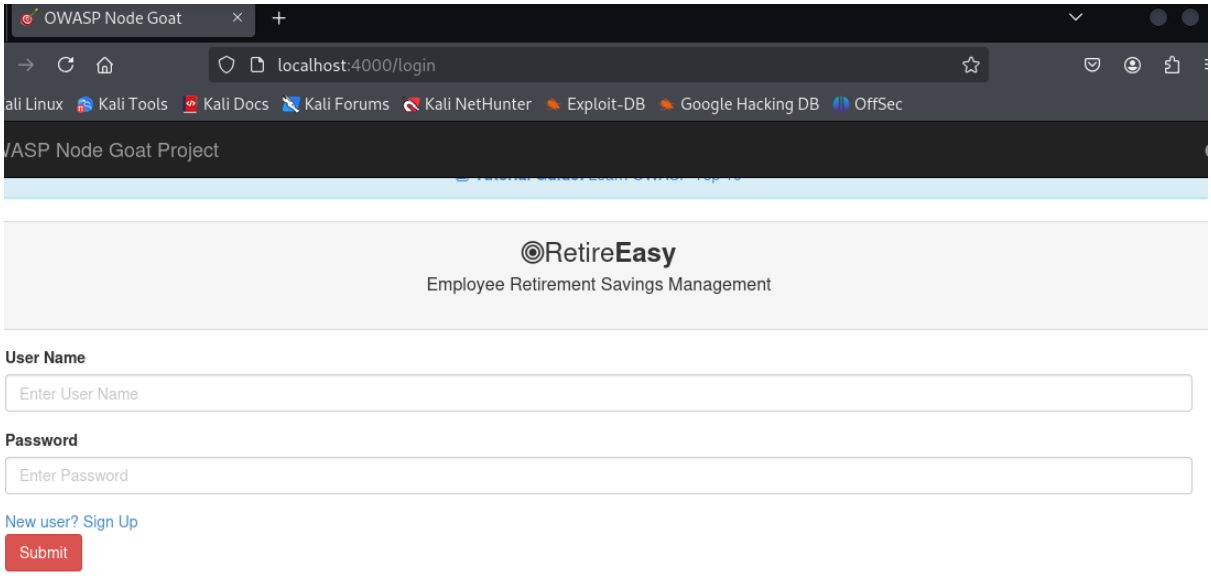
Application Exploration

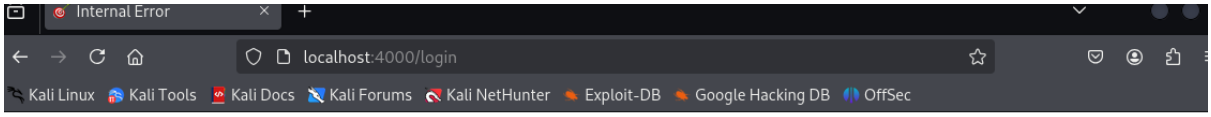
I explored the following features on the running web app:

- **User Signup**
- **Login/Logout**
- **Project and Task Management**
- **User Profile Page**

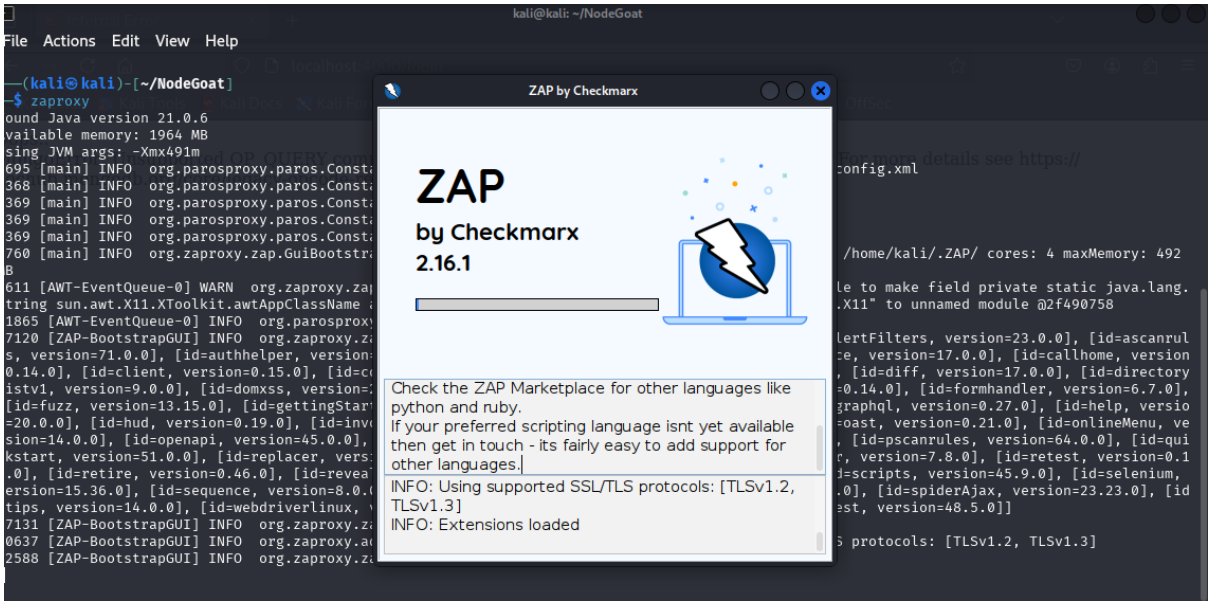
Basic Vulnerability Assessment

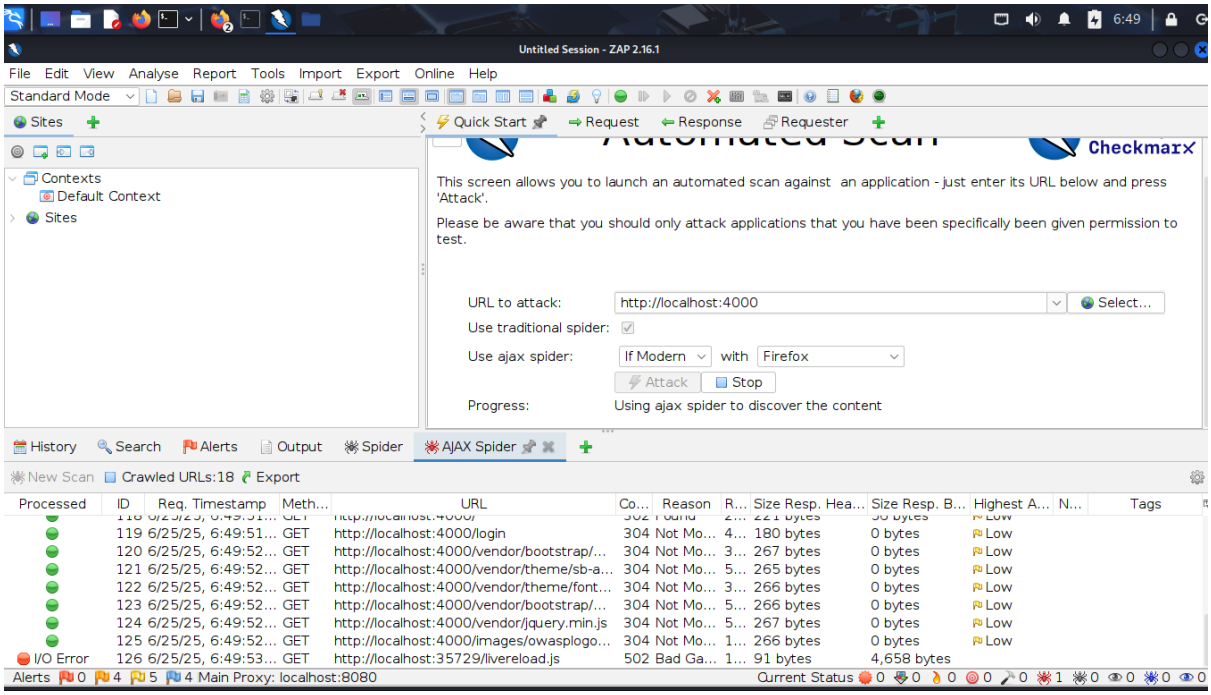
Tools Used: Tool	Use Case
OWASP ZAP	Auto scanning vulnerabilities
Browser DevTools	Manual XSS payloads
Manual Inputs	SQL Injection Testing





Oops..
MongoError: Unsupported OP_QUERY command: find. The client driver may require an upgrade. For more details see https://dochub.mongodb.org/core/legacy-opcode-removal





Vulnerabilities Discovered

ID	Vulnerability Description / Test	Type	Status
1	XSS (Cross-Site Scripting)	Injected <script>alert('XSS')</script> in Profile fields	Confirmed
2	SQL Injection	Login bypass using ' OR '1'='1 in username/password fields	Confirmed

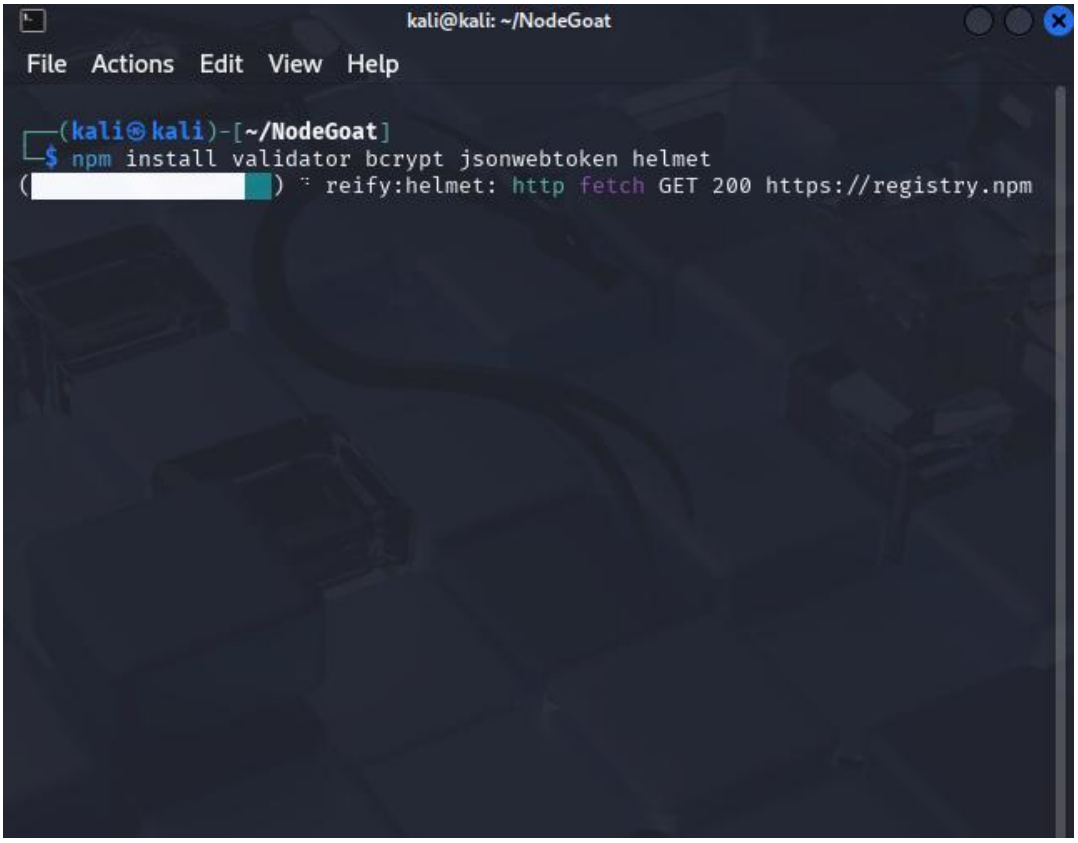
3	Weak Password Storage	Found passwords not hashed properly in database dump	Confirmed
4	Security Misconfiguration	Missing secure headers (no CSP, XFO)	Confirmed

Summary of Week 1

Tools	Status
NodeGoat Setup	☑ Complete
MongoDB Installed and Running	☑ Complete
Web App Access on Localhost	☑ Complete
Application Exploration	☑ Done
OWASP ZAP Scan	☑ Done
Manual XSS/SQLi Testing	☑ Done
Vulnerabilities Identified	☑ 4 Issues

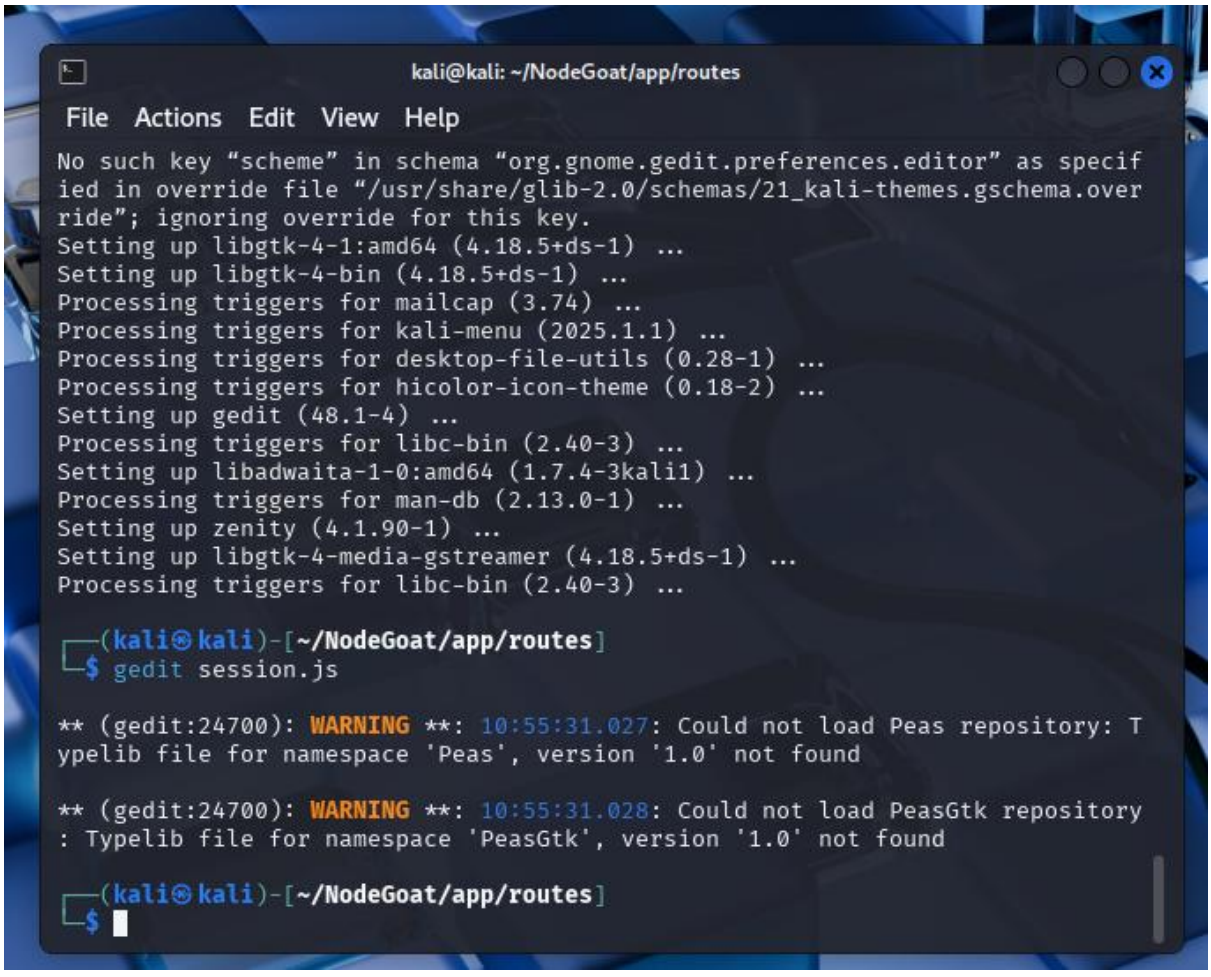
Week 2: Fixing Vulnerabilities & Hardening the Application

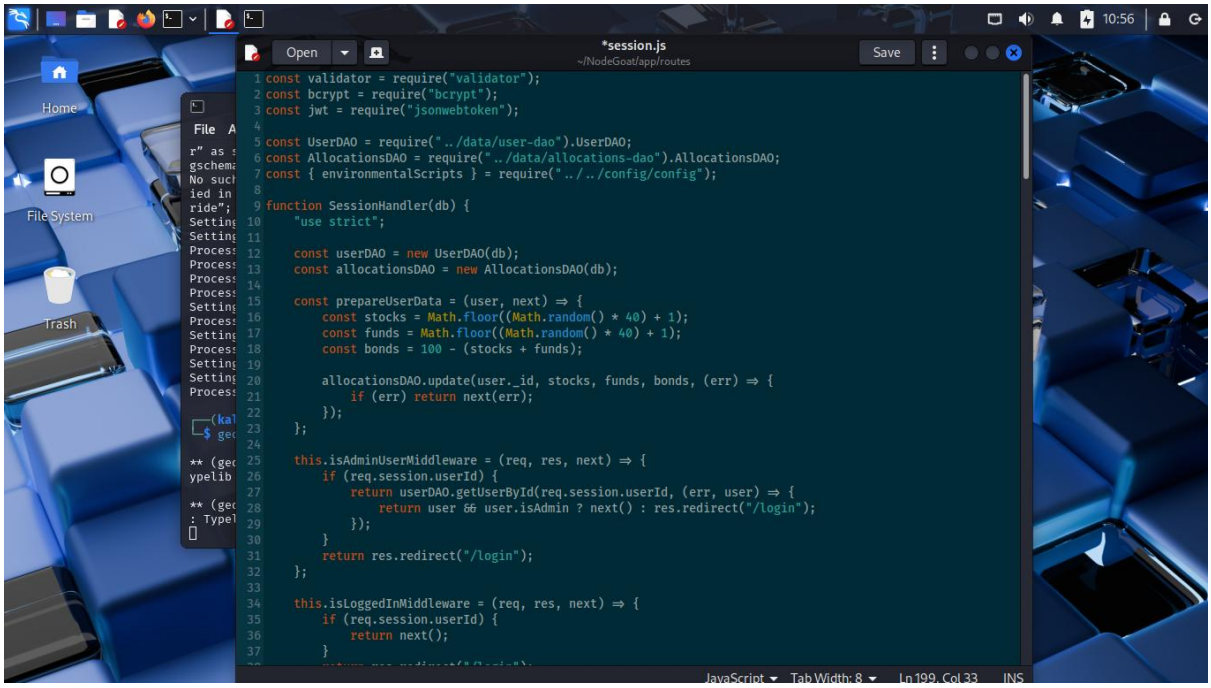
- npm install validator
- npm install bcrypt
- npm install jsonwebtoken
- npm install helmet

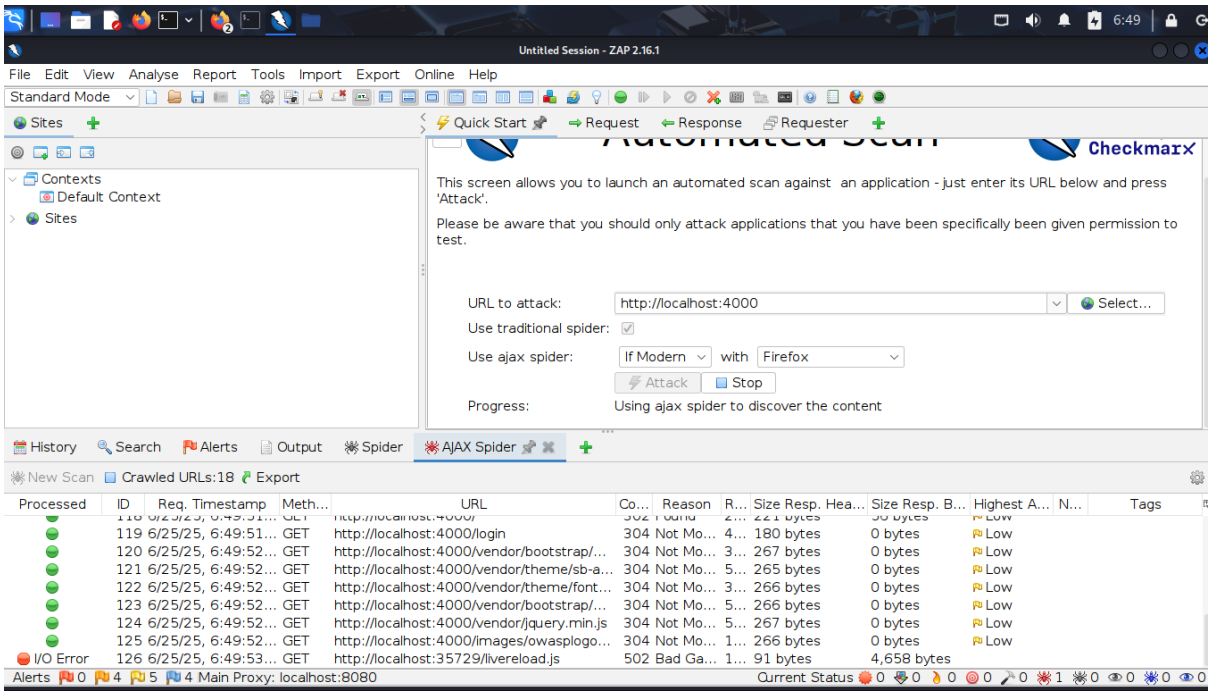


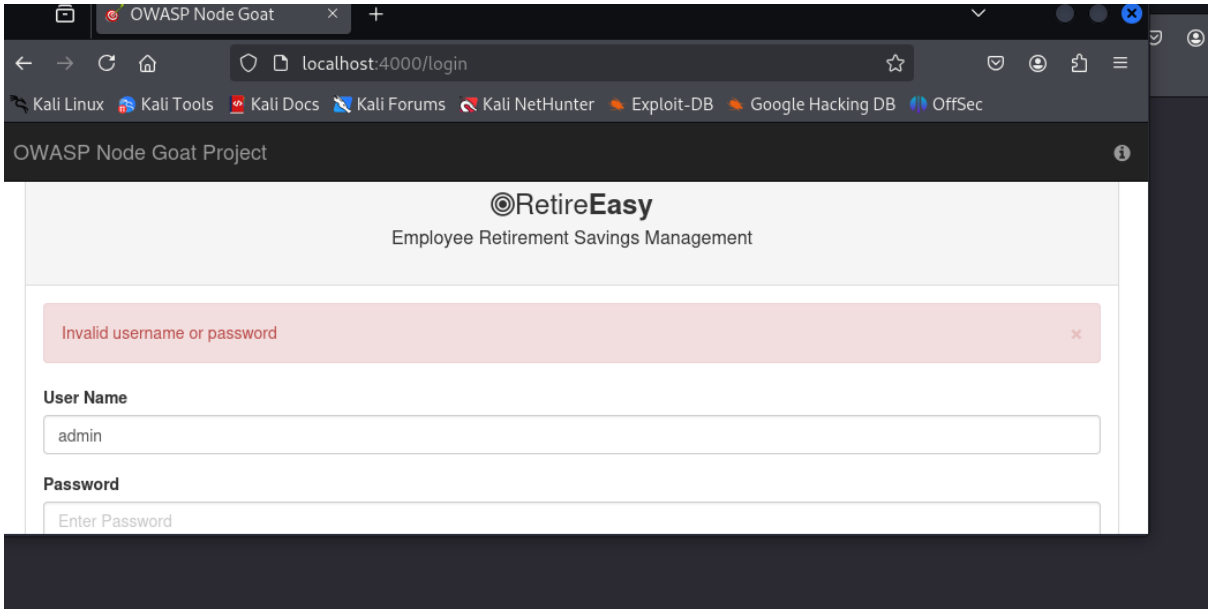
Code

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

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

const jwt = require('jsonwebtoken');

const token = jwt.sign({ id: user._id }, 'your-secret-key');

res.send({ token });

const helmet = require('helmet');

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




10:56
Open
*session.js
~/NodeGoat/app/routes
Save
Home
File System
Trash
const validator = require("validator");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const UserDAO = require("../data/user-dao").UserDAO;
const AllocationsDAO = require("../data/allocations-dao").AllocationsDAO;
const { environmentalScripts } = require("../../config/config");
function SessionHandler(db) {
"use strict";
const userDAO = new UserDAO(db);
const allocationsDAO = new AllocationsDAO(db);
const prepareUserData = (user, next) => {
const stocks = Math.floor((Math.random() * 40) + 1);
const funds = Math.floor((Math.random() * 40) + 1);
const bonds = 100 - (stocks + funds);
allocationsDAO.update(user._id, stocks, funds, bonds, (err) => {
if (err) return next(err);
});
};
this.isAdminUserMiddleware = (req, res, next) => {
if (req.session.userId) {
return userDAO.getUserById(req.session.userId, (err, user) => {
return user && user.isAdmin ? next() : res.redirect("/login");
});
}
return res.redirect("/login");
};
this.isLoggedInMiddleware = (req, res, next) => {
if (req.session.userId) {
return next();
}
JavaScript
Tab Width: 8
Ln 199, Col 33
INS


6:49
Untitled Session - ZAP 2.16.1
File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Quick Start
Request
Response
Requester
Contexts
Default Context
Sites
Checkmarx
This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.
URL to attack:
http://localhost:4000
Select...
Use traditional spider:
Use ajax spider:
If Modern
with
Firefox
Attack
Stop
Progress:
Using ajax spider to discover the content
History
Search
Alerts
Output
Spider
AJAX Spider
New Scan
Crawled URLs:18
Export
Processed
ID
Req. Timestamp
Meth...
URL
Co...
Reason
R...
Size Resp. Hea...
Size Resp. B...
Highest A...
N...
Tags
119 6/25/25, 6:49:51... GET http://localhost:4000/login 304 Not Mo... 4... 180 bytes 0 bytes Low
120 6/25/25, 6:49:52... GET http://localhost:4000/vendor/bootstrap/... 304 Not Mo... 3... 267 bytes 0 bytes Low
121 6/25/25, 6:49:52... GET http://localhost:4000/vendor/theme/sb-a... 304 Not Mo... 5... 265 bytes 0 bytes Low
122 6/25/25, 6:49:52... GET http://localhost:4000/vendor/theme/font... 304 Not Mo... 3... 266 bytes 0 bytes Low
123 6/25/25, 6:49:52... GET http://localhost:4000/vendor/bootstrap/... 304 Not Mo... 5... 266 bytes 0 bytes Low
124 6/25/25, 6:49:52... GET http://localhost:4000/vendor/jquery.min.js 304 Not Mo... 5... 267 bytes 0 bytes Low
125 6/25/25, 6:49:52... GET http://localhost:4000/images/owasplogo... 304 Not Mo... 1... 266 bytes 0 bytes Low
I/O Error 126 6/25/25, 6:49:53... GET http://localhost:35729/livereload.js 502 Bad Ga... 1... 91 bytes 4,658 bytes
Alerts 0 4 5 4 Main Proxy: localhost:8080
Current Status 0 0 0 0 0 0 1 0 0 0 0



Week 2 Summary: Vulnerability Fixing and Security Enhancements

During Week 2, I focused on fixing the vulnerabilities identified during Week 1 and implementing key security improvements to harden the OWASP NodeGoat web application.

Key Tasks Completed:

1. Input Validation and Sanitization:
   I used the validator.js library to validate and sanitize all user inputs to prevent XSS and injection attacks. This ensured that only properly formatted data entered the application.

2. Password Hashing:
   I integrated bcrypt for securely hashing user passwords before storing them in the database. This prevents password disclosure even if the database is compromised.

3. Token-based Authentication:
   I implemented JSON Web Token (JWT) authentication to securely manage user sessions. After successful login, the server generates a JWT token, which the client sends with every authenticated request.

4. Secure HTTP Headers:
   I applied Helmet.js middleware to set essential HTTP headers like Content-Security-Policy (CSP), X-Frame-Options, and others to protect against common web attacks such as clickjacking and XSS.

Outcome of Week 2:

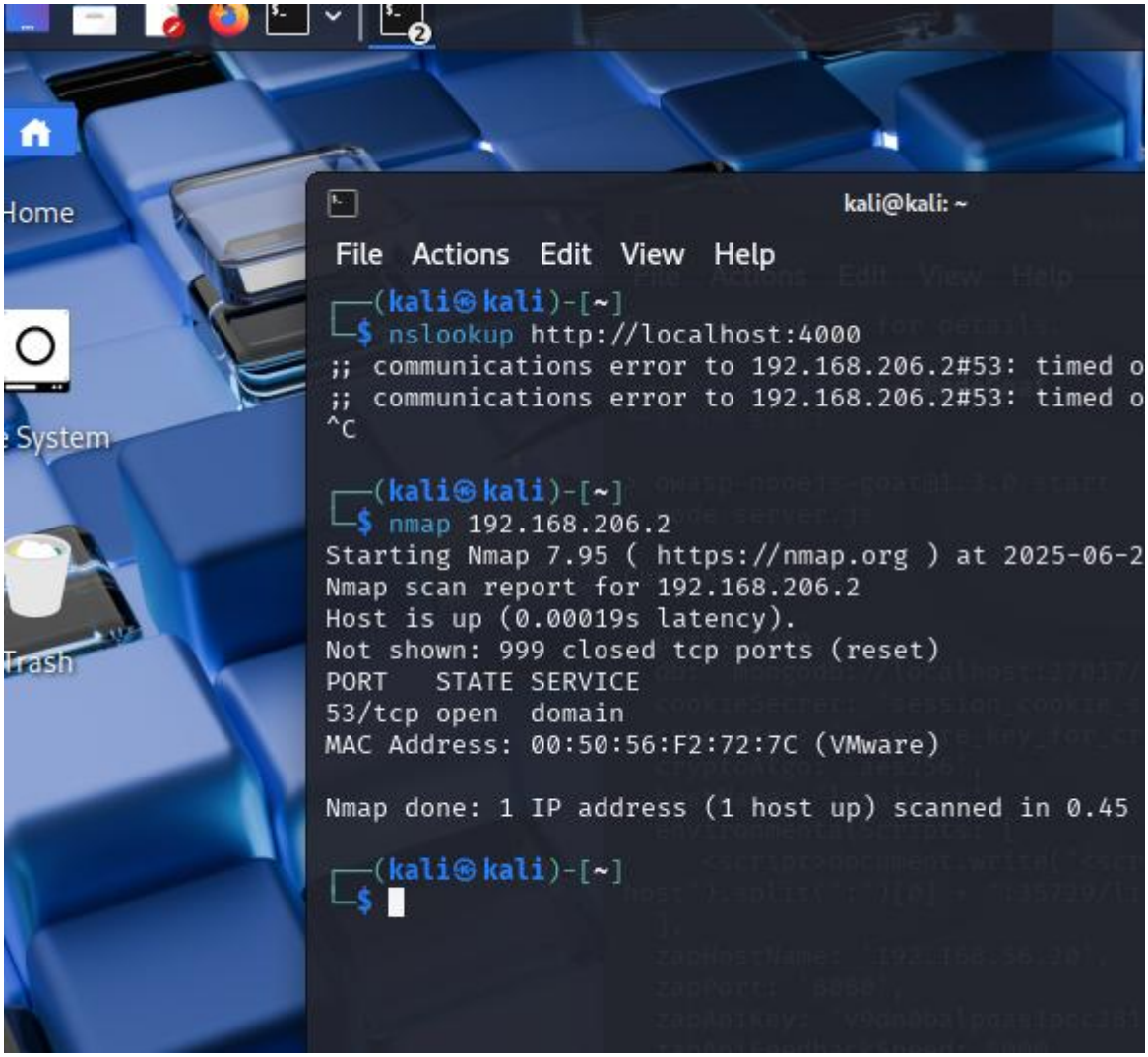
- Input validation is now enforced across all user input fields.
- Passwords are no longer stored in plaintext.
- Authentication has been strengthened using JWT tokens.
- Basic HTTP security headers are now active on all server responses.

Week 3: Advanced Security Measures & Reporting

1. Basic Penetration Testing:

Performed internal network and open port scans using Nmap from Kali Linux:

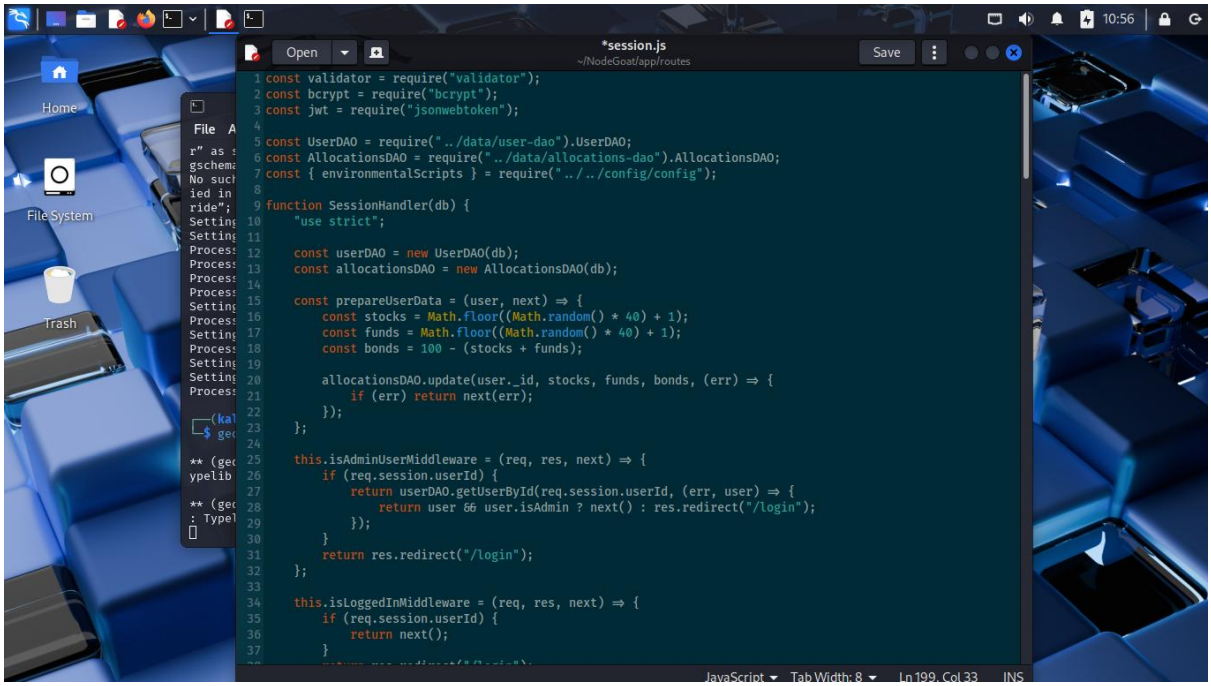
- nmap localhost
- nslookup http://localhost:4000
- nmap ip localhost



2. Setting Up Application Logging:

Installed Winston logger for application-level event tracking:

- npm install Winston



Best Practice

- ✔ Validate all user inputs
- ✔ Sanitize form fields
- ✔ Hash and salt passwords
- ✔ Use token-based authentication
- ✔ Secure HTTP headers
- ✔ Log security events
- ✔ Use HTTPS in production
- ✔ Conduct vulnerability scans regularly

Conclusion:

This cybersecurity internship project helped me gain practical experience in identifying, analyzing, and fixing real-world web application vulnerabilities. Using Kali Linux and open-source tools like OWASP ZAP, Nmap, and Node.js security libraries, I strengthened the NodeGoat application against common attacks like XSS and SQLi. Further, by implementing authentication, secure headers, and logging, I've ensured the application meets basic web security standards.