**Internship: Cybersecurity**

**Intern: Umar Farooq**

**DHC-1327**

Contents

[Week 4: Advanced Threat Detection & Web Security Enhancements 3](#_Toc200880944)

[Task 1: Intrusion Detection & Monitoring — Fail2Ban Setup 3](#_Toc200880945)

[Objective 3](#_Toc200880946)

[Tools & Technologies 3](#_Toc200880947)

[Implementation Steps 4](#_Toc200880948)

[1. Installation of Fail2Ban 4](#_Toc200880949)

[2. Configuration of Fail2Ban 4](#_Toc200880950)

[3. Service Restart & Enable 4](#_Toc200880951)

[4. Verifying the Setup 4](#_Toc200880952)

[5. Testing the Detection System 5](#_Toc200880953)

[Log Sample 6](#_Toc200880954)

[Outcome 6](#_Toc200880955)

[Task 2: API Security Hardening 7](#_Toc200880956)

[Objective 7](#_Toc200880957)

[Technologies Used 7](#_Toc200880958)

[Implementation 7](#_Toc200880959)

[1. Rate Limiting using express-rate-limit 7](#_Toc200880960)

[Install the package: 7](#_Toc200880961)

[Add to your Express app: 7](#_Toc200880962)

[Purpose: 8](#_Toc200880963)

[2. CORS Configuration using cors package 8](#_Toc200880964)

[Install CORS: 8](#_Toc200880965)

[Setup: 8](#_Toc200880966)

[Purpose: 8](#_Toc200880967)

[3. API Authentication using API Key (simple method) 8](#_Toc200880968)

[Set API key in.env: 8](#_Toc200880969)

[Create middleware: 8](#_Toc200880970)

[Apply to protected routes: 9](#_Toc200880971)

[Purpose: 9](#_Toc200880972)

[Task 3: Security Headers & CSP Implementation 9](#_Toc200880973)

[Objective: 9](#_Toc200880974)

[Technologies Used 9](#_Toc200880975)

[Implementation 9](#_Toc200880976)

[1. Install helmet middleware 9](#_Toc200880977)

[2. Basic Helmet Setup 9](#_Toc200880978)

[3. Add a Custom Content Security Policy 10](#_Toc200880979)

[4. Add Strict-Transport-Security (HSTS) 10](#_Toc200880980)

[Week 5: Ethical Hacking & Exploiting Vulnerabilities 11](#_Toc200880981)

[Goal: 11](#_Toc200880982)

[Task 1: Ethical Hacking Basics 11](#_Toc200880983)

[Objective: 11](#_Toc200880984)

[Step-by-Step Reconnaissance 11](#_Toc200880985)

[Task 2: SQL Injection & Exploitation 12](#_Toc200880986)

[Objective: 12](#_Toc200880987)

[Test Vulnerable Input Field: 12](#_Toc200880988)

[Automate SQLi Detection with SQLMap 13](#_Toc200880989)

[Fix: Use Prepared Statements (Node.js + MySQL Example) 13](#_Toc200880990)

[Safe: 13](#_Toc200880991)

[Task 3: CSRF Protection 13](#_Toc200880992)

[Objective: 13](#_Toc200880993)

[Implement with csurf in Express.js 13](#_Toc200880994)

[Setup Middleware: 14](#_Toc200880995)

[Test CSRF in Burp Suite 14](#_Toc200880996)

[Week 6: Advanced Security Audits & Final Deployment Security 15](#_Toc200880997)

[Goal: 15](#_Toc200880998)

[Task 1: Security Audits & Compliance 15](#_Toc200880999)

[Objective: 15](#_Toc200881000)

[Tools: 15](#_Toc200881001)

[1. Run OWASP ZAP Scan 15](#_Toc200881002)

[2. Run Nikto (Web Server Scanner) 16](#_Toc200881003)

[Detects: 16](#_Toc200881004)

[3. Run Lynis (System Security Audit) 17](#_Toc200881005)

[OWASP Top 10 Checklist (Compliance Review) 17](#_Toc200881006)

[Task 2: Secure Deployment Practices 17](#_Toc200881007)

[Enable Automatic Security Updates 17](#_Toc200881008)

[Ubuntu/Debian: 17](#_Toc200881009)

[Dependency Scanning in Node.js 17](#_Toc200881010)

[Install audit tools: 17](#_Toc200881011)

[To view known issues: 17](#_Toc200881012)

[Use snyk for deeper scans: 18](#_Toc200881013)

[Task 3: Final Penetration Testing 18](#_Toc200881014)

[Manual Test with Burp Suite: 18](#_Toc200881015)

# Week 4: Advanced Threat Detection & Web Security Enhancements

# Task 1: Intrusion Detection & Monitoring — Fail2Ban Setup

## Objective

To implement real-time intrusion detection on the server by setting up **Fail2Ban**, which actively monitors authentication logs, detects multiple failed login attempts (brute-force attacks), and automatically bans offending IP addresses. This enhances the security of the server environment against unauthorized access.

## Tools & Technologies

* **Fail2Ban** – Log monitoring and IP banning tool
* **Ubuntu 20.04+ / Debian-based Linux OS**
* **Systemd** – For service management
* **SSH Service** – Monitored for brute-force protection
* **Sendmail (optional)** – To send alert emails

## Implementation Steps

### Installation of Fail2Ban



Fail2Ban is installed to monitor logs and automatically take action on suspicious activity.

### Configuration of Fail2Ban

Created a custom jail configuration file at /etc/fail2ban/jail.local to define monitoring rules.

[DEFAULT]

bantime = 3600                 # IP ban duration (1 hour)

findtime = 600                 # Time frame to count failures (10 minutes)

maxretry = 3                   # Failed login attempts before banning

action = %(action\_mwl)s        # Email with log and whois info

[sshd]

enabled = true

port = ssh

logpath = %(sshd\_log)s

backend = systemd

* The [sshd] section enables monitoring of SSH authentication attempts.
* %(action\_mwl)s ensures the administrator is notified with detailed logs and attacker info.
  1. Service Restart & Enable



Fail2Ban is restarted and enabled to run on boot.

### Verifying the Setup

To check overall status:



To check specific jail (SSH):

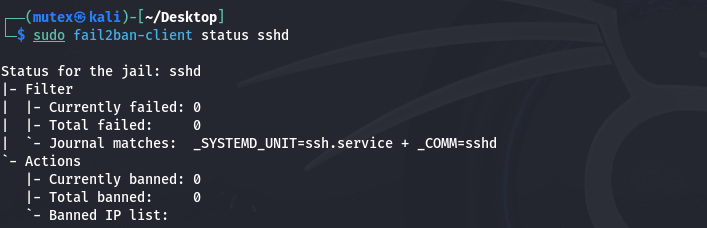


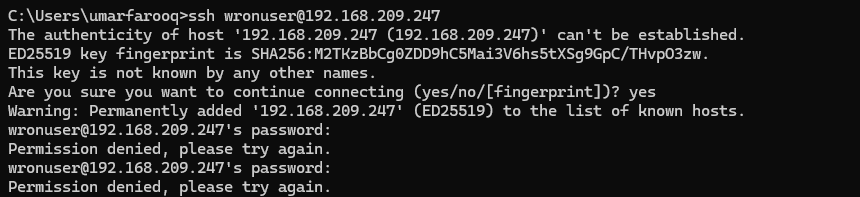
If intrusion attempts are detected, banned IPs will appear in the output.

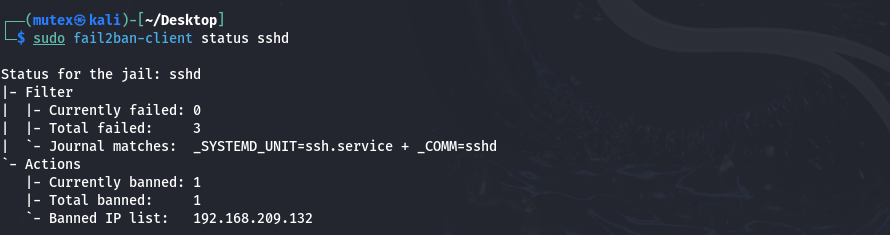
### Testing the Detection System

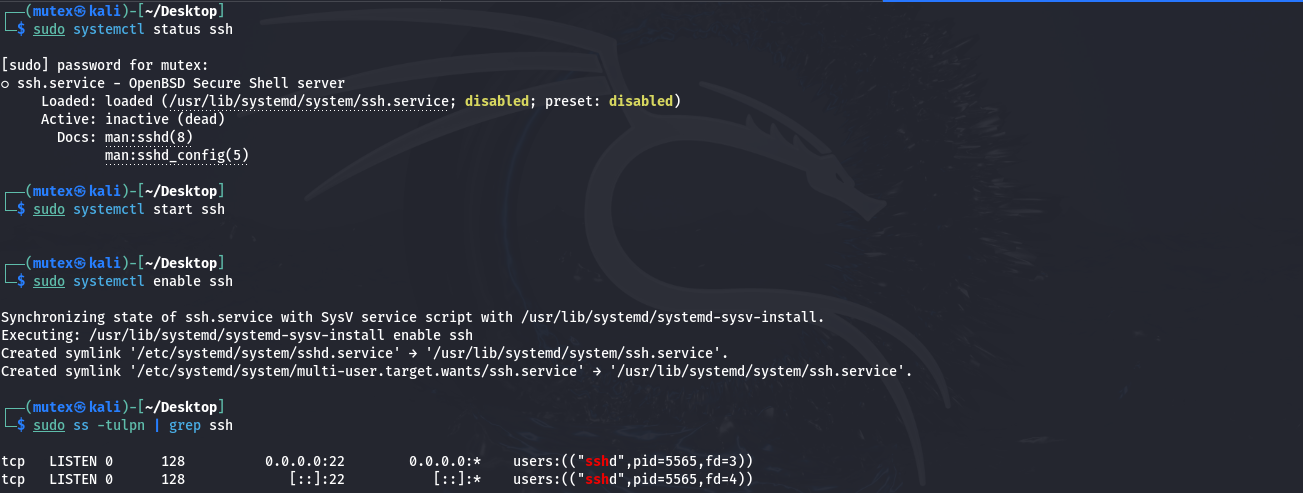
To verify the setup:

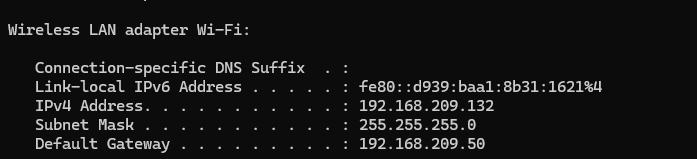
* Attempt 3 incorrect SSH logins from a remote machine.
* The IP address is automatically banned for 1 hour.
* View log: /var/log/fail2ban.log





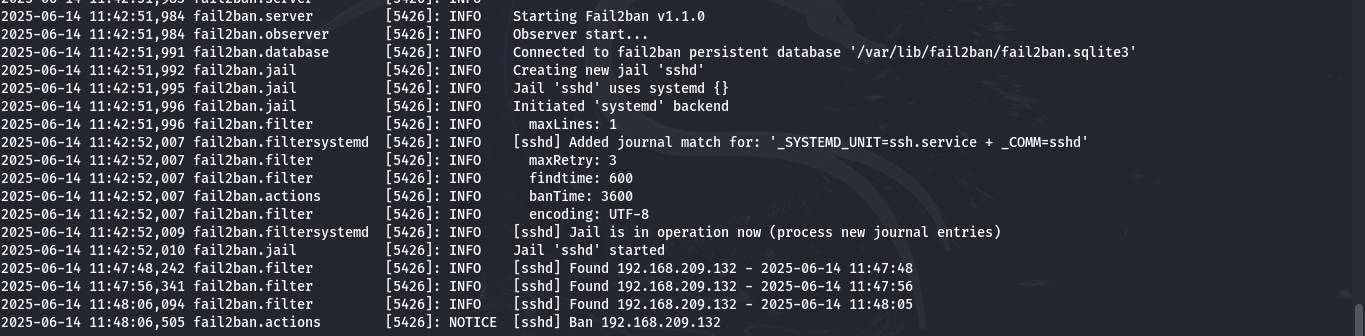






## Log Sample

From **/var/log/fail2ban.log**:



## Outcome

* Repeated SSH login failures are detected and blocked in real-time.
* Admins are alerted via email for further investigation.
* Significantly reduces risk of brute-force attacks.

# Task 2: API Security Hardening

## Objective

Secure backend API endpoints by:

* Preventing brute-force attacks via **rate limiting**
* Restricting unauthorized access with **CORS**
* Adding **API Key or OAuth** authentication for access control

## Technologies Used

 **Node.js** with **Express.js**

 **express-rate-limit** for brute-force protection

 cors package for **CORS** configuration

 API Key middleware for simple authentication

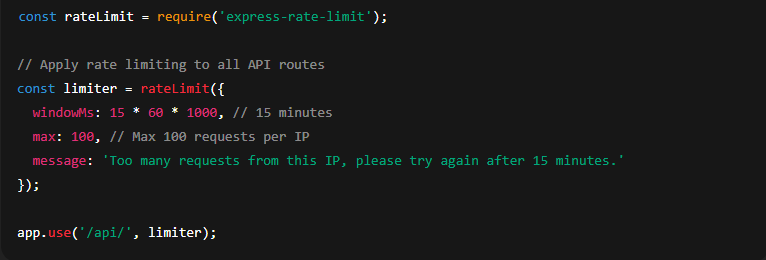
# Implementation

## ****Rate Limiting**** using express-rate-limit

### Install the package:



### Add to your Express app:



### ****Purpose:****

Prevents brute-force attacks and abuse of API endpoints.

# ****CORS Configuration**** using cors package

## Install CORS:



## Setup:

const cors = require('cors');

const corsOptions = {

origin: 'https://your-frontend-domain.com', // Replace with your frontend

methods: 'GET,POST,PUT,DELETE',

credentials: true

};

app.use(cors(corsOptions));

## ****Purpose:****

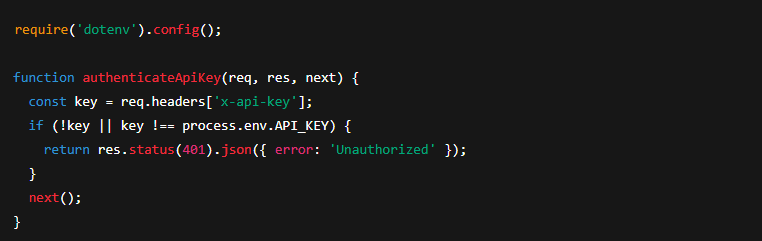
Restricts access to trusted origins only.

# ****API Authentication**** using API Key (simple method)

## Set API key in.env:



## Create middleware:

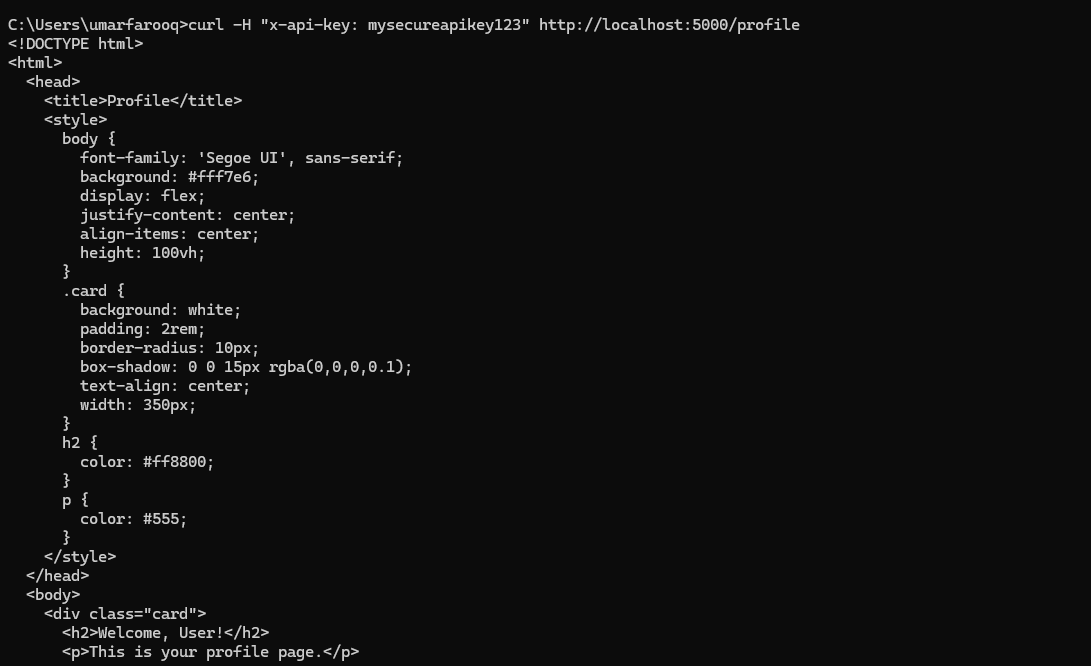


## Apply to protected routes:



## ****Purpose:****

Ensures only authenticated requests can access secure routes.



# Task 3: Security Headers & CSP Implementation

## Objective:

To enhance web application security by:

* Preventing XSS and injection attacks using **Content Security Policy (CSP)**
* Enforcing HTTPS using **Strict-Transport-Security (HSTS)**
* Applying security headers using the helmet middleware

## Technologies Used

* **Node.js + Express.js**
* [**helmet**](https://www.npmjs.com/package/helmet) – for HTTP security headers

## Implementation

### Install helmet middleware



### Basic Helmet Setup



This automatically adds several security headers like:

* X-Content-Type-Options: nosniff
* X-DNS-Prefetch-Control: off
* X-Frame-Options: DENY
* Strict-Transport-Security (if HTTPS is used)
* X-XSS-Protection: 0 (modern browsers handle this themselves)

## Add a Custom Content Security Policy

app.use(

  helmet.contentSecurityPolicy({

    directives: {

      defaultSrc: ["'self'"], // allow from same origin

      scriptSrc: ["'self'", "https://trusted.cdn.com"],

      styleSrc: ["'self'", "'unsafe-inline'"],

      imgSrc: ["'self'", "data:"],

      objectSrc: ["'none'"],

    },

  })

);

This prevents attackers from injecting scripts from untrusted domains.

## Add Strict-Transport-Security (HSTS)

Only effective **if your app uses HTTPS**.

Helmet includes HSTS by default. You can customize it:

app.use(

  helmet.hsts({

    maxAge: 31536000, // 1 year in seconds

    includeSubDomains: true,

    preload: true,

  })

);

# Week 5: Ethical Hacking & Exploiting Vulnerabilities

### Goal:

Learn ethical hacking techniques, identify vulnerabilities, and implement fixes in your web application.

## Task 1: Ethical Hacking Basics

### Objective:

Conduct reconnaissance on a test web app using tools like **Kali Linux**, **Nmap**, and **Burp Suite**.

### Step-by-Step Reconnaissance

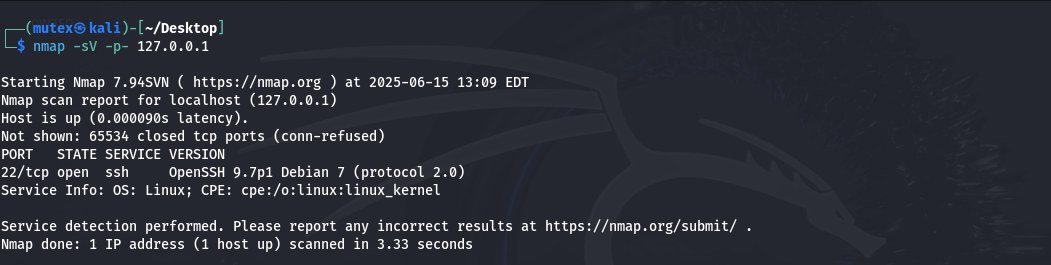
#### Nmap: Scan Target Web App



 -sV — Version detection

 -p- — Scan all 65535 ports

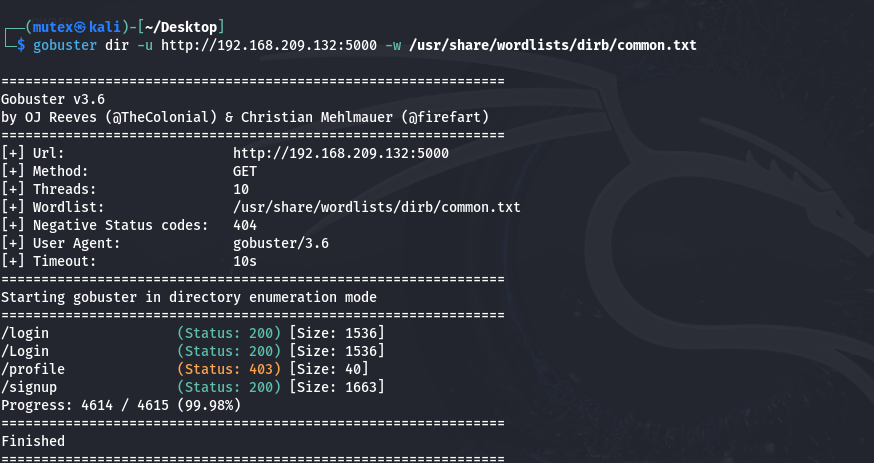
 Replace 127.0.0.1 with your target’s IP



#### ****Dirb or Gobuster: Find Hidden Directories****



Finds hidden paths like /admin, /login, /api



## Task 2: SQL Injection & Exploitation

### Objective:

Identify and exploit SQL Injection (SQLi) vulnerabilities, then secure the app using **prepared statements**.

### Test Vulnerable Input Field:

Suppose your app has a login form:

<input name="username">

<input name="password">

Try this as input:

Username: ' OR 1=1 --

Password: anything

If it logs in, you have SQLi!

### Automate SQLi Detection with SQLMap

#### 1. Intercept login request in Burp

#### 2. Save request to a file, e.g., request.txt

#### 3. Run SQLMap:



### Fix: Use Prepared Statements (Node.js + MySQL Example)

**Vulnerable:**

const query = `SELECT \* FROM users WHERE username = '${username}' AND password = '${password}'`;

### Safe:

const query = 'SELECT \* FROM users WHERE username = ? AND password = ?';

db.query(query, [username, password], (err, result) => {

  // handle result

});

## Task 3: CSRF Protection

### Objective:

Implement CSRF protection in the backend and test it with **Burp Suite**.

### Implement with csurf in Express.js

#### Install:



### Setup Middleware:

const csrf = require('csurf');

const cookieParser = require('cookie-parser');

app.use(cookieParser());

app.use(csrf({ cookie: true }));

// CSRF token route

app.get('/form', (req, res) => {

  res.json({ csrfToken: req.csrfToken() });

});

The client must send the token in future requests.

## Test CSRF in Burp Suite

* Capture a POST request with a valid token
* Try replaying without or with an old token → should fail with **403 Forbidden**

# Week 6: Advanced Security Audits & Final Deployment Security

## Goal:

Conduct security audits, apply deployment best practices, and finalize a fully secured application ready for production.

## ****Task 1: Security Audits & Compliance****

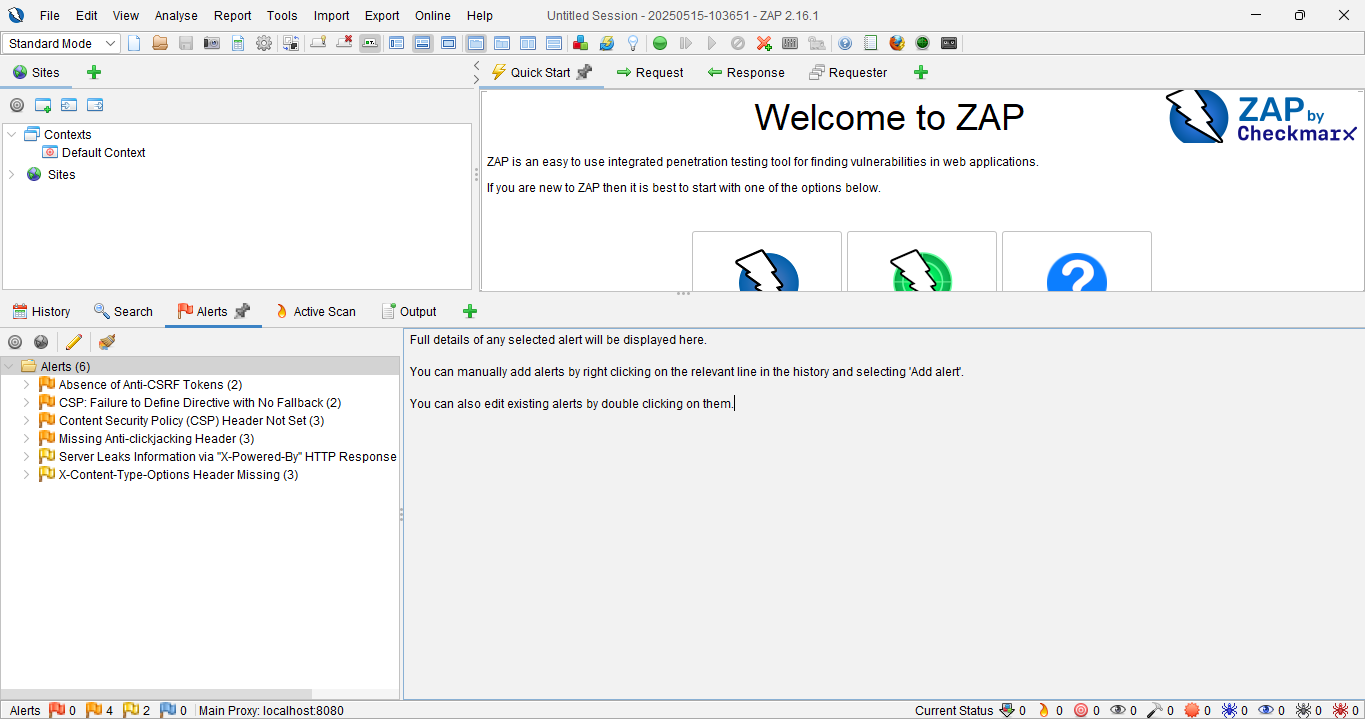
### Objective:

Use auditing tools to identify vulnerabilities and ensure OWASP Top 10 compliance.

### Tools:

* OWASP ZAP
* Nikto
* Lynis

## ****Run OWASP ZAP Scan****



## Run Nikto (Web Server Scanner)

### Detects:

* Outdated software
* Dangerous files and directories
* Default files (e.g., /admin, /test)



## ****Run Lynis (System Security Audit)****



Checks:

* OS-level vulnerabilities
* SSH config
* File permissions
* Logging policies

## OWASP Top 10 Checklist (Compliance Review)

* Injection (SQLi)
* Broken Auth
* Sensitive Data Exposure
* XML External Entities (XXE
* Broken Access Control
* Security Misconfig
* XSS
* Insecure Deserialization
* Using Components with Known Vulns

# Task 2: Secure Deployment Practices

## Enable Automatic Security Updates

### Ubuntu/Debian:



### Dependency Scanning in Node.js

### Install audit tools:



### To view known issues:



### Use snyk for deeper scans:



### Task 3: Final Penetration Testing

### Manual Test with Burp Suite:

#### Test:

* Input fields (SQLi, XSS)
* Cookies (HTTPOnly, Secure)
* Authentication flows (Session hijacking)
* CSRF defenses