Cybersecurity Internship Report

Intern Name: Ehmaan Shafqat

Project Title: Strengthening Security Measures for a Web Application

Submitted to: Faizan Khan

Date: 26 June, 2025

Week 4: Advanced Threat Detection & Web Security Enhancements

Repository Overview

This repository contains the implementation of **Advanced Threat Detection (ATD)** and **Web Security Enhancements** for securing APIs and web applications. It includes:

- **Real-time intrusion detection** (Fail2Ban, OSSEC).
- **API security hardening** (Rate limiting, CORS, OAuth2).
- **Security headers & CSP implementation** (Helmet.js, HSTS).

Setup

Prerequisites

- Node.js (v16+)
- npm / yarn
- Linux server (for Fail2Ban/OSSEC)
- Environment variables (`.env` file)

Installation

```
git clone https://github.com/your-repo/web-security-enhancements.git
cd web-security-enhancements
npm install
cp .env.example .env  # Configure API keys & secrets
```

Task Implementation Details

1. Intrusion Detection & Monitoring

Fail2Ban Setup:

- Monitors logs for brute-force attacks (SSH, HTTP).
- **Configuration:**

```
# /etc/fail2ban/jail.local
```

```
[sshd]
enabled = true
maxretry = 3
bantime = 1h
```

- **Verify Status:**

```
sudo systemctl status fail2ban
sudo fail2ban-client status sshd
```

OSSEC (Host-Based IDS)

- Analyzes logs for suspicious activity.
- **Key Commands:**

```
sudo ossec-control start
tail -f /var/ossec/logs/alerts.log
```

2. API Security Hardening

Rate Limiting (express-rate-limit)

- Prevents brute-force & DDoS attacks.
- **Code Snippet:**

```
const limiter = rateLimit({ windowMs: 15 * 60 * 1000, max: 100 });
app.use("/api/", limiter);
```

CORS Restrictions

- Whitelists trusted domains only.
- **Example:**

```
app.use(cors({ origin: ["https://trusted-domain.com"] }));
```

Authentication (OAuth2/API Keys)

- **API Key Middleware:**

```
if (req.headers["x-api-key"] !== process.env.API_KEY) {
  return res.status(403).json({ error: "Unauthorized" });
}
```

3. Security Headers & CSP

Helmet.js (Secure Headers)

- **Implementation:**

```
app.use(helmet());
app.use(helmet.hsts({ maxAge: 63072000 }));
```

Content Security Policy (CSP)

- Blocks XSS & data injection.
- **Example Policy:**

```
directives: {
  defaultSrc: ["'self'"],
  scriptSrc: ["'self'", "trusted-cdn.com"],
}
```

Deliverables Report

Task	Status	Details
Fail2Ban Setup	Done	Blocks IPs after 3 failed SSH attempts.
OSSEC Log Monitoring	Done	Alerts on suspicious activity.
API Rate Limiting	Done	100 requests/IP/15 mins.
CORS Restrictions	Done	Whitelisted `trusted-domain.com`.

Task	Status	Details
API Key / OAuth2	Done	Key-based & OAuth2 authentication.
Security Headers (HSTS/CSP)	Done	Enforced via Helmet.js.

Testing & Validation

1. **Rate Limiting Test:**

```
curl -X GET http://localhost:3000/api/data -H "x-api-key: YOUR_KEY"
# (Repeat 100+ times to test blocking)
```

2. **CSP Violation Check:**

- Open browser console, check for CSP errors.

3. **Fail2Ban Test:**

```
ssh user@server -o PasswordAuthentication=yes
# Enter wrong password 3 times to trigger ban.
```

Repository Structure

```
.
├── README.md            # This documentation
├── app.js               # Main API with security middleware
├── .env.example         # Template for environment variables
├── fail2ban-config/     # Fail2Ban jail.local example
└── ossec-config/        # OSSEC alert rules
```

Summary

This project successfully implements:

Real-time threat detection (Fail2Ban, OSSEC).

Hardened API security (Rate limiting, CORS, Auth).

Security headers & CSP (Mitigates XSS, data leaks).