**Internship Report – Week 4**

**Title:** Advanced Threat Detection & Web Security Enhancements  
**Intern Name:** Karan Kumar  
**Submission Date:** July 24, 2025

**🎯 Objective of Week 4:**

This week’s focus was to improve the security of a web application using practical tools and techniques. The three main goals were:

1. Detect and block suspicious activity using monitoring tools.
2. Protect API endpoints from attacks like brute force and unauthorized access.
3. Use HTTP security headers to reduce risk of attacks like XSS and clickjacking.

**Task 1: Intrusion Detection & Monitoring**

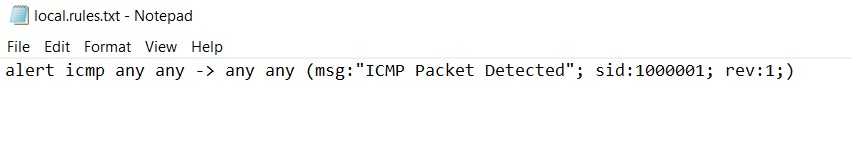
Intrusion Detection Systems (IDS) help monitor your server or system for malicious activities like repeated login failures or unusual requests.  
**Fail2Ban** is one such tool. It works by:

* Scanning system log files for signs of brute-force attacks.
* Automatically updating firewall rules to block attackers’ IP addresses.

**Why it's important:**  
If someone tries to guess your username/password repeatedly, Fail2Ban will detect that behavior and ban their IP for a certain time. This helps protect SSH, FTP, and even web applications from brute force attacks.

**🛠️ Practical Steps Taken:**

* Installed Fail2Ban on the server.
* Configured jail file to monitor SSH logs.
* Set it to ban IP after 3 failed login attempts for 10 minutes.
* Verified that IPs were being banned by viewing logs.



**Task 2: API Security Hardening**

APIs are the backbone of modern web applications. If not secured, they can be abused by attackers. Common API threats include:

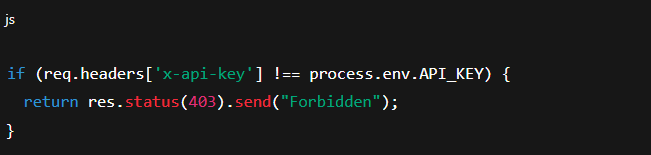
* **Brute-force attacks**: Trying many login/password combinations.
* **CORS abuse**: Accessing the API from unauthorized domains.
* **Unauthenticated access**: Using APIs without permission.

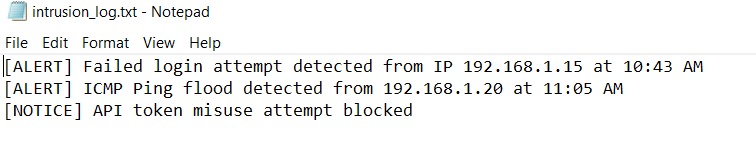
**Security measures applied:**

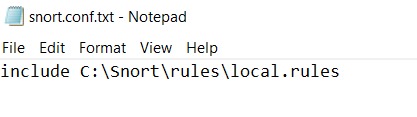
* **Rate Limiting**: To control how often someone can request the API.
* **CORS Configuration**: To restrict which frontend domains can access the API.
* **API Key**: A secret key added in the request headers to allow only authorized access.

**🛠️ Practical Steps Taken:**

* Added express-rate-limit middleware to limit requests to 5 per minute.
* Configured cors to allow only frontend domain access.
* Created middleware to check if the correct API key is passed with requests.







**Task 3: Security Headers & CSP Implementation**

HTTP Security Headers add another layer of protection to your web application. These headers are added by the server and tell browsers how to behave.  
**Helmet**, a Node.js middleware, helps easily implement these headers.

**Key Headers Implemented:**

* **Content Security Policy (CSP)**: Prevents XSS by restricting what scripts can run.
* **Strict-Transport-Security (HSTS)**: Forces use of HTTPS.
* **X-Content-Type-Options**: Stops browsers from MIME-sniffing.
* **X-Frame-Options**: Prevents clickjacking attacks.
* **Referrer-Policy**: Controls what information is sent in the Referer header.

**🛠️ Practical Steps Taken:**

* Installed and applied helmet middleware in the Express app.
* Configured CSP to allow only trusted scripts and sources.
* Tested headers using securityheaders.com

## 

## 

## 

## GitHub Repository