**DEVELOPERS HUB INTERNSHIP**

**Week 4: Advanced Threat Detection & Web Security Enhancements**

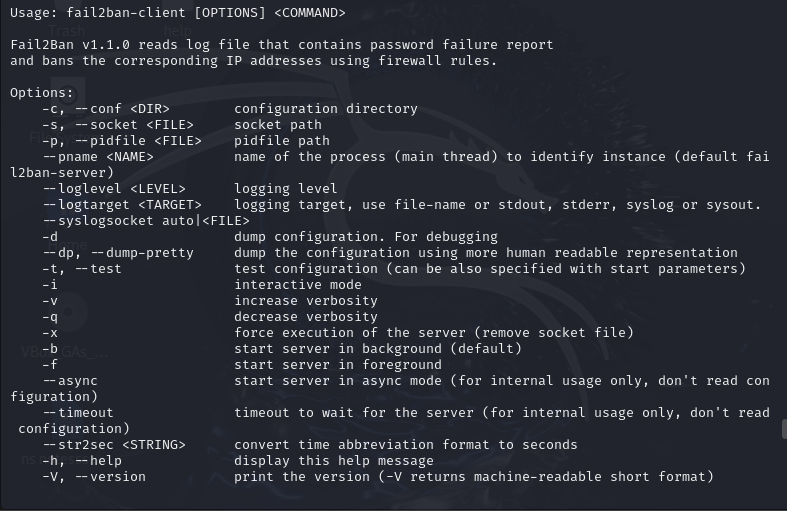
1. ***Intrusion Detection & Monitoring***

**In this task, I used tools Fail2Ban and OSSEC to set up real time monitoring. Now before the monitoring and setup, it was important to know the practicality and work of these tools.**

**Fail2Ban protects from Brute force attacks, after too many failed login attempts by an IP it directly blocks it. It monitors the log file for this. For example, the auth.log, as it says it’s more of a reactive tool and depends on your firewall rules for blocking.**

**OSSEC is a Host based intrusion detection system, monitors the system activity, it just doesn’t focus on the log file, in fact tries to analyze the logs from multiple sources and attempts like unauthorized file changes, abnormal logins, rootkit installation are recorded and alerted by this tool.**

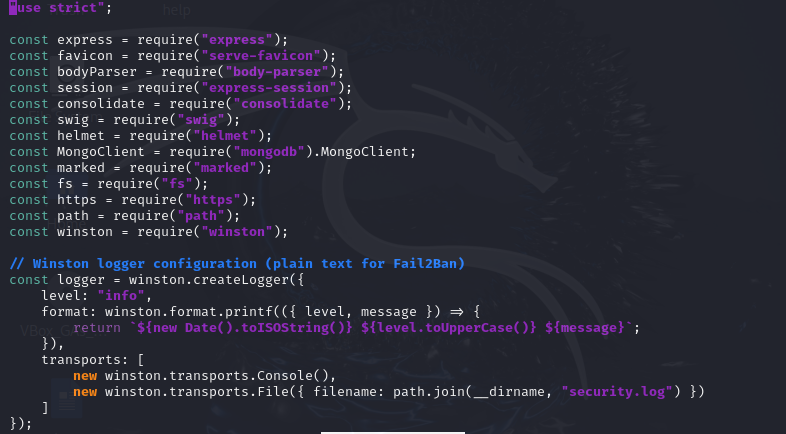
**INSTALLED FAIL2BAN:**

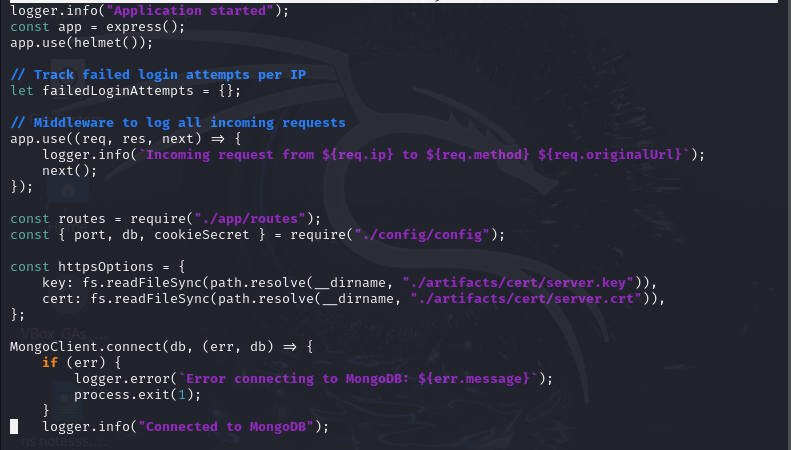


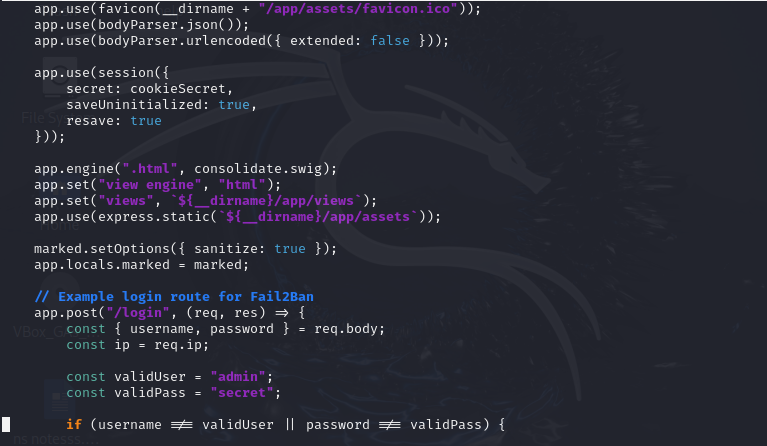
**Now that we have the tool in our system, we will first add Good logs in our NodeGoat, although NodeGoat already has a log file but for Fail2ban to detect and block according to rules we need proper logs, detailed you can say.**

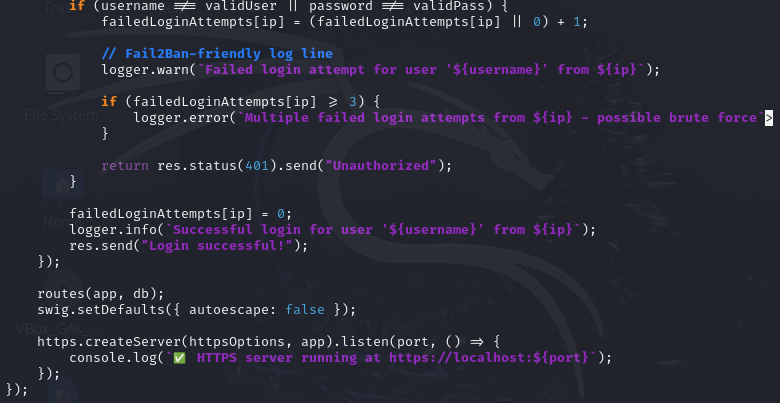
**Modifying the server.js as we have required Winston in this file, which is a library used for logging activities. Our security.log file is the file that gets all the logs printed in it. We make all the logs go into this file because fail2ban needs a log file to compare and get the logs. Created a failed Login Attempts object in server.js keyed by IP. Each failed login increments the counter for that IP. When counter exceeds 3 or is 3, log a warning/error about possible brute-force.**

**Server.js code:**

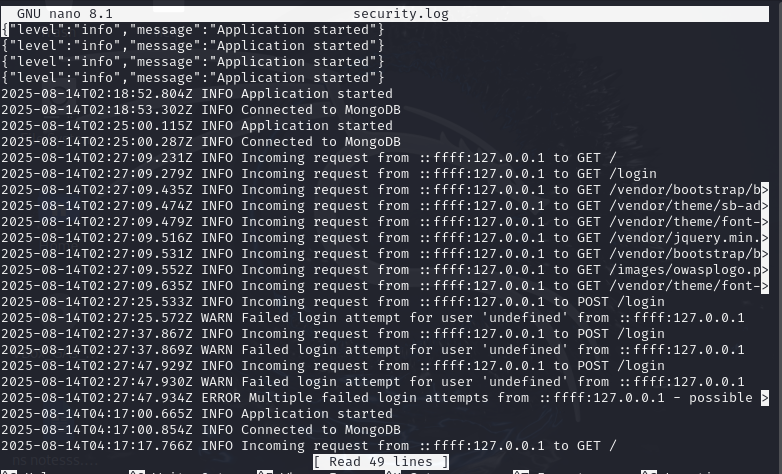




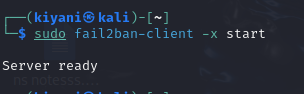




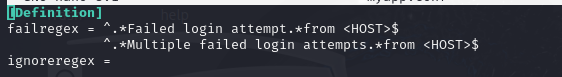
**SECURITY.LOG:**



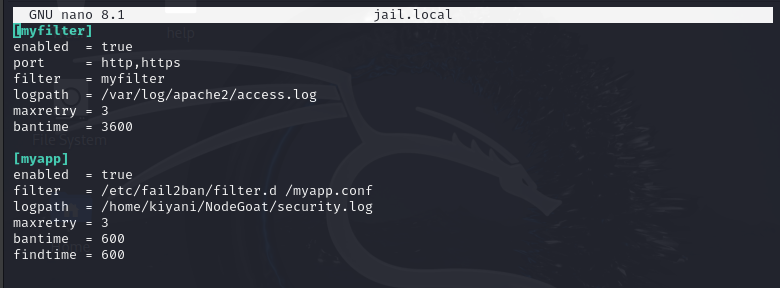
**Now moving on to the next step, setting up the Fail2Bin for it to access the logs and alerting:**



**Next step was to first set up the filter file, this is basically that contains the failure regex, so it will be compared with the security.log file and show eventually how many failures occurred and matched with the security log file:**

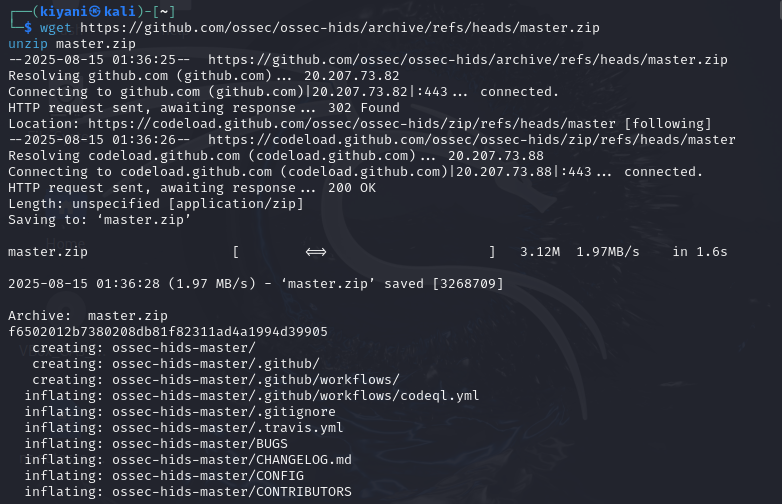


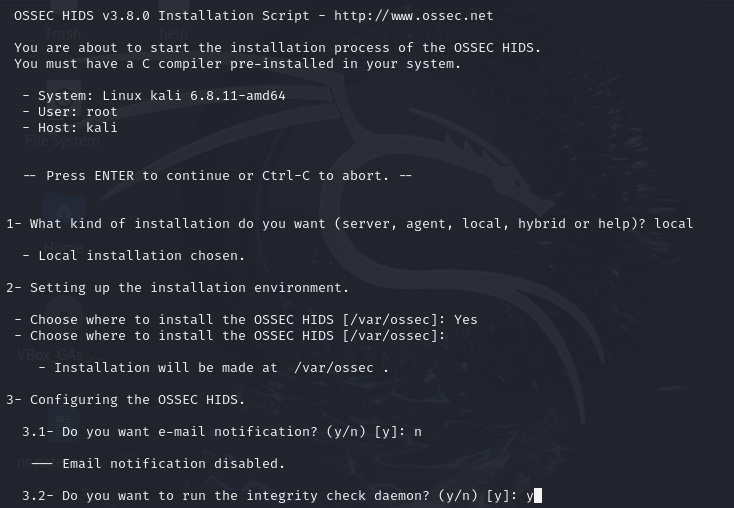
**Next we were supposed to set up the jail, I named it my app and set up the jail in jail.d directory named as jail.local, Now for some reason my app jail was not being recognized by fail2ban, so I just added the whole content of that jail into the SSHD jail and hence it basically lists the blocked up, attempts of login and all, fail2ban doesn’t really work well with custom apps :**

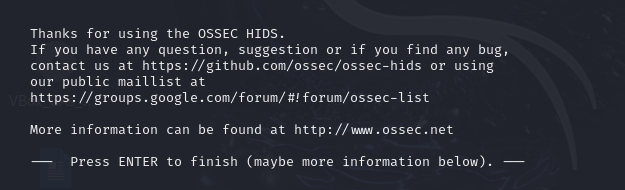


**I then wanted to interact with ossec, although the task goal was achieved by fail2ban but OSSEC was something I wanted to try as well:**

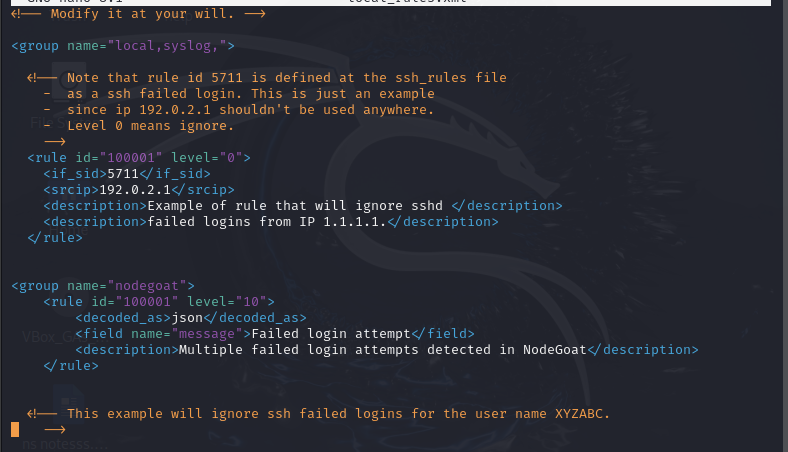
**Download OSSEC:**



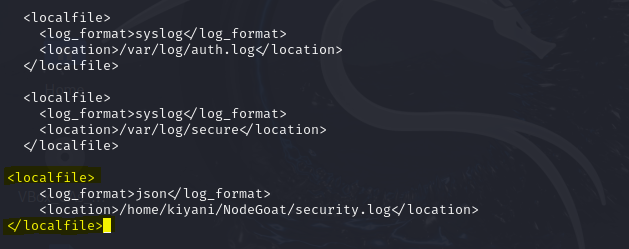




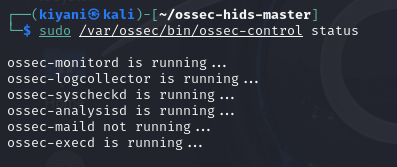
**We first add the rule in the local rules xml file of ossec, which contains our NodeGoat logging format and messages and we set a level here for example 10 as level 0 means to ignore:**



**Then in ossec\_config we add a local file block whoch contains the block of security.log file of ours.**



**Ensuring my ossec is working:**



1. ***API security hardening:***

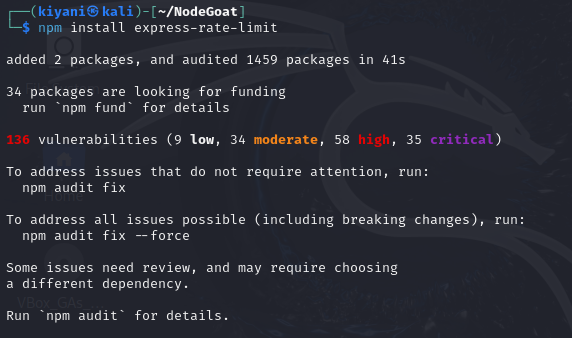
**Here we are dealing with APIs which basically acts as a middle man between the client and the server.**

**● *Apply rate limiting using express-rate-limit to prevent brute-force attacks.***

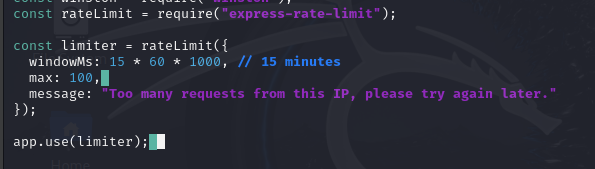
**As attackers can send a lot of requests to your application or server causing a brute force attack and disrupting your tasks so you apply rate limiting.**

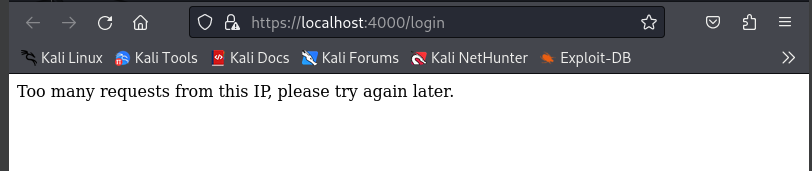
**Installation of express-rate-limit:**





**We add the require statement in our server code and add the code snippet for rate limiting, in 15 mins we are allowing 100 reqs, if requests exceed that we show the message below:**



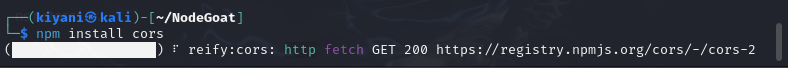


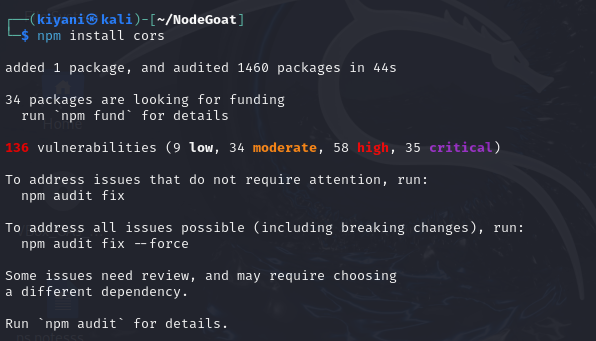
* ***Configure CORS (Cross-Origin Resource Sharing)***

**CORs restricts which domains can access your API, if we don’t implement this, any site can make a request to your API, which can be a great security hole.**

**When a request goes from the browser to the server, in the header it also states what domain or site is making the request. The server responds with CORs header that basically can tell what domains are allowed to see if match, what methods are allowed, what headers, so u see if it matches the rules, if it does the browser lets the request pass through or else not.**

**Installation of CORS:**

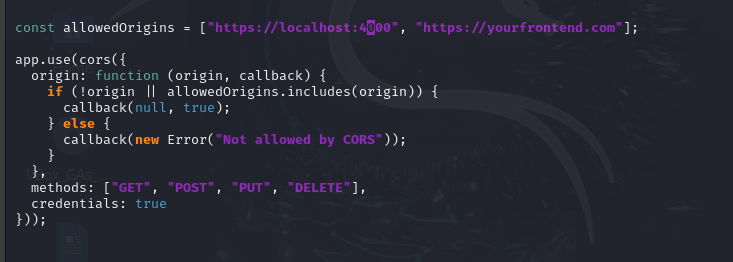




**Implementation of CORs, by first requiring cors in our server.js code:**



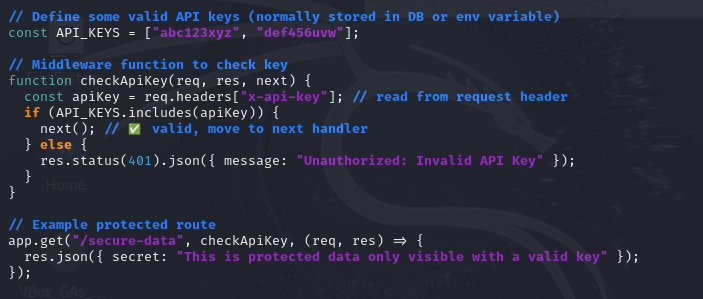
**The allowed origins show that these domains are allowed to make a request. If the site trying to connect is in our allow origin list then we are supposed to allow it, or else reject it.**



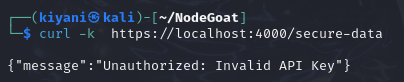
* ***Secure APIs***

**Secure APIs are important as not everyone should be able to access the APIs of our website just because the URL is public, only trusted users should. Hence if a client wants to access our APIs they send this key that we define in the request header and server checks if its valid or not.**

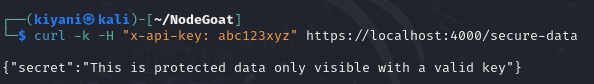
**In our implementation a client tries to access secure data but before directly running the route, server first checks the API key if its valid or not.**



**Without the valid API key:**



**With the valid API key:**



1. ***Security Headers & CSP Implementation:***

Content Security Policy (CSP):

It limits where our application loads pages, and also protects from XSS attacks.

We start by using helmet which already is in our server code, it is a middleware (what sits between the request and response ) that sets quite a few security http security headers for us and makes it difficult for our application to be broken into.

Next we set our the CSP logic in our server code, This means:

* No inline scripts, No scripts from other sites, No external fonts/styles unless you explicitly allow



HSTS (HTTP STRICT TRANSPORT SECURITY):

It tells the browser to always use https for a particular domain for specific seconds. Prevents downgrade attacks (like forcing HTTP).

