

Advanced Threat Detection & Web Security Enhancements

This report is the week 4 task of the Implementing advanced threat detection mechanisms and enhancing the security of the web applications.

Developers Hub Cybersecurity Internship Task July 2025

Week: 4

Internee name:

Athar Imran

<https://github.com/atharimran728/Web-Application-Security-Strengthening/tree/main>

GOAL: Secure API with rate-limiting and authentication mechanisms.

Implement security headers with proper configuration.

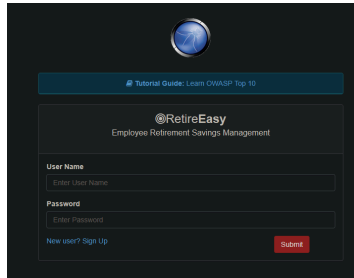
❖ Setting up a Web Application:

(Alternatively, follow the official tutorial: <https://github.com/OWASP/NodeGoat>)

1. Download and install Docker from the official source. After finishing the installation, ensure that it was installed correctly.
2. Now clone NodeGoat: `git clone https://github.com/OWASP/NodeGoat.git`
3. Got the NodeGoat directory and built Docker image using: `docker-compose build`.
This command reads the `Dockerfile` and `docker-compose.yml` to build the necessary images for the application and the database.

4. Run the application using `docker-compose up`. And access at <http://localhost:4000/>. Now our application starts listening on http port

4000:



1- Intrusion Detection & Monitoring:

For this task we will be using Fail2Ban, because configuring OSSEC on current requisites is much complex. Fail2Ban will run within your WSL2 Ubuntu environment and monitor the logs generated by your Dockerized Nodegoat application. You'll need to configure your Nodegoat application to output logs to a location that Fail2Ban can access.

a. Integrating Docker terminal with remote log file:

We need to access or otherwise create a file where Nodegoat authentication logs will be saved, so that we can provide Fail2Ban with those logs as input.

We will use the Direct File Logging technique, which involves modifying Nodegoat's Node.js application code to explicitly write logs to a file, and then mounting that file's directory as a Docker volume to the WSL2 host.

The NodeGoat file that deals with authentication routes is [Session.js](#) under `app/routes`. I have modified it so that authentication events are now logged:

Importing required modules:

```
5 } = require("../../config/config");
6 const fs = require('fs'); // Import the file system module
7 const path = require('path'); // Import the path module
8
```

Defining the path of the log file:

```

15
16 // Define log file path. Use an environment variable for flexibility.
17 // Default to a path within the container that can be mounted as a volume.
18 const logFilePath = process.env.LOG_FILE_PATH || '/app/logs/auth.log';
19
20 // Ensure the log directory exists
21 const logDir = path.dirname(logFilePath);
22 if (!fs.existsSync(logDir)) {
23   fs.mkdirSync(logDir, { recursive: true });
24 }
25
26 // Create a write stream for authentication logs. 'a' flag appends to the file.
27 const logStream = fs.createWriteStream(logFilePath, { flags: 'a' });
28

```

Capturing the IP of the client:

```

73 // Capture the client's IP address
74 const clientIp = req.ip || req.connection.remoteAddress || 'UNKNOWN_IP';
75

```

Log the error:

```

81 if (err) {
82   if (err.noSuchUser) {
83     // Log failed login attempt to file for Fail2Ban
84     const logMessage = `${new Date().toISOString()} - Failed login: No such user '${userName}' from IP ${clientIp}\n`;
85     logStream.write(logMessage);
86     console.error(logMessage.trim()); // Also log to stderr for Docker's default logs
87
88     return res.render("login", {
89       userName: userName,
90       password: "",
91       loginError: invalidUserNameErrorMessage,
92       environmentalScripts
93     });
94   } else if (err.invalidPassword) {
95     // Log failed login attempt to file for Fail2Ban
96     const logMessage = `${new Date().toISOString()} - Failed login: Invalid password for user '${userName}' from IP ${clientIp}\n`;
97     logStream.write(logMessage);
98     console.error(logMessage.trim()); // Also log to stderr for Docker's default logs
99
100     return res.render("login", {
101       userName: userName,
102       password: "",
103       loginError: invalidPasswordErrorMessage,
104       environmentalScripts
105     });
106   } else {
107     return next(err);
108   }

```

Log the success entry:

```

111 // If login is successful
112 const successLogMessage = `${new Date().toISOString()} - Successful login for user '${userName}' from IP ${clientIp}\n`;
113 logStream.write(successLogMessage);
114 console.log(successLogMessage.trim());
115
116 // Fix the problem by regenerating a session in each login
117 // by wrapping the below code as a function callback for the method req.session.regenerate()
118 // i.e:
119 // `req.session.regenerate(() => {})`
120 req.session.regenerate(() => { // Regenerate session ID on successful login
121   req.session.userId = user._id;
122   return res.redirect(user.isAdmin ? "/benefits" : "/dashboard");
123 });
124 });
125 };

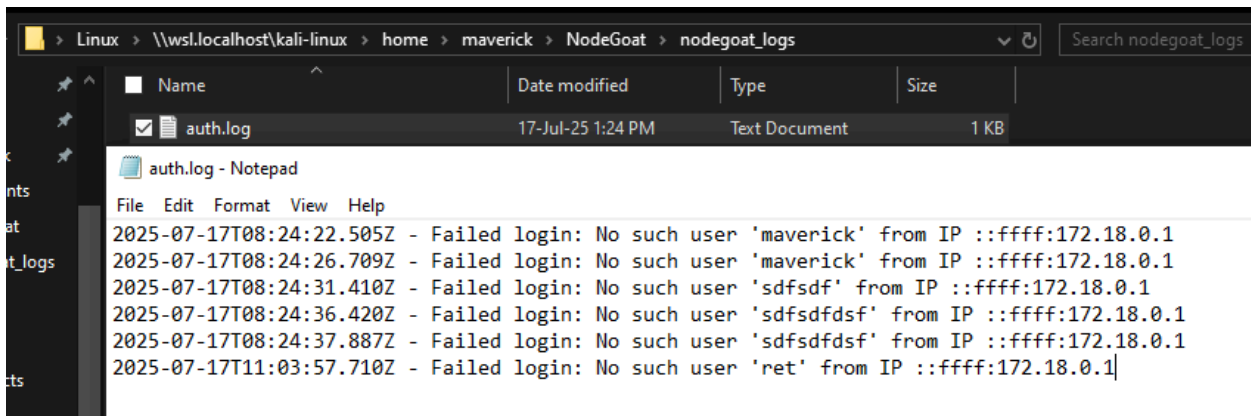
```

Now that we have created the mechanism to log the authentication event, we will update `docker-compose.yml`, and add the log path file environment variable and the volume mount.



```
1 version: "3.7"
2
3 services:
4   web:
5     build: .
6     environment:
7       NODE_ENV: production
8       MONGODB_URI: mongodb://mongo:27017/nodegoat
9
10    LOG_FILE_PATH: /app/logs/auth.log # Path inside the container where auth logs will be written
11    SESSION_SECRET: 7130711126b27c4ca05c1e3e544fe63947ab7188a9c2726cf97a93084f91589c
12    # -----
13    command: sh -c "until nc -z -w 2 mongo 27017 && echo 'mongo is ready for connections' && node artifacts/db-reset.js && npm start; do sleep 2; done"
14    ports:
15      - "4000:4000"
16    volumes:
17      - ./nodegoat_logs:/app/logs # Mounts a 'nodegoat_logs' directory on host to /app/Logs in the container
18      # -----
19
20  D> Run Service
21  mongo:
22    image: mongo:4.4
23    user: mongod
24    expose:
25      - 27017
```

Now that everything is set for the e-log file, we can observe every log saved in the log file:



Name	Date modified	Type	Size
auth.log	17-Jul-25 1:24 PM	Text Document	1 KB

```
File Edit Format View Help
2025-07-17T08:24:22.505Z - Failed login: No such user 'maverick' from IP ::ffff:172.18.0.1
2025-07-17T08:24:26.709Z - Failed login: No such user 'maverick' from IP ::ffff:172.18.0.1
2025-07-17T08:24:31.410Z - Failed login: No such user 'sdfsdf' from IP ::ffff:172.18.0.1
2025-07-17T08:24:36.420Z - Failed login: No such user 'sdfsdfdsf' from IP ::ffff:172.18.0.1
2025-07-17T08:24:37.887Z - Failed login: No such user 'sdfsdfdsf' from IP ::ffff:172.18.0.1
2025-07-17T11:03:57.710Z - Failed login: No such user 'ret' from IP ::ffff:172.18.0.1
```

b. Configuring Fail2Ban (on Host System):

Now that logs are exposed, we can configure Fail2Ban on our host machine.

Install fail2ban: `sudo apt install fail2ban`

Copy the default configuration (backup): `sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local.`

There will be a few settings to review in jail. Local, like bantime, findtime, and maxretry.

This is where we will define "jails" - specific rulesets that tell Fail2Ban what logs to monitor, what patterns to look for, and what actions to take when those patterns are detected.

Create a Fail2Ban Filter for NodeGoat. Fail2Ban uses filters to define patterns to match in log files: `sudo nano /etc/fail2ban/filter.d/nodegoat.conf`

```
GNU nano 8.4 /etc/fail2ban/filter.d/nodegoat.conf
[INCLUDES]
before = common.conf

[Definition]
daemon = nodegoat-web-1 # Or web-1 if your Docker logging prefix is more reliable

# Regex for failed login attempts
# This matches lines like: "2025-07-17T07:40:00.237Z - Failed login: Invalid password for user 'admin' from IP ::ffff:1
# And: "2025-07-17T07:45:19.569Z - Failed login: No such user 'admindfgefg' from IP ::ffff:172.18.0.1"
failregex = ^\s*\S+ - Failed login: (?No such user|Invalid password) for user '.*?' from IP <HOST>$

# Regex for successful login attempts
successregex = ^\s*\S+ - Successful login for user '.*?' from IP <HOST>$

#
```

Create a new jail file: `sudo nano /etc/fail2ban/jail.local` and add this filter below:

```
[nodegoat]
enabled = true
port = http,https,4000
filter = nodegoat # Refers to /etc/fail2ban/filter.d/nodegoat.conf
logpath = /home/maverick/NodeGoat/nodegoat_logs/auth.log
maxretry = 5 # Number of failed attempts before a ban
findtime = 1m # Time window (10 minutes) for maxretry
bantime = 1m # Ban duration (1 hour)
action = iptables-multiport[name=NodeGoat, port="http,https,4000", protocol=tcp]
```

Now restart: `sudo systemctl restart fail2ban`

```
(maverick@ DESKTOP-NNBUF8A) ~
$ sudo fail2ban-client status nodegoat
Status for the jail: nodegoat
|- Filter
|   |- Currently failed: 0
|   |- Total failed: 78
|   `-- File list: /home/maverick/NodeGoat/nodegoat_logs/auth.log
`- Actions
    |- Currently banned: 1
    |- Total banned: 2
    `-- Banned IP list: 172.18.0.1
```

See now IPs are getting blocked that tries to attempt 5 wrong authentication attempts within 1 minute, get blocked for 1 minute.

2- API Security Hardening:

This task requires modifying the NodeGoat application's source code, specifically its Node.js Express.js server, which is the `server.js` file inside the NodeGoat directory.

First, install all the required dependencies in the Nodegoat dir: `npm install`

a. Express Rate Limit:

Now we will install another dependency that will help protect against brute force: `npm install express-rate-limit`

We will modify the `server.js` file to apply a rate limit before all routes. I have this script at the top of the file:

```
21 const app = express();
22
23 const limiter = rateLimit({
24   windowMs: 1 * 60 * 1000,
25   max: 5,
26   message:
27     "Too many requests from this IP, please try again after 1 minutes",
28   standardHeaders: true,
29   legacyHeaders: false,
30 });
31
32 app.use(limiter);
```

b. Cross-Origin Resource Sharing:

Install the dependency that will implement a security mechanism in web browsers that allows web applications to access resources from domains other than the one serving the application.

`npm install cors`

Now modify `server.js` and add the following lines:

```
3 const cors = require('cors');

36 const corsOptions = {
37   origin: 'http://localhost',
38   methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
39   credentials: true,
40   optionsSuccessStatus: 204
41 };
42
43
44 app.use(cors(corsOptions));
45
46 app.use(limiter);
```

c. API Keys authentication:

In this task, we will implement api keys authentication.

Add the dependency:

```
3   const crypto = require('crypto')
```

Part one: Add dummy api keys for testing purposes.

Part two: Every time someone tries to access a protected API part, this Middleware checks if they have a valid "secret key" (the API key). If the key is correct, the request is allowed to pass. If it's missing or wrong, the middleware stops the request and sends an "Access Denied" message.

```
49  // --- 1. Mock API Key Storage ---
50  const VALID_API_KEYS = [
51    { key: "NG_API_KEY_DEV_123", description: "Development Team Key", permissions:
52      ["read", "write"] },
53    { key: "NG_API_KEY_ANALYTICS_456", description: "Analytics Dashboard Key",
54      permissions: ["read"] }
55  ];
56  // --- 2. API Key Authentication Middleware ---
57  function authenticateApiKey(req, res, next) {
58    const apiKey = req.headers['x-api-key'] || req.query.api_key;
59
60    if (!apiKey) {
61      return res.status(401).json({ message: "Access Denied: API Key missing." });
62    }
63
64    const foundKey = VALID_API_KEYS.find(validKey => validKey.key === apiKey);
65
66    if (foundKey) {
67      req.apiKeyInfo = foundKey;
68      console.log(`API Key authenticated: ${foundKey.description}`);
69      next();
70    } else {
71      console.log("API Key provided is invalid.");
72      res.status(403).json({ message: "Access Denied: Invalid API Key." });
73    }
74  }
```

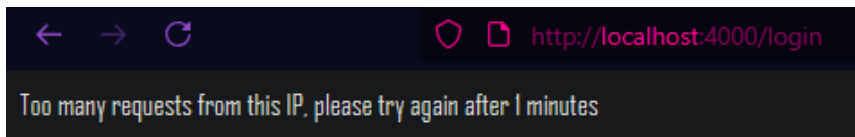
This is where we tell our Express app which specific API routes the Middleware should guard.

We've set it up so that our new /api/status route requires an API key.

```
193 // --- 3. Apply the API Key Middleware to specific API routes ---
194
195 app.get('/api/status', authenticateApiKey, (req, res) => {
196
197   console.log("Serving /api/status request.");
198   res.json({
199     status: "Operational",
200     message: "NodeGoat API is running smoothly.",
201     timestamp: new Date().toISOString(),
202     accessedBy: req.apiKeyInfo ? req.apiKeyInfo.description : 'Unknown'
203   });
204 });
205
```

d. Teseting

When I try more than 5 times within 1 minute, my IP gets blocked for 1 minute:



```
(maverick@DESKTOP-WNBUP8A) [~]
$ for i in {1..7}; do curl -o /dev/null -w "%{http_code}\n" http://localhost:4000/api/status -H "X-API-Key: NG_API_KEY_DEV_123"; done
100 145 100 145 0 0 9684 0 --:--:-- --:--:-- --:--:-- 10357
200
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 145 100 145 0 0 16367 0 --:--:-- --:--:-- --:--:-- 18125
200
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 145 100 145 0 0 16701 0 --:--:-- --:--:-- --:--:-- 18125
200
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 145 100 145 0 0 11811 0 --:--:-- --:--:-- --:--:-- 12083
200
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 145 100 145 0 0 18324 0 --:--:-- --:--:-- --:--:-- 20714
200
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 64 100 64 0 0 11120 0 --:--:-- --:--:-- --:--:-- 12800
429
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 64 100 64 0 0 10047 0 --:--:-- --:--:-- --:--:-- 10666
429
```


CORS can be checked in the request header:

```
(maverick@ DESKTOP-NNBUF8A) ~  
$ curl -v http://localhost:4000/api/status -H "X-API-Key: NG_API_KEY_DEV_123" -H "Origin: http://localhost"  
* Host localhost:4000 was resolved.  
* IPv6: ::1  
* IPv4: 127.0.0.1  
*   Trying [::1]:4000...  
* Connected to localhost (::1) port 4000  
* using HTTP/1.x  
> GET /api/status HTTP/1.1  
> Host: localhost:4000  
> User-Agent: curl/8.13.0  
> Accept: */*  
> X-API-Key: NG_API_KEY_DEV_123  
> Origin: http://localhost  
>  
* Request completely sent off  
< HTTP/1.1 200 OK  
< X-Powered-By: Express  
< Access-Control-Allow-Origin: http://localhost  
< Vary: Origin  
< Access-Control-Allow-Credentials: true  
< Ratelimit-Policy: 5;w=60  
< Ratelimit-Limit: 5  
< Ratelimit-Remaining: 4  
< Ratelimit-Reset: 60  
< Content-Type: application/json; charset=utf-8  
< Content-Length: 145  
* ETAG: W/"901-0444d10V5w3241Uwv40X4Uv3-"
```

Also, API keys can be tested differently:

```
(maverick@ DESKTOP-NNBUF8A) ~  
$ curl -v http://localhost:4000/api/status  
* Host localhost:4000 was resolved.  
* IPv6: ::1  
* IPv4: 127.0.0.1  
*   Trying [::1]:4000...  
* Connected to localhost (::1) port 4000  
* using HTTP/1.x  
> GET /api/status HTTP/1.1  
> Host: localhost:4000  
> User-Agent: curl/8.13.0  
> Accept: */*  
>  
* Request completely sent off  
* HTTP/1.1 401 Unauthorized  
< X-Powered-By: Express  
< Access-Control-Allow-Origin: http://localhost  
< Vary: Origin  
< Access-Control-Allow-Credentials: true  
* Ratelimit-Policy: 5;w=60
```

No API key

```
(maverick@ DESKTOP-NNBUF8A) ~  
$ curl -v http://localhost:4000/api/status -H "X-API-Key: NG_API_KEY_DEV_123"  
* Host localhost:4000 was resolved.  
* IPv6: ::1  
* IPv4: 127.0.0.1  
*   Trying [::1]:4000...  
* Connected to localhost (::1) port 4000  
* using HTTP/1.x  
> GET /api/status HTTP/1.1  
> Host: localhost:4000  
> User-Agent: curl/8.13.0  
> Accept: */*  
> X-API-Key: NG_API_KEY_DEV_123  
>  
* Request completely sent off  
< HTTP/1.1 200 OK  
< X-Powered-By: Express  
< Access-Control-Allow-Origin: http://localhost
```

Valid API Key

```

(maverick@DESKTOP-NNBUF8A) ~
$ curl -v http://localhost:4000/api/status -H "X-API-Key: THIS_IS_WRONG"
* Host localhost:4000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
* Trying [::1]:4000...
* Connected to localhost (::1) port 4000
* using HTTP/1.x
> GET /api/status HTTP/1.1
> Host: localhost:4000
> User-Agent: curl/8.13.0
> Accept: */*
> X-API-Key: THIS_IS_WRONG
>
* Request completely sent off
< HTTP/1.1 403 Forbidden
< X-Powered-By: Express
< Access-Control-Allow-Origin: http://localhost
< Vary: Origin
< Access-Control-Allow-Credentials: true

```

Invalid API Key

3- Security Headers & CSP Implementation:

First, we will set up security headers in server.js.

Install helmet.js, which is a collection of 14 middleware functions that help secure Express apps by setting various HTTP headers: `npm install helmet`

Add dependency:

```
5 const helmet = require('helmet');
```

And simply add the helmet for headers:

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

a. CSP Implementation

CSP is powerful for preventing XSS and data injection attacks. This is also handled by Helmet, but requires careful configuration.

Modify server.js to add CSP:

```

80 // -----
81 // Content Security Policy (CSP)
82 app.use(
83   helmet.contentSecurityPolicy({
84     directives: {
85       defaultSrc: ["'self'"],
86       scriptSrc: ["'self'", "'unsafe-inline'", "'unsafe-eval'", "https://trusted-cdn.com"],
87       styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
88       imgSrc: ["'self'", "data:"],
89       fontSrc: ["'self'", "https://fonts.gstatic.com"],
90       connectSrc: ["'self'", "ws://localhost:4000"], //
91       objectSrc: ["'none'"],
92     },
93   })
94 );
95 // -----

```

b. Enforce HTTPS using HSTS Headers:

HSTS tells browsers to only connect to your site using HTTPS. This header is typically set by Helmet.

```

98 app.use(
99   helmet.hsts({
100     maxAge: 31536000,
101     includeSubDomains: true,
102     preload: true
103   })
104 );

```

Now the Nodegoat is more secure than before. HTTPS is implemented and headers use secure policies now.

Tasks:

1. Intrusion Detection & Monitoring

- ☒ Set up real-time monitoring using Fail2Ban or OSSEC;
- ☒ Configure alert systems for multiple failed login attempts.

2. API Security Hardening

- ☒ ~~Apply rate limiting using express-rate-limit to prevent brute force attacks.~~
- ☒ ~~Properly configure CORS to restrict unauthorized access.~~
- ☒ ~~Secure APIs using API keys or OAuth authentication.~~

3. Security Headers & CSP Implementation

- ☒ ~~Implement Content Security Policy (CSP) to prevent script injections.~~
- ☒ ~~Enforce HTTPS using Strict Transport Security (HSTS) headers.~~

Submitted by **Athar Imran**

<https://github.com/atharImran728>