Advanced Security Report

This report is the last and follow-up report of the Implementing Security Measures task, in which we patched up a few vulnerabilities identified before as the Web Security engineering task. In this task, in the third week of the internship, to wrap up things, we will launch a few attacks to check whether our patch worked or not.

Developers Hub Cybersecurity Internship Task June 2025 Week: ***3***

Internee name: ***Athar Imran***

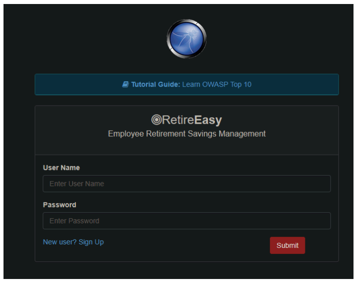
https://github.com/atharimran728/Web-Application-Security-Strengthening/tree/main

GOAL: **Simulate an Attack and set up logging**

❖ 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:



★ Checklist:

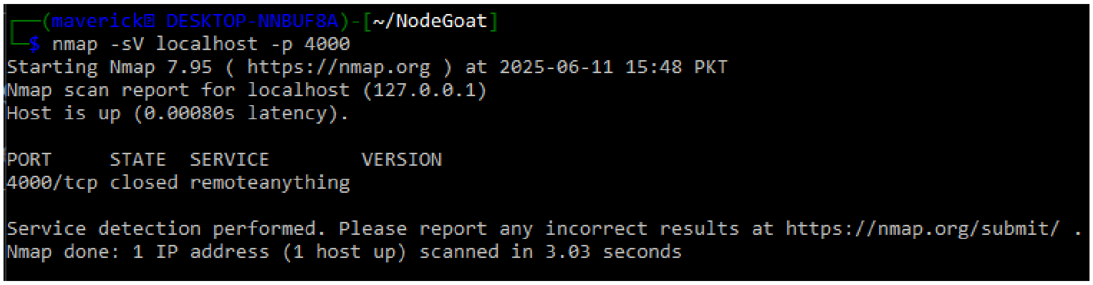
- ☑ ~~Logging is enabled with Winston~~
- ☑ ~~All inputs validated~~
- ☑ ~~Passwords are hashed using bcrypt~~
- ☑ ~~JWT was implemented for authentication~~
- ☑ ~~The helmet is used for headers~~

Below are the security checks for those patches:

1- Simulate Attacks with Nmap:

Nmap is a network scanner, and we will use it to create logs inside the Noadgoat server.

1. Download and install Nmap.
2. Identify the port number of localhost running NodeGoat. (By default, it's 4000)
3. Now run a simple nmap command that attempts to determine service version information of localhost:4000.
   a. Command: `nmap -sV localhost -p 4000`

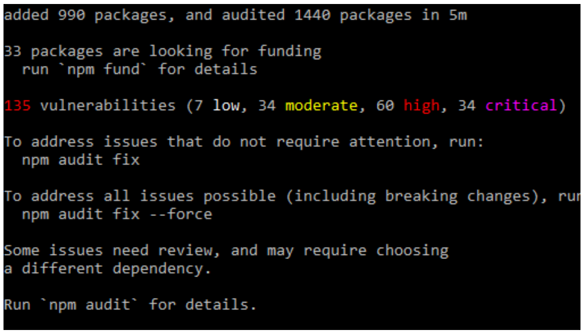


```
┌──(maverick㉿ DESKTOP-NNBUF8A)-[~/NodeGoat]
└─$ nmap -sV localhost -p 4000
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-11 15:48 PKT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00080s latency).

PORT     STATE  SERVICE        VERSION
4000/tcp closed remoteanything

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.03 seconds
```

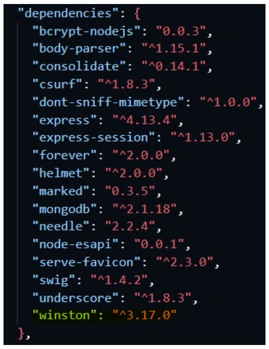
2- Add Logging with Winston:

This step involves modifying the NodeGoat application's source code.

1. First, we will install Winston in the nodegoat directory.
   a. `npm install winston`



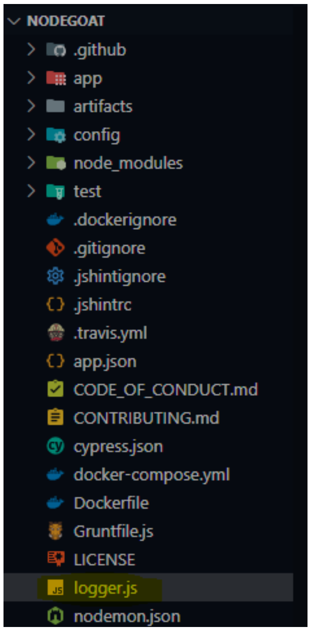
```
added 990 packages, and audited 1440 packages in 5m

33 packages are looking for funding
  run `npm fund` for details

135 vulnerabilities (7 low, 34 moderate, 60 high, 34 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), rur
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```



```
"dependencies": {
  "bcrypt-nodejs": "0.0.3",
  "body-parser": "^1.15.1",
  "consolidate": "^0.14.1",
  "csurf": "^1.8.3",
  "dont-sniff-mimetype": "^1.0.0",
  "express": "^4.13.4",
  "express-session": "^1.13.0",
  "forever": "^2.0.0",
  "helmet": "^2.0.0",
  "marked": "0.3.5",
  "mongodb": "^2.1.18",
  "needle": "2.2.4",
  "node-esapi": "0.0.1",
  "serve-favicon": "^2.3.0",
  "swig": "^1.4.2",
  "underscore": "^1.8.3",
  "winston": "^3.17.0"
},
```

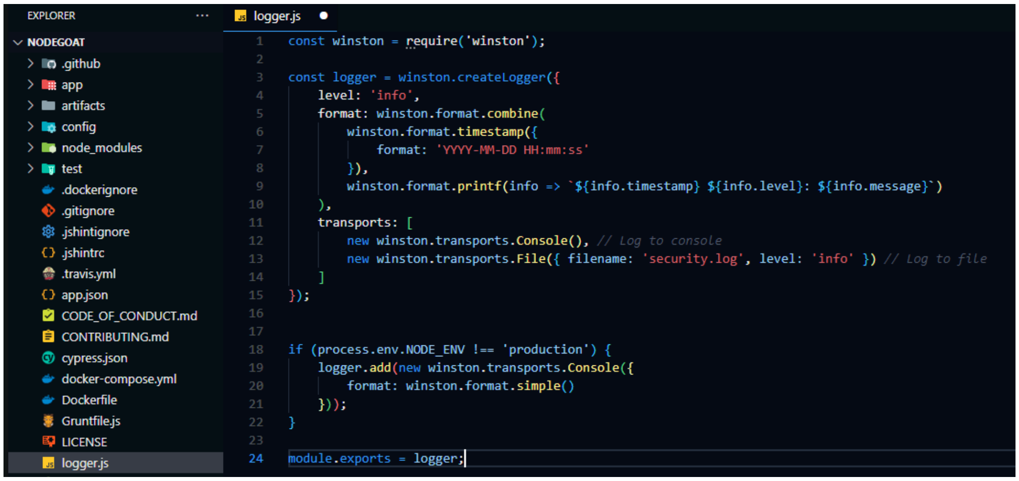
This will add `winston` to `package.json` and install it in `node_modules`.

2. Next up, we will create and add a `logger.js` file, which will be in the root directory.



```
NODEGOAT
  > .github
  > app
  > artifacts
  > config
  > node_modules
  > test
    .dockerignore
    .gitignore
    .jshintignore
    .jshintrc
    .travis.yml
    app.json
    CODE_OF_CONDUCT.md
    CONTRIBUTING.md
    cypress.json
    docker-compose.yml
    Dockerfile
    Gruntfile.js
    LICENSE
    logger.js
    nodemon.json
```

3. Add the following code lines to this file:



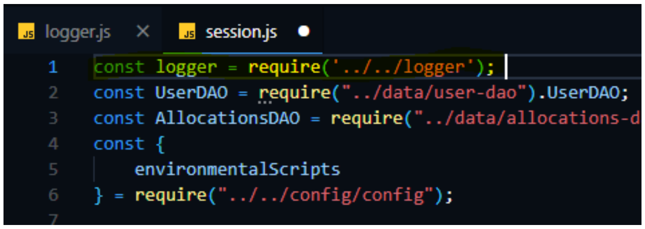
```
EXPLORER ...
NODEGOAT
  > .github
  > app
  > artifacts
  > config
  > node_modules
  > test
    .dockerignore
    .gitignore
    .jshintignore
    .jshintrc
    .travis.yml
    app.json
    CODE_OF_CONDUCT.md
    CONTRIBUTING.md
    cypress.json
    docker-compose.yml
    Dockerfile
    Gruntfile.js
    LICENSE
    logger.js
```

```
logger.js

const winston = require('winston');

const logger = winston.createLogger({
    level: 'info',
    format: winston.format.combine(
        winston.format.timestamp({
            format: 'YYYY-MM-DD HH:mm:ss'
        }),
        winston.format.printf(info => `${info.timestamp} ${info.level}: ${info.message}`)
    ),
    transports: [
        new winston.transports.Console(), // Log to console
        new winston.transports.File({ filename: 'security.log', level: 'info' }) // Log to file
    ]
});


if (process.env.NODE_ENV !== 'production') {
    logger.add(new winston.transports.Console({
        format: winston.format.simple()
    }));
}

module.exports = logger;
```

This code sets up a *robust logging system* for a Node.js application using the *Winston* library. Its primary purpose is to centralize and standardize how log messages are handled, making it easier to monitor application behavior, debug issues, and track security-related events.

4. The file is created, now we will integrate this logger into NodeGoat's routes. NodeGoat has many routes for the logins, but we are interested in the login routes. So, access the `session.js` file in *route* folder where the login functionality is programmed.

5. Add the logger file we created to the session file at the top of the code.

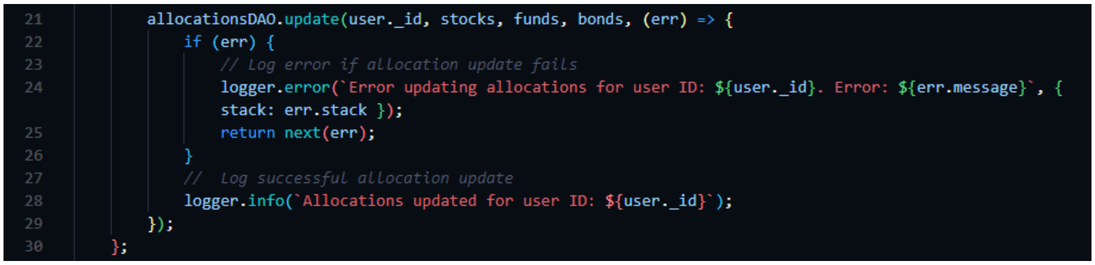


```
const logger = require('../../logger');
const UserDAO = require("../data/user-dao").UserDAO;
const AllocationsDAO = require("../data/allocations-d
const {
    environmentalScripts
} = require("../../config/config");
```

This will create a `logger` variable that will point to the logger file we created when called. As the logger file is two directories past, so ../../ will points in the two directory back.

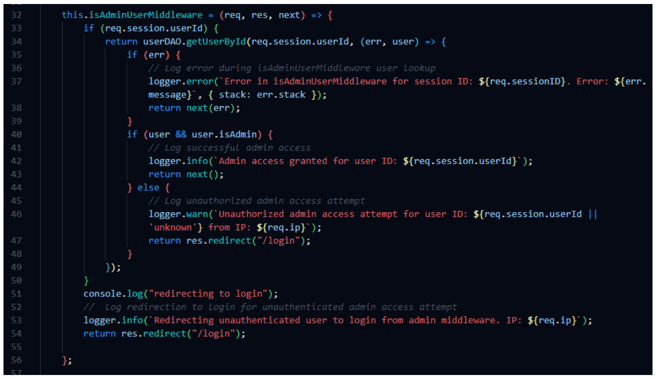
6. In this section, we will add some lines of code to the session program file, which will route the login logs into the logger.

7. These lines will update the allocation of the user with id:



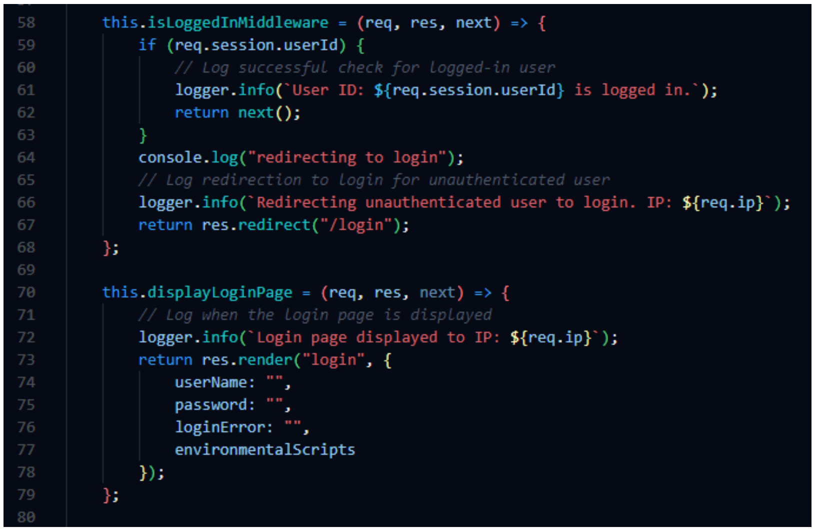
```
        allocationsDAO.update(user._id, stocks, funds, bonds, (err) => {
            if (err) {
                // Log error if allocation update fails
                logger.error(`Error updating allocations for user ID: ${user._id}. Error: ${err.message}`, {
                stack: err.stack });
                return next(err);
            }
            //  Log successful allocation update
            logger.info(`Allocations updated for user ID: ${user._id}`);
        });
    };
```

8. These lines will create logs for admin login activity:



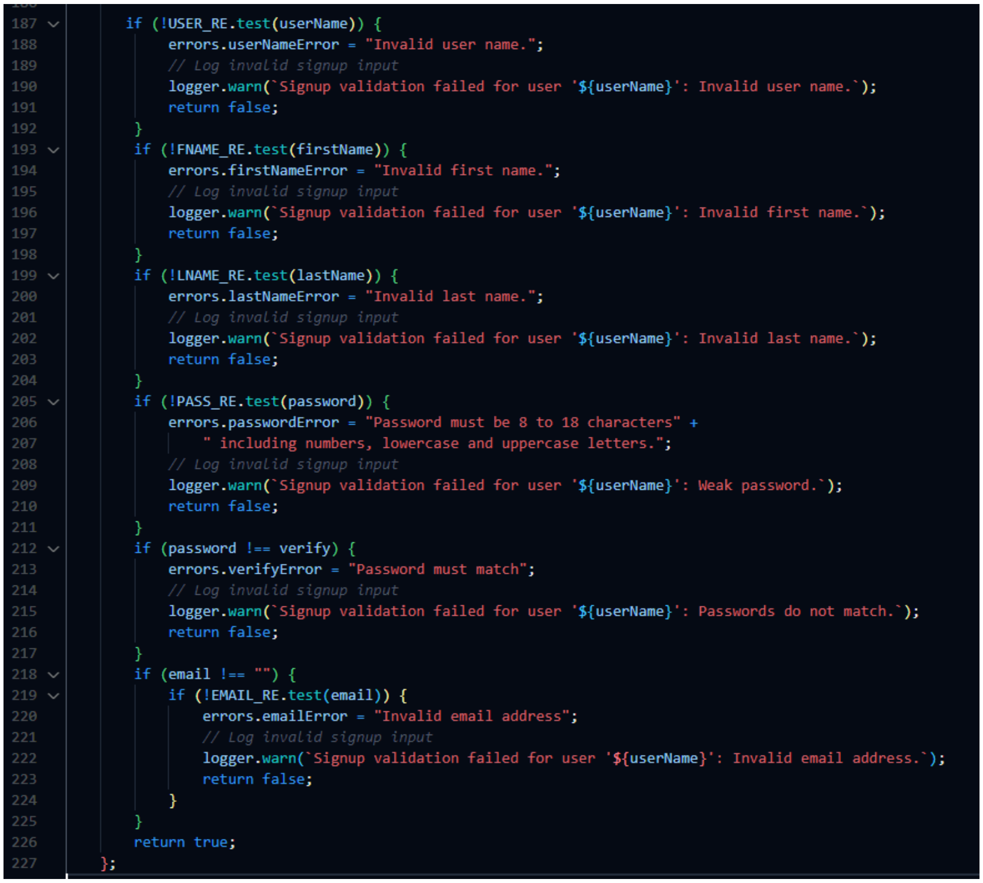
```
this.isAdminUserMiddleware = (req, res, next) => {
    if (req.session.userId) {
        return userDAO.getUserById(req.session.userId, (err, user) => {
            if (err) {
                // Log error during isAdminUserMiddleware user lookup
                logger.error(`Error in isAdminUserMiddleware for session ID: ${req.sessionID}. Error: ${err.
                message}`, { stack: err.stack });
                return next(err);
            }
            if (user && user.isAdmin) {
                // Log successful admin access
                logger.info(`Admin access granted for user ID: ${req.session.userId}`);
                return next();
            } else {
                // Log unauthorized admin access attempt
                logger.warn(`Unauthorized admin access attempt for user ID: ${req.session.userId ||
                'unknown'} from IP: ${req.ip}`);
                return res.redirect("/login");
            }
        });
    }
    console.log("redirecting to login");
    //  Log redirection to login for unauthenticated admin access attempt
    logger.info(`Redirecting unauthenticated user to login from admin middleware. IP: ${req.ip}`);
    return res.redirect("/login");

};
```

9. These lines will create a log when a successful check for a logged-in user, for an unauthenticated user, and when the login page is displayed.



```
    this.isLoggedInMiddleware = (req, res, next) => {
        if (req.session.userId) {
            // Log successful check for logged-in user
            logger.info(`User ID: ${req.session.userId} is logged in.`);
            return next();
        }
        console.log("redirecting to login");
        // Log redirection to login for unauthenticated user
        logger.info(`Redirecting unauthenticated user to login. IP: ${req.ip}`);
        return res.redirect("/login");
    };

    this.displayLoginPage = (req, res, next) => {
        // Log when the login page is displayed
        logger.info(`Login page displayed to IP: ${req.ip}`);
        return res.render("login", {
            userName: "",
            password: "",
            loginError: "",
            environmentalScripts
        });
    };

```

10. These lines of code will create logs for the event listed:



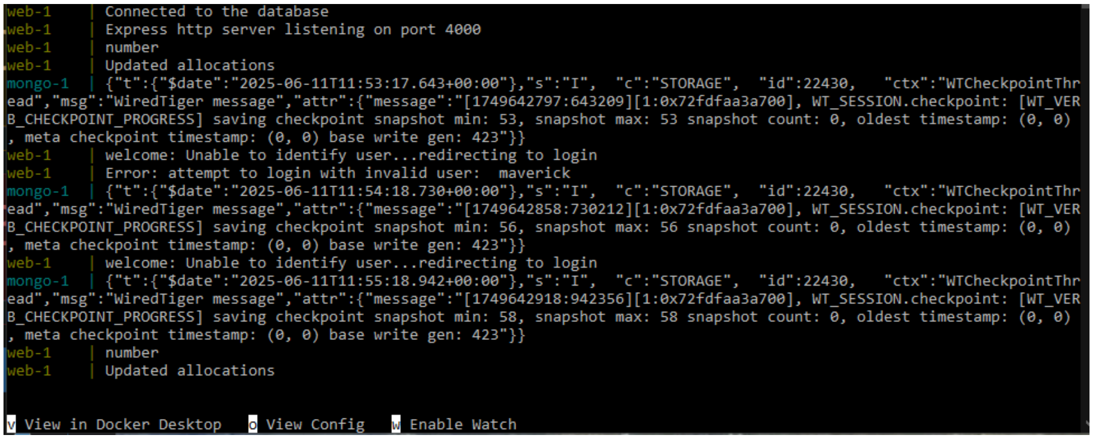
```
        if (!USER_RE.test(userName)) {
            errors.userNameError = "Invalid user name.";
            // Log invalid signup input
            logger.warn(`Signup validation failed for user '${userName}': Invalid user name.`);
            return false;
        }
        if (!FNAME_RE.test(firstName)) {
            errors.firstNameError = "Invalid first name.";
            // Log invalid signup input
            logger.warn(`Signup validation failed for user '${userName}': Invalid first name.`);
            return false;
        }
        if (!LNAME_RE.test(lastName)) {
            errors.lastNameError = "Invalid last name.";
            // Log invalid signup input
            logger.warn(`Signup validation failed for user '${userName}': Invalid last name.`);
            return false;
        }
        if (!PASS_RE.test(password)) {
            errors.passwordError = "Password must be 8 to 18 characters" +
                " including numbers, lowercase and uppercase letters.";
            // Log invalid signup input
            logger.warn(`Signup validation failed for user '${userName}': Weak password.`);
            return false;
        }
        if (password !== verify) {
            errors.verifyError = "Password must match";
            // Log invalid signup input
            logger.warn(`Signup validation failed for user '${userName}': Passwords do not match.`);
            return false;
        }
        if (email !== "") {
            if (!EMAIL_RE.test(email)) {
                errors.emailError = "Invalid email address";
                // Log invalid signup input
                logger.warn(`Signup validation failed for user '${userName}': Invalid email address.`);
                return false;
            }
        }
        return true;
    };
```

There are a lot more events where we can ask the logger to create and save the log. But for the sake of the task, I think that would be enough.

Now save the file and rebuild, and rerun Docker to compose the files:

```
docker-compose build
docker-compose run
```

Now you can see the logs created as the event occurs for what the logger is programmed for, inside the Docker terminal:



```
web-1     | Connected to the database
web-1     | Express http server listening on port 4000
web-1     | number
web-1     | Updated allocations
mongo-1   | {"t":{"$date":"2025-06-11T11:53:17.643+00:00"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx":"WTCheckpointThr
ead","msg":"WiredTiger message","attr":{"message":"[1749642797:643209][1:0x72fdfaa3a700], WT_SESSION.checkpoint: [WT_VER
B_CHECKPOINT_PROGRESS] saving checkpoint snapshot min: 53, snapshot max: 53 snapshot count: 0, oldest timestamp: (0, 0)
, meta checkpoint timestamp: (0, 0) base write gen: 423"}}
web-1     | welcome: Unable to identify user...redirecting to login
web-1     | Error: attempt to login with invalid user:  maverick
mongo-1   | {"t":{"$date":"2025-06-11T11:54:18.730+00:00"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx":"WTCheckpointThr
ead","msg":"WiredTiger message","attr":{"message":"[1749642858:730212][1:0x72fdfaa3a700], WT_SESSION.checkpoint: [WT_VER
B_CHECKPOINT_PROGRESS] saving checkpoint snapshot min: 56, snapshot max: 56 snapshot count: 0, oldest timestamp: (0, 0)
, meta checkpoint timestamp: (0, 0) base write gen: 423"}}
web-1     | welcome: Unable to identify user...redirecting to login
mongo-1   | {"t":{"$date":"2025-06-11T11:55:18.942+00:00"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx":"WTCheckpointThr
ead","msg":"WiredTiger message","attr":{"message":"[1749642918:942356][1:0x72fdfaa3a700], WT_SESSION.checkpoint: [WT_VER
B_CHECKPOINT_PROGRESS] saving checkpoint snapshot min: 58, snapshot max: 58 snapshot count: 0, oldest timestamp: (0, 0)
, meta checkpoint timestamp: (0, 0) base write gen: 423"}}
web-1     | number
web-1     | Updated allocations


v View in Docker Desktop   o View Config   w Enable Watch
```

We can also check the security log file, instead of the terminal.

3- Preparing the Security Checklist:

In this last section, we will create a checklist ensuring that all patches work correctly as expected.

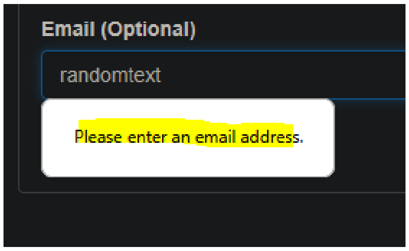
---

1. Inputs validated:

As we discussed in two areas of input validation under the credentials page, we will only check those two inputs here:

a. Email Pattern Validation:

We will enter any text rather than email pattern and check the response:



Email (Optional)

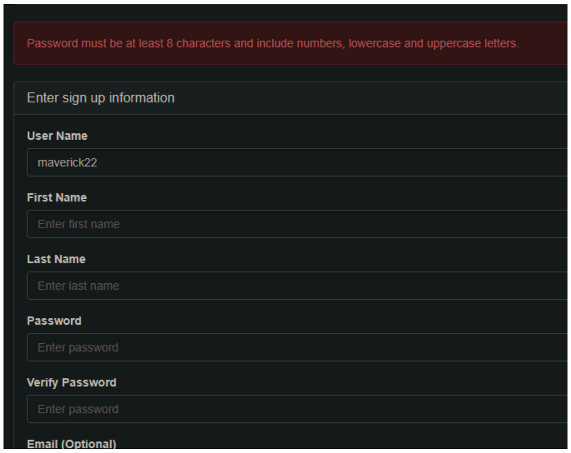
randomtext

Please enter an email address.

On putting random text in the email field, it doesn't validate it.

b. Minimum Password length check:

We entered a password with fewer than 8 characters, and here is the response:



Password must be at least 8 characters and include numbers, lowercase and uppercase letters.

Enter sign up information

User Name: maverick22

First Name: Enter first name

Last Name: Enter last name

Password: Enter password

Verify Password: Enter password

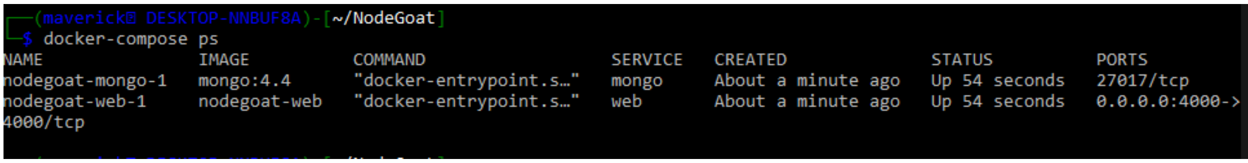
Email (Optional)

---

2. Passwords hashed:

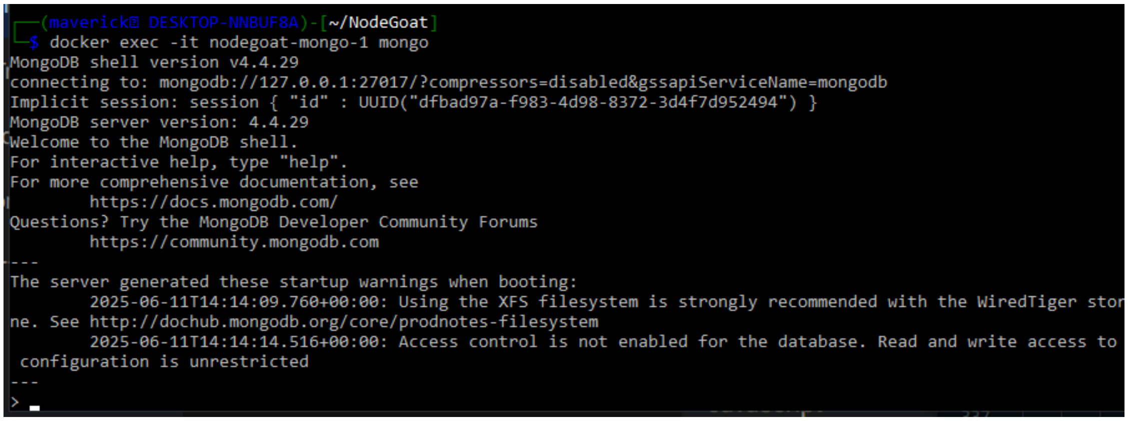
We cannot directly observe whether the passwords are being hashed or not from the front end, so we need to access the database where the passwords are stored:

1. While running the Docker nodegoat build behind, list the services the container is running:



```
┌──(maverick㉿ DESKTOP-NNBUF8A)-[~/NodeGoat]
└─$ docker-compose ps
NAME               IMAGE          COMMAND                   SERVICE   CREATED              STATUS          PORTS
nodegoat-mongo-1   mongo:4.4      "docker-entrypoint.s…"   mongo     About a minute ago   Up 54 seconds   27017/tcp
nodegoat-web-1     nodegoat-web   "docker-entrypoint.s…"   web       About a minute ago   Up 54 seconds   0.0.0.0:4000->
4000/tcp
```

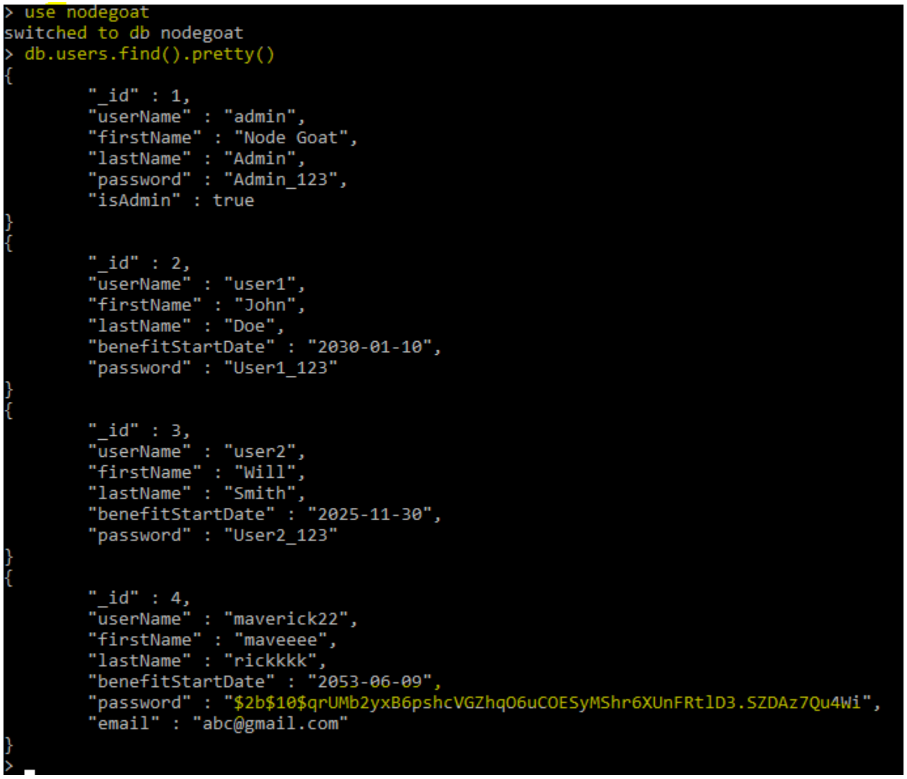
2. Now access the first service, nodegoat-mongo-1 with mongo:



```
┌──(maverick㉿ DESKTOP-NNBUF8A)-[~/NodeGoat]
└─$ docker exec -it nodegoat-mongo-1 mongo
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("dfbad97a-f983-4d98-8372-3d4f7d952494") }
MongoDB server version: 4.4.29
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2025-06-11T14:14:09.760+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger stor
ne. See http://dochub.mongodb.org/core/prodnotes-filesystem
        2025-06-11T14:14:14.516+00:00: Access control is not enabled for the database. Read and write access to
 configuration is unrestricted
---
> 
```

This will make us enter the database.

3. After entry into the database, use nodegoat, and retrieve users' data:

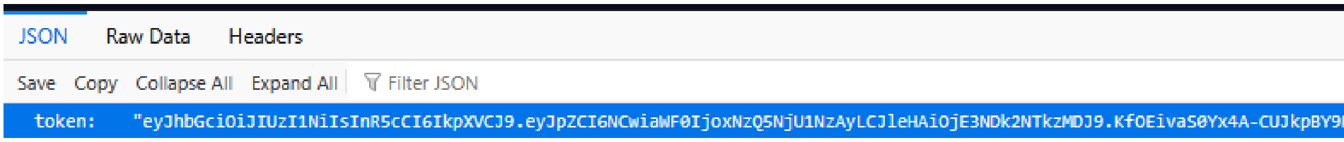


```
> use nodegoat
switched to db nodegoat
> db.users.find().pretty()
{
        "_id" : 1,
        "userName" : "admin",
        "firstName" : "Node Goat",
        "lastName" : "Admin",
        "password" : "Admin_123",
        "isAdmin" : true
}
{
        "_id" : 2,
        "userName" : "user1",
        "firstName" : "John",
        "lastName" : "Doe",
        "benefitStartDate" : "2030-01-10",
        "password" : "User1_123"
}
{
        "_id" : 3,
        "userName" : "user2",
        "firstName" : "Will",
        "lastName" : "Smith",
        "benefitStartDate" : "2025-11-30",
        "password" : "User2_123"
}
{
        "_id" : 4,
        "userName" : "maverick22",
        "firstName" : "maveeee",
        "lastName" : "rickkkk",
        "benefitStartDate" : "2053-06-09",
        "password" : "$2b$10$qrUMb2yxB6pshcVGZhqO6uCOESyMShr6XUnFRtlD3.SZDAz7Qu4Wi",
        "email" : "abc@gmail.com"
}
>
```

Onwards, we updated the code, and all the users' passwords will be stored as hashed and processed accordingly on the login step.

3. JWT implemented:

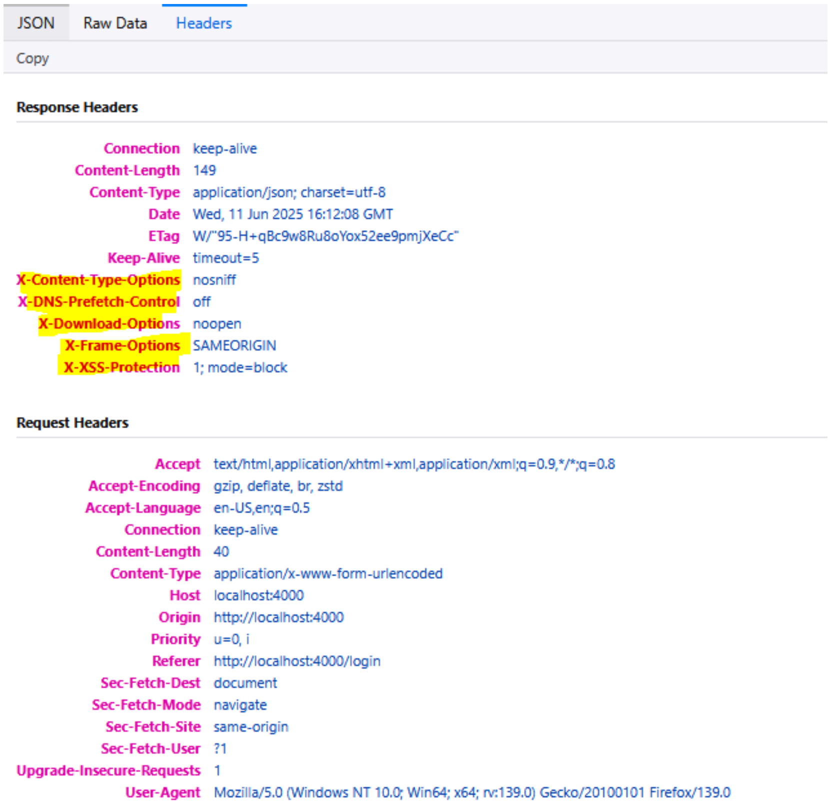
You can simply observe JWT implemented in the developer tools:



```
token:   "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NCwiaWF0IjoxNzQ5NjU1NzAyLCJleHAiOjE3NDk2NTkzMDJ9.KfOEivaS0Yx4A-CUJkpBY9
```

4. Helmet on headers:

Security headers can also be observed through web dev tools, like Inspect or Burp Suite. We will use simple inspection to make things simple. So, inspect for any web request while the user is logged in and observe:



Response Headers

Connection	keep-alive
Content-Length	149
Content-Type	application/json; charset=utf-8
Date	Wed, 11 Jun 2025 16:12:08 GMT
ETag	W/"95-H+qBc9w8Ru8oYox52ee9pmjXeCc"
Keep-Alive	timeout=5
X-Content-Type-Options	nosniff
X-DNS-Prefetch-Control	off
X-Download-Options	noopen
X-Frame-Options	SAMEORIGIN
X-XSS-Protection	1; mode=block

Request Headers

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding	gzip, deflate, br, zstd
Accept-Language	en-US,en;q=0.5
Connection	keep-alive
Content-Length	40
Content-Type	application/x-www-form-urlencoded
Host	localhost:4000
Origin	http://localhost:4000
Priority	u=0, i
Referer	http://localhost:4000/login
Sec-Fetch-Dest	document
Sec-Fetch-Mode	navigate
Sec-Fetch-Site	same-origin
Sec-Fetch-User	?1
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:139.0) Gecko/20100101 Firefox/139.0

These headers now act as a security helmet for the web request.

Checklists are checks above on the second page.

Submitted by ***Athar Imran***

https://github.com/atharImran728