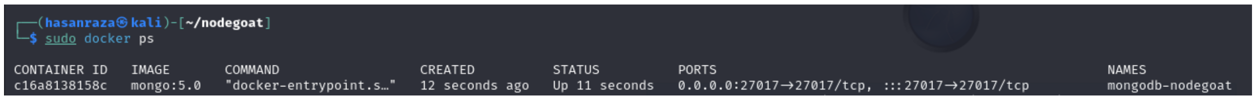
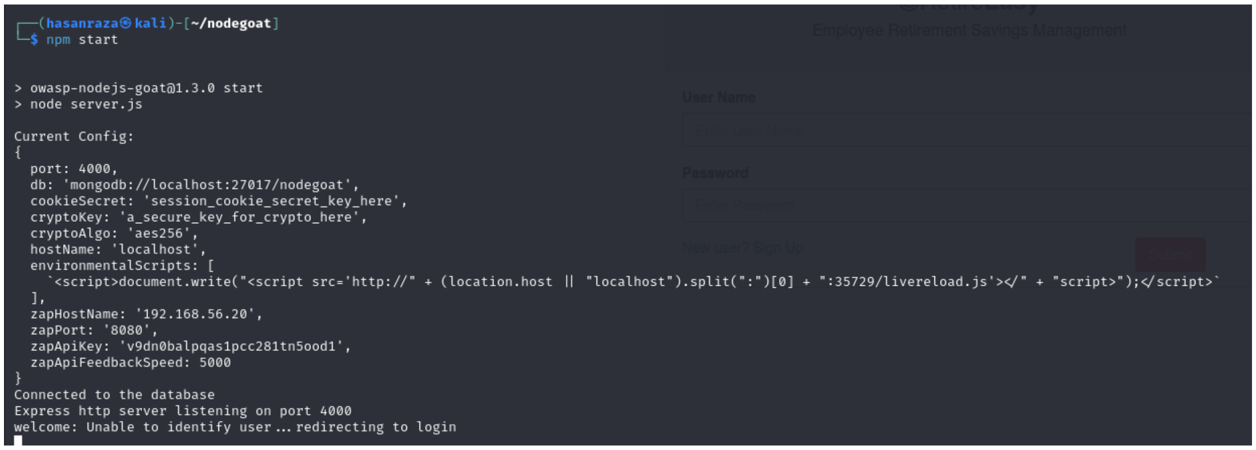
Developer’s Hub (Cyber Security Internship)
Submitted by Hasan Raza

Week 2: Implementing Security Measures

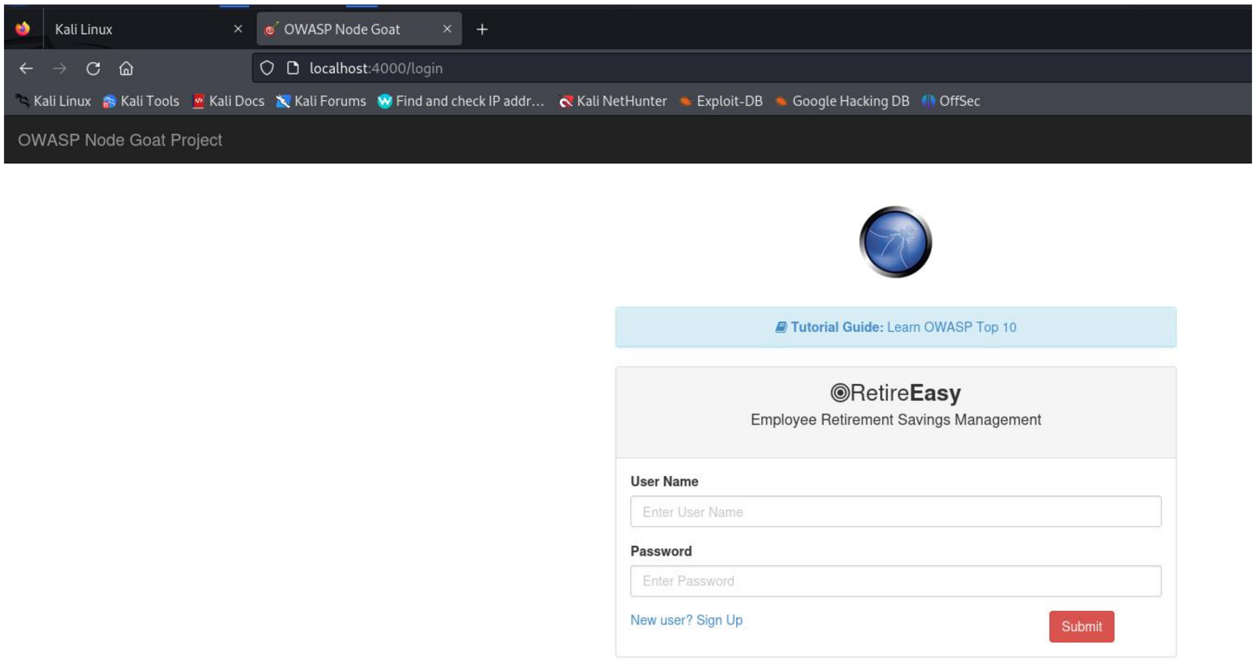
Using NodeGoat for this task



```
┌──(hasanraza㉿kali)-[~/nodegoat]
└─$ sudo docker ps

CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                           NAMES
c16a8138158c   mongo:5.0   "docker-entrypoint.s…"   12 seconds ago   Up 11 seconds   0.0.0.0:27017→27017/tcp, :::27017→27017/tcp     mongodb-nodegoat
```



```
┌──(hasanraza㉿kali)-[~/nodegoat]
└─$ npm start


> owasp-nodejs-goat@1.3.0 start
> node server.js

Current Config:
{
  port: 4000,
  db: 'mongodb://localhost:27017/nodegoat',
  cookieSecret: 'session_cookie_secret_key_here',
  cryptoKey: 'a_secure_key_for_crypto_here',
  cryptoAlgo: 'aes256',
  hostName: 'localhost',
  environmentalScripts: [
    `<script>document.write("<script src='http://" + (location.host || "localhost").split(":")[0] + ":35729/livereload.js'></" + "script>");</script>`
  ],
  zapHostName: '192.168.56.20',
  zapPort: '8080',
  zapApiKey: 'v9dn0balpqas1pcc281tn5ood1',
  zapApiFeedbackSpeed: 5000
}
Connected to the database
Express http server listening on port 4000
welcome: Unable to identify user ... redirecting to login
```



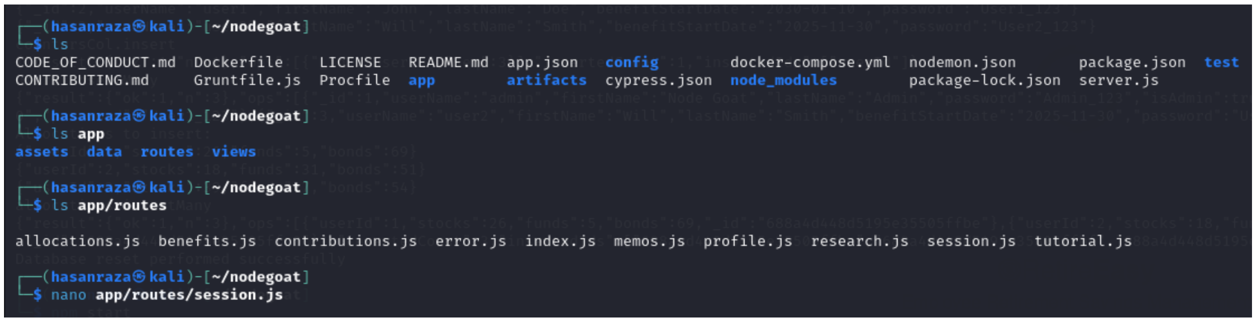
1. Fix Vulnerabilities

Sanitizing and Validating Inputs

Location of Changes:

All changes were made in:

nodegoat/app/routes/session.js
This file handles both user signup and login logic.



```
┌──(hasanraza㉿kali)-[~/nodegoat]
└─$ ls
CODE_OF_CONDUCT.md  Dockerfile    LICENSE   README.md  app.json   config        docker-compose.yml  nodemon.json       package.json  test
CONTRIBUTING.md     Gruntfile.js  Procfile  app        artifacts  cypress.json  node_modules        package-lock.json  server.js

┌──(hasanraza㉿kali)-[~/nodegoat]
└─$ ls app
assets  data  routes  views

┌──(hasanraza㉿kali)-[~/nodegoat]
└─$ ls app/routes
allocations.js  benefits.js  contributions.js  error.js  index.js  memos.js  profile.js  research.js  session.js  tutorial.js

┌──(hasanraza㉿kali)-[~/nodegoat]
└─$ nano app/routes/session.js
```

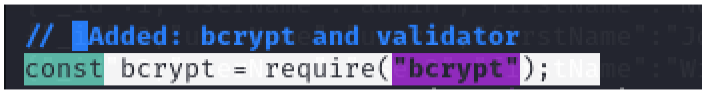
Use the validator library to validate user inputs:

Installed via:

npm install validator

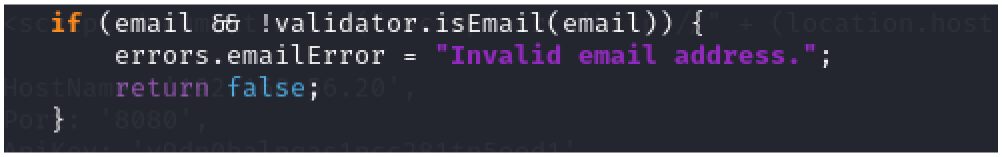
Integrated using:

const validator = require("validator");



```
// Added: bcrypt and validator
const bcrypt = require("bcrypt");
```

Used in signup validation:



```
if (email && !validator.isEmail(email)) {
    errors.emailError = "Invalid email address.";
    return false;
}
```

Uses:

- Prevents injection or malformed inputs
- Ensures email addresses are properly formatted
- Mitigates input-based vulnerabilities like log injection

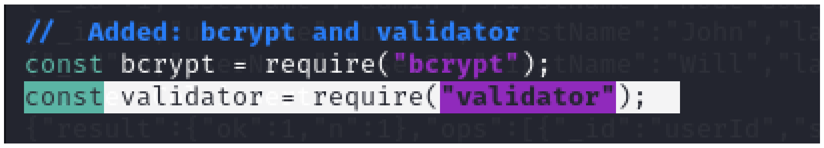
Password Hashing: Use bcrypt to hash:

Installed via:

npm install bcrypt

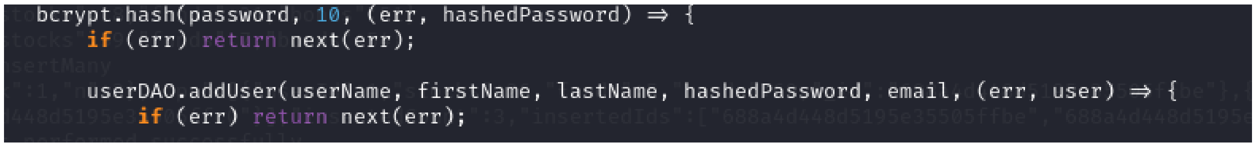
Integrated using:

const bcrypt = require("bcrypt");



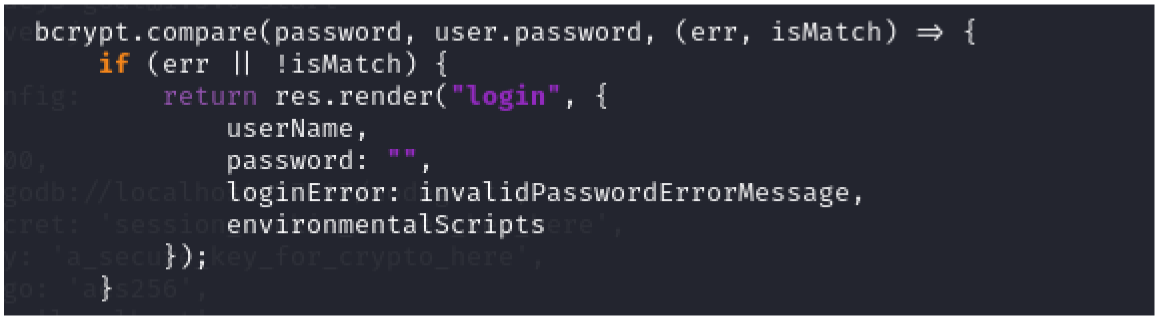
```
// Added: bcrypt and validator
const bcrypt = require("bcrypt");
const validator = require("validator");
```

Used during signup:



```
bcrypt.hash(password, 10, (err, hashedPassword) => {
    if (err) return next(err);

    userDAO.addUser(userName, firstName, lastName, hashedPassword, email, (err, user) => {
        if (err) return next(err);
```

Used during login:



```
bcrypt.compare(password, user.password, (err, isMatch) => {
    if (err || !isMatch) {
        return res.render("login", {
            userName,
            password: "",
            loginError: invalidPasswordErrorMessage,
            environmentalScripts
        });
    }
```

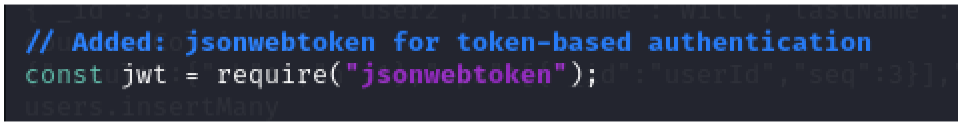
USES:

- Storing plain-text passwords is a major vulnerability.
- `bcrypt` hashes passwords with salt, preventing dictionary and rainbow table attacks.
- Adds a crucial layer of security in case of database compromise.

2. Enhance Authentication – Add JWT

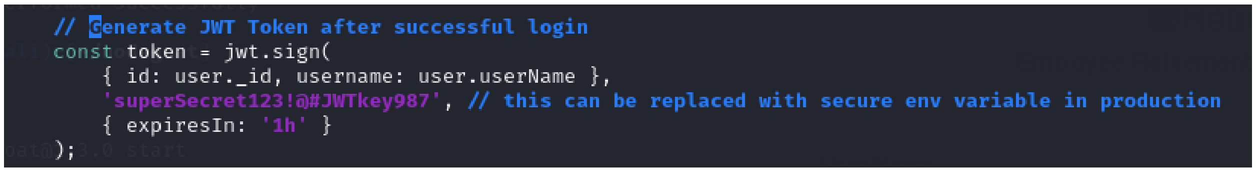
Installed jsonwebtoken via npm install jsonwebtoken

Added: jsonwebtoken for token-based authentication



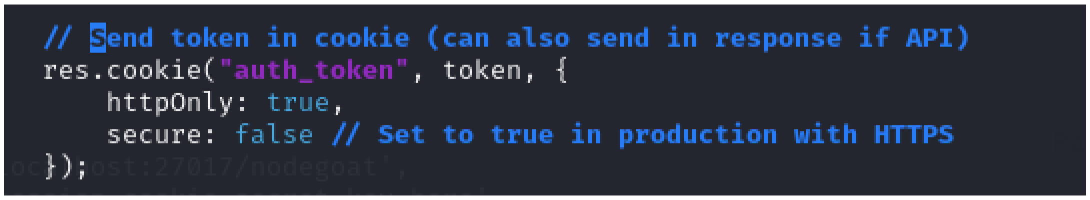
```
// Added: jsonwebtoken for token-based authentication
const jwt = require("jsonwebtoken");
```

Inside this.handleLoginRequest
After the user logs in successfully using bcrypt.compare, a secure **JWT token** is generated:



```
    // Generate JWT Token after successful login
    const token = jwt.sign(
        { id: user._id, username: user.userName },
        'superSecret123!@#JWTkey987', // this can be replaced with secure env variable in production
        { expiresIn: '1h' }
    );
```

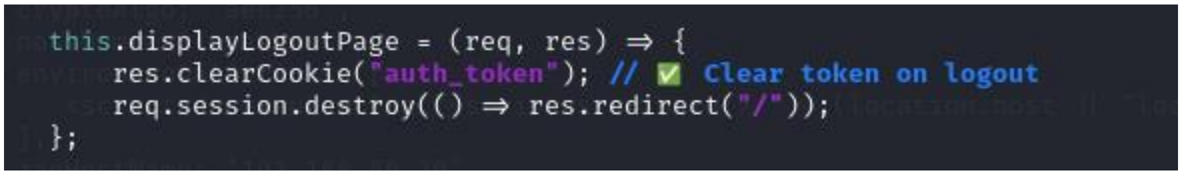
Directly below the `jwt.sign(...)` call
The token is sent to the browser using an **HTTP-only cookie,** preventing JavaScript access (helps against XSS)



```
// Send token in cookie (can also send in response if API)
res.cookie("auth_token", token, {
    httpOnly: true,
    secure: false // Set to true in production with HTTPS
});
```

Inside `this.displayLogoutPage`

On logout, the JWT cookie is cleared to invalidate the session.



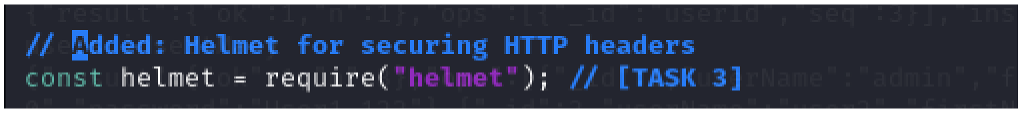
```
this.displayLogoutPage = (req, res) => {
    res.clearCookie("auth_token"); // ✅ Clear token on logout
    req.session.destroy(() => res.redirect("/"));
};
```

3. **Secure Data Transmission Use Helmet.js to secure HTTP headers:**

<u>Installed with:</u>

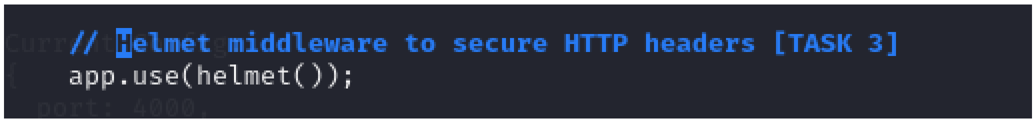
npm install helmet

<u>Added to `server.js`:</u>



```
// Added: Helmet for securing HTTP headers
const helmet = require("helmet"); // [TASK 3]
```

<u>Enabled Helmet middleware:</u>



```
// Helmet middleware to secure HTTP headers [TASK 3]
app.use(helmet());
```

<u>Effect:</u>

Enables secure HTTP headers like:

- Content-Security-Policy
- X-Frame-Options
- X-XSS-Protection
- Strict-Transport-Security