User Authentification Exercise

COMP.SEC.300–2023–2024-1 Secure Programming

Johanna Jaatinen

General description

GitHub repository: https://github.com/miajaa/userauth.git

The program has two parts in it, backend and frontend. I chose to do the backend in Python, due to the familiarity of it. I have not had too many backend programming courses in about a year so i went with what seemed most comfortable for me. In the backend I have utilized several different libraries to take away the risk of reimplementing some methods myself in a less secure way. This way I tried to mitigate the issues in my code security as well. The frontend is a basic JavaScript React application that I created a few years back, meaning Game.js and Game.css because i wanted to try to do something fun during my time off. This project required me to build also some frontend components for login page, registry and then build the game over and restart logic as well as the register and login modules. So, in the end there are multiple new things added during this course.

Tools and techologies

Backend:

- Flask: This is the main web framework used for building the application.
- Flask-CORS: This extension is used to handle Cross-Origin Resource Sharing (CORS) in Flask applications.
- Flask-Bcrypt: This extension provides bcrypt hashing utilities for Flask.
- Flask-JWT-Extended: Used for token creation adding a security layer to the user authentificstion
- Flask-SQLAlchemy: This extension integrates SQLAlchemy, a SQL toolkit and Object-Relational Mapping (ORM) library, with Flask for database operations simplification
- SQLite: Relational database management system used by the application. It's lightweight and requires minimal setup, making it suitable for small to medium-sized applications and therefore was a suitable option for the scope of the exercise
- Requests: Libray used for the sending of HTTP requests, particularly for verifying reCAPTCHA and OAuth2 tokens.
- dotenv: Library, used to load environment variables from a .env file, providing a convenient way to manage configuration settings.
- Logging: Python's built-in logging module is used for logging various events and errors that occur within the application.
- Secrets: Used for generating secure random tokens.
- os: A module that provides a portable way to use operating system-dependent functionalities, such as accessing environment variables and file paths and such

- re: Re provides support for regular expressions, which are used for validating email addresses and password complexity requirements in the app.

Frontend

- React: This is the main JavaScript library
- React Router DOM: Router DOM is used for declarative routing in the React application, allowing navigation between different components/pages.
- React Google ReCAPTCHA: Google reCAPTCHA v2 with the React application for preventing bots and spam.
- Axios: Axios is a promise-based HTTP client for making asynchronous requests to the server from the frontend. It's used for sending HTTP requests to the backend endpoints for registration, token validation and logging in.
- CSS: basic layouts and visuals for the components in the frontend, also with the game itself.

The added parts to the program are:

- Whole backend directory
- LandingPage.js
- RegisterModule.js
- RegisterModule.css
- Game.js: handleStartGame(), handleLogout(), handleCollision(), checkCollision() and the JSX components such buttons to trigger the functionality.

How the program is used

- Step 1. Open cmd
- Step 2. Navigate to backend root with “cd backend”
- Step 3. Run the following commands.

```
python -m venv env

.\env\Scripts\activate

pip install Flask Flask-Cors Flask-Bcrypt Flask-JWT-Extended Flask-SQLAlchemy requests python-dotenv Flask-Talisman Flask-Limiter

python server.py
```

- Step 4. Run the frontend by opening a new window to cmd and navigate to frontend with command “cd frontend”
- Step 5. Run npm install
- Step 6. Run npm start. The frontend should now be running in localhost:3000 or bigger if there is already pre-existing operations in the assigned port.

Secure programming practises used

- Input Validation: Email and password inputs are validated to make sure they are not empty. Validation happens with regex. Password strength checked for length, uppercase, lowercase, digits, and special characters.
- Hashing Passwords: Passwords are hashed using the bcrypt library before putting them into the database. This is to make sure that even if the database is compromised, passwords are not exposed.
- JWT Authentication: JSON Web Tokens are used in the authentification with login and registration, frontend send the token back to backend making sure it is valid before granting access. This eliminates the need to store session information on the server and reduces the risk of session hijacking.
- JWT tokens are signed using a secret key stored in the .env files for both backend and frontend.
- OAuth2 Authentication: OAuth2 authentication is implemented with Google's OAuth2 API for user login functionality
- Access tokens obtained from OAuth2 are verified through Google's token info endpoint before granting access.
- CAPTCHA Verification: Google reCAPTCHA is used to prevent automated bots from submitting registration forms.
- The CAPTCHA response is verified with Google's reCAPTCHA API before allowing form submission.
- Error Handling: Error handling is implemented for certain scenarios, including validation errors, duplicate user creation, failed CAPTCHA verification, and server errors. This prevents leaking sensitive information to users and helps in diagnosing issues when they happen. The errors are also logged in a separate file.
- Environment Variables: Secrets and sensitive configuration variables are stored in environment variables and loaded using dotenv. This prevents accidental exposure of sensitive information in source code repositories.
- Cross-Origin Resource Sharing (CORS): CORS headers are implemented to control access to resources from different origins, goal is to prevent unauthorized cross-origin requests.
- Flask logging in backend, logging statements are added to capture successful user registrations and errors during registration, login, and OAuth2 callback processes. The log can be read in the app.log file-
- SQL Injection Prevention: The use of SQLAlchemy's ORM (Object-Relational Mapping) helps in preventing SQL injection attacks by automatically sanitizing input and escaping special characters.
- Rate limiting: By controlling the rate of incoming requests, this helps stabilize traffic and to mitigate potential attacks. Rate limiting is in use for login and registration actions

Security issues, vulnerabilities or not yet implemented

HTTPS: The code does not enforce HTTPS encryption. It could though. HTTPS is a good way to ensure security between clients and the server, preventing eavesdropping and data tampering but for the local usage purposes I did not go as far as to implement the HTTPS

Dotenv: instead of putting the secrets into the .env i could have opted for other more secure options as well such as using a secure key management system or encrypted storage.

Unresolved issues:

Token validation always fails the token, therefore it is left out. However, this would be a critical error in the program if it was actually deployed like this to service because it will open up the possibility of unauthorized access and attackers could exploit this easily to gain access to the application. This token was set to mimic the email verification link sending with the popup during registration as well as silently in the background w

If this were to be an application, for example, healthcare or finance, this inadequate security measure could and would probably lead to legal and compliance issues. It would also put the data under risk meaning the failing token validation could lead to data breaches, therefore leading to a breach of privacy and trust. In reality this sort of access token implementation that

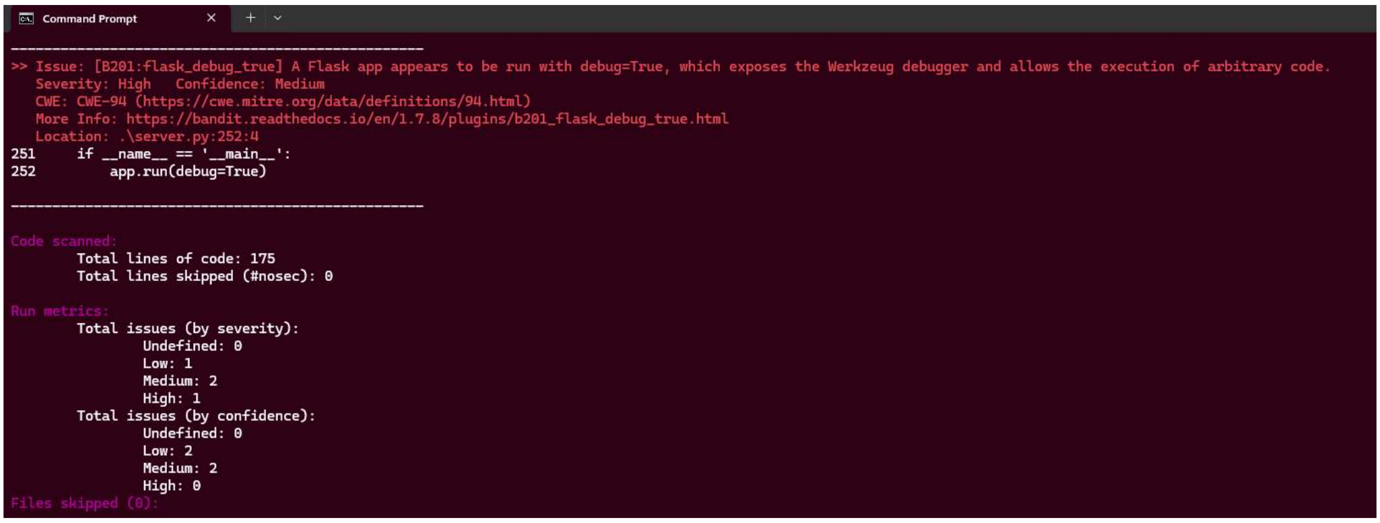
ReCAPTCHA is not showing sometimes, page needs to be refreshed (ctrl + r on WIN)

Testing

I have performed manual testing on the implementation throughout the development and I fixed/added things based on the results of the testing.

Backend

For the backend static analysis I used Bandit, with the guide from https://bandit.readthedocs.io/en/latest/start.html.



```
Command Prompt
--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b201_flask_debug_true.html
   Location: .\server.py:252:4
251     if __name__ == '__main__':
252         app.run(debug=True)

--------------------------------------------------

Code scanned:
        Total lines of code: 175
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 2
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 2
                Medium: 2
                High: 0
Files skipped (0):
```

I fixed it by replacing the final line in the server.py from

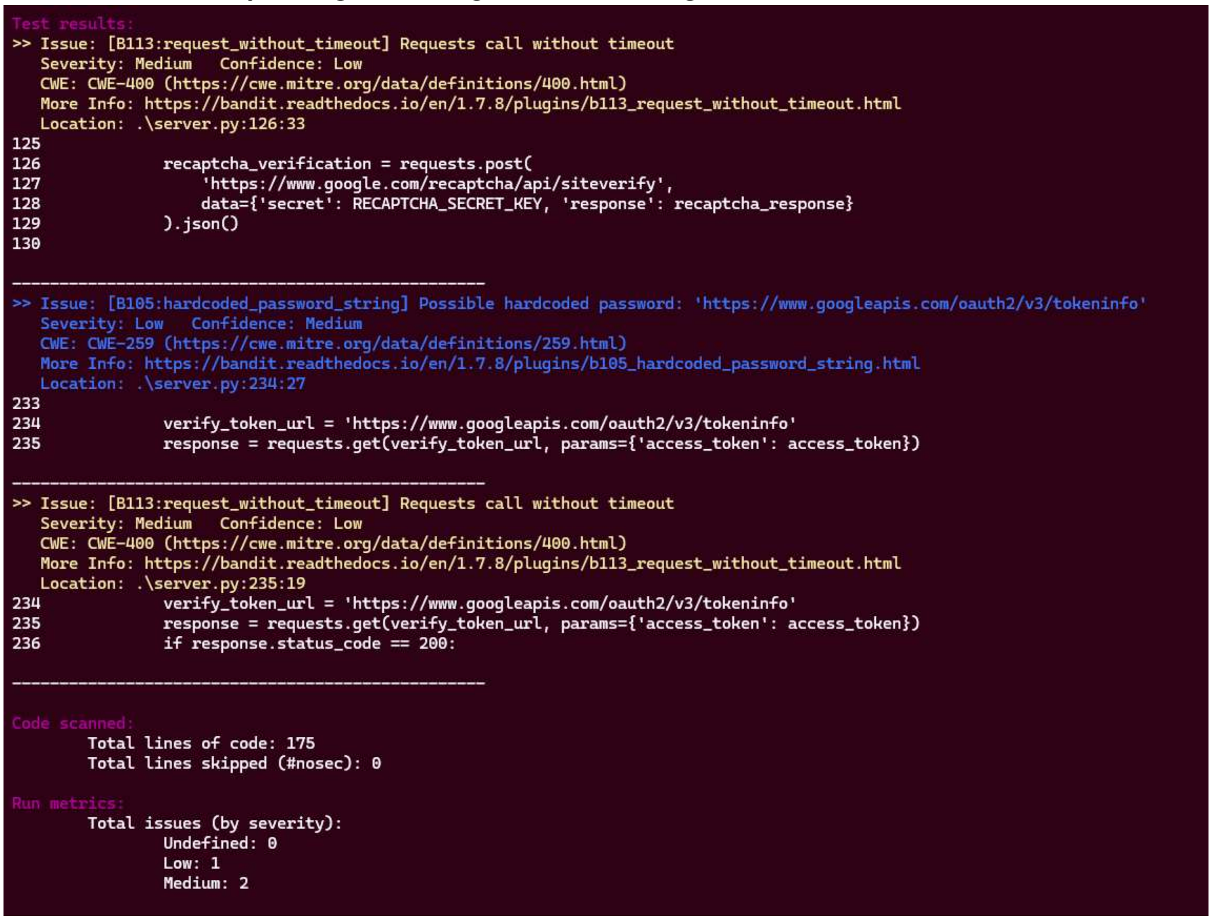
```
app.run(debug=True)
```

To

```
app.run(debug=False)
```

With this change my Flask backend will not run in debug mode by default which helps mitigate the security risks if this app would never be deployed outside local development environment.

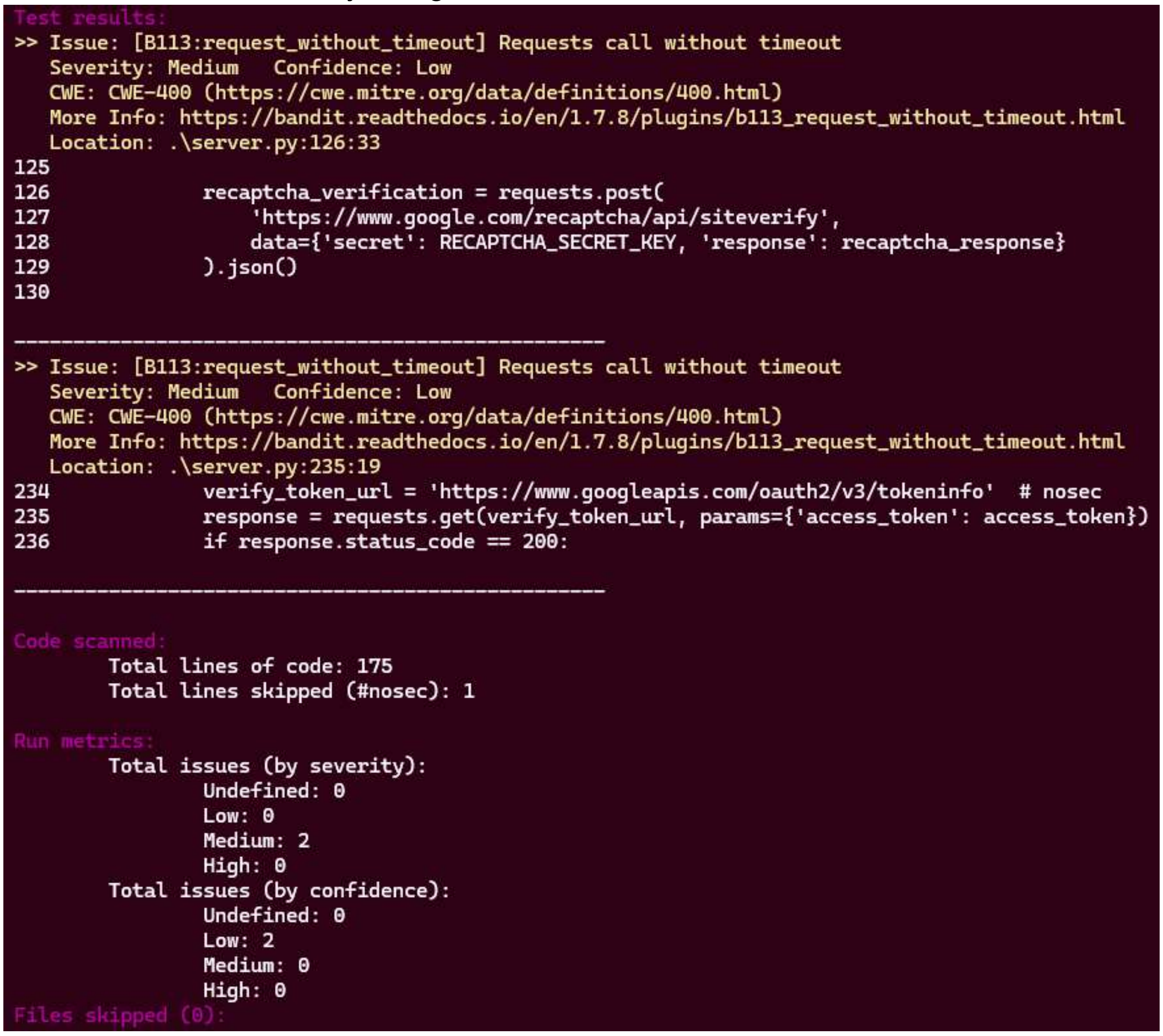
And ran the analyser again and got he following:



```
Test results:
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b113_request_without_timeout.html
   Location: .\server.py:126:33
125
126             recaptcha_verification = requests.post(
127                 'https://www.google.com/recaptcha/api/siteverify',
128                 data={'secret': RECAPTCHA_SECRET_KEY, 'response': recaptcha_response}
129             ).json()
130

--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'https://www.googleapis.com/oauth2/v3/tokeninfo'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b105_hardcoded_password_string.html
   Location: .\server.py:234:27
233
234             verify_token_url = 'https://www.googleapis.com/oauth2/v3/tokeninfo'
235             response = requests.get(verify_token_url, params={'access_token': access_token})

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b113_request_without_timeout.html
   Location: .\server.py:235:19
234             verify_token_url = 'https://www.googleapis.com/oauth2/v3/tokeninfo'
235             response = requests.get(verify_token_url, params={'access_token': access_token})
236             if response.status_code == 200:

--------------------------------------------------

Code scanned:
        Total lines of code: 175
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 2
```

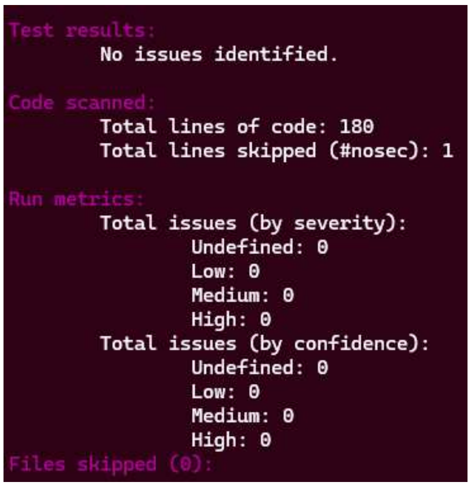
Funny enough using the URL for Google's OAuth tokeninfo is flagged it as hardcoded password when it is not in fact one. I could hide that google endpoint into the .env file as well as the other secrets, but I don't think in this case I need to flag this too high so i added the # nosec comment after this which means the Bandit will ignore the specific line for scanning the issues. This action of course needs to be always made to be uses when you are completely certain that the flagged code is not a security issue. Here in this case the API endpoint does not inherently pose a security risk.

After that i reran the analyser again:



```
Test results:
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b113_request_without_timeout.html
   Location: .\server.py:126:33
125
126             recaptcha_verification = requests.post(
127                 'https://www.google.com/recaptcha/api/siteverify',
128                 data={'secret': RECAPTCHA_SECRET_KEY, 'response': recaptcha_response}
129             ).json()
130

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b113_request_without_timeout.html
   Location: .\server.py:235:19
234             verify_token_url = 'https://www.googleapis.com/oauth2/v3/tokeninfo'  # nosec
235             response = requests.get(verify_token_url, params={'access_token': access_token})
236             if response.status_code == 200:

--------------------------------------------------

Code scanned:
        Total lines of code: 175
        Total lines skipped (#nosec): 1

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 2
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 2
                Medium: 0
                High: 0
Files skipped (0):
```

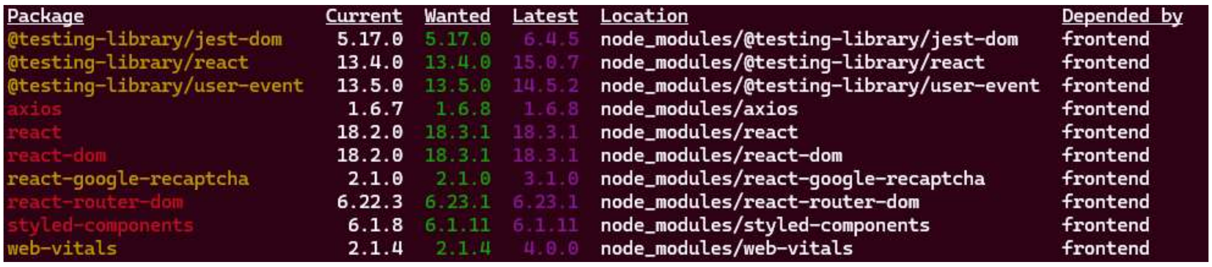
And it seems the recaptcha has no timeouts set so lets add them to make sure the application doesn't hang on indefinitely and result in uncontrolled behaviour. After i added the request.post and request.get timouts to 5 seconds the test results came back with zero.



```
Test results:
        No issues identified.

Code scanned:
        Total lines of code: 180
        Total lines skipped (#nosec): 1

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
Files skipped (0):
```

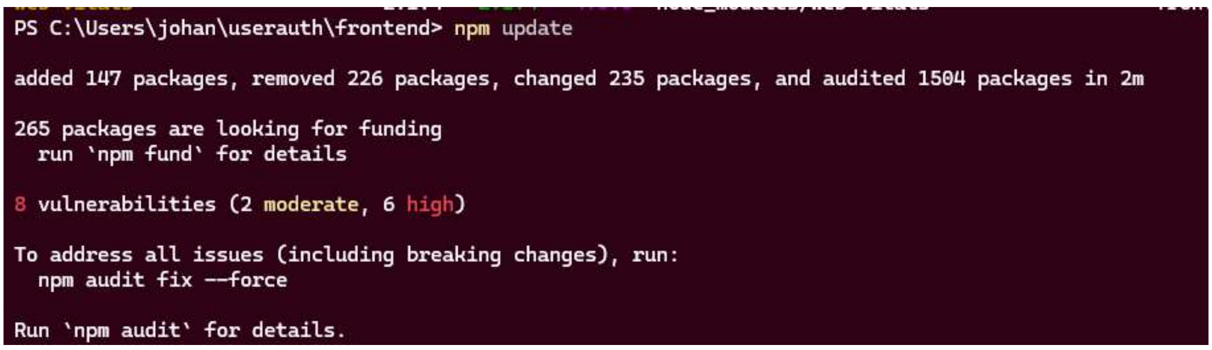
Frontend

Since the frontend is a react JS application, the easiest way to first check all the dependencies that I have is with npm so that is what i did and the results were not surprising as the app itself has been set up with basic react commands that will result in outdated packages. Using outdated packages can result in attacak vectors and security holes that cannot be fixed with just changing the source code itself.



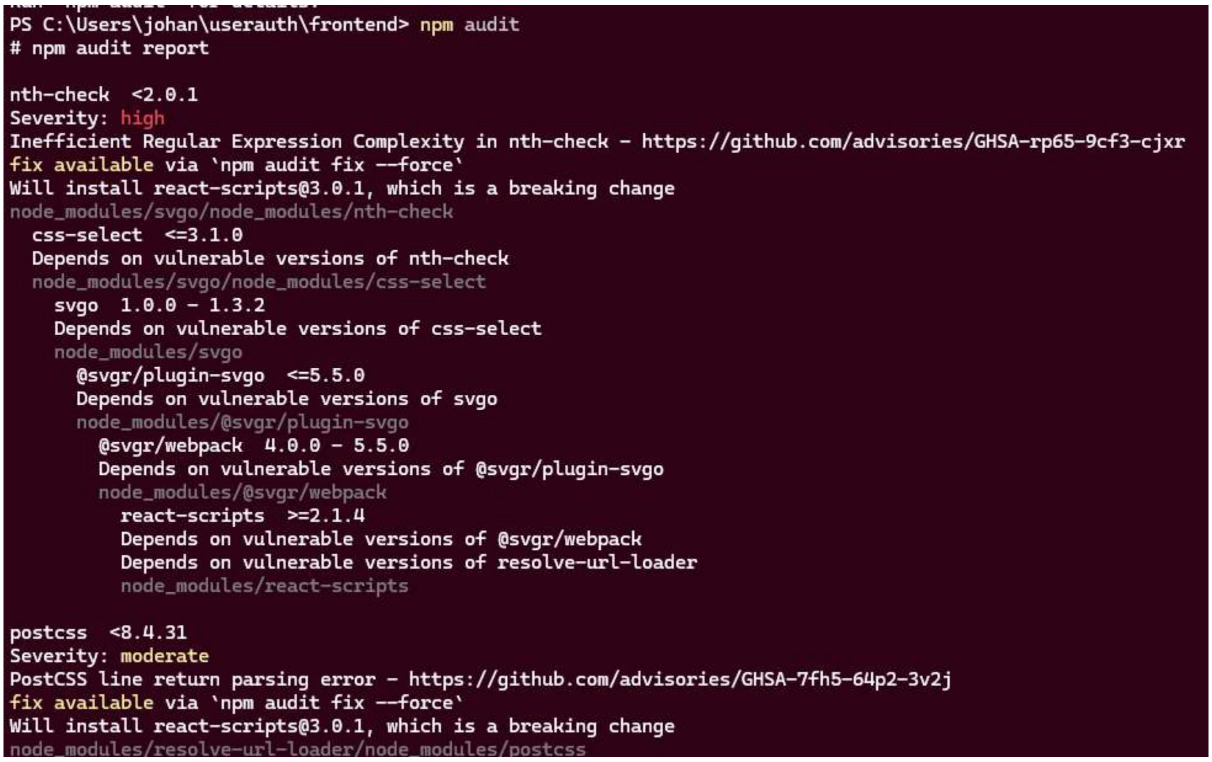
```
Package                         Current  Wanted  Latest  Location                                  Depended by
@testing-library/jest-dom        5.17.0  5.17.0   6.4.5  node_modules/@testing-library/jest-dom    frontend
@testing-library/react           13.4.0  13.4.0  15.0.7  node_modules/@testing-library/react       frontend
@testing-library/user-event     13.5.0  13.5.0  14.5.2  node_modules/@testing-library/user-event  frontend
axios                             1.6.7   1.6.8   1.6.8  node_modules/axios                        frontend
react                            18.2.0  18.3.1  18.3.1  node_modules/react                        frontend
react-dom                        18.2.0  18.3.1  18.3.1  node_modules/react-dom                    frontend
react-google-recaptcha            2.1.0   2.1.0   3.1.0  node_modules/react-google-recaptcha       frontend
react-router-dom                 6.22.3  6.23.1  6.23.1  node_modules/react-router-dom             frontend
styled-components                 6.1.8  6.1.11  6.1.11  node_modules/styled-components            frontend
web-vitals                        2.1.4   2.1.4   4.0.0  node_modules/web-vitals                   frontend
```

So i updated them with npm update.



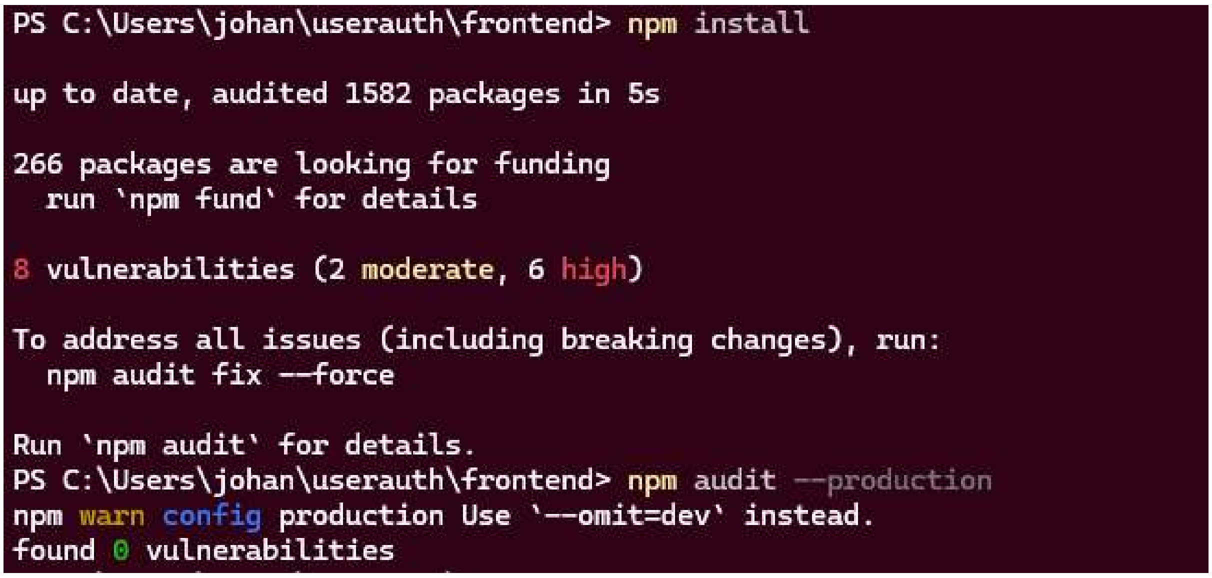
```
PS C:\Users\johan\userauth\frontend> npm update

added 147 packages, removed 226 packages, changed 235 packages, and audited 1504 packages in 2m

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

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

This results in many vulnerabilities that need to be addressed.



```
PS C:\Users\johan\userauth\frontend> npm audit
# npm audit report

nth-check  <2.0.1
Severity: high
Inefficient Regular Expression Complexity in nth-check - https://github.com/advisories/GHSA-rp65-9cf3-cjxr
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/svgo/node_modules/nth-check
  css-select  <=3.1.0
  Depends on vulnerable versions of nth-check
  node_modules/svgo/node_modules/css-select
    svgo  1.0.0 - 1.3.2
    Depends on vulnerable versions of css-select
    node_modules/svgo
      @svgr/plugin-svgo  <=5.5.0
      Depends on vulnerable versions of svgo
      node_modules/@svgr/plugin-svgo
        @svgr/webpack  4.0.0 - 5.5.0
        Depends on vulnerable versions of @svgr/plugin-svgo
        node_modules/@svgr/webpack
          react-scripts  >=2.1.4
          Depends on vulnerable versions of @svgr/webpack
          Depends on vulnerable versions of resolve-url-loader
          node_modules/react-scripts

postcss  <8.4.31
Severity: moderate
PostCSS line return parsing error - https://github.com/advisories/GHSA-7fh5-64p2-3v2j
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/resolve-url-loader/node_modules/postcss
```

After a while of trying to update the dependencies and getting empty responses also from npm outdated and still getting the same vulnerability notes, I resorted to googlin and found out that I have to remove react-scripts from dependencies and move them to devDependencies in package.json and run the npm audit with production flag.



```
PS C:\Users\johan\userauth\frontend> npm install

up to date, audited 1582 packages in 5s

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

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\Users\johan\userauth\frontend> npm audit --production
npm warn config production Use `--omit=dev` instead.
found 0 vulnerabilities
```

Since Create React App is a build tool, it runs at the build time during development and produces static assets. Npm audit is made for Node apps, and it therefore flags issues that can occur when running those apps in production, which is not how React Apps work. That is why it was false positive.