Static Code Analysis

1. Overview

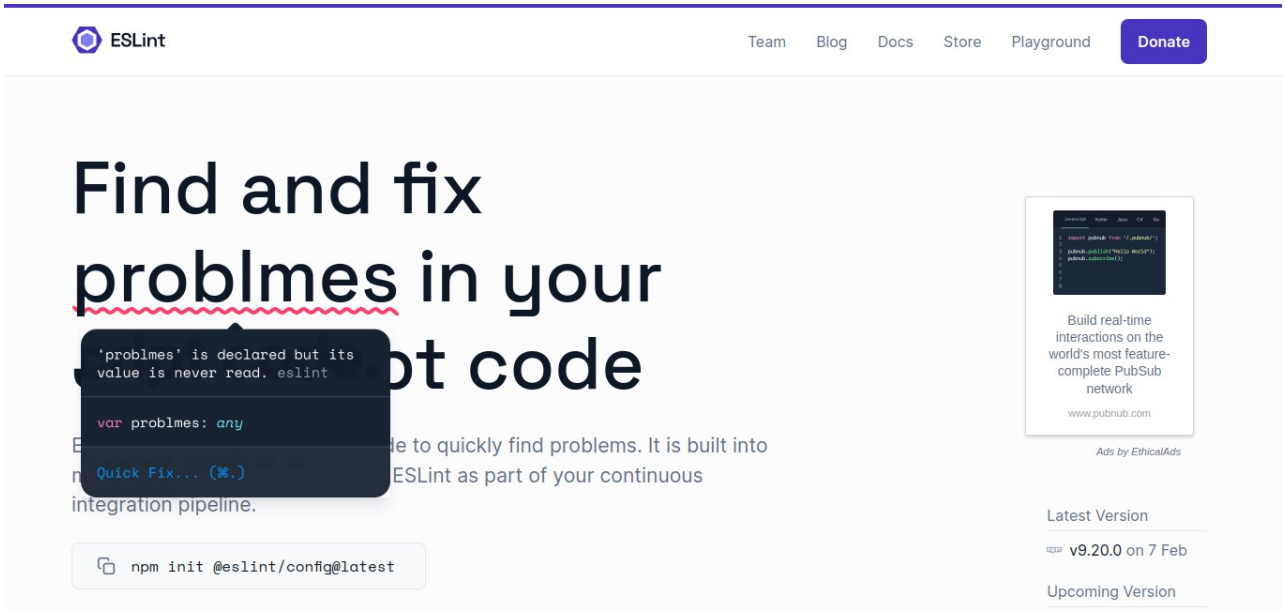


Figure 1: ESLint Website (https://eslint.org/)

In this lab, you are going to learn how to use ESLint and its security plugin to generate static code analysis report.

For ease of integration to Github Actions, you need a tool that generates reports in a format compatible with GitHub Code Scanning, SARIF format.

2. Outcomes

Upon completion of this session, you should be able to

- Use ESLint to analysis source code
- Use eslint-plugin-security for security analysis
- Select the suitable SAST for your team project
- Start incorporating Static Code Analysis into your team project

3. Configure the GitHub Actions Workflow

First install and configure eslint for your project. ESLint is used for javascript web application or node application. In addition, Github Actions uses SARIF report format (github/codeql-action/upload-sarif).

The following instructions is a starting point for you to get it working for your project. Please try out and adapt Eslint configuration, accordiing to your requirements.

Install eslint and sarif formatter

```
npm install eslint –save-dev
npm install @microsoft/eslint-formatter-sarif -save-dev
```

Initialize eslint for your project

```
npx eslint –init
```

Test Locally

```
npx eslint .
npx eslint . --format=@microsoft/eslint-formatter-sarif
--output-file=reports/eslint-results.sarif
```

Remember to add eslint files to your repo

git add .eslintrc.yml

Github Acitions Workflow

Copy the provided cda-eslint.yml to your github workflow directory.

Push this workflow file to `.github/workflows/eslint.yml`

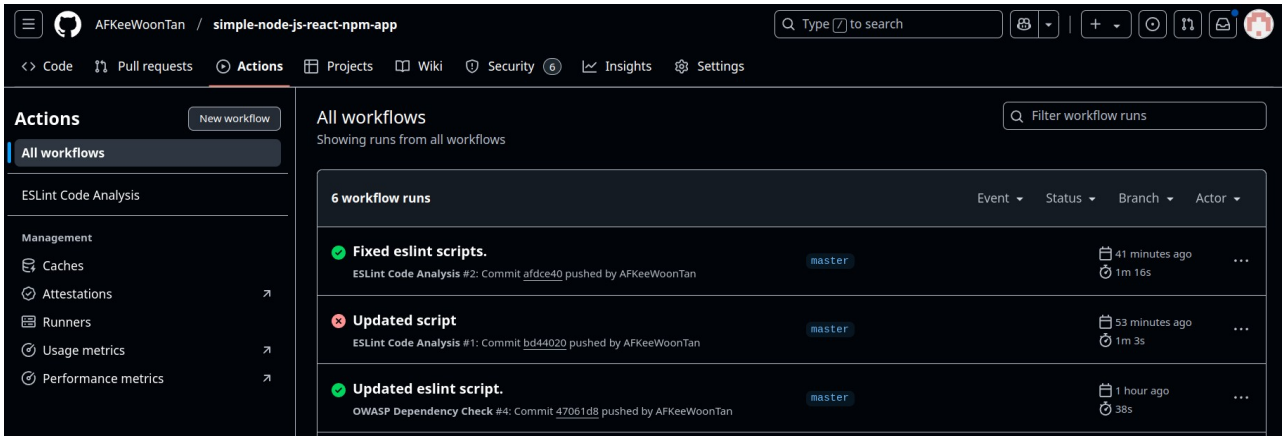
This workflow will:

- Run on push and pull requests to the `main` branch
- Install dependencies
- Run ESLint and fail if any issues are found

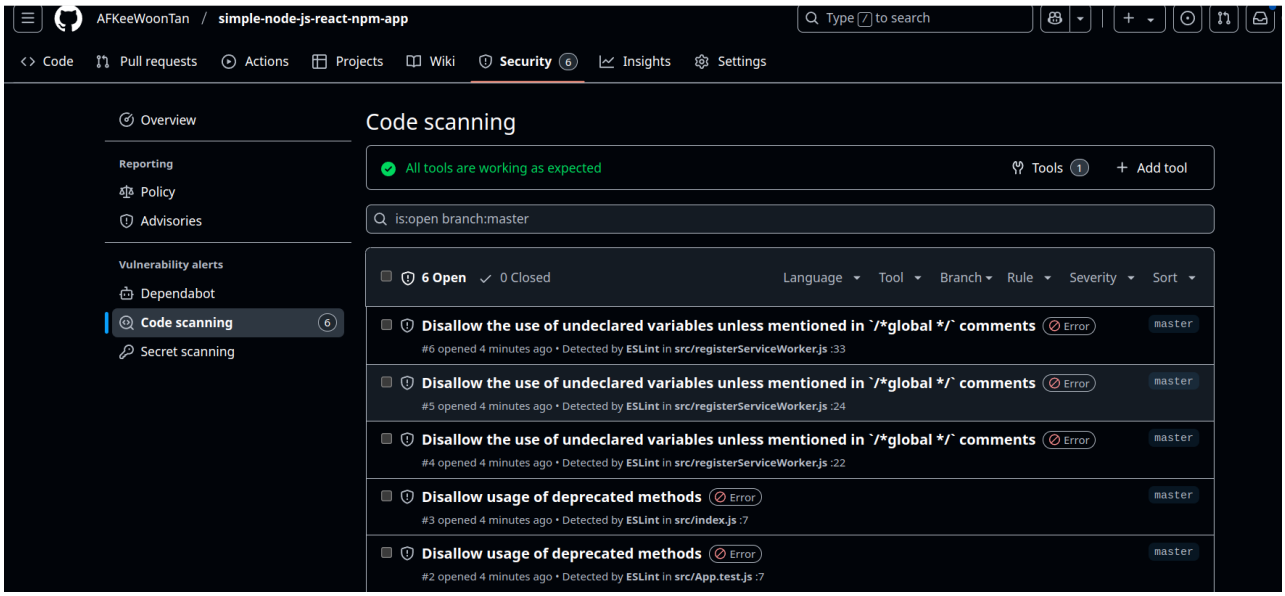
`Trigger Github Acitions Workflow`

Commit and push your changes to Github

The status of your Github Workflow run can be viewed under Actions.



`The ESlint report is found under Security.`



4. ESLint Security Plugins

ESLint performs static code analysis, but its primary focus is on code quality, style, and potential bugs, rather than security vulnerabilities. However, it can be extended to detect security issues by using plugins like:

eslint-plugin-security.

Detects common security issues in JavaScript, such as:

- Use of `eval()`
- Potential Regular Expression Denial of Service (ReDoS)
- Insecure use of `Buffer()`

NOTE: Using plugins with ESLint requires configuration according to your project

requirements. Please explore the plugin documentations and define the necessary security rules needed. And treat the plugin instructions as templates that you should adapt for your project.

4.1 ESLint-Plugin-Security

Installation of ESLint-Plugin-Security

```
npm install eslint-plugin-security -save-dev
```

Update .eslintrc.js with security configuration:

```
env:
  browser: true
  es2021: true

extends:
  - 'eslint:recommended'
  - 'plugin:react/recommended'
  - 'plugin:@typescript-eslint/recommended'
  - 'plugin:security/recommended'

parser: '@typescript-eslint/parser'

parserOptions:
  ecmaFeatures:
    jsx: true
  ecmaVersion: 12
  sourceType: module

plugins:
  - react
  - '@typescript-eslint'
  - security

rules: {
    "security/detect-eval-with-expression": "error"
}
```

Update eslint.config.mjs with security configuration

```
import globals from "globals";
import pluginJs from "@eslint/js";
import pluginReact from "eslint-plugin-react";
import pluginSecurity from "eslint-plugin-security";

/** @type {import('eslint').Linter.Config[]} */
export default [
  {files: ["**/*.{js,mjs,cjs,jsx}"]},
  {languageOptions: { globals: globals.browser }},
  pluginJs.configs.recommended,
  pluginReact.configs.flat.recommended,
  {
    plugins: {
      security: pluginSecurity
    },
    rules: {
      ...pluginJs.configs.recommended.rules,
      ...pluginReact.configs.flat.recommended.rules,
      "security/detect-eval-with-expression": "error",
    }
  }
];
```

Testing

If you are unable to get ESLint-Plugin-Security working for your project, you can test your config with a simpler project or test.js class.

For example, you can fork another javascript project for testing:

https://github.com/jenkins-docs/simple-node-js-react-npm-app

Or you can use a test.js class:

```
// test.js
const expression = '1 + 1';
eval(`console.log(${expression})`); // This evaluates the expression 1 + 1
```

And test locally using:

```
npx eslint test.js
```

```
npx eslint --debug test.js

npx eslint test.js --format=@microsoft/eslint-formatter-sarif --output-
file=reports/eslint-results.sarif
```

4.2 eslint-plugin-security-node plugin

The eslint-plugin-security-node plugin focused on **Node.js security** vulnerabilities.

- Detects issues like:
    - Insecure HTTP headers
    - Improper handling of user input

Please try out this plugin and adapt it for your project, especially if you are using NodeJS.

The full configuration for this plugin is not provided in this lab document.

Installation of eslint-plugin-security-node

```
npm install eslint-plugin-security-node --save-dev
```

Add to `.eslintrc.js extend:`

```
- 'plugin:security-node/recommended'
```

Add to `.eslintrc.js plugins:`

```
- security-node
```

4.3 eslint-plugin-no-unsanitized plugin

Helps prevent Cross-Site Scripting (XSS) by flagging improper DOM manipulation.

Please try out this plugin and adapt it for your project.

The full configuration for this plugin is not provided in this lab document.

Installation of eslint-plugin-no-unsanitized

```
npm install eslint-plugin-no-unsanitized --save-dev
```

Add to .eslintrc.js extend:

```
- 'plugin:no-unsanitized/recommended'
```

Add to .eslintrc.js plugins:

```
- no-unsanitized
```

5. Github CodeQL

To scan your repository for vulnerabilities using CodeQL, update and use the GitHub Actions workflow file (codeql.yml) provided to you.

6. Lab Notes

1. While ESLint helps with security best practices, it doesn't replace security-focused static analysis tools like:
    1. Semgrep – Lightweight, customizable security scanning.
    2. SonarQube – Detects security vulnerabilities in JavaScript/TypeScript.
2. Some other common tools that supports SARIF format include:
    1. Flake8 for Python
    2. Checkstyle for Java
    3. PMD for Java and Apex
    4. SonarCloud for multi-language analysis
3. We will explore SonarQube in the next lab.

7. Reference

Github SARIF format: https://docs.github.com/en/code-security/code-scanning/integrating-with-code-scanning/sarif-support-for-code-scanning

ESLint: ESLint

ESLint Plugin Security: https://www.npmjs.com/package/eslint-plugin-security

ESLint Plugin Security Node: https://www.npmjs.com/package/eslint-plugin-security-node

ESLint Plugin No Unsanitized: https://www.npmjs.com/package/eslint-plugin-no-unsanitized

Github CodeQL: https://github.com/github/codeql-action

END OF DOCUMENT