

Documentation: Backend & Frontend Vulnerability Testing with CodeQL

1. Objective

The objective of this task was to introduce **sample vulnerabilities** into both the backend and frontend of the **AutoAudit project**, then verify whether GitHub's **CodeQL analysis** and **CI/CD workflows** were able to detect them. This process simulates real-world scenarios where insecure coding practices could be introduced into financial/compliance systems like AutoAudit, which deal with **data processing, audits, and user inputs**.

2. Backend Testing

2.1 Vulnerabilities Added

Two vulnerabilities were added under backend/tests/ to reflect common risks in backend systems like AutoAudit:

1. **SQL Injection**
 - **Description:** SQL Injection occurs when user input is concatenated directly into SQL queries without sanitization, allowing attackers to manipulate queries and access unauthorized data.
 - **Why Chosen:** SQL Injection is one of the most common backend API vulnerabilities and directly applies to AutoAudit's backend where queries are processed.
 - **Dataset Reference:** Adapted from GitHub dataset [Broken-Vulnerable-Code-Snippets – SQL Injection](#).
 - **Implementation:** Created a test file sql-injection.js under backend/tests/ with intentionally unsafe string concatenation queries.

The screenshot shows a code editor interface with the title bar "AutoAuditN". The left sidebar is the "EXPLORER" view, showing a project structure with files like .github, backend-api, tests, main.py, test-server.js, README.md, requirements.txt, DevOps, engine, env, frontend, feature-cards-metric-tiles, feature-navbar, public, src, tests, insecure-eval.js, and xss-test.js. The right pane is the "CODE" view, displaying a file named "sql-injection.js". The code contains several lines of JavaScript, including database queries and middleware definitions. A red X icon in the gutter indicates a linting error on line 2, stating "Linting error intentionally added: unused variable 'noor'". Another red X icon on line 3 indicates a vulnerability: "SQL injection via string concatenation (unsafe)". The bottom status bar shows "Ln 39, Col 1" and other standard editor settings.

```
You, 11 hours ago | 1 author (You)
1 // backend-api/backend > tests > JS sql-injection.js > ...
2 // X Linting error intentionally added: unused variable 'noor'
3 // X Vulnerability: SQL injection via string concatenation (unsafe)
4
5 const express = require('express');
6 const router = express.Router();
7 const sqlite3 = require('sqlite3').verbose();
8
9 // unused var to trigger ESLint no-unused-vars
10 let noor = "noor";
11
12 const db = new sqlite3.Database(':memory:');
13
14 // create a tiny table for demonstration
15 db.serialize() => {
16   db.run("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT)");
17   db.run("INSERT INTO users (name) VALUES ('Alice')");
18   db.run("INSERT INTO users (name) VALUES ('Bob')");
19 };
20
21 // Vulnerable endpoint: user input concatenated directly into SQL
22 router.get('/user', (req, res) => {
23   const userId = req.query.id;
24   // UNSAFE: direct string concatenation allows SQL injection
25   const query = `SELECT * FROM users WHERE id = ${userId}`;
26
27   db.all(query, [], (err, rows) => {
28     if (err) {
29       res.status(500).send('Error');
30       return;
31     }
32     res.json(rows);
33   });
34
35 module.exports = router;
36
37
38 // test trigger for CodeQL
39
```

Figure 1: SQL-Injection Code

Commit Link -

<https://github.com/nvirdi21/AutoAuditN/commit/900fc16ae1cf92509e21ecab9207c715926608d6>

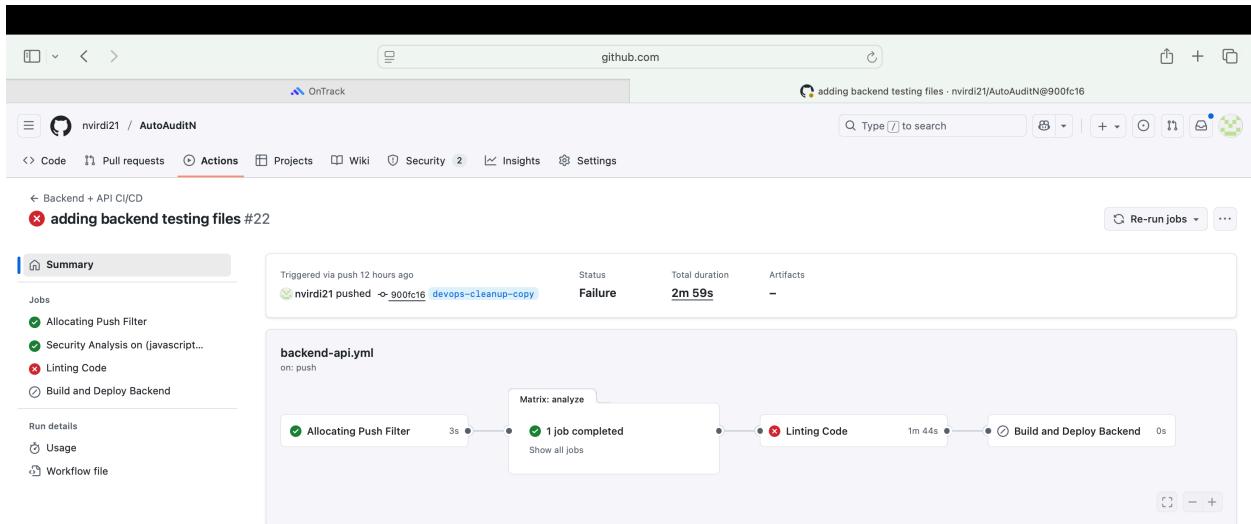


Figure 2: Backend CI — Linting failed (intentional lint error added for testing)

Link [Backend CI/CD] - <https://github.com/nvirdi21/AutoAuditN/actions/runs/17849115964>

The screenshot shows a GitHub repository page for 'nvirdi21 / AutoAuditN'. The 'Security' tab is selected, displaying a list of code scanning alerts. The first alert is titled 'Missing rate limiting' and is categorized under 'Code scanning alerts' with a status of 'In branch' and a timestamp of '10 hours ago'. The alert details a SQL injection vulnerability found in 'backend-api/backend/tests/sql-injection.js:22'. The code snippet shows a direct string concatenation in a query:19 });
20
21 // Vulnerable endpoint: user input concatenated directly into SQL
22 router.get('/user', (req, res) => {
23 const userId = req.query.id;
24 // UNSAFE: direct string concatenation allows SQL injection
25 const query = `SELECT * FROM users WHERE id = \${userId}`;
26
27 db.all(query, [], (err, rows) => {
28 if (err) {
29 res.status(500).send('Error');
30 return;
31 }
32 res.json(rows);
33 });
34 });
35
36 module.exports = router;A callout box highlights the line 'const query = `SELECT * FROM users WHERE id = \${userId}`;' with the text 'This route handler performs a database access, but is not rate-limited.' To the right of the code, the alert summary includes:

- Severity**: High
- Affected branches**: devops-cleanup-copy (First detected about 12 hours ago)
- Development**: Link a branch, pull request, or Create a new branch to start working on this alert.
- Tags**: security
- Weaknesses**: CWE-307, CWE-400, CWE-770

Below the main alert, there's a 'Tool' section with 'Rule ID: CodeQL js/missing-rate-limiting' and a 'Query' link. A note states: 'HTTP request handlers should not perform expensive operations such as accessing the file system, executing an operating system command or interacting with a database without limiting the rate at which requests are accepted. Otherwise, the application becomes vulnerable to denial-of-service attacks where an attacker can cause the application to crash or become unresponsive by issuing a large number of requests at the same time.' A 'Show more' link is present.

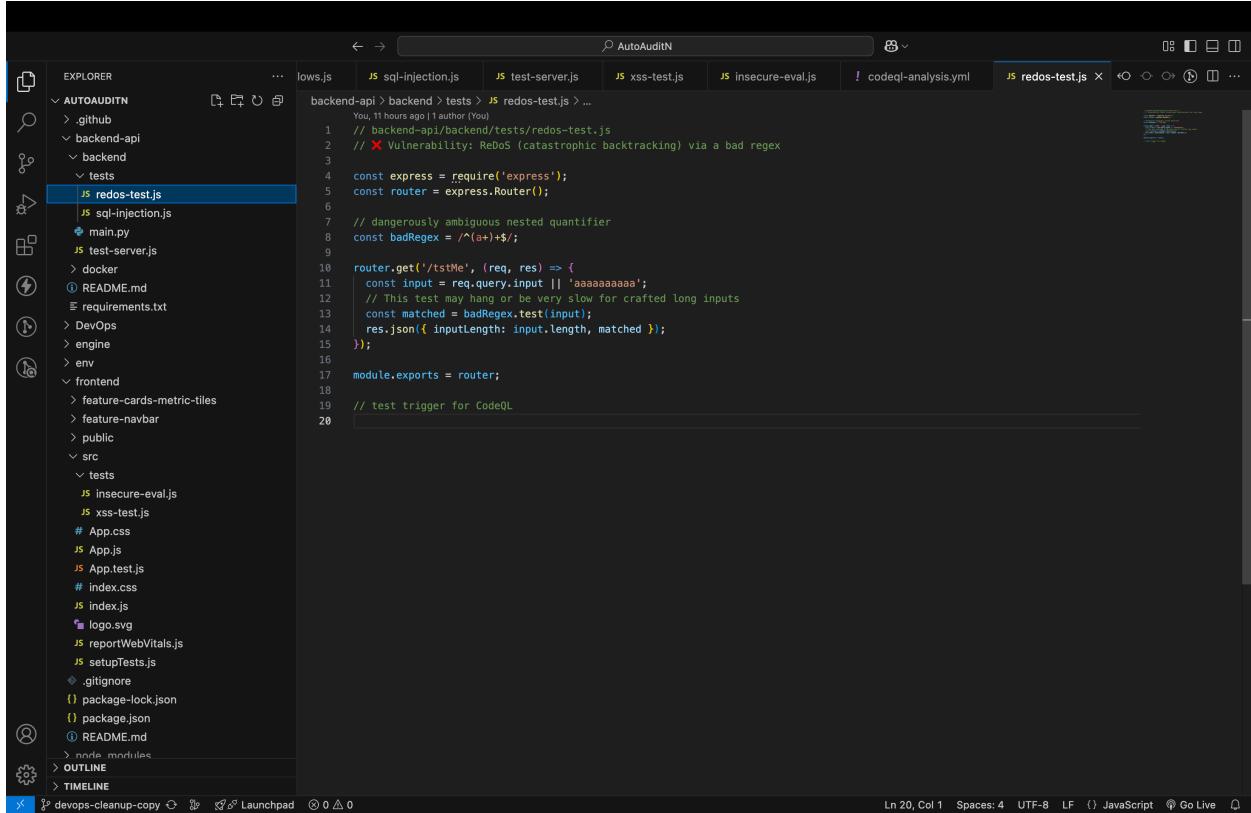
At the bottom, a timeline shows the alert was first detected in commit 12 hours ago, with a note about adding backend testing files and a specific commit hash: 980fc16.

Figure 3: CodeQL security alert confirming SQL Injection Vulnerability

Link to the Security tab where CodeQL is flagging the security alerts -
<https://github.com/nvirdi21/AutoAuditN/security/code-scanning/5>

2. ReDoS (Regular Expression Denial of Service)

- **Description:** ReDoS exploits poorly written regex expressions that cause excessive backtracking, leading to denial of service by consuming high CPU.
- **Why Chosen:** Regex is heavily used in backend validation and logging in AutoAudit, so this vulnerability is directly relevant.
- **Dataset Reference:** Adapted from GitHub dataset [Broken-Vulnerable-Code-Snippets – ReDoS](#).
- **Implementation:** Created a test file redos-test.js under backend/tests/ with a vulnerable regex `^(a+)+$/` that hangs on long malicious input.



```
You, 11 hours ago | author (You)
1 // backend-api/backend/tests/redos-test.js
2 // ✘ Vulnerability: ReDoS (catastrophic backtracking) via a bad regex
3
4 const express = require('express');
5 const router = express.Router();
6
7 // dangerously ambiguous nested quantifier
8 const badRegex = /^(a+)+$/;
9
10 router.get('/tstMe', (req, res) => {
11   const input = req.query.input || 'aaaaaaaaaa';
12   // This test may hang or be very slow for crafted long inputs
13   const matched = badRegex.test(input);
14   res.json({ inputLength: input.length, matched });
15 });
16
17 module.exports = router;
18
19 // test trigger for CodeQL
20
```

Figure 4: ReDoS Code

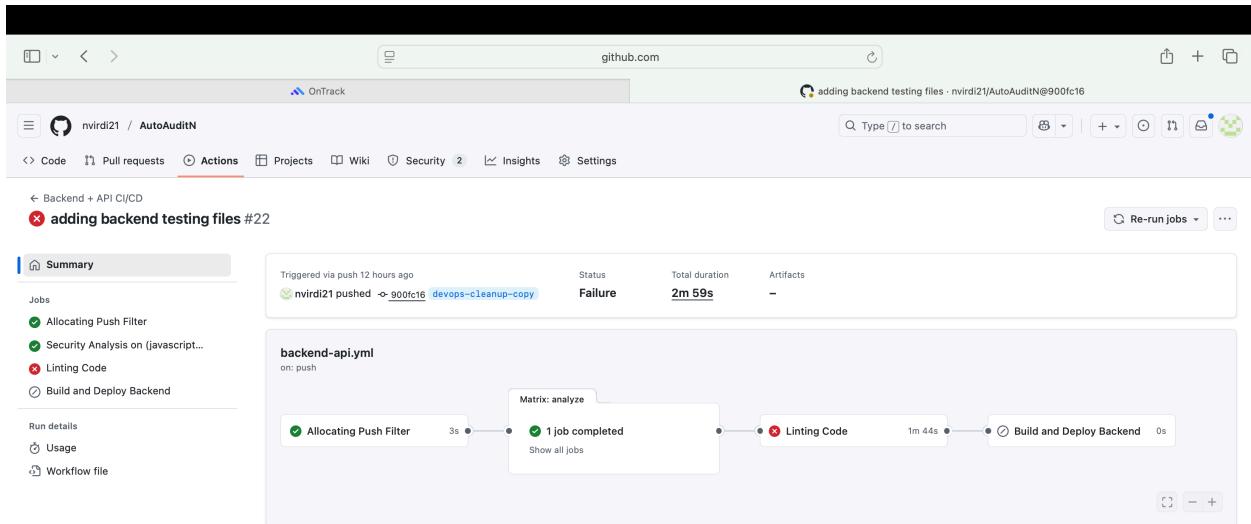


Figure 5: Backend CI — Linting failed (intentional lint error added for testing)

Link [Backend CI/CD] - <https://github.com/nvirdi21/AutoAuditN/actions/runs/17849115964>

The screenshot shows a GitHub repository page for 'nvirdi21 / AutoAuditN'. The 'Security' tab is selected, and the 'Code scanning alerts' section is active. A specific alert is highlighted: 'Polynomial regular expression used on uncontrolled data - Code scanning alert #6 - nvirdi21/AutoAuditN'. The alert details a 'Rule js/polynomial-redos is not supported by Copilot Autofix for CodeQL' found in the 'devops-cleanup-copy' branch. The code snippet shows a regular expression that can be slow for crafted long inputs. The alert is categorized as 'High' severity, affecting the 'devops-cleanup-copy' branch. It includes links for 'Dismiss alert', 'Generate fix', and 'View source'. The alert also lists associated tags like 'security' and weaknesses such as CWE-1333, CWE-400, and CWE-730.

Figure 6: *CodeQL security alert confirming ReDos Vulnerability*

Link [Security Tab] - <https://github.com/nvirdi21/AutoAuditN/security/code-scanning/6>

2.2 Issues Faced & Fixes

- **Initial Problem:**
 - GitHub workflows flagged intentional **linting errors** (e.g., unused variables), but CodeQL was not flagging the vulnerabilities.
- **Fixes:**
 1. Added codeql-analysis.yml workflow file.
 - ◆ *Why: By introducing this workflow file, integrated CodeQL into the CI/CD pipeline, enabling automated security scans on each commit. This allowed the injected vulnerabilities (SQL Injection, ReDoS, XSS, etc.) to be properly detected and flagged under the repository's Security → Code scanning alerts tab.*
 2. Still no alerts visible in the **Security tab** initially.
 3. Root cause: Security tab was filtering **only main branch**.
 4. Switching the filter to my working branch (devops-cleanup-copy) showed the alerts correctly.

The screenshot shows a GitHub repository interface with the following details:

- EXPLORER:** Shows files like `flows.js`, `sql-injection.js`, `test-server.js`, `xss-test.js`, `insecure-eval.js`, and `codeql-analysis.yml`.
- CODEQL ANALYSIS:** The `codeql-analysis.yml` file is selected and displayed in the main editor area.
- Content of codeql-analysis.yml:**

```
name: "CodeQL"
on:
  push:
    branches: [ main, devops-cleanup-copy ]
  pull_request:
    branches: [ main, devops-cleanup-copy ]
  schedule:
    - cron: '0 2 * * 1' # every Monday at 2am
  jobs:
    analyze:
      name: Analyze
      runs-on: ubuntu-latest
      permissions:
        actions: read
        contents: read
        security-events: write
      strategy:
        fail-fast: false
      matrix:
        language: [ 'javascript', 'python' ] # add/remove depending on your repo
      steps:
        - name: Checkout repository
          uses: actions/checkout@v4
        - name: Initialize CodeQL
          uses: github/codeql-action/init@v3
          with:
            languages: ${{ matrix.language }}
        - name: Autobuild
          uses: github/codeql-action/autobuild@v3
        - name: Perform CodeQL Analysis
          uses: github/codeql-action/analyze@v3
```
- Bottom Status Bar:** Shows "Ln 39, Col 1" and "Spaces: 2, UTF-8, LF, YAML, Go Live".

Figure 7: CodeQL Analysis Code

Commit Link -

<https://github.com/nvirdi21/AutoAuditN/commit/6c9c156238f76f5654fb46f44c2ae50b9118587e>

The screenshot shows the GitHub Code scanning alerts page for the repository `nvidia21/AutoAuditN`. The main content area displays a list of 8 open code scanning issues, all categorized under "Code injection" (Critical). The issues are listed as follows:

- #9 opened 11 hours ago - Detected by CodeQL in frontend/.../tests/insecure-eval.js :12
- #8 opened 11 hours ago - Detected by CodeQL in backend-api/.../tests/redis-test.js :14
- #10 opened 11 hours ago - Detected by CodeQL in frontend/.../tests/xss-test.js :16
- #7 opened 12 hours ago - Detected by CodeQL in backend-api/.../tests/redis-test.js :8
- #6 opened 12 hours ago - Detected by CodeQL in backend-api/.../tests/sql-injection.js :13
- #5 opened 12 hours ago - Detected by CodeQL in backend-api/.../tests/sql-injection.js :27
- #4 opened 12 hours ago - Detected by CodeQL in backend-api/.../tests/sql-injection.js :27
- #1 opened last week - Detected by CodeQL in security/aa_ui/ui.html :157

A sidebar on the left provides navigation links: Overview, Reporting, Policy, Advisories, Vulnerability alerts, Dependabot, Code scanning (selected), and Secret scanning. The GitHub logo and URL "github.com" are visible at the top.

Figure 8: Selecting the correct flutter

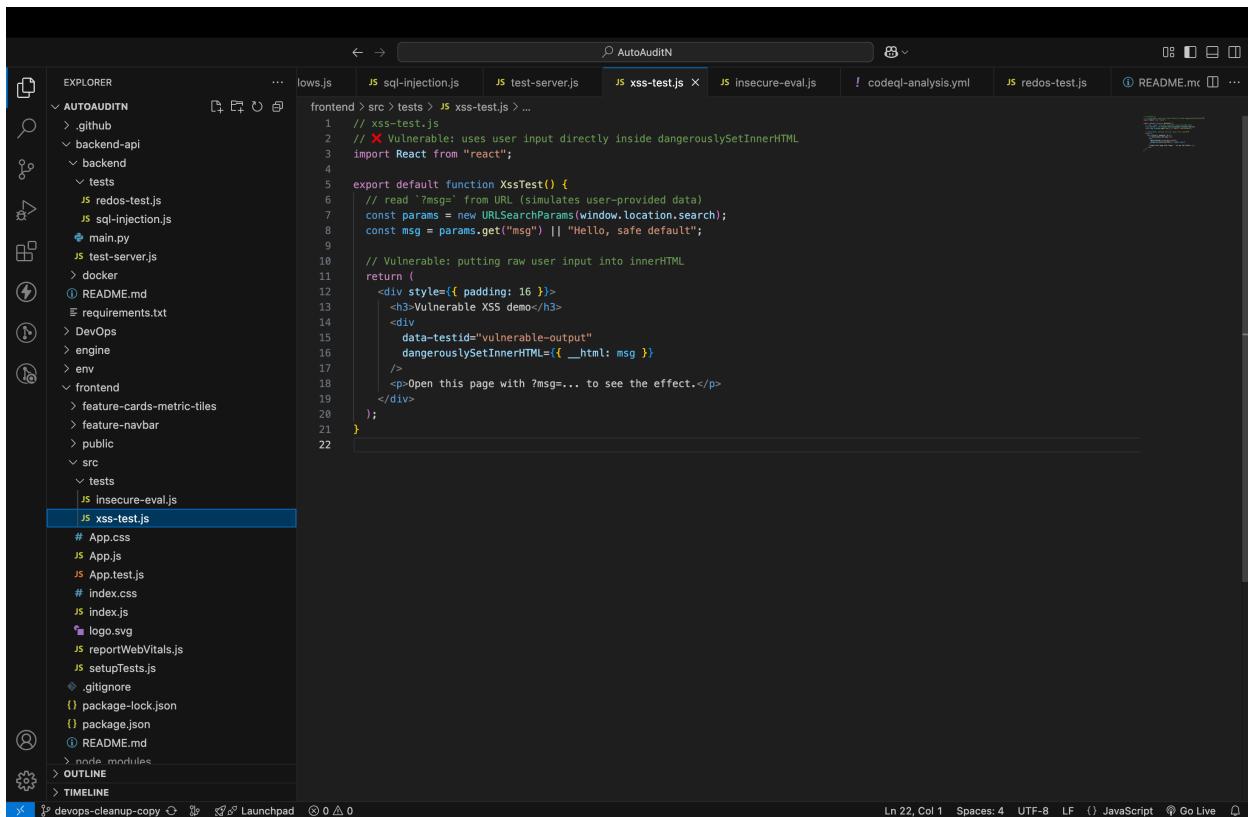
3. Frontend Testing

3.1 Vulnerabilities Added

Two vulnerabilities were added under frontend/src/tests/ to reflect common risks in frontend systems like AutoAudit:

1. Cross-Site Scripting (XSS)

- **Description:** XSS occurs when user input is rendered into the DOM without sanitization, allowing attackers to inject malicious scripts.
- **Why Chosen:** XSS is a frequent vulnerability in frontend projects. AutoAudit's frontend (React-based) dynamically renders user-facing data, making it relevant.
- **Dataset Reference:** Reviewed GitHub dataset [Broken-Vulnerable-Code-Snippets - XSS](#).
- **Note:** The dataset code does not directly match React (used in AutoAudit). The concept was adapted into a React component xss-test.js under frontend/src/tests/ to simulate unsafe rendering of user input using dangerouslySetInnerHTML.



```
// xss-test.js
// ✘ Vulnerable: uses user input directly inside dangerouslySetInnerHTML
import React from "react";

export default function XssTest() {
  // read '?msg=' from URL (simulates user-provided data)
  const params = new URLSearchParams(window.location.search);
  const msg = params.get("msg") || "Hello, safe default";

  // Vulnerable: putting raw user input into innerHTML
  return (
    <div style={{ padding: 16 }}>
      <h3>Vulnerable XSS demo</h3>
      <div data-testid="vulnerable-output" dangerouslySetInnerHTML={{ __html: msg }}>
        />
        <p>Open this page with ?msg=... to see the effect.</p>
      </div>
    </div>
  );
}

# App.css
JS App.js
JS App.test.js
# index.css
JS index.js
# logo.svg
JS reportWebVitals.js
JS setupTests.js
# .gitignore
# package-lock.json
# package.json
# README.md
# node_modules
> OUTLINE
> TIMELINE
```

Figure 9: XSS Test Code

Commit Link -

<https://github.com/nvirdi21/AutoAuditN/commit/8a533118fce4caa2d078f873649af272e468d4c>

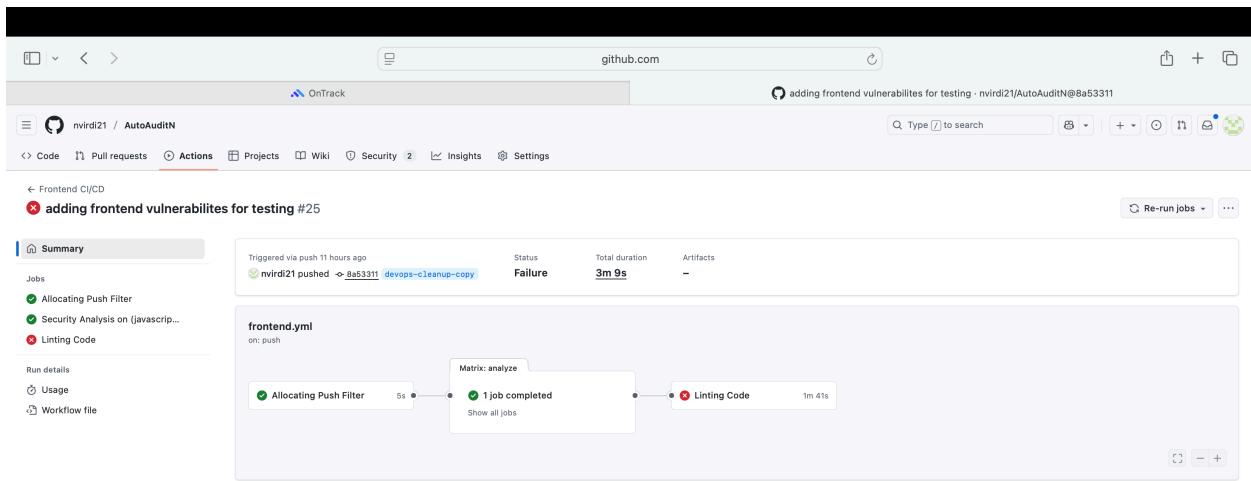


Figure 10: Frontend CI — Linting failed (intentional lint error added for testing)

Link [Frontend CI/CD] - <https://github.com/nvirdi21/AutoAuditN/actions/runs/17850315156>

The screenshot shows a GitHub repository page for 'nvirdi21 / AutoAuditN'. The 'Security' tab is selected, displaying a list of code scanning alerts. The first alert is titled 'Client-side cross-site scripting' and is categorized as 'High' severity. It points to a file 'frontend/src/tests/xss-test.js' at line 16, which contains the following code snippet:

```
frontend/src/tests/xss-test.js:16 ↗ Test
13     <h3>Vulnerable XSS demo</h3>
14     <div
15       data-testid="vulnerable-output"
16       dangerouslySetInnerHTML={(_html: msg)}
Cross-site scripting vulnerability due to user-provided value.
CodeQL Show paths
17   />
18   <p>Open this page with ?msg=... to see the effect.</p>
19 </div>
```

The alert details section includes:

- Severity:** High
- Affected branches:** devops-cleanup-copy (First detected about 12 hours ago)
- Development:** Link a branch, pull request, or Create a new branch to start working on this alert.
- Tags:** security
- Weaknesses:** CWE-79, CWE-116

At the bottom, there is a note: 'First detected in commit 11 hours ago' and a link to 'adding frontend vulnerabilities for testing'.

Figure 11: CodeQL security alert confirming XSS Vulnerability

Link [Security Tab] - <https://github.com/nvirdi21/AutoAuditN/security/code-scanning/10>

2. Insecure eval Usage

- **Description:** Using eval() to execute user-provided input allows arbitrary code execution, making it a critical frontend security risk.
- **Why Chosen:** Although React projects do not normally use eval(), this vulnerability was added to demonstrate detection of code execution risks in JavaScript projects.
- **Dataset Reference:** Reviewed GitHub dataset [Broken-Vulnerable-Code-Snippets – Code Injection](#).
- **Note:** The dataset uses PHP and NodeJS examples, which do not match React directly. The concept was adapted into a React file insecure-eval.js under frontend/src/tests/ where eval() executes unsanitized input.

```
frontend > src > tests > JS insecure-eval.js > ...
1 // insecure-eval.js
2 // ✖ Vulnerable: running eval() on user input
3 // This file exports a tiny function and uses a query param to demonstrate misuse.
4
5 export function runUserCodeFromQuery() {
6     try {
7         const params = new URLSearchParams(window.location.search);
8         const code = params.get("code") || "2 + 2"; // user-supplied code
9
10        // Vulnerable: evaluate arbitrary user code
11        // eslint-disable-next-line no-eval
12        const result = eval(code);
13
14        return { ok: true, code, result };
15    } catch (err) {
16        return { ok: false, error: String(err) };
17    }
18}
```

Figure 12: Insecure - eval Code

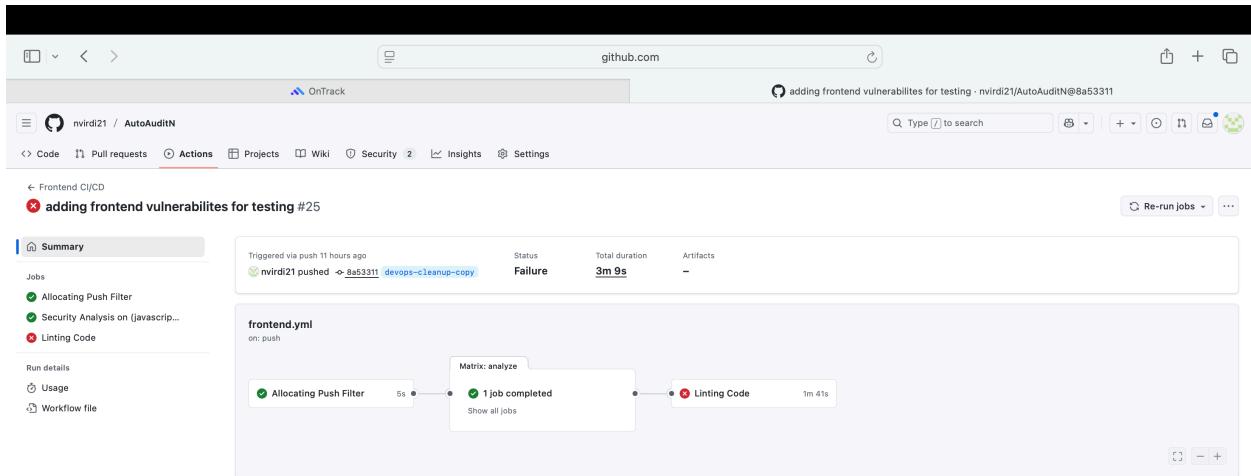


Figure 13: Frontend CI — Linting failed (intentional lint error added for testing)

Link [Frontend CI/CD] - <https://github.com/nvirdi21/AutoAuditN/actions/runs/17850315156>

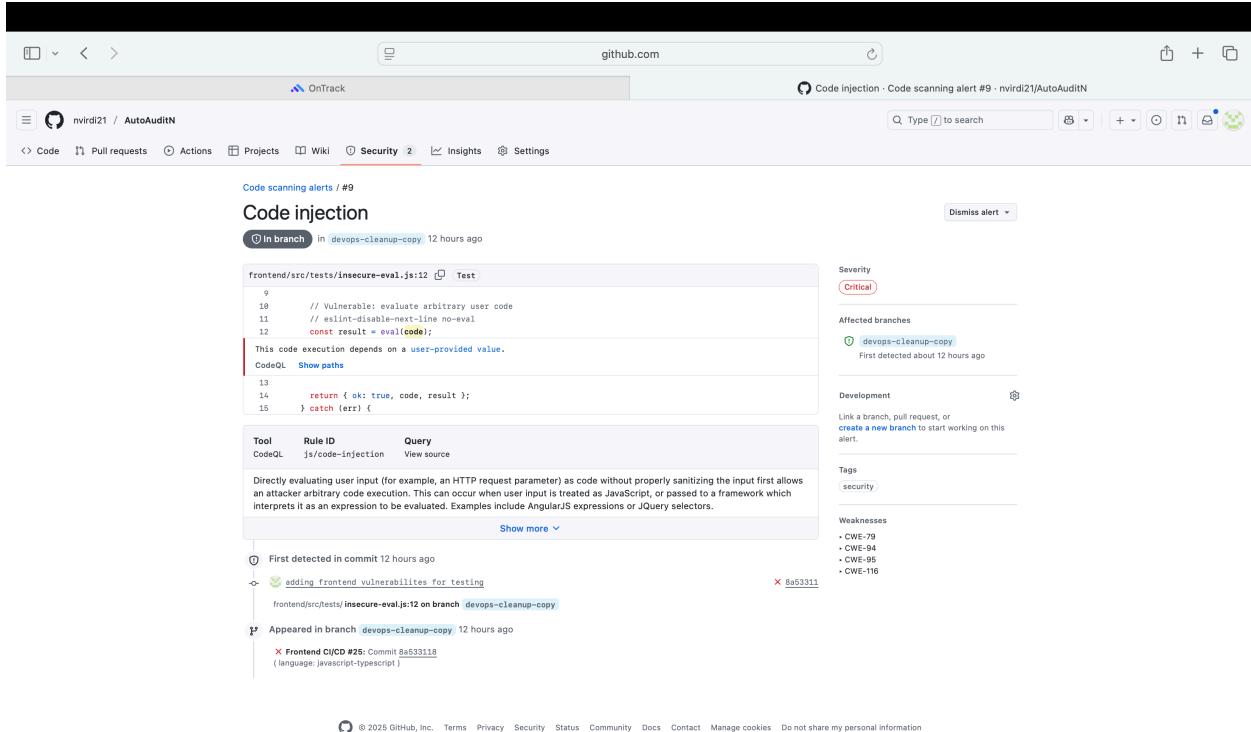


Figure 14: Frontend CI — Linting failed (intentional lint error added for testing)

Link [Security Tab] - <https://github.com/nvirdi21/AutoAuditN/security/code-scanning/9>

3.2 Issues Faced & Fixes

- Same CodeQL flagging issue as backend.
 - Once vulnerabilities were pushed, CodeQL flagged the eval() injection as Critical and the XSS vulnerability as High under the *devops-cleanup-copy* branch filter.
-

4. Results

- **Backend:** SQL Injection + ReDoS vulnerabilities successfully added and flagged by CodeQL after branch filter adjustment.
 - **Frontend:** Code Injection (eval) flagged as Critical; XSS vulnerability introduced for testing.
 - Demonstrated full cycle: introduce vulnerabilities → commit → GitHub Actions workflow → CodeQL analysis → Security tab alerts.
-