AI Coder - CS467 Capstone Project (Spring 2025)

Overview

AI Coder is a capstone project exploring the integration of generative AI tools—such as ChatGPT and GitHub Copilot—throughout the entire software development lifecycle. Our focus is not just on the resulting software product, but on documenting the role AI plays as a collaborative assistant in development, penetration testing, and secure coding.

To this end, we will develop a full-stack web application that includes user authentication, persistent data storage, and a responsive frontend. Once built, the app will be subjected to penetration testing simulating the OWASP Top 10 vulnerabilities. Every stage of development and testing will incorporate AI assistance, and our observations will be documented in detail.

Table of Contents

Team Members

- **Alexander Ngo** – ngoalex@oregonstate.edu
- **Samuel David Levy** - Levys@oregonstate.edu
- **Brian Heath Anderson** - anderbr4@oregonstate.edu

⚠ Disclaimer

This application is **intentionally vulnerable** and should **only be used for educational or testing purposes in a controlled environment**. Do **NOT** deploy this to any public-facing server.

React WebApp

📦 Prerequisites

- Node.js and npm installed (Download Node.js)
- PostgreSQL installed (Download PostgreSQL)

🐘 Setting Up PostgreSQL

1. Download and install PostgreSQL from the official page:
   👉 https://www.postgresql.org/download/

2. During setup, you'll be asked to:

   - Choose a **username** (default: `postgres`)
   - Set a **password** — write this down

3. After installation:

   - You can use **pgAdmin** or the `psql` CLI to manage your database

4. (Optional) Create a new PostgreSQL user and database:

   ```
   CREATE USER aicoder WITH PASSWORD 'securepassword';
   ALTER USER aicoder CREATEDB;
   CREATE DATABASE aicoder_db OWNER aicoder;
   ```

5. Update the environment variables in both website-react/server/.env and website-vulnerable/server/.env

---

🚀 Set Up the Web App

1. Clone the repository or download the source code.

2. Navigate to root directory (./), install all dependencies:

   ```
   npm run setup
   ```

   This will automatically install all required packages in all required locations. If you want to do it manually, do the following:

   ```
   npm install
   ```

   then proceed to step 3

3. This step is only if you decide to install packages manually. You will need to perform this in both versions of the webpage (./website-react and ./webbsite-vulnerable). Make sure you are in each of the aforementioned directories before you run this:

   ```
   cd client
   npm install
   cd ../server
   npm install
   ```

▶️ Run the Web App

In each web app root directory:

```
npm run start
```

This will:

- For the secure server by default:
  - Start the backend server at http://localhost:5000
  - Start the React frontend at http://localhost:3000
  - Automatically initialize the required PostgreSQL tables (users, todos)
- For the vulnerable server by default:
  - Start the backend server at http://localhost:5001
  - Start the React frontend at http://localhost:4000
  - Automatically initialize the required PostgreSQL tables (usersv, todosv)

🔁 Reset the Database

To delete all tables created by the app, navigate to either website directory:

```
npm run reset
```

📄 Where to Find Logs

- Log file: ./{website-react or website-vulnerable}/sever.log
- Events logged include but not limited to:
  - User registrations and logins
  - Todo list fetches
  - Todo creations, updates, and deletions
  - Any backend request

🔐 Security Test Modules

This section describes how to run various security tests against the vulnerable web application.

> ⚠ **Note:** These tests require the at least one version website to be running.
> You should be in the corresponding website directory when you attempt these commands. Commands also assume that default urls are not changed. So if you manually selected different URLs, please update them in the package.json files in the root of each website directory. Some vulnerabilities can be

> easily run using the new testing frontend that can be accessed from either running website via the link at the bottom of the page. You can read more about each vulnerability here: Vulnerability Directory

---

🗂 BAC Test – Broken Access Control

This test simulates Broken Access Control by using an insecure backend (`BAC_index.js`) that does **not validate user ownership**. Users can retrieve or modify other users' todos.

🧪 Testing

```
npm run test:bac
```

- Sends requests as one user to view or modify another user's data.
- A successful test demonstrates unauthorized access.

🧪 UVC Test – Using Components with Known Vulnerabilities

This test demonstrates **CVE-2020-7660**, which affects `serialize-javascript@1.6.1`. It allows unsafe JavaScript functions to be injected into HTML contexts, enabling XSS.

🧪 Testing

```
npm run test:uvc
```

- Injects a crafted payload into the backend using `serialize-javascript`.
- A successful test causes an XSS-like effect via serialized output.

📄 LogChecker – Insufficient Logging and Monitoring

This test verifies whether **suspicious activity is logged**. The vulnerable system conditionally disables logging via `.env`.

🧪 Testing

```
npm run test:log
```

- Performs sensitive operations (e.g. unauthorized login attempts).
- Then checks whether they appear in `server.log`.
- Failing to record key events demonstrates insufficient monitoring.

🧬 ID Test – Insecure Deserialization

This test attempts to exploit **unsafe deserialization** by sending a maliciously crafted payload that, if deserialized without validation, triggers side effects (e.g., writing a file).

🧪 Testing

```
npm run test:des
```

- Sends a deserialization payload to the backend.
- A successful test results in a file (`hacked.txt`) being created in the server directory.

🧨 XXE Test – XML External Entity Injection

This test simulates an **XXE (XML External Entity) injection attack**, where an attacker exploits unsafe XML parsers to gain access to sensitive files or internal resources. The vulnerable server accepts raw XML input without proper validation, making it a target for this type of exploit.

🧪 Testing

```
npm run test:xxe
```