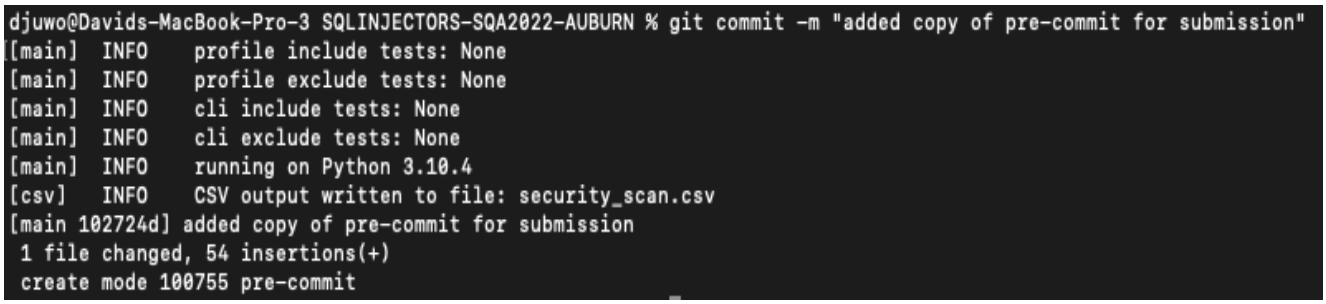
https://github.com/djuwon/SQLINJECTORS-SQA2022-AUBURN

David Jung:

The portion I worked on was the Git Hooks section. The objective was to create a git hook that reports all the security vulnerabilities in the project given python file changes. When thinking of making file changes and git, I thought back to the in class workshop where we worked with git and precommits. I realized that a hook I could utilize was the precommit where every time I make a file change and commit, the pre commit file would automatically run through a script execution. I combined that with the bandit library which scans through files and analyzes vulnerabilities within the file. I could then use that to output a CSV file containing said vulnerabilities. I added a bandit script to the pre commit that recursively searched through the TestOrchestrator4ML folder and outputted to a CSV file.



Chloe Coward:

I worked on adding fuzzing to the project. One challenge of this was deciding what arguments to use when fuzzing various methods. I was originally going to hard code in certain arguments, which would have given the same results every time the fuzzer was run unless the methods being fuzzed were changed. This might have been fine, but lackluster in what kind of bugs it could find. I had the idea to randomly choose function arguments from a list, and even randomly generate some of those values. This lets the fuzzer have much more dynamic behavior, and cover a wider variety of bugs. Setting up an action to run this was simple, as github provides basic actions like setting up a shell and python environment for you. A small sample of the output from the fuzzing action is included.

David Joy:

I worked on adding forensics to the project. I had to add these logging forensics to five methods, and selected:

- getDevCount from select_repos\dev_count.py
- getTopRepos from select_repos\dev_count.py
- generateUnitTest from generation\main.py
- generateAttack from generation\main.py
- identifyAlgo from generation\indentify_algo\main.py

I selected these methods as they all performed a significant task for the project (instead of being a minor helper method), had data worth logging, and some had errors that were just being printed to the screen. To decide what to log in addition to errors, I imagined I was going back to the method because I thought it had a defect, and imagined what intermediate results and program flow I would want to fix or verify the method.

I learned that attempting to add (meaningful) logging is a good way to assess your understanding of an existing/inherited project, since adding logging requires you to understand the dataflow and purpose of a program.