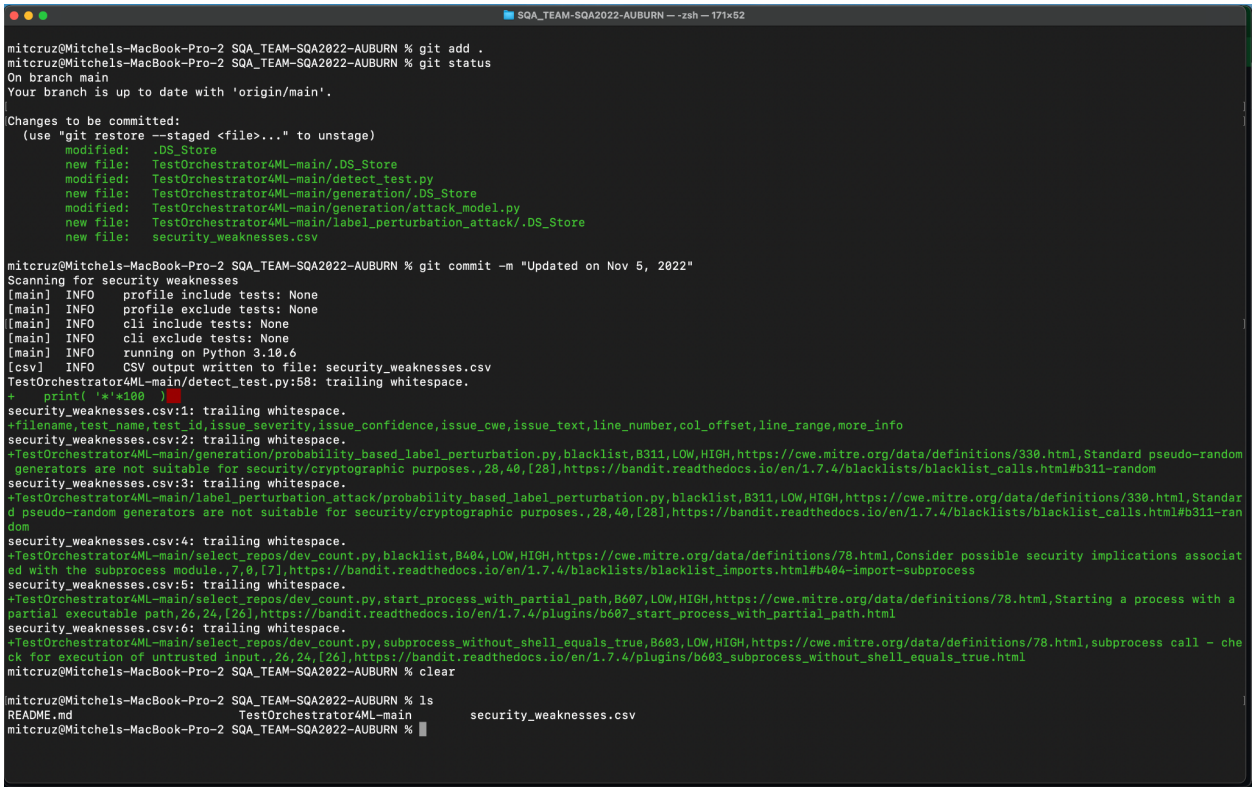
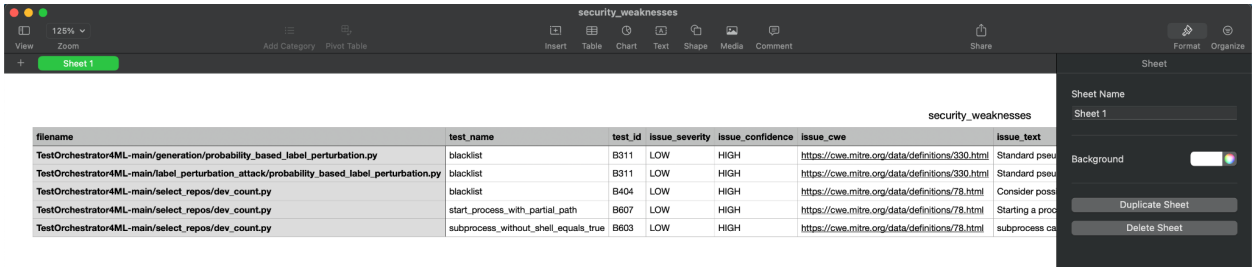
Section 4.a (Mitchel Santillan)

Activities Performed: Once our local repo had been connected to our remote repo, I navigated to the .git directory and went inside the hooks file. In there, I activated the pre-commit file and added a line which ran and reported all security weaknesses in the TestOrchestrator4ML-main directory in a csv file whenever any python file was changed and committed. I provided screenshots of the results of committing a random python file below and the resulting csv file containing the security weaknesses it identified. Because this does not allow me to push this local repo to the remote one because of the bandit statement in the pre-commit file, I disabled the pre-commit file and made a copy of this file and placed it in a directory called 4.a. along with the output csv file in the following path SQA_TEAM-SQA2022-AUBURN/

Lessons Learned: I learned that bandit is a much more useful tool than I originally thought. I had no idea it could be run on a specified directory which could then produce a csv file which is very useful in my opinion. I also had the chance to learn some additional terminal flags along the way. I initially did not know why my code would not push to my remote repo but I found out that it was because the bandit statement I included in the pre-commit file prevented me from doing so. Overall, this has been a great learning experience and helped me expand my knowledge on the topic.

Logs





security_weaknesses

filename	test_name	test_id	issue_severity	issue_confidence	issue_cwe	issue_text
TestOrchestrator4ML-main/generation/probability_based_label_perturbation.py	blacklist	B311	LOW	HIGH	https://cwe.mitre.org/data/definitions/330.html	Standard pseu
TestOrchestrator4ML-main/label_perturbation_attack/probability_based_label_perturbation.py	blacklist	B311	LOW	HIGH	https://cwe.mitre.org/data/definitions/330.html	Standard pseu
TestOrchestrator4ML-main/select_repos/dev_count.py	blacklist	B404	LOW	HIGH	https://cwe.mitre.org/data/definitions/78.html	Consider poss
TestOrchestrator4ML-main/select_repos/dev_count.py	start_process_with_partial_path	B607	LOW	HIGH	https://cwe.mitre.org/data/definitions/78.html	Starting a proc
TestOrchestrator4ML-main/select_repos/dev_count.py	subprocess_without_shell_equals_true	B603	LOW	HIGH	https://cwe.mitre.org/data/definitions/78.html	subprocess ca

Section 4.b (Jamie Whitehead)

Activities Performed

I went through the code that was given to us and chose 5 methods that I wanted to fuzz. I then created a fuzz.py file and imported the methods I wanted to fuzz into the file. I wrote methods to use bad data as inputs to these methods and the fuzz.py file will run using Github actions.

Lessons Learned

Fuzzing can be very helpful in figuring out security problems within your code and is a different way of testing your code. It can also be helpful in finding out how your code reacts to wrong inputs of data.

Logs

```
ModuleNotFoundError: No module named 'constants'
PS C:\Users\jrw10\Documents\SQA_TEAM-SQA2022-AUBURN\TestOrchestrator4ML-main> python fuzz.py
Error: Wrong value for algo_list.
Error: kVals has the wrong value.
Error: author_emails is not a list.
Error: predictions did not equal a list.
Error: Import_list is not a list.
PS C:\Users\jrw10\Documents\SQA_TEAM-SQA2022-AUBURN\TestOrchestrator4ML-main>
```

Section 4.c (SQA_Team)

Activities Performed: We created a file called forensics.py in the following path SQA_TEAM-SQA2022-AUBURN/TestOrchestrator4ML-main which produces a file called FORENSICS.LOG in the same directory. This log file will be populated every time you run a python file that we modified. The screenshots below are examples of us running a modified python file and data being logged in our log file. You can find the python files we modified along with their paths below.

File name: detect_test.py
Method altered: giveTimeStamp()
Path: SQA_TEAM-SQA2022-AUBURN/TestOrchestrator4ML-main

File name: attack_model.py
Method altered: calculate_k()
Path: SQA_TEAM-SQA2022-AUBURN/TestOrchestrator4ML-main/generation

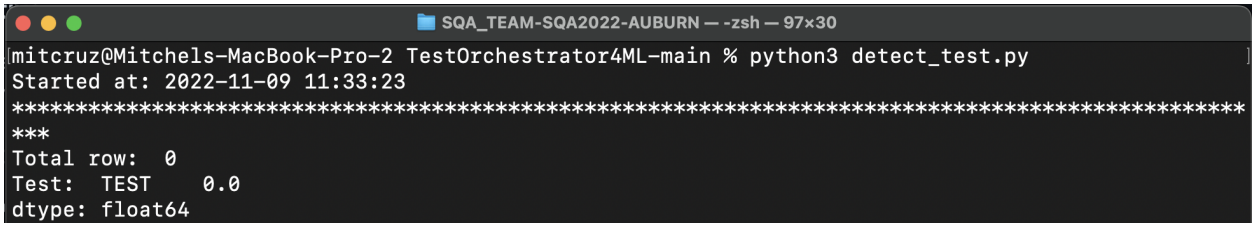
File name: py_parser.py
Method altered: getFunctionAssignments()
Path: SQA_TEAM-SQA2022-AUBURN/TestOrchestrator4ML-main/generation

File name: main.py
Method altered: giveTimeStamp()
Path: SQA_TEAM-SQA2022-AUBURN/TestOrchestrator4ML-main/detection

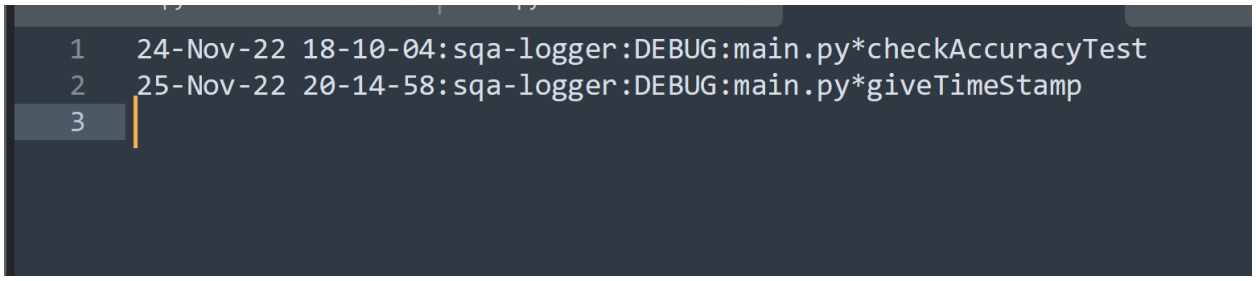
File name: main.py
Method altered: get_test_details
Path: SQA_TEAM-SQA2022-AUBURN/TestOrchestrator4ML-main/detection

Lessons Learned: During this section, we learned how to set up, record, and view logs that we placed throughout our code. We dug deeper into the logging module and found that it provides a lot of flexibility and functionality that we have not utilized yet. We learned a bit about the different kinds of logging levels and found that we can even create our own levels. We played around with the formatting but decided to keep it simple for this project and use a similar formatting as workshop 9. Overall, it gave us good experience with logging and forensics and allowed us to learn more about other modules that we have not used before.

Logs


SQA_TEAM-SQA2022-AUBURN — -zsh — 97×30
mitcruz@Mitchels-MacBook-Pro-2 TestOrchestrator4ML-main % python3 detect_test.py
Started at: 2022-11-09 11:33:23
****
***
Total row: 0
Test: TEST   0.0
dtype: float64


FORENSICS.LOG
Reveal
Now
Clear
Reload
Share
Search
09-Nov-22 11-21-52:sqa-logger:DEBUG:detect_test.py*giveTimeStamp
09-Nov-22 11-22-08:sqa-logger:DEBUG:detect_test.py*giveTimeStamp
09-Nov-22 11-22-08:sqa-logger:DEBUG:detect_test.py*giveTimeStamp
09-Nov-22 11-23-12:sqa-logger:DEBUG:detect_test.py*giveTimeStamp
09-Nov-22 11-33-23:sqa-logger:DEBUG:detect_test.py*giveTimeStamp


1
2
3
24-Nov-22 18-10-04:sqa-logger:DEBUG:main.py*checkAccuracyTest
25-Nov-22 20-14-58:sqa-logger:DEBUG:main.py*giveTimeStamp