COMP6710  
Dr. Akond Rahman  
26 April 2023

BIMM-SQA2023-Project Report  
*Brayden Simpson, Ian Larson, Matthew Ippolito, Melvin Moreno*

**Activities:**

*Git Hook*

We created a Git Hook that runs and reports all security weaknesses in the project into a CSV file whenever a Python file is changed and committed. We did this by creating a **.pre-commit-config.yaml** file and configuring our security vulnerability tool **Bandit** to work on commit and output to the *output.csv* file.

Detailed Instructions:

1. Modify any file  
   Text

   Description automatically generated
2. Git add the files  
   Text

   Description automatically generated
3. Git commit  
   Text

   Description automatically generated

* Bandit runs recursively on all files. If you want to work only on the file you committed, change the .pre-commit-config.yaml file and remove the --recursive flag
* Because bandit failed the git hook, you cannot commit or push changes. To get past this, add the --no-verify flag at the end of your commit.

1. Get results  
   Table, calendar

   Description automatically generated

*Fuzzing*

We created a ‘fuzz.py’ file that will automatically fuzz 5 Python methods of our choice. ‘fuzz.py’ will be automatically executed from GitHub actions. We reported the bugs we discovered.

Detailed Activities:

1. We selected the 5 methods to perform fuzzing on:  
   - Graphtaint.py: getYAMLFiles  
   - Graphtain.py: getSHFiles  
   - Scanner.py: isValidUsername  
   - Scanner.py: isValidPassword  
   - Scanner.py: isValidKey
2. Created the ‘fuzz.py’ file to send random values to these methods to see how they would react.  
   Text

   Description automatically generated
3. Setup the workflow to run ‘fuzz.py’ automatically using ‘fuzz.yaml’  
   Graphical user interface, application

   Description automatically generated  
   - The workflow will run every time a push is made.
4. Tested doing our own commits.

Graphical user interface, text

Description automatically generated

*Forensics*

We integrated forensics by modifying 5 Python methods of our choice.

Detailed Activities:

1. Selected 5 methods for logging based on what methods we would be fuzzing.
2. Added ‘logger.py’ function and modified it to take relevant information from fuzzing.
3. Inserted logging statements into each selected method.
4. Logged at the information level to catch all events up to and including errors from fuzzing.

**Lessons Learned**

By implementing various tools we learned throughout the course and workshops, we were able to learn significantly more about software engineering and software quality assurance tools.

*Git Tools*

* Setting up a GitHub repository and using it within a team environment
* Commands like git push, git pull, git status, etc.
* Git Hooks
* Git flags
* Git actions & debugging with Git actions

*Fuzzing and Forensics*

* How to select methods for fuzzing and logging.
* Different loggings levels and what each should catch.
* Structure of a logging file and how to incorporate it into a larger project.
* Being able to look at a project that already has significant work on it and knowing/understanding what each file does and how it works with others.
* How workflows work

*Other Tools*

* YAML file configuration was a large part of our project. We learned how to add flags and arguments to get our desired outputs.
* We read documentation on Bandit to properly understand how to get our desired output and how to run it within our project.
* Formatting within files like YAML were critical and ensuring our project ran properly
* Working with a team in a GitHub repository and how to commit, push, and pull changes