



# **Project for Software Quality Assurance**

## **COMP6710**

Team Members:

Daniel Burris

Nan Yang

Krystal Lin

For:

Auburn University – Akond Rahman, PhD

December 1<sup>st</sup>, 2023

## Notes and Observations

When we initially pulled the base source file KubSec.zip we were unable to run main.py. Eventually with some debug we figured out all the 3<sup>rd</sup> party packages and libraries we needed to install on our docker image to get it to run, but still we ran into errors when executing it. To get around this quickly we simply dived into the code and added several try/except statements as well as some additional logic statements to get it to run to completion.

All the changes we made can be seen under the following commit:

<https://github.com/Dburris13/WAREAGLE-SQA2023-AUBURN/commit/f26bf94f49bb7f94b396665ac73fd5d2f26df1ff>

The docker image we used to complete this project has been uploaded on docker hub and can be pulled with the following command:

**docker pull mokraton/wareagle\_project\_sqa2023**

Once inside the docker image you can navigate to /WAREAGLE-SQA2023-AUBURN and execute a git pull command to pull the latest source from our GitHub repository.

## Screenshots and Logs

This section of the report contains screenshots and snippets of log files that demonstrates the team members successfully integrated and validated all the requirements set forth by the governing markdown text found here: <https://github.com/paser-group/continuous-secsoft/blob/master/sqa2023/project/project.md>

### GitHub Repository

All source files, commit history, changelogs, output files as well as additional information contained in README's can be found on the team GitHub repository here: <https://github.com/Dburris13/WAREAGLE-SQA2023-AUBURN> which henceforth will be referred to as the "Team's Repository".

### Execution of git hooks

The following artifacts are designed to provide evidence of successful integration and validation of requirement "4a. Create a Git Hook that will run and report all security weaknesses in the project in a CSV file whenever a Python file is changed and committed. (10%)".

As the pre-commit hook is a client-side hook, the contents of it are not stored on the Team's Repository per standard Git practice. Instead the pre-commit hook file is simply uploaded in a separate repository labeled "hooks".

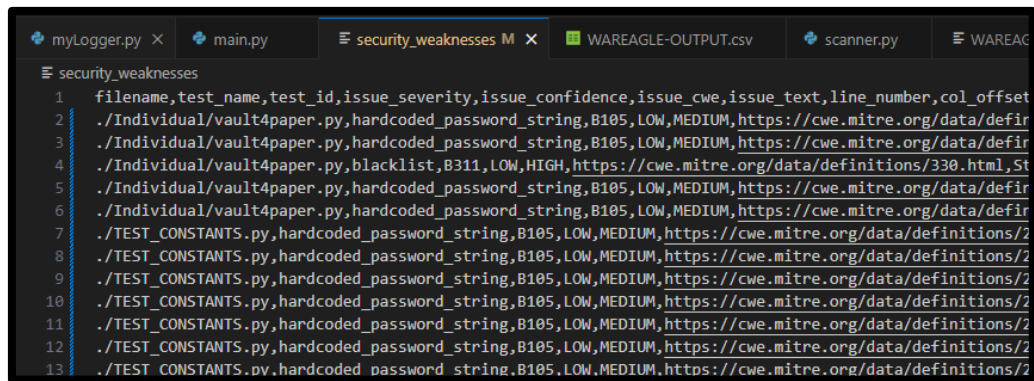
Below is a sample of the git hook running after a test commit. This commit was not pushed to the remote repository.

```

root@f555c7dfe84f:/WAREAGLE-SQA2023-AUBURN# git add -A
root@f555c7dfe84f:/WAREAGLE-SQA2023-AUBURN# git commit -m "testing static analysis"
Running bandit security analysis on all python files
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 3.10.12
[csv] INFO CSV output written to file: security_weaknesses
[main a08d5f1] testing static analysis
1 file changed, 2 deletions(-)
root@f555c7dfe84f:/WAREAGLE-SQA2023-AUBURN#

```

Snippet of the output file created by the pre-commit hook.



```

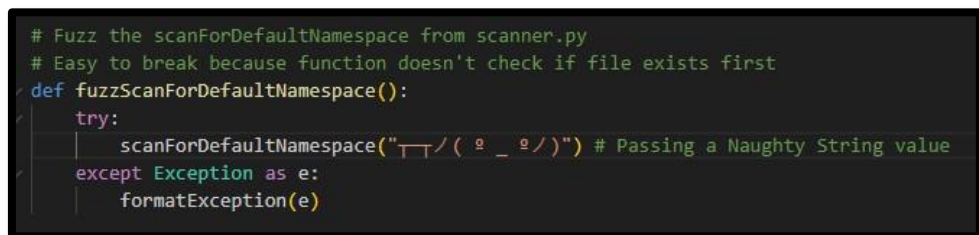
security_weaknesses
1 filename,test_id,issue_severity,issue_confidence,issue_cwe,issue_text,line_number,col_offset
2 ./Individual/vault4paper.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/defin
3 ./Individual/vault4paper.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/defin
4 ./Individual/vault4paper.py,blacklist,B311,LOW,HIGH,https://cwe.mitre.org/data/definitions/330.html,5t
5 ./Individual/vault4paper.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/defin
6 ./Individual/vault4paper.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/defin
7 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2
8 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2
9 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2
10 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2
11 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2
12 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2
13 ./TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/2

```

## Execution of Fuzzing

The following artifacts are designed to provide evidence of successful integration and validation of requirement 4b. *“Create a fuzz.py file that will automatically fuzz 5 Python methods of your choice. Report any bugs you discovered by the fuzz.py file. fuzz.py will be automatically executed from GitHub actions. (20%).”*

Below is an example of one of the “fuzzing” functions we created to provide unexpected input to the function `scanForDefaultNamespace`. As you can see it is within a try statement so the error will not stop continuous execution to the next fuzzing function.



```

# Fuzz the scanForDefaultNamespace from scanner.py
# Easy to break because function doesn't check if file exists first
def fuzzScanForDefaultNamespace():
    try:
        scanForDefaultNamespace("T_T / ( 9 _ 9 / )") # Passing a Naughty String value
    except Exception as e:
        formatException(e)

```

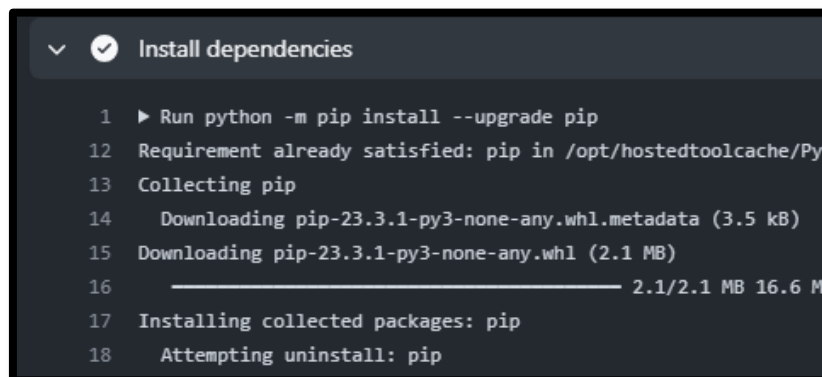
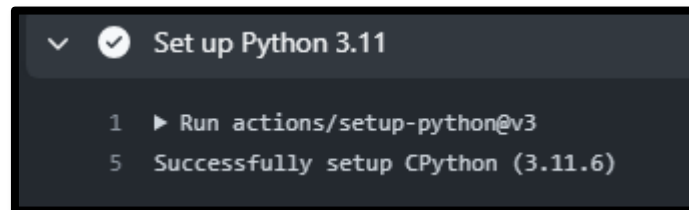
Below is the Github action we wrote to instruct GitHub to run this fuzz.py file on every push to the remote repo. This file can be found on the Team’s Repository in the file `.github/workflows/python-app.yml`.

```
jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Set up Python 3.11
        uses: actions/setup-python@v3
        with:
          python-version: "3.11"
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install ruamel.yaml
          pip install pandas
          pip install numpy
          pip install sarif_om
          pip install jschema_to_python
      - name: Run fuzz.py
        run: |
          python fuzz.py
```

Following are a series of snippets showing the execution of this github workflow and the “All checks have passed” statement.



```

✓ Run fuzz.py

1 ▶ Run python fuzz.py
7 Exception Caught in function checkIfWeirdYAML from parser.py:
8     EXCEPTION: argument of type 'int' is not iterable
9 Exception Caught in function checkIfWeirdYAML from parser.py:
10    EXCEPTION: [Errno 2] No such file or directory: 'T/(\ _ /)'
11 Exception Caught in function checkIfWeirdYAML from parser.py:
12    EXCEPTION: [Errno 9] Bad file descriptor
13 Exception Caught in function checkIfWeirdYAML from parser.py:
14    EXCEPTION: expected str, bytes or os.PathLike object, not int
15 Exception Caught in function checkIfWeirdYAML from parser.py:
16    EXCEPTION: too many values to unpack (expected 2)

```



## Execution of forensics

The following artifacts are designed to provide evidence of successful integration and validation of requirement 4c. *“Integrate forensics by modifying 5 Python methods of your choice. (20%)”*.

The logger we created for our project is a simple modification of the logger created in “Workshop 8 – Software Forensics ...”.

```

# Got rid of the method so I could re-use logger in multiple files with from myLogger import myLogObj
fileName = 'WAREAGLE_sqa2023_project.log'
formatStr = '%(asctime)s %(message)s'
logging.basicConfig(format=formatStr, filename=fileName, level=logging.INFO)
myLogObj = logging.getLogger('sqa2023-logger')

```

The modifications made allowed us to create a simple, global logger object that can be imported with a single statement.

```
from myLogger import myLogObj
```

Below is an example of a logging function:

```
# I want to log the keys read for each YAML file
# The reason this is in scanForOverPrivileges and not in getKeyRecursively is
# because I simply want it logged once, not for every different scan of the same file
myLogObj.info(f"Key list for file {script_path}: {key_lis}")
```

The following snippet is from the WAREAGLE\_sqa2023\_project.log logging output file. This file can be re-generated by simply running the following command **python3 main.py**.

```
WAREAGLE_sqa2023_project.log
1 2023-11-27 04:53:12,846 Start of Log File
2 2023-11-27 04:53:12,846 Directory to be scanned: /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/
3 2023-11-27 04:53:12,846 Results file located here: /WAREAGLE-SQA2023-AUBURN/WAREAGLE-OUTPUT.csv
4 2023-11-27 04:53:13,015 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/gcr.deploymer
5 2023-11-27 04:53:14,587 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/fp.no.reso4.y
6 2023-11-27 04:53:16,994 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/fp.secu.conte
7 2023-11-27 04:53:18,118 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/fp.no.reso3.y
8 2023-11-27 04:53:22,211 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/host-net.tp.y
9 2023-11-27 04:53:26,118 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/multi.doc.yam
10 2023-11-27 04:53:30,710 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/present.roll.
11 2023-11-27 04:53:33,140 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/tp.nsp.dflt.y
12 2023-11-27 04:53:34,467 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/nginx.deployw
13 2023-11-27 04:53:37,723 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/fp.nspace2.ya
14 2023-11-27 04:53:39,608 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/k8s.doc.netwc
15 2023-11-27 04:53:42,605 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/empty.yml: [(
16 2023-11-27 04:53:48,413 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/fp.no.reso9.y
17 2023-11-27 04:53:50,885 Invalid K8S YAML File /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/githubooks.
18 2023-11-27 04:53:51,836 Key list for file /WAREAGLE-SQA2023-AUBURN/TEST_ARTIFACTS/present.deplc
```

## Lessons Learned

Daniel Burris:

The biggest lesson I learned while working on this project was working with docker. Prior to this class I had knowledge of docker and had done basic things with it, but this project really enabled me to look into some of the more powerful features of docker. I was running images then commit containers back into images, I created a docker hub account to store my docker image. I was able to provision my docker image with all utilities needed to execute SLI-KUBE then share that image with my teammates. It also taught me more about GitHub Actions. I have worked with automation software before like Jenkins, but this was my first time working with GitHub actions and it was good to see some of the features that are often for free from GitHub. This project also just helped strengthen my python skills when I had to read through the source code to figure out the best places to start logging output.

Nan Yang:

About this project include the workshop knowledge we did in the past, and implementing into this project, the lessons I learned is when we try to run the project, the schedule has to be made, first we need to set up a Github Repository, and try to run the git hook, after that run bandit tools get the security weakness report, so all those workshop we did, actually it connect to each other somehow, Conclusion is applying the software quality assurance along with the software development.