Brayden Simpson
Ian Larson
Matthew Ippolito
Melvin Moreno
COMP6710
Dr. Akond Rahman
26 April 2023

BIMM-SQA2023-Project Report

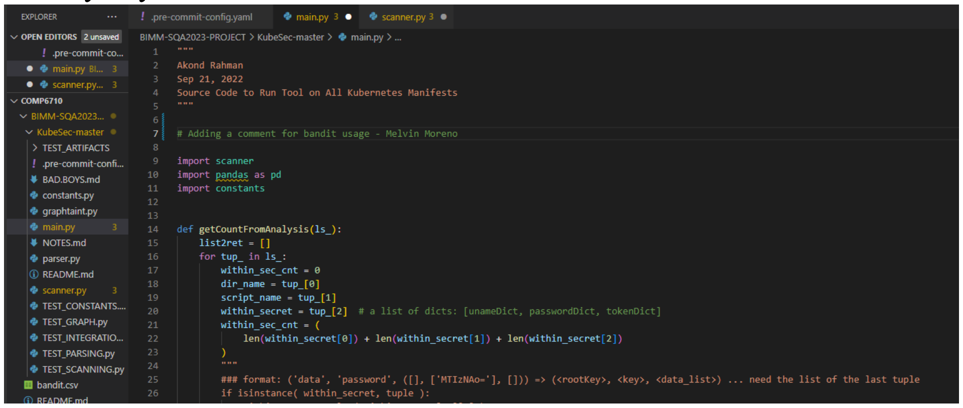
Activities:

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.

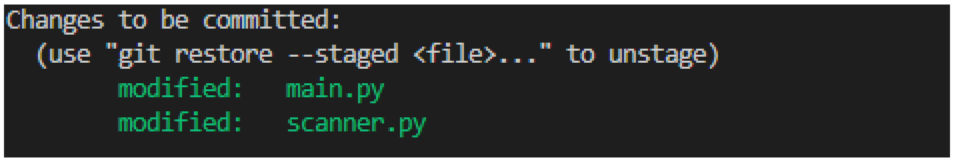
Detailed Instructions:

1. Modify any file



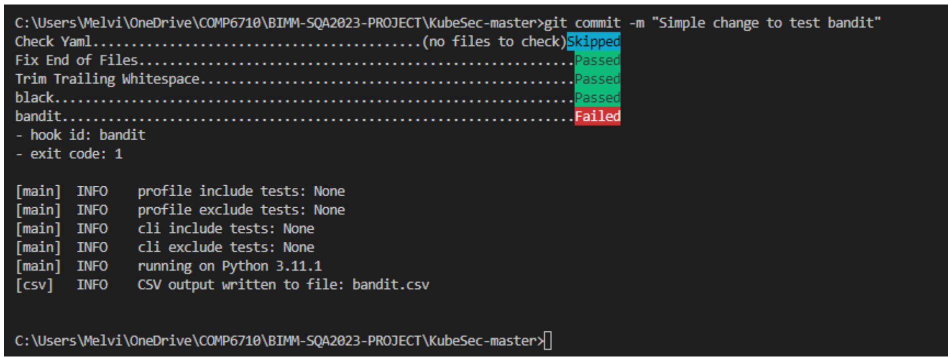
```
"""
Akond Rahman
Sep 21, 2022
Source Code to Run Tool on All Kubernetes Manifests
"""

# Adding a comment for bandit usage - Melvin Moreno

import scanner
import pandas as pd
import constants


def getCountFromAnalysis(ls_):
    list2ret = []
    for tup_ in ls_:
        within_sec_cnt = 0
        dir_name = tup_[0]
        script_name = tup_[1]
        within_secret = tup_[2]  # a list of dicts: [unameDict, passwordDict, tokenDict]
        within_sec_cnt = (
            len(within_secret[0]) + len(within_secret[1]) + len(within_secret[2])
        )
        """
        ### format: ('data', 'password', ([], ['MTIzNAo='], [])) => (<rootKey>, <key>, <data_list>) ... need the list of the last tuple
        if isinstance( within_secret, tuple ):
```

2. Git add the files



```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   main.py
        modified:   scanner.py
```

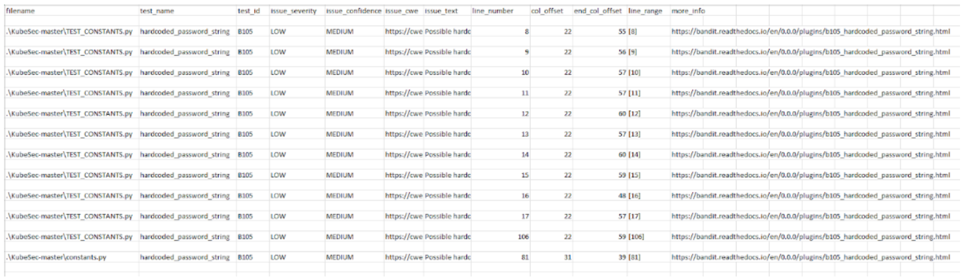
3. Git commit



```
C:\Users\Melvi\OneDrive\COMP6710\BIMM-SQA2023-PROJECT\KubeSec-master>git commit -m "Simple change to test bandit"
Check Yaml.........................................(no files to check)Skipped
Fix End of Files.......................................................Passed
Trim Trailing Whitespace...............................................Passed
black..................................................................Passed
bandit.................................................................Failed
- hook id: bandit
- exit code: 1

[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 3.11.1
[csv]   INFO    CSV output written to file: bandit.csv


C:\Users\Melvi\OneDrive\COMP6710\BIMM-SQA2023-PROJECT\KubeSec-master>
```

- 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.

4. Get results



filename	test_name	test_id	issue_severity	issue_confidence	issue_cwe	issue_text	line_number	col_offset	end_col_offset	line_range	more_info
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	8	22	55	[8]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	9	22	56	[9]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	10	22	57	[10]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	11	22	57	[11]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	12	22	60	[12]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	13	22	57	[13]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	14	22	60	[14]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	15	22	59	[15]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	16	22	48	[16]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	17	22	57	[17]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	106	22	59	[106]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html
.\KubeSec-master\constants.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe	Possible hardc	81	31	39	[81]	https://bandit.readthedocs.io/en/0.0.0/plugins/b105_hardcoded_password_string.html

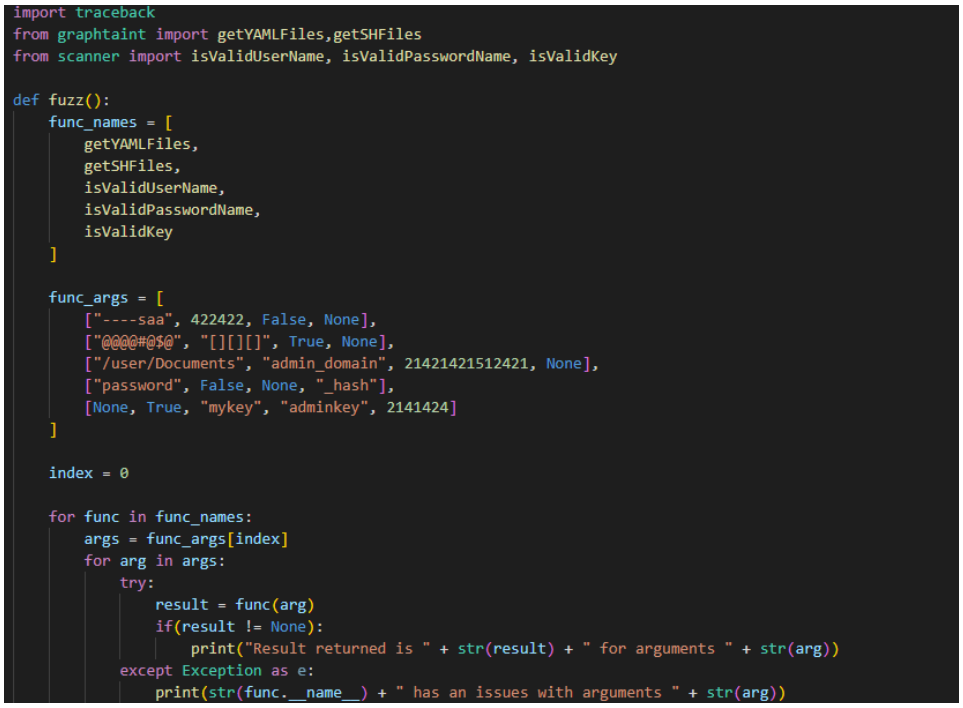
Fuzzing

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:

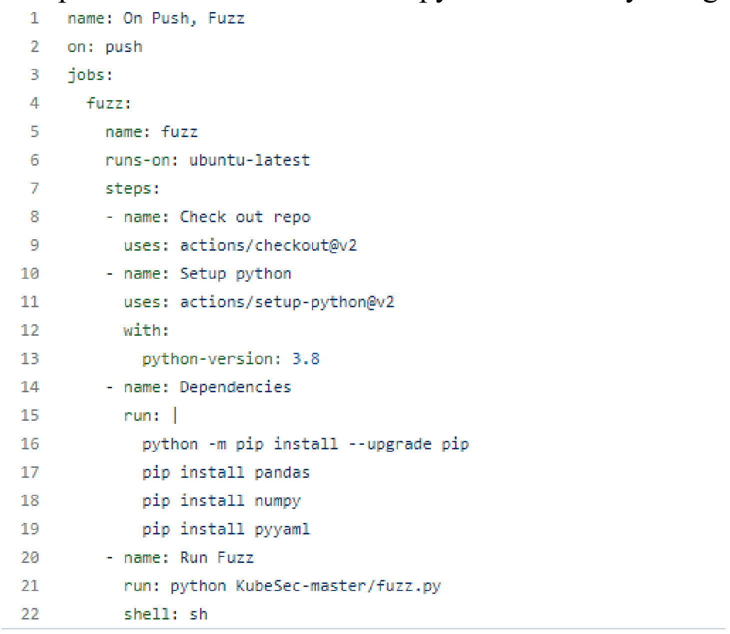
1. We selected the 5 methods to perform fuzzing on:
   - Graphtaint.py: getYAMLFiles
   - Graphtain.py: getSHFiles
   - Scanner.py: isValidUsername
   - Scanner.py: isValidPassword
   - Scanner.py: isValidKey

2. Created the ‘fuzz.py’ file to send random values to these methods to see how they would react.



```python
import traceback
from graphtaint import getYAMLFiles,getSHFiles
from scanner import isValidUserName, isValidPasswordName, isValidKey

def fuzz():
    func_names = [
        getYAMLFiles,
        getSHFiles,
        isValidUserName,
        isValidPasswordName,
        isValidKey
    ]

    func_args = [
        ["----saa", 422422, False, None],
        ["@@@@#@$@", "[][][]", True, None],
        ["/user/Documents", "admin_domain", 21421421512421, None],
        ["password", False, None, "_hash"],
        [None, True, "mykey", "adminkey", 2141424]
    ]

    index = 0

    for func in func_names:
        args = func_args[index]
        for arg in args:
            try:
                result = func(arg)
                if(result != None):
                    print("Result returned is " + str(result) + " for arguments " + str(arg))
            except Exception as e:
                print(str(func.__name__) + " has an issues with arguments " + str(arg))
```

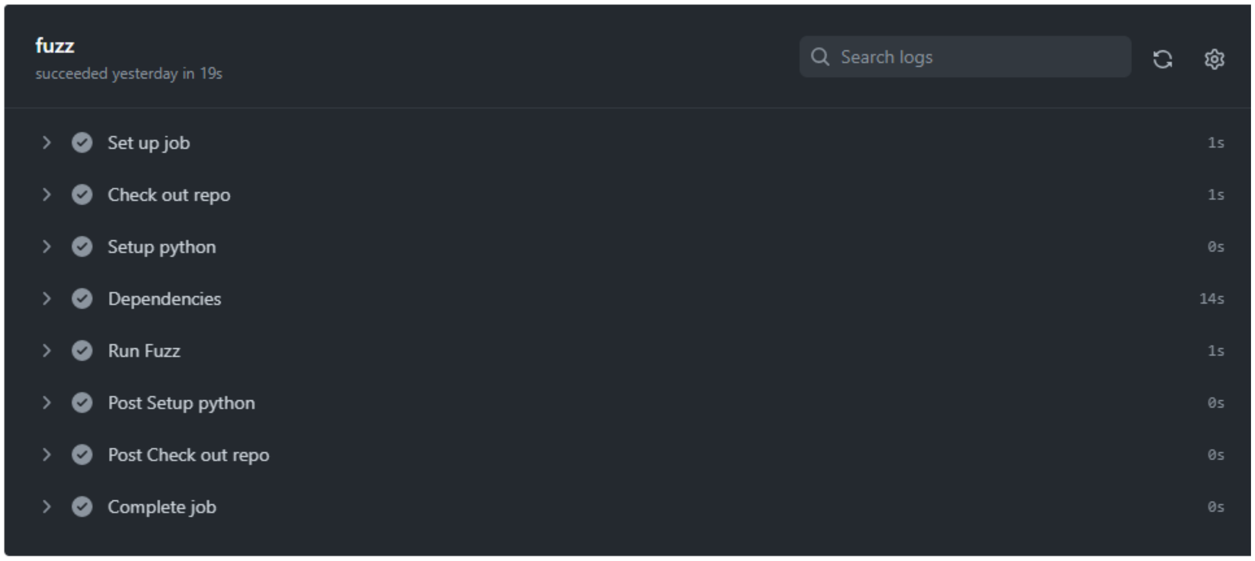
3. Setup the workflow to run ‘fuzz.py’ automatically using ‘fuzz.yaml’



```yaml
name: On Push, Fuzz
on: push
jobs:
  fuzz:
    name: fuzz
    runs-on: ubuntu-latest
    steps:
    - name: Check out repo
      uses: actions/checkout@v2
    - name: Setup python
      uses: actions/setup-python@v2
      with:
        python-version: 3.8
    - name: Dependencies
      run: |
        python -m pip install --upgrade pip
        pip install pandas
        pip install numpy
        pip install pyyaml
    - name: Run Fuzz
      run: python KubeSec-master/fuzz.py
      shell: sh
```

- The workflow will run every time a push is made.

4. Tested doing our own commits.



```
fuzz
succeeded yesterday in 19s

> Set up job              1s
> Check out repo          1s
> Setup python            0s
> Dependencies           14s
> Run Fuzz                1s
> Post Setup python       0s
> Post Check out repo     0s
> Complete job            0s
```

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