Software Quality Assurance COMP 6710
Project Report
QualityControllers – SQA2023 – AUBURN

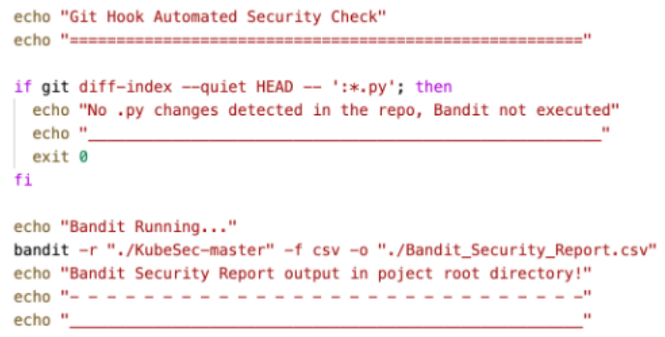
Team 15: Christopher Casey, Nihitha Reddy Seelam, Pooja Chilukuru, Yinbo Chen

Contents:

1)Security Weakness:

For identifying common security weaknesses in Python code, Bandit is run on the repo to help identify and locate any unknown weakness. Due to the importance and regularity which a team may run a tool such as Bandit, it has been included in a pre-commit git hook. A pre-commit hook is a script that runs automatically before an actual commit takes place. It allows teams to incorporate automatic actions and checks on repository code every time they commit any changes. Including Bandit in a pre-commit hook is an easy and efficient way to ensure that all code that enters the repository meets a certain security standard and that all collaborators are aware of any weaknesses. Further, it helps a team identify issues early and often because of how it was included within a team's workflow.

The Code:



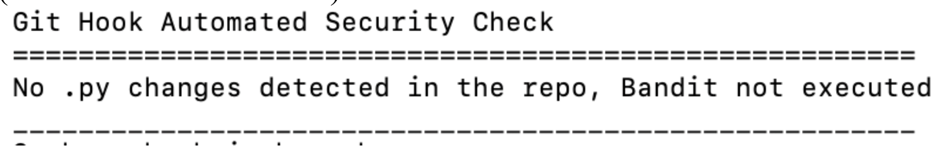
```
echo "Git Hook Automated Security Check"
echo "======================================================"

if git diff-index --quiet HEAD -- ':*.py'; then
  echo "No .py changes detected in the repo, Bandit not executed"
  echo "______________________________________________________"
  exit 0
fi

echo "Bandit Running..."
bandit -r "./KubeSec-master" -f csv -o "./Bandit_Security_Report.csv"
echo "Bandit Security Report output in poject root directory!"
echo "- - - - - - - - - - - - - - - - - - - - - - - - - - - -"
echo "______________________________________________________"
```

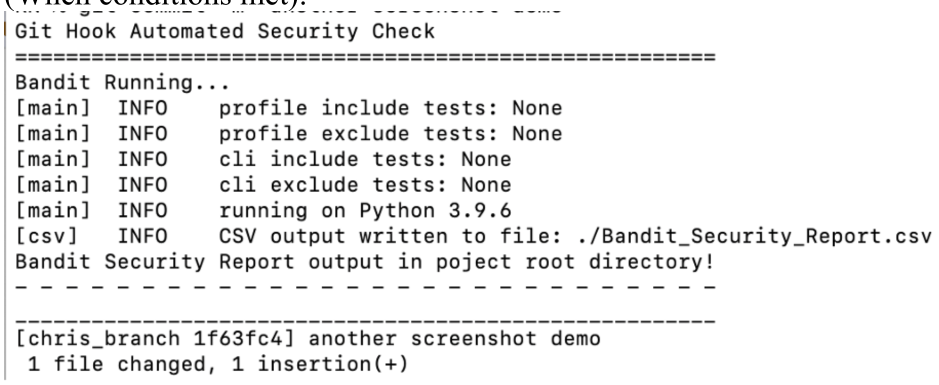
Command Line Output:

(When conditions not met):



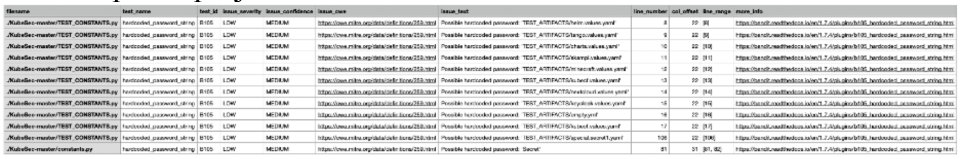
```
Git Hook Automated Security Check
=======================================================
No .py changes detected in the repo, Bandit not executed
_______________________________________________________
```

(When conditions met):



```
Git Hook Automated Security Check
========================================================
Bandit Running...
[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 3.9.6
[csv]   INFO    CSV output written to file: ./Bandit_Security_Report.csv
Bandit Security Report output in poject root directory!
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
________________________________________________________
[chris_branch 1f63fc4] another screenshot demo
 1 file changed, 1 insertion(+)
```

CSV output in project root:



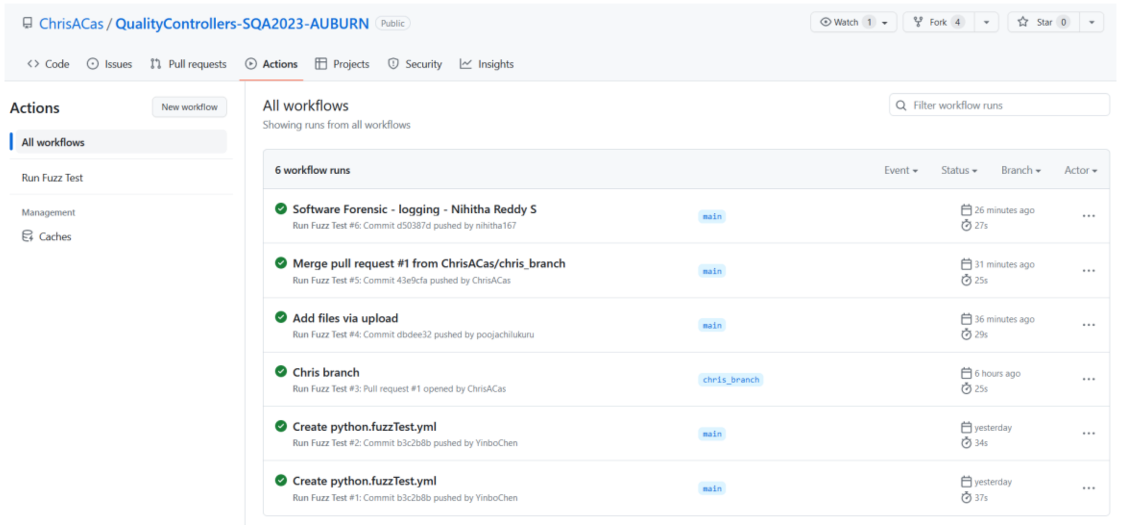
filename	test_name	test_id	issue_severity	issue_confidence	issue_cwe	issue_text	line_number	col_offset	line_range	more_info
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/helm.values.yaml'	8	22	[8]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/tango.values.yaml'	9	22	[9]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/charts.values.yaml'	10	22	[10]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/skampi.values.yaml'	11	22	[11]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/minecraft.values.yaml'	12	22	[12]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/u.bedf.values.yaml'	13	22	[13]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/nextcloud.values.yaml'	14	22	[14]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/keycloak.values.yaml'	15	22	[15]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/empty.yml'	16	22	[16]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/u.bedf.values.yaml'	17	22	[17]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/TEST_CONSTANTS.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'TEST_ARTIFACTS/special.secret1.yaml'	106	22	[106]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm
./KubeSec-master/constants.py	hardcoded_password_string	B105	LOW	MEDIUM	https://cwe.mitre.org/data/definitions/259.html	Possible hardcoded password: 'Secret'	81	31	[81, 82]	https://bandit.readthedocs.io/en/1.7.4/plugins/b105_hardcoded_password_string.htm

2)Fuzzing:

Fuzzing is a negative s/w testing method that feeds malformed input to a program, device/system for finding security-related defects or any critical flaws leading to denial of service, degradation of service or other undesired behavior.
Programs and frameworks used to create fuzz tests or perform fuzzing are called **fuzzers**.
Target Functions:
For performing the Fuzzy tests. We chose five functions from the component 'scanner.py'. They are 'isValidUserName()', 'isValidPasswordName()', 'isValidKey()', ' checkIfValidSecret()', 'checkIfValidKeyValue()'. Each function has been confined and only specified input can go through the execution. All functions' input values are strings. The input and output values are shown below:

1. The 'isValidUserName()' takes a string as input, then checks the input value with constant values to see if it is getting any matching (constant values be treated as a substring of input string). After checking the input data format, if it's a correct string type then continually execute to match strings. The return value is either 'True' or 'False'.
2. The 'IsValidpasswordName()' function's execution process is like the first one. The only difference is that using different constant comparison lists.
3. The 'IsValidKey()' is the same as the previous two functions.
4. The 'CheckIfValidSecret()' function's input is a string too. It transformed input letters to lowercase and removed spaces if there were any. Then do matching and return 'True' or 'False'.
5. The 'CheckIfValidKeyValue()' function's logic is similar to the first one above.

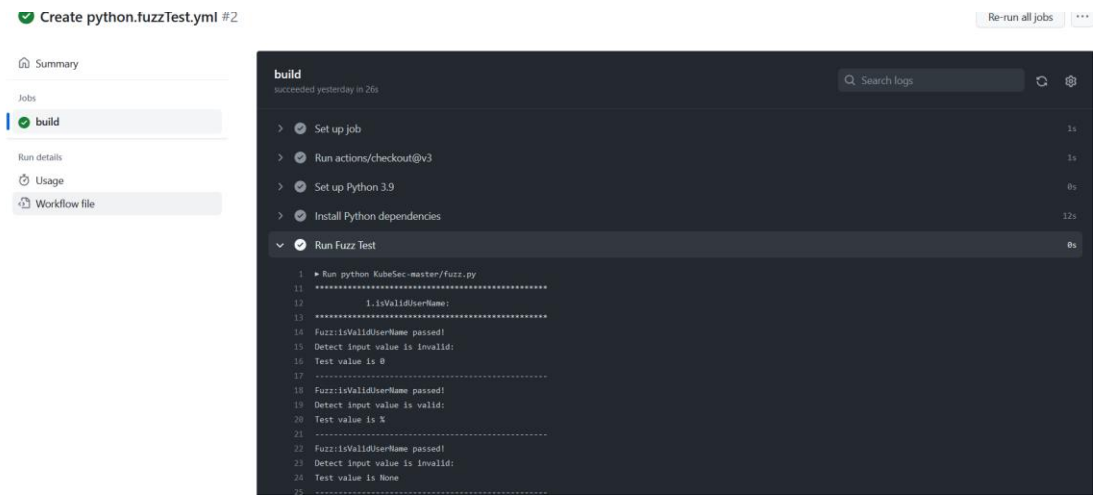
Fuzz Testing Input:
The input data structure is a list. The possible input value types are 'int', 'float', 'None', 'strings (without constant substring)', 'string (with constant substring)', and symbols.
Result:
All functions returned correct results. We didn't find any bugs or defects. The screenshots are shown below.



All workflows
Showing runs from all workflows

6 workflow runs

Software Forensic - logging - Nihitha Reddy S Run Fuzz Test #6: Commit d50387d pushed by nihitha167	main	26 minutes ago 27s
Merge pull request #1 from ChrisACas/chris_branch Run Fuzz Test #5: Commit 43e9cfa pushed by ChrisACas	main	31 minutes ago 25s
Add files via upload Run Fuzz Test #4: Commit dbdee32 pushed by poojachilukuru	main	36 minutes ago 29s
Chris branch Run Fuzz Test #3: Pull request #1 opened by ChrisACas	chris_branch	6 hours ago 25s
Create python.fuzzTest.yml Run Fuzz Test #2: Commit b3c2b8b pushed by YinboChen	main	yesterday 34s
Create python.fuzzTest.yml Run Fuzz Test #1: Commit b3c2b8b pushed by YinboChen	main	yesterday 37s



build
succeeded yesterday in 26s

```
Set up job                              1s
Run actions/checkout@v3                 1s
Set up Python 3.9                       0s
Install Python dependencies            12s
Run Fuzz Test                           0s
```

```
▸ Run python KubeSec-master/fuzz.py
**************************************************
          1.isValidUserName:
**************************************************
Fuzz:isValidUserName passed!
Detect input value is invalid:
Test value is 0
--------------------------------------------------
Fuzz:isValidUserName passed!
Detect input value is valid:
Test value is %
--------------------------------------------------
Fuzz:isValidUserName passed!
Detect input value is invalid:
Test value is None
--------------------------------------------------
```

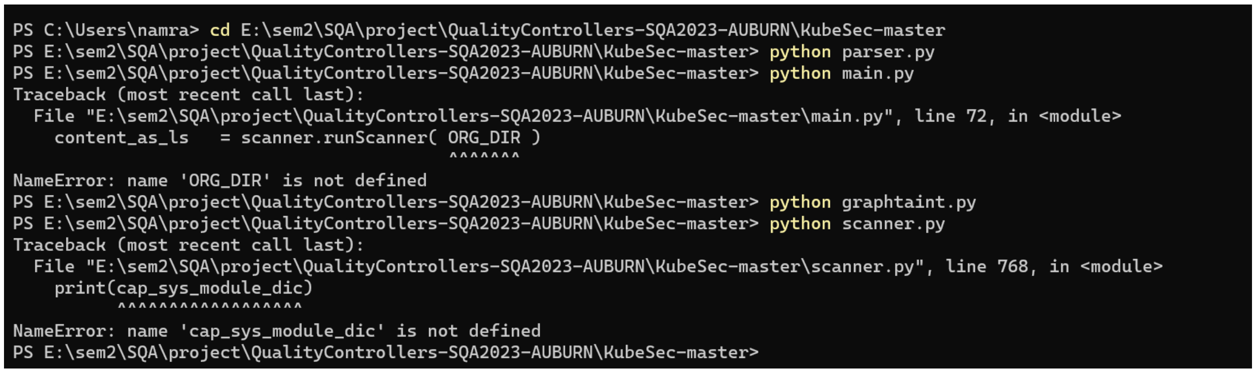
3)Forensics:

Forensics is a way to integrate logs into software. Which is useful when unwanted event occurs it is easy to detect, and trouble shoot.
Mainly issues like access issues, change issues, exceptions and so on are detected.
Software forensics should be implemented from the initial stage of developing an application/Software feature.

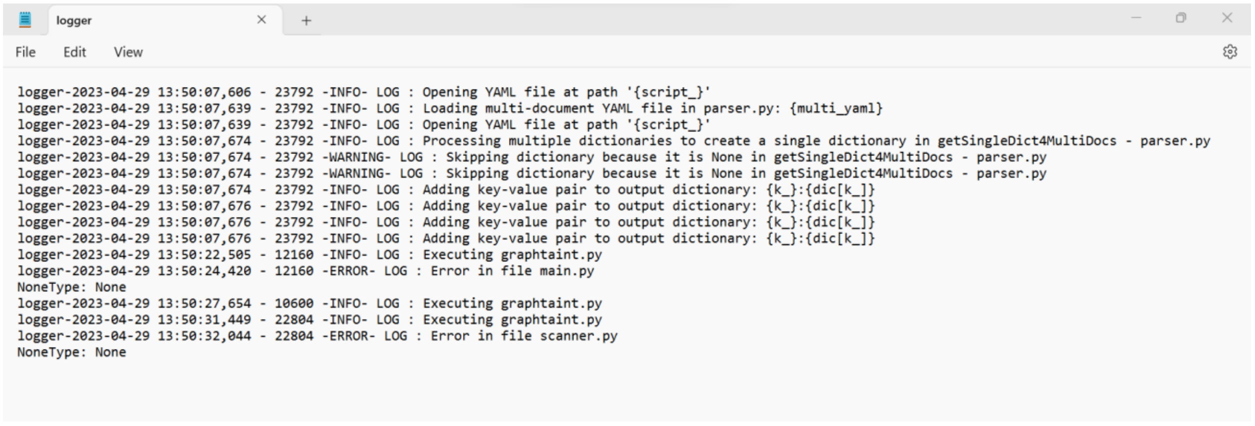
As part of our project, logging_forensics.py has the giveMeLoggingObject()function that provides a way creation of logging object using logging module. Function sets up a basic configuration for the logging object, including the format of log message and the file to which the logs will be written.
All the logs got stored in the logger.log.

logging is done in more than 5 methods in each s/w. **Logs are implemented** in parser.py, main.py, scanner.py and graphtaint.py.

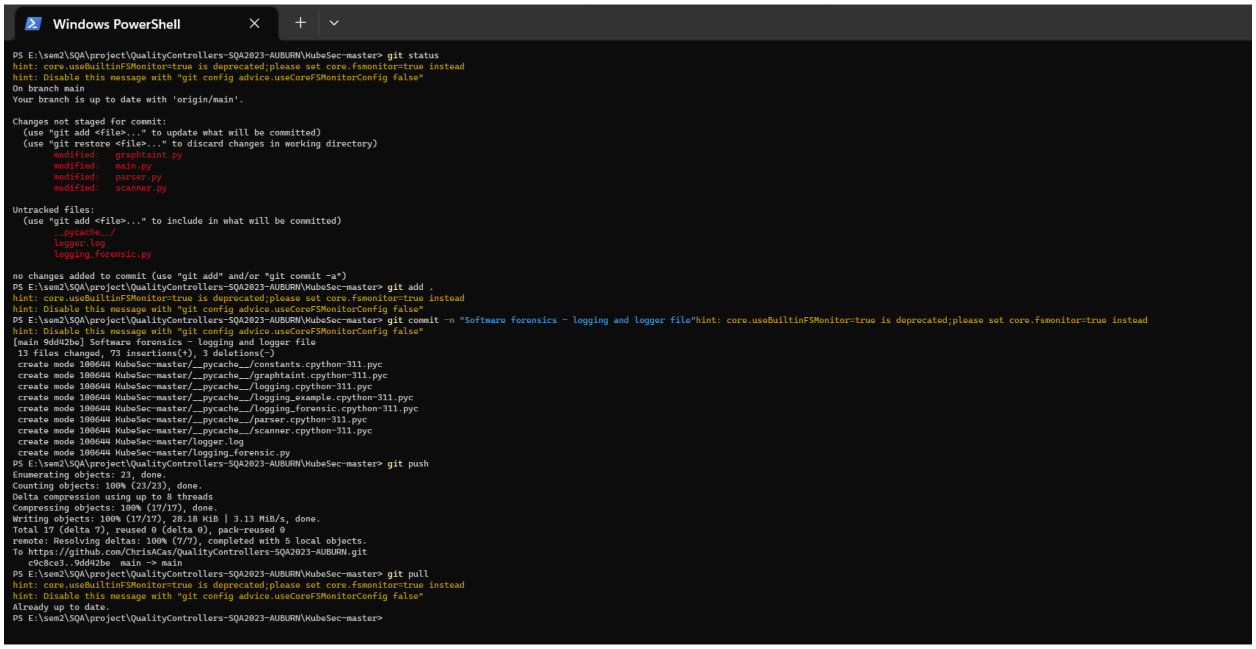
Following are the snapshots of the implementation:



```
PS C:\Users\namra> cd E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> python parser.py
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> python main.py
Traceback (most recent call last):
  File "E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master\main.py", line 72, in <module>
    content_as_ls   = scanner.runScanner( ORG_DIR )
                                          ^^^^^^^
NameError: name 'ORG_DIR' is not defined
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> python graphtaint.py
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> python scanner.py
Traceback (most recent call last):
  File "E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master\scanner.py", line 768, in <module>
    print(cap_sys_module_dic)
          ^^^^^^^^^^^^^^^^^^
NameError: name 'cap_sys_module_dic' is not defined
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master>
```



```
logger-2023-04-29 13:50:07,606 - 23792 -INFO- LOG : Opening YAML file at path '{script_}'
logger-2023-04-29 13:50:07,639 - 23792 -INFO- LOG : Loading multi-document YAML file in parser.py: {multi_yaml}
logger-2023-04-29 13:50:07,639 - 23792 -INFO- LOG : Opening YAML file at path '{script_}'
logger-2023-04-29 13:50:07,674 - 23792 -INFO- LOG : Processing multiple dictionaries to create a single dictionary in getSingleDict4MultiDocs - parser.py
logger-2023-04-29 13:50:07,674 - 23792 -WARNING- LOG : Skipping dictionary because it is None in getSingleDict4MultiDocs - parser.py
logger-2023-04-29 13:50:07,674 - 23792 -WARNING- LOG : Skipping dictionary because it is None in getSingleDict4MultiDocs - parser.py
logger-2023-04-29 13:50:07,674 - 23792 -INFO- LOG : Adding key-value pair to output dictionary: {k_}:{dic[k_]}
logger-2023-04-29 13:50:07,676 - 23792 -INFO- LOG : Adding key-value pair to output dictionary: {k_}:{dic[k_]}
logger-2023-04-29 13:50:07,676 - 23792 -INFO- LOG : Adding key-value pair to output dictionary: {k_}:{dic[k_]}
logger-2023-04-29 13:50:07,676 - 23792 -INFO- LOG : Adding key-value pair to output dictionary: {k_}:{dic[k_]}
logger-2023-04-29 13:50:22,505 - 12160 -INFO- LOG : Executing graphtaint.py
logger-2023-04-29 13:50:24,420 - 12160 -ERROR- LOG : Error in file main.py
NoneType: None
logger-2023-04-29 13:50:27,654 - 10600 -INFO- LOG : Executing graphtaint.py
logger-2023-04-29 13:50:31,449 - 22804 -INFO- LOG : Executing graphtaint.py
logger-2023-04-29 13:50:32,044 - 22804 -ERROR- LOG : Error in file scanner.py
NoneType: None
```



```
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> git status
hint: core.useBuiltinFSMonitor=true is deprecated;please set core.fsmonitor=true instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   graphtaint.py
        modified:   main.py
        modified:   parser.py
        modified:   scanner.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        __pycache__/
        logger.log
        logging_forensic.py

no changes added to commit (use "git add" and/or "git commit -a")
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> git add .
hint: core.useBuiltinFSMonitor=true is deprecated;please set core.fsmonitor=true instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> git commit -m "Software forensics - logging and logger file"hint: core.useBuiltinFSMonitor=true is deprecated;please set core.fsmonitor=true instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[main 9dd42be] Software forensics - logging and logger file
 13 files changed, 73 insertions(+), 3 deletions(-)
 create mode 100644 KubeSec-master/__pycache__/constants.cpython-311.pyc
 create mode 100644 KubeSec-master/__pycache__/graphtaint.cpython-311.pyc
 create mode 100644 KubeSec-master/__pycache__/logging.cpython-311.pyc
 create mode 100644 KubeSec-master/__pycache__/logging_example.cpython-311.pyc
 create mode 100644 KubeSec-master/__pycache__/logging_forensic.cpython-311.pyc
 create mode 100644 KubeSec-master/__pycache__/parser.cpython-311.pyc
 create mode 100644 KubeSec-master/__pycache__/scanner.cpython-311.pyc
 create mode 100644 KubeSec-master/logger.log
 create mode 100644 KubeSec-master/logging_forensic.py
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> git push
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (17/17), 28.18 KiB | 3.13 MiB/s, done.
Total 17 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), completed with 5 local objects.
To https://github.com/ChrisACas/QualityControllers-SQA2023-AUBURN.git
   c9c8ce3..9dd42be  main -> main
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master> git pull
hint: core.useBuiltinFSMonitor=true is deprecated;please set core.fsmonitor=true instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
Already up to date.
PS E:\sem2\SQA\project\QualityControllers-SQA2023-AUBURN\KubeSec-master>
```

4) Lessons learnt:

With respect to Security Weakness:

Using a pre-commit hook comes with a couple of important lessons of note. Such as, if you perform a commit within the pre-commit hook without the proper flags [--no-verify], you may crash the repository by sending it into an infinite loop. It might be important to include a *git commit –no-verify* in your pre-commit hook to include any newly generated report files in you next *git push.* Another lesson would be the ability to set up conditional blocks that check the changes committed to a repo for certain constraints to be met. This might be useful to prevent the developer from becoming inundated with unnecessary reports that provide no useful data. This allows developers to focus on the checks and reports that pertain only to the code changes in specific commits. Lastly, if a collaborator wants to share their repo automated checks and scripts, they can include named directories containing these scripts within the repo itself. A collaborator can then set their own git to use these scripts using a git config command such as 'git config core.hooksPath .githooks'

With respect to Fuzzing:

For performing the fuzz test. The main lesson we learned was to maintain the coverage of test cases and make sure that the functions component of the fuzzer itself should be secured and included in the handler of expectations.
For the github workflows. First, the script file should have the correct execution path as well as the updated checkout version. For testing purposes, it should also contain the testing section before doing 'push' and 'pull' requests. Second, when building an action script should consider the target file's dependency with the valid version controlling.

With respect to Software Forensics:

While performing logging, it's essential to capture enough data to reconstruct the events accurately such as capturing the date and time of the event, the user details who triggered the event, and any event relevant details.
Usage of different log levels while dealing with different events to capture the data.
Centralized logging can make it easier to manage logs from multiple systems and applications.