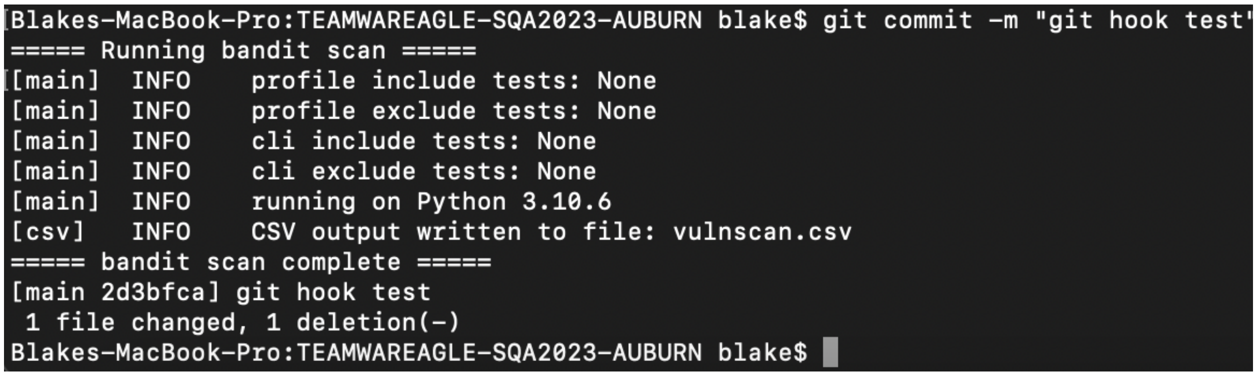
Team War Eagle Project Report

Carson Herren
Nathan Harrison
Tullis Nelson

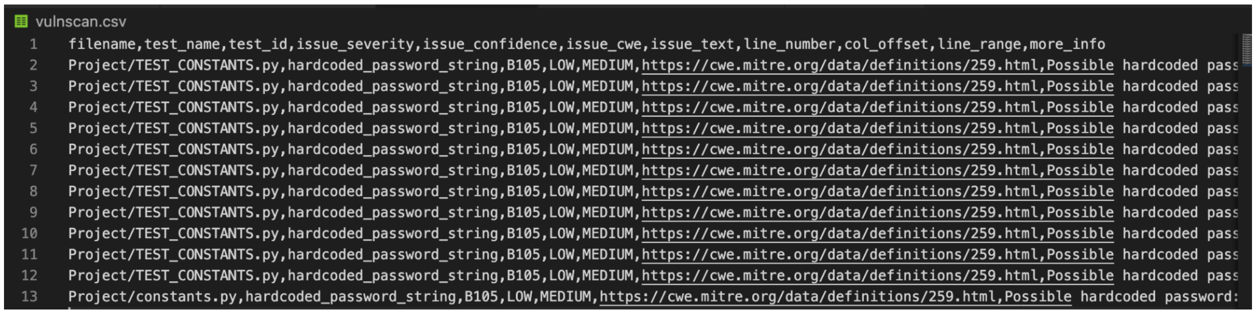
Github Repo

Git Hook - Carson Herren

Using workshops 2 and 4 I implemented a git hook to scan all python files for vulnerabilities. I did this by editing the pre-commit file to run a bandit command that scans all python files and outputs the result in csv format after a commit has occurred. This bandit command uses the -f, -r, and -o flags to specify the output type, directory to be scanned, and output file name. This part of the project allowed me to use what I learned previously from the workshops to implement a git hook using bandit instead of cppcheck. An example output from the commit is as follows:



```
[Blakes-MacBook-Pro:TEAMWAREAGLE-SQA2023-AUBURN blake$ git commit -m "git hook test"
===== Running bandit scan =====
[[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 3.10.6
[csv]   INFO    CSV output written to file: vulnscan.csv
===== bandit scan complete =====
[main 2d3bfca] git hook test
 1 file changed, 1 deletion(-)
Blakes-MacBook-Pro:TEAMWAREAGLE-SQA2023-AUBURN blake$
```

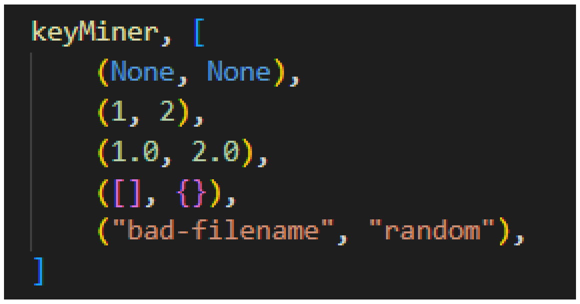


vulnscan.csv

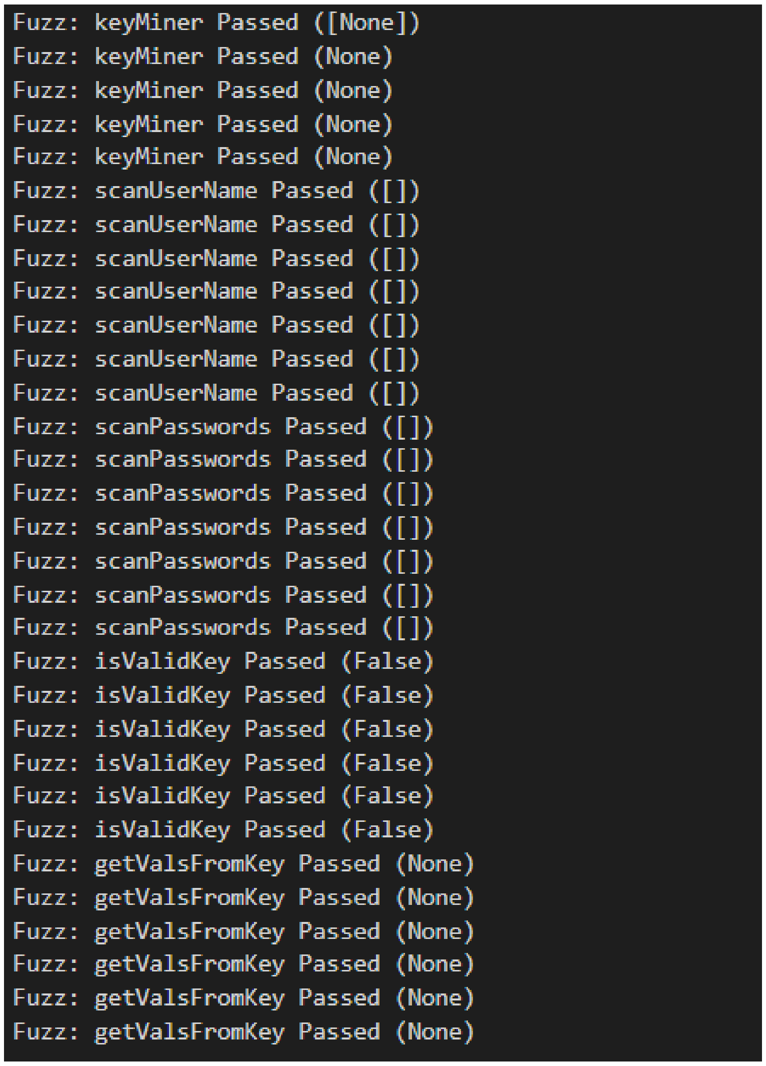
```
filename,test_name,test_id,issue_severity,issue_confidence,issue_cwe,issue_text,line_number,col_offset,line_range,more_info
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/TEST_CONSTANTS.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded pass
Project/constants.py,hardcoded_password_string,B105,LOW,MEDIUM,https://cwe.mitre.org/data/definitions/259.html,Possible hardcoded password:
```

Fuzzing - Nathan Harrison

Using the knowledge I gained from the fuzzing workshop I was able to implement a fuzz.py file to automatically fuzz 5 python methods.   We fed random, incorrect arguments into each python method. For each of the methods we tested none of them failed as shown below.



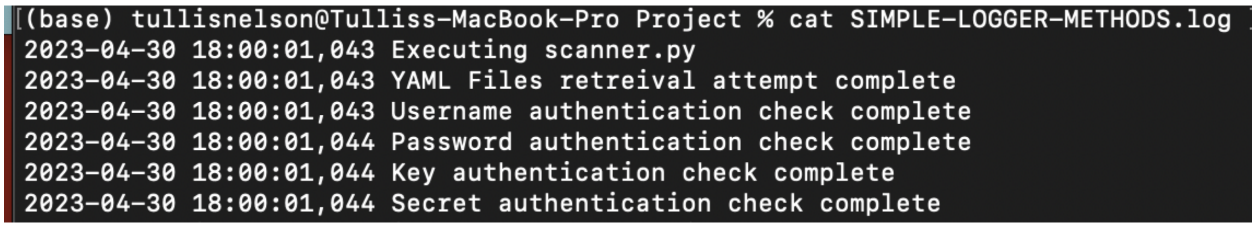
```
keyMiner, [
    (None, None),
    (1, 2),
    (1.0, 2.0),
    ([], {}),
    ("bad-filename", "random"),
]
```



```
Fuzz: keyMiner Passed ([None])
Fuzz: keyMiner Passed (None)
Fuzz: keyMiner Passed (None)
Fuzz: keyMiner Passed (None)
Fuzz: keyMiner Passed (None)
Fuzz: scanUserName Passed ([])
Fuzz: scanUserName Passed ([])
Fuzz: scanUserName Passed ([])
Fuzz: scanUserName Passed ([])
Fuzz: scanUserName Passed ([])
Fuzz: scanUserName Passed ([])
Fuzz: scanUserName Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: scanPasswords Passed ([])
Fuzz: isValidKey Passed (False)
Fuzz: isValidKey Passed (False)
Fuzz: isValidKey Passed (False)
Fuzz: isValidKey Passed (False)
Fuzz: isValidKey Passed (False)
Fuzz: isValidKey Passed (False)
Fuzz: getValsFromKey Passed (None)
Fuzz: getValsFromKey Passed (None)
Fuzz: getValsFromKey Passed (None)
Fuzz: getValsFromKey Passed (None)
Fuzz: getValsFromKey Passed (None)
Fuzz: getValsFromKey Passed (None)
```

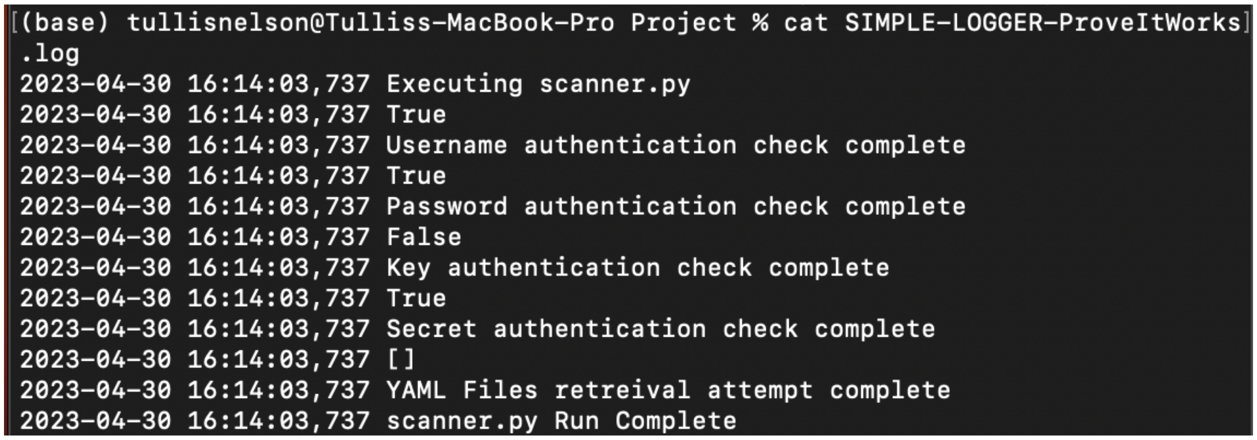
Forensics - Tullis Nelson

We used our logger.py file to import a function that will save a log as functions are run. They are saved with a time and date stamp so we know when things are happening.



```
(base) tullisnelson@Tulliss-MacBook-Pro Project % cat SIMPLE-LOGGER-METHODS.log
2023-04-30 18:00:01,043 Executing scanner.py
2023-04-30 18:00:01,043 YAML Files retreival attempt complete
2023-04-30 18:00:01,043 Username authentication check complete
2023-04-30 18:00:01,044 Password authentication check complete
2023-04-30 18:00:01,044 Key authentication check complete
2023-04-30 18:00:01,044 Secret authentication check complete
```

The above image shows what the log will look like. We can see that each thing is saving a sentence telling them as they happen and that each one is happening. SIMPLE_LOGGER.log is what we would want our actual log to look like.



```
(base) tullisnelson@Tulliss-MacBook-Pro Project % cat SIMPLE-LOGGER-ProveItWorks
.log
2023-04-30 16:14:03,737 Executing scanner.py
2023-04-30 16:14:03,737 True
2023-04-30 16:14:03,737 Username authentication check complete
2023-04-30 16:14:03,737 True
2023-04-30 16:14:03,737 Password authentication check complete
2023-04-30 16:14:03,737 False
2023-04-30 16:14:03,737 Key authentication check complete
2023-04-30 16:14:03,737 True
2023-04-30 16:14:03,737 Secret authentication check complete
2023-04-30 16:14:03,737 []
2023-04-30 16:14:03,737 YAML Files retreival attempt complete
2023-04-30 16:14:03,737 scanner.py Run Complete
```

The above image is not how we would actually want to log things as we dont want to know the inputs or outputs of them but it does show that our system is validating that these things are being run and when as they receive the results. SIMPLE_LOGGER_ProveItWorks is not a proper log file however as we should not save the inputs or outputs, only that the functions are occurring and when.