INTRODUCTION

In software development, continuous integration (CI) is the practice of frequently integrating code changes into a shared repository. This helps to catch integration issues early and enables teams to deliver working software more rapidly. As part of the CI process, it's important to also perform code analysis to catch potential security vulnerabilities and maintain code quality. Git hooks are scripts that run automatically before or after certain Git events, such as a commit. One popular code analysis tool for Python projects is Bandit, which detects common security issues in Python code. In this project, we implemented a pre-commit Git hook that runs Bandit on any changed Python files and generates a report of any security weaknesses found.

BACKGROUND

Python is a widely used programming language that is known for its simplicity, readability, and versatility. However, like any programming language, it is prone to vulnerabilities and errors that can lead to safety and security issues. Static and dynamic analysis tools can help identify these issues and mitigate their impact.

Static analysis is a technique that involves analyzing code without executing it. This is typically done by examining the source code to detect issues such as syntax errors, logical errors, and other potential issues. Static analysis tools can help identify issues in code before it is compiled or executed, which can reduce the likelihood of errors occurring at runtime. Examples of static analysis tools for Python include Bandit, Pylint, and Flake8.

Dynamic analysis, on the other hand, involves analyzing code during run-time. This can be done by using a debugger or a profiler, or by running the code in a test environment. Dynamic analysis tools can help identify issues such as memory leaks, performance problems, and security vulnerabilities. Examples of dynamic analysis tools for Python include PyCharm, PyDev, and Spyder.

Both static and dynamic analysis tools are important for identifying safety and security issues in Python code. For example, a static analysis tool can identify a potential buffer overflow vulnerability in code before it is compiled, while a dynamic analysis tool can identify an actual buffer overflow vulnerability as the code is running. By using both types of tools, developers can identify and address issues throughout the development process.

One specific example of the importance of static and dynamic analysis for safety and security in Python is the Heartbleed bug, which affected the OpenSSL cryptographic software library in 2014. The bug allowed attackers to read sensitive information from memory, including passwords and encryption keys. The bug was caused by a programming error in the OpenSSL code that resulted in a buffer overread. The bug went undetected for two years until it was discovered by a security researcher who used both static and dynamic analysis tools to identify the issue. This highlights the importance of using both types of analysis tools to identify potential safety and security issues in Python code.

STATIC CODE ANALYSIS USING BANDIT

Bandit is a widely used static analysis tool for Python code that is specifically designed to identify security vulnerabilities. It is an open-source tool that is used by developers and security professionals to analyze Python code for potential security issues. One of the main reasons that Bandit is used for Python code static analysis for safety and security issues is that it is designed to be easy to use and integrate into existing workflows. It can be run as a command-line tool, as part of a continuous integration (CI) pipeline or integrated into an integrated development environment (IDE). This makes it a flexible tool that can be used in a variety of contexts. Bandit works by analyzing Python code to identify potential security vulnerabilities such as SQL injection, cross-site scripting (XSS), and command injection. It does this by looking for patterns in the code that are known to be indicative of security vulnerabilities. For example, it can detect the use of untrusted inputs in SQL queries, which can be a potential vulnerability if the input is not properly sanitized.

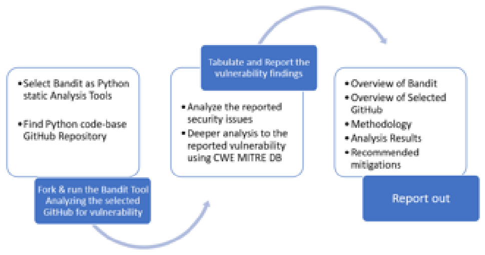
Another reason that Bandit is used for Python code static analysis for safety and security issues is that it is customizable. It comes with a set of default rules that cover common security issues, but it also allows users to define their own rules. This means that organizations can tailor the tool to their specific needs and identify potential security issues that are unique to their environment or application.

Bandit is also actively maintained and has a large community of contributors. This means that it is regularly updated with new security rules and bug fixes, ensuring that it remains an effective tool for identifying security vulnerabilities in Python code.

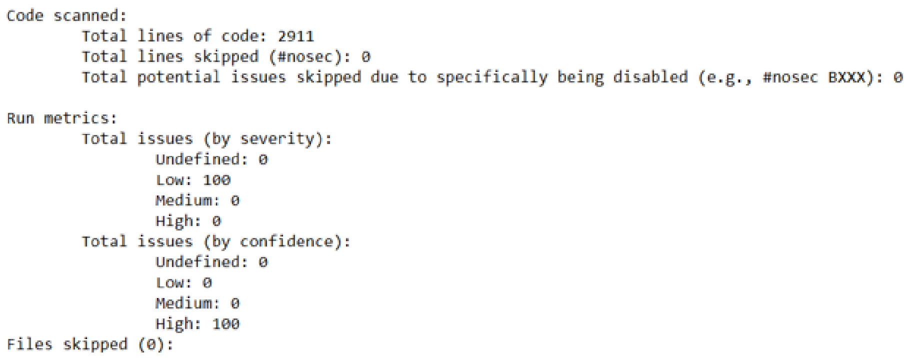
In summary, Bandit is a popular and effective static analysis tool for identifying security vulnerabilities in Python code. Its ease of use, flexibility, and customizable rules make it a valuable addition to any development or security workflow. By using Bandit in conjunction with other analysis tools and best practices, developers and security professionals can help ensure that Python code is secure and free from potential vulnerabilities.

METHODOLOGY

This project utilizes available open-source tools for static code analysis to expose code base security and vulnerability issues. We conducted a study for 62 open-source software repositories [7] to systematically characterize security issues using bandit static analysis tools [2] to identify & report all the security issues and frequency of occurrence. Figure 1 below shows the workflow and report snippets from Bandit analysis file. An example of security issues analysis in [2] and common weakness enumerations MITRE database[3] along with proposal to mitigate the security issues guided by [2] and [3] where recommended.



•Select Bandit as Python static Analysis Tools
•Find Python code-base GitHub Repository
Fork & run the Bandit Tool Analyzing the selected GitHub for vulnerability
Tabulate and Report the vulnerability findings
•Analyze the reported security issues
•Deeper analysis to the reported vulnerability using CWE MITRE DB
•Overview of Bandit
•Overview of Selected GitHub
•Methodology
•Analysis Results
•Recommended mitigations
Report out



```
Code scanned:
        Total lines of code: 2911
        Total lines skipped (#nosec): 0
        Total potential issues skipped due to specifically being disabled (e.g., #nosec BXXX): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 100
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 100
Files skipped (0):
```

Figure 1 project plan and Selected Code Base Repository

RESULTS & ANALYSIS

After installing the bandit tool, forking the selected repository, we were able to automatically run Bandit on any changed Python files during the CI process. This helped to catch potential security vulnerabilities early in the development process and provided a report of any issues found. The report was included in the Git commit [8]. This process will start the analysis and produce a report that lists any security issues that were found in the code. The report [4]. includes information about the severity of each issue, as well as a description of the vulnerability and a suggested solution. After completing all the analysis, we identify 3,973 security issues for all the 62 open-source software repositories. Figure 2 below shows the distribution of security issues. Found 3064 low severity, 326 medium severity and 46 high seveity.

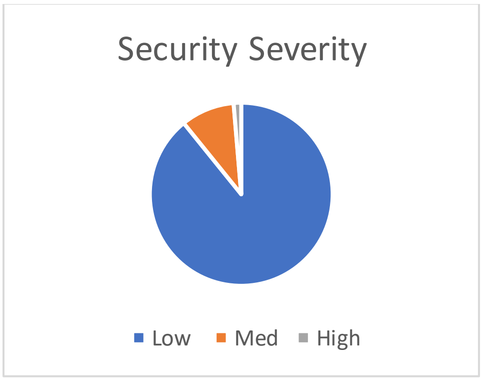


Figure 1 62 open-source software repositories security issues severity distributions

As an example of security issue analysis, we have selected one of the repository tabulated below in table 1.

Severity		CWE Issue	Location
High	3 B[605]	CWE-78 (https://cwe.mitre.org/data/definitions/78.html)	[1].theseus\examples\homography_estimation.py:62:12 [2].theseus\examples\homography_estimation.py:65:12 [3]. heseus\examples\homography_estimation.py:240:4
Med	2 B[102] B[307]	CWE-78 (https://cwe.mitre.org/data/definitions/78.html)	theseus\examples\example.py:5:0 theseus\examples\pose_graph\pose_graph_benchmark.py:39:12
Low	540 B[101]	703, 78	

Table 1 Reported Security Weakness by Bandit

To analyze the reported issues, once we have run the Bandit tool and obtained the results, we use the Common Weakness Enumeration (CWE) database wiki to analyze the findings. The CWE database is a comprehensive list of common security vulnerabilities and weaknesses that can be found in software. To use the CWE database, you first need to identify the CWE ID of each security issue that was found by Bandit. The CWE ID is a unique identifier that is used to categorize each vulnerability. For example, if Bandit identified a Starting a process with a shell, possible injection detected, security issue, the CWE ID would be CWE-78. Once you have identified the CWE ID, we can search for it in the CWE database wiki. The wiki provides detailed information about each vulnerability, including a description of the weakness, examples of how it can be exploited, and suggestions for how to prevent it. By analyzing the Bandit results using the CWE database, we can gain a deeper understanding of the security issues in Python code and develop more effective strategies for addressing them.

As an example, the vulnerability issue "Issue: [B605:start_process_with_a_shell] Starting a process with a shell, possible injection detected, security issue". We first analyze the Bandit issue by reviewing the python code "homography_estimation.py" and locate the security code line 62:12, then we can review the bandit reported issue(https://bandit.readthedocs.io/en/1.7.4/plugins/b605_start_process_with_a_shell.html) starting a process with shell, that highlights that this type of subprocess invocation is dangerous as it is vulnerable to various shell injection attacks. Great care should be taken to sanitize all input to mitigate this risk. Calls of this type are identified using certain commands which are known to use shells. Bandit will report a LOW severity warning. Next step is to review the CWE-78 to get more insight and understand available mitigations, the issues show that this could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process does not follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increase the amount of damage.

A possible mitigation to protect against any probably vulnerability of using ***os.system(cmd)*** python command is to use subprocess access ***subprocess.*** So instead of using os.system(cmd) the safer option is to use subprocess.call(arg) or subprocess.run(arg) [5] & [6].

CONSULSION

Bandit is a powerful tool for Python software static analysis that can help identify common security vulnerabilities. By using the tool in combination with the Common Weakness Enumeration (CWE) database wiki, developers can gain a better understanding of the security issues in their code and develop more effective strategies for addressing them. Overall, the use of static analysis tools such as Bandit is an essential component of any comprehensive software security strategy.

REFERENCES

[1]. Facebook research theseus *A library for differentiable nonlinear optimization* github (https://github.com/facebookresearch/theseus).

[2]. Bandit: tool designed to find common security issues in Python code (https://bandit.readthedocs.io/en/latest/#)

[3]. MITRE: Common Weakness Enumeration Data base (https://cwe.mitre.org/data/definitions/699.html)

[4]. Security weakness project report (https://github.com/IssaAljanabi/theseus/blob/main/theseus-bandit-report.txt)

[5]. Calling Shell Commands from Python: OS.system vs Subprocess (Calling Shell Commands from Python: OS.system vs Subprocess | HackerNoon)

[6]. 10 common security gotchas in Python and how to avoid them (10 common security gotchas in Python and how to avoid them | HackerNoon)

[7] open-source repositories for bandit static code analysis security test reports

https://github.com/danijar/dreamerv2
https://github.com/erlerobot/gym-gazebo
https://github.com/ARISE-Initiative/robosuite
https://github.com/nicrusso7/rex-gym
https://github.com/microsoft/PromptCraft-Robotics
https://github.com/facebookresearch/habitat-lab
https://github.com/gradslam/gradslam
https://github.com/DLR-RM/rl-baselines3-zoo
https://github.com/facebookresearch/theseus
https://github.com/yangyanli/PointCNN
https://github.com/petercorke/robotics-toolbox-python
https://github.com/charlesq34/frustum-pointnets
https://github.com/xinshuoweng/AB3DMOT
https://github.com/facebookresearch/votenet
https://github.com/google/brax
https://github.com/MichaelGrupp/evo
https://github.com/zauberzeug/nicegui
https://github.com/DLR-RM/stable-baselines3
https://github.com/AtsushiSakai/PythonRobotics
https://github.com/navneet-nmk/pytorch-rl
https://github.com/danijar/dreamer
https://github.com/hungpham2511/toppra
https://github.com/opendr-eu/opendr
https://github.com/undera/pylgbst
https://github.com/Phylliade/ikpy
https://github.com/RobotLocomotion/pytorch-dense-correspondence
https://github.com/anki/vector-python-sdk
https://github.com/moble/quaternion
https://github.com/BYU-PCCL/holodeck
https://github.com/google/mobly
https://github.com/araffin/robotics-rl-srl
https://github.com/cbfinn/gps
https://github.com/evildmp/BrachioGraph
https://github.com/devendrachaplot/Neural-SLAM
https://github.com/ros/ros_comm
https://github.com/mithi/hexapod-robot-simulator
https://github.com/utiasDSL/gym-pybullet-drones

https://github.com/PRBonn/kiss-icp
https://github.com/andyzeng/visual-pushing-grasping
https://github.com/tianheyu927/mil
https://github.com/engnadeau/pybotics
https://github.com/qgallouedec/panda-gym
https://github.com/aravindr93/mjrl
https://github.com/Tobias-Fischer/rt_gene
https://github.com/microsoft/AirSim-NeurIPS2019-Drone-Racing
https://github.com/DJTobias/Cherry-Autonomous-Racecar
https://github.com/jr-robotics/robo-gym
https://github.com/PRBonn/bonnet
https://github.com/splintered-reality/py_trees
https://github.com/wil3/gymfc
https://github.com/AcutronicRobotics/gym-gazebo2
https://github.com/avisingh599/reward-learning-rl
https://github.com/utiasDSL/safe-control-gym
https://github.com/Vincentqyw/cv-arxiv-daily
https://github.com/pypose/pypose
https://github.com/yuce/pyswip
https://github.com/google-research/ravens
https://github.com/dougsm/ggcnn
https://github.com/clvrai/furniture
https://github.com/BerkeleyAutomation/gqcnn
https://github.com/foxglove/mcap
https://github.com/engcang/SLAM-application
https://github.com/wenbowen123/iros20-6d-pose-tracking
https://github.com/yuanmingqi/rl-exploration-baselines
https://github.com/Tai-Wang/Depth-from-Motion
https://github.com/NVIDIA-Omniverse/Orbit
[8]. https://github.com/IssaAljanabi/COMP-6702-D02-Project Bandit security reports for all the repositories in [7]