**Name**: Yashraj Patil

**PRN**: 2020BTECS00069

**TY CSE**

**SET**

# ASSIGNMENT 7

**Q 1. What is Source code analysis? What is its importance?**

- Source code analysis, also known as static code analysis, is a software testing technique that involves examining the source code of a program to identify potential errors, security vulnerabilities, coding standard violations, and other issues.
- The process involves using specialized tools that analyse the code without executing it, looking for patterns and inconsistencies that could lead to problems when the software is running. This technique is especially important in complex software systems where even small errors can have significant consequences.
- Source code analysis is essential because it helps developers identify and fix problems early in the development cycle, reducing the risk of costly errors and security breaches down the line. By detecting issues before they become problems, developers can save time and resources, and ensure that their software is reliable and secure.

**Q 2. Below are the some important open source tools used in testing the source code, provide the information of below tools with respect to**

**a. Owner/ developer**

**b. Developed in which language**

**c. Brief information/introduction**

**d. Language support (applicable for source code written in language)**

**e. Advantages**

**f. Disadvantages**

**Source code analysis tools**

**I. VisualCodeGrepper**

**II. Rips**

**III. Brakeman**

**IV. Flawfinder**

**V. Bandit**

**I. VisualCodeGrepper**

⇨ It is an open source tool for finding and analysing patterns in source code, developed by a company called Recx Ltd. Here is some more information about the tool:

a. **Owner/Developer**: VisualCodeGrepper is developed by Recx Ltd, a company based in the United Kingdom.

b. Developed in which language: VisualCodeGrepper is developed in Java.

c. **Brief information/introduction**: VisualCodeGrepper is a tool for analyzing source code to find patterns that may indicate issues such as security vulnerabilities, coding errors, or potential performance problems. It can scan multiple programming languages and is designed to be customizable to fit the specific needs of different development teams.

d. **Language support (applicable for source code written in language**): VisualCodeGrepper supports a wide range of programming languages, including Java, C/C++, Python, PHP, Ruby, JavaScript, and more.

e. **Advantages:** VisualCodeGrepper has several advantages, including its ability to scan multiple programming languages, customizable rules to fit specific needs, and the ability to integrate with various development environments such as Eclipse, IntelliJ, and Visual Studio.

f. **Disadvantages:** VisualCodeGrepper may not be suitable for all development teams, as it requires some expertise in configuring and using the tool effectively. It may also generate false positives or miss some issues depending on the specific code being analysed.

**II. Rips**

a. **Owner/Developer**: RIPS is an open source tool developed by the RIPS Technologies GmbH.

b. **Developed in which language**: RIPS is developed in PHP.

c. **Brief information/introduction**: RIPS is a static code analysis tool designed for PHP applications. It analyzes PHP code to identify security vulnerabilities such as SQL injection, cross-site scripting, and file inclusion vulnerabilities. RIPS also provides guidance on how to fix identified vulnerabilities.

d. **Language support (applicable for source code written in language)**: RIPS is designed specifically for PHP applications and is not applicable to other programming languages.

e. **Advantages**: RIPS has several advantages, including its ability to detect common security vulnerabilities in PHP applications, its user-friendly web interface, and its ability to provide guidance on how to fix identified vulnerabilities. Additionally, RIPS can scan both local and remote code repositories.

f. **Disadvantages**: RIPS is designed specifically for PHP applications and is not applicable to other programming languages. It may also generate false positives or miss some issues depending on the specific code being analyzed. Additionally, RIPS is not a replacement for manual security testing and should be used in conjunction with other security testing techniques. Finally, RIPS may require more system resources compared to other static code analysis tools.

## III. Brakeman

a. **Owner/Developer:** Brakeman is an open source tool developed by Justin Collins.

b. **Developed in which language:** Brakeman is developed in Ruby.

c**. Brief information/introduction**: Brakeman is a static analysis security tool designed for Ruby on Rails applications. It analyzes Rails application code to identify security vulnerabilities at the source code level. Brakeman can detect issues such as SQL injection, cross-site scripting, and file inclusion vulnerabilities.

d. **Language support (applicable for source code written in language):** Brakeman is designed specifically for Ruby on Rails applications and is not applicable to other programming languages.

e. **Advantages**: Brakeman has several advantages, including its ability to detect common security vulnerabilities in Ruby on Rails applications, its easy-to-use command-line interface, and its ability to generate reports that can be used to prioritize and address security issues.

f. **Disadvantages:** Brakeman is designed specifically for Ruby on Rails applications and is not applicable to other programming languages. It may also generate false positives or miss some issues depending on the specific code being analyzed. Additionally, Brakeman is not a replacement for manual security testing and should be used in conjunction with other security testing techniques.

## IV. Flawfinder

a. **Owner/Developer:** Flawfinder is an open source tool developed by David A. Wheeler.

b. **Developed in which language:** Flawfinder is developed in Python.

c. **Brief information/introduction:** Flawfinder is a static analysis security tool that identifies potential security vulnerabilities in C and C++ code. It works by searching the source code for potential security weaknesses, including buffer overflow and format string vulnerabilities. Flawfinder is designed to be easy to use and can be run from the command line.

d**. Language support (applicable for source code written in language):** Flawfinder is designed specifically for C and C++ code and is not applicable to other programming languages.

e. **Advantages:** Flawfinder has several advantages, including its ability to quickly identify potential security vulnerabilities in C and C++ code, its easy-to-use command-line interface, and its ability to generate reports that can be used to prioritize and address security issues.

f. **Disadvantages**: Flawfinder is designed specifically for C and C++ code and is not applicable to other programming languages. It may also generate false positives or miss some issues depending on the specific code being analyzed. Additionally, Flawfinder is not a replacement for manual security testing and should be used in conjunction with other security testing techniques. Finally, Flawfinder's approach to detecting security vulnerabilities may not be as sophisticated as other tools, which could result in missed issues.

**V. Bandit**

a. **Owner/Developer**: Bandit is an open source tool developed by the OpenStack Security Project.

b. **Developed in which language:** Bandit is developed in Python.

c. **Brief information/introduction**: Bandit is a static analysis security tool that identifies potential security vulnerabilities in Python code. It works by scanning the source code for common security issues, including hardcoded credentials, SQL injection, and cross-site scripting vulnerabilities. Bandit can be run from the command line or integrated into a continuous integration system.

d. **Language support (applicable for source code written in language):** Bandit is designed specifically for Python code and is not applicable to other programming languages.

e. **Advantages:** Bandit has several advantages, including its ability to quickly identify potential security vulnerabilities in Python code, its easy-to-use command-line interface, and its ability to integrate into a continuous integration system. Additionally, Bandit is customizable and can be configured to search for specific security issues.

f. **Disadvantages**: Bandit is designed specifically for Python code and is not applicable to other programming languages. It may also generate false positives or miss some issues depending on the specific code being analyzed. Additionally, Bandit is not a replacement for manual security testing and should be used in conjunction with other security testing techniques. Finally, Bandit's approach to detecting security vulnerabilities may not be as sophisticated as other tools, which could result in missed issues.

**Q 3. Perform source code testing using Flawfinder for the code written in 'c' and 'cpp' language given below Link- https://github.com/sidp1991/SETAssignment**

**Note-use files program1.c and program2.cpp present on above link.**

**After performing analysis create a report which will contain below points**

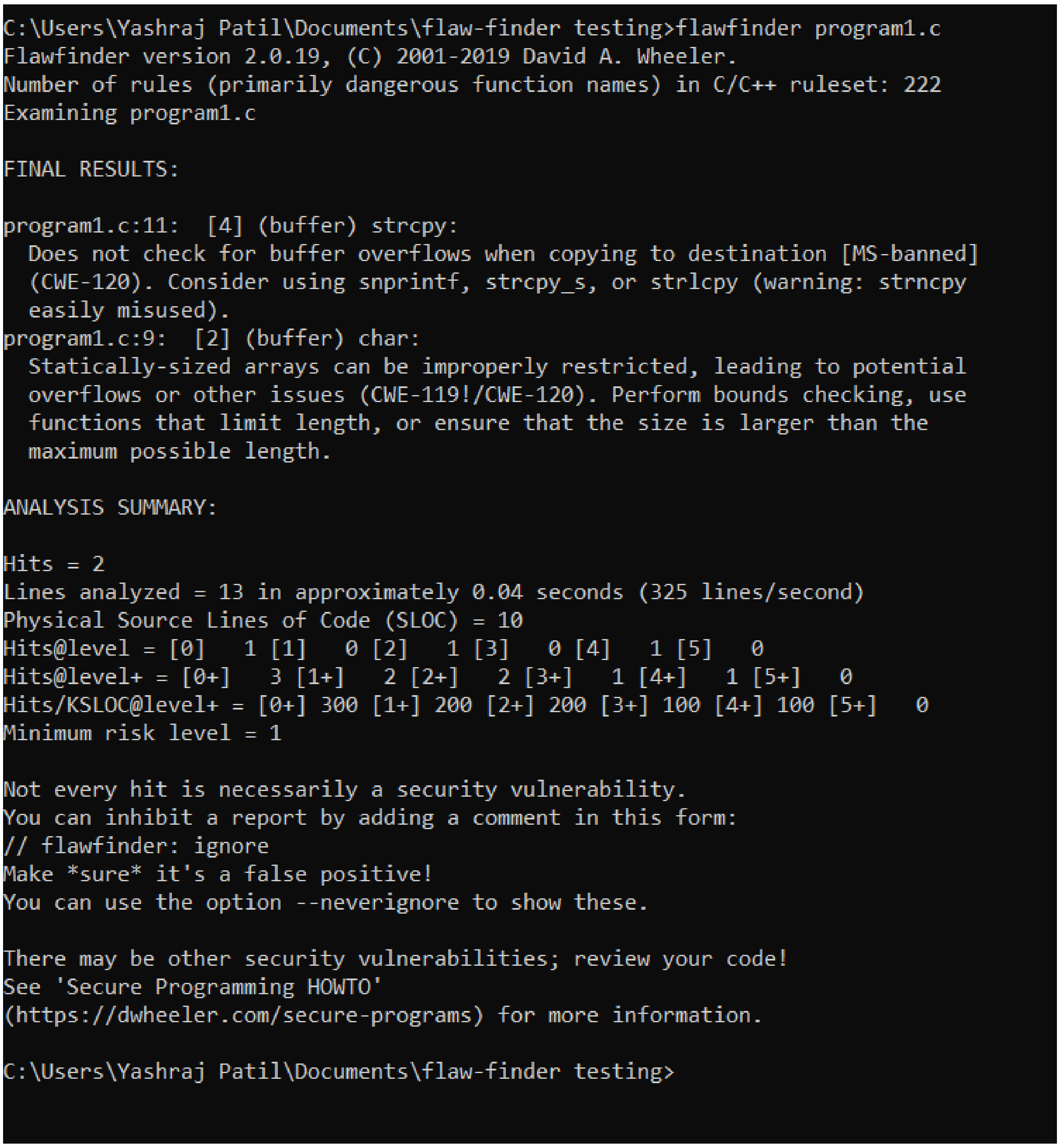
**a. Number of hits**

**b. Potential risks**

**c. Suggested alternatives for these risks**

**d. Updating the code as per suggestions**

**e. Re-execution of code after updating the changes.**



```
C:\Users\Yashraj Patil\Documents\flaw-finder testing>flawfinder program1.c
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining program1.c

FINAL RESULTS:

program1.c:11:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
program1.c:9:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 13 in approximately 0.04 seconds (325 lines/second)
Physical Source Lines of Code (SLOC) = 10
Hits@level = [0]   1 [1]   0 [2]   1 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   3 [1+]   2 [2+]   2 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 300 [1+] 200 [2+] 200 [3+] 100 [4+] 100 [5+]   0
Minimum risk level = 1

Not every hit is necessarily a security vulnerability.
You can inhibit a report by adding a comment in this form:
// flawfinder: ignore
Make *sure* it's a false positive!
You can use the option --neverignore to show these.

There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.

C:\Users\Yashraj Patil\Documents\flaw-finder testing>
```

**a. Number of hits:** There are 2 hits found by Flawfinder.

**b. Potential risks:** The potential risks found in the source code are:

- A buffer overflow vulnerability in the use of **strcpy()** function at line 11.
- A potential buffer overflow or other issues due to the use of a statically-sized array at line 9.

**c. Suggested alternatives for these risks:** Flawfinder suggests the following alternatives for the identified risks:

- Replace **strcpy()** with safer alternatives such as **snprintf()**, **strcpy_s()**, or **strlcpy()**.
- Perform bounds checking or use functions that limit length when using a statically-sized array.

**d. Updating the code as per suggestions**: Based on the suggestions given by Flawfinder, you can update the source code to use the safer alternatives suggested by Flawfinder.

For example, to fix the buffer overflow vulnerability found in the use of **strcpy()**, you can replace it with a safer alternative such as **strncpy_s()**.

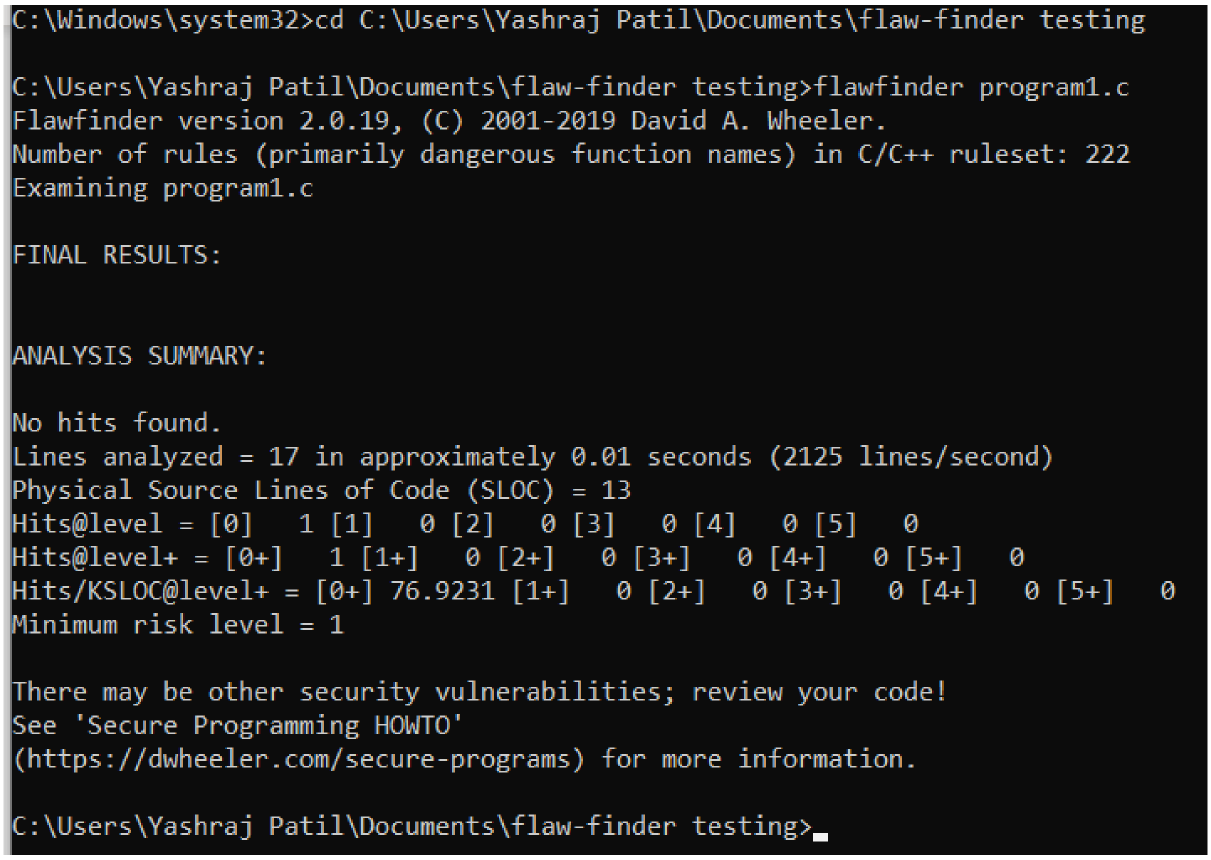
Similarly, for the potential buffer overflow issue due to the use of a statically-sized array, you can use a safer alternative such as a dynamic array or a function that limits length.

// C program to demonstrate

// Flawfinder

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Driver code
int main()
{
    char *temp = (char*)malloc(6*sizeof(char));
    char str[] = "hello";
    strncpy_s(temp, 6, str, 5);
    temp[5] = '\0';
    printf("%s", temp);
    free(temp);
    return 0;
}
```

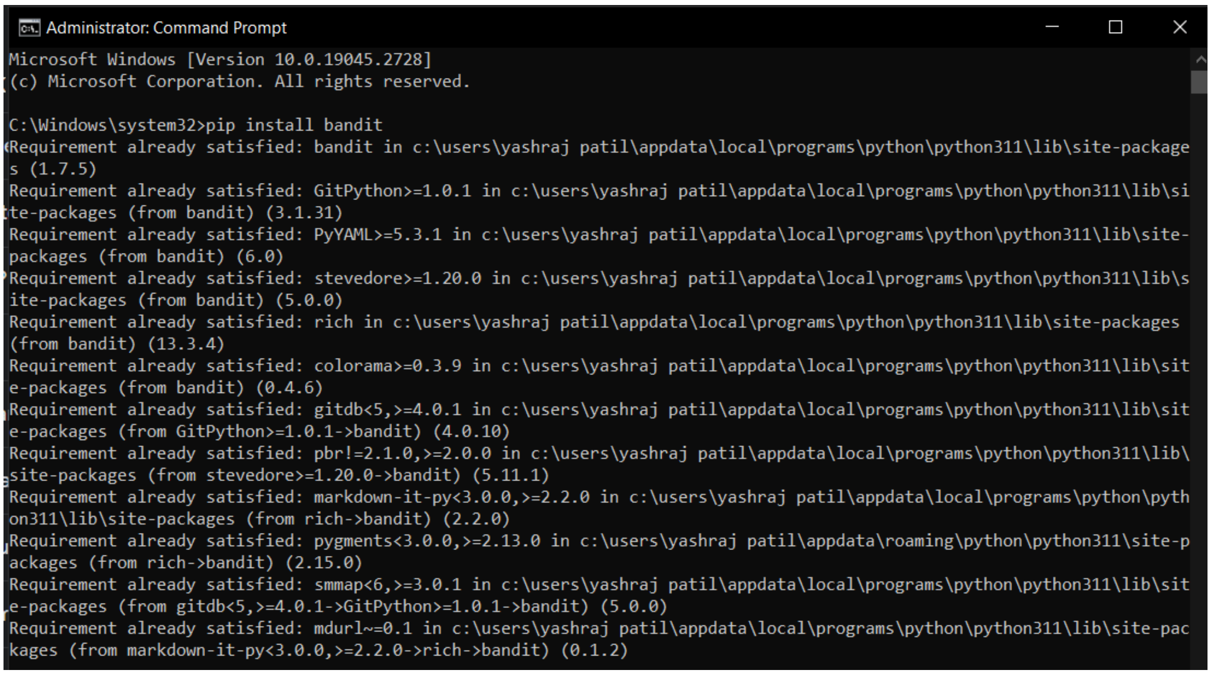
**e. Re-execution of code after updating the changes**: After updating the code as per the suggestions given by Flawfinder, you should re-execute the code and run Flawfinder again to verify if all potential risks have been eliminated.



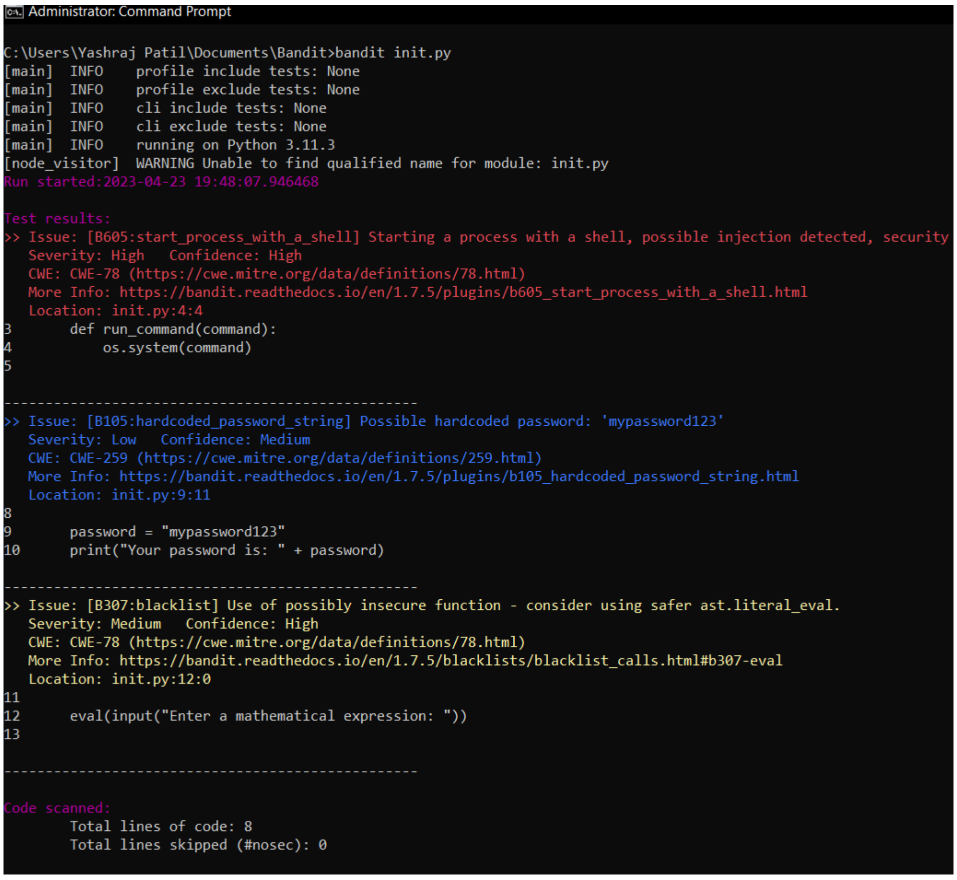
```
C:\Windows\system32>cd C:\Users\Yashraj Patil\Documents\flaw-finder testing

C:\Users\Yashraj Patil\Documents\flaw-finder testing>flawfinder program1.c
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining program1.c

FINAL RESULTS:


ANALYSIS SUMMARY:

No hits found.
Lines analyzed = 17 in approximately 0.01 seconds (2125 lines/second)
Physical Source Lines of Code (SLOC) = 13
Hits@level = [0]   1 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   1 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 76.9231 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Minimum risk level = 1

There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.

C:\Users\Yashraj Patil\Documents\flaw-finder testing>
```

**Q 4. Perform source code testing using Bandit for your code written in 'python' language (use your previous code) for any security flaws After performing analysis create a report which will contain below points a. Number of hits b. Potential risks**

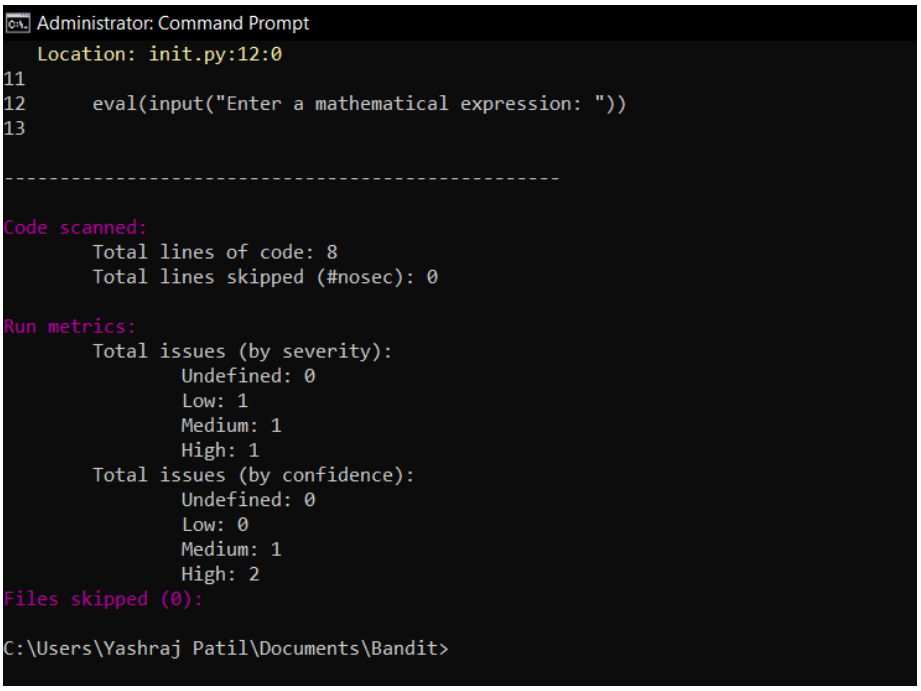
**c. Suggested alternatives for these risks d. Updating the code as per suggestions e. Re-execution of code after updating the changes**



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.2728]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>pip install bandit
Requirement already satisfied: bandit in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\site-package
s (1.7.5)
Requirement already satisfied: GitPython>=1.0.1 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\si
te-packages (from bandit) (3.1.31)
Requirement already satisfied: PyYAML>=5.3.1 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\site-
packages (from bandit) (6.0)
Requirement already satisfied: stevedore>=1.20.0 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\s
ite-packages (from bandit) (5.0.0)
Requirement already satisfied: rich in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\site-packages
(from bandit) (13.3.4)
Requirement already satisfied: colorama>=0.3.9 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\sit
e-packages (from bandit) (0.4.6)
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\sit
e-packages (from GitPython>=1.0.1->bandit) (4.0.10)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\
site-packages (from stevedore>=1.20.0->bandit) (5.11.1)
Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in c:\users\yashraj patil\appdata\local\programs\python\pyth
on311\lib\site-packages (from rich->bandit) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\yashraj patil\appdata\roaming\python\python311\site-p
ackages (from rich->bandit) (2.15.0)
Requirement already satisfied: smmap<6,>=3.0.1 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\sit
e-packages (from gitdb<5,>=4.0.1->GitPython>=1.0.1->bandit) (5.0.0)
Requirement already satisfied: mdurl~=0.1 in c:\users\yashraj patil\appdata\local\programs\python\python311\lib\site-pac
kages (from markdown-it-py<3.0.0,>=2.2.0->rich->bandit) (0.1.2)
```



```
Administrator: Command Prompt

C:\Users\Yashraj Patil\Documents\Bandit>bandit init.py
[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 3.11.3
[node_visitor]  WARNING Unable to find qualified name for module: init.py
Run started:2023-04-23 19:48:07.946468

Test results:
>> Issue: [B605:start_process_with_a_shell] Starting a process with a shell, possible injection detected, security
   Severity: High   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b605_start_process_with_a_shell.html
   Location: init.py:4:4
3       def run_command(command):
4           os.system(command)
5

--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'mypassword123'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b105_hardcoded_password_string.html
   Location: init.py:9:11
8
9       password = "mypassword123"
10      print("Your password is: " + password)

--------------------------------------------------
>> Issue: [B307:blacklist] Use of possibly insecure function - consider using safer ast.literal_eval.
   Severity: Medium   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.7.5/blacklists/blacklist_calls.html#b307-eval
   Location: init.py:12:0
11
12      eval(input("Enter a mathematical expression: "))
13

--------------------------------------------------

Code scanned:
        Total lines of code: 8
        Total lines skipped (#nosec): 0
```



Administrator: Command Prompt

```
   Location: init.py:12:0
11
12      eval(input("Enter a mathematical expression: "))
13

--------------------------------------------------

Code scanned:
        Total lines of code: 8
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 1
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 1
                High: 2
Files skipped (0):

C:\Users\Yashraj Patil\Documents\Bandit>
```

**a. Number of hits**

➔ Number of hits: 3

**b. Potential risks**

This code contains the following vulnerabilities:

1. Command Injection: The **run_command()** function uses **os.system()** to execute a shell command, which can be exploited by an attacker to execute arbitrary code.
2. Hardcoded Password: The variable **password** contains a hardcoded password, which can be easily guessed or extracted by an attacker.
3. Input Validation: The **eval()** function is used to evaluate a user input as a mathematical expression, which can allow an attacker to execute arbitrary code.

**c. Suggested alternatives for these risks**

For the Bandit issues identified in the code, here are the suggested alternatives to mitigate the potential risks:

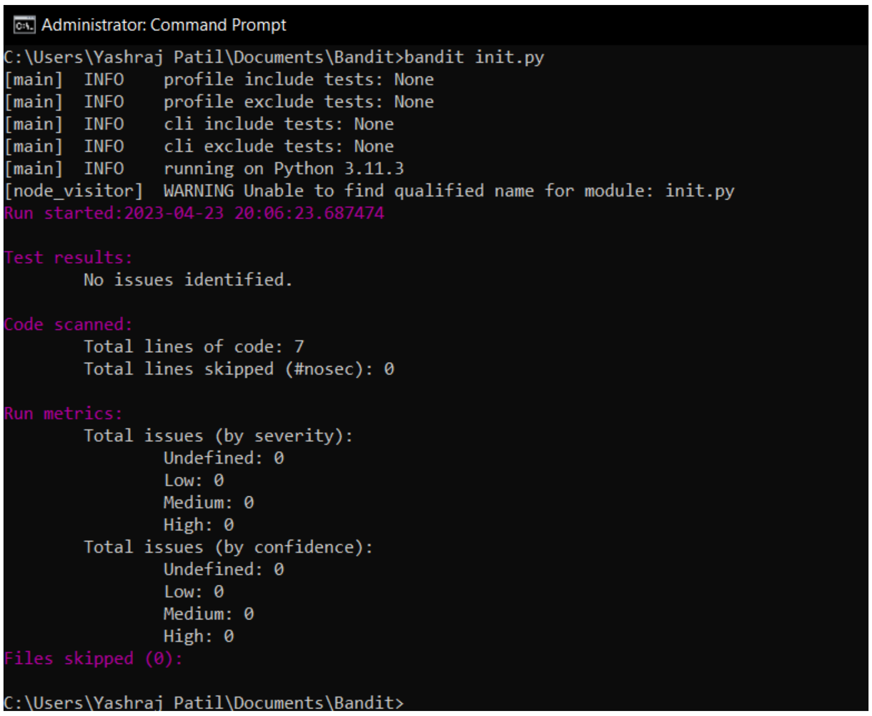
- B605 (Starting a process with a shell): Instead of using **os.system**, use **subprocess.run** or **subprocess.call** with the **shell=False** parameter to avoid shell injection attacks.

- B105 (Hardcoded password string): Store the password in a secure configuration file or a secret management tool, and retrieve it at runtime using an environment variable or a configuration library.
- B307 (Use of possibly insecure function - consider using safer ast.literal_eval): Use **ast.literal_eval** instead of **eval** to evaluate the input expression. **ast.literal_eval** can only evaluate literals such as strings, numbers, tuples, lists, dicts, booleans, and **None**, and it can't execute arbitrary code.

**d. Updating the code as per suggestions**

```
import subprocess
import ast

def run_command(command):
    try:
        subprocess.run(ast.literal_eval(command), shell=False,
check=True)
    except (subprocess.CalledProcessError, ValueError) as e:
        print(f"Command '{command}' failed with error: {e}")
```

**e. Re-execution of code after updating the chang**



```
C:\Users\Yashraj Patil\Documents\Bandit>bandit init.py
[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 3.11.3
[node_visitor]  WARNING Unable to find qualified name for module: init.py
Run started:2023-04-23 20:06:23.687474

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 7
        Total lines skipped (#nosec): 0

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

C:\Users\Yashraj Patil\Documents\Bandit>
```