Software Engineering Tools Lab
Assignment No-7

PRN: 2020BTECS00033
Name: Prathamesh Raje
Batch: T6

Q. What is Source code analysis? What is its importance?
Ans:

1. Source code analysis is the automated testing of a program's source code with the purpose of finding faults and fixing them before the application is sold or distributed.
2. The aim is to find bugs and faults that may not be obvious to a programmer.
3. It is meant to find faults like possible buffer overflows or untidy use of pointers and misuse of garbage collection functions, all of which may be exploitable by a hacker.
4. Source code analysis is synonymous to static code analysis, where the source code is analysed simply as code and the program is not running.
5. This removes the need for creating and using test cases, and may separate itself from feature-specific bugs like buttons being a different colour than what the specifications say.
6. It concentrates on finding faults in the program that may be detrimental to its proper function like crash-causing lines of code.

Q. 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**

Ans:

1) VisualCodeGrepper:

a. Owner/ developer: National Computing Center (NCC) Group.

b. Developed in which language: Visual Basic .NET.

c. Brief information/introduction:
VCG is an automated code security review tool, which is intended to speed up the code review process by identifying bad/insecure code.
It has a few features that should make it useful. In addition to performing some more complex checks it also has a config file for each language that basically allows you to add any bad functions (or other text) that you want to search for. It attempts to find phrases within comments that can indicate broken code and it provides stats and a pie chart (for the entire codebase and for individual files) showing relative proportions of code, whitespace, comments, 'ToDo'-style comments and bad code.

d. Language support (applicable for source code written in language):
C++, C#, VB, PHP, Java, PL/SQL and COBOL.

e. Advantages:
   1. Not only does this tool perform complex checks, it also has a configuration file for each language that allows the user to run custom queries.
   2. It tells its users the security level of the vulnerabilities they find in their code.
   3. Searches for any specific violations of OWASP recommendations.
   4. This tool is constantly updated since its release in 2012.

f. Disadvantages:
1. Even though it has the capability of analyzing many languages, the user has to always tell the tool what language their code is written in.
2. None of the vulnerabilities it finds cannot be modified right there on the application itself -- users are prompted to use a separate tool for that.
3. It is not fully automated.

2) Rips:

a. Owner/ developer: RIPS (Research and Innovation to Promote Security) Technologies.

b. Developed in which language: PHP, CSS, JS.

c. Brief information/introduction:
RIPS (Research and Innovation to Promote Security) is a static code analysis software for the automated detection of security vulnerabilities in PHP and Java applications. It uses abstract syntax trees, control-flow graphs, and context-sensitive taint analysis in order to accurately identify even complex security vulnerabilities that are based on second-order data flows or misplaced security mechanisms.
RIPS is available as on-premises software and as Software-as-a-Service.

d. Language support (applicable for source code written in language):
PHP & Java.

e. Advantages:

1. Accurate identification of security vulnerabilities: The RIPS tool is highly accurate in identifying security vulnerabilities in PHP code. It uses a combination of static code analysis and heuristic analysis to identify potential security issues.

2. Easy to use: The RIPS tool is easy to use, even for those who are not experts in security. It has a userfriendly interface that allows users to quickly scan their code for vulnerabilities.

3. Customizable: The RIPS tool is highly customizable, allowing users to configure it to their specific needs. Users can choose which types of vulnerabilities to scan for, as well as configure various other settings.

4. Integration with development tools: The RIPS tool can be integrated with various development tools, including IDEs, build systems, and continuous integration servers. This allows developers to incorporate security scanning into their existing workflows.

5. Detailed reports: The RIPS tool provides detailed reports on the vulnerabilities it finds, including information on the location of the vulnerability, the severity of the issue, and recommended remediation steps. This helps developers quickly identify and fix security issues.

f. Disadvantages:

1. False Positives: Like any automated testing tool, RIPS can generate false positives, flagging code as vulnerable when it is actually secure. This can lead to wasted time and effort in reviewing and testing code that doesn't need it.

2. False Negatives: On the other hand, RIPS can also miss certain types of vulnerabilities, especially if they are complex or well-hidden. This can give developers a false sense of security, leading them to overlook critical issues.

3. Limited Language Support: RIPS is designed specifically for PHP applications, which means it may not be the best choice for organizations that work with other programming languages. This can be a limitation for companies with a diverse tech stack.

4. Cost: While RIPS offers a free version, the fullfeatured tool requires a paid license. For small organizations or individual developers, the cost may be prohibitive.

5. Learning Curve: RIPS has a steep learning curve, and it can take some time for developers to become proficient in using the tool effectively. This can be a barrier for smaller teams or companies with limited resources.

3) Brakeman:

a. Owner/ developer: Justin Collins.

b. Developed in which language: Ruby.

c. Brief information/introduction:
Brakeman is a security scanner for Ruby on Rails applications. Unlike many web security scanners, Brakeman looks at the source code of your application. This means you do not need to set up your whole

application stack to use it. Once Brakeman scans the application code, it produces a report of all security issues it has found.

d. Language support (applicable for source code written in language): Ruby on Rails.

e. Advantages:

1. No Configuration Necessary: Brakeman requires zero setup or configuration once it is installed. Just run it.

2. Run It Anytime: Because all Brakeman needs is source code, Brakeman can be run at any stage of development: you can generate a new application with rails new and immediately check it with Brakeman.

3. Better Coverage: Since Brakeman does not rely on spidering sites to determine all their pages, it can provide more complete coverage of an application. This includes pages which may not be 'live' yet. In theory, Brakeman can find security vulnerabilities before they become exploitable.

4. Best Practices: Brakeman is specifically built for Ruby on Rails applications, so it can easily check configuration settings for best practices.

5. Flexible Testing: Each check performed by Brakeman is independent, so testing can be limited to a subset of all the checks Brakeman comes with.

6. Speed: While Brakeman may not be exceptionally speedy, it is much faster than "black box" website scanners. Even large applications should not take more than a few minutes to scan.

f. Disadvantages:

1. False Positives: Only the developers of an application can understand if certain values are dangerous or not. By default, Brakeman is extremely suspicious. This can lead to many "false positives."

2. Unusual Configurations: Brakeman assumes a "typical" Rails setup. There may be parts of an application which are missed because they do not fall within the normal Rails application layout.

3. Only Knows Code: Dynamic vulnerability scanners which run against a live website are able to test the entire application stack, including the webserver and database. Naturally, Brakeman will not be able to report if a webserver or other software has security issues.

4. Isn't Omniscient: Brakeman cannot understand everything which is happening in the code. Sometimes it just makes reasonable assumptions. It may miss things. It may misinterpret things. But it tries its best. Remember, if you run across something strange, feel free to file an issue for it.

4) Flawfinder:

a. Owner/ developer: David A. Wheeler.

b. Developed in which language: Python & Roff.

c. Brief information/introduction:
FlawFinder is a python based tool that helps in finding vulnerabilities in a C/C++ source code. It examines the source code and gives the list of possible vulnerabilities/flaws in the code as the output. It is free for anyone to use and is available as open source software (OSS).

d. Language support (applicable for source code written in language): C, C++.

e. Advantages:

1. Determines level of risk- Flawfinder renders a list of potential security vulnerabilities which are sorted by risk. Functions and parameters used in the code determine the level of risk. For example, constant string values are less risky in comparison to variable strings. In some cases, FlawFinder may be able to determine that the construct isn't risky at all, reducing false positives.

2. Provides analysis summary- It produces an analysis summary as output and mentions the no. of hits i.e. vulnerabilities found in the code.

3. Code is not compiled- The source code is never compiled and hence even if the code is not working the tool will still render the list of vulnerabilities almost immediately.

f. Disadvantages:

1. Does not guarantee to find all the vulnerabilities- Every hit produced doesn’t imply a security vulnerability and neither is every vulnerability found. For instance, in a simple division program, a number is divided by 0. Ideally, the tool should show division by 0 as a hit but it fails to do so. This is because the tool cannot understand the logic of the program.

2. Cannot detect malicious codes- Flawfinder looks for specific patterns known to be common mistakes in application code. Thus, it is likely to be less effective in analyzing programs that might contain malicious codes.

5) Bandit:

a. Owner/ developer: PyCQA (Python Code Quality Authority).

b. Developed in which language: Python.

c. Brief information/introduction: Bandit is a tool designed to find common security issues in Python code. To do this Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report.

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

e. Advantages:

1. Automated testing: Bandit tools provide automated testing of Python code for security vulnerabilities. This can save a lot of time compared to manual testing, especially when testing large codebases.

2. Identifies common vulnerabilities: Bandit tools have built-in rules that check for common security vulnerabilities in Python code. This can help identify issues that might be overlooked during manual testing.

3. Customizable: Bandit tools allow users to create their own rules or modify existing rules to suit their specific needs.

4. Integration with CI/CD pipelines: Bandit tools can be integrated with Continuous Integration/Continuous Deployment (CI/CD) pipelines, allowing for automated security testing as part of the software development process.

5. Open source: Bandit is an open-source tool, meaning that it is free to use and has a large community of contributors who are constantly updating and improving the tool.

f. Disadvantages:

1. Limited Scope: The bandit tool is designed to detect security issues, but it has a limited scope and may not catch all vulnerabilities. It is important to use other testing tools and methods in addition to bandit to ensure comprehensive testing.

2. False Positives: Static analysis tools like bandit can sometimes generate false positive results, meaning they report a vulnerability that does not actually exist. This can be time-consuming to investigate and can reduce the effectiveness of the tool.

3. False Negatives: Conversely, bandit may also fail to detect real vulnerabilities, resulting in false negatives. This can lead to a false sense of security and potentially leave the code open to exploitation.

4. Limited Language Support: Bandit is specifically designed for Python code, and may not be effective for testing code written in other programming languages.

5. Lack of Context: Static analysis tools like bandit rely solely on the code itself, without taking into account the larger context in which it is used. This can make it difficult to identify complex security issues that may require a more nuanced approach to testing.

Q. 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

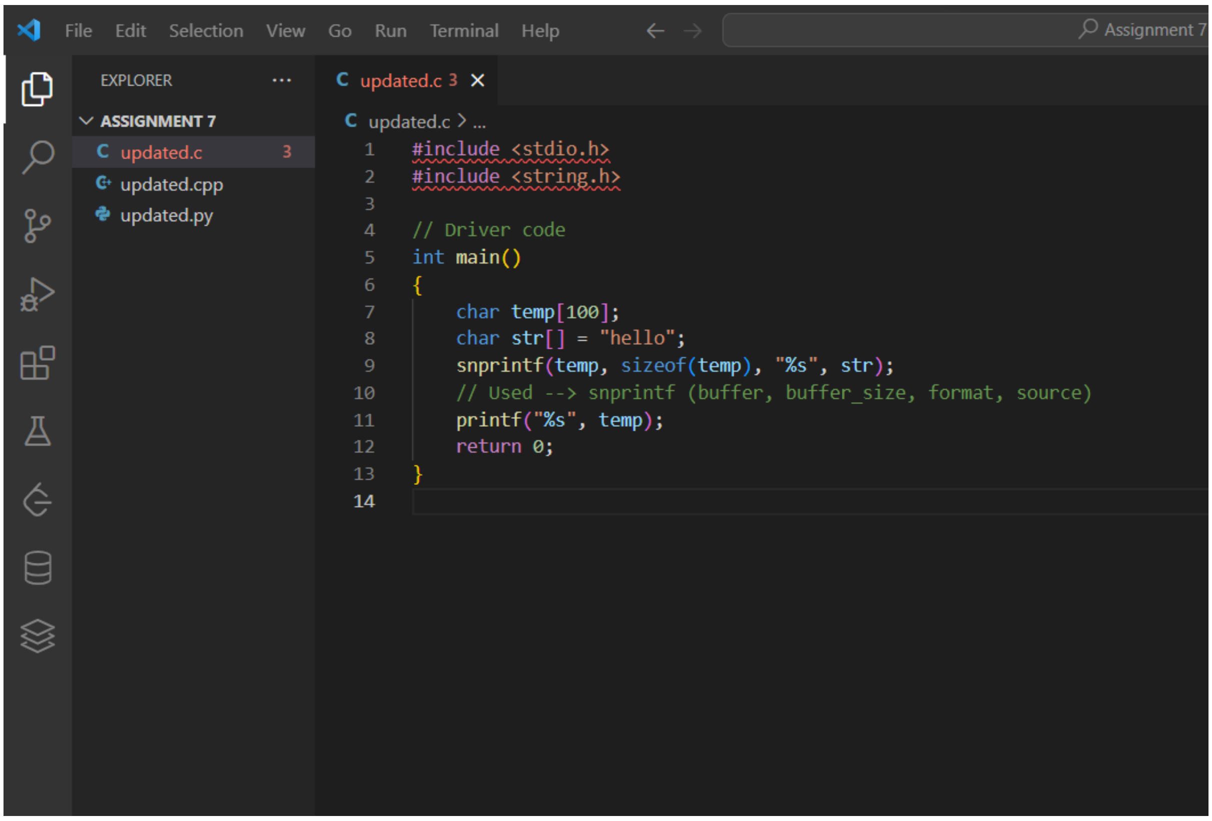
d. Updating the code as per suggestions

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

Ans:

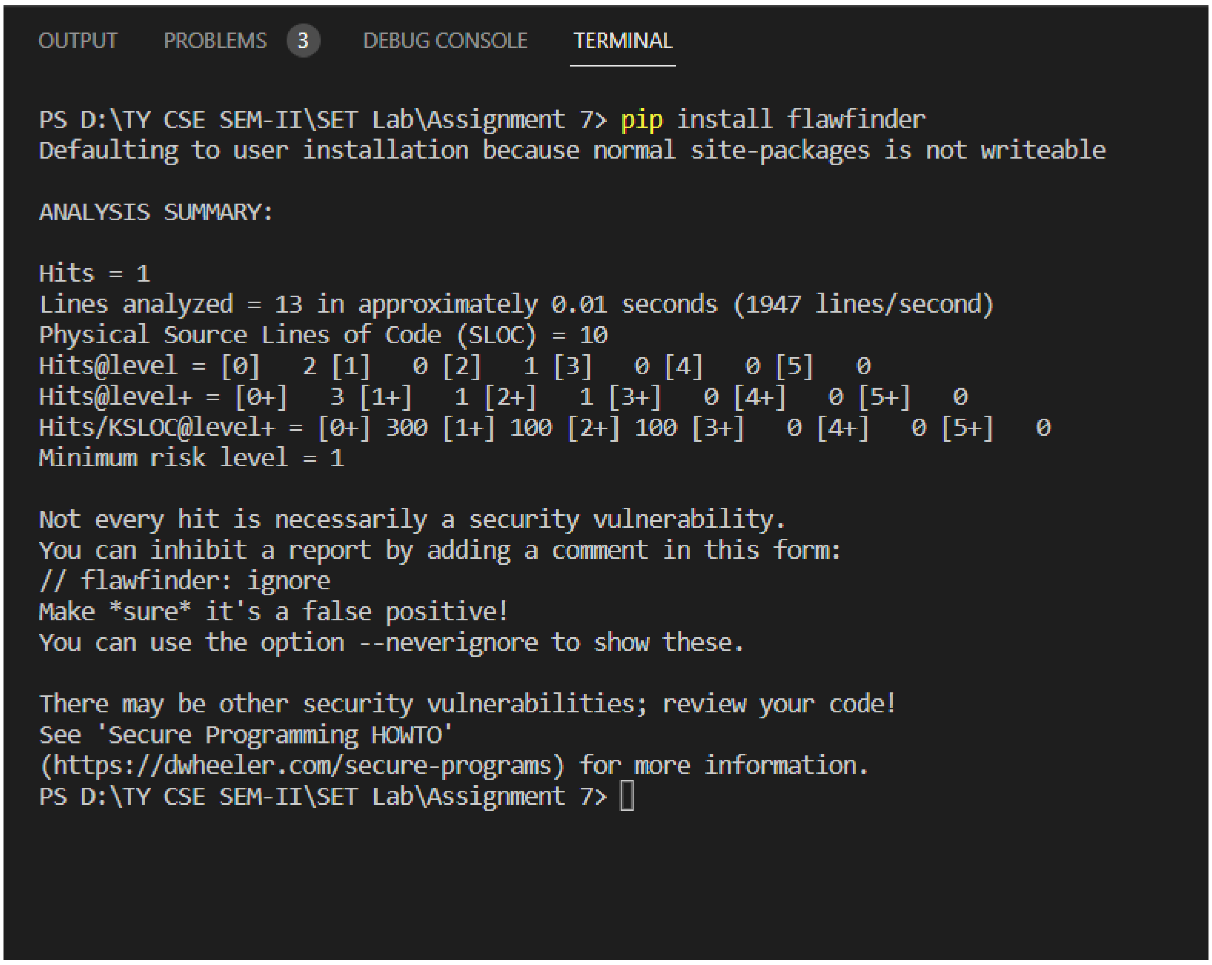
Program 1:

a. Number of hits: 2.
b. Potential risks: 2.
c. Suggested alternatives for these risks:
   i. Buffer overflows: Consider using snprintf, strcpy_s, or strlcpy.
   ii. Statically-sized arrays can be improperly restricted: Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.
d. Updating the code as per suggestions:



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

// Driver code
int main()
{
    char temp[100];
    char str[] = "hello";
    snprintf(temp, sizeof(temp), "%s", str);
    // Used --> snprintf (buffer, buffer_size, format, source)
    printf("%s", temp);
    return 0;
}
```

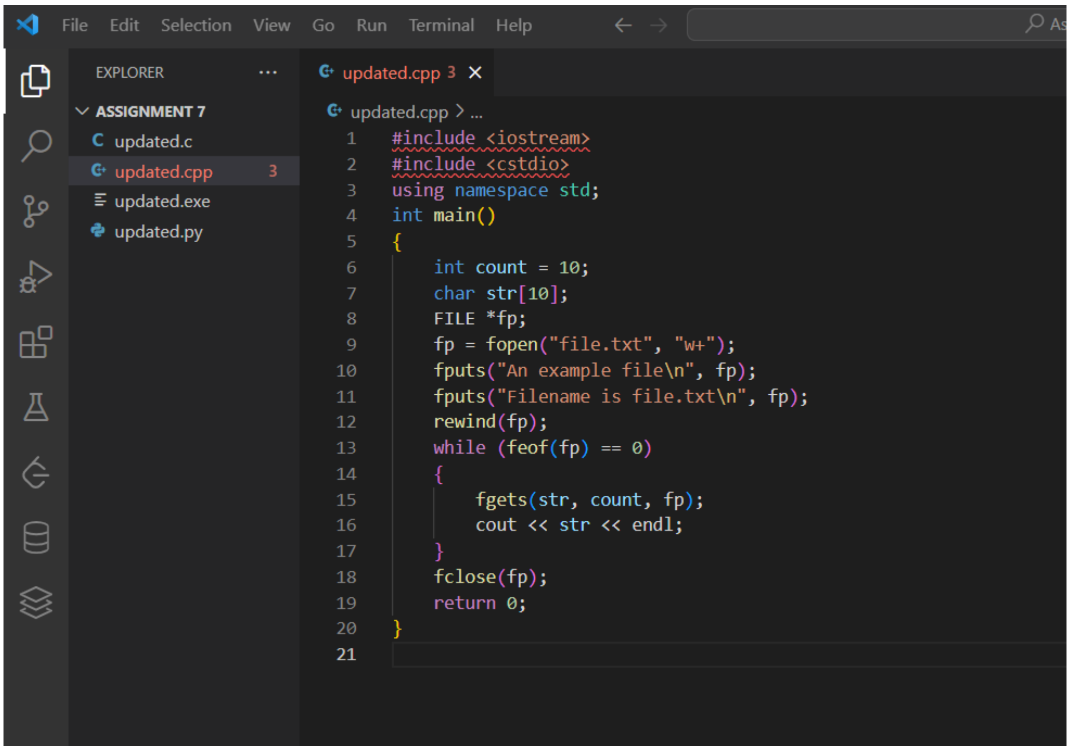
e. Re-execution of code after updating the changes:



```
PS D:\TY CSE SEM-II\SET Lab\Assignment 7> pip install flawfinder
Defaulting to user installation because normal site-packages is not writeable

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 13 in approximately 0.01 seconds (1947 lines/second)
Physical Source Lines of Code (SLOC) = 10
Hits@level = [0]   2 [1]   0 [2]   1 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   3 [1+]   1 [2+]   1 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 300 [1+] 100 [2+] 100 [3+]   0 [4+]   0 [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.
PS D:\TY CSE SEM-II\SET Lab\Assignment 7> 
```

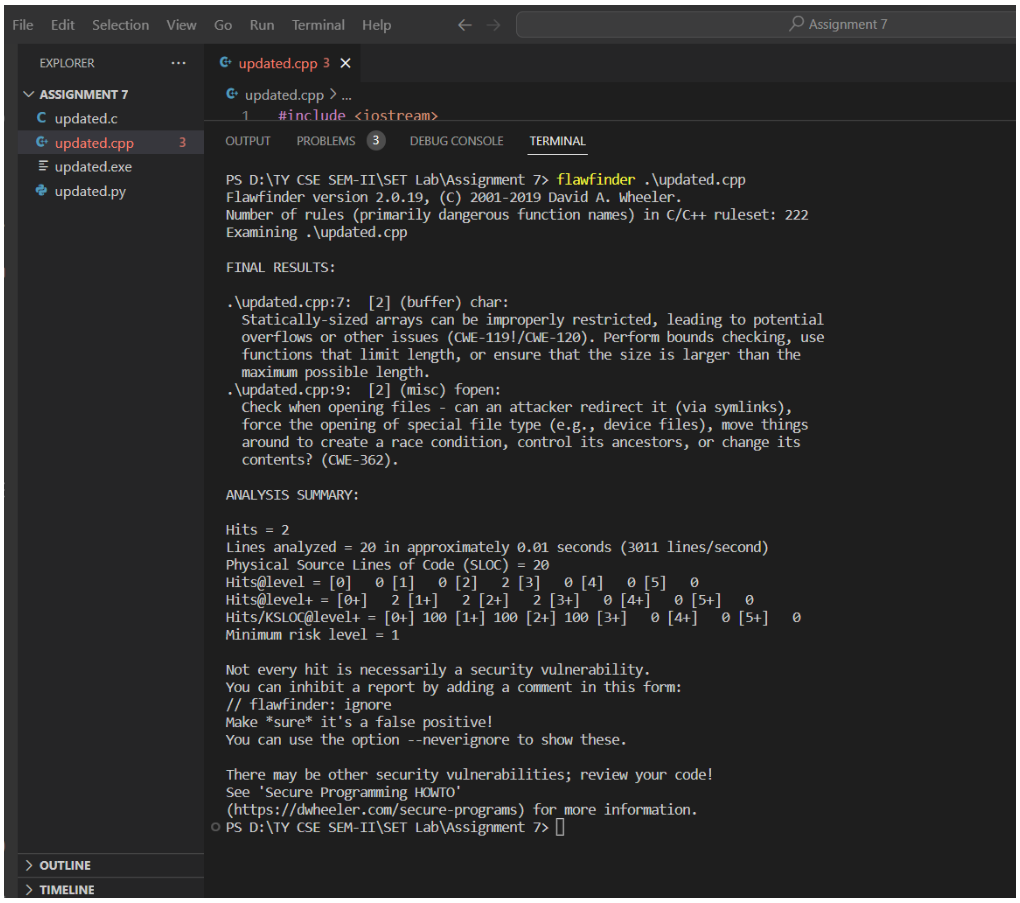
Program 2:

a. **Number of hits: 2.**
b. **Potential risks: 2.**
c. **Suggested alternatives for these risks:**
   i. Statically-sized arrays can be improperly restricted: Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.
   ii. Check when opening files: Can an attacker redirect it (via symlinks), force the opening of special file type (e.g., device files), move things around to create a race condition, control its ancestors, or change its contents?
d. **Updating the code as per suggestions:**



```cpp
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int count = 10;
    char str[10];
    FILE *fp;
    fp = fopen("file.txt", "w+");
    fputs("An example file\n", fp);
    fputs("Filename is file.txt\n", fp);
    rewind(fp);
    while (feof(fp) == 0)
    {
        fgets(str, count, fp);
        cout << str << endl;
    }
    fclose(fp);
    return 0;
}
```

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

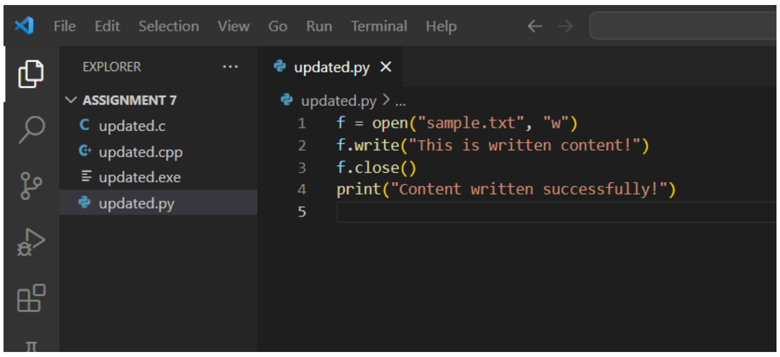


```
PS D:\TY CSE SEM-II\SET Lab\Assignment 7> flawfinder .\updated.cpp
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining .\updated.cpp

FINAL RESULTS:

.\updated.cpp:7:  [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.
.\updated.cpp:9:  [2] (misc) fopen:
  Check when opening files - can an attacker redirect it (via symlinks),
  force the opening of special file type (e.g., device files), move things
  around to create a race condition, control its ancestors, or change its
  contents? (CWE-362).

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 20 in approximately 0.01 seconds (3011 lines/second)
Physical Source Lines of Code (SLOC) = 20
Hits@level = [0]   0 [1]   0 [2]   2 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   2 [1+]   2 [2+]   2 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 100 [1+] 100 [2+] 100 [3+]   0 [4+]   0 [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.
PS D:\TY CSE SEM-II\SET Lab\Assignment 7>
```

Q. 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:

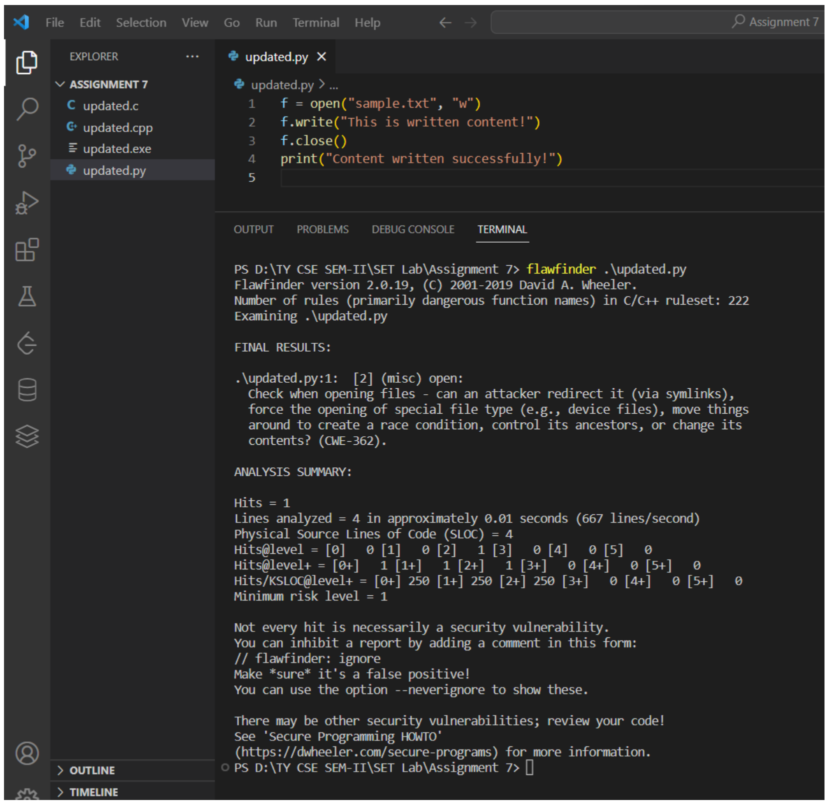
a. **Number of hits**
b. **Potential risks**
c. **Suggested alternatives for these risks**
d. **Updating the code as per suggestions**

Ans.



```python
f = open("sample.txt", "w")
f.write("This is written content!")
f.close()
print("Content written successfully!")
```

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



```python
f = open("sample.txt", "w")
f.write("This is written content!")
f.close()
print("Content written successfully!")
```

```
PS D:\TY CSE SEM-II\SET Lab\Assignment 7> flawfinder .\updated.py
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining .\updated.py

FINAL RESULTS:

.\updated.py:1:  [2] (misc) open:
  Check when opening files - can an attacker redirect it (via symlinks),
  force the opening of special file type (e.g., device files), move things
  around to create a race condition, control its ancestors, or change its
  contents? (CWE-362).

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 4 in approximately 0.01 seconds (667 lines/second)
Physical Source Lines of Code (SLOC) = 4
Hits@level = [0]   0 [1]   0 [2]   1 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   1 [1+]   1 [2+]   1 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 250 [1+] 250 [2+] 250 [3+]   0 [4+]   0 [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.
PS D:\TY CSE SEM-II\SET Lab\Assignment 7> 
```

BandIt Report:

a. **Number of hits: 0.**
b. **Potential risks: 0.**
c. **Suggested alternatives for these risks: N/A**
d. **Updating the code as per suggestions: - N/A**
e. **Re-execution of code after updating the changes: N/A**