Software Engineering Tools Lab

Assignment No-7

PRN : 2019BTECS00090

Name:Vishal Chauhan
Batch : T8

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

Source code analysis is the process of analyzing the source code of software to identify and eliminate any potential security vulnerabilities, bugs, or other defects.

The importance of source code analysis lies in its ability to catch errors early in the development process, which helps to reduce the cost and time associated with fixing problems later on.

By analyzing the code, developers can identify and address any issues before the software is released, improving the overall quality and security of the product.

Q 2. Below are 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:

a. Owner/ Developer: Developed by Joris van de Vis.

b. Developed in which language: Developed in Java.

c. Brief Information/Introduction: VisualCodeGrepper is a static code analysis tool used to detect security vulnerabilities in software code.

d. Language Support: Supports source code written in C/C++, Java, and C#.

e. Advantages: User-friendly interface, easy to configure, supports multiple programming languages.

f. Disadvantages: Limited to certain programming languages, may have false positives.

II. Rips:

a. Owner/ Developer: Developed by RIPS Technologies.

b. Developed in which language: Developed in PHP.

c. Brief Information/Introduction: Rips is a static code analysis tool used to detect security vulnerabilities in PHP applications.

d. Language Support: Supports PHP.

e. Advantages: Accurate and efficient, easy to use, can scan large codebases.

f. Disadvantages: Limited to PHP applications, may have false positives.

III. Brakeman:

a. Owner/ Developer: Developed by Justin Collins.

b. Developed in which language: Developed in Ruby.

c. Brief Information/Introduction: Brakeman is a static code analysis tool used to detect security vulnerabilities in Ruby on Rails applications.

d. Language Support: Supports Ruby on Rails.

e. Advantages: Accurate and efficient, easy to integrate with development workflow, supports continuous scanning.

f. Disadvantages: Limited to Ruby on Rails applications, may have false positives.

IV. Flawfinder:

a. Owner/ Developer: Developed by David A. Wheeler.

b. Developed in which language: Developed in Perl.

c. Brief Information/Introduction: Flawfinder is a static code analysis tool used to detect potential security vulnerabilities in software code.

d. Language Support: Supports C, C++, and other languages with similar syntax.

e. Advantages: Lightweight and easy to use, can identify common coding mistakes, can be customized with user-defined rules.

f. Disadvantages: May generate false positives, may miss some vulnerabilities.

V. Bandit:

a. Owner/ Developer: Developed by PyCQA.

b. Developed in which language: Developed in Python.

c. Brief Information/Introduction: Bandit is a static code analysis tool used to detect security vulnerabilities in Python applications.

d. Language Support: Supports Python.

e. Advantages: Easy to install and use, integrates with popular Python testing frameworks, can detect common security issues.

f. Disadvantages: May generate false positives, limited to Python applications.

Q 3. Perform source code testing using Flawfinder for the code written in ‘c’ and ‘cpp’ language given below

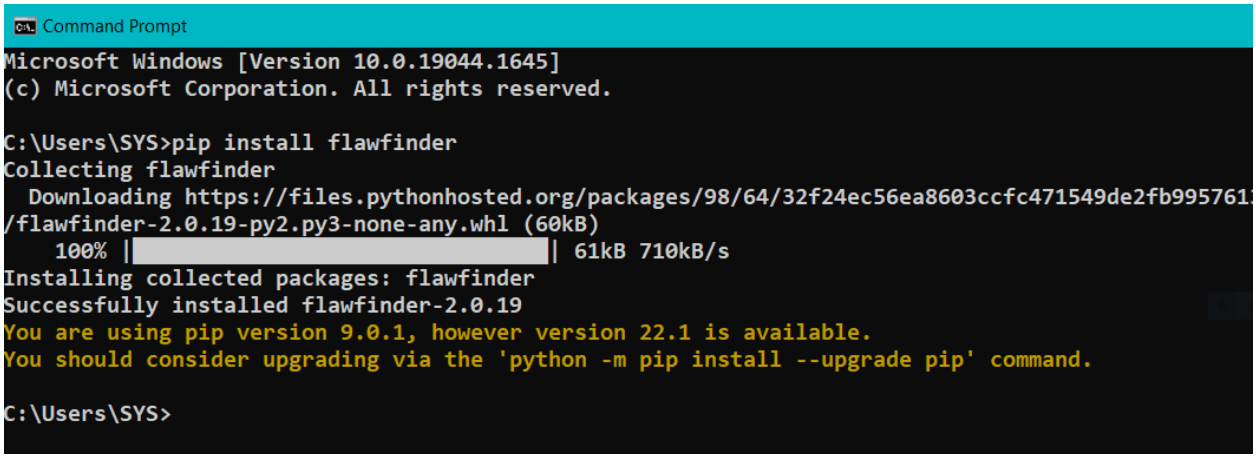
Link-https://github.com/sidp1991/SETAssignment

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.

Installing Flawfinder:



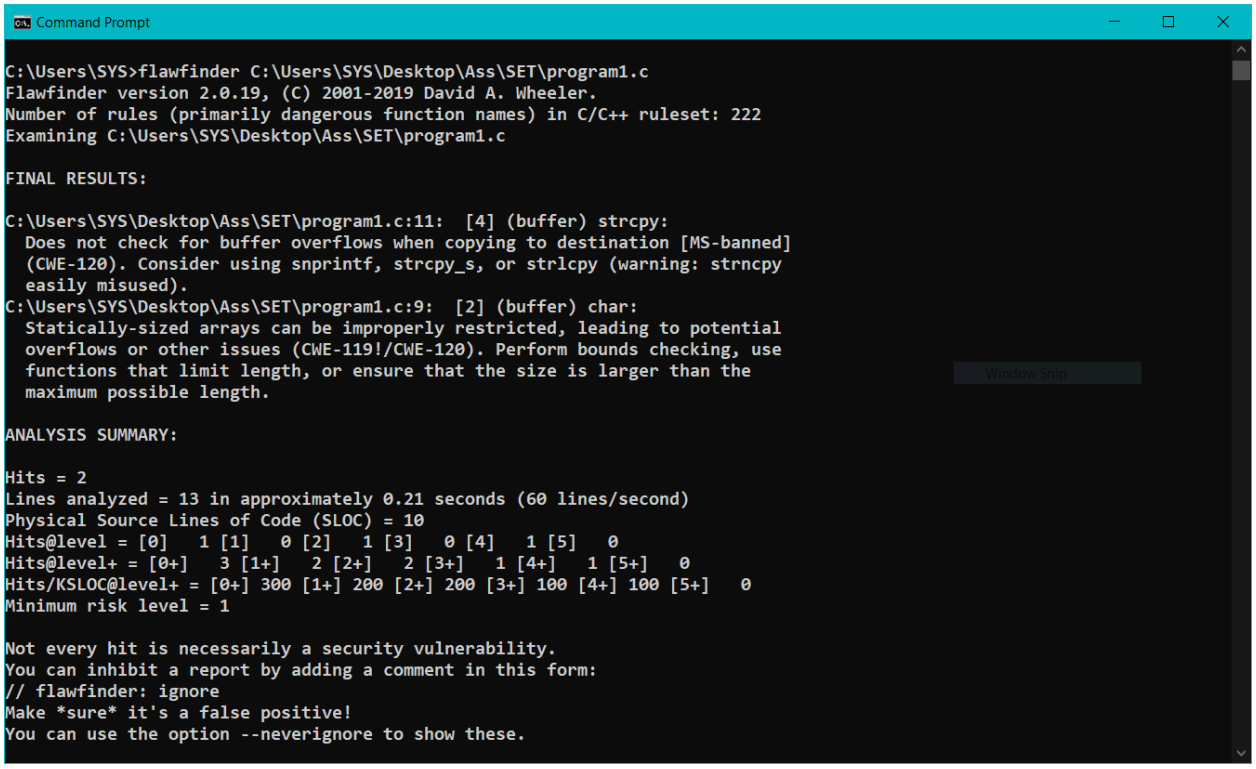
Program1.c file:

a. Number of hits: 2

b. Potential risks:

1. First one is use of strcpy function. It does not check for buffer overflows when copying to destination.

2. Another vulnerability is the use of a char array. Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues. Instead, functions can be used to check the limit length and ensure that size is larger than the maximum possible length.



c. Suggested alternatives for these risks:

1. It suggests to use snprintf, strcpy_s, or strlcpy.
2. Perform bounds checking, usefunctions that limit length, or ensure that the size is larger than themaximum possible length.

d. Updating the code:

```
// C program to demonstrate
// Flawfinder
#include <stdio.h>
#include <string.h>

// Driver code
int main()
{
char temp[];
   char str[] = "hello";
```

```
strcpy_s(temp, str);

printf("%s", temp);

   return 0;

         }
```

e. Re-execution of code after updating:

No hits found.

```
C:\Users\SYS>flawfinder C:\Users\SYS\Desktop\Ass\SET\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 C:\Users\SYS\Desktop\Ass\SET\program1.c

FINAL RESULTS:


ANALYSIS SUMMARY:

No hits found.
Lines analyzed = 14 in approximately 0.03 seconds (469 lines/second)
Physical Source Lines of Code (SLOC) = 10
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+] 100 [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\SYS>flawfinder C:\Users\SYS\Desktop\Ass\SET\program1.c
```

Program2.cpp

a. Number of hits: 2

b. Potential risks:

1. Statically-sized arrays can be improperly restricted, leading to potentialoverflows or other issues.
2. Check when opening files - can an attacker redirect it (via symlinks),force the opening of special file type (e.g., device files), move thingsaround to create a race condition, control its ancestors, or change itscontents?

```
C:\Users\SYS>flawfinder C:\Users\SYS\Desktop\Ass\SET\program2.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 C:\Users\SYS\Desktop\Ass\SET\program2.cpp

FINAL RESULTS:

C:\Users\SYS\Desktop\Ass\SET\program2.cpp: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.
C:\Users\SYS\Desktop\Ass\SET\program2.cpp:12:  [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 = 25 in approximately 0.04 seconds (597 lines/second)
Physical Source Lines of Code (SLOC) = 19
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+] 105.263 [1+] 105.263 [2+] 105.263 [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.
```

c. Suggested alternatives for these risks:

- Perform bounds checking, usefunctions that limit length, or ensure that the size is larger than themaximum possible length.

d. Updating the code:

```
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
   int count = 10;
   char str[];
   FILE *fp;

fp = tmpfile();
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:

```
C:\Users\SYS\Desktop\Ass\SET\program2.cpp:11:  [2] (tmpfile) tmpfile:
  Function tmpfile() has a security flaw on some systems (e.g., older System
  V systems) (CWE-377).

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 24 in approximately 0.05 seconds (508 lines/second)
Physical Source Lines of Code (SLOC) = 19
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+] 52.6316 [1+] 52.6316 [2+] 52.6316 [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.
```

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

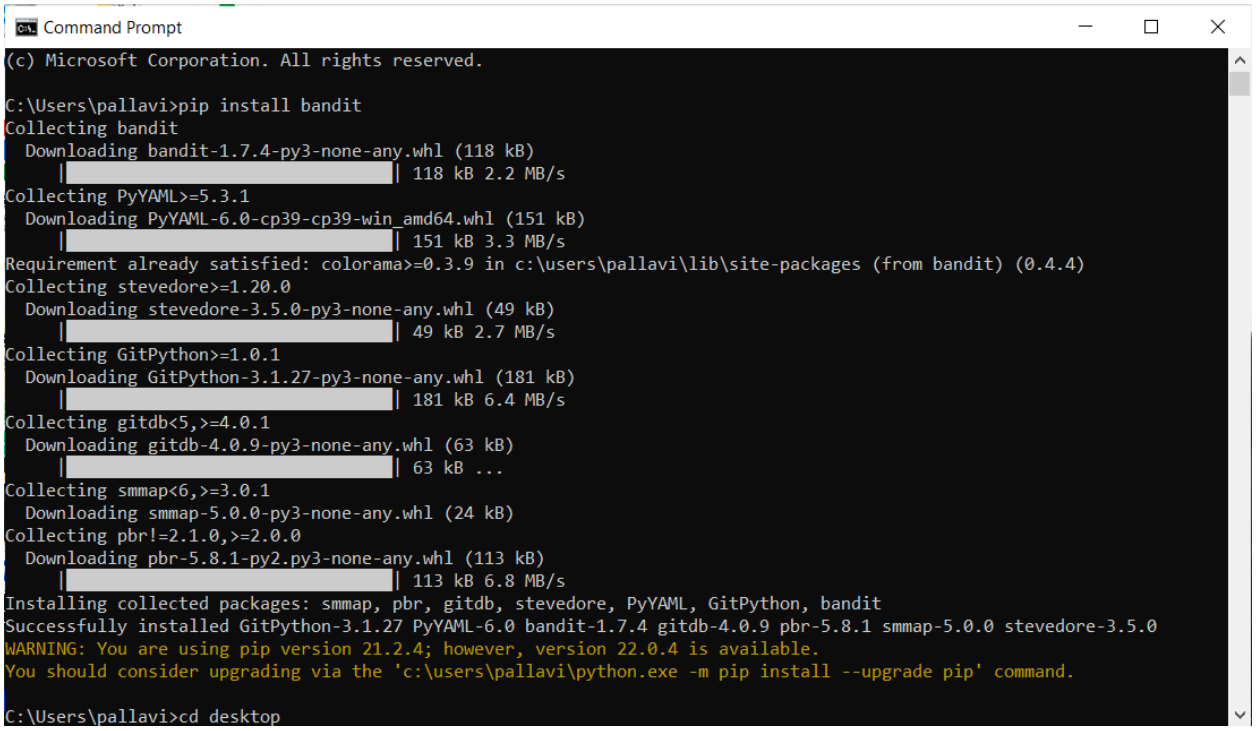
b. Potential risks

c. Suggested alternatives for these risks

d. Updating the code as per suggestions

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

Install bandit



a. Number of hits: 2

b. Potential risks:

Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimized byte code

```
Test results:
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   Location: exception.py:11:8
   More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b101_assert_used.html
10          try:
11              assert (temp1 >= 0),"Value less than absolut temperature"
12              #if temp1 >= 0:
13              #    print("Value less than absolut temperature")
14              temp2 = ((temp1 - 273) * 1.8 ) + 32

--------------------------------------------------
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   Location: exception.py:29:8
   More Info: https://bandit.readthedocs.io/en/1.7.4/plugins/b101_assert_used.html
28          try:
29              assert (temp1 >= 0),"Value less than absolut temperature"
30              #if temp1 >= 0:
31              #    print("Value less than absolut temperature")
32              temp2 = temp1 - 273.15
```

```
Code scanned:
        Total lines of code: 55
        Total lines skipped (#nosec): 0

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

c. Suggested alternatives for these risks:

Use if statements instead of assert.

d. Updating the code as per suggestions

```
def Kelvin_to_Fahrenheit(temp1):
    try:
        #assert (temp1 >= 0),"Value less than absolut temperature"
        if temp1 >= 0:
            print("Value less than absolut temperature")
        temp2 = ((temp1 - 273) * 1.8 ) + 32
        print("\nTemperature in Kelvin:",temp1)
        print("\nTemperature in Fahrenheit:",temp2)
```

```
def Kelvin_to_Celcius(temp1):
    try:
        #assert (temp1 >= 0),"Value less than absolut temperature"
        if temp1 >= 0:
            print("Value less than absolut temperature")
        temp2 = temp1 - 273.15
        print("\nTemperature in Kelvin:",temp1)
        print("\nTemperature in Celcius:",temp2)
```

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

```
C:\Users\pallavi\Desktop\Test>bandit exception.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 3.9.0
[node_visitor]  WARNING Unable to find qualified name for module: exception.py
Run started:2022-05-03 14:50:11.650771

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 57
        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):
```