**Software Engineering Tools Lab**

**Assignment No-7**

(Module 5- Source code testing using tools)

**PRN: 2019BTECS00017**

**Full name: Muskan Raju Attar**

**Batch: T1**

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

Ans:

**Source Code Analysis:**

Source code analysis is basically automated code debugging. The aim is to find bugs and faults that may not be obvious to a programmer. 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.

**Importance of Source Code Analysis:**

1. Code analyzers work using rules that tell it what to look for. With too little precision, an analyzer might spew out too many false positives and flood the user with useless warnings, while too much precision might take too long to finish; therefore, has to be a balance.
2. **Scans Code Fragments and Non-Compiling Code**

An SCA tool is capable of scanning code fragments, regardless of compilation errors arising from syntactic or other errors. Both auditors and developers can scan incomplete code in the midst of the development process without having to achieve a build, ultimately allowing the discovery of vulnerabilities much earlier during the Software development Life Cycle (SDLC).

1. **Supports Cloud Compiled Language**

New coding language breeds have developed under cloud computing scenarios. In these cases, the developer codes in the PaaS-provider’s language, while the PaaS-provider is responsible for the validation, proprietary compilation and execution of the programs. In these cases, the code has no manifestation as byte-code nor as binary, and the SCA must be done on the source code itself. The most known example is the Force.com platform supplied by Salesforce.com. This platform is based on the server-based language called Apex, and client-based language called VisualForce. Only an SCA product can support this new paradigm.

**4) Compiler Agnostic**

In a multi-compiler environment- typically found at code auditors and large corporations- the SCA standard provides a one solution fits all. This is starkly opposed to the BCA which must support an endless number of compilers and versions. The reason? Each compiler transforms source code into its own version of binary/ byte code forcing the BCA tool to read, understand and analyze the different outputs of different compilers. However, since an SCA tool runs on the code itself – and not post-compilation, the SCA provides a single standard irrelevant to the compiler version or compiler upgrades.

**5) Assesses Security of Non Linking Code**

In the case where the code references infrastructure libraries for which their source is missing, the BCA tools immediately fails on the unfortunate “Missing Library” message. Days may be spent building stubs for these missing parts, just to make the code compile – a lot of hard work without any added value.

An SCA product easily identifies vulnerabilities, such as SQL Injection – even when the actual library code of the executing SQL function call is missing.

**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**

1. **Owner/ developer**
2. **Developed in which language**
3. **Brief information/introduction**
4. **Language support (applicable for source code written in language)**
5. **Advantages**
6. **Disadvantages**

**Source code analysis tools-**

1. **VisualCodeGrepper**

**a. Owner/ developer:** Microsoft

**b. Developed in which language:**

**c. Brief information/introduction:** The Grepper VSCode Add-On brings all the core Grepper functionality right to VSCode, if you are part of the Grepper community and use VSCode, this Add-On is a must. Grepper is the query and answer system for the developer community**.**

**Features:**

* Search Grepper code answers directly from VSCode
* Copy code answers quickly by pressing c
* Add code answers by highlighting code, right clicking and "Add Grepper Answer"
* View and edit your Grepper code answers
* View your Grepper profile, recent answers and more

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

C/C++, Java, C#, VB and PL/SQL

**e. Advantages:**

1. 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.  
2. It attempts to find a range of around 20 phrases within comments that can indicate broken code ("ToDo", "FixMe", "Kludge", etc.)  
3. It provides a nice pie chart (for the entire codebase and for individual files) showing relative proportions of code, whitespace, comments, "ToDo" style comments and bad code.  
It also searches intelligently to identify buffer overflows and signed/unsigned comparisons.

1. **Rips**

**a. Owner/ developer:** RIPS Technologies

**b. Developed in which language:**

**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. The initial tool was written by Johannes Dahse and released during the Month of PHP Security[1] in May 2010 as open-source software.

**Open-Source Version (PHP)**

The open-source version tokenizes PHP code (lexical analysis) based on PHP's tokenizer extension and performs semantic analysis to build a program model. Based on previously analyzed variable assignments, it performs backwards-directed inter-procedural taint analysis of sensitive sinks. Its strength is the ability to scan PHP applications very fast for PHP-specific vulnerabilities. It supports the detection of 15 different vulnerability types, including Cross-Site Scripting, SQL Injection, Local File Inclusion, and others. Detected vulnerabilities are presented in a web interface with the minimum set of affected code lines as well as a vulnerability summary.

**Commercial Version (Java, PHP)**

The commercial version supports the analysis of PHP and Java code. It was built from scratch and leverages new code analysis techniques which are specifically tailored to the intricacies of each programming language and its features. It uses abstract syntax trees, control-flow graphs, and context-sensitive taint analysis in order to accurately identify even complex security vulnerabilities that base on second-order data flow or misplaced security mechanisms.[7] Further, it simulates each language's built-in features, libraries and frameworks to minimize false positives. It supports the automated detection of over 200 different vulnerability types, code quality issues and misconfiguration weaknesses.

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

Java, PHP, C, C++, C#, Python, Cobol

**e. Advantages**

* Automatically detect vulnerabilities in PHP applications
* Offers an integrated code audit framework
* Fast static code analysis

1. **Brakeman**

**a. Owner/ developer:** Justin Collins, Jim Manico, Neil Matatall

**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. Brakeman requires zero setup or configuration once it is installed. Just run it.
2. 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. 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. Brakeman is specifically built for Ruby on Rails applications, so it can easily check configuration settings for best practices.
5. Each check performed by Brakeman is independent, so testing can be limited to a subset of all the checks Brakeman comes with.
6. 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. 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. 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. 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. 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.
5. **Flawfinder**

**a. Owner/ developer:** David Wheeler

**b. Developed in which language:** Python

**c. Brief information/introduction:**

Flawfinder, a simple program that examines C/C++ source code and reports possible security weaknesses (“flaws”) sorted by risk level. It’s very useful for quickly finding and removing at least some potential security problems before a program is widely released to the public. 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. Flawfinder works by using a built-in database of C/C++ functions with well-known problems. You don’t have to create this database - it comes with the tool.
2. Flawfinder also knows about gettext and will treat constant strings passed through gettext as though they were constant strings; this reduces the number of false hits in internationalized programs.
3. Flawfinder may be able to determine that the construct isn’t risky at all, reducing false positives.
4. It knows to ignore comments and the insides of strings, and it will also examine parameters to estimate risk levels.

**f. Disadvantages**

1. Flawfinder is fundamentally a naive program.
2. It doesn’t even know about the data types of function parameters.
3. It certainly doesn’t do control flow or data flow analysis.
4. One should analyze a copy of the source code he/she is evaluating.
5. **Bandit**

**a. Owner/ developer:** PyCQA

**b. Developed in which language:** Python

**c. Brief information/introduction:**

Bandit is commonly used for code analysis. Bandit is a tool that can be used during development or afterward. Typically this is used by developers to find common security issues in Python code before putting the code in production. Another use-case would be to use this tool to analyze existing projects and find possible flaws. Target users for this tool are developers, pen testers, and security professionals.

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**

* More than 50 contributors
* The source code of this software is available
* Supported by a large company

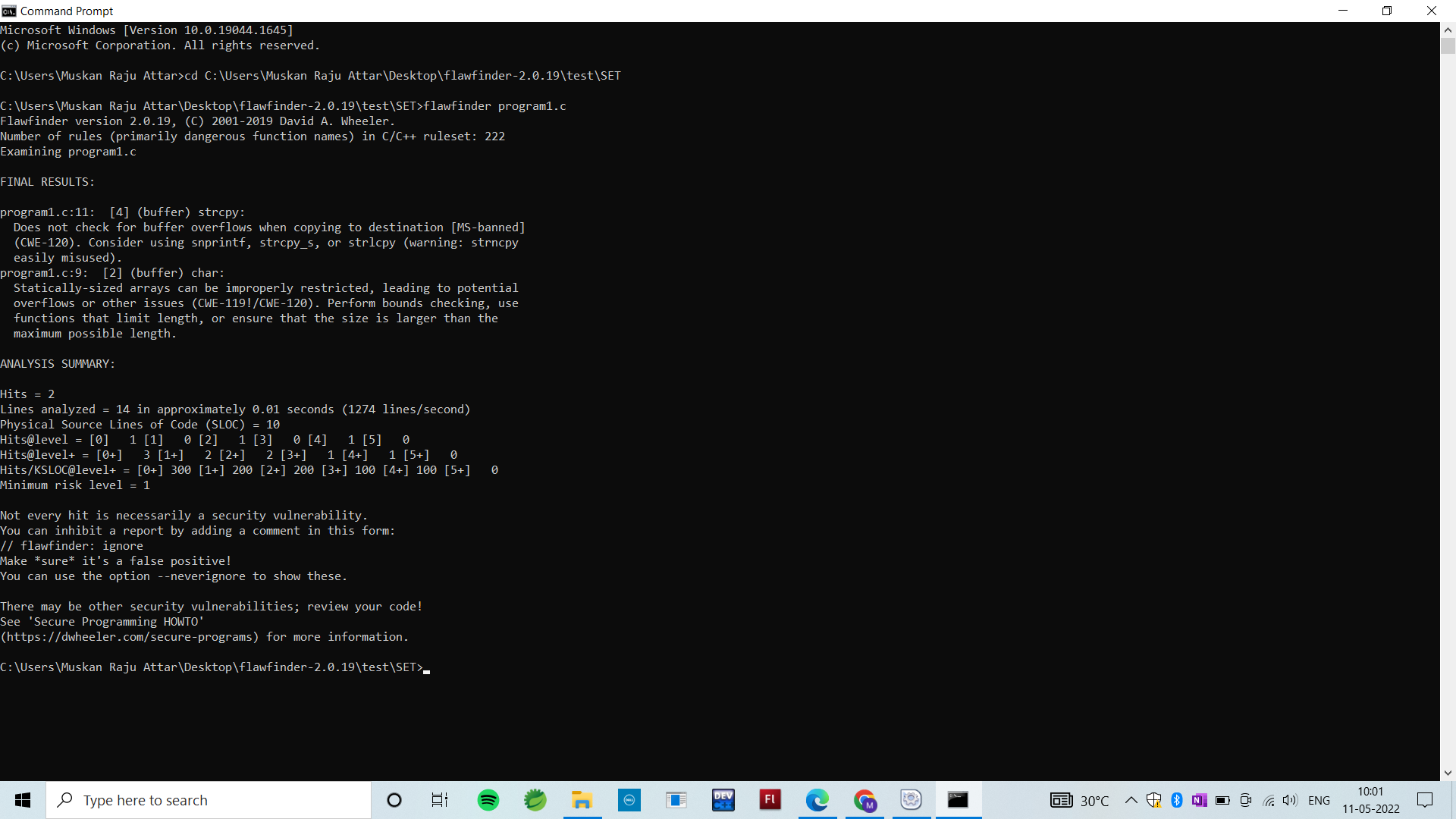
Q 3. Perform source code testing using **Flawfinder** for the code written in ‘c’ and ‘cpp’ language given below

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

Program1.c:



1. Number of hits:2
2. 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.

1. Suggested alternatives for these risks

1)It suggests to use snprintf, strcpy\_s, or strlcpy.

2)Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.

1. Updating the code as per suggestions

// C program to demonstrate

// Flawfinder

#include <stdio.h>

#include <string.h>

// Driver code

int main()

{

**char temp[];**

char str[] = "hello";

**strcpy\_s(temp, str);**

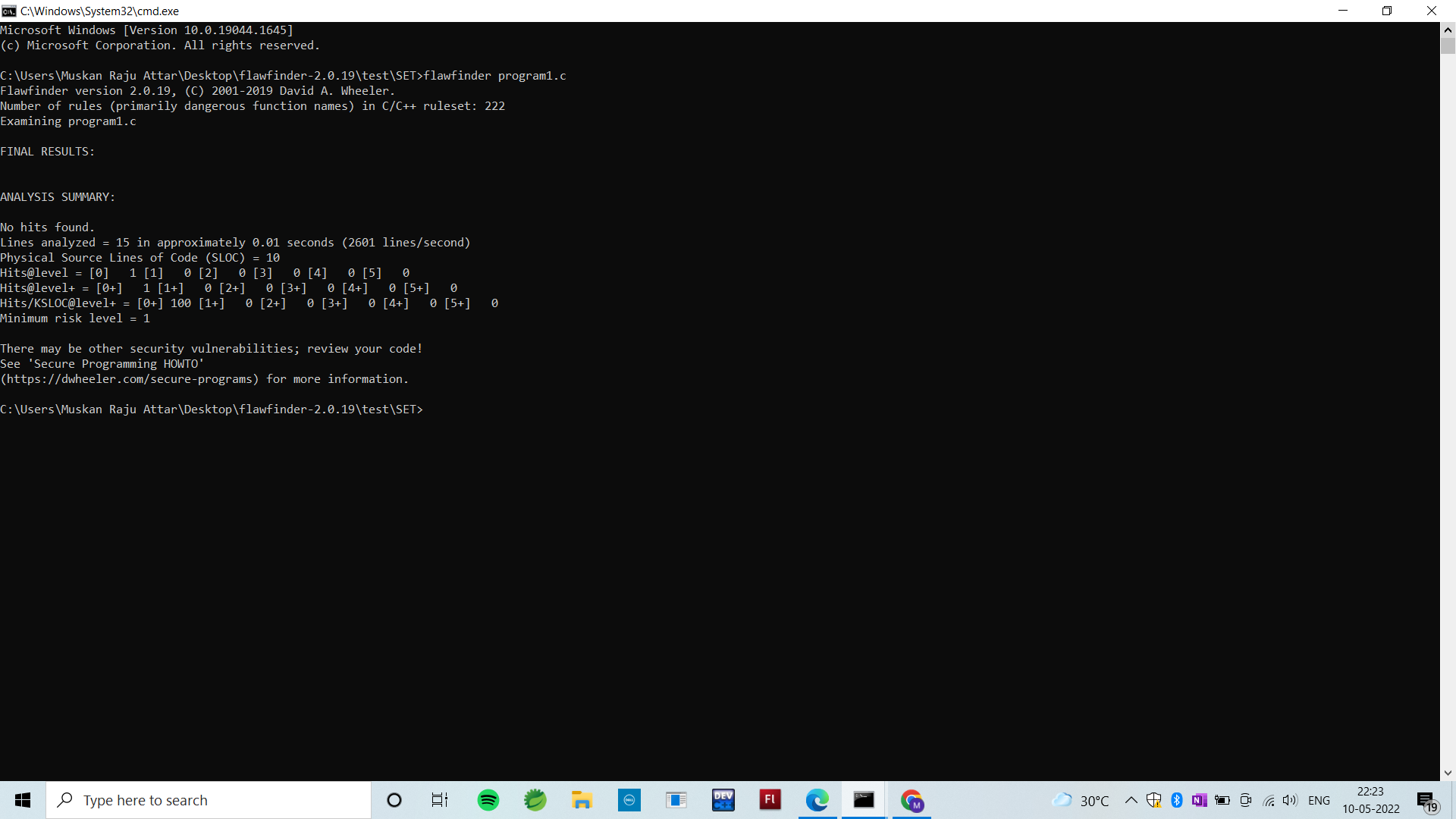
printf("%s", temp);

return 0;

}

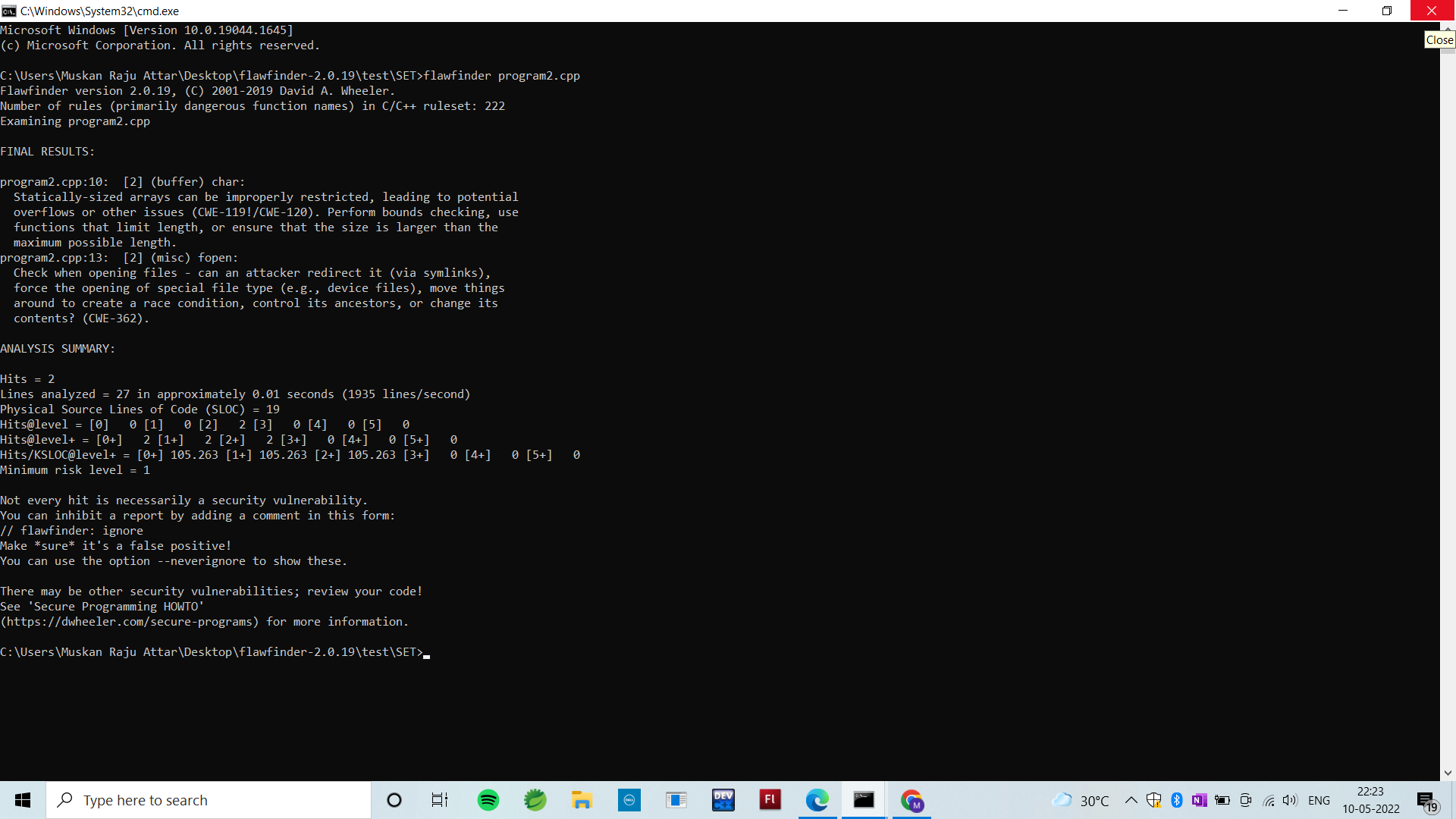
1. Re-execution of code after updating the changes.

Hits:0



Program2.cpp

1. Number of hits:2



1. Potential risks:

1)Statically-sized arrays can be improperly restricted, leading to potential overflows or around to create a race condition, control its ancestors, or change its contents?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 things

1. Suggested alternatives for these risks

Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.

1. Updating the code as per suggestions

#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);

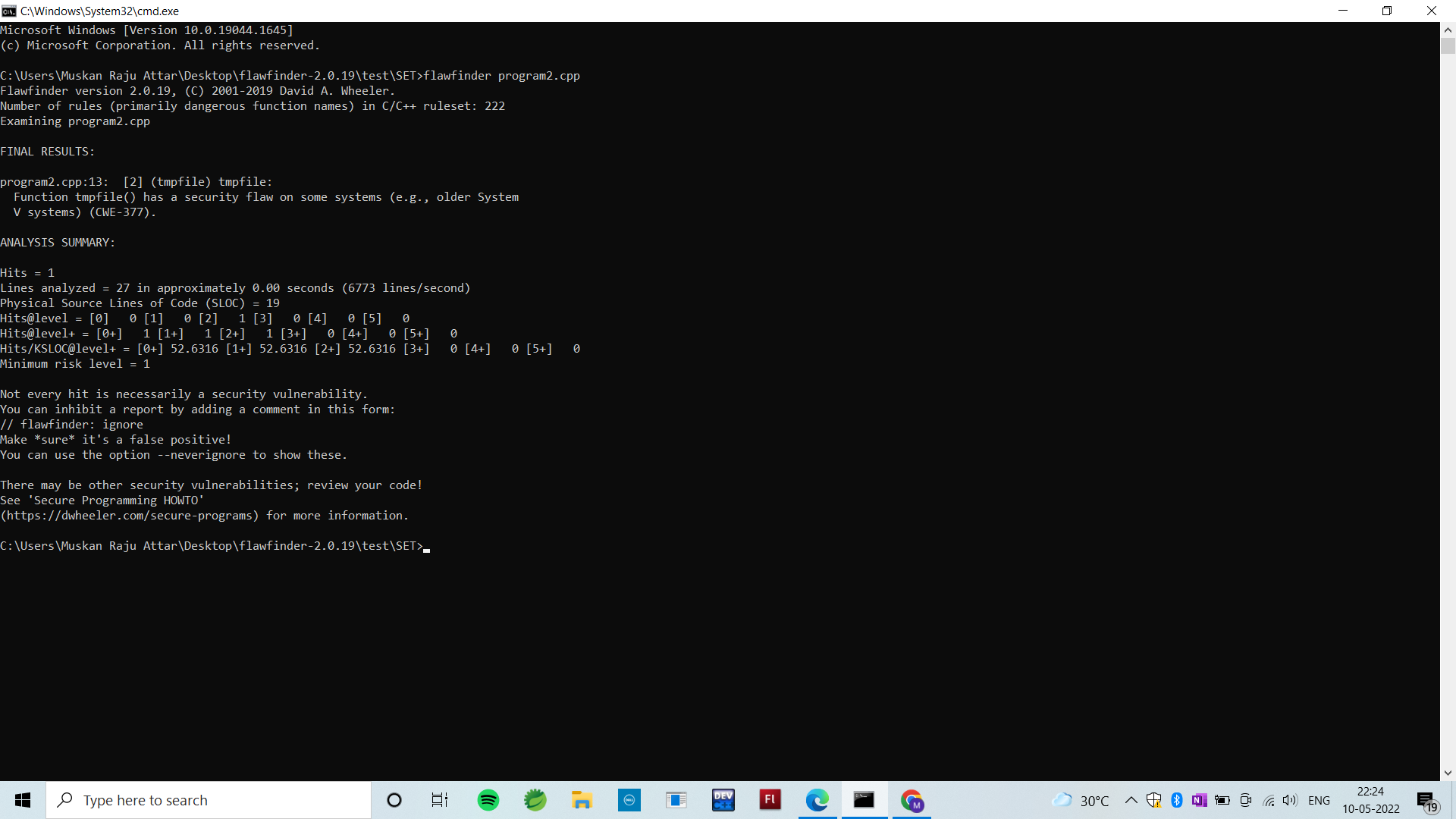
cout << str << endl;

}

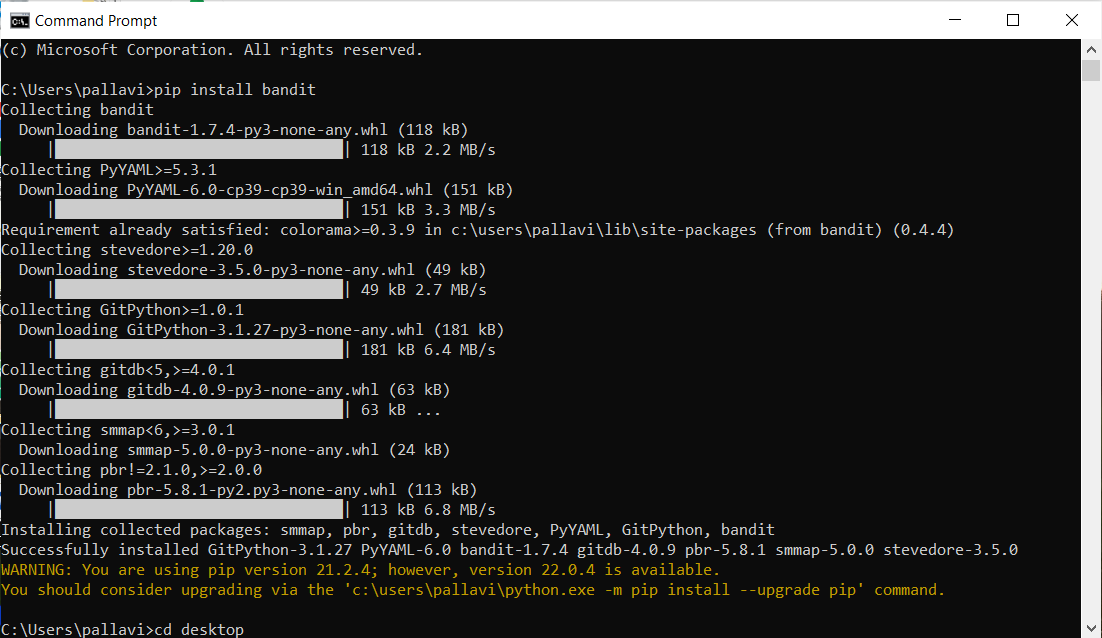
fclose(fp);

return 0;

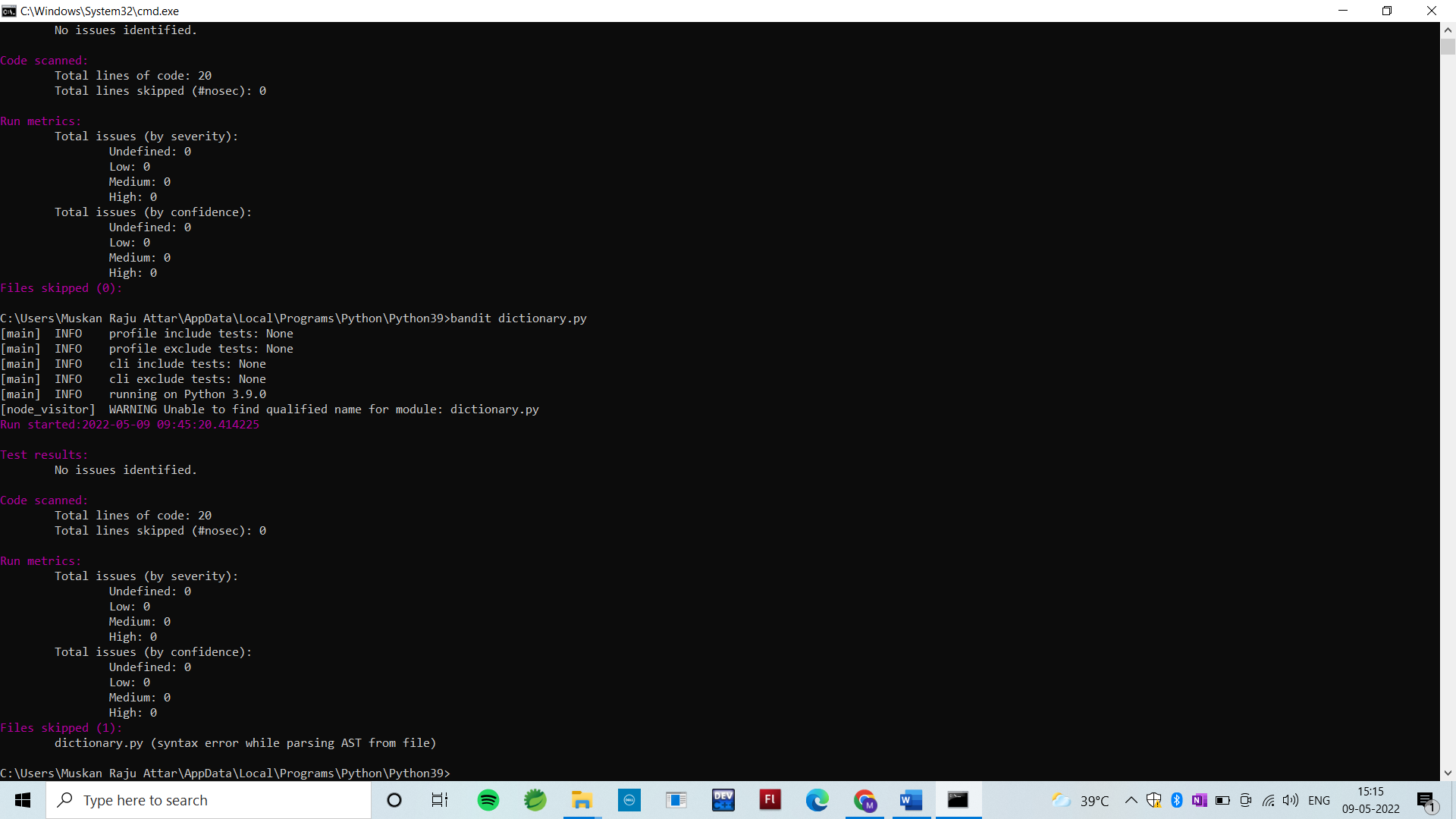
1. Re-execution of code after updating the changes.



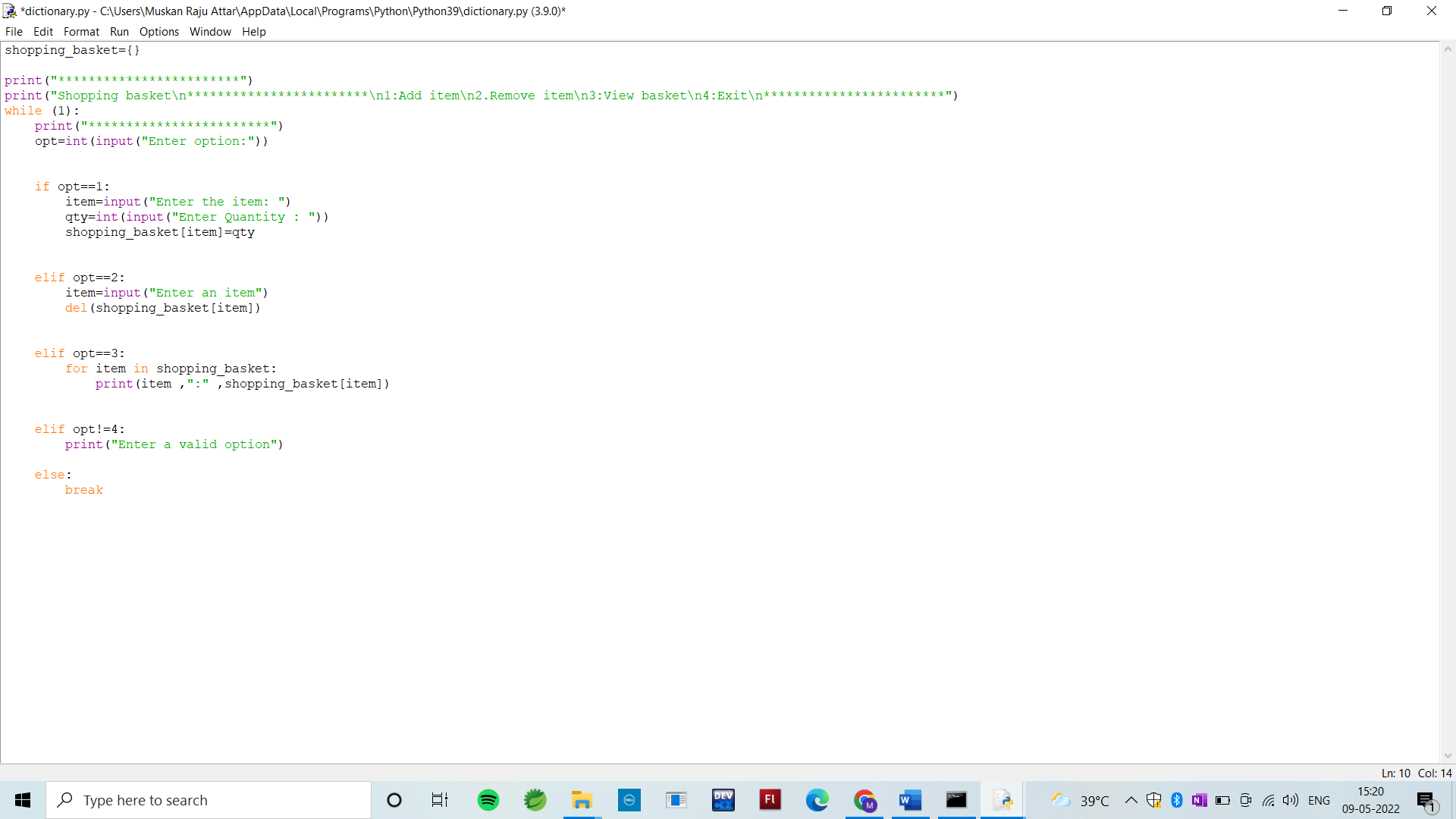
Q 4. Perform source code testing using **Bandit** for your code written in ‘python’ language (use your previous code) for any security flaws

****

After performing analysis create a report which will contain below points



1. Number of hits:0
2. Potential risks:
3. Suggested alternatives for these risks: Syntax error while passing AST from file
4. Updating the code as per suggestions



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

