

Practical 1 Static code analysis using an open source tool; Flawfinder.

Author: Sagar Sanjay Sutar

Course: MSc CS

Roll no: F010

Date: 10th July 2025

Table of Contents

[Write Up](#)

[Static Code Analysis & Benefits](#)

[Vulnerability](#)

[Flawfinder](#)

[Running Flawfinder on a code with no error](#)

[Running Flawfinder on a code with an error](#)

Write Up

Static Code Analysis & Benefits

- A static code analyzer is a tool to develop secure software. It helps in finding common vulnerabilities.
- One of the best parts is it analyses the code without executing it, enabling the user to locally test code which he may not be able to compile or run on his system.
- Furthermore, it offers early error detection which saves development cost.

Note:

- Since **no** one tool is perfect, utilizing as many as possible is best practice.
- A code analyser is not a substitute for manual code audit. Before using any tools, one must ensure to not repeat common mistakes made by past developers.

Vulnerability

A software can be vulnerable to following threats

- *Threats to Confidentiality*: Man-in-middle attack threatens the confidentiality of the app. The attacker is allowed to intercept the communication b/w the client & the server.
- *Threats to Availability*: DDOS attack to an app or its services threatens the availability of the app or its core service to legitimate users.

- *Threats to Integrity*: SQL injection to an app threatens the integrity of the app as malicious code is allowed to enter the app.

Flawfinder

- Flawfinder is a static code analyser tool that parses source code to find well-known security vulnerabilities in C++.
- It does not understand the code but merely does a string comparison against its own database containing well-known vulnerabilities.
- It sorts the vulnerabilities by the risky levels.

Running Flawfinder on a code with no error

1. The below code finds out the smallest number in an array/vector.

```

Minimum-Num.cpp > main()
Sagar, 20 months ago | 1 author (Sagar)
1  /**
2   * @brief An example of Linear algorithm O(N) as we have 1 loop that iterates through all elements to find the minimum number.
3   * The running time of this program will increase with size of input. *
4   */
5  #include <iostream>
6  #include <vector>
7
8  using namespace std;
9
10 int main()
11 {
12     vector<int> numVec = {45, 43, 84, 33, 11, 9, 84, 3};
13     int minNum;
14     for(int num = 0; num < numVec.size(); num++){
15
16         // Check if last element is the smallest number.
17         if(num == numVec.size() - 1 && numVec.at(numVec.size() - 1) < minNum){
18             minNum = numVec.at(numVec.size() - 1);
19             break;
20         }
21
22         // Check if rest of the elements are smaller as compared to the following elements.
23         if(numVec.at(num) < numVec.at(num + 1)){
24             minNum = numVec.at(num);
25         }
26     }
27
28     cout << "Minimum Number: \t" << minNum << endl;
29     return 0;
30 }
Sagar, 20 months ago * Added Linear & Quadratic code snippets
31

```

2. Running flawfinder on the above code gives no hits i.e. no possible vulnerabilities.

```

ssutar@ENG-47-LX:~/Tutorials/Programming-Exercises/Data-Structure-Exercises$ python /home/ssutar/Downloads/flawfinder-master/flawfinder.py Minimum-Num.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 Minimum-Num.cpp

FINAL RESULTS:

ANALYSIS SUMMARY:

No hits found.
Lines analyzed = 30 in approximately 0.00 seconds (11900 lines/second)
Physical Source Lines of Code (SLOC) = 19
Hits@level = [0]  0 [1]  0 [2]  0 [3]  0 [4]  0 [5]  0
Hits@level+ = [0+] 0 [1+] 0 [2+] 0 [3+] 0 [4+] 0 [5+] 0
Hits/KSLOC@level+ = [0+] 0 [1+] 0 [2+] 0 [3+] 0 [4+] 0 [5+] 0
Minimum risk level = 1

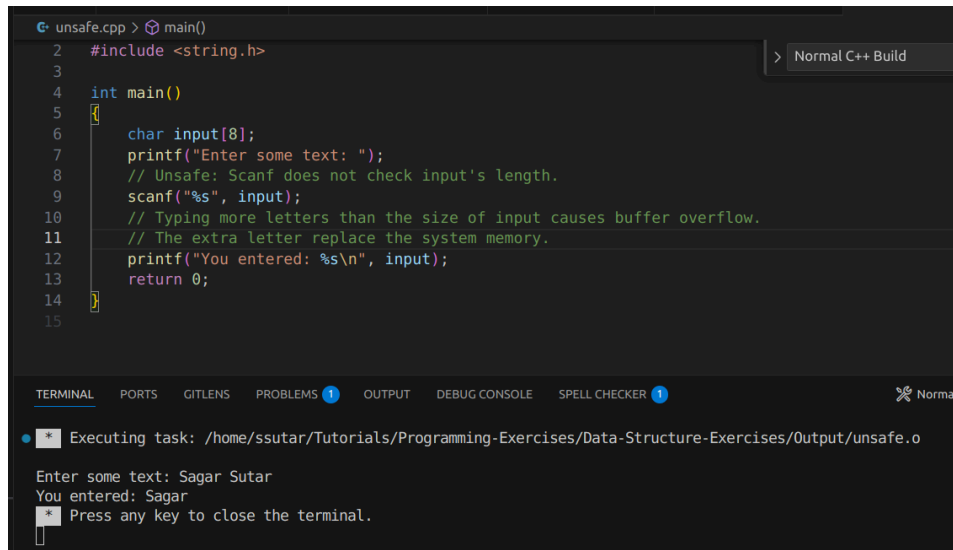
There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.
ssutar@ENG-47-LX:~/Tutorials/Programming-Exercises/Data-Structure-Exercises$

```

Running Flawfinder on a code with an error

Buffer Overflow

Here `scanf` function is used which is an unsafe function that does not check the size of variable in which it is inserting the user value. This could cause buffer overflow. In the output terminal below, entering "Sagar Sutar" results in "Sagar" only. Rest of the characters are in vain.



```
unsafe.cpp > main()
2  #include <string.h>
3
4  int main()
5  {
6      char input[8];
7      printf("Enter some text: ");
8      // Unsafe: Scanf does not check input's length.
9      scanf("%s", input);
10     // Typing more letters than the size of input causes buffer overflow.
11     // The extra letter replace the system memory.
12     printf("You entered: %s\n", input);
13     return 0;
14 }
15
```

Normal C++ Build

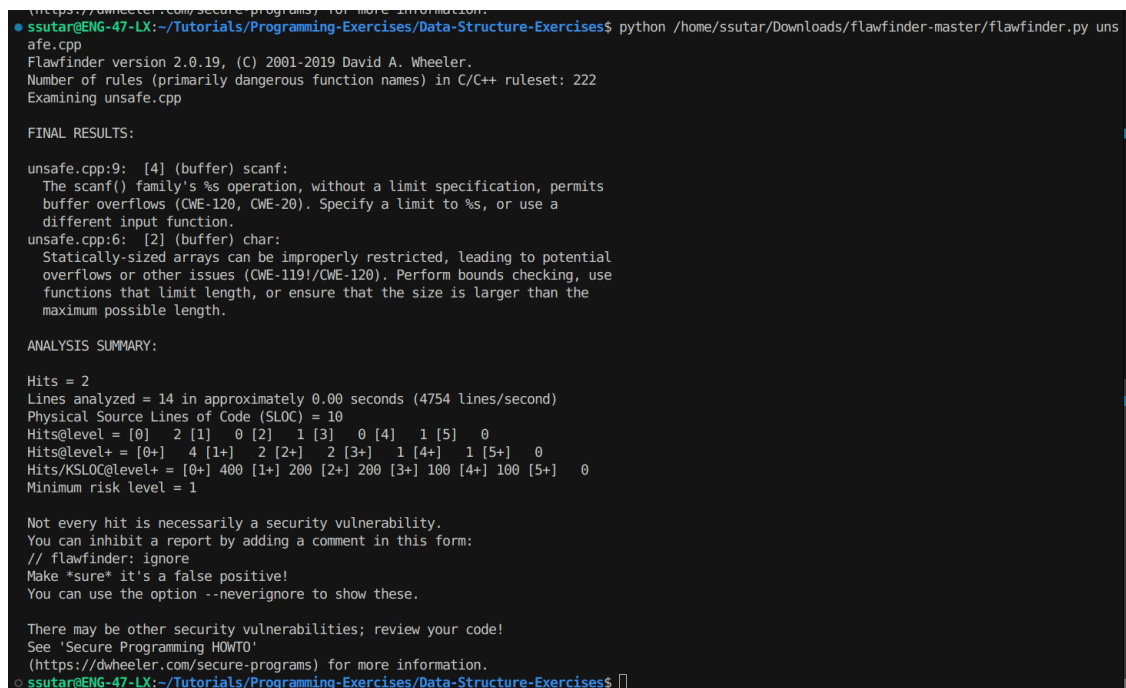
TERMINAL PORTS GITLENS PROBLEMS 1 OUTPUT DEBUG CONSOLE SPELL CHECKER 1

Executing task: /home/ssutar/Tutorials/Programming-Exercises/Data-Structure-Exercises/Output/unsafe.o

Enter some text: Sagar Sutar
You entered: Sagar
Press any key to close the terminal.

Below is o/p of passing the above code to the flawfinder. The interpretation is as follows:

1. No. of Hits: 2 :- There are 2 potential issues.
2. Prioritization:- The riskiest code is on line 9 i.e. the scanf function followed by line 6 i.e. the char initialization.



```
(https://dwtweller.com/secure-programs) for more information.
ssutar@ENG-47-LX:~/Tutorials/Programming-Exercises/Data-Structure-Exercises$ python /home/ssutar/Downloads/flawfinder-master/flawfinder.py unsafe.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 unsafe.cpp

FINAL RESULTS:

unsafe.cpp:9: [4] (buffer) scanf:
The scanf() family's %s operation, without a limit specification, permits
buffer overflows (CWE-120, CWE-20). Specify a limit to %s, or use a
different input function.
unsafe.cpp:6: [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 = 14 in approximately 0.00 seconds (4754 lines/second)
Physical Source Lines of Code (SLOC) = 10
Hits@level+ = [0] 2 [1] 0 [2] 1 [3] 0 [4] 1 [5] 0
Hits@level+ = [0+] 4 [1+] 2 [2+] 2 [3+] 1 [4+] 1 [5+] 0
Hits/KSLOC@level+ = [0+] 400 [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://dwtweller.com/secure-programs) for more information.
ssutar@ENG-47-LX:~/Tutorials/Programming-Exercises/Data-Structure-Exercises$
```

String Problem

Here strcpy is an unsafe function. It does not check the size of the variable in which it would copy the value.

```
String-Problem.cpp > main()

1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char name[10];
7      strcpy(name, "secret123dsdsdsds");
8      return 0;
9  }
```

Output of Flawfinder: It pointed out the no. of hits & problem code.

```
(https://www.wisecoder.com/secure-programs) for more information.
● ssutar@ENG-47-LX:~/Tutorials/Programming-Exercises/Data-Structure-Exercises$ python /home/ssutar/Downloads/flawfinder-master/flawfinder.py String-Problem.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 String-Problem.cpp

FINAL RESULTS:

String-Problem.cpp:6: [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.
String-Problem.cpp:7: [2] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strncpy (warning: strncpy
  easily misused). Risk is low because the source is a constant string.

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 8 in approximately 0.00 seconds (2245 lines/second)
Physical Source Lines of Code (SLOC) = 8
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+] 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:
```

Race Condition

This is a logical error that occurs when multiple simultaneous entities try to access the same resource or piece of a code. This cannot be detected by the Flaw Finder. In a manual code audit, “thread safe code” standards must be verified.