Mitesh Mahesh Salunkhe F003
M.SC. computer Science

Mithibai College of Arts,
Chauhan Institute of Science &
Amrutben Jivanlal College of Commerce and Economics
(AUTONOMOUS)
Affiliated to University of Mumbai
NAAC Reaccredited 'A' Grade, CGPA : 3.57 (February 2016)
Granted under RUSA for Enhancing Quality & Excellence in select Autonomous Colleges
GRANTED UNDER FIST-DST & STAR COLLEGE SCHEME OF DBT, GOVERNMENT OF INDIA
BEST COLLEGE (2016-17), University of Mumbai

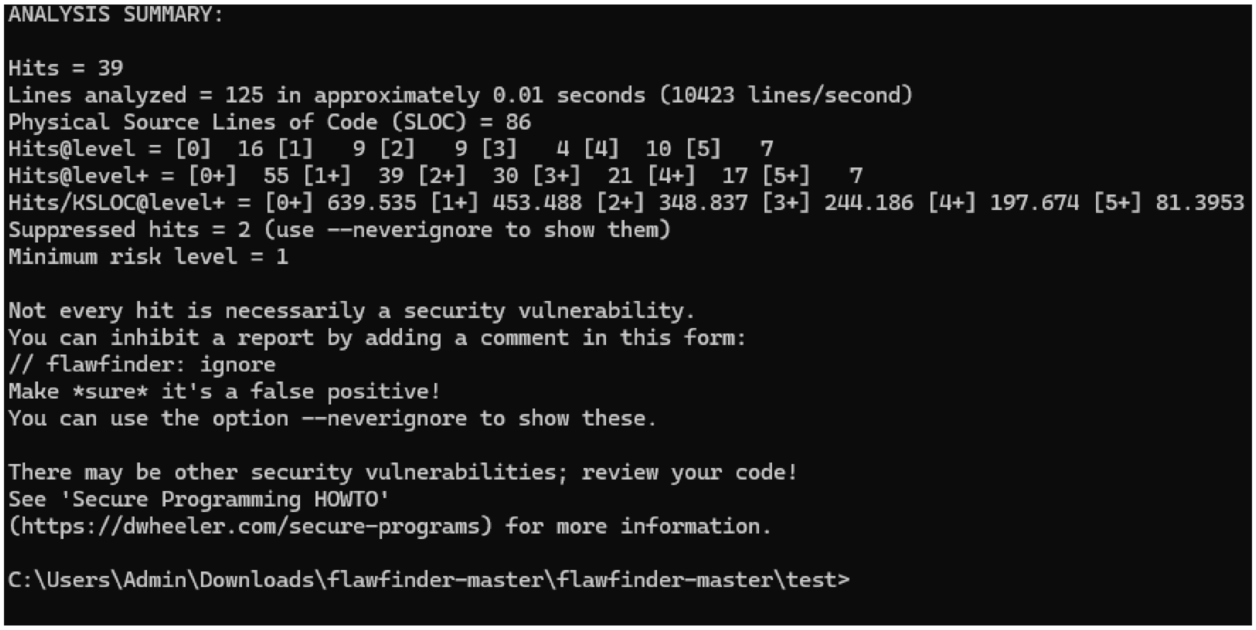
Practical-1 Static code analysis using open source tool Flawfinder

Date:-12/07/2024   Submission Date:- 19/07/2024

Write-up:-

Static code Analysis & Benefits
Vulnerability
Flawfinder

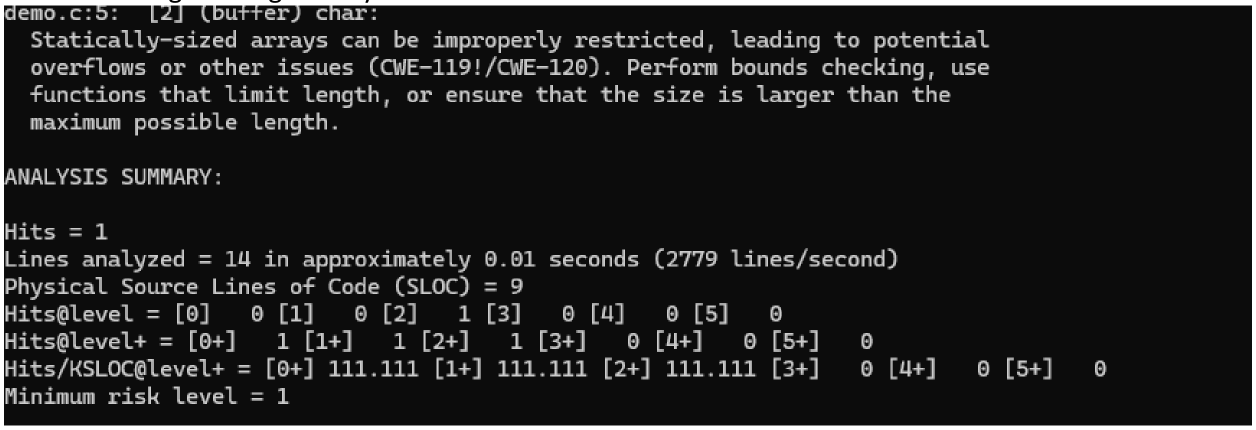
Implement static code analysis using open source tool Flawfinder for the following:
Buffer overflow
String problem
Race conditions,etc
Steps taken to perform the practical:
Download Flawfinder from Github
Using "pip install flawfinder" Install flawfinder using command prompt
Once flawfinder is installed we will test the given 'test.c' file given in the 'test' folder in the flawfinder folder.
Open command prompt and type "flawfinder test.c"
It will run and give the following hits



```
ANALYSIS SUMMARY:

Hits = 39
Lines analyzed = 125 in approximately 0.01 seconds (10423 lines/second)
Physical Source Lines of Code (SLOC) = 86
Hits@level = [0]  16 [1]   9 [2]   9 [3]   4 [4]  10 [5]   7
Hits@level+ = [0+]  55 [1+]  39 [2+]  30 [3+]  21 [4+]  17 [5+]   7
Hits/KSLOC@level+ = [0+] 639.535 [1+] 453.488 [2+] 348.837 [3+] 244.186 [4+] 197.674 [5+] 81.3953
Suppressed hits = 2 (use --neverignore to show them)
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\Admin\Downloads\flawfinder-master\flawfinder-master\test>
```

Mitesh Mahesh Salunkhe F003
M.SC. computer Science

Created C Program to get array error

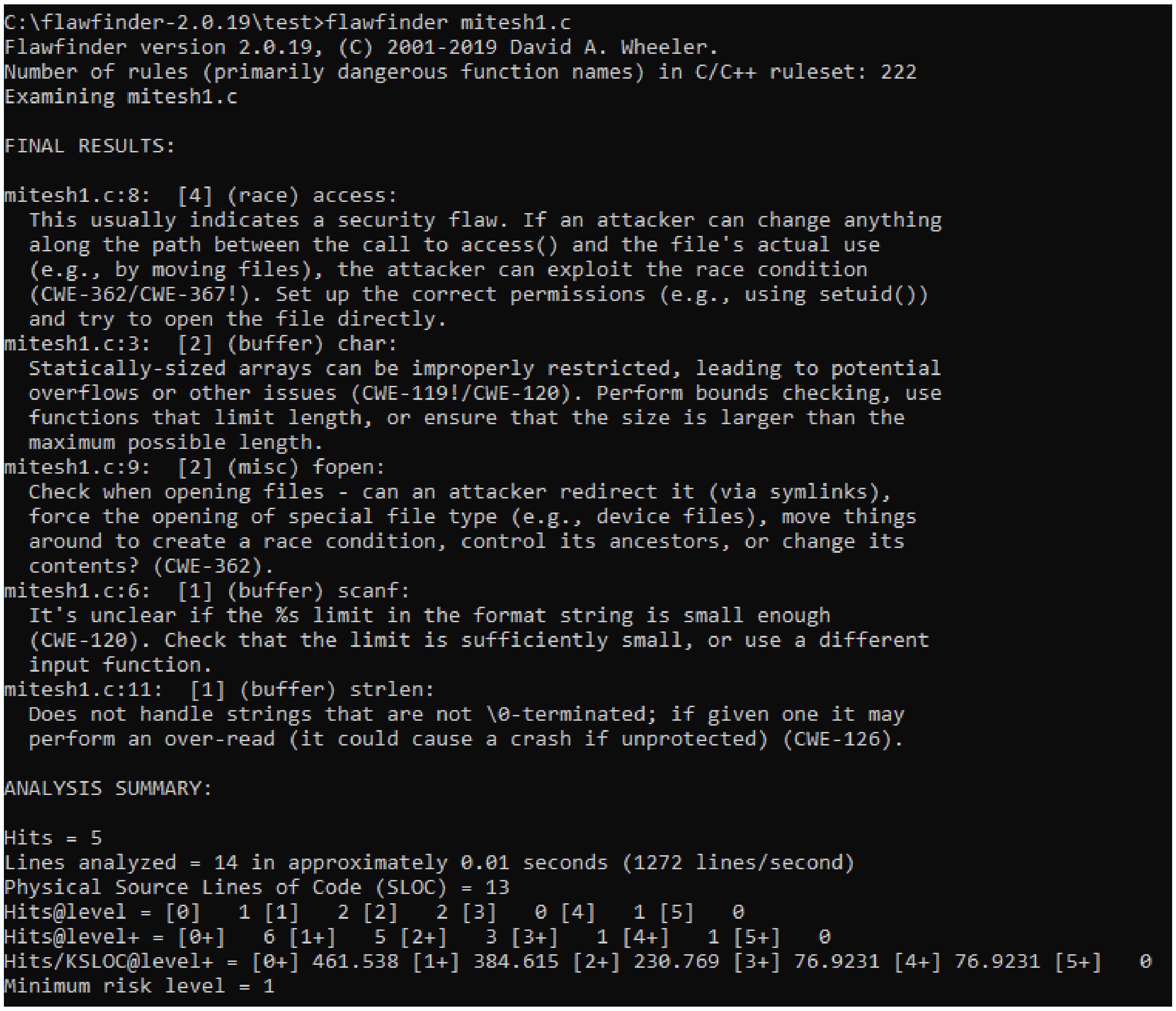


```
demo.c:5:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 14 in approximately 0.01 seconds (2779 lines/second)
Physical Source Lines of Code (SLOC) = 9
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+] 111.111 [1+] 111.111 [2+] 111.111 [3+]   0 [4+]   0 [5+]   0
Minimum risk level = 1
```

Created C Program to get race condition error:
Race Condition prac 1 paper 2

```
int main() {
   char * fn = "/tmp/XYZ";
   char buffer[60];
   FILE *fp;
   /* get user input */
   scanf("%50s", buffer );

   if(!access(fn, W_OK)){
      fp = fopen(fn, "a+");
      fwrite("\n", sizeof(char), 1, fp);
      fwrite(buffer, sizeof(char), strlen(buffer), fp);
      fclose(fp);
   }
   else printf("No permission \n");
}
```

Mitesh Mahesh Salunkhe F003
M.SC. computer Science



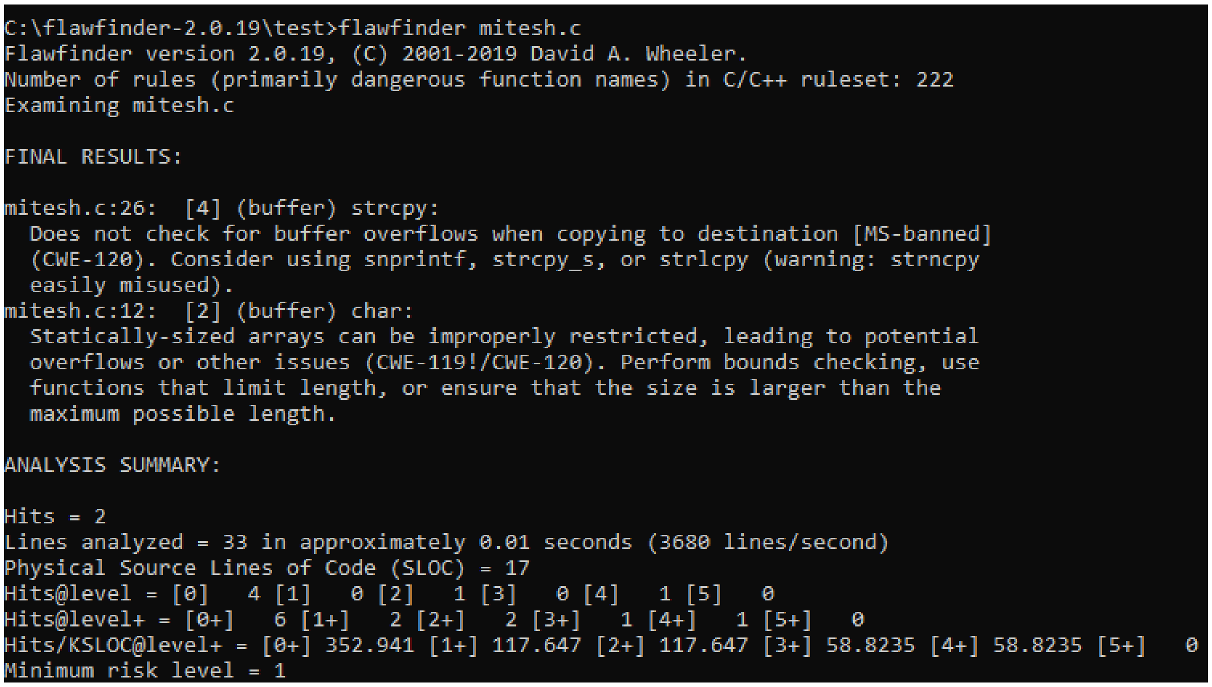
```
C:\flawfinder-2.0.19\test>flawfinder mitesh1.c
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining mitesh1.c

FINAL RESULTS:

mitesh1.c:8:  [4] (race) access:
  This usually indicates a security flaw. If an attacker can change anything
  along the path between the call to access() and the file's actual use
  (e.g., by moving files), the attacker can exploit the race condition
  (CWE-362/CWE-367!). Set up the correct permissions (e.g., using setuid())
  and try to open the file directly.
mitesh1.c:3:  [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.
mitesh1.c: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).
mitesh1.c:6:  [1] (buffer) scanf:
  It's unclear if the %s limit in the format string is small enough
  (CWE-120). Check that the limit is sufficiently small, or use a different
  input function.
mitesh1.c:11:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given one it may
  perform an over-read (it could cause a crash if unprotected) (CWE-126).

ANALYSIS SUMMARY:

Hits = 5
Lines analyzed = 14 in approximately 0.01 seconds (1272 lines/second)
Physical Source Lines of Code (SLOC) = 13
Hits@level = [0]   1 [1]   2 [2]   2 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   6 [1+]   5 [2+]   3 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 461.538 [1+] 384.615 [2+] 230.769 [3+] 76.9231 [4+] 76.9231 [5+]   0
Minimum risk level = 1
```

Created program to create buffer overflow error

```
// A C program to demonstrate buffer overflow
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{

  // Reserve 5 byte of buffer plus the terminating NULL.
  // should allocate 8 bytes = 2 double words,
  // To overflow, need more than 8 bytes...
  char buffer[5]; // If more than 8 characters input
                  // by user, there will be access
                  // violation, segmentation fault

  // a prompt how to execute the program...
```

Mitesh Mahesh Salunkhe F003
M.SC. computer Science

```
   if (argc < 2)
   {
          printf("strcpy() NOT executed....\n");
          printf("Syntax: %s <characters>\n", argv[0]);
          exit(0);
   }

   // copy the user input to mybuffer, without any
   // bound checking a secure version is strcpy_s()
   strcpy(buffer, argv[1]);
   printf("buffer content= %s\n", buffer);

   // you may want to try strcpy_s()
   printf("strcpy() executed...\n");

   return 0;
```



```
C:\flawfinder-2.0.19\test>flawfinder mitesh.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 mitesh.c

FINAL RESULTS:

mitesh.c:26:  [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).
mitesh.c:12:  [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 = 33 in approximately 0.01 seconds (3680 lines/second)
Physical Source Lines of Code (SLOC) = 17
Hits@level = [0]   4 [1]   0 [2]   1 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   6 [1+]   2 [2+]   2 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 352.941 [1+] 117.647 [2+] 117.647 [3+] 58.8235 [4+] 58.8235 [5+]   0
Minimum risk level = 1
```

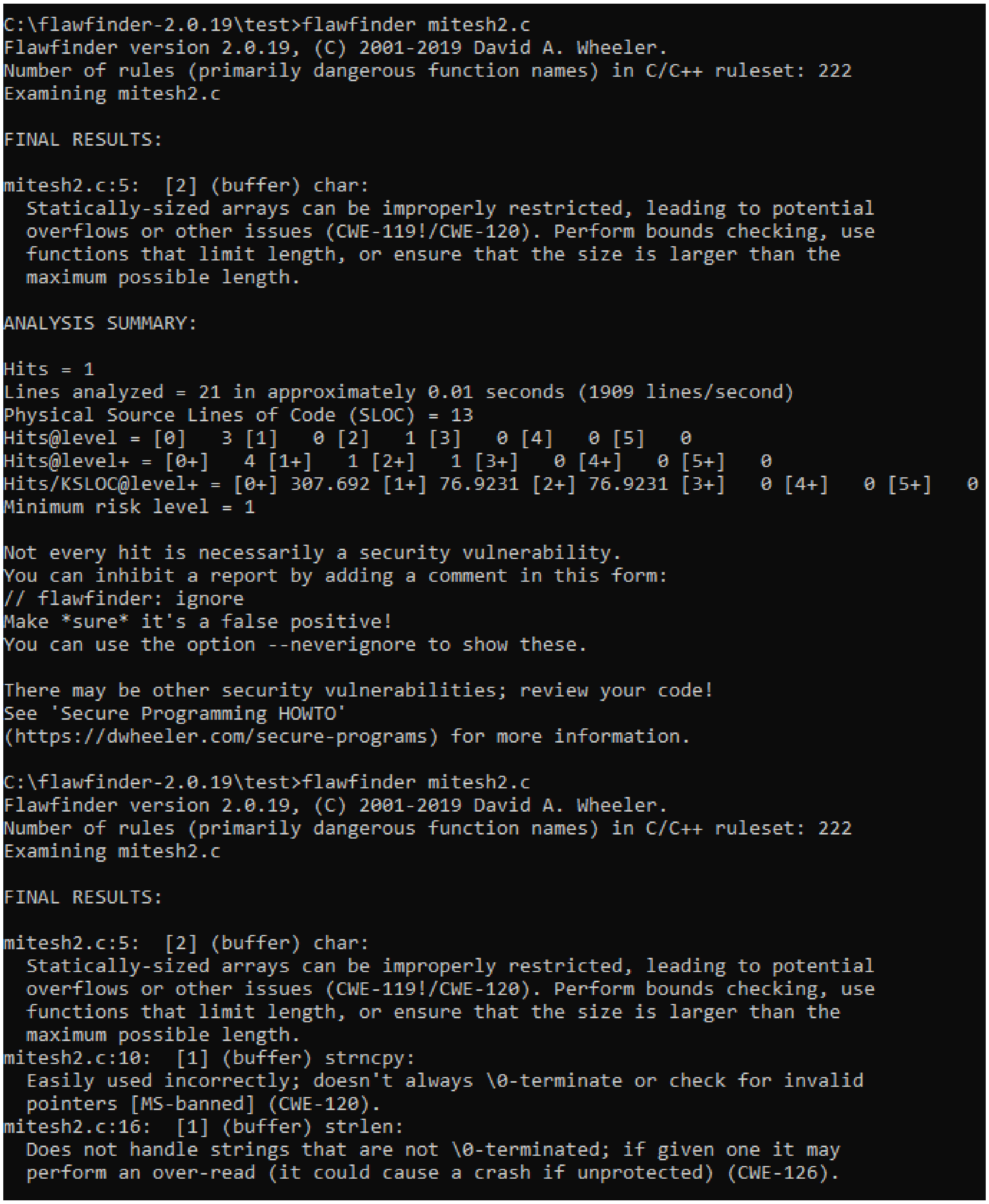
Created program to for string error

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

int main() {
   char buffer[5]; // Buffer to store strings (size 4 + 1 for null terminator)
   char *fruits[] = {"apple", "banana", "mango", "carrot", "chikoo", "jackfruit"};
   int i;

   for (i = 0; i < sizeof(fruits) / sizeof(fruits[0]); i++) {
```

Mitesh Mahesh Salunkhe F003
M.SC. computer Science

```
        strncpy(buffer, fruits[i], sizeof(buffer) - 1);
        buffer[sizeof(buffer) - 1] = '\0'; // Ensure null-termination

        printf("String %d: %s\n", i + 1, buffer);

        // If the string is longer than the buffer, handle accordingly
        if (strlen(fruits[i]) > sizeof(buffer) - 1) {
            printf("Warning: String '%s' truncated to fit buffer size.\n", fruits[i]);
        }
    }

    return 0;
}
```

Mitesh Mahesh Salunkhe F003
M.SC. computer Science



```
C:\flawfinder-2.0.19\test>flawfinder mitesh2.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 mitesh2.c

FINAL RESULTS:

mitesh2.c:5:  [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 = 1
Lines analyzed = 21 in approximately 0.01 seconds (1909 lines/second)
Physical Source Lines of Code (SLOC) = 13
Hits@level = [0]   3 [1]   0 [2]   1 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   4 [1+]   1 [2+]   1 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 307.692 [1+] 76.9231 [2+] 76.9231 [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.

C:\flawfinder-2.0.19\test>flawfinder mitesh2.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 mitesh2.c

FINAL RESULTS:

mitesh2.c:5:  [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.
mitesh2.c:10:  [1] (buffer) strncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).
mitesh2.c:16:  [1] (buffer) strlen:
  Does not handle strings that are not \0-terminated; if given one it may
  perform an over-read (it could cause a crash if unprotected) (CWE-126).
```