PRACTICAL No: 1

AIM Static code analysis using Open Source tool Flawfinder

THEORY:

Static Code Analysis

i) Static code analysis is a process for analyzing an application's code for potential errors.

ii) It analyzes the code without running the code that also defines the name 'static'.

iii) Static code analysis is used for testing applications for security flaws and vullnerabilites.

iv) Static analysis is generally done before software testing in early development.

v) It is used for Q.A by quality assurance teams.

Benefits ~~of~~ & drawbacks of static analysis

i) It can evaluate all the code in an application, increasing code quality.

ii) It provides speed in using automated tools compared to manual code review.

iii) Paired with normal methods, static testings allow for more depth into debugging code.

iv) It can be done in an offline development environment.

Vulnerability:

i) Vulnerability is a weakness that can be exploited by cybercriminals to gain unauthorized access to a computer system
ii) Once a vulnerability is exploited cyber attacker may run malicious code that ~~tot~~ could be exploited or triggered by a threat source.

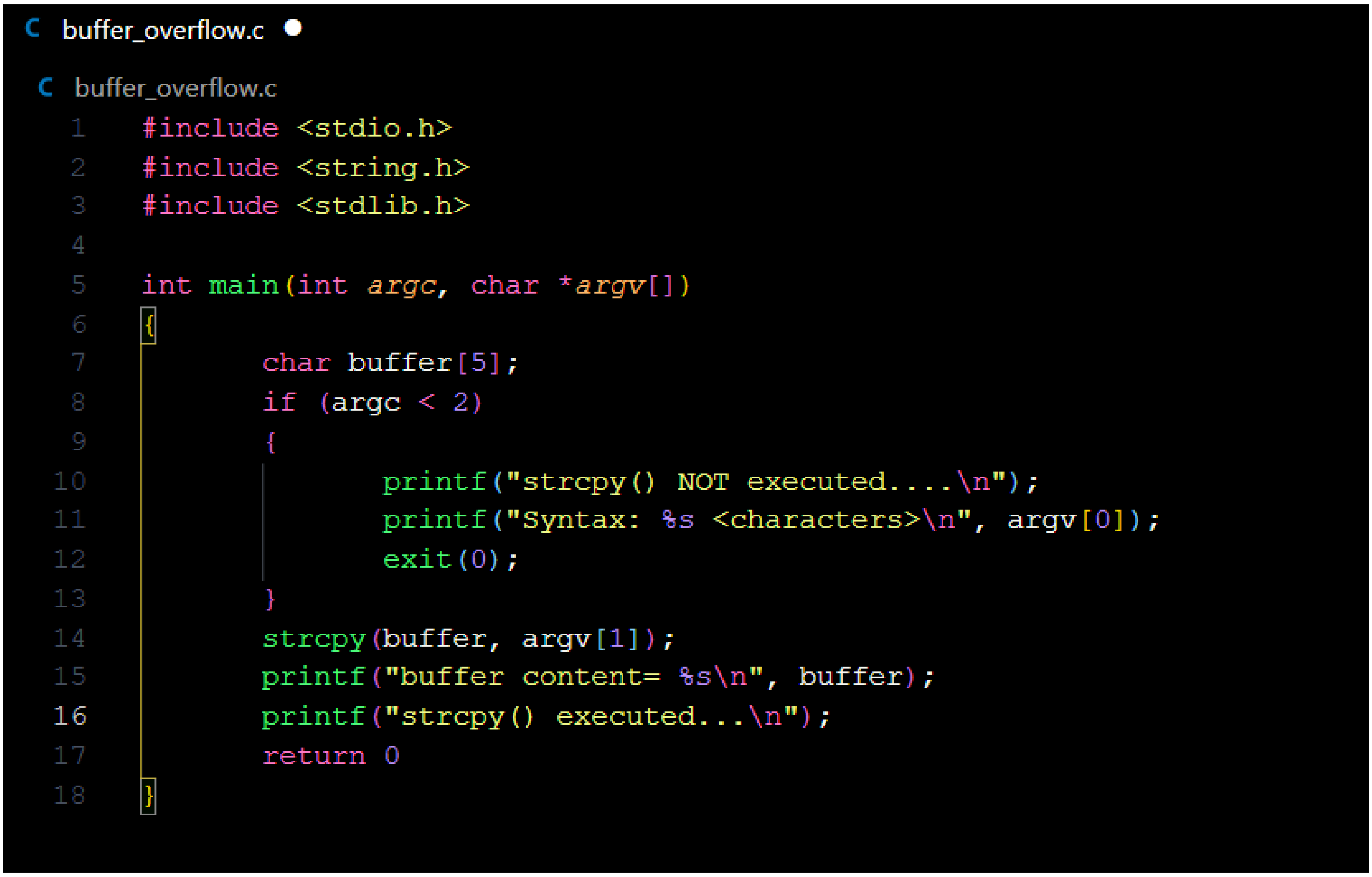
Flawfinder:

a) Flawfinder is an exceptional source-scanning tool that programmers can depend on to find the most common security problem with C program.
b> A simple program that examine C/C++ source code and reports possible security weaknesses sorted by risk level.
c> It's very useful for quickly finding and removing at least some potential security flaws sorted by risk level

Static Code Analysis using Flawfinder:

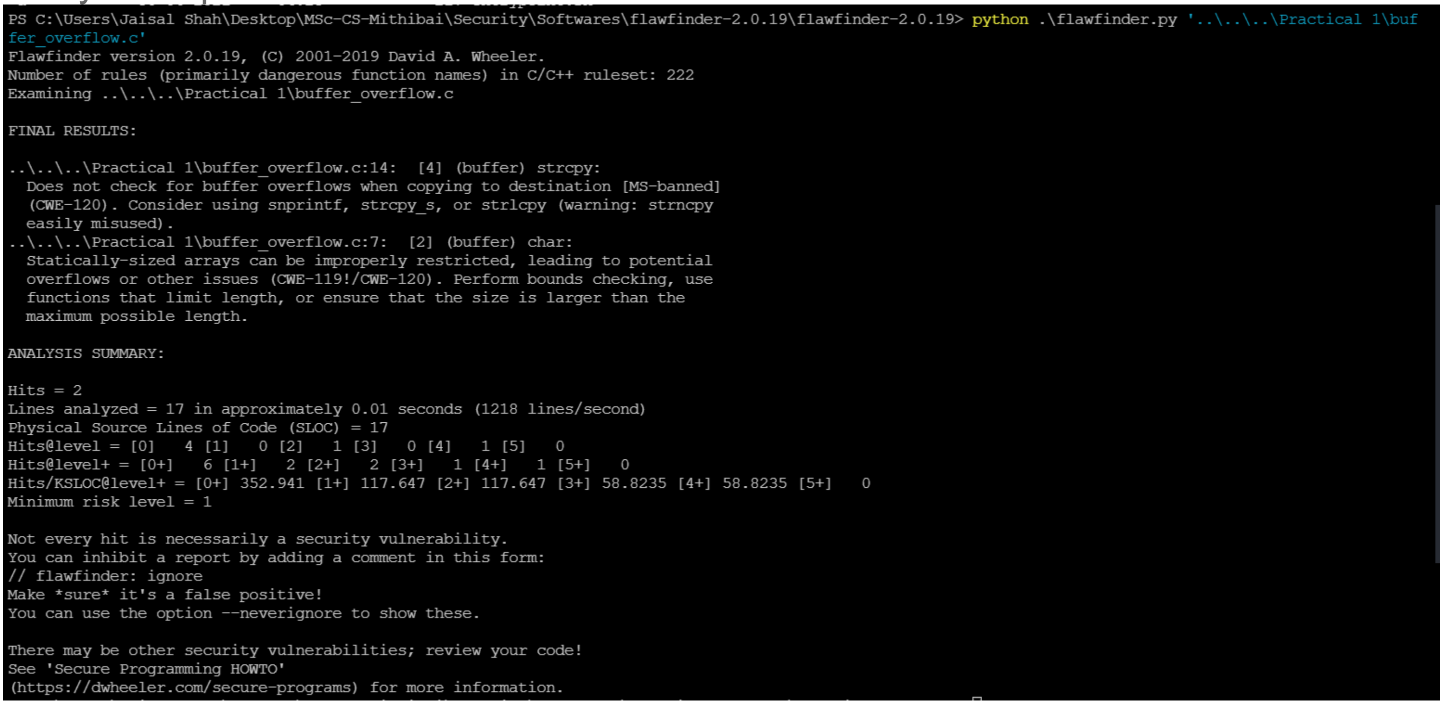
i) Buffer Overflow

Code:



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

int main(int argc, char *argv[])
{
        char buffer[5];
        if (argc < 2)
        {
                printf("strcpy() NOT executed....\n");
                printf("Syntax: %s <characters>\n", argv[0]);
                exit(0);
        }
        strcpy(buffer, argv[1]);
        printf("buffer content= %s\n", buffer);
        printf("strcpy() executed...\n");
        return 0
}
```

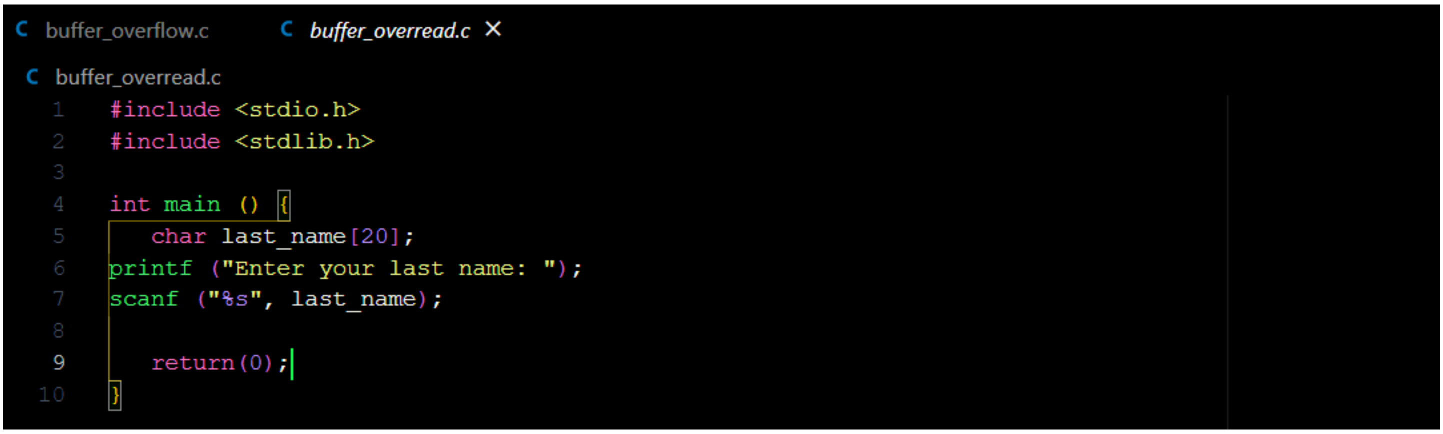
Analysis Report:



```
PS C:\Users\Jaisal Shah\Desktop\MSc-CS-Mithibai\Security\Softwares\flawfinder-2.0.19\flawfinder-2.0.19> python .\flawfinder.py '..\..\..\Practical 1\buf
fer_overflow.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 ..\..\..\Practical 1\buffer_overflow.c

FINAL RESULTS:

..\..\..\Practical 1\buffer_overflow.c:14:  [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).
..\..\..\Practical 1\buffer_overflow.c: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.

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 17 in approximately 0.01 seconds (1218 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

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.
```

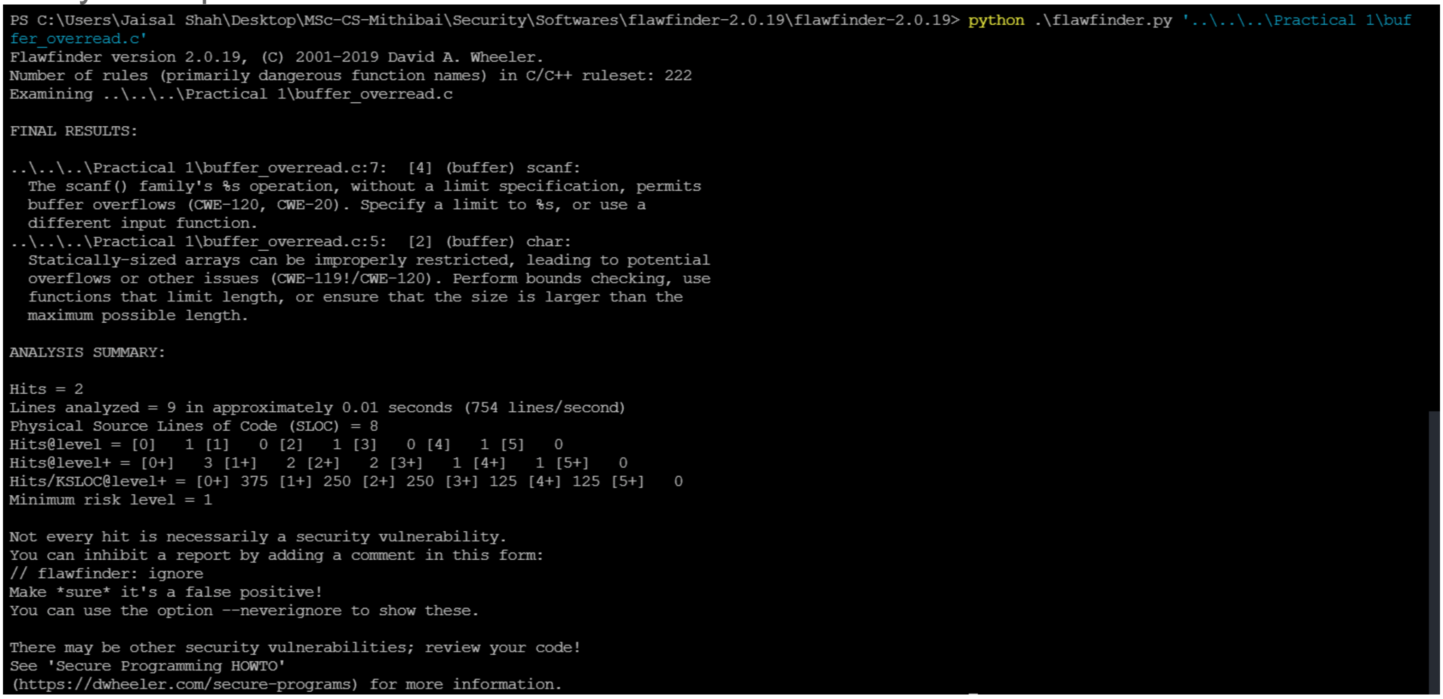
ii) Buffer Overread:

Code:



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

int main () {
    char last_name[20];
printf ("Enter your last name: ");
scanf ("%s", last_name);

    return(0);
}
```

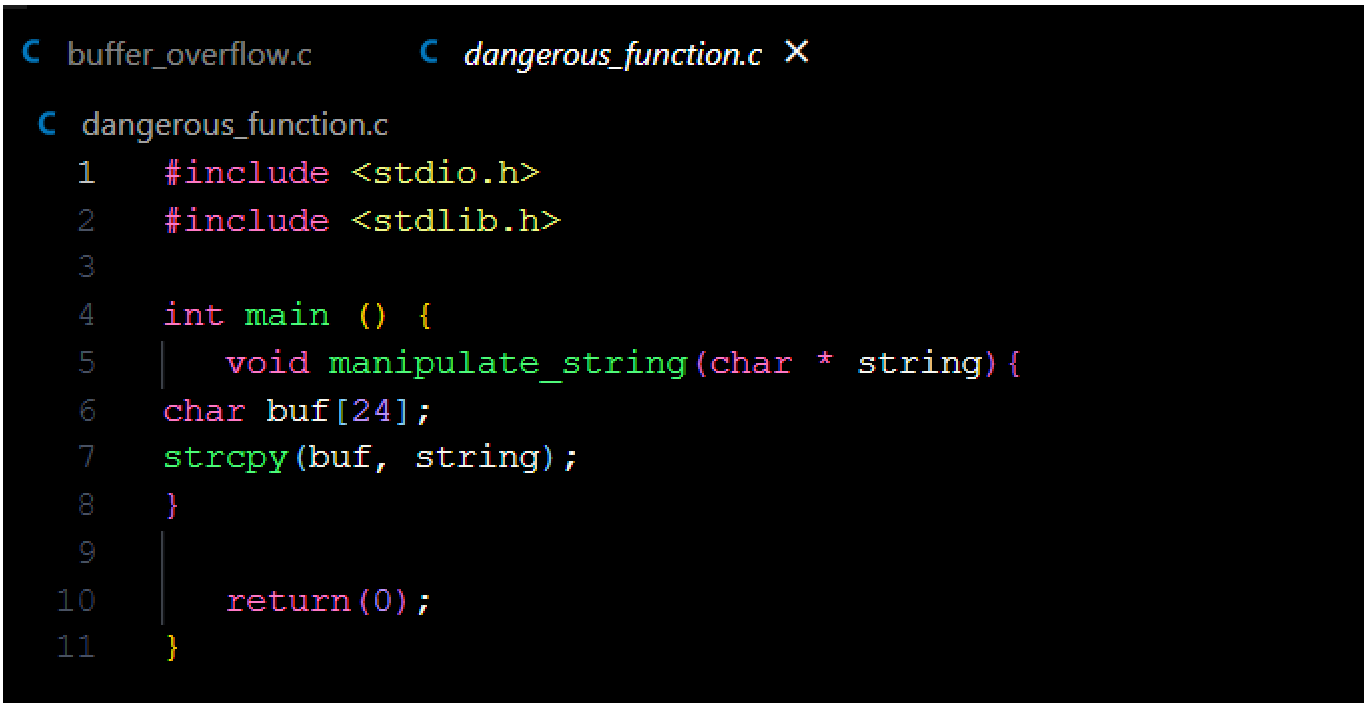
Analysis Report:



```
PS C:\Users\Jaisal Shah\Desktop\MSc-CS-Mithibai\Security\Softwares\flawfinder-2.0.19\flawfinder-2.0.19> python .\flawfinder.py '..\..\..\Practical 1\buf
fer_overread.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 ..\..\..\Practical 1\buffer_overread.c

FINAL RESULTS:

..\..\..\Practical 1\buffer_overread.c:7:  [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.
..\..\..\Practical 1\buffer_overread.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 = 2
Lines analyzed = 9 in approximately 0.01 seconds (754 lines/second)
Physical Source Lines of Code (SLOC) = 8
Hits@level = [0]   1 [1]   0 [2]   1 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   3 [1+]   2 [2+]   2 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 375 [1+] 250 [2+] 250 [3+] 125 [4+] 125 [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.
```

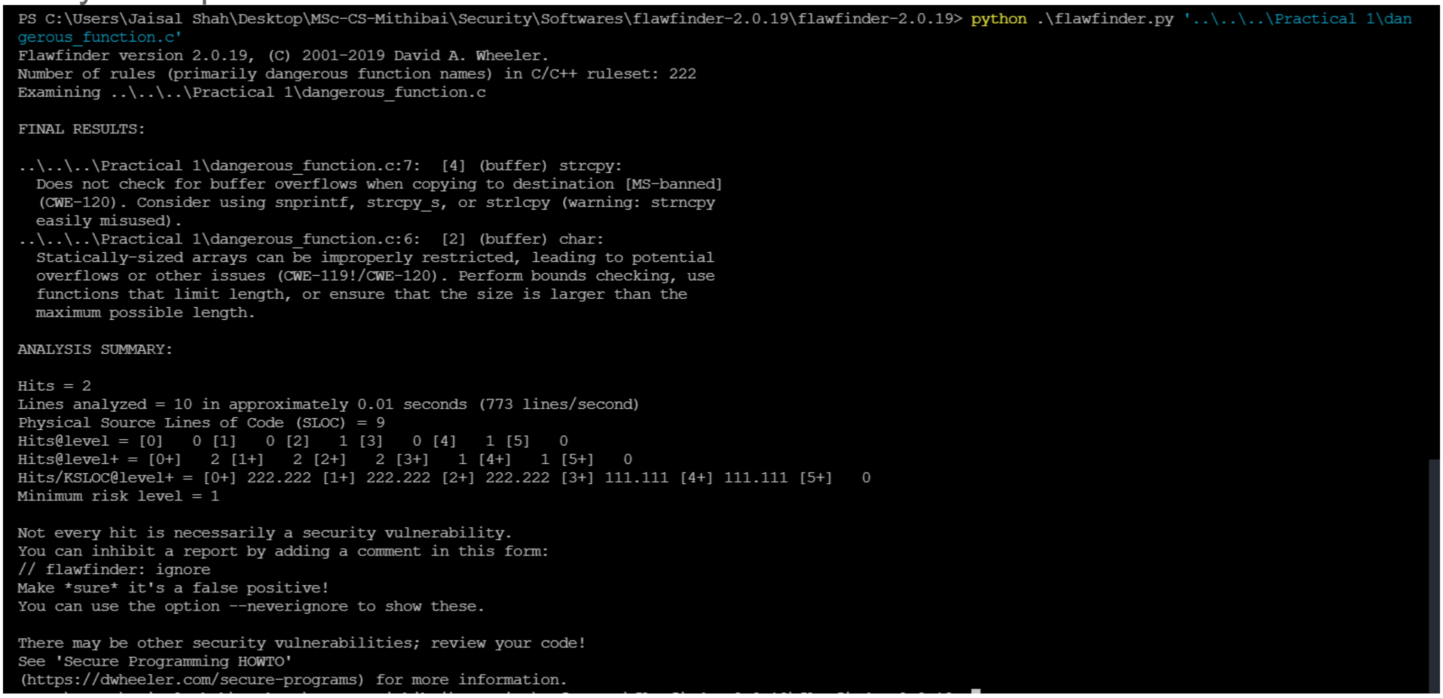
iii) Dangerous Function

Code:



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

int main () {
    void manipulate_string(char * string){
char buf[24];
strcpy(buf, string);
}

    return(0);
}
```

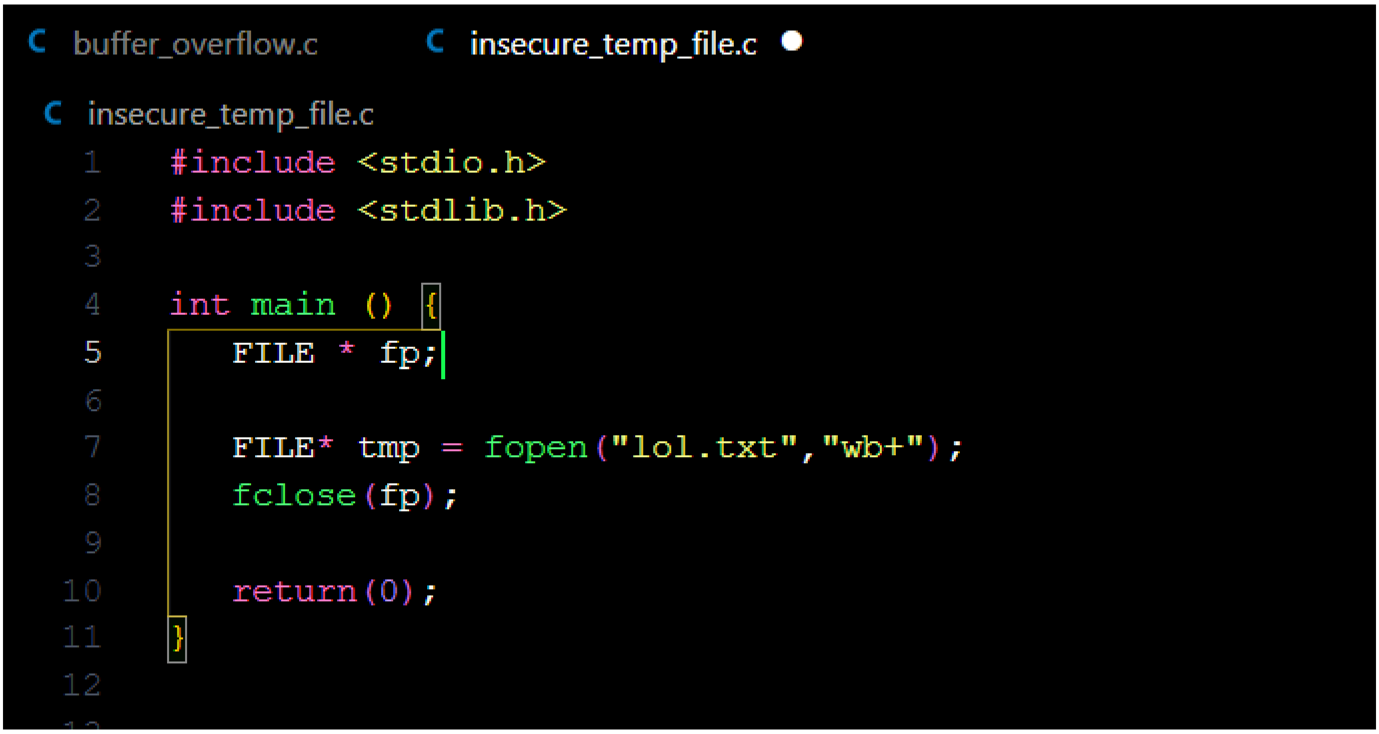
Analysis Report



```
PS C:\Users\Jaisal Shah\Desktop\MSc-CS-Mithibai\Security\Softwares\flawfinder-2.0.19\flawfinder-2.0.19> python .\flawfinder.py '..\..\..\Practical 1\dan
gerous_function.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 ..\..\..\Practical 1\dangerous_function.c

FINAL RESULTS:

..\..\..\Practical 1\dangerous_function.c:7:  [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).
..\..\..\Practical 1\dangerous_function.c: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 = 10 in approximately 0.01 seconds (773 lines/second)
Physical Source Lines of Code (SLOC) = 9
Hits@level = [0]   0 [1]   0 [2]   1 [3]   0 [4]   1 [5]   0
Hits@level+ = [0+]   2 [1+]   2 [2+]   2 [3+]   1 [4+]   1 [5+]   0
Hits/KSLOC@level+ = [0+] 222.222 [1+] 222.222 [2+] 222.222 [3+] 111.111 [4+] 111.111 [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.
```

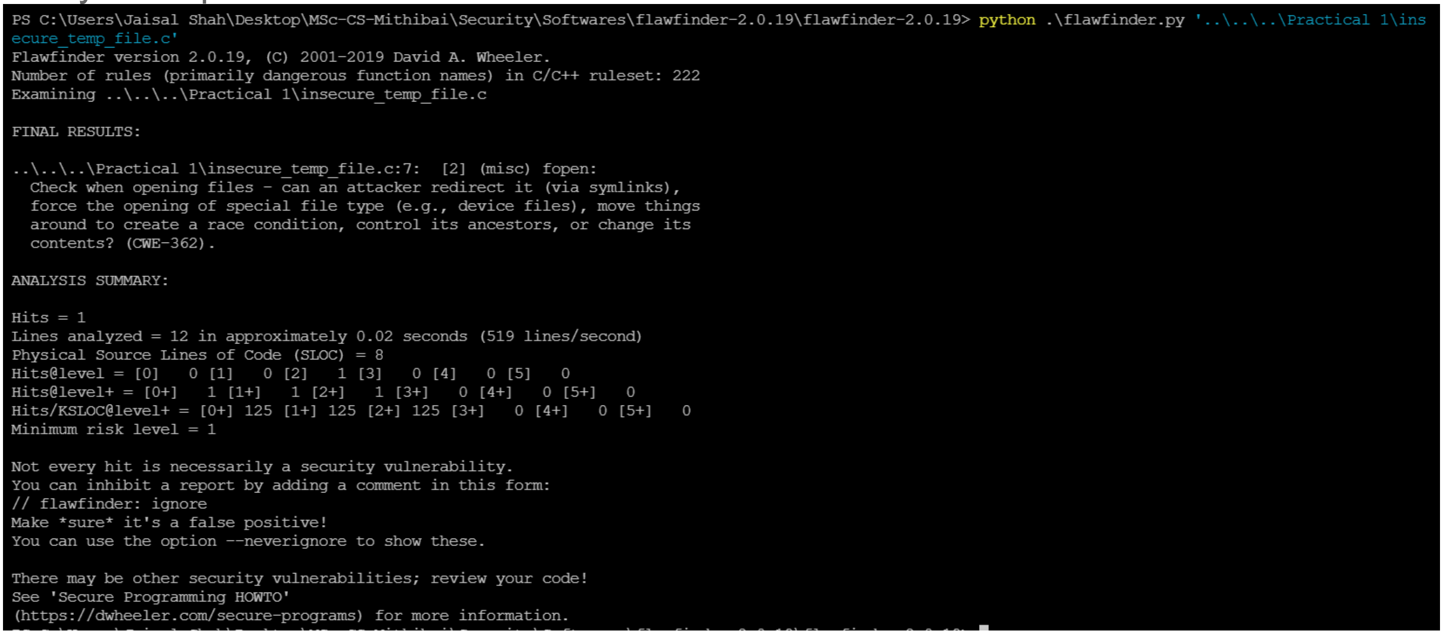
iv) Insecure File Access:

Code:



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

int main () {
    FILE * fp;

    FILE* tmp = fopen("lol.txt","wb+");
    fclose(fp);

    return(0);
}
```

Analysis Report:



```
PS C:\Users\Jaisal Shah\Desktop\MSc-CS-Mithibai\Security\Softwares\flawfinder-2.0.19\flawfinder-2.0.19> python .\flawfinder.py '..\..\..\Practical 1\ins
ecure_temp_file.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 ..\..\..\Practical 1\insecure_temp_file.c

FINAL RESULTS:

..\..\..\Practical 1\insecure_temp_file.c:7:  [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 = 1
Lines analyzed = 12 in approximately 0.02 seconds (519 lines/second)
Physical Source Lines of Code (SLOC) = 8
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+] 125 [1+] 125 [2+] 125 [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.
```

v) Code without any vulnerability

Code: