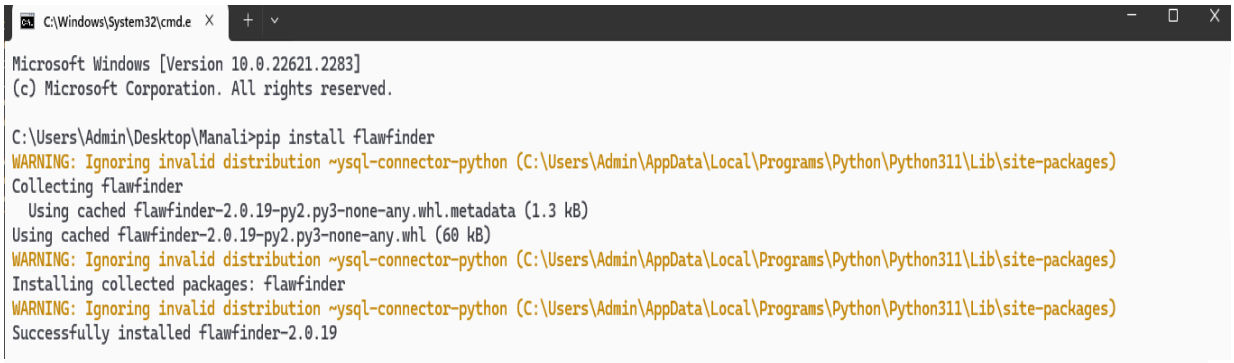
Name:-Manali Pradeep Kuware
Roll No:-F009
SAP ID:-40772240008

Practical 1

1.First step is to install flawfinder from Github



```
C:\Windows\System32\cmd.e  X
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin\Desktop\Manali>pip install flawfinder
WARNING: Ignoring invalid distribution ~ysql-connector-python (C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages)
Collecting flawfinder
  Using cached flawfinder-2.0.19-py2.py3-none-any.whl.metadata (1.3 kB)
Using cached flawfinder-2.0.19-py2.py3-none-any.whl (60 kB)
WARNING: Ignoring invalid distribution ~ysql-connector-python (C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages)
Installing collected packages: flawfinder
WARNING: Ignoring invalid distribution ~ysql-connector-python (C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages)
Successfully installed flawfinder-2.0.19
```

```
C:\Users\Admin\Desktop\Manali>cd test

C:\Users\Admin\Desktop\Manali\test>flawfinder test
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
*** No input files

C:\Users\Admin\Desktop\Manali\test>flawfinder test.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 test.c

FINAL RESULTS:

test.c:32:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
test.c:56:  [5] (buffer) strncat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) [MS-banned] (CWE-120). Consider strcat_s, strlcat, snprintf,
  or automatically resizing strings. Risk is high; the length parameter
  appears to be a constant, instead of computing the number of characters
  left.
test.c:57:  [5] (buffer) _tcsncat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) [MS-banned] (CWE-120). Consider strcat_s, strlcat, or
  automatically resizing strings. Risk is high; the length parameter appears
  to be a constant, instead of computing the number of characters left.
test.c:60:  [5] (buffer) MultiByteToWideChar:
  Requires maximum length in CHARACTERS, not bytes (CWE-120). Risk is high,
  it appears that the size is given as bytes, but the function requires size
  as characters.
test.c:62:  [5] (buffer) MultiByteToWideChar:
  Requires maximum length in CHARACTERS, not bytes (CWE-120). Risk is high,
  it appears that the size is given as bytes, but the function requires size
  as characters.
test.c:73:  [5] (misc) SetSecurityDescriptorDacl:
  Never create NULL ACLs; an attacker can set it to Everyone (Deny All
  Access), which would even forbid administrator access (CWE-732).
test.c:73:  [5] (misc) SetSecurityDescriptorDacl:
  Never create NULL ACLs; an attacker can set it to Everyone (Deny All
  Access), which would even forbid administrator access (CWE-732).
test.c:17:  [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).
test.c:20:  [4] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf.
test.c:21:  [4] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf.
test.c:22:  [4] (format) sprintf:
  Potential format string problem (CWE-134). Make format string constant.
test.c:23:  [4] (format) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.
```

```
test.c:25:  [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.
test.c:27:  [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.
test.c:38:  [4] (format) syslog:
  If syslog's format strings can be influenced by an attacker, they can be
  exploited (CWE-134). Use a constant format string for syslog.
test.c:49:  [4] (buffer) _mbscpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using a function version that stops copying at the end
  of the buffer.
test.c:52:  [4] (buffer) lstrcat:
  Does not check for buffer overflows when concatenating to destination
  [MS-banned] (CWE-120).
test.c:75:  [3] (shell) CreateProcess:
  This causes a new process to execute and is difficult to use safely
  (CWE-78). Specify the application path in the first argument, NOT as part
  of the second, or embedded spaces could allow an attacker to force a
  different program to run.
test.c:75:  [3] (shell) CreateProcess:
  This causes a new process to execute and is difficult to use safely
  (CWE-78). Specify the application path in the first argument, NOT as part
  of the second, or embedded spaces could allow an attacker to force a
  different program to run.
test.c:91:  [3] (buffer) getopt_long:
  Some older implementations do not protect against internal buffer overflows
  (CWE-120, CWE-20). Check implementation on installation, or limit the size
  of all string inputs.
test.c:16:  [2] (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). Risk is low because the source is a constant string.
test.c:19:  [2] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf. Risk is low because the source has a constant maximum length.
test.c:45:  [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.
test.c:46:  [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.
test.c:50:  [2] (buffer) memcpy:
  Does not check for buffer overflows when copying to destination (CWE-120).
  Make sure destination can always hold the source data.
test.c:51:  [2] (buffer) CopyMemory:
  Does not check for buffer overflows when copying to destination (CWE-120).
```

```
  Make sure destination can always hold the source data.
test.c:51:  [2] (buffer) CopyMemory:
  Does not check for buffer overflows when copying to destination (CWE-120).
  Make sure destination can always hold the source data.
test.c:97:  [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).
test.c:15:  [1] (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). Risk is low because the source is a constant character.
test.c:18:  [1] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf. Risk is low because the source is a constant character.
test.c:26:  [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.
test.c:53:  [1] (buffer) strncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).
test.c:54:  [1] (buffer) _tcsncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).
test.c:55:  [1] (buffer) strncat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) [MS-banned] (CWE-120). Consider strcat_s, strlcat, snprintf,
  or automatically resizing strings.
test.c:58:  [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).
test.c:64:  [1] (buffer) MultiByteToWideChar:
  Requires maximum length in CHARACTERS, not bytes (CWE-120). Risk is very
  low, the length appears to be in characters not bytes.
test.c:66:  [1] (buffer) MultiByteToWideChar:
  Requires maximum length in CHARACTERS, not bytes (CWE-120). Risk is very
  low, the length appears to be in characters not bytes.
```

1.Static Hits

Code

```
/* Test flawfinder.  This program won't compile or run; that's not necessary
   for this to be a useful test. */

#include <stdio.h>
#define hello(x) goodbye(x)
#define WOKKA "stuff"

main() {
 printf("hello\n");
}

/* This is a strcpy test. */

int demo(char *a, char *b) {
 strcpy(a, "\n"); // Did this work?
 strcpy(a, gettext("Hello there")); // Did this work?
 strcpy(b, a);
 sprintf(s, "\n");
 sprintf(s, "hello");
 sprintf(s, "hello %s", bug);
 sprintf(s, gettext("hello %s"), bug);
 sprintf(s, unknown, bug);
 printf(bf, x);
 scanf("%d", &x);
 scanf("%s", s);
 scanf("%10s", s);
 scanf("%s", s);
 gets(f); // Flawfinder: ignore
 printf("\\");
 /* Flawfinder: ignore */
 gets(f);
 gets(f);
 /* These are okay, but flawfinder version < 0.20 incorrectly used
    the first parameter as the parameter for the format string */
 syslog(LOG_ERR,"cannot open config file (%s): %s",filename,strerror(errno))
 syslog(LOG_CRIT,"malloc() failed");
 /* But this one SHOULD trigger a warning. */
 syslog(LOG_ERR, attacker_string);

}

demo2() {
  char d[20];
  char s[20];
  int n;

  _mbscpy(d,s); /* like strcpy, this doesn't check for buffer overflow */
  memcpy(d,s);
  CopyMemory(d,s);
```

```
  lstrcat(d,s);
  strncpy(d,s);
  _tcsncpy(d,s);
  strncat(d,s,10);
  strncat(d,s,sizeof(d)); /* Misuse - this should be flagged as riskier. */
  _tcsncat(d,s,sizeof(d)); /* Misuse - flag as riskier */
  n = strlen(d);
  /* This is wrong, and should be flagged as risky: */
  MultiByteToWideChar(CP_ACP,0,szName,-1,wszUserName,sizeof(wszUserName));
  /* This is also wrong, and should be flagged as risky: */
  MultiByteToWideChar(CP_ACP,0,szName,-1,wszUserName,sizeof wszUserName);
  /* This is much better: */
  MultiByteToWideChar(CP_ACP,0,szName,-
1,wszUserName,sizeof(wszUserName)/sizeof(wszUserName[0]));
  /* This is much better: */
  MultiByteToWideChar(CP_ACP,0,szName,-1,wszUserName,sizeof wszUserName
/sizeof(wszUserName[0]));
  /* This is an example of bad code - the third paramer is NULL, so it creates
     a NULL ACL.  Note that Flawfinder can't detect when a
     SECURITY_DESCRIPTOR structure is manually created with a NULL value
     as the ACL; doing so would require a tool that handles C/C++
     and knows about types more that flawfinder currently does.
     Anyway, this needs to be detected: */
  SetSecurityDescriptorDacl(&sd,TRUE,NULL,FALSE);
  /* This one is a bad idea - first param shouldn't be NULL */
  CreateProcess(NULL, "C:\\Program Files\\GoodGuy\\GoodGuy.exe -x", "");
  /* Test interaction of quote characters */
  printf("%c\n", 'x');
  printf("%c\n", '"');
  printf("%c\n", '\"');
  printf("%c\n", '\'');
  printf("%c\n", '\177');
  printf("%c\n", '\xfe');
  printf("%c\n", '\xd');
  printf("%c\n", '\n');
  printf("%c\n", '\\');
  printf("%c\n", '''');
}


int getopt_example(int argc,char *argv[]) {
    while ((optc = getopt_long (argc, argv, "a",longopts, NULL )) != EOF) {
    }
}

int testfile() {
  FILE *f;
  f = fopen("/etc/passwd", "r");
  fclose(f);
}
```

```
/* Regression test: handle \\\n after end of string */

#define assert(x) {\
 if (!(x)) {\
 fprintf(stderr,"Assertion failed.\n"\
 "File: %s\nLine: %d\n"\
 "Assertion: %s\n\n"\
 ,__FILE__,__LINE__,#x);\
 exit(1);\
 };\
 }

int accesstest() {
  int access = 0; /* Not a function call.  Should be caught by the
                     false positive test, and NOT labelled as a problem. */
}
```

```
ANALYSIS SUMMARY:

Hits = 36
Lines analyzed = 116 in approximately 0.02 seconds (6347 lines/second)
Physical Source Lines of Code (SLOC) = 80
Hits@level = [0]  16 [1]   9 [2]   7 [3]   3 [4]  10 [5]   7
Hits@level+ = [0+]  52 [1+]  36 [2+]  27 [3+]  20 [4+]  17 [5+]   7
Hits/KSLOC@level+ = [0+] 650 [1+] 450 [2+] 337.5 [3+] 250 [4+] 212.5 [5+] 87.5
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\Desktop\Manali\test>
```

2.Array Error

Code

```
std::string MyClass::randomGenerator(odb::nullable<int> maxLength) {

   struct timeval tmnow;

   struct tm *tm;

   char buf[100];

   gettimeofday(&tmnow, NULL);

   tm = localtime(&tmnow.tv_sec);

   strftime(buf, 100, "%m%d%H%M%S", tm);

   string micro = std::to_string(((int)tmnow.tv_usec / 10000));

   strlcat(buf, micro.c_str(), sizeof(buf));
```

```
    std::stringstream stream;

    stream << std::hex << stoll(buf);

    std::string result(stream.str());

    Utilities::find_and_replace(result, "0", "h");

    Utilities::find_and_replace(result, "1", "k");

    std::transform(result.begin(), result.end(),result.begin(), ::toupper);

    if (maxLength) {

        return result.substr(result.size() - maxLength.get(), result.size() - 1);

    } else {

        return result ;

    }

}
```

```
C:\Users\Admin\Desktop\Manali\test>flawfinder test1
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
Examining test1\test1.c

FINAL RESULTS:

test1\test1.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 = 1
Lines analyzed = 40 in approximately 0.01 seconds (7514 lines/second)
Physical Source Lines of Code (SLOC) = 21
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+] 47.619 [1+] 47.619 [2+] 47.619 [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:\Users\Admin\Desktop\Manali\test>
```

3.String Error

Code

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

void risky_function(const char *input) {
    char buffer[10];
```

```
    int num;

    // Buffer overflow vulnerability
    strcpy(buffer, input);  // Dangerous if input is larger than buffer

    // Format string vulnerability
    printf(input);  // Dangerous if input contains format specifiers

    // Buffer overflow vulnerability
    num = atoi(input);  // Unsafe conversion, no validation
    printf("Number: %d\n", num);  // No direct issue here but related to the previous line
}

int main() {
    char user_input[50];

    printf("Enter a string: ");
    fgets(user_input, sizeof(user_input), stdin);

    // Remove trailing newline character from fgets
    user_input[strcspn(user_input, "\n")] = '\0';

    risky_function(user_input);

    return 0;
}
```

```
ANALYSIS SUMMARY:

No hits found.
Lines analyzed = 0 in approximately 0.00 seconds (0 lines/second)
Physical Source Lines of Code (SLOC) = 0
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

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\Admin\Desktop\Manali\test>flawfinder test2
Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
```