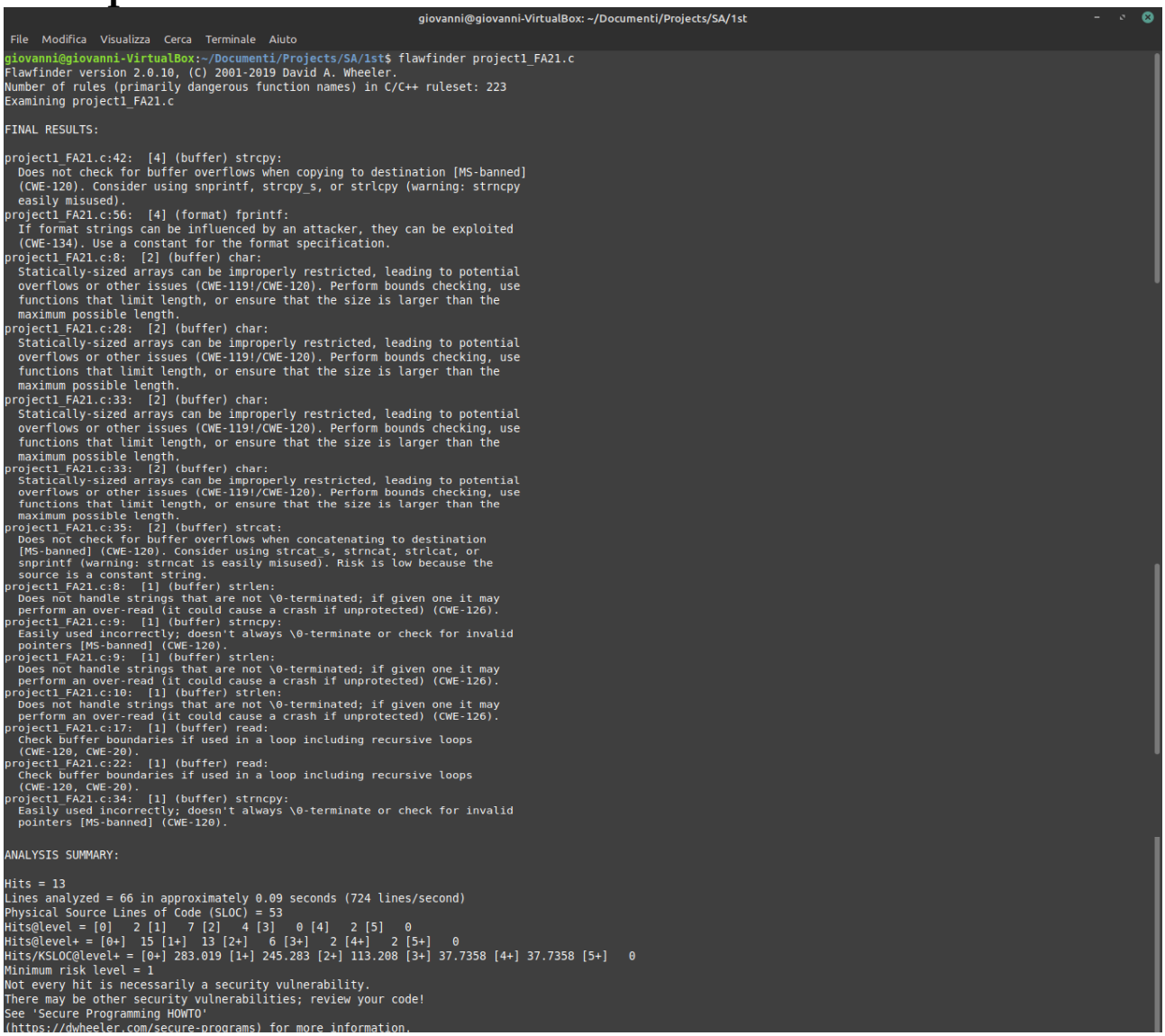
Project 1
Security In Software Applications
A.Y. 2021-2022

Pica Giovanni 1816394

1 Flawfinder tool

This tool is very useful for quickly and easily find and remove some potential security problem, but it doesn't really understand the semantics of the code, because it doesn't know the system goal.

2 Output of the tool



3 Analysis

In this paragraph I analyze every warning returned by the tool. Each warning has two images, one for the output of the tool and one for the code.

1st warning:

```
project1_FA21.c:42:  [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).
```

```
void func4(char *foo)
{
char *buffer = (char *)malloc(10 * sizeof(char));
strcpy(buffer, foo);
}
```

This is about strcpy() usage, this function does not know the size of the foo array and so if we got to copy a large array to a small one it causes buffer overflow. I use strncpy and \0-terminate the buffer (that I prefer static because we know the size).

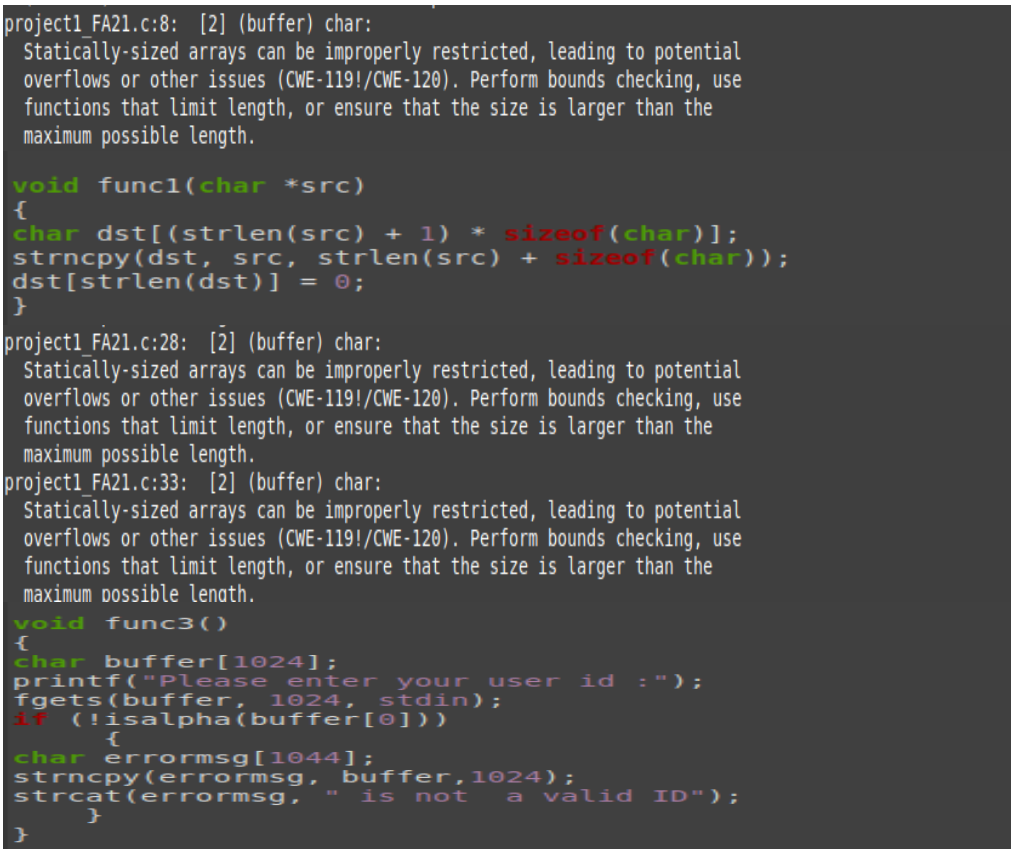
2nd warning:

```
project1_FA21.c:56:  [4] (format) fprintf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification.
```

```
main()
{
 int y=10;
 int a[10];

func4("foooooooooooooooooooooooooooooooooooooooooooooooooo");

{try{func3();}
catch(char *message)
{    fprintf(stderr, message);}
;
fprintf(aFile, "%s", "hello world")

     while (y>=0)
     {    a[y]=y;
          y=y-1;

     }
     return 0;

}
```

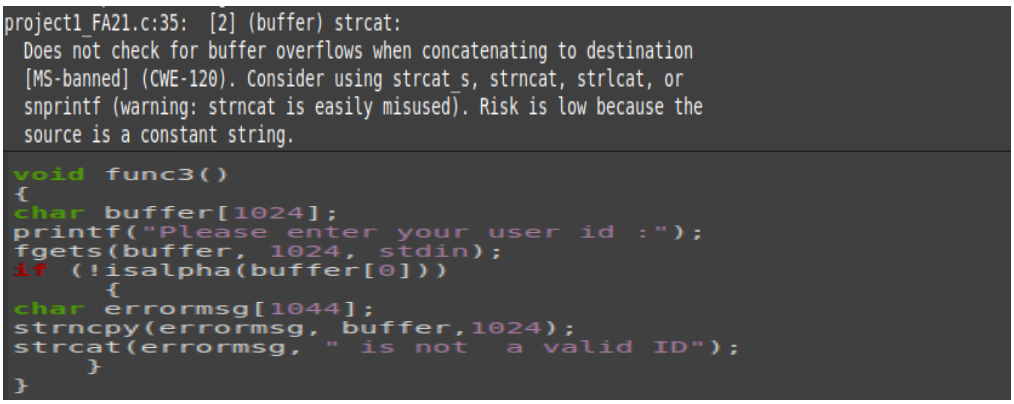
This is about first fprintf usage without a format string operator. If the message contains operators such as %s,%n,%x exc., it causes sensitive data or stack overflow and so data and code are mixed. So I add “%s” as the second parameter.

3rd, 4th, 5th warnings:



Usage of statically-sized arrays that have a fixed size and so for example when we copy a larger buffer we got buffer overflow. It is a false positive, because we have not in this example the issue of a bigger array because we know all the sizes.

6th warning:



This is about the strcat(dest,src) usage. Is an issue because dest buffer must have size that contains the final concatenated string (and also '\0') and if not there is buffer overflow. I use the strlcat function to solve this problem.

7th, 9th, 10th warnings:

```
project1_FA21.c:8:  [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).
project1_FA21.c:9:  [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).
project1_FA21.c:10:  [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).

void func1(char *src)
{
char dst[(strlen(src) + 1) * sizeof(char)];
strncpy(dst, src, strlen(src) + sizeof(char));
dst[strlen(dst)] = 0;
}
```

Is about strlen(). I replaced the last two with sizeof for clarity reasons and the first if we put a maximum size we can mitigate buffer overflows, but when we got a not null terminated string we got an undefined behaviour.

8th, 13th warnings:

```
project1_FA21.c:9:  [1] (buffer) strncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).

void func1(char *src)
{
char dst[(strlen(src) + 1) * sizeof(char)];
strncpy(dst, src, strlen(src) + sizeof(char));
dst[strlen(dst)] = 0;
}
project1_FA21.c:34:  [1] (buffer) strncpy:
  Easily used incorrectly; doesn't always \0-terminate or check for invalid
  pointers [MS-banned] (CWE-120).
void func3()
{
char buffer[1024];
printf("Please enter your user id :");
fgets(buffer, 1024, stdin);
if (!isalpha(buffer[0]))
        {
char errormsg[1044];
strncpy(errormsg, buffer,1024);
strcat(errormsg, " is not  a valid ID");
    }
}
```

Is about strncpy() usage that expect dst as a null-terminated string and if it isn't it causes buffer overflow. The first is a false positive because when we put sizeof correctly we got a null terminated string, the second one I prefer to use strlcpy.

11th, 12th warnings:

```
project1_FA21.c:17:  [1] (buffer) read:
  Check buffer boundaries if used in a loop including recursive loops
  (CWE-120, CWE-20).
project1_FA21.c:22:  [1] (buffer) read:
  Check buffer boundaries if used in a loop including recursive loops
  (CWE-120, CWE-20).
```

```
void func2(int fd)
{
char *buf;
size_t len;
read(fd, &len, sizeof(len));

if (len > 1024)
return;
buf = malloc(len+1);
read(fd, buf, len);
buf[len] = '\0';
}
```

Is about read() and it can be exploited when the len is negative and it is a false positive. In this case we got size_t that is an unsigned type, also the maximum value problem is mitigated because there is the if and also buf is \0-terminated.

4 Checking

In this paragraph I discuss if there are vulnerabilities that the tool cannot find. One example of a vulnerability not found is on the line 61.

```
main()
{
 int y=10;
 int a[10];

func4("fooooooooooooooooooooooooooooooooooooooooooooooooooooooooo");

{try{func3();}
catch(char *message)
{    fprintf(stderr, message);}
;
fprintf(aFile, "%s", "hello world")

    while (y>=0)
    {    a[y]=y;
         y=y-1;

    }
    return 0;

}
```

Inside the while we have “a[y] = y”, that it’s a vulnerability. Because y = 10, a is an array of 10 elements (a[0],…,a[9]) and a[y] (which is in the first case a[10]) access outside memory (Buffer Overflow) and so we need to change with “a[y -1]” and also the while condition with > 0.

5 Correction

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

#define MAX_SIZE 65535

void func1(char *src)
{
if (src == NULL)
      return;
size_t len = strlen(src); // flawfinder: ignore
if (len > MAX_SIZE - 1)
      return;
char dst[(len + 1) * sizeof(char)]; // flawfinder: ignore
strncpy(dst, src, sizeof(dst) - 1); // flawfinder: ignore
dst[sizeof(dst) - 1] = '\0';
}

void func2(int fd)
{
char *buf;
size_t len;
read(fd, &len, sizeof(len)); // flawfinder: ignore

if (len > 1024) // standard maximum size is bigger than this so it's ok
return;
buf = malloc(len+1);
read(fd, buf, len); // flawfinder: ignore
buf[len] = '\0';
}
void func3()
{
char buffer[1024]; // flawfinder: ignore
printf("Please enter your user id :");
fgets(buffer, 1024, stdin);
if (!isalpha(buffer[0]))
     {
char errormsg[1044]; // flawfinder: ignore
strlcpy(errormsg, buffer, sizeof(errormsg));
strlcat(errormsg, " is not  a valid ID", sizeof(errormsg)); // third argument is the sizeof the destination array (not like strncat)
    }
}

void func4(char *foo)
{
char buffer[10]; // flawfinder: ignore
strncpy(buffer, foo, sizeof(buffer) - 1); // flawfinder: ignore
buffer[sizeof(buffer) - 1] = '\0';
}


main()
{
 int y=10;
 int a[10];

func4("fooooooooooooooooooooooooooooooooooooooooooooooooooo");

{try{func3();}
catch(char *message)
{    fprintf(stderr, "%s", message);}
;
fprintf(aFile, "%s", "hello world");

      while (y>0)
      {    a[y - 1]=y;
           y=y-1;

      }
      return 0;

}
```

```
giovanni@giovanni-VirtualBox:~/Documenti/Projects/SA/1st$ flawfinder project1_FA
21.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 project1_FA21.c

FINAL RESULTS:


ANALYSIS SUMMARY:

No hits found.
Lines analyzed = 75 in approximately 0.04 seconds (1961 lines/second)
Physical Source Lines of Code (SLOC) = 60
Hits@level = [0]   3 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   3 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+]  50 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Suppressed hits = 9 (use --neverignore to show them)
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.
giovanni@giovanni-VirtualBox:~/Documenti/Projects/SA/1st$
```