CS628 Assignment 2a – Vulnerabilities & Countermeasures
Souvik Mukherjee
Roll Number: 231110405

1. Functions and their potential vulnerabilities categorization

Q.No	Function	Buffer Overflow	Integer Overflow	Format String vulnerability
1a(i)	function1()	No	No	Yes
1a(i)	function2()	No	No	No
1a(i)	function3()	No	No	Yes
1a(i)	function4()	No	No	No
1a(i)	function5()	No	No	No
1a(i)	main()	No	No	No
1a(ii)	copy()	Yes (Stack)	No	No
1a(ii)	concatenation()	Yes (Heap)	No	Yes
1a(ii)	main()	Yes (Heap)	Yes	No

2. Problems, Vulnerable line, exploitation & countermeasures in "Q_1a_i.c"

2.1. function1()

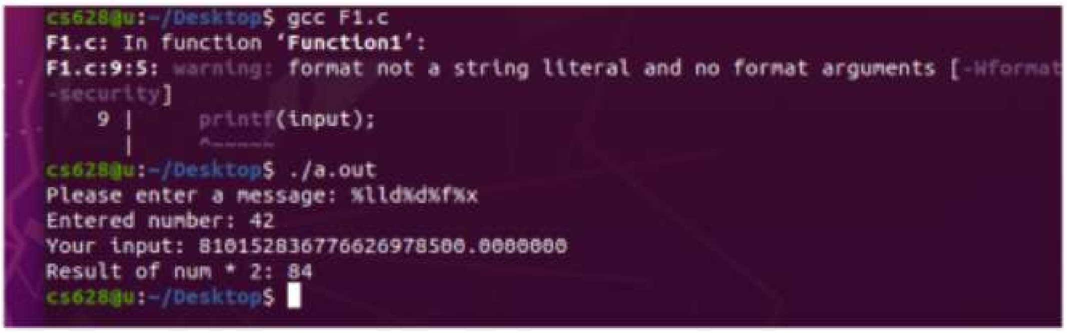
2.1.1. Problem, Vulnerable Lines, exploitation & countermeasures

Buffer Overflow (BO): The Function1() restricts bufferoverflow by using a pointer to a specified fixed dimension array defined in main, the array is constrained to be of dimension [100] and also in main, scanf "%99s" is used, so buffer of main cannot overflow, and the value is passed by reference, since buffer of main is out of scope of function1() to overflow, we do not have the problem of bufferoverflow here.

Format String (FS): Function1() is using "printf(input);" in which we do not have any format specifiers, hence is vulnerable to leaking information stored in the registers. ***Hence, format string attack is possible in this context. (Even compiler is showing warning)***

Exploit: An attacker can leak sensitive information, or codes by passing format specifier in the input itself. We can use **"%s" to jump to another location**, if that location is out of our access permission, the **program can crash**. We can use **"%d, %x, %lld", etc to print values from stack**.With proper calculation and payload we can also **use "%n" to overwrite a stack value.**

<u>Avoidance:</u> Use the “%s” format specifier, Ex: Use “printf(“Your inpur %s”, input);”



```
cs628@u:~/Desktop$ gcc F1.c
F1.c: In function ‘Function1’:
F1.c:9:5: warning: format not a string literal and no format arguments [-Wformat
-security]
    9 |     printf(input);
      |     ^~~~~~
cs628@u:~/Desktop$ ./a.out
Please enter a message: %lld%d%f%x
Entered number: 42
Your input: 8101528367766269785OO.OOOOOOO
Result of num * 2: 84
cs628@u:~/Desktop$ 
```

<u>Integer Overflow (IO):</u> If we can enter a number in the variable “number” which in this case is predefined as 42, we could have encounterd an integer overflow.
In this scenario, for a successful integer overflow, we would have to give a number, such that when we do (num*2) we exceed the arcitecture limit of integer (4 Byte)
[ex: 0111 (max positive number of a 4 bit arcitecture), when we do +1, we get 1000, which is the largest negative number, so while doing (7+1), insted of 8, we’re getting (-8), which is a integer overflow.]. So, here we do not have any IO.

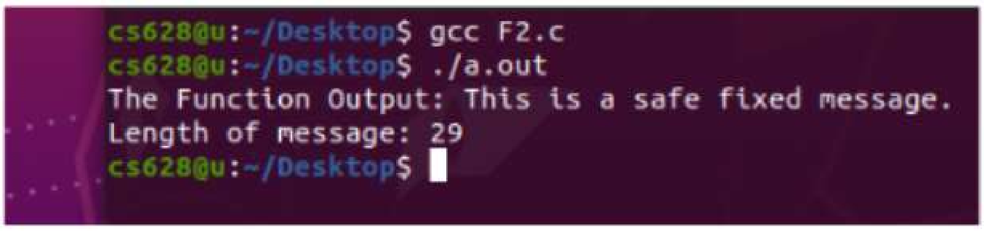
2.2. function2()

2.2.1. Problem, Vulnerable Lines, exploitation & countermeasures

<u>BO:</u> As we are not taking any input from user, in the buffer (fixedMessage), there is no scope of bufferoverflow here. The (fixedMessage) buffer is predefined with a small string.

<u>FS:</u> There is proper use of format specifier, so there is no scope of format string error, and even if there was no format specifier “%s”, here, there would have been no issue, as string is not taken as input from user, no one can pass format specifers as input in this scenario.

<u>IO:</u> There is no scope for integer overflow, it is only possible here, if we pass an input so huge that it exceeds the capacity of integer. But, here we are giving a fixed input of small size.



```
cs628@u:~/Desktop$ gcc F2.c
cs628@u:~/Desktop$ ./a.out
The Function Output: This is a safe fixed message.
Length of message: 29
cs628@u:~/Desktop$ 
```

2.3. function3()

2.3.1. Problem, Vulnerable Lines, exploitation & countermeasures

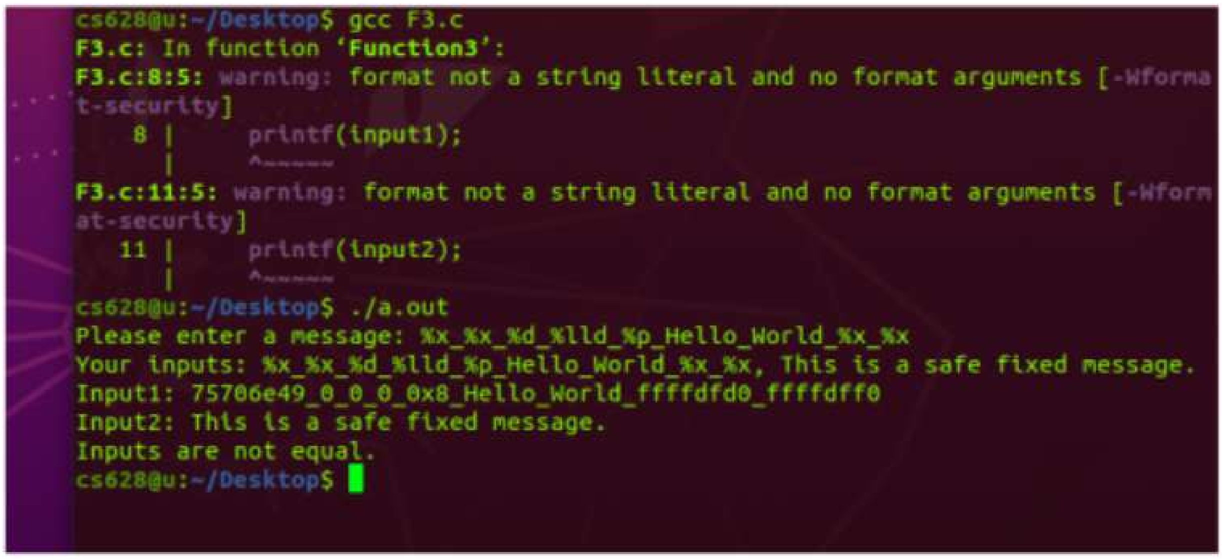
<u>BO:</u> The Function3() restricts bufferoverflow by using a pointer to a specified fixed dimension array defined in main, the array is constrained to be of dimension [100] and also in main, scanf “%99s” is used, so buffer of main cannot overflow, and the value is passed by reference, since buffer of main is out of scope of function3() to overflow, we do not have the problem of bufferoverflow here.

As we are not taking any input from user, in the buffer (fixedMessage), there is no scope of bufferoverflow here. The (fixedMessage) buffer is predefined with a small string.
So, we do not have any buffer overflow.

FS: Function3() is using "printf(input1);" in which we do not have any format specifiers, hence is vulnerable to leaking information stored in the registers. ***Hence, format string attack is possible in this context. (Even compiler is showing warning)***

The "printf(input2)" has no scope of FS, even though we dont have any format specifier here, as we are having a fixed input, and we as user cannot pass any input here.
Exploit: An attacker can leak sensitive information, or codes by passing format specifier in the input itself. We can use "**%s" to jump to another location**, if that location is out of our access permission, the **program can crash**. We can use **"%d, %x, %lld", etc to print values from stack**.With proper calculation and payload we can also **use "%n" to overwrite a stack value.**

Avoidance: Use the "%s" format specifier, Ex: Use "printf("Your inpur %s", input1);"



```
cs628@u:~/Desktop$ gcc F3.c
F3.c: In function 'Function3':
F3.c:8:5: warning: format not a string literal and no format arguments [-Wforma
t-security]
    8 |     printf(input1);
      |     ^~~~~~
F3.c:11:5: warning: format not a string literal and no format arguments [-Wform
at-security]
   11 |     printf(input2);
      |     ^~~~~~
cs628@u:~/Desktop$ ./a.out
Please enter a message: %x_%x_%d_%lld_%p_Hello_World_%x_%x
Your inputs: %x_%x_%d_%lld_%p_Hello_World_%x_%x, This is a safe fixed message.
Input1: 75706e49_0_0_0_0x8_Hello_World_ffffdfd0_ffffdff0
Input2: This is a safe fixed message.
Inputs are not equal.
cs628@u:~/Desktop$
```

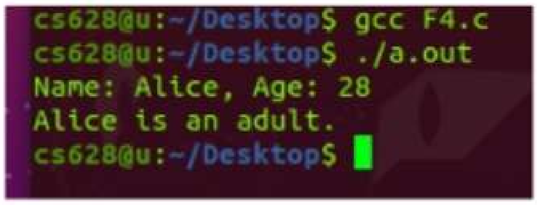
IO: There is no scope for integer overflow here, as we do not have any integer in our scope.

2.4. function4()

2.4.1. Problem, Vulnerable Lines, exploitation & countermeasures

BO & FS & IO: There is no scope for any kind of vulnerabilities in the code, as we are pre-inserting the inputs within the function call "Function4("Alice", 28);" and we are also using proper format specifiers in the function declaration, i.e. "%s" and "%d".

- No function like strcpy for any BO.
- Proper use of format specifiers and no inputs taken from user, so no scope of FS.
- We have only one integer "28" which is not undergoing any arithmetic operation, so no IO.



```
cs628@u:~/Desktop$ gcc F4.c
cs628@u:~/Desktop$ ./a.out
Name: Alice, Age: 28
Alice is an adult.
cs628@u:~/Desktop$ 
```

2.5. function5()

2.5.1. Problem, Vulnerable Lines, exploitation & countermeasures

We are placing ("Received: " + userProvidedInput) in buffer[100]; using snprintf, which is safe to buffer overflow attack, and we are reversing it using the entered "userProvidedInput", accessed by the pointer char array "input", and we are storing the reversed string in "reversed" array.

What Function5() is doing, INSHORT

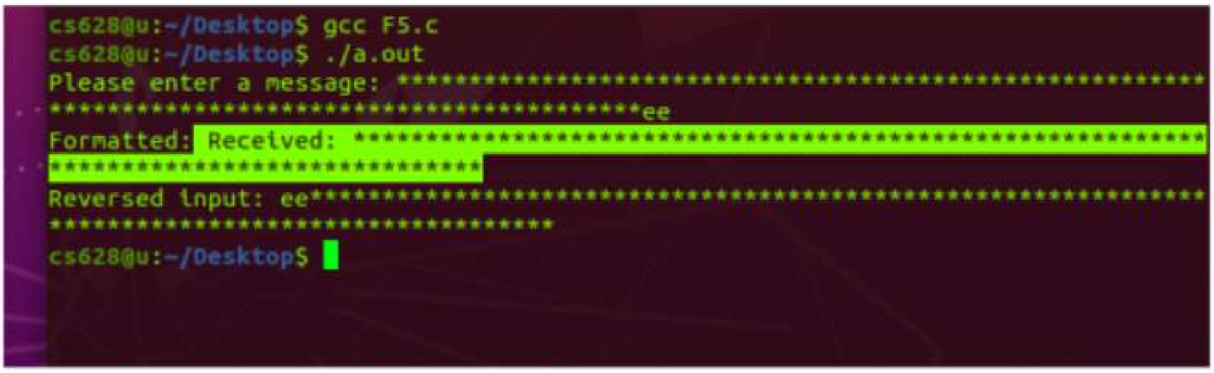
The string "Received" and the entered input (userProvidedInput) is stored in the buffer[100], and this is displayed in the o/p as "Formatted: **Received*******"**
>> snprintf(buffer, sizeof(buffer), "Received: %s", input);
("Received" is placed inside the "buffer array", using snprintf).

And, the reversed array is populated using the entered input only, neglecting the "Received" string. The o/p is displayed as "Reversed input: ******"
(There is no "Received" here, in the reversed array)

BO: As we are restricting input from user, using "%99" in the main and the size of "userProvidedInput" is also [100], we cannot pass an input exceeding the size of 99 to function5(). Function5() is also using a "buffer" of size [100], so here also we are having restrictions, so any kind of buffer overflow is not possible in this scope.

FS: There is proper use of format specifier, so there is no scope of format string error, and If there was no format specifier "%s", here, then there would have been issue, as string is taken as input from user.

IO: There is no scope for integer overflow, it is only possible here, if we pass an input string "userProvidedInput" so huge that it exceeds the capacity of integer. But, here we are restricting the input size, in main(), by using "%99s". So there is no scope of integer overflow here.



```
cs628@u:~/Desktop$ gcc F5.c
cs628@u:~/Desktop$ ./a.out
Please enter a message: ********************************************************
*****************************************ee
Formatted: Received: ***********************************************************
******************************
Reversed input: ee**************************************************************
***********************************
cs628@u:~/Desktop$ 
```

2.6. main()

2.6.1. Problem, Vulnerable Lines, exploitation & countermeasures

BO & FS & IO: There is no scope for any kind of vulnerabilities in the main().

- No function like strcpy for any BO.
- We are using proper format specifier during scan "scanf("%99s", userProvidedInput);" and we are not printing any variable, so do not need any format specifier while printing, so no scope of any FS.
- We have only one integer declaration "int number = 42;" which is not undergoing any arithmetic operation, so no IO.

***** Thank You *****

***** Next Question *****

3. Problems, Vulnerable line, exploitation & countermeasures in "Q_1a_ii.c"

3.1. copy()

3.1.1. Problem, Vulnerable Lines, exploitation & countermeasures

We have a global variable named "secret", placed in heap. We are placing a random value in it, on which 'bitwise AND' is applied with "0xffff". So by doing this we are restricting the overflow of integer on the "secret" variable.

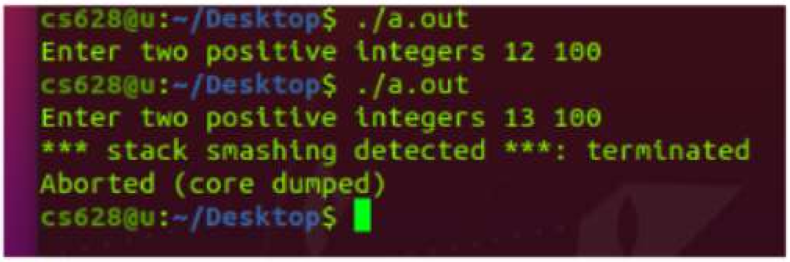
Buffer Overflow (BO): We are accepting a integer with proper format specifier in main, named as "Num1", so max size of this integer will be "INT_MAX". Now we are allocating memory during runtime(heap allocation) to a string named as "str", we are allocating memory as long as "Num1*sizeOf(char)".
Now we are inserting stuffs from a file named "file" into this string using "fread". The str can be atmost as big as "Num1*sizeOf(char)".

We are accepting this str inthe function "copy(char *str)" and we have an array named "buffer[12]" in copy, and, we also have a variable named "guard", here in copy. This guard is copying the global variable "secret" in the scope of "copy() function".

We are copying the content of "str" in "buffer" using "strcpy(buffer, str)" which is a dangereous function.
*So if we somehow pass a value greater than 12 in string, we will overflow the buffer and change the"guard" variable, which will in turn fail the "if function" and the program will exit with an exception of "*** stack smashing detected ***:"*

So, we do have a buffer overflow here in the stack region of copy()

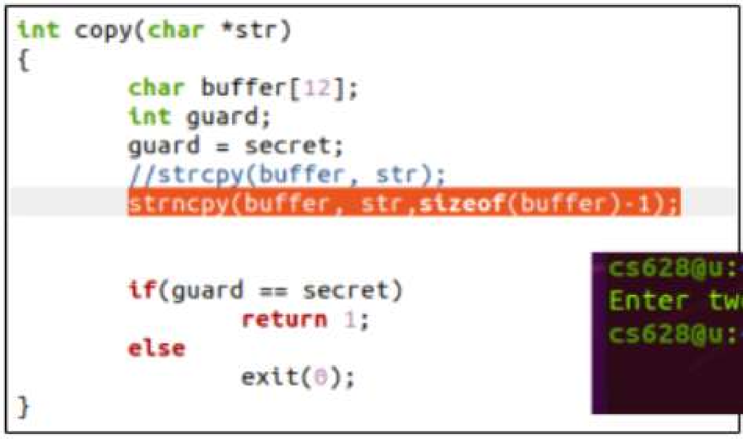
In the diagram below, I have intentionally kept the value of (Num1+Num2 >100), as I do not want to include the concardination() function in this subpart of copy().



```
cs628@u:~/Desktop$ ./a.out
Enter two positive integers 12 100
cs628@u:~/Desktop$ ./a.out
Enter two positive integers 13 100
*** stack smashing detected ***: terminated
Aborted (core dumped)
cs628@u:~/Desktop$
```

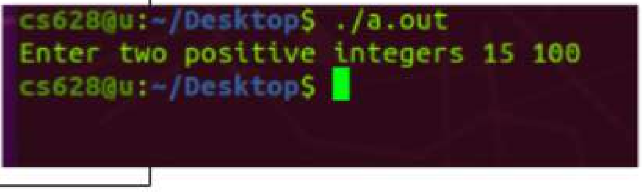
Exploit: An attacker can update the return address, with proper calculation, and run malicious codes, or escalate our privilidges by opening a root shell, or can modify stack data, according to his/her needs.

Avoidance: *We can use "**strncpy(buffer, str, size(buffer)-1)**;" instead of "strcpy(buffer, str);"*



```
int copy(char *str)
{
        char buffer[12];
        int guard;
        guard = secret;
        //strcpy(buffer, str);
        strncpy(buffer, str,sizeof(buffer)-1);


        if(guard == secret)
                return 1;
        else
                exit(0);
}
```



```
cs628@u:~/Desktop$ ./a.out
Enter two positive integers 15 100
cs628@u:~/Desktop$
```

Format String (FS): Copy() has no printf or scanf statement, so even if we had passed "%x","%n" via the "file" and through the "str" to copy() function, we cannot exploit, here in copy function. So, there is no Format String problem here.

Integer Overflow (IO): There is no scope for integer overflow here, as we do not have any integer in our scope. The only integer we have here is guard, which is copying it's value from secret, and as discussed earlier, secret is limited with 'bitwise AND' applied with "0xffff".

3.2. concatenation()

3.2.1. Problem, Vulnerable Lines, exploitation & countermeasures

BO: We are using "strcat(str, buffer)", and our destination is in the heap region, and is unbounded. **So, there is a potential buffer overlow in the heap region.**

Exploit: An attcker may update the value of the "secret" value present in heap, or it may also update other sensitive informations in the heap. If we can overflow way too much, we can also breakfree heap and reach stack, and then update some return address of some function, to run some malicious code.

Avoidance: We can fix things, by checking wheather there is enough space to accomodate the concartination of str and buffer in str. We can use a if statement.

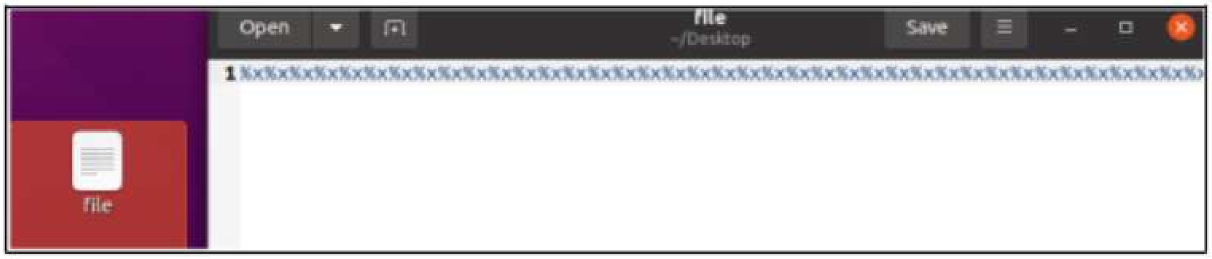
*** if(strlen(str)+strlen(buffer) < sixeOf(str)) { We'll perform strcat} else {we'll throw an error, saying, we do not have enough space}**

*** or we can simply use "strncat(str, buffer,sizeOf(str));"**

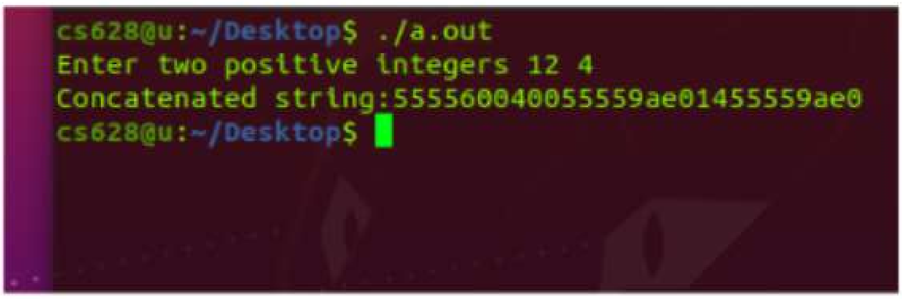
Also, sizeOf(str)==Num1

IO: There is no scope for integer overflow here, as we do not have any integer in our scope.

FS: We are having "printf(str);" which can cause problems, if we keep format specifiers in the "file", which will then get copied to the "str" and will print out with respect to the format specifiers. **So, format string attack/vulnerability is possible here.**



```
1 %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%
```

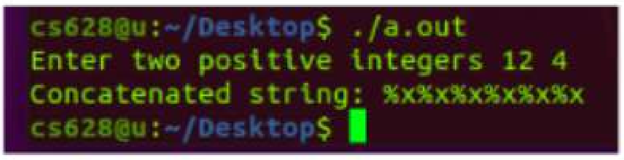


```
cs628@u:~/Desktop$ ./a.out
Enter two positive integers 12 4
Concatenated string:55556004005559ae01455559ae0
cs628@u:~/Desktop$ 
```

Exploit: An attacker can leak sensitive information, or codes by passing format specifier in the 'file', which will get copied to the 'str', and we are printing 'str' without any format specifiers.

We can use **"%s" to jump to another location**, if that location is out of our access permission, the **program can crash**. We can use **"%d, %x, %lld", etc to print values from stack**.With proper calculation and payload we can also **use "%n" to overwrite a stack value.**

Avoidance: Use the "%s" format specifier.
Ex: Use "printf("Concatenated string: %s", str);"



```
cs628@u:~/Desktop$ ./a.out
Enter two positive integers 12 4
Concatenated string: %x%x%x%x%x%x%x
cs628@u:~/Desktop$ 
```

3.3. main()

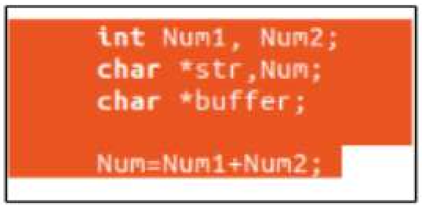
3.3.1. Problem, Vulnerable Lines, exploitation & countermeasures

FS: There is no scope for any kind of FS vulnerabilities in the code, as we are taking proper integer inputs here, with proper format specifiers, and creating dynamic runtime memory in the heap, to store strings of a size of the magnitude of the integer. We are the copying the content of a file to the 'str' string, and passing it to 'copy()' function. We are then passing 'buffer' and 'str' to the concardination() function, if Num1+Num2<100.

- No function like strcpy for any BO.
- Proper use of format specifiers and no inputs taken from user, so no scope of FS.

IO: **There is a scope for a inetger overflow here.**

We are taking two integers Num1 and Num2, and storing their sum in a "char Num", so, whenever we cross the physical arcitecture of a char, i.e. 256 bits, we'll be getting an overflow and the number (Num) will be updated to zero and above.



```
int Num1, Num2;
char *str,Num;
char *buffer;

Num=Num1+Num2;
```

Exploit: We dont get expected results
(simply, in Num, we get (Num1+Num2)%256)

Ex. if, Num1+Num2 = 255. Num will be 255, Num: 1 1 1 1 1 1 1 1

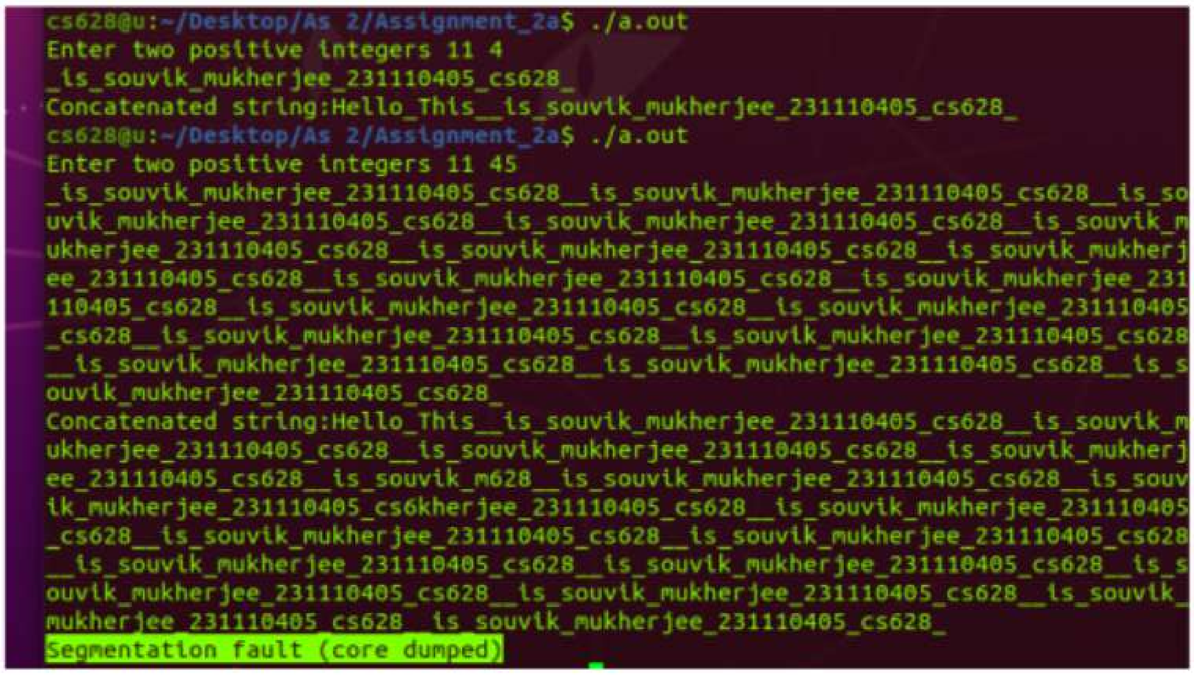
Ex. if, Num1+Num2 = 256. Num will be 0, Num: 1 0 0 0 0 0 0 0 0

Avoidence: Use data types like "long, long long, long long int, etc" when we suspect that our mathematical operation may cause an overflow.

BO: We are using "***gets(buffer);***", in main(), which is a dangereous function. The size of buffer is pre-defined as "Num2", now if user input exceeds this size, we'll have an overflow.

So, there is a potential buffer overlow here.

Here, I am taking str as a small value (11B) and passing a huge value in buffer.



```
cs628@u:~/Desktop/As 2/Assignment_2a$ ./a.out
Enter two positive integers 11 4
_is_souvik_mukherjee_231110405_cs628_
Concatenated string:Hello_This__is_souvik_mukherjee_231110405_cs628_
cs628@u:~/Desktop/As 2/Assignment_2a$ ./a.out
Enter two positive integers 11 45
_is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_so
uvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_m
ukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherj
ee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231
110405_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405
_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628
__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_s
ouvik_mukherjee_231110405_cs628_
Concatenated string:Hello_This__is_souvik_mukherjee_231110405_cs628__is_souvik_m
ukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherj
ee_231110405_cs628__is_souvik_m628__is_souvik_mukherjee_231110405_cs628__is_souv
ik_mukherjee_231110405_cs6kherjee_231110405_cs628__is_souvik_mukherjee_231110405
_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628
__is_souvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_s
ouvik_mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628__is_souvik_
mukherjee_231110405_cs628__is_souvik_mukherjee_231110405_cs628_
Segmentation fault (core dumped)
```

Exploit: An attcker may update the value of the "secret" value present in heap, or it may also update other sensitive informations in the heap. If we can overflow way too much, we can also breakfree heap and reach stack, and then update some return address of some function, to run some malicious code.

Avoidance: We can fix things, by using "fgets or getline" instead of "gets".

***** Thank You *****
