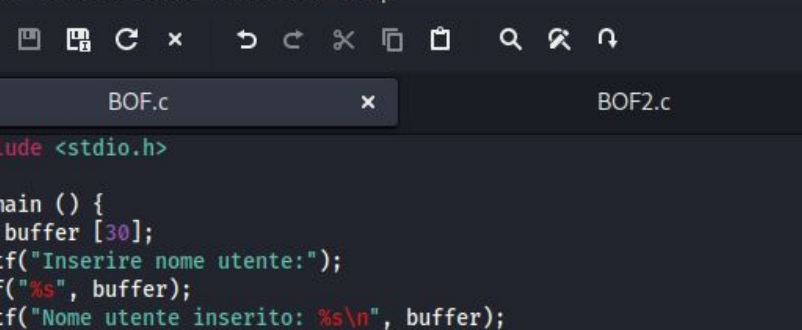




# Esercizio S7 L4



La dimensione dell'array è stata aumentata a 30, tuttavia senza alcun tipo di controllo sull'input è ancora possibile causare un errore del tipo "segmentation fault" inserendo una stringa ben più lunga di 30 elementi.




The image shows a Kali Linux desktop environment. In the foreground, a terminal window is open, displaying the execution of a C program. The prompt is `(kali㉿kali)-[~/Desktop]`. The user enters `./BOF`, and the program prompts for a username. The user enters `fiwfifcdnis`, and the program prints `Nome utente inserito: fiwfifcdnis`. The user then enters a long string of characters, causing a segmentation fault. The terminal output is as follows:

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Inserire nome utente:fiwfifcdnis
Nome utente inserito: fiwfifcdnis
Inserire nome utente:ruhfrbwyfgyiufgrwefiyrfgfw7egy7rfwe7gfrq97rfg9g7wefy9fr7gwqe69
Nome utente inserito: ruhfrbwyfgyiufgrwefiyrfgfw7egy7rfwe7gfrq97rfg9g7wefy9fr7gwqe69
zsh: segmentation fault ./BOF
```

In the background, a code editor window is open, showing the source code of the `BOF.c` file. The code is as follows:

```
#include <stdio.h>


int main () {
    char buffer [30];
    printf("Inserire nome utente:");
    scanf("%s", buffer);
    printf("Nome utente inserito: %s\n", buffer);
    return 0;
}
```



Il codice qua in figura implementa una serie di controlli sull'input per prevenire errori del tipo "segmentation fault". Adesso la dimensione dell'array viene definita solo dopo che l'utente dà il suo input, se l'input dell'utente è lungo 20 o meno caratteri il programma creerà un array che contenga l'input per intero e la cui dimensione corrisponderà con quella dell'input. Se l'utente dà un input lungo più di 20 caratteri il programma scarterà tutti i caratteri inseriti dall'utente dopo il ventesimo. Inoltre restituisce per ogni carattere nell'array il suo indirizzo di memoria e in più restituisce l'indirizzo di memoria dei primi 10 spazi dopo il nostro array.

```
File Edit Search View Document Help
[Icons]

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     char *inputArray; // Dynamic array to store user input
6     int length, i;
7
8     // Prompt user for input
9     printf("Enter a string: ");
10    scanf("%m[^\n]", &inputArray); // Read the input until newline
11
12    // Calculate the length of the input
13    length = 0;
14    while (inputArray[length] != '\0') {
15        length++;
16    }
17
18    // Adjust the length to be at most 20 characters
19    if (length > 20) {
20        length = 20;
21        inputArray[20] = '\0'; // Null-terminate the string at the 21st character
22    }
23
24    // Output each character along with its memory address
25    printf("\nOutputting characters along with their memory addresses:\n");
26    for (i = 0; i < length; i++) {
27        printf("Character: %c, Memory Address: %p\n", inputArray[i], (void*)&inputArray[i]);
28    }
29
30    // Output memory addresses of the first 10 memory slots after the array
31    printf("\nMemory addresses of the first 10 slots after the array:\n");
32    for (i = 0; i < 10; i++) {
33        printf("Memory Address: %p\n", (void*)&inputArray[length + i]);
34    }
35
36    // Free the dynamically allocated memory
37    free(inputArray);
38
39    return 0;
40 }
41
```



Possiamo vedere il codice di prima in esecuzione, in questo caso è stato fornito un input di lunghezza 29 caratteri, come osserviamo il codice considera solamente i primi 20 senza andare in segmentation fault, inoltre possiamo vedere che gli indirizzi di memoria per i vari caratteri che compongono il nostro input sono adiacenti.

Più sotto sono riportati gli indirizzi di memoria dei 10 spazi immediatamente successivi a quelli occupati dal nostro array, se non avessimo implementato i controlli sull'input questa memoria verrebbe sovrascritta col resto dell'input e potrebbe causare comportamenti inaspettati oltre che ad errori di tipo segmentation fault.

```
(kali@kali)-[~/Desktop]
$ ./BOF2
Enter a string: PauraDiTuttiRispettoDiNessuno

Outputting characters along with their memory addresses:
Character: P, Memory Address: 0x56126a6006b0
Character: a, Memory Address: 0x56126a6006b1
Character: u, Memory Address: 0x56126a6006b2
Character: r, Memory Address: 0x56126a6006b3
Character: a, Memory Address: 0x56126a6006b4
Character: D, Memory Address: 0x56126a6006b5
Character: i, Memory Address: 0x56126a6006b6
Character: T, Memory Address: 0x56126a6006b7
Character: u, Memory Address: 0x56126a6006b8
Character: t, Memory Address: 0x56126a6006b9
Character: t, Memory Address: 0x56126a6006ba
Character: i, Memory Address: 0x56126a6006bb
Character: R, Memory Address: 0x56126a6006bc
Character: i, Memory Address: 0x56126a6006bd
Character: s, Memory Address: 0x56126a6006be
Character: p, Memory Address: 0x56126a6006bf
Character: e, Memory Address: 0x56126a6006c0
Character: t, Memory Address: 0x56126a6006c1
Character: t, Memory Address: 0x56126a6006c2
Character: o, Memory Address: 0x56126a6006c3

Memory addresses of the first 10 slots after the array:
Memory Address: 0x56126a6006c4
Memory Address: 0x56126a6006c5
Memory Address: 0x56126a6006c6
Memory Address: 0x56126a6006c7
Memory Address: 0x56126a6006c8
Memory Address: 0x56126a6006c9
Memory Address: 0x56126a6006ca
Memory Address: 0x56126a6006cb
Memory Address: 0x56126a6006cc
Memory Address: 0x56126a6006cd
```