

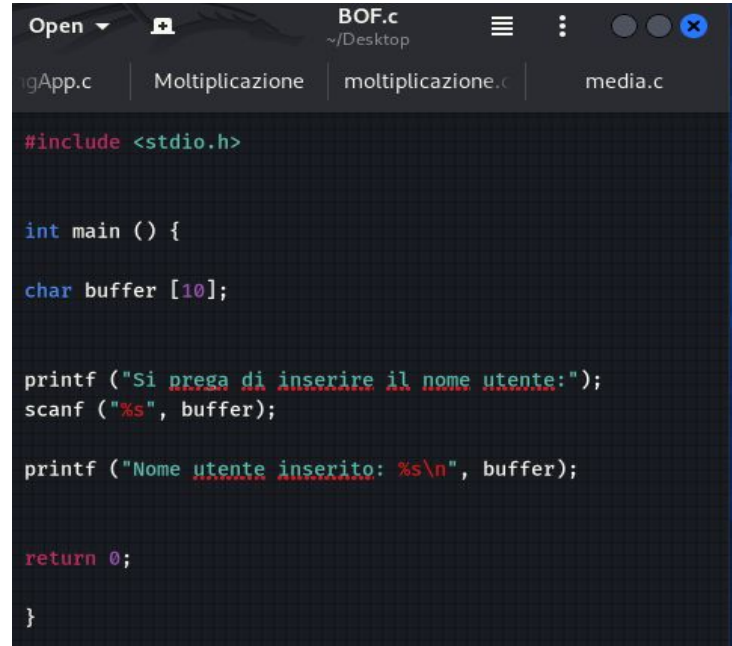


S7/L4 Epicode Cybersecurity

Buffer Overflow



Nell'esercizio di oggi andiamo a testare un codice in linguaggio C per capire meglio il buffer overflow. Per prima cosa andiamo a creare un file contenente il nostro codice.

A screenshot of a code editor window titled "BOF.c" with a path of "~/Desktop". The editor has a dark theme and shows a C program. The code includes a header file, a main function, a character array, and two printf/scanf statements. The file explorer on the left shows other files like "igApp.c", "Moltiplicazione", "moltiplicazione.c", and "media.c".

```
Open ▾ + BOF.c ~/Desktop
igApp.c | Moltiplicazione | moltiplicazione.c | media.c

#include <stdio.h>

int main () {
    char buffer [10];

    printf ("Si prega di inserire il nome utente:");
    scanf ("%s", buffer);

    printf ("Nome utente inserito: %s\n", buffer);

    return 0;
}
```

Fatto questo, andiamo a compilare il codice ed eseguirlo, inserendo un nome più lungo del consentito per ricevere l'errore "segmentation fault".

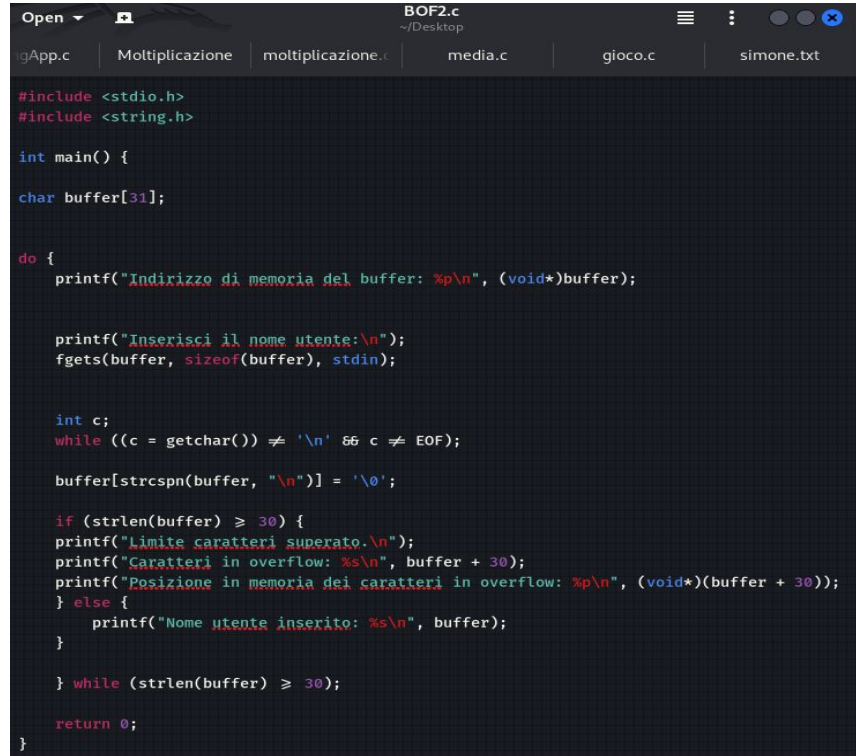
```
(kali㉿kali)-[~/Desktop]
$ gcc -o BOF BOF.c

(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:ciaociaociaociaociao
Nome utente inserito: ciaociaociaociaociao
zsh: segmentation fault ./BOF
```

Per completezza, eseguiamo di nuovo il codice e inseriamo un nome che rispetti la lunghezza massima consentita.

```
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:benvenuto
Nome utente inserito: benvenuto
```

Adesso creiamo un secondo codice, utilizzando il primo come base, che soddisfi i requisiti dell'esercizio. Inseriamo quindi la libreria `<string.h>`, che contiene molte dichiarazioni di funzioni, aumentiamo la quantità di caratteri a 31, aggiungiamo delle stringhe che, durante l'esecuzione, ci mostrino l'indirizzo di memoria del buffer e, eventualmente, anche quello di overflow.



```
BOF2.c
~/Desktop

gApp.c | Moltiplicazione | moltiplicazione.c | media.c | gioco.c | simone.txt

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

int main() {
    char buffer[31];

    do {
        printf("Indirizzo di memoria del buffer: %p\n", (void*)buffer);

        printf("Inserisci il nome utente:\n");
        fgets(buffer, sizeof(buffer), stdin);

        int c;
        while ((c = getchar()) != '\n' && c != EOF);

        buffer[strcspn(buffer, "\n")] = '\0';

        if (strlen(buffer) >= 30) {
            printf("Limite caratteri superato.\n");
            printf("Caratteri in overflow: %s\n", buffer + 30);
            printf("Posizione in memoria dei caratteri in overflow: %p\n", (void*)(buffer + 30));
        } else {
            printf("Nome utente inserito: %s\n", buffer);
        }
    } while (strlen(buffer) >= 30);

    return 0;
}
```

Compiliamo quindi il codice e lo eseguiamo. Possiamo vedere come, adesso, venga mostrato un indirizzo di memoria, non sia più presente il “segmentation fault” ma appaia un messaggio di errore e, come ultima modifica, venga mostrato l'indirizzo di memoria dei caratteri che vanno in overflow.

```
(kali@kali)-[~/Desktop]
$ ./BOF2
Indirizzo di memoria del buffer: 0x7fff0e12bfd0
Inserisci il nome utente:
ciaociaociaociaociaociaociaociaociaociaociaociaociaociao
Limite caratteri superato.
Caratteri in overflow:
Posizione in memoria dei caratteri in overflow: 0x7fff0e12bfef
Indirizzo di memoria del buffer: 0x7fff0e12bfd0
Inserisci il nome utente:
sapijfwlklfslkdveòruhgwijfshfjksguòwrhglshdshfg
Limite caratteri superato.
Caratteri in overflow:
Posizione in memoria dei caratteri in overflow: 0x7fff0e12bfef
Indirizzo di memoria del buffer: 0x7fff0e12bfd0
Inserisci il nome utente:
```