

Pratica S7L4

Consideriamo il seguente programma:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main () {
5     char buffer [30];
6     int lunghezza_nome = 0;
7
8     printf("Inserire il nome utente: ");
9     scanf("%s", buffer);
10    lunghezza_nome = strlen(buffer); //assegno a lunghezza_nome la lunghezza del nome inserito dall'utente
11    if (lunghezza_nome > 30) {
12        for (int i=0; i < lunghezza_nome; i++) { //incremento i finché non ho raggiunto la lunghezza del nome
13            printf("indirizzo di memoria del carattere %c in posizione %d: %p\n", buffer[i], i, &buffer[i]);
14        }
15        printf("\n\nnome utente inserito (overflow): %s\n", buffer);
16    }
17    else {
18        printf("nome utente inserito: %s\n", buffer);
19    }
20 }
21
22 return 0;
23
24 }
25
26
```

Il programma accetta in input una stringa dall'utente di dimensione predefinita (30 caratteri). Se il nome inserito è minore di 30 caratteri restituisce a schermo il nome inserito. Vediamo un esempio:

```
(luca@kali)-[~/Desktop]
$ ./BOF_indirizzi
Inserire il nome utente: LucaDanelli
nome utente inserito: LucaDanelli
```

La situazione cambia se l'utente sceglie di inserire un input molto più lungo, che vada ad eccedere il buffer preallocato per la variabile buffer. Una volta dichiarata una variabile, C assegna uno spazio in memoria di dimensione predefinita, in questo caso andrà a riservare spazio per una stringa da 30 caratteri come da dichiarazione alla riga 5. Per chiarire meglio la situazione, il programma stampa gli indirizzi di memoria corrispondenti a ciascun carattere inserito dall'utente con un ciclo for (riga 11) servendosi della funzione **strlen()** in modo che esca dal ciclo una volta raggiunta la lunghezza della stringa inserita dall'utente. Vediamo i risultati andando a stampare una stringa di 33 caratteri:

```
(luca@kali)-[~/Desktop]
$ ./BOF_indirizzi
Inserire il nome utente: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
indirizzo di memoria del carattere a in posizione 0: 0x7ffe9ba429c0
indirizzo di memoria del carattere a in posizione 1: 0x7ffe9ba429c1
indirizzo di memoria del carattere a in posizione 2: 0x7ffe9ba429c2
indirizzo di memoria del carattere a in posizione 3: 0x7ffe9ba429c3
indirizzo di memoria del carattere a in posizione 4: 0x7ffe9ba429c4
indirizzo di memoria del carattere a in posizione 5: 0x7ffe9ba429c5
indirizzo di memoria del carattere a in posizione 6: 0x7ffe9ba429c6
indirizzo di memoria del carattere a in posizione 7: 0x7ffe9ba429c7
indirizzo di memoria del carattere a in posizione 8: 0x7ffe9ba429c8
indirizzo di memoria del carattere a in posizione 9: 0x7ffe9ba429c9
indirizzo di memoria del carattere a in posizione 10: 0x7ffe9ba429ca
indirizzo di memoria del carattere a in posizione 11: 0x7ffe9ba429cb
indirizzo di memoria del carattere a in posizione 12: 0x7ffe9ba429cc
indirizzo di memoria del carattere a in posizione 13: 0x7ffe9ba429cd
indirizzo di memoria del carattere a in posizione 14: 0x7ffe9ba429ce
indirizzo di memoria del carattere a in posizione 15: 0x7ffe9ba429cf
indirizzo di memoria del carattere a in posizione 16: 0x7ffe9ba429d0
indirizzo di memoria del carattere a in posizione 17: 0x7ffe9ba429d1
indirizzo di memoria del carattere a in posizione 18: 0x7ffe9ba429d2
indirizzo di memoria del carattere a in posizione 19: 0x7ffe9ba429d3
indirizzo di memoria del carattere a in posizione 20: 0x7ffe9ba429d4
indirizzo di memoria del carattere a in posizione 21: 0x7ffe9ba429d5
indirizzo di memoria del carattere a in posizione 22: 0x7ffe9ba429d6
indirizzo di memoria del carattere a in posizione 23: 0x7ffe9ba429d7
indirizzo di memoria del carattere a in posizione 24: 0x7ffe9ba429d8
indirizzo di memoria del carattere a in posizione 25: 0x7ffe9ba429d9
indirizzo di memoria del carattere a in posizione 26: 0x7ffe9ba429da
indirizzo di memoria del carattere a in posizione 27: 0x7ffe9ba429db
indirizzo di memoria del carattere a in posizione 28: 0x7ffe9ba429dc
indirizzo di memoria del carattere a in posizione 29: 0x7ffe9ba429dd
indirizzo di memoria del carattere a in posizione 30: 0x7ffe9ba429de
indirizzo di memoria del carattere a in posizione 31: 0x7ffe9ba429df
indirizzo di memoria del carattere a in posizione 32: 0x7ffe9ba429e0

nome utente inserito (overflow): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Come vediamo gli indirizzi in memoria di ogni carattere sono incrementali, ed espressi in esadecimale. Ci siamo serviti del puntatore **&buffer[i]** per accedere all'indirizzo di memoria all'indice **i** ad ogni iterazione del ciclo **for**. Per ora però il programma funziona ancora bene, nel senso che la stringa riportata in output è esattamente quella in input. Proviamo quindi ad aumentare ancora la lunghezza della stringa inserita dall'utente, portandola ad esempio a 62 caratteri:


```

indirizzo di memoria del carattere a in posizione 17: 0x7fff55c143c1
indirizzo di memoria del carattere a in posizione 18: 0x7fff55c143c2
indirizzo di memoria del carattere a in posizione 19: 0x7fff55c143c3
indirizzo di memoria del carattere a in posizione 20: 0x7fff55c143c4
indirizzo di memoria del carattere a in posizione 21: 0x7fff55c143c5
indirizzo di memoria del carattere a in posizione 22: 0x7fff55c143c6
indirizzo di memoria del carattere a in posizione 23: 0x7fff55c143c7
indirizzo di memoria del carattere a in posizione 24: 0x7fff55c143c8
indirizzo di memoria del carattere a in posizione 25: 0x7fff55c143c9
indirizzo di memoria del carattere a in posizione 26: 0x7fff55c143ca
indirizzo di memoria del carattere a in posizione 27: 0x7fff55c143cb
indirizzo di memoria del carattere a in posizione 28: 0x7fff55c143cc
indirizzo di memoria del carattere a in posizione 29: 0x7fff55c143cd
indirizzo di memoria del carattere a in posizione 30: 0x7fff55c143ce
indirizzo di memoria del carattere a in posizione 31: 0x7fff55c143cf
indirizzo di memoria del carattere a in posizione 32: 0x7fff55c143d0
indirizzo di memoria del carattere a in posizione 33: 0x7fff55c143d1
indirizzo di memoria del carattere a in posizione 34: 0x7fff55c143d2
indirizzo di memoria del carattere a in posizione 35: 0x7fff55c143d3
indirizzo di memoria del carattere a in posizione 36: 0x7fff55c143d4
indirizzo di memoria del carattere a in posizione 37: 0x7fff55c143d5
indirizzo di memoria del carattere a in posizione 38: 0x7fff55c143d6
indirizzo di memoria del carattere a in posizione 39: 0x7fff55c143d7
indirizzo di memoria del carattere > in posizione 40: 0x7fff55c143d8
indirizzo di memoria del carattere in posizione 41: 0x7fff55c143d9
indirizzo di memoria del carattere in posizione 42: 0x7fff55c143da
indirizzo di memoria del carattere in posizione 43: 0x7fff55c143db
indirizzo di memoria del carattere , in posizione 44: 0x7fff55c143dc
indirizzo di memoria del carattere in posizione 45: 0x7fff55c143dd
indirizzo di memoria del carattere in posizione 46: 0x7fff55c143de
indirizzo di memoria del carattere in posizione 47: 0x7fff55c143df
indirizzo di memoria del carattere a in posizione 48: 0x7fff55c143e0
indirizzo di memoria del carattere a in posizione 49: 0x7fff55c143e1
indirizzo di memoria del carattere a in posizione 50: 0x7fff55c143e2
indirizzo di memoria del carattere a in posizione 51: 0x7fff55c143e3
indirizzo di memoria del carattere a in posizione 52: 0x7fff55c143e4
indirizzo di memoria del carattere a in posizione 53: 0x7fff55c143e5
indirizzo di memoria del carattere a in posizione 54: 0x7fff55c143e6
indirizzo di memoria del carattere a in posizione 55: 0x7fff55c143e7
indirizzo di memoria del carattere a in posizione 56: 0x7fff55c143e8
indirizzo di memoria del carattere a in posizione 57: 0x7fff55c143e9
indirizzo di memoria del carattere a in posizione 58: 0x7fff55c143ea
indirizzo di memoria del carattere a in posizione 59: 0x7fff55c143eb
indirizzo di memoria del carattere a in posizione 60: 0x7fff55c143ec
indirizzo di memoria del carattere a in posizione 61: 0x7fff55c143ed

nome utente inserito (overflow): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>
zsh: segmentation fault ./BOF_indirizzi

```

In questo caso possiamo notare due cose. Il programma smette di leggere correttamente la stringa dalla posizione 40 in poi, dove legge un carattere “>” nonostante il nostro input da tastiera fosse “a”. Inoltre termina restituendo l'errore **segmentation fault**. Questa situazione prende il nome di **buffer overflow**. In caso il codice sia vulnerabile come in questo caso, un attaccante potrebbe sfruttare questa situazione per mandare in crash il programma o potenzialmente anche accedere ad aree di memoria attigue a quelle previste per gli input, dato che come si vede dall'output sopra al momento della lettura della stringa la memoria è stata richiamata fino alla cella con indirizzo **0x7fff55c143ed** nonostante il buffer compreso di tolleranza arrivasse sino all'indirizzo **0x7fff55c143d7**