

```
File Actions Edit View Help
File Edit Options Buffers Tools C Help
#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;
}

kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)-[~/Desktop]
$ sudo nano bof.c
[sudo] password for kali:

(kali@kali)-[~/Desktop]
$ gcc -g bof.c -o BOF

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

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

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

(kali@kali)-[~/Desktop]
$
```

```
kali@kali: ~/Desktop
File Actions Edit View Help
GNU nano 8.1 bof.c *
#include <stdio.h>

int main() {
char buffer[30];

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

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

return 0;
}
```

```
(kali@kali)~[~/Desktop]
$ sudo nano bof.c

(kali@kali)~[~/Desktop]
$ gcc -g bof.c -o BOF

(kali@kali)~[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:DILETTA
Nome utente inserito: DILETTA

(kali@kali)~[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:GHSDHFHIFHWEHUEJIEIUEHJFHJKHFJKDSJKCBDNBBCSBDShFAKLDSDIWIFHBFBDBHD
BFKEUIEIF
Nome utente inserito: GHSDHFHIFHWEHUEJIEIUEHJFHJKHFJKDSJKCBDNBBCSBDShFAKLDSDIWIFHBFBDBHDBFKEUIEIF
zsh: segmentation fault ./BOF

(kali@kali)~[~/Desktop]
$
```

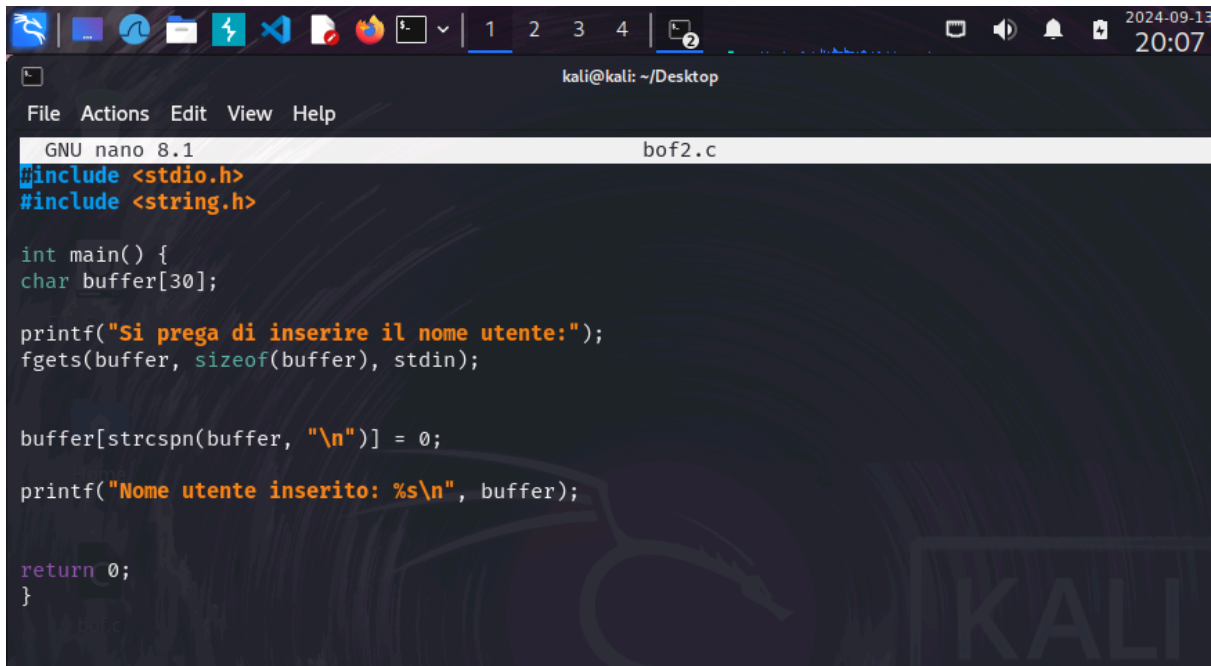
Nel tentativo di riprodurre l'errore di buffer overflow (BOF) ho aumentato la dimensione del buffer da 10 a 30 caratteri, seguendo le istruzioni dell'esercizio. Dopo questa modifica, ho provato a inserire una stringa di lunghezza superiore ai 30 caratteri per testare la robustezza del nuovo buffer. Nonostante l'aumento della dimensione, il programma ha comunque generato un errore di BOF quando la lunghezza della stringa inserita ha superato i 30 caratteri.

Questa osservazione conferma che semplicemente aumentare la dimensione del buffer non elimina completamente la possibilità di un overflow, se non si implementano ulteriori controlli sull'input. Infatti, senza una verifica rigorosa della lunghezza dell'input rispetto alla capacità

del buffer, ogni buffer è potenzialmente vulnerabile a overflow se gli input superano la sua capacità.

Per mitigare questo rischio, è essenziale implementare controlli come l'utilizzo di funzioni più sicure che limitano il numero di caratteri letti in base alla dimensione del buffer, ad esempio `fgets` al posto di `scanf`, oppure verificare la lunghezza dell'input prima di procedere con la lettura o la copia dei dati nel buffer.

Ho provato a modificare il codice e poi ad eseguirlo.



```
GNU nano 8.1 bof2.c
#include <stdio.h>
#include <string.h>

int main() {
char buffer[30];

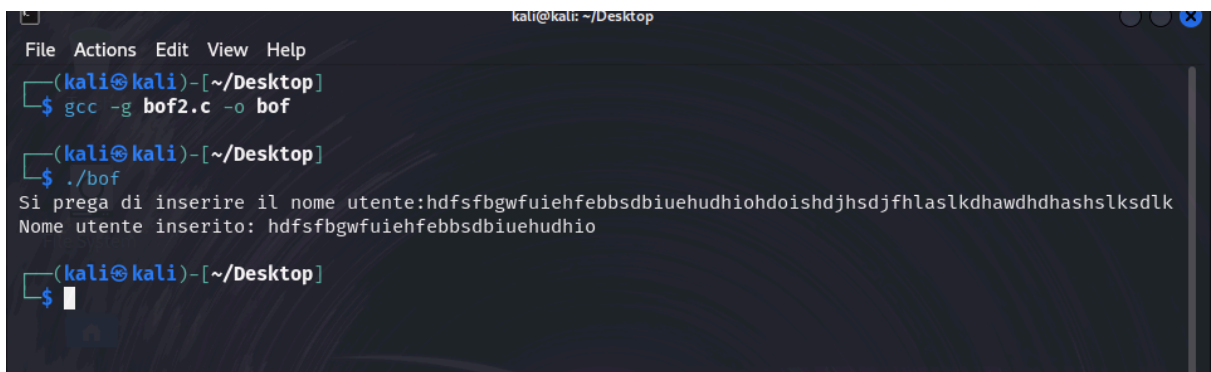
printf("Si prega di inserire il nome utente:");
fgets(buffer, sizeof(buffer), stdin);

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

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

return 0;
}
```

In questa versione, ho mantenuto l'uso di `fgets` che legge al massimo 29 caratteri più il terminatore di stringa `\0`. La funzione `fgets` inserirà automaticamente il terminatore di stringa alla fine dell'input, garantendo che non ci sia un overflow del buffer. Il programma stampa direttamente il contenuto del buffer, inclusi eventuali caratteri di "a capo", il che può essere accettabile a seconda del contesto di utilizzo del programma.



```
kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)~[~/Desktop]
$ gcc -g bof2.c -o bof
(kali@kali)~[~/Desktop]
$ ./bof
Si prega di inserire il nome utente:hdfsfbgwfuiehfdbbsdbiuehudhiohdoishdjhdsdjfhlaslkdhawdhhashslksdlk
Nome utente inserito: hdfsfbgwfuiehfdbbsdbiuehudhio
(kali@kali)~[~/Desktop]
$
```

Contrariamente a quanto accadeva con `scanf`, il programma non ha più generato un errore di segmentation fault. Questo conferma che l'uso di `fgets` è efficace nel prevenire il buffer

overflow, poiché questa funzione limita il numero di caratteri letti alla dimensione del buffer, includendo il terminatore di stringa. L'output del programma mostra che il buffer viene correttamente riempito fino al suo limite senza oltrepassarlo, il che evita l'errore di memoria che prima causava il crash del programma.