

# S7 L4

## Svolgimento Esercizio

Giulia Salani

# Buffer Overflow: definizione

Il buffer overflow avviene quando la quantità di dati scritta in un buffer supera la sua dimensione allocata, sconfinando in aree di memoria adiacenti. Un buffer è un'area di memoria designata per immagazzinare dati temporanei come array di caratteri.

## Cause:

Spesso causato da input non validato o malintenzionato, può essere sfruttato per sovrascrivere l'esecuzione del programma, compromettendo la sicurezza.

## Rischi associati:

L'overflow può portare a corruzione di dati, crash del programma e, in particolare, a esecuzione di codice dannoso (exploit), mettendo a rischio la sicurezza del sistema.

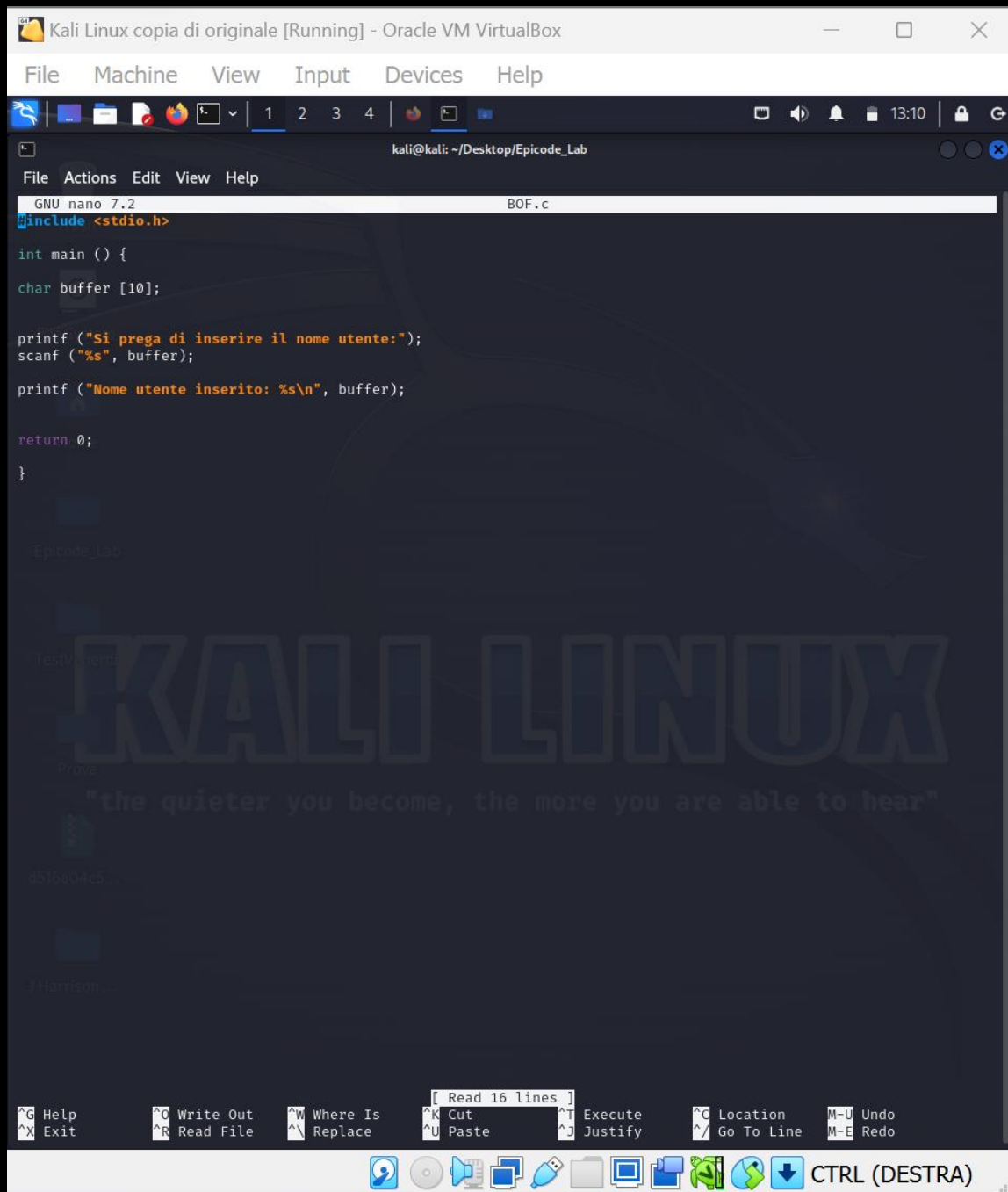
## Modalità di sfruttamento:

Un attaccante può inserire dati progettati per sfruttare la vulnerabilità, sovrascrivendo indirizzi di memoria critici o eseguendo codice dannoso.

## Conseguenze:

Il buffer overflow può compromettere la riservatezza, l'integrità e la disponibilità dei dati, fornendo agli attaccanti il controllo del sistema.

Codice originale



Kali Linux copia di originale [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

kali@kali: ~/Desktop/Epicode\_Lab

```
GNU nano 7.2 BOF.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;
}
```

Epicode\_Lab

Text Editor

Prova

"the quieter you become, the more you are able to hear"

ASTROBAC

J Harrison

Read 16 lines

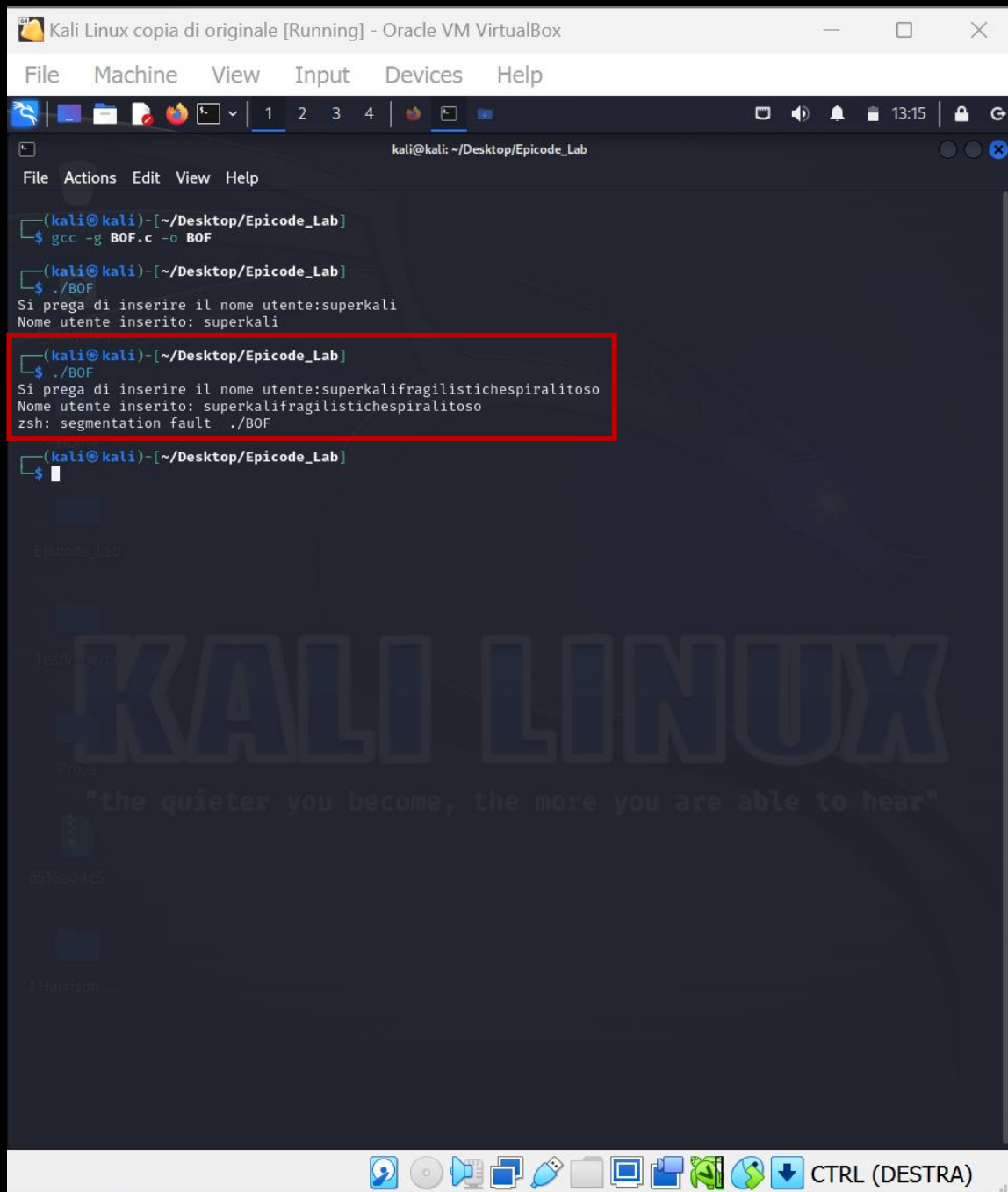
Help Exit Write Out Read File Where Is Replace Cut Paste Execute Justify Location Go To Line Undo Redo

CTRL (DESTRA)

Copiamo il codice di partenza su Kali e lo analizziamo.

Il codice utilizza la funzione **scanf** per leggere una stringa di caratteri dall'input utente e memorizzarla nel buffer dichiarato come `char buffer[10]`.

La criticità principale consiste nell'uso di `%s` come formato di scansione in `scanf`. Questo formato non specifica la dimensione massima della stringa da leggere, il che può portare a un overflow del buffer se l'utente inserisce una stringa più lunga di quanto il buffer può contenere.



```
kali@kali: ~/Desktop/Epicode_Lab
File Actions Edit View Help

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

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

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

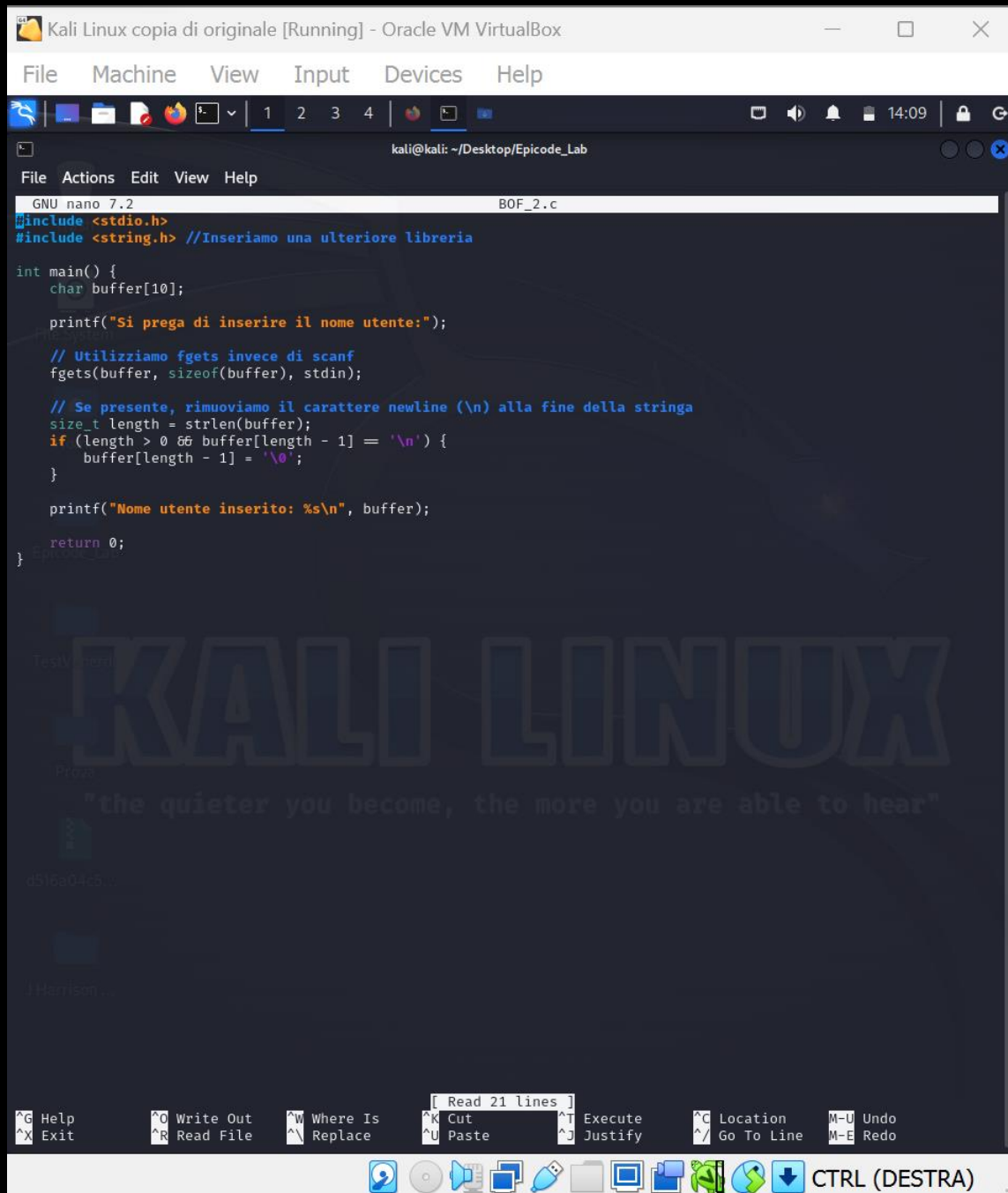
(kali@kali)-[~/Desktop/Epicode_Lab]
$
```

Vediamo il programma all'opera.

Se inseriamo in input un nome utente di lunghezza inferiore a 10 caratteri, il programma restituisce correttamente la nostra stringa.

Se invece inseriamo in input un nome utente di lunghezza maggiore di 10 caratteri, il programma restituisce il nome utente nella sua interezza e un messaggio di errore.

Codice modificato



```
GNU nano 7.2 BOF_2.c
#include <stdio.h>
#include <string.h> //Inseriamo una ulteriore libreria

int main() {
    char buffer[10];

    printf("Si prega di inserire il nome utente:");

    // Utilizziamo fgets invece di scanf
    fgets(buffer, sizeof(buffer), stdin);

    // Se presente, rimuoviamo il carattere newline (\n) alla fine della stringa
    size_t length = strlen(buffer);
    if (length > 0 && buffer[length - 1] == '\n') {
        buffer[length - 1] = '\0';
    }

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

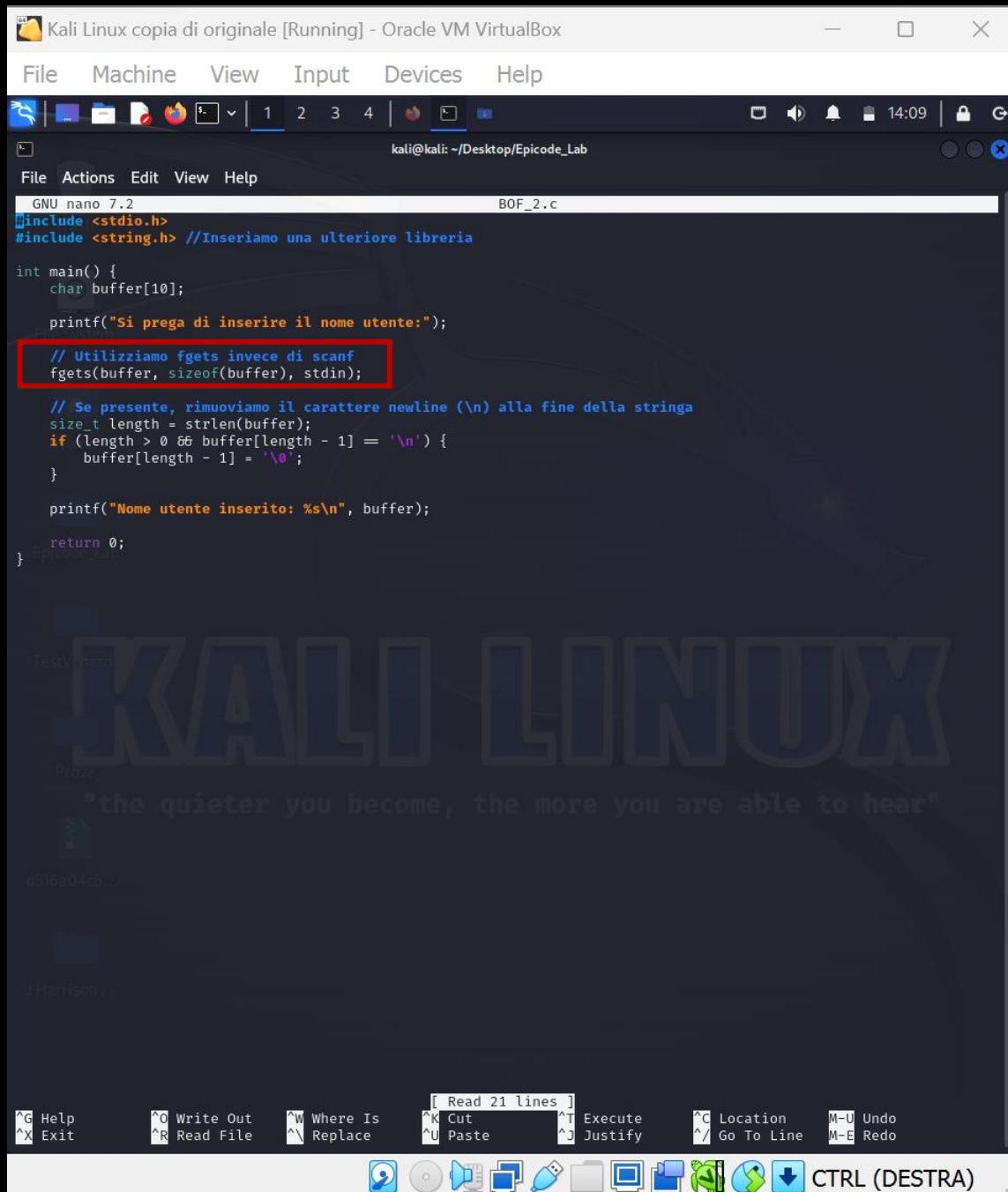
    return 0;
}
```

Modifichiamo il programma in un nuovo file (BOF\_2.c).

Le modifiche sono essenzialmente 3:

- 1) L'inserimento di una ulteriore libreria;
- 2) La sostituzione di scanf con fgets;
- 3) La rimozione del carattere newline a fine stringa;

Vediamo nel dettaglio il punto 2 e 3 rispettivamente nelle due slide a seguire.



```
GNU nano 7.2 BOF_2.c
#include <stdio.h>
#include <string.h> //Inseriamo una ulteriore libreria

int main() {
    char buffer[10];

    printf("Si prega di inserire il nome utente:");

    // Utilizziamo fgets invece di scanf
    fgets(buffer, sizeof(buffer), stdin);

    // Se presente, rimuoviamo il carattere newline (\n) alla fine della stringa
    size_t length = strlen(buffer);
    if (length > 0 && buffer[length - 1] == '\n') {
        buffer[length - 1] = '\0';
    }

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

    return 0;
}
```

## Punto 2:

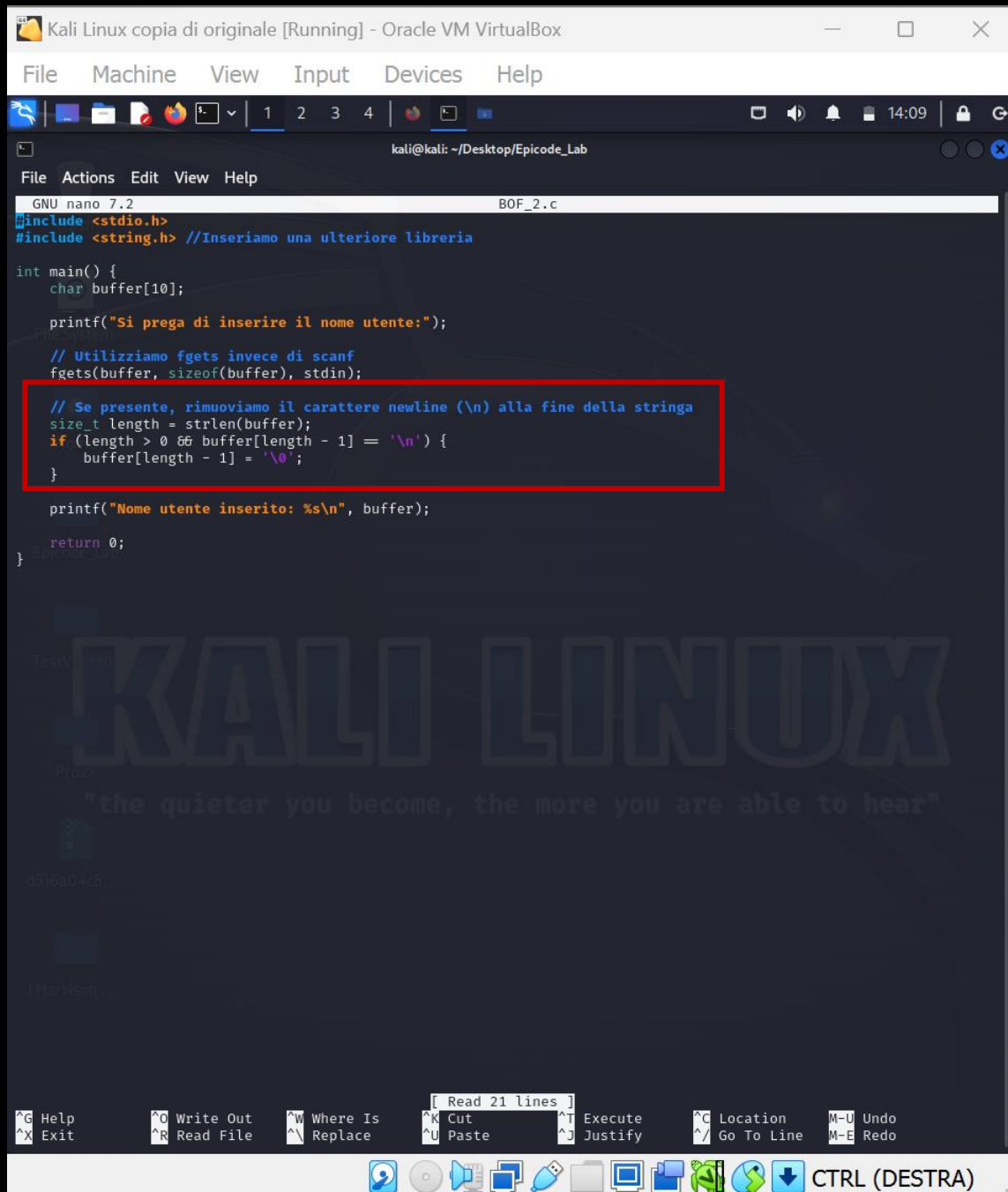
**fgets:** Chiamiamo la funzione fgets, che viene utilizzata per leggere una stringa dall'input di tastiera.

**buffer:** Il primo argomento di fgets è il puntatore al buffer in cui memorizzare la stringa. In questo caso, buffer è un array di caratteri dichiarato precedentemente: `char buffer[10];`. Questo array fungerà da contenitore per la stringa inserita dall'utente.

**sizeof(buffer):** Il secondo argomento specifica la dimensione massima della stringa che fgets può leggere. In questo caso, viene utilizzato `sizeof(buffer)`, che restituisce la dimensione totale in byte dell'array buffer. Questo è cruciale per prevenire buffer overflow: fgets leggerà al massimo `sizeof(buffer) - 1` caratteri dalla tastiera, riservando uno spazio per il terminatore nullo `\0`.

**stdin:** Il terzo argomento è il puntatore al file da cui leggere la stringa. In questo caso, `stdin` rappresenta l'input standard, ovvero la tastiera.





```
GNU nano 7.2 BOF_2.c
#include <stdio.h>
#include <string.h> //Inseriamo una ulteriore libreria

int main() {
    char buffer[10];

    printf("Si prega di inserire il nome utente:");

    // Utilizziamo fgets invece di scanf
    fgets(buffer, sizeof(buffer), stdin);

    // Se presente, rimuoviamo il carattere newline (\n) alla fine della stringa
    size_t length = strlen(buffer);
    if (length > 0 && buffer[length - 1] == '\n') {
        buffer[length - 1] = '\0';
    }

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

    return 0;
}
```

### Punto 3:

Questa parte di codice gestisce la rimozione del carattere newline (\n) dalla stringa letta con fgets e garantisce che la stringa sia correttamente terminata con un terminatore nullo (\0). Vediamo in dettaglio cosa fa ciascuna riga:

**size\_t length = strlen(buffer);**: Calcola la lunghezza effettiva della stringa letta da fgets utilizzando la funzione strlen. Il risultato, memorizzato nella variabile length, rappresenta il numero di caratteri nella stringa, escluso il terminatore nullo \0.

**if (length > 0 && buffer[length - 1] == '\n') {**: Verifica se la lunghezza della stringa è maggiore di zero e se l'ultimo carattere della stringa è un newline (\n). Questa condizione è controllata per assicurarsi che ci sia almeno un carattere nella stringa e che il newline sia presente.

**buffer[length - 1] = '\0';**: Se la condizione dell'if è soddisfatta, questo significa che il newline è presente alla fine della stringa. Quindi, il codice sostituisce il carattere newline con il terminatore nullo (\0). In questo modo, la stringa viene correttamente terminata, e il newline viene eliminato. Questo passaggio è importante per evitare eventuali problemi derivanti dalla presenza del newline nella stringa.