

Cyber Security Report



W17D4

31/10/2025

Autore :

Pace Massimiliano

email : *efmpas@gmail.com*

Indice :

° Introduzione pag. 2

° Spiegazione esercizio e svolgimento pag.2 - 4

° Conclusioni pag. 5

INTRODUZIONE BUFFER OVERFLOW

° Il buffer overflow è una vulnerabilità conseguenza di una mancanza di controllo dei limiti dei buffer che accettano input utente. Creeremo un esempio di codice in C volutamente vulnerabile ai BOF, e scateneremo una situazione di errore particolare chiamata «segmentation fault», ovvero un errore di memoria che si presenta quando un programma cerca inavvertitamente di scrivere su una posizione di memoria dove non gli è permesso scrivere (come può essere ad esempio una posizione di memoria dedicata a funzioni del sistema operativo).

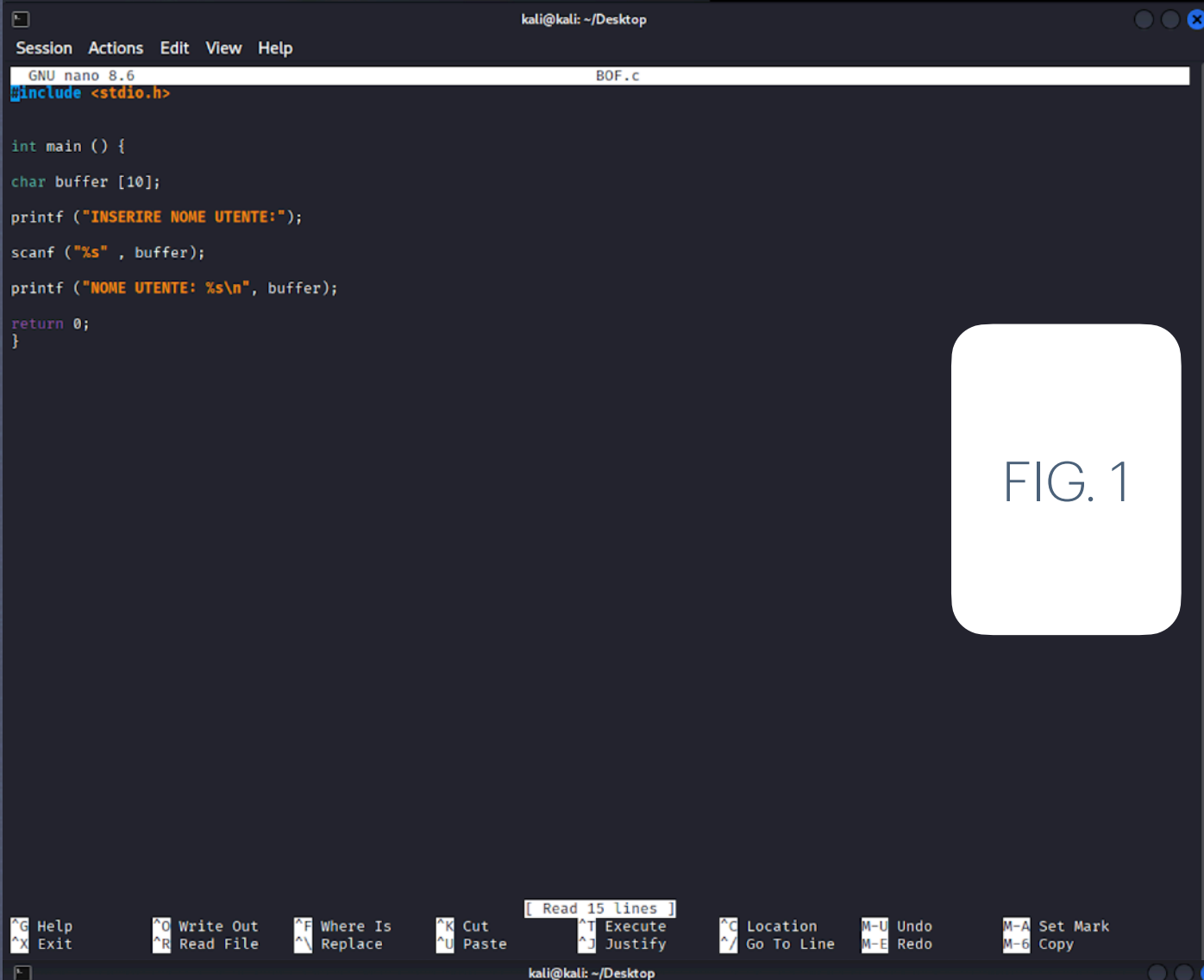
SPIEGAZIONE ESERCIZIO

° Step 1: Avvio delle VM Kali

° Step 2 : creiamo il codice in c con nano e diamogli il nome di BOF.c fig. 1

° Step 3 :compiliamo il nostro codice con il comando `gcc -g BOF.c -o BOF`

° Step 4 :avviamo BOF appena creato con `./BOF`
Ci aspettiamo un errore appena andremo ad inserire un utente superiore a 10 caratteri fig. 2



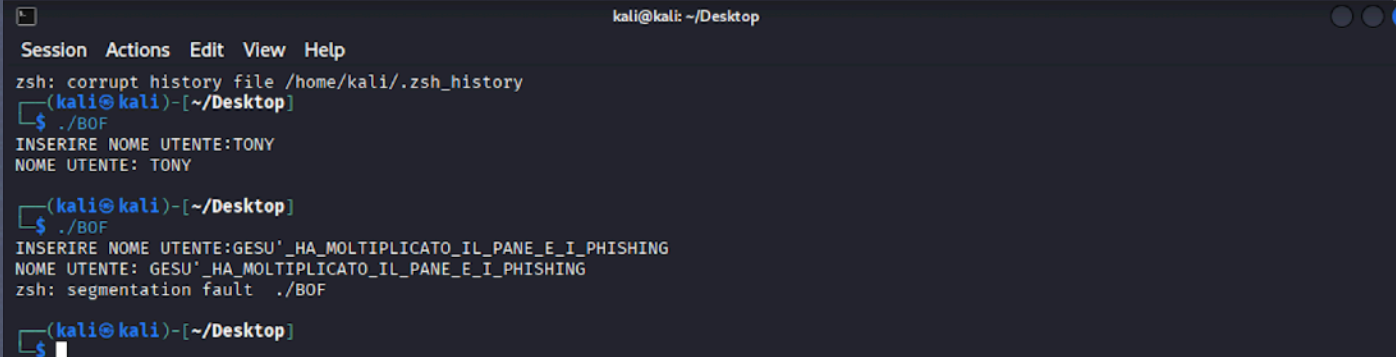
The screenshot shows a terminal window with the nano text editor open. The file being edited is named `BOF.c`. The code is a simple C program designed to demonstrate a buffer overflow. It includes `<stdio.h>`, defines a `main` function, and declares a `char buffer[10]`. The program prompts the user to "INSERIRE NOME UTENTE:" and reads input into the `buffer` using `scanf`. It then prints the contents of the buffer with `printf` and returns 0. The terminal window has a menu bar with "Session", "Actions", "Edit", "View", and "Help". A status bar at the bottom shows various keyboard shortcuts for navigation and editing.

```
GNU nano 8.6 BOF.c
#include <stdio.h>

int main () {
char buffer [10];
printf ("INSERIRE NOME UTENTE:");
scanf ("%s" , buffer);
printf ("NOME UTENTE: %s\n", buffer);
return 0;
}
```

^G Help ^O Write Out ^F Where Is ^K Cut [Read 15 lines ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^_ Justify ^_ Go To Line M-E Redo M-6 Copy

FIG. 1



The screenshot shows a terminal window with the command prompt `(kali@kali)-[~/Desktop]`. The user has run `./BOF`, which has prompted them to enter a name. They have entered `TONY`, and the program has printed `NOME UTENTE: TONY`. The user has run `./BOF` again, and the program has prompted them to enter a name. They have entered `GESU' HA MOLTIPLICATO IL PANE E I PHISHING`, and the program has printed `NOME UTENTE: GESU' HA MOLTIPLICATO IL PANE E I PHISHING`. The terminal window has a menu bar with "Session", "Actions", "Edit", "View", and "Help". A status bar at the bottom shows various keyboard shortcuts for navigation and editing.

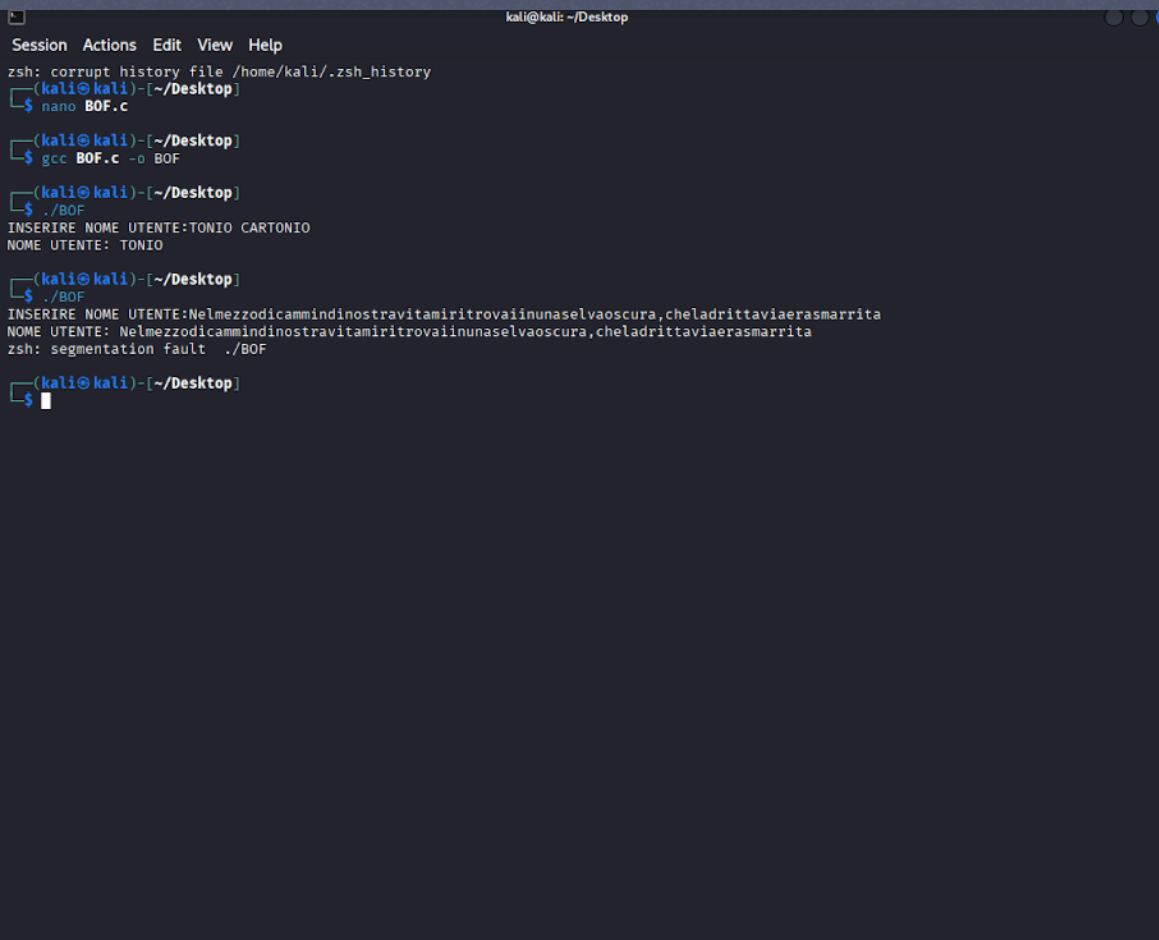
```
zsh: corrupt history file /home/kali/.zsh_history
(kali@kali)-[~/Desktop]
$ ./BOF
INSERIRE NOME UTENTE:TONY
NOME UTENTE: TONY

(kali@kali)-[~/Desktop]
$ ./BOF
INSERIRE NOME UTENTE:GESU' HA MOLTIPLICATO IL PANE E I PHISHING
NOME UTENTE: GESU' HA MOLTIPLICATO IL PANE E I PHISHING
zsh: segmentation fault ./BOF

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

FIG. 2

° Step 5 : Aumentiamo la dimensione del vettore da 10 a 30 così da aumentare il limite dei caratteri fig. 3



```
kali@kali: ~/Desktop
Session Actions Edit View Help
zsh: corrupt history file /home/kali/.zsh_history
(kali@kali)~-[~/Desktop]
$ nano BOF.c
(kali@kali)~-[~/Desktop]
$ gcc BOF.c -o BOF
(kali@kali)~-[~/Desktop]
$ ./BOF
INSERIRE NOME UTENTE:TONIO CARTONIO
NOME UTENTE: TONIO
(kali@kali)~-[~/Desktop]
$ ./BOF
INSERIRE NOME UTENTE:Nelmezzodicammindinostravitamiritrovaiinunaselvaoscura,cheladrittaviaerasmarrita
NOME UTENTE: Nelmezzodicammindinostravitamiritrovaiinunaselvaoscura,cheladrittaviaerasmarrita
zsh: segmentation fault ./BOF
(kali@kali)~-[~/Desktop]
$
```

FIG. 3

° Step 6 : La soluzione per limitare il BOF :

1) SOLUZIONE CON SCANF:

`scanf("%9s", buffer);` questo imposta un limite di lunghezza così da fargli leggere un massimo di 9 caratteri evitando la sovrascrittura del buffer.

2) SOLUZIONE CON FGETS :

`fgets(buffer, sizeof(buffer), stdin);` Il metodo più sicuro usa `fgets`, che permette di leggere una riga dal terminale specificando la dimensione del buffer così non si rischia mai overflow, perché `fgets` rispetta sempre la dimensione indicata.

CONCLUSIONI :

Un buffer è una zona di memoria dove un programma mette temporaneamente i dati. Se arrivano dati più grandi di quello che il buffer può contenere, questi “traboccano” in altre aree di memoria contigue, sovrascrivendo dati importanti o il codice del programma.

Questo può causare diversi problemi:

- Il programma può bloccarsi o funzionare in modo errato.
- Un attaccante può sfruttare questa vulnerabilità per inserire e far eseguire codice malevolo, prendendo il controllo del sistema.
- Può portare a perdita o corruzione di dati.
- Può creare una porta d'accesso non autorizzata o causare interruzione del servizio (DoS).

In parole semplici, il buffer overflow è una falla che può permettere a chi vuole fare danni di entrare nel sistema, causare guasti o rubare dati. Per questo è fondamentale scrivere codice che limiti l'input per evitare che il buffer si riempia troppo e trabocchi, proteggendo così la sicurezza del software e del sistema