

Traccia:

Provate a riprodurre l'errore di segmentazione modificando il programma come di seguito:

Aumentando la dimensione del vettore a 30;

Fare la prova dell'errore modificare il codice in modo che l'errore non si verifichi (es aumentare il vettore a 30 o fare dei controlli)

Verificare, modificando il codice, dove va a scrivere i caratteri in overflow

Il programma fornito inizialmente è un esempio di codice in C volutamente vulnerabile ai buffer overflow (BOF).

L'obiettivo è dimostrare come una mancanza di controllo dei limiti dei buffer possa portare a vulnerabilità di sicurezza, in questo caso, causando un errore di segmentazione durante l'esecuzione del programma.

```
#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;

}
```

Come si evince dalle slide dell'esercizio, questo comando riproduce volutamente un errore se si digitano più di 10 caratteri.

Il programma si avvia chiedendoci di inserire un nome utente:

- Inserendo un nome utente di 5 caratteri, il programma non ci riporta nessun problema, infatti come sappiamo il buffer accetta fino a 10 caratteri. Cosa succede se inseriamo 30 caratteri? **Proviamo**

```
(kali@kali)~/Desktop
$ ./BOF
Si prega di inserire il nome utente:test1
Nome utente inserito: test1
(kali@kali)~/Desktop
$
```

```
Si prega di inserire il nome utente:qwertyuiopasdfghjklzxcvbnmqwer
Nome utente inserito: qwertyuiopasdfghjklzxcvbnmqwer
zsh: segmentation fault ./BOF
(kali@kali)~/Desktop
$
```

Se inseriamo 30 caratteri il programma ci ritorna un errore, «segmentation fault», ovvero errore di segmentazione. L'errore di segmentazione avviene quando un programma, come abbiamo detto in precedenza, tenta di scrivere contenuti su una porzione di memoria alla quale non ha accesso. Questo è un chiaro esempio di BOF, abbiamo inserito 30 caratteri in un buffer che ne può contenere solamente 10 e di conseguenza alcuni caratteri stanno sovrascrivendo aree di memorie inaccessibili.

10

Quindi ci viene richiesto di correggere questo codice in C aumentando i caratteri fino a 30.

Obiettivi delle Modifiche:

1. Aumentare la dimensione del buffer a 30 per simulare un buffer overflow.
2. Provocare e rilevare un errore di segmentazione durante l'esecuzione del programma.
3. Correggere il programma utilizzando una pratica sicura, come l'uso di **fgets**, per evitare buffer overflow e errori di segmentazione.

1. AUMENTO DEI CARATTERI FINO A 30 UTILIZZANDO PROGRAMMA IN LINGUAGIO C:

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

int main() {
    // Aumento la dimensione del vettore a 30
    char buffer[30];

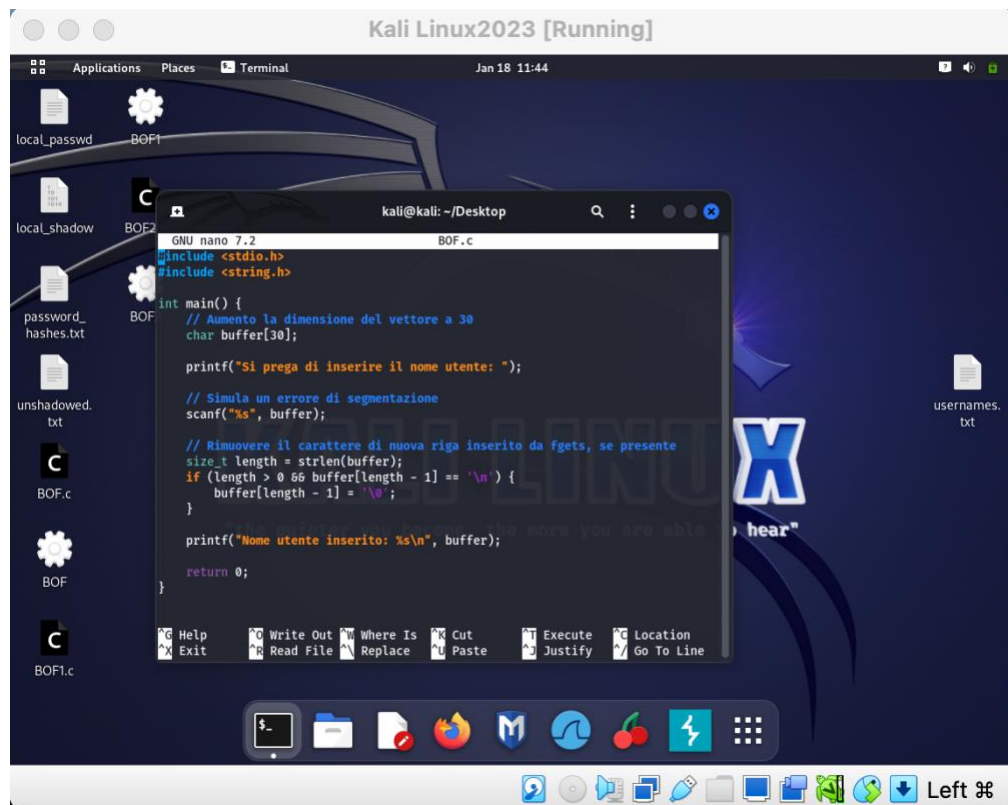
    printf("Si prega di inserire il nome utente: ");

    // Simula un errore di segmentazione
    scanf("%s", buffer);

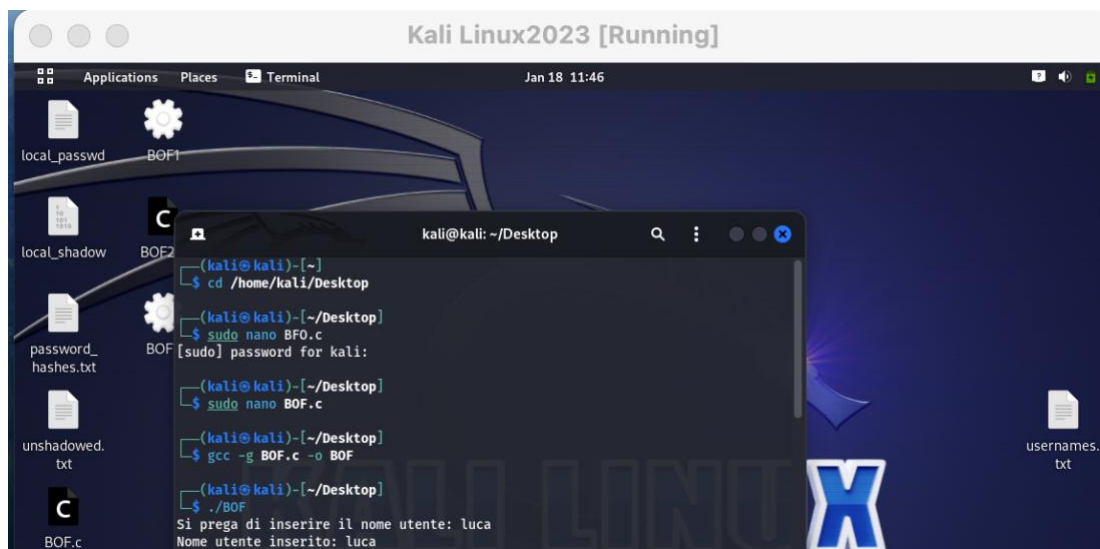
    // Rimuovere il carattere di nuova riga inserito da fgets, se presente
    size_t length = strlen(buffer);
    if (length > 0 && buffer[length - 1] == '\n') {
        buffer[length - 1] = '\0';
    }

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

    return 0;
}
```



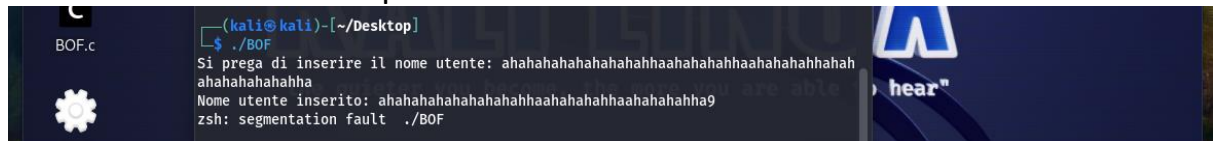
ESECUZIONE:
gcc -g BOF.c -o BOF
./BOF



Il programma funziona e non riporta nessun problema durante l'esecuzione del codice.

2. ESECUZIONE DELL'ERRORE PER VERIFICARE IL BUFFEROVERFLOW

Ho inserito solo 4 caratteri con il nome luca e funziona correttamente ma cosa accade se inserisco più di 30 caratteri?



```
(kali@kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente: ahahahahahahahahhaahahahhaahahahahah
ahahahahahha
Nome utente inserito: ahahahahahahahahhaahahahhaahahahha9
zsh: segmentation fault ./BOF
```

Inserendo più di 30 caratteri il terminale riporta un messaggio di errore **zsh: segmentation fault ./BOF.**

Dunque ci ritroviamo in un bufferoverflow e ricordiamo cos'è un bufferoverflow:

Un buffer overflow è una vulnerabilità di sicurezza che si verifica quando un programma, durante l'elaborazione di dati in input, scrive oltre i limiti di un buffer di memoria. I buffer sono aree di memoria allocate per immagazzinare dati temporanei, come stringhe di testo o array di byte. Tuttavia, se un programma non effettua un controllo adeguato sulla dimensione dei dati in input, può verificarsi un overflow di buffer.

Quando un buffer overflow si verifica, i dati in input in eccesso possono sovrascrivere la memoria adiacente al buffer, compromettendo l'integrità del programma e del sistema operativo. Questo può portare a comportamenti imprevisti, crash del programma o, in situazioni più gravi, a esecuzioni di codice dannoso o attacchi hacker.

3. CORREZIONE DEL BUFFEROVERFLOW:

Per la correzione ho usato una funzione più sicura **fgets** con il seguente codice in linguaggio C

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

int main() {
    // Aumento la dimensione del vettore a 30
    char buffer[30];

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

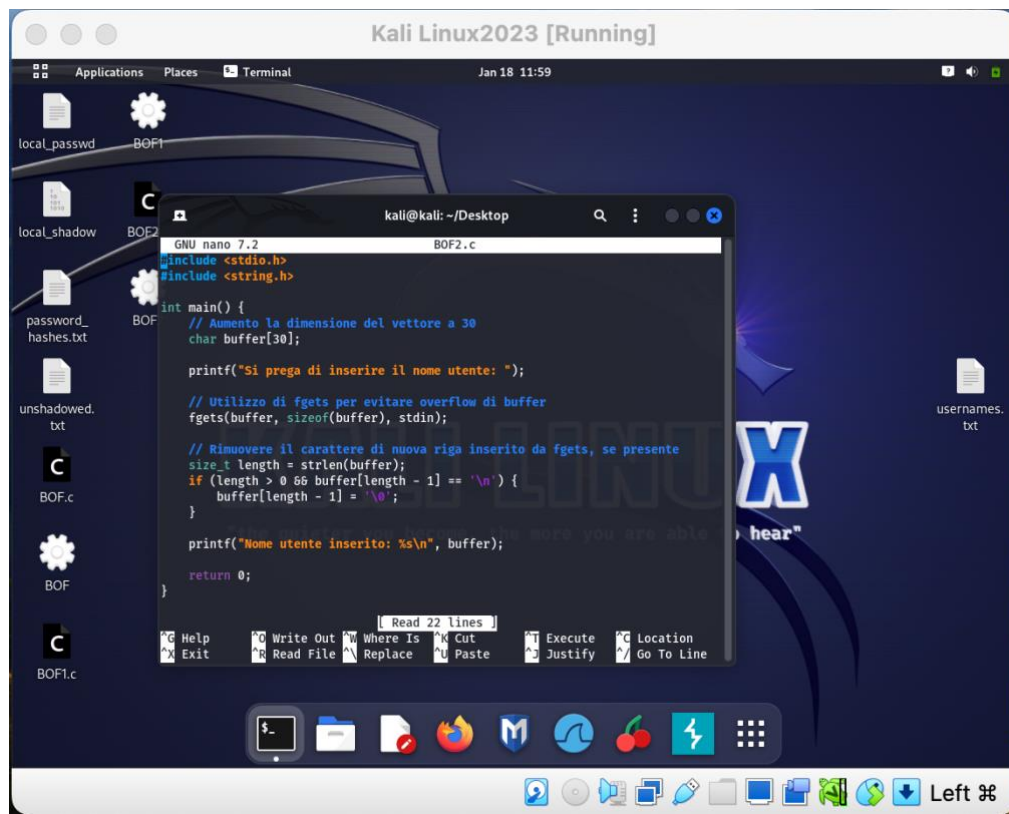
    // Utilizzo di fgets per evitare overflow di buffer
    fgets(buffer, sizeof(buffer), stdin);

    // Rimuovere il carattere di nuova riga inserito da fgets, se presente

    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;
}
```



Procedendo con l'esecuzione di `./BOF2` noterò che inserendo anche più di 30 caratteri il programma mi leggerà solo i 30 caratteri e che non mi darà nessun errore grazie alla funzione **fgets**.

```
(kali@kali)-[~/Desktop]
$ sudo nano BOF2.c

(kali@kali)-[~/Desktop]
$ gcc -g BOF2.c -o BOF2

(kali@kali)-[~/Desktop]
$ ./BOF2
Si prega di inserire il nome utente: lualclulclualahshshdhgdgdglucahahlcucgagah
uclcuuagsgdjchc7hags77
Nome utente inserito: lualclulclualahshshdhgdgdglu
```

CODICE 1 VS CODICE 2:

Vediamo le differenze principali e perché la seconda versione è preferibile:

Dimensione del Buffer:

Primo Codice: **char buffer[10];**

Secondo Codice: **char buffer[30];**

Nel primo codice, il buffer è limitato a 10 caratteri.

Se l'utente inserisce più di 10 caratteri, si verificherà un overflow di buffer.

Nel secondo codice, la dimensione del buffer è stata aumentata a 30, il che offre più spazio per l'input dell'utente e riduce il rischio di overflow.

Funzione di Input:

Primo Codice: **scanf("%s", buffer);**

Secondo Codice: **fgets(buffer, sizeof(buffer), stdin);**

Nel primo codice, **scanf** viene utilizzato senza specificare la dimensione massima del buffer, il che può causare overflow di buffer. Nel secondo codice, **fgets** è utilizzato con la dimensione massima del buffer specificata, rendendo più sicura la lettura dell'input.

Controllo della Lunghezza:

Secondo Codice:

```
size_t length = strlen(buffer);  
if (length > 0 && buffer[length - 1] == '\n') {  
    buffer[length - 1] = '\0';  
}
```

Dopo l'utilizzo di **fgets**, il secondo codice controlla la lunghezza della stringa letta e rimuove il carattere di nuova riga (**\n**) se presente. Questo è un passo importante per gestire correttamente l'input dell'utente e evitare comportamenti imprevisti.

In generale, la seconda versione del codice è più robusta e sicura poiché tiene conto della dimensione del buffer e utilizza **fgets** per leggere l'input in modo sicuro, evitando così i buffer overflow.

È sempre una pratica consigliata specificare la dimensione massima del buffer e utilizzare funzioni di input sicure per prevenire vulnerabilità come i buffer overflow.