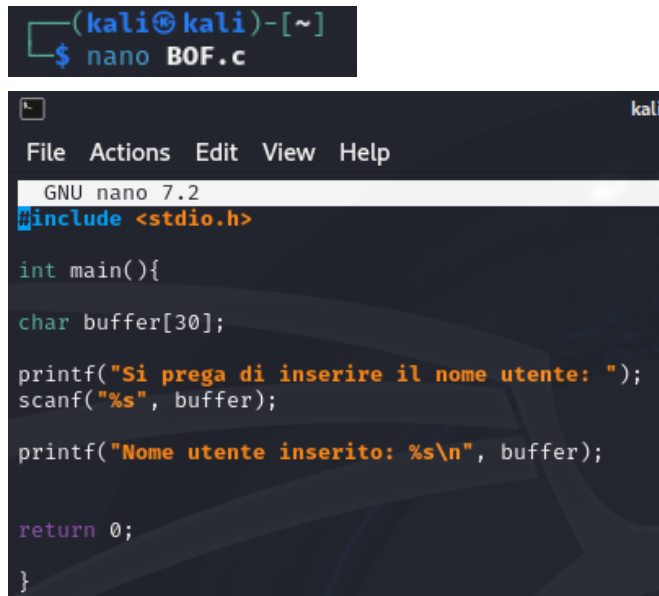


## S7.L4 Buffer overflow

### Fase 1: Modifica e Preparazione del Codice



```
(kali㉿kali)-[~]  
$ nano BOF.c  
  
File Actions Edit View Help  
GNU nano 7.2  
#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;  
}
```

#### Descrizione della Fase 1

consiste nella creazione e modifica del file sorgente C che contiene il programma vulnerabile al buffer overflow.

#### Passaggi Dettagliati

##### Apertura dell'Editor di Testo

- Utilizzando il comando nano, viene aperto un editor di testo per creare e modificare il file sorgente C denominato BOF.c.

##### nano BOF.c

- Questo comando apre l'editor nano e crea il file BOF.c se non esiste già, oppure apre il file per modifiche se esiste.

##### Scrittura del Codice

- All'interno dell'editor nano, viene scritto il codice seguente:

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

```
int main() {
```

```
    char buffer[30]; // Dichiarazione di un buffer di dimensione 30
```

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

```
    scanf("%s", buffer); // Lettura dell'input utente e memorizzazione nel buffer
```

```
    printf("Nome utente inserito: %s\n", buffer); // Visualizzazione del nome utente
```

```
    inserito
```

```
    return 0; // Terminazione del programma
```

```
}
```

##### Salvataggio e Uscita

- Dopo aver scritto il codice, si salva il file e si esce dall'editor nano. In nano, questo si fa tipicamente premendo CTRL+O per scrivere le modifiche e CTRL+X per uscire dall'editor.

Questa fase è fondamentale perché prepara il programma che verrà compilato ed eseguito nelle fasi successive. È qui che viene stabilita la struttura del buffer e viene introdotta la potenziale vulnerabilità del buffer overflow.

## Fase 2: Compilazione ed Esecuzione del Programma

```
(kali㉿kali)-[~]
$ gcc BOF.c -o BOF
```

```
(kali㉿kali)-[~]
$ ./BOF
Si prega di inserire il nome utente: danilo
Nome utente inserito: danilo
```

### Descrizione della Fase 2

La fase 2 consiste nella compilazione del programma scritto nella Fase 1 e nella sua successiva esecuzione per verificare il corretto funzionamento del codice.

#### Passaggi Dettagliati

##### Compilazione del Codice

- Utilizzando il compilatore gcc (GNU Compiler Collection), viene compilato il file sorgente BOF.c per creare un file eseguibile chiamato BOF.

##### gcc BOF.c -o BOF

- **Analisi del Comando:**
  - gcc: Invoca il compilatore GNU per il linguaggio C.
  - BOF.c: Specifica il file sorgente C da compilare.
  - -o BOF: Specifica il nome del file eseguibile da creare. In questo caso, il file eseguibile sarà chiamato BOF.

##### Esecuzione del Programma

- Dopo aver compilato il programma con successo, eseguo l'eseguibile BOF per verificare il comportamento del programma.

##### ./BOF

- Durante l'esecuzione, il programma chiede all'utente di inserire un nome utente. In questo esempio, inserisco "danilo".
  - Input dell'utente: danilo
  - Output del programma: Nome utente inserito: danilo

##### Verifica del Comportamento

- L'output mostra che il programma ha correttamente memorizzato e stampato il nome utente inserito senza causare errori. Questo conferma che, con un

input di lunghezza adeguata (inferiore a 30 caratteri), il programma funziona correttamente.

## Fase Finale: Dimostrazione del Buffer Overflow

```
(kali@kali)-[~]  
$ ./BOF  
Si prega di inserire il nome utente: idfidffmwinrfwrnfwrfcowirfgoiwrngfuiehwuthf3erfyeryweryvfueryf  
Nome utente inserito: idfidffmwinrfwrnfwrfcowirfgoiwrngfuiehwuthf3erfyeryweryvfueryf  
zsh: segmentation fault ./BOF
```

### Descrizione della Fase Finale

Nella fase finale, eseguo nuovamente il programma, questa volta fornendo un input che eccede la dimensione del buffer per dimostrare come un buffer overflow possa causare un errore di segmentazione (segmentation fault).

#### Passaggi Dettagliati

##### Esecuzione del Programma con Input Troppo Lungo

- Eseguo il programma BOF compilato in precedenza, fornendo un input che supera la capacità del buffer di 30 caratteri.

**./BOF**

**Input dell'Utente:** Viene inserito un nome utente molto lungo

**idfidffmwinrfwrnfwrfcowirfgoiwrngfuiehwuthf3erfyeryweryvfueryf**

**Output del Programma:** Il programma tenta di memorizzare e stampare l'input fornito, ma l'eccesso di caratteri rispetto alla capacità del buffer provoca un errore di segmentazione.

**Nome utente inserito:**

**idfidffmwinrfwrnfwrfcowirfgoiwrngfuiehwuthf3erfyeryweryvfueryf**

**zsh: segmentation fault ./BOF**

#### Analisi del Risultato

- **Errore di Segmentazione (Segmentation Fault):** Il programma si interrompe con un errore di segmentazione, segnalato dal messaggio `zsh: segmentation fault ./BOF`.
- **Causa:** Questo errore si verifica perché il buffer definito nel programma può contenere solo 30 caratteri, ma l'input dell'utente supera questa lunghezza. Quando il programma tenta di memorizzare l'input eccedente, sovrascrive aree di memoria adiacenti, causando l'errore di segmentazione.

#### Importanza della Fase Finale

- **Dimostrazione della Vulnerabilità:** Questa fase dimostra chiaramente come un buffer overflow possa verificarsi quando i dati in input eccedono la capacità del buffer. L'errore di segmentazione è una conseguenza diretta del tentativo di scrivere oltre i limiti del buffer.
- **Implicazioni di Sicurezza:** Tale vulnerabilità può essere sfruttata da un attaccante per eseguire codice arbitrario o causare crash del programma, evidenziando

l'importanza di prevenire buffer overflow attraverso tecniche di programmazione sicura.

## Alternativa: Aumento della Dimensione del Buffer

### Descrizione dell'Alternativa

Questa soluzione consiste nell'aumentare significativamente la dimensione del buffer per prevenire l'overflow anche con input molto lunghi.

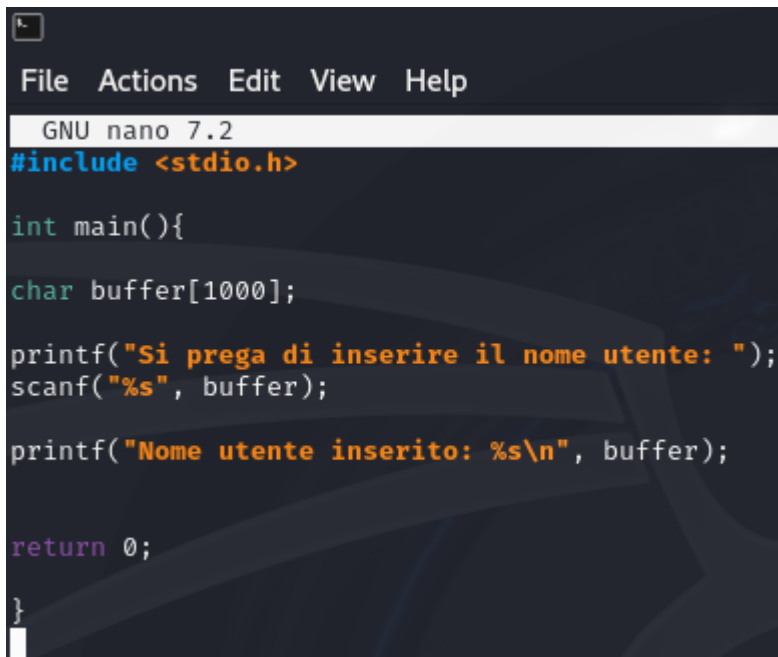
### Passaggi Dettagliati

#### Modifica del Codice

- Apertura dell'Editor di Testo

```
(kali㉿kali)-[~]  
$ nano BOF.c
```

### Scrittura del Codice



```
File Actions Edit View Help  
GNU nano 7.2  
#include <stdio.h>  
  
int main(){  
  
char buffer[1000];  
  
printf("Si prega di inserire il nome utente: ");  
scanf("%s", buffer);  
  
printf("Nome utente inserito: %s\n", buffer);  
  
return 0;  
}
```

- **Analisi del Codice:**
  - `char buffer[1000];`: Invece di un buffer di 30 caratteri, viene dichiarato un buffer di 1000 caratteri. Questo spazio dovrebbe essere sufficiente per la maggior parte degli input utente senza causare un buffer overflow.

### Compilazione del Codice

- Utilizzando il compilatore gcc per creare l'eseguibile

```
(kali㉿kali)-[~]  
$ gcc -g BOF.c -o BOF
```

## Esecuzione del Programma con Input Lungo

```
(kali㉿kali)-[~]  
$ ./BOF
```

**Input dell'Utente:** inserisco un nome utente molto lungo

uishfiwrjnfquiwjchwqrifhywr87yf78whrc87wyr87wyr87wyr87wyr87wyr8weyrc874yrc87  
wyr8w7yf8w7fwc87yf

**Output del Programma**

Nome utente inserito:

uishfiwrjnfquiwjchwqrifhywr87yf78whrc87wyr87wyr87wyr87wyr87wyr8weyrc874yrc87  
wyr8w7yf8w7fwc87yf

```
(kali㉿kali)-[~]  
$ ./BOF  
Si prega di inserire il nome utente: uishfiwrjnfquiwjchwqrifhywr87yf78whrc87wyr8weyrc874yrc87wyr8w7yfwc  
r87yfcwr78yc  
Nome utente inserito: uishfiwrjnfquiwjchwqrifhywr87yf78whrc87wyr8weyrc874yrc87wyr8w7yfwcr87yfcwr78yc
```

## Analisi del Risultato

- **Assenza di Errore di Segmentazione:** Con un buffer di dimensione 1000, il programma può gestire input molto lunghi senza causare un errore di segmentazione. Questo dimostra che aumentando la dimensione del buffer, si può prevenire l'overflow per input più lunghi.
- **Considerazioni sull'efficienza della Memoria:** Sebbene questa soluzione prevenga l'overflow, non è sempre efficiente utilizzare buffer di dimensioni molto grandi, specialmente se la maggior parte degli input sono di lunghezza inferiore.

## Conclusioni e Raccomandazioni

- **Aumento della Dimensione del Buffer:** Aumentare la dimensione del buffer può essere una soluzione semplice per prevenire overflow, ma potrebbe non essere ottimale in termini di utilizzo della memoria.
- **Validazione dell'Input:** È sempre una buona pratica validare l'input per garantire che non superi la dimensione prevista del buffer, anche se il buffer è grande.
- **Utilizzo di Funzioni Sicure:** Funzioni come `strncpy` possono aiutare a limitare il numero di caratteri copiati nel buffer, fornendo una protezione aggiuntiva contro overflow accidentali.

Questa alternativa mostra un approccio diverso per gestire buffer overflow, aumentando significativamente la dimensione del buffer per evitare errori di segmentazione con input molto lunghi.