

# BUFFER OVERFLOW

MATTIA PASTORELLI



# COMANDA:

Abbiamo già parlato del buffer overflow, una vulnerabilità che è conseguenza di una mancanza di controllo dei limiti dei buffer che accettano input utente.

Nelle prossime slide vedremo un esempio di codice in C volutamente vulnerabile ai BOF, e come scatenare 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).

# Cos'è il Buffer Overflow?

Un buffer overflow è una vulnerabilità informatica che si verifica quando un programma, per qualche motivo, scrive più dati in un buffer di quanto esso sia stato progettato per gestire.

I buffer overflow sono il risultato di errori di programmazione o di progettazione del software. Se un attaccante riesce a sfruttare questa vulnerabilità, potrebbe sovrascrivere parti di memoria adiacenti al buffer, portando a comportamenti imprevisti o compromettendo la sicurezza del sistema.



# CODICE IN .C

il primo step è quello di scrivere un programma in C, quindi per prima cosa apriamo un documento vuoto e gli diamo il nome “BOF.c”.

Dopodichè procediamo alla scrittura del codice come in figura.

Possiamo notare che il programma ha la funzione di chiedere all’utente di inserire un nome Utente, ma con un limite di caratteri di 10.

```
File Actions Edit View 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;
}
```

# Avviare il codice

Una volta scritto e salvato il codice, bisogna compilarlo per poterlo startare, essendo che i codici in C bisogna prima effettuare una compilazione su di essi.

Quindi apriamo il terminale sul Desktop, in quanto abbiamo salvato il nostro codice proprio su questa directory e attraverso il comando:

**gcc -g BOF.c -o BOF**

Portiamo a compilazione il codice e dopodichè lo attiviamo con il comando :**./BOF**

Possiamo notare che inserendo un nome utente a meno di 10 caratteri viene letto correttamente, mentre un nome utente a più di 10 caratteri genera un errore: "segmentation fault".

Questo indica che il codice sta cercando di sovrascrivere blocchi di memoria dedicati ad altre attività.

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

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

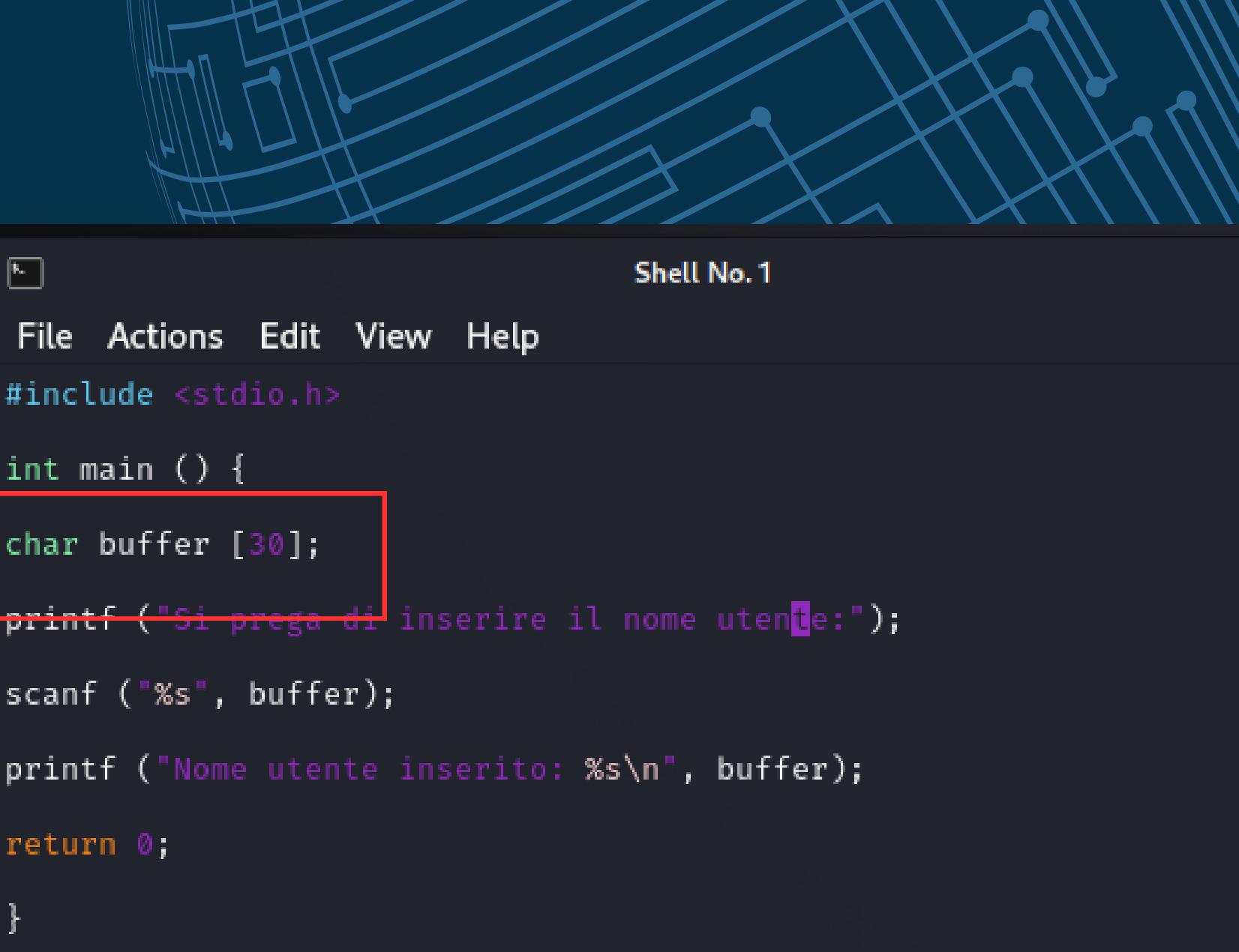
(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente:Aoisfdjioasdhioashdihqwiodhoiashdiaoqwhdia
sdhioqwhdiowhoaisuhdihoiashd
Nome utente inserito: Aoisfdjioasdhioashdihqwiodhoiashdiaoqwhdiasdhioqwhdiowho
aisuhdihoiashd
zsh: segmentation fault ./BOF

(kali㉿kali)-[~/Desktop]
$
```

# COME FIXARE L'ERRORE?

Tornando al codice, andiamo a cambiare il numero massimo di caratteri inseribili da 10 a 30 (come in figura 1). Dopodiché compiliamo e avviamo nuovamente il codice, inseriamo un nome utente più lungo di 10 caratteri e possiamo notare che questa volta prende interamente il nostro nome utente.

Ovviamente, se si superassero i 30 caratteri si ripresenterebbe l'errore precedente, quindi bisognerebbe implementare il codice con una sanitizzazione dello stesso in modo tale che l'utente non possa inserire più di un totale di 30 caratteri.



```
Shell No. 1
File Actions Edit View Help
#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;
}
```

```
zip@root@kali:[/home/kali/Desktop]
# ./BOF
Si prega di inserire il nome utente:asbdiuqwhiudqwhduwqhiusdhuaisd
Nome utente inserito: asbdiuqwhiudqwhduwqhiusdhuaisd
```