

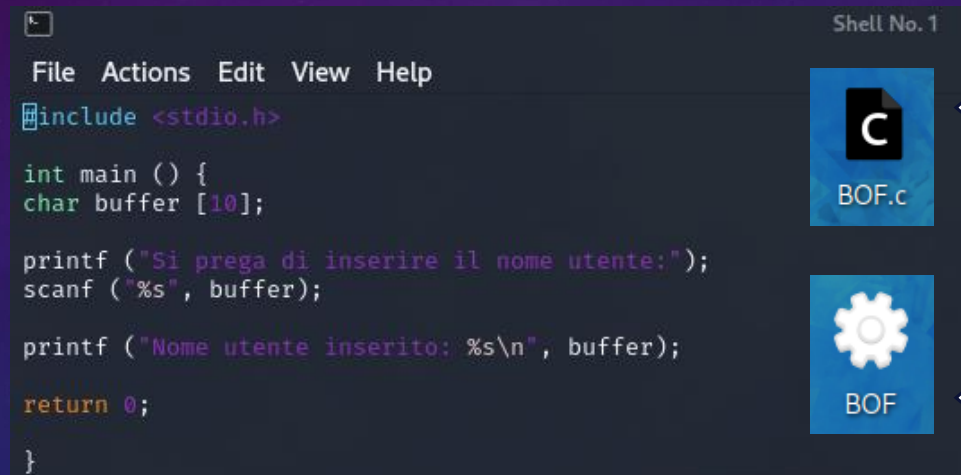
The background is a dark blue gradient with a subtle pattern of white dots. On the left side, there are several concentric circles and a large circular scale with degree markings from 140 to 260. Some circles have arrows indicating a clockwise direction. The overall aesthetic is technical and modern.

BUFFER OVERFLOW

ESERCIZIO S7/L4

IN QUESTO ESERCIZIO VEDIAMO UN SEMPLICE PROGRAMMA SCRITTO IN LINGUAGGIO C ESPOSTO ALLA VULNERABILITÀ DI «BOF» E UNA SITUAZIONE DI ERRORE CHIAMATA «SEGMENTATION FAULT»

- Creiamo il nostro programma su Kali e lo salviamo su Desktop come estensione .c



```
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;
}
```

- Ora con il comando «gcc -g BOF.c -o BOF compiliamo il nostro programma.

(occhio il programma va compilato ogni volta che viene modificato il codice)

Bene ora il nostro programma è pronto per partire

Facciamo partire il nostro programma con il comando «./BOF»

Vediamo il programma in esecuzione che ci propone una stringa dove ci chiede di inserire un nome utente:

Inserendo il nostro nome utente di 4 caratteri non abbiamo nessun problema, ma se proviamo ad inserire più di 10 caratteri ecco che il programma ci ritorna con un errore di «segmentation fault» cos'è?

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

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

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

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

«segmentation fault»

È un errore critico di un'esecuzione di un programma, che si verifica quando quest'ultimo tenta di accedere a una parte della memoria specifica a cui non ne ha il permesso di accedere.

Il nostro buffer può contenere soltanto 10 caratteri, proviamo ad aumentare la dimensione del buffer a 30

```
kali@kali: ~/Desktop
File Actions Edit View Help
GNU nano 7.2 BOF.c *
#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;
}
```

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

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

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

(kali@kali)-[~/Desktop]
$ nano BOF.c

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

Andando ad inserire i caratteri fino a 30 abbiamo una risposta positiva,
ma se proviamo a superare i 30 caratteri il programma ci ritorna di nuovo con un errore di «segmentation fault»

Abbiamo visto che aumentando la dimensione del buffer diamo la possibilità di inserire una stringa di più caratteri, ma per migliorare la funzionalità del nostro programma andiamo a migliorare il nostro codice per ulteriore sicurezza inserendo qualche funzione in più.

- Aggiungiamo la libreria «string.h» per utilizzare la funzione «strlen», che è utilizzata per calcolare la lunghezza di una stringa
- Aggiungiamo anche la funzione «snprintf» che è utilizzata in C per formattare e scrivere una stringa in un buffer, limitando la lunghezza massima della stringa risultante. «snprintf» formatta la stringa e la scrive nel buffer «buffer».

Se la dimensione del buffer non è sufficiente, la stringa viene troncata e verrà restituito un valore superiore o uguale a sizeof(buffer).

Questa funzione è simile a «printf», ma offre un controllo più sicuro sulla lunghezza della stringa risultante per evitare buffer overflow.

IL CODICE OTTIMALE CHE ANDIAMO A PROVARE È QUESTO
COME POSSIAMO VEDERE FUNZIONA CORRETTAMENTE E SODDISFA
I REQUISITI DI SICUREZZA CHE CI INTERESSANO

```
GNU nano 7.2 BOF.c *
#include <stdio.h>
#include <string.h>

int main() {
    char buffer[20];

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

    if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
        // Gestione errore di fgets
        printf("Errore nella lettura dell'input\n");
        return 1;
    }

    size_t length = strlen(buffer);

    // Controllo di overflow
    if (length == sizeof(buffer) - 1 && buffer[length - 1] != '\n') {
        // Il buffer potrebbe essere troppo piccolo
        printf("Errore: Nome utente troppo lungo\n");
        return 1;
    }

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

    return 0;
}
```

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

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

(kali㉿kali)-[~/Desktop]
$ ./BOF
Si prega di inserire il nome utente: Kovakljdkldkhasjkdndjkadjkajkdajk
Errore: Nome utente troppo lungo

(kali㉿kali)-[~/Desktop]
$
```

IL **BUFFER OVERFLOW** È UNA VULNERABILITÀ CRITICA CHE SI VERIFICA QUANDO VENGONO SCRITTI DATI
OLTRE I LIMITI DI UN BUFFER, CORROMPENDO LA MEMORIA.
QUESTO PUÒ CAUSARE Malfunzionamenti, LA COMPROMISSIONE DELLA SICUREZZA
DEL SISTEMA E PERMETTE A UN ATTACCANTE DI ESEGUIRE CODICE DANNOSO.
DURANTE LO SVILUPPO DEL SOFTWARE È ESSENZIALE PER GARANTIRE LA STABILITÀ E LA SICUREZZA
DELLE APPLICAZIONI LA GESTIONE CORRETTA DI QUESTA VULNERABILITÀ.