

# Buffer Overflow

**Samuel Grillo**  
CEO & Fondatore

**Build Week**  
20 Novembre, 2024



# Cos'è un buffer?



## Buffer

Un buffer è un'area di memoria temporanea utilizzata per archiviare dati mentre vengono trasferiti da un luogo a un altro.



## Utilizzo

I buffer aiutano a gestire le differenze di velocità tra componenti hardware o software, evitando ritardi o perdita di dati.



# Buffer Overflow

Una vulnerabilità critica da conoscere e un caso studio

## Sfruttamento dei buffer

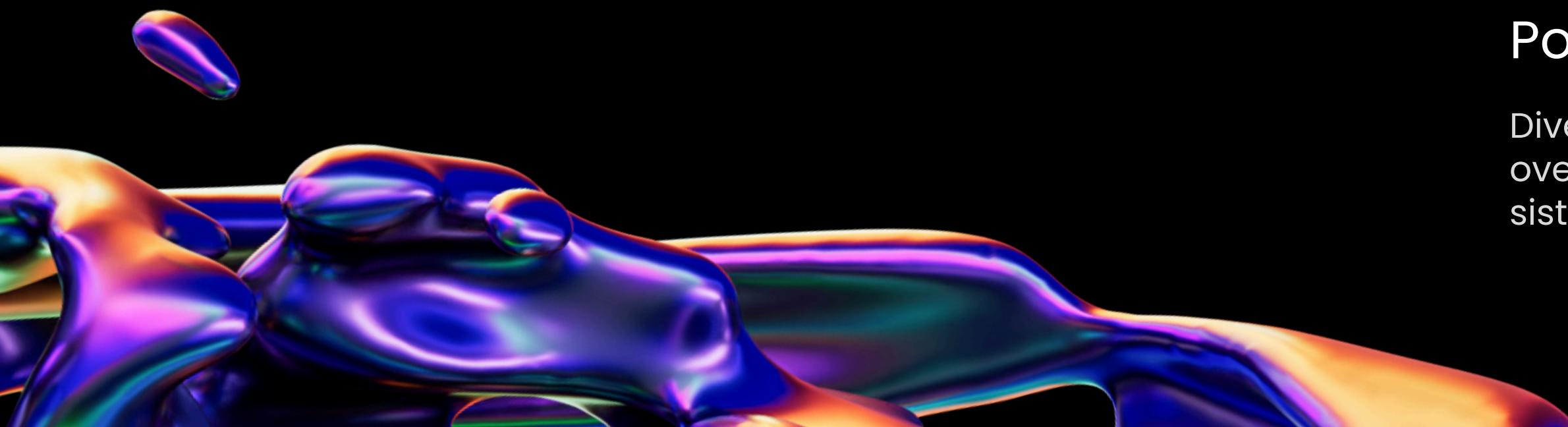
Il programma sfrutta i buffer allocati alla memoria per poter far strapiare il "bicchiere" di informazioni digitali

## Inquinamento della memoria

Il programma corrompe la memoria, contaminando i dati con un flusso incontrollato di informazioni digitali

## Possibilità multiple

Diverse strade per sfruttare il buffer overflow e compromettere l'integrità del sistema



# Buffer in C

In C, i buffer vengono utilizzati per gestire input e output, migliorando l'efficienza nella lettura e scrittura dei dati.

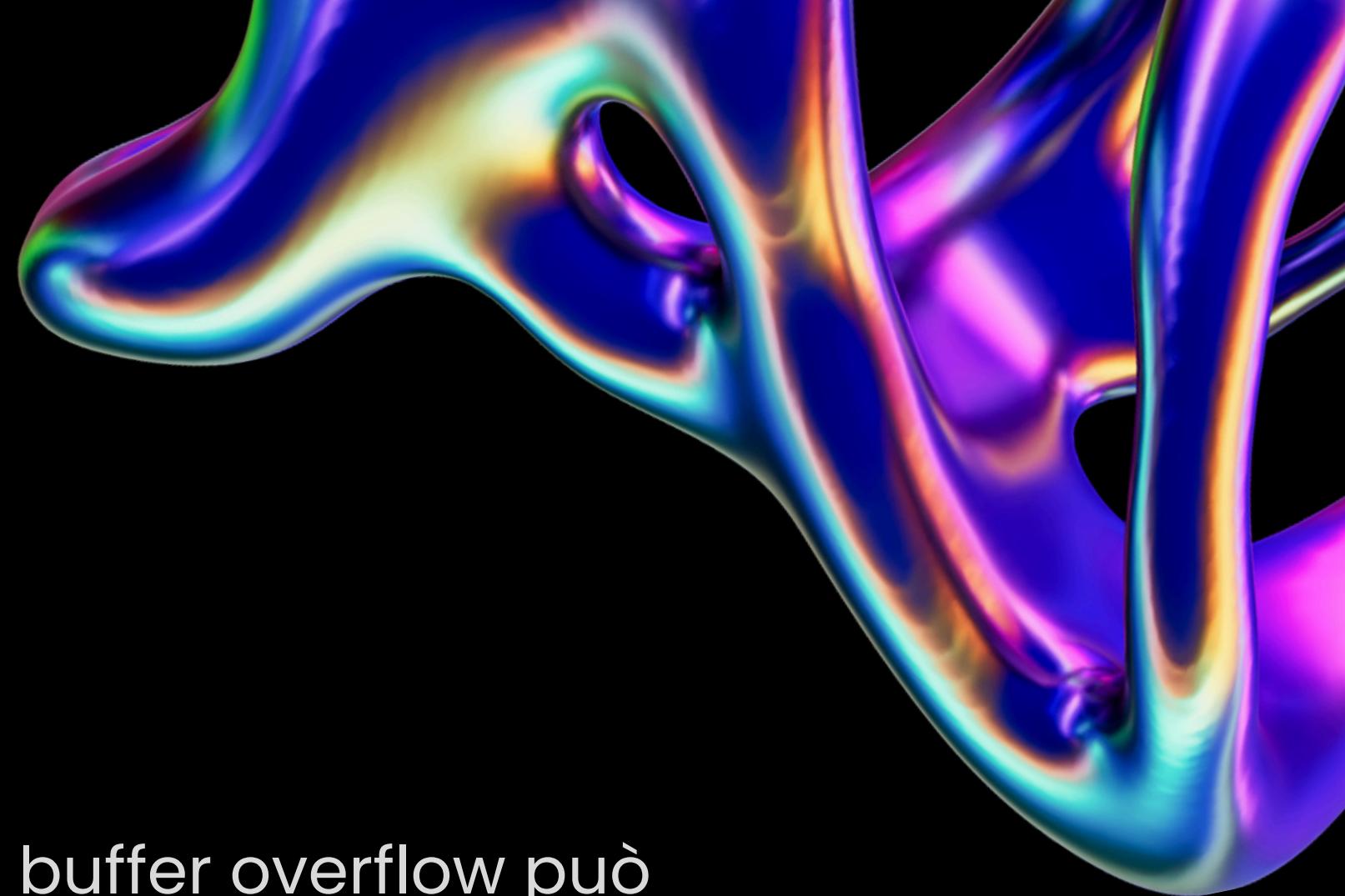
Essi aiutano a prevenire l'accesso diretto alla memoria, proteggendo l'integrità dei dati durante le operazioni di input/output.



# I rischi di C

Il buffer overflow è pericoloso in C perché il linguaggio non offre controlli automatici sui limiti degli array, permettendo a dati extra di sovrascrivere aree critiche della memoria, causando vulnerabilità

In C, un buffer overflow può compromettere la sicurezza del sistema, consentendo a un attaccante di eseguire codice arbitrario o manipolare la memoria per controllare il flusso del programma.



# Buffer Overflow in C



1

## Vulnerabilità iniziale

Il programma alloca un buffer di una dimensione fissa in memoria, ma non verifica se i dati in ingresso superano il limite previsto.

2

## Inquinamento della memoria

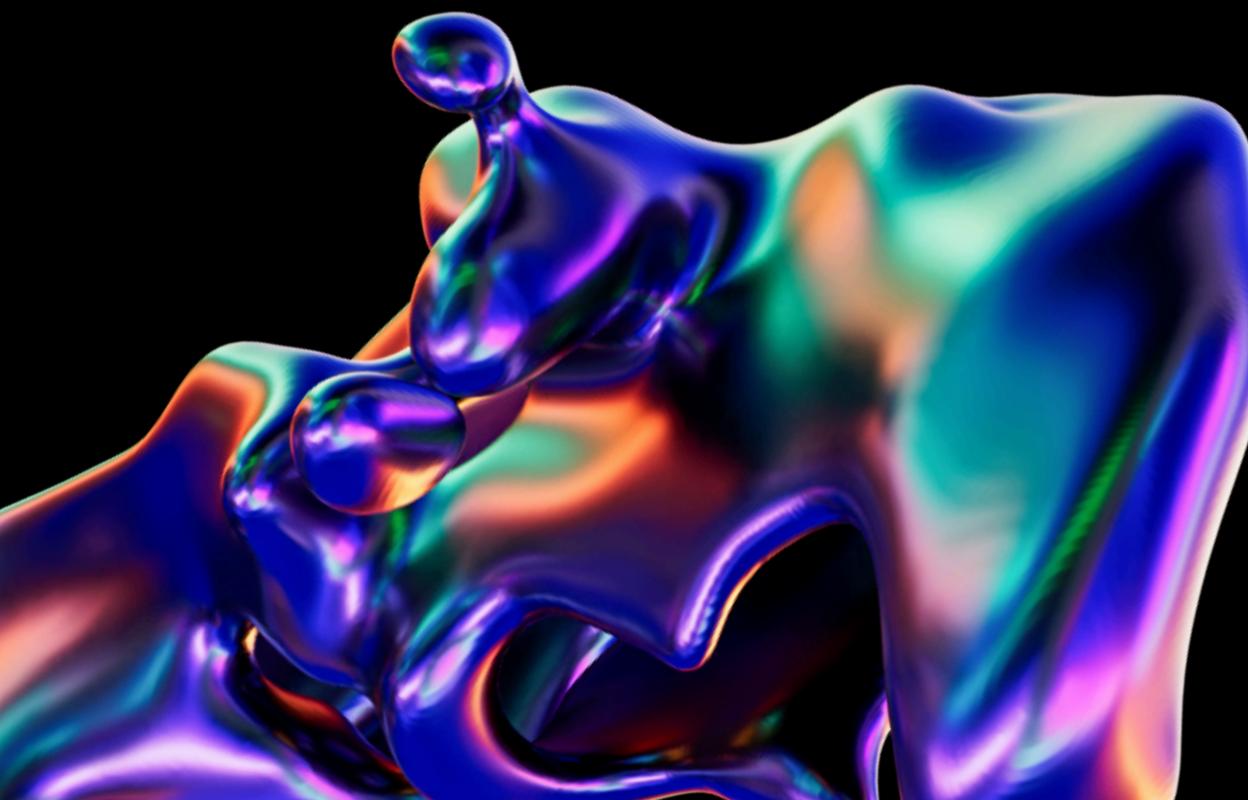
Dati in eccesso vengono inseriti nel buffer, straripando e sovrascrivendo le aree adiacenti di memoria, che possono includere variabili critiche o indirizzi di ritorno.

3

## Esecuzione codice malevolo

Un attaccante sfrutta questa vulnerabilità per inserire codice malevolo o manipolare il flusso di esecuzione, ottenendo il controllo del programma.

# Tipi di exploit



## Esecuzione remota

L'attaccante esegue codice malevolo su un sistema remoto

## Controllo del flusso

Manipolazione degli indirizzi di ritorno o del flusso di esecuzione per ottenere il controllo del programma.

## Iniezione di codice

Un attaccante inserisce codice dannoso in un programma tramite un buffer overflow per manipolare l'esecuzione.

## Compromissione di sistema:

Un buffer overflow consente all'attaccante di acquisire accesso non autorizzato o eseguire operazioni dannose sul sistema.

# Codice originale

```
#include <stdio.h>

int main () {
    int vector [10], i, j, k;
    int swap var;

    printf ("Inserire 10 interi:\n");
    for ( i = 0 ; i < 10 ; i++)
    {
        int c= i+1;
        printf("[%d]: ", c);
        scanf ("%d", &vector[i]);
    }

    printf ("Il vettore inserito e':\n");
    for ( i = 0 ; i < 10 ; i++)
    {
        int t= i+1;
        printf("[%d]: %d", t, vector[i]);
        printf("\n");
    }

    for (j = 0 ; j < 10 - 1; j++)
    {
        for (k = 0 ; k < 10 - j - 1; k++)
        {
            if (vector[k] > vector[k+1])
            {
                swap_var=vector[k];
                vector[k]=vector[k+1];
                vector[k+1]=swap_var;
            }
        }
    }
    printf("Il vettore ordinato e':\n");
    for (j = 0; j < 10; j++)
    {
        int g = j+1;
        printf("[%d]: ", g);
        printf("%d\n", vector[j]);
    }
    return 0;
}
```

## Esecuzione del codice

Il codice chiede all'utente di inserire 10 numeri interi, che vengono poi memorizzati in un array (vettore). Successivamente, il programma stampa il vettore inserito e lo ordina in ordine crescente utilizzando l'algoritmo di ordinamento "bubble sort".

Dopo aver ordinato i numeri, il programma mostra il vettore finale, che contiene gli stessi numeri, ma disposti dal più piccolo al più grande. Questo permette all'utente di vedere il risultato dell'ordinamento.



Il codice è innanzitutto stato letto per intero per avere un'idea di cosa ci aspettasse; è stato definito l'array, che avremmo usato numeri interi, le variabili e anche non conoscendo il linguaggio di C i printf per indicare all'utente dove inserire i dati e successivamente quelli per stampare i risultati ci fanno intuire che è un programma di ordinamento di vettori

# Primo esempio di exploit



```
#include <stdio.h>

int main () {
    int vector [10], i, j, k;
    int swap_var;

    printf ("Inserire 10 interi:\n");
    for ( i = 0 ; i < 11 ; i++)
    {
        int c= i+1;
        printf("[%d]:", c);
        scanf ("%d", &vector[i]);
    }

    printf ("Il vettore inserito e':\n");
    for ( i = 0 ; i < 11 ; i++)
    {
        int t= i+1;
        printf("[%d]: %d", t, vector[i]);
        printf("\n");
    }

    for (j = 0 ; j < 10 - 1; j++)
    {
        for (k = 0 ; k < 10 - j - 1; k++)
        {
            if (vector[k] > vector[k+1])
            {
                swap_var=vector[k];
                vector[k]=vector[k+1];
                vector[k+1]=swap_var;
            }
        }
    }
    printf("Il vettore ordinato e':\n");
    for (j = 0; j < 10; j++)
    {
        int g = j+1;
        printf("[%d]:", g);
        printf("%d\n", vector[j]);
    }

    return 0;
}
```

Inserire 10 interi:

[1]:1  
[2]:2  
[3]:3  
[4]:4  
[5]:5  
[6]:6  
[7]:7  
[8]:8  
[9]:9  
[10]:10  
[11]:11

Il vettore inserito e':

[1]: 1  
[2]: 2  
[3]: 3  
[4]: 4  
[5]: 5  
[6]: 6  
[7]: 7  
[8]: 8  
[9]: 9  
[10]: 10  
[11]: 11

Il vettore ordinato e':

[1]:1  
[2]:2  
[3]:3  
[4]:4  
[5]:5  
[6]:6  
[7]:7  
[8]:8  
[9]:9  
[10]:10

\*\*\* stack smashing detected \*\*\*: terminated

...Program finished with exit code 134  
Press ENTER to exit console.[]

## Dimostrazione del concetto di Buffer Overflow

Modificando il numero di input consentiti nel programma, inserendo 11 valori invece di 10, si verifica un buffer overflow, che causa lo straripamento dei dati e potenzialmente la corruzione della memoria.

Quando il programma viene compilato online e testato, l'inserimento di un dato in eccesso genera un stack smashing error, indicando che la memoria adiacente al buffer è stata sovrascritta in modo pericoloso.

Lo stack smashing avviene quando un programma scrive oltre i limiti di uno stack, sovrascrivendo dati cruciali, come variabili o indirizzi di ritorno, causando vulnerabilità.

# Secondo esempio di exploit



```
File Edit Search View Document Help
+ F S X C M I Q R E
1 #include <stdio.h>
2
3 int main() {
4     int vector[10], i, j, k;
5     int swap_var;
6
7     // Inserimento dei primi 10 numeri da parte dell'utente
8     printf("Inserire 10 interi:\n");
9     for (i = 0; i < 10; i++) {
10         printf("[%d]: ", i + 1);
11         scanf("%d", &vector[i]);
12     }
13
14     // Riempimento automatico di altri 200 numeri nel buffer (sovrascrivendo)
15     for (i = 10; i < 210; i++) { // Dal 10° al 210° elemento
16         vector[i] = i; // Assegna valori sequenziali o un pattern desiderato
17     }
18
19     // Stampa del vettore inserito
20     printf("Il vettore inserito e':\n");
21     for (i = 0; i < 10; i++) {
22         printf("[%d]: %d\n", i + 1, vector[i]);
23     }
24
25     // Ordinamento del vettore (solo i primi 10 valori per il bubble sort)
26     for (j = 0; j < 10 - 1; j++) {
27         for (k = 0; k < 10 - j - 1; k++) {
28             if (vector[k] > vector[k + 1]) {
29                 swap_var = vector[k];
30                 vector[k] = vector[k + 1];
31                 vector[k + 1] = swap_var;
32             }
33         }
34     }
35
36     // Stampa del vettore ordinato (solo i primi 10 valori)
37     printf("Il vettore ordinato e':\n");
38     for (j = 0; j < 10; j++) {
39         printf("[%d]: %d\n", j + 1, vector[j]);
40     }
41
42     return 0;
43 }
44
```

## Exploit più impattante

L'exploit sfrutta un programma che, dopo i 10 input iniziali dell'utente, inietta automaticamente 200 valori, causando un buffer overflow e sovrascrivendo aree di memoria, senza necessità di ulteriori input manuali.

Questo approccio è particolarmente insidioso perché l'utente non deve inserire più di 10 valori, riducendo la probabilità che l'errore venga notato, mentre il programma continua ad apparire funzionante ma vulnerabile.

# Funzionamento codice BOF

```
(root㉿kali)-[~/home/kali/Desktop]
# ./bof
Inserire 10 interi:
[1]: 1
[2]: 2
[3]: 3
[4]: 4
[5]: 5
[6]: 6
[7]: 7
[8]: 8
[9]: 9
[10]: 10
Il vettore inserito e':
[1]: 1
[2]: 2
[3]: 3
[4]: 4
[5]: 5
[6]: 6
[7]: 7
[8]: 8
[9]: 9
[10]: 10
Il vettore ordinato e':
[1]: 1
[2]: 2
[3]: 3
[4]: 4
[5]: 5
[6]: 6
[7]: 7
[8]: 8
[9]: 9
[10]: 10
zsh: segmentation fault ./bof
(root㉿kali)-[~/home/kali/Desktop]
#
```

## Segmentation fault

Abbiamo testato il codice su Kali Linux, dove, dopo l'inserimento dei 10 input, si verifica un segmentation fault a causa della sovrascrittura della memoria causata dal buffer overflow.

Una volta raggiunto il limite degli input, il programma genera un segmentation fault, dimostrando come l'overflow del buffer possa corrompere la memoria e interrompere l'esecuzione corretta.

Un segmentation fault si verifica quando un programma tenta di accedere a memoria non valida o protetta, causando l'interruzione dell'esecuzione per evitare danni al sistema.