```c
#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 >= 0 ; 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;


}
```

Il codice fornito è un programma in linguaggio C che ordina un vettore di 10 interi inseriti dall'utente utilizzando l'algoritmo di ordinamento a bolle (bubble sort).

Di seguito, spiegherò ogni parte del codice riga per riga.

L'indice del vettore viene utilizzato per memorizzare gli input dell'utente nelle posizioni corrette del vettore.

1° CICLO FOR

Riga 11: Questa riga inizia un ciclo for che esegue 10 iterazioni. La variabile i viene inizializzata a 0 e viene incrementata di 1 ad ogni iterazione. Il ciclo viene eseguito fintanto che i è minore di 10.

Riga 13: All'interno del ciclo, viene dichiarata una variabile c che rappresenta un contatore incrementato di 1. Questo contatore è utilizzato per numerare in modo chiaro l'input dell'utente, in quanto gli indici dei vettori in C partono da 0, ma l'output viene formattato per partire da 1.

Riga 14: Questa riga utilizza la funzione printf per stampare a schermo un prompt per l'utente, chiedendo di inserire un intero. La stringa di formattazione ("[%d]:") specifica che deve essere stampato un intero seguito dai due punti. L'intero effettivo stampato è il valore di c, il contatore incrementato di 1.

Riga 15: Utilizza la funzione scanf per acquisire l'input dell'utente. La stringa di formattazione ("%d") indica chedeve essere letto un intero. Il valore inserito dall'utente viene memorizzato nell'elemento corrente del vettore, cioè vector[i].

Riga 19: Stampa il vettore inserito dall'utente per conferma.

2° CICLO FOR

Riga 20: Questa parte di codice stampa a schermo ogni elemento del vettore insieme al suo indice incrementato di 1, fornendo una rappresentazione chiara e numerata degli elementi inseriti dall'utente. Questa riga inizia un ciclo for che esegue 10 iterazioni. La variabile i viene inizializzata a 0 e viene incrementata di 1 ad ogni iterazione. Il ciclo viene eseguito fintanto che i è minore di 10.

Riga 22: All'interno del ciclo, viene dichiarata una variabile t che rappresenta l'indice corrente del vettore incrementato di 1. Questo è fatto perché gli indici dei vettori in C partono da 0, ma l'output viene formattato per partire da 1.

Riga 23: Questa riga utilizza la funzione printf per stampare a schermo il contenuto del vettore e il suo indice formattato. La stringa di formattazione ("[%d]: %d") specifica che devono essere stampati due valori: un intero (%d) e un altro intero (%d). I valori effettivi sono forniti come argomenti successivi alla stringa di formattazione. t è l'indice incrementato di 1 e vector[i] è il valore corrente del vettore all'indice i.

Riga 24: Aggiunge una nuova linea per separare ogni coppia indice-valore nel vettore.

3° CICLO FOR

Riga 28: Quindi, il doppio ciclo for implementa l'algoritmo di bubble sort, che confronta e scambia gli elementi del vettore fino a quando l'intero vettore è ordinato in modo crescente. Inizia un ciclo esterno che esegue 9 iterazioni (fino a j = 8). Il motivo per cui si ferma a 10 - 1 è che l'algoritmo confronta due elementi adiacenti nel vettore, quindi nell'ultima iterazione, il confronto sarà fatto con gli ultimi due elementi.

Riga 30: All'interno del ciclo esterno, inizia un ciclo interno che esegue un numero decrescente di iterazioni a ogni passaggio del ciclo esterno. Questo è tipico dell'algoritmo di bubble sort, dove alla fine di ogni passaggio del ciclo esterno, il valore massimo raggiunge la sua posizione finale.

Riga 32: Controlla se l'elemento corrente (vector[k]) è maggiore dell'elemento successivo (vector[k + 1]). Se la condizione è vera, significa che è necessario scambiare i due elementi per ottenere un ordinamento crescente.

Riga 34: Salva il valore dell'elemento corrente vector[k] nella variabile temporanea swap_var. Questo passo è fondamentale poiché, dopo lo scambio, avremo bisogno del valore originale dell'elemento corrente per assegnarlo alla nuova posizione. Riga 35 Sovrascrive l'elemento corrente vector[k] con il valore

dell'elemento successivo vector[k + 1]. Questo passo è parte dello scambio, in modo da spostare il valore maggiore verso la fine del vettore.

Riga 36: Assegna il valore originale dell'elemento corrente (salvato in swap_var) alla posizione successiva vector[k + 1]. Questo completa lo scambio, posizionando il valore più piccolo nella posizione successiva nel vettore.

Riga 40: Stampa il vettore ordinato.

4° CICLO FOR

Riga 41: In sintesi, questa parte di codice stampa a schermo ogni elemento del vettore ordinato insieme al suo indice incrementato di 1, fornendo una rappresentazione chiara enumerata del vettore ordinato. Inizia un ciclo for che esegue 10 iterazioni. La variabile j viene inizializzata a 0 e viene incrementata di 1 ad ogni iterazione. Il ciclo viene eseguito fino a quando j è minore di 10.

Riga 43: All'interno del ciclo, viene dichiarata una variabile g che rappresenta l'indice corrente del vettore incrementato di 1. Questo è fatto perché gli indici dei vettori in C partono da 0, ma l'output viene formattato per partire da 1.

Riga 44: Questa riga utilizza la funzione printf per stampare a schermo un prompt numerato, indicando l'indice dell'elemento corrente nel vettore. La stringa di formattazione ("[%d]:") specifica che deve essere stampato un intero seguito dai due punti. L'intero effettivo stampato è il valore di g, l'indice incrementato di 1.

Riga 45: Questa riga utilizza la funzione printf per stampare a schermo il valore dell'elemento corrente nel vettore. La stringa di formattazione ("%d\n") specifica che deve essere stampato un intero seguito da un carattere di nuova linea, che va a capo. L'intero effettivo stampato è il valore dell'elemento nel vettore all'indice j.

Riga 48: Restituisce 0 per indicare al sistema operativo che il programma è stato eseguito correttamente.

BUFFER OVERFLOW

Nel contesto informatico, "BOF" sta per "Buffer Overflow" (letteralmente "Trabocco del Buffer"). Un buffer è una zona di memoria temporanea utilizzata per immagazzinare dati mentre vengono trasferiti da un'area all'altra. Il "buffer overflow" si verifica quando vengono scritti più dati di quelli che il buffer può contenere. Questo può portare a problemi di sicurezza e a comportamenti imprevisti del programma, poiché i dati in eccesso possono sovrascrivere parti della memoria che contengono altre informazioni critiche o codice eseguibile. Le vulnerabilità di buffer overflow sono state sfruttate da molti attacchi informatici, inclusi quelli che consentono agli hacker di eseguire codice dannoso o prendere il controllo di un sistema. Pertanto, gli sviluppatori di software prestano molta attenzione alla gestione dei buffer per prevenire tali problemi di sicurezza.

IMPATTO

L'impatto di una vulnerabilità di tipo Buffer Overflow è tendenzialmente molto critico e può portare, principalmente, a queste conseguenze:

Esecuzione di codice arbitrario: l'attaccante può eseguire del codice sul sistema, consentendogli di prendere il controllo del sistema o di rubare dati, ovviamente con rispetto ai privilegi con cui quel programma/servizio è stato eseguito. Nel caso in cui il programma sia inteso per essere fruito attraverso internet, questa tipologia di vulnerabilità può essere anche catalogata come Remote Code Execution

Disservizio: mediante una vulnerabilità di tipo Buffer Overflow è possibile ottenere un disservizio sotto vari aspetti, di seguito elencati. In ogni caso, in molti dei casi elencati l'effetto complessivo è quello di ottenere un Denial of Service:

Crash del programma: il programma in esecuzione può dunque essere arrestato in modo non previsto causando, potenzialmente, anche perdita di dati.

Memory Consumption: il programma può esaurire tutta la memoria disponibile sul sistema, rendendolo dunque inutilizzabile

Distorsione dei dati: avendo controllo sul buffer, i dati al suo interno possono essere danneggiati rendendoli dunque potenzialmente illeggibili o inutilizzabili

Instabilità del sistema: il sistema operativo può diventare instabile causando, a sua volta, crash o altri problemi operativi.

L'impatto è quindi statisticamente molto critico.

TIPOLOGIE D'ATTACCO

Esistono diversi tipi di vulnerabilità Buffer Overflow, sostanzialmente distinte a seconda delle posizioni in cui si verifica l'errore. Quelli più comuni sono:

Overflow dello stack: lo stack è quella parte della memoria usata per memorizzare i record di attivazione delle funzioni costituiti da:

L'indirizzo di codice dell'istruzione successiva a quella che ha invocato la funzione (indirizzo di ritorno o "return address");

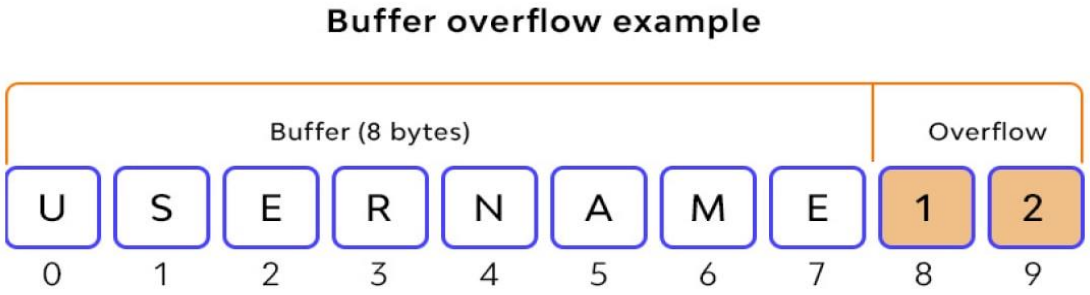
I parametri della funzione

Le variabili locali della funzione.

Overflow del heap: l'heap, o memoria dinamica, è la parte della memoria che consente di allocare spazi di memoria la cui dimensione si conosce solo in fase di esecuzione. Viene utilizzata per memorizzare oggetti creati durante l'esecuzione del programma: quando un oggetto viene creato, infatti, l'heap alloca spazio per l'oggetto stesso e restituisce un puntatore all'oggetto in modo che il programma possa accedere all'oggetto.

Overflow di memoria condivisa: la memoria condivisa è un tipo di memoria condivisa tra più processi che viene utilizzata per migliorare l'efficienza dei programmi consentendo loro di accedere agli stessi dati senza doverli copiare tra i processi. Ad esempio, se l'area di memoria contiene un puntatore ad un'altra area di memoria, l'overrun potrebbe modificare il puntatore.

wallarm



```c
#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 >= 0 ; i++)
        {
        int t= i+1;
        printf("[%d]: %d", t, vector[i]);
    printf("\n");
    }
```

Andando a modificare il codice nella riga 20 come in figura, diamo la possibilità al programma di andare in Buffer overflow, permettendo l'inserimento di un ciclo for infinito.

Quindi, una volta che il ciclo viene inizializzato, continuerà ad eseguire il suo blocco di istruzioni senza mai terminare, poiché la condizione di uscita non verrà mai soddisfatta.

```
[1389]: 1380275295
[1390]: 1346454349
[1391]: 1030714207
[1392]: 825252635
[1393]: 1832071995
[1394]: 1397050368
[1395]: 1163157331
[1396]: 1094929746
[1397]: 1702059856
[1398]: 811277117
[1399]: 1162608749
[1400]: 1415533395
[1401]: 1129140805
[1402]: 1969180737
[1403]: 1528511859
[1404]: 842218289
[1405]: 1162608749
[1406]: 1415533395
[1407]: 1129140805
[1408]: 1969180737
[1409]: 1528511845
[1410]: 1593863472
[1411]: 1869098813
[1412]: 1798268269
[1413]: 795438177
[1414]: 1802724676
[1415]: 795897716
[1416]: 1329737518
[1417]: 791543878
[1418]: 4607810
[1419]: 0
[1420]: 0
zsh: segmentation fault  ./BOF
```