ERRORI COMUNI
sabato 24 dicembre 2022 10:56

# ERRORI COMUNI

## 1. mancato controllo di allocazione

- `malloc()` non garantisce il successo dell'operazione:
  - se non c'è memoria disponibile
  - se il programma ha superato il limite di memoria che può utilizzare
- In questo caso, viene restituito `NULL`.
- Molto spesso i programmatori non verificano il valore restituito da `malloc()`:
  - È possibile accedere a un puntatore impostato a `NULL`
  - SEGFAULT!

## 2. memory leak

È uno sgocciolamento di memoria.

IL PROGRAMMA RICHIEDE DINAMICAMENTE LA MEMORIA UTILIZZANDO MALLOC, POI PERDE RIFERIMENTO A QUELL'ARIA DI MEMORIA PERCHÈ MAGARI SOVRASCRIVE QUEL PUNTATORE CHE PUNTAVA A QUELL'AREA, QUELL'AREA DI MEMORIA È DEFINITIVAMENTE IRRAGGIUNGIBILE, E SE NON AVEVO CHIAMATO FREE, QUELL'AREA DI MEMORIA È PERSA PER SEMPRE!

Se noi commettiamo questo errore sui nostri PC per programmi didattici, una volta finita l'esecuzione del programma, tutta la memoria allocata dinamicamente viene rilasciata!

E NON È UN PROBLEMA!

MA SE L'APPLICAZIONE È UN SERVER E ABBIAMO UN PICCOLO LEAK DOVE PERDIAMO 1 BYTE ALLA VOLTA, QUESTI LEAK SI ACCUMULANO, E A UN CERTO PUNTO L'APPLICAZIONE MUORE.

## Un esempio di memory leak

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

bool leak(void)
{
        char *s = malloc(4096);

        if(s == NULL) {
                return false;
        } else {
                s[0] = 'A';
                return true;
        }
}

int main(void)
{
        while(leak());
        return 0;
}
```

- Un memory leak si genera quando ad una chiamata a `malloc()` non corrisponde una chiamata a `free()`

# 3. use after free

Se io chiamo MALLOC, chiamo free e dereferenzio il puntatore, Undefined Behaviour;
E questo undefined behaviour avviene anche quando passiamo il puntatore ad una funzione che lo dereferenzierà!

```c
int *ptr = malloc(sizeof(int));
free(ptr);
*ptr = 0; /* undefined behavior */
printf("%p", ptr); /* undefined behavior */
```

Dopo una free scrivo esplicitamente NULL! ← PER EVITARE ;

- Ci sono alcune tecniche per ridurre la probabilità di accedere a *dangling pointer* (puntatori pendenti)
- Resettare esplicitamente il puntatore:

```c
free(ptr);
ptr = NULL;
```

- Utilizzare correttamente gli scope delle variabili (laddove possibile):

```c
int *ptr = NULL;
{
  int *ptr = malloc(sizeof(int));
  free(ptr);
}
*ptr = 200; /* the application will crash! */
```

# 4. freeing wrong pointers

SE ALLA FREE() PASSO UN PUNTATORE CHE NON È STATO RESTITUITO DA MALLOC HO UNDEFINED BEHAVIOUR;

```c
char *msg = "Una stringa globale";
int tbl[100];
free(msg); /* undefined behavior */
free(tbl); /* undefined behavior */
```

↳ IL VETTORE DECADE IN UN PUNTATORE

5

## double free corruption

- Una *corruzione da doppia liberazione* si verifica quando si chiede alla libreria di liberare più volte lo stesso buffer di memoria

```c
int *ptr = malloc(sizeof(int));
free(ptr);
free(ptr);
```

- Si tratta di un ulteriore *undefined behavior*
- Nel caso peggiore, l'applicazione potrebbe andare in crash molto tempo dopo, quando si chiede nuova memoria tramite `malloc()`