UNIVERSIDAD DE MÁLAGA

PROGRAMACIÓN SEGURA

Práctica 4: programación segura en memoria dinámica

CIBERSEGURIDAD E INTELIGENCIA ARTIFICIAL

Pablo Jesús Delgado Muñoz

Noviembre de 2024

ÍNDICE GENERAL

CAPÍTULO I: PREPARACIÓN Y ANÁLISIS

Abrimos el código original:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void q0(int*qfoo,int qbar){for(int qbaz=0;qbaz<=qbar;qbaz++)
{printf("%d ",qfoo[qbaz]);}printf("\n");}int*q1(int
qbar){int*qfoo=(int*)malloc(qbar*sizeof(int)
);for(int qbaz=0;qbaz<qbar;qbaz++){qfoo[qbaz]=rand()
%100;}return qfoo;}int main(){srand(time(NULL));int
qbar;printf("Ingrese el tamaño del Array: ");
scanf("%d",&qbar);int*q2=q1(qbar);q0(q2,qbar);
free(q2);free(q2);return 0;}
```

1.1 Preparación del código

Podemos observar que el código no está totalmente ordenado, por lo tanto emplearemos el comando Indent para evitar errores y hacer de forma automática el indentado del código.

Por lo tanto usaremos indent para que podamos trabajar con él.

1.2 Análisis código preparado

Abrimos el código de nuevo y podemos ver que ya está correctamente formateado.

ps@programacionsegura:~/Documents/Prac4PS$ indent practica3_13.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void q0 (int *qfoo, int qbar) {
  for (int qbaz = 0; qbaz <= qbar; qbaz++)
    {
    printf ("%d ", qfoo[qbaz]);
    } printf ("\n");
} int * q1 (int qbar)
{
  int *qfoo = (int *) malloc (qbar * sizeof (int));
  for (int qbaz = 0; qbaz < qbar; qbaz++)
    {
      qfoo[qbaz] = rand () % 100;
    } return qfoo;
}
int main ()
{
  srand (time (NULL));
  int qbar;
  printf ("Ingrese el tamaño del Array: ");
  scanf ("%d", &qbar);
  int *q2 = q1 (qbar);
  q0 (q2, qbar);
  free (q2);
  free (q2);
  return 0;
}
```

1.3 Errores de memoria dinámica

Vamos a buscar errores:
1. Free doble - Solución: quitar un free.
Nada más compilar y tratar de ejcutar vemos que el programa no funciona correctamente y se nos avisa del uso de un doble free.
2. Error en el bucle q0 - Solución: cambiar <= por <.

```
    void q0 (int *qfoo, int qbar)
{
  for (int qbaz = 0; qbaz < qbar; qbaz++) { //acceso fuera de los lim
      printf ("%d ", qfoo[qbaz]); //qfoo[qbaz == qbar] no existe
    }
    printf ("\n");
```

3.**Falta de verificación de asignación de memoria exitosa**:

- La función `q1` no verifica si la llamada a `malloc` devuelve un puntero nulo, lo que podría llevar a la pérdida de la referencia del puntero nulo si la memoria no se asigna correctamente.
- **Solución**: Añadir una comprobación `if (qfoo == NULL)` después de la llamada a `malloc`.

```
    if (qfoo == NULL) {
     printf("Error en la asignación de memoria. \n");
     exit(1);
}
```

4. Inicialización insegura del tamaño del array:

- El tamaño del arreglo `qbar` es leído desde la entrada sin validación. Esto puede provocar problemas si el usuario introduce un valor negativo o demasiado grande.
- **Solución**: Validar que `qbar > 0` antes de proceder con la asignación.

```
    if (qbar <= 0) {
        printf("Tamaño qbar inválido. \n");
        return 1;
      }
```

1.4 Mejoras y test en valgrind

1. Usaremos calloc en lugar de malloc para asegurar la incialización de la memoria a cero.

2. Nos aseguramos de que rand() sí crea un número aleatorio mayor o igual a cero.

```
  for (int qbaz = 0; qbaz < qbar; qbaz++) {
      int random = rand(); //Se supone que rand() dará números mayore
      if (random < 0) {  //pero desconfío de ello...
          printf("Error en la creación de un random. \n");
          exit(1);
```

3. Sobreescribir la memoria que es liberada, para así evitar posibles leaks de datos.

```
  //free (q2); Comentamos uno de los free.
  for (int i = 0; i < qbar; i++) {
      q2[i] = 0; //sobreescribimos la memoria a 0
      antes de hacer el free.
}
  free (q2);
```

Ingrese el tamaño del Array: 99999999999999999999999999999999999999999999...

Tamaño qbar inválido.
==5954==
==5954== HEAP SUMMARY:
==5954== in use at exit: 0 bytes in 0 blocks
==5954== total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==5954==
==5954== All heap blocks were freed – no leaks are possible
==5954==
==5954== For lists of detected and suppressed errors, rerun with: -s
==5954== ERROR SUMMARY**: 0 errors from 0 contexts** (suppressed: 0 from 0)

Vemos que valgrind devuelve 0 errores y que el programa nos da como salida que el tamaño de qbar es inválido ya que dicha entrada es demasaido grande como para guardarse en un int, por lo tanto saldrá del programa y se han corregido los errores de buffer overflow. Además hemos usado técnicas de programación defensiva al haber incorporado calloc, y también al haber limitado la entrada de qbar a solo números mayores a cero.

1.5 CÓDIGO FINAL

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void q0 (int *qfoo, int qbar)
{
  for (int qbaz = 0; qbaz < qbar; qbaz++) { //acceso fuera de los lim
      printf ("%d ", qfoo[qbaz]); //qfoo[qbaz == qbar] no existe
    }
    printf ("\n");
} int *q1 (int qbar)
{
  int *qfoo = (int *) calloc (qbar, sizeof (int)); //cambio a calloc
  //comprobacion de asignacion de memoria exitoso
  if (qfoo == NULL) {
      printf("Error en la asignación de memoria. \n");
      exit(1);
}
  for (int qbaz = 0; qbaz < qbar; qbaz++) {
      int random = rand(); //Se supone que rand() dará números mayore
      if (random < 0) {  //pero desconfío de ello...
          printf("Error en la creación de un random. \n");
          exit(1);
    }
      qfoo[qbaz] = random % 100;
    }
    return qfoo;
}

int main()
{
```

```
  srand (time (NULL));
  int qbar;
  printf ("Ingrese el tamaño del Array: ");
  scanf ("%d", &qbar);
  if (qbar <= 0) {
      printf("Tamaño qbar inválido. \n");
      return 1;
    }
  int *q2 = q1 (qbar);
  q0 (q2, qbar);
  //free (q2); Comentamos uno de los free.
  for (int i = 0; i < qbar; i++) {
      q2[i] = 0; //sobreescribimos la memoria a 0 antes
    de hacer el free.
}
  free (q2);
  return 0;
}
```