**Scenario: Memory Leak Investigation in C**

**Summary:**  
Detected and fixed a memory leak in a loop allocation scenario, using GDB and Valgrind to diagnose the issue.

**Problem Description**

The program allocated memory repeatedly inside a loop without freeing previous allocations, causing **memory leaks**. This type of bug can cause crashes or high memory usage in production.

**Tools & Languages**

* **Language:** C
* **Debugging Tools:** GDB, Valgrind

**Initial Code**

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr;

int n = 5;

for (int i = 0; i < n; i++) {

arr = (int \*)malloc(10 \* sizeof(int)); // memory allocated each loop

arr[i] = i \* 2;

printf("arr[%d] = %d\n", i, arr[i]);

// forgot to free → memory leak

}

return 0;

}

**Issue Detected:**

* Valgrind reported 4 lost blocks (160 bytes)
* GDB inspection revealed changing pointer addresses across loop iterations

**Fixed Code**

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr;

int n = 5;

for (int i = 0; i < n; i++) {

arr = (int \*)malloc(10 \* sizeof(int));

if (!arr) return 1;

arr[i] = i \* 2;

printf("arr[%d] = %d\n", i, arr[i]);

free(arr); // memory freed immediately

}

return 0;

}

**Outcome**

* No memory leaks confirmed with Valgrind
* Program runs as expected:

arr[0] = 0

arr[1] = 2

arr[2] = 4

arr[3] = 6

arr[4] = 8

**Lessons Learned**

* Always pair malloc with free to avoid memory leaks
* Carefully consider memory allocation in loops
* GDB and Valgrind are key tools for diagnosing and verifying memory issues