**Scenario - Array Bounds Breach in C**

**Scenario / Summary**  
Investigated and fixed an out-of-bounds array access in C. The program wrote past the allocated array size, causing undefined behavior. GDB was used to identify the exact line where memory corruption occurred.

**Problem Description**  
The original program declared an array of 5 integers but attempted to write to 11 elements. This type of bug can lead to program crashes, corrupted memory, or unpredictable behavior in production systems.

**Tools & Languages**

* Language: C
* Debugging Tools: GDB

**Initial (Buggy) Code**

#include <stdio.h>

int main() {

int arr[5]; // only 5 elements: arr[0]..arr[4]

// intentionally go out of bounds

for (int i = 0; i <= 10; i++) {

arr[i] = i \* 3; // writing past arr[4]

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

}

return 0;

}

**Issue Detected**

* Running the program produced incorrect or undefined output after arr[4].
* GDB revealed that the program attempted to access memory outside the array bounds, leading to memory corruption.

**Fixed Code**

#include <stdio.h>

int main() {

int arr[5]; // valid indices: 0-4

int n = 5;

for (int i = 0; i < n; i++) { // fix: loop within bounds

arr[i] = i \* 3;

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

}

return 0;

}

**Outcome / Validation**

* Program runs as expected:

arr[0] = 0

arr[1] = 3

arr[2] = 6

arr[3] = 9

arr[4] = 12

* No out-of-bounds access detected with GDB.

**Lessons Learned**

* Always ensure loops respect array boundaries.
* Off-by-one errors are a common source of memory corruption in C.
* GDB is an essential tool for pinpointing memory access issues.