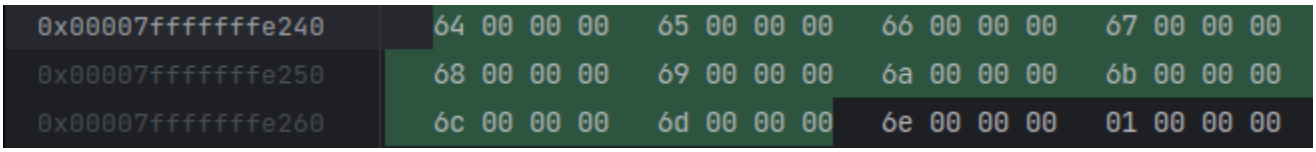
McDonnell – Analyzing Simple Buffer Overflow Conditions

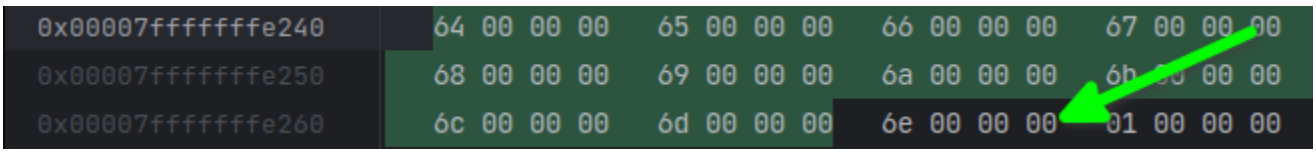
Code Problem

First to understand what's happening here we must analyze our memory structure as follows.

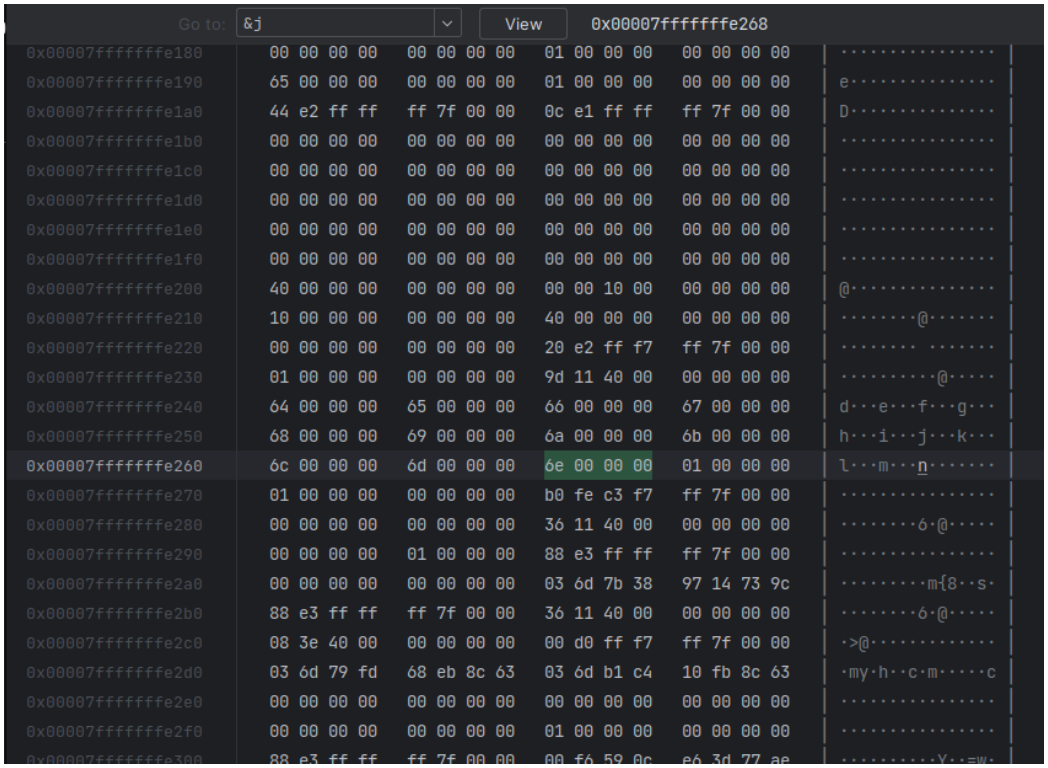
We first declare a 10 element 4-byte integer buffer whose address structure and stack space of is highlighted in green here:



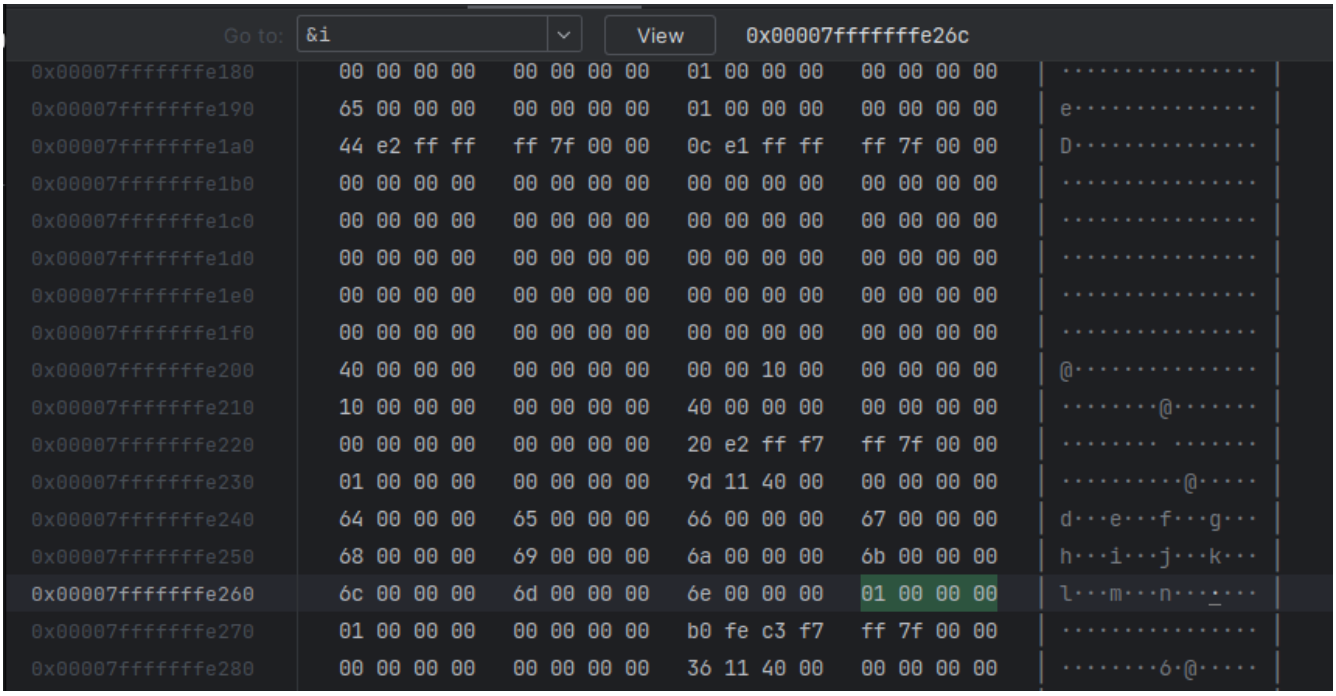
Next, the green arrow is pointing to the single 4-byte address space for our incremental integer j:



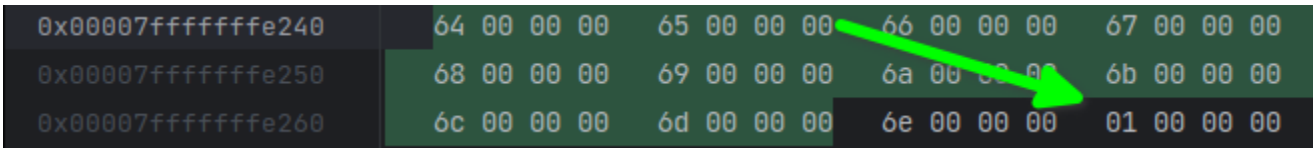
This is easily verified in our memory viewer by selecting the address of j which gets highlighted below:



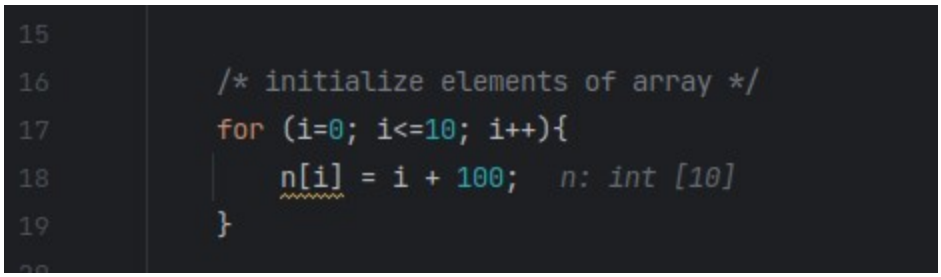
Next we verify the address of our single 4-byte address space for our incremental integer i:



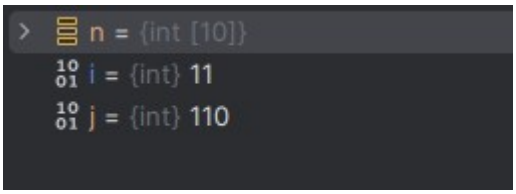
Which we can see is also sequentially assigned next to j and our integer buffer



Thus our problem begins with our over bounded loop:



Where the count is actually going beyond the 10 element array since counting zero + 10 is 11 elements we begin to write into the next address space or overflow into j as seen here:



As seen above, j becomes the next element in the array unintentionally. Thus it's fair to say any calls beyond the array n[9] are actually calling the same memory address as int j and anything further eventually would call int i.

Since j and n[10] are the same memory space we get a bug where our later for loop whose incrementer is j begins to change values of n[10].

```
27      /* initialize elements of array */
28      for(j=0; j <= 10; j++){
29          printf( format: "Element[%d] = %d\n", j, n[j]);
30
31      }
32
```

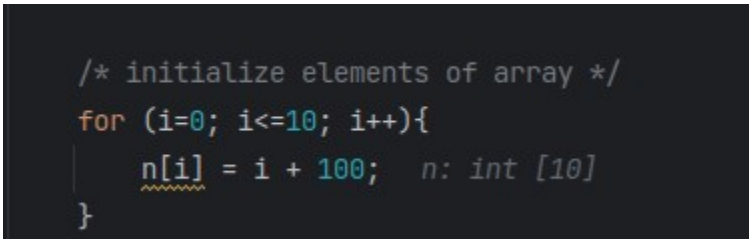
Here during the loop we display n[10] as the value being looped through by j which is 10 at that point in the loop state. Eventually, j increments to the total value of 11 (again 0 + 10) is 11 counted elements – thus once it finally exits the for loop the address space at j and n[10] is set to 11 which is why calling it after as seen here:

```
printf( format: "Element[%d] = %d\n", j, n[10]);
```

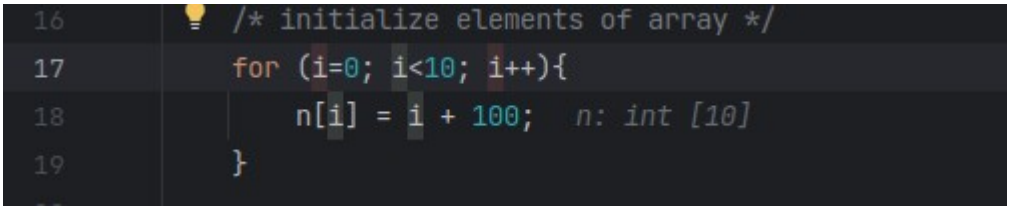
Is displaying out loops final count instead of its looping iteration value.

Solution

The solution for this particular issue is kind of moot as it was a demonstration of bounds overflowing but the main issue is actually common and that is having an incorrect loop stop value that exceeds the buffer being looped through. It's important to remember to stop your loop at the appropriate point and not equal or exceed to a point beyond its declaration. So in this case changing :



To this:



Would fix the loop so it wont overwrite into j. The same for the display loop. Calling the array directly beyond its bounds would likely be intentional or a typo and the only fix would be not doing it.