

這一次的bonus其實相當的有趣(x

我這次嘗試完全不上網查的情況下找出原因

首先我先把code原封不動放到gcc 跑一遍

過程不斷的出現了warning 叫我用%hd 去讀取 int16\_t

在我把它改成%hd 之後, 真的就讀進去了

不過題目有說不是要改正方法, 而是要我們探討原因。

我們可以從兩者code的差別開始去尋找問題所在

為甚麼當 i 從 0 ~ 4 的時候就不會有問題, 但當 i 是從 4 回到 0 的時候就有問題呢?

我這次嘗試讓他邊輸入就先邊印, 我發現原本 Bob 的程式碼是有把 1,2,3,4,5 讀進去的, 但是最後卻只有保留了 5

這說明了甚麼? **代表在輸入後面的數字的時候, 前面的記憶體位置的值被覆蓋了**

接下來我們做一個實驗。

邊輸入, 邊輸出所有的陣列元素

```
andy@lu4146@ub20:~/CFile$ ./a.out
1
0 0 0 0 1
2
0 0 0 2 0
3
0 0 3 0 0
4
0 4 0 0 0
5
5 0 0 0 0
5 0 0 0 0
andy@lu4146@ub20:~/CFile$
```

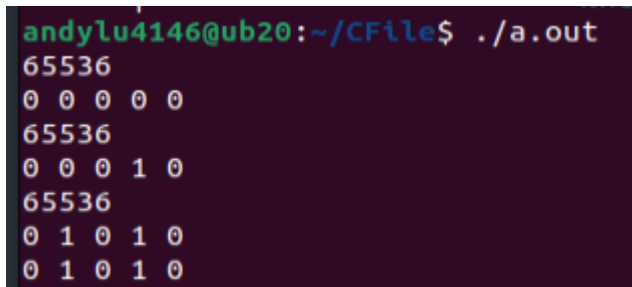
我們發現了一個問題, 正如我們前面思考的, 當輸入新的值的時候, 前面的就被覆蓋掉了。那如果我們改成一次輸入間隔兩個idx 會怎麼樣呢?

```
andy@lu4146@ub20:~/CFile$ ./a.out
1
0 0 0 0 1
2
0 0 2 0 1
3
3 0 2 0 1
3 0 2 0 1
```

看到這邊基本上應該可以知道原因了(?)

**原因就是我們目前scanf 是讀取了32bit, 但我們陣列的每個index的size是16bit, 所以當你讀取第 i 個的時候, 第 i + 1個會被你覆蓋。**

為了確保我想的是正確的，我們再做一個實驗。  
我們輸入的大小超過 `int16_t` 試試看



```
andy@lu4146@ub20:~/CFile$ ./a.out
65536
0 0 0 0 0
65536
0 0 0 1 0
65536
0 1 0 1 0
0 1 0 1 0
```

可以看到，65536 因為是  $2^{16}$  次方，所以 `int16_t` 存不下了。印證了我們前面說的，會把超過 `int16_t` 大小的移到 `i + 1` 的位置。

因此，就變成 `idx 1` 和 `3` 有數字啦（是 `1` 是因為  $2^{16} = 65536$ ，所以第17個位元的位置會是1，而第十七個位元跑到 `index i + 1` 去了）

接下來我看到的下面老師寫的如果把陣列長度改成5會怎麼樣呢，跑出了一個 `stack smashing detected`。但其實原因已經很清楚了，我們前面有提到當你讀取第 `i` 個的時候，第 `i + 1` 個也會被動到。所以假如 `i = 4` 的時候，他會去動到 第五個 `index` 的記憶體位置，但我們陣列只有開 5 個大小，所以沒有 `index = 5` 的這個位置，因此觸發了 `stack smashing detected`（換句話說就是 `index out of range` 啦）。

印證的話只要把陣列大小開成 6，跑一次就會發現沒有這個問題了，因為陣列大小是 6 的時候就會有 `index = 5` 啦~