XBS

Chạy thử chương trình

```
┌──(kali㉿ DESKTOP-6TCCJEL)-[/mnt/c/Users/BILL/Desktop/Release_3/XBS]
└─$ ./a.out
fsafwfwefwf
Try again!
```

Phân tích với IDA

```
15  v3 = 0;
16  v15 = __readfsqword(0x28u);
17  do
18  {
19    __isoc99_scanf("%d", &v14);
20    v4 = v14;
21    v5 = 30LL;
22    v6 = 0;
23    v7 = 0;
24    while ( ((v4 >> v5) & 1) == 0 )
25    {
26LABEL_9:
27      if ( v5-- == 0 )
28      {
29        if ( v6 )
30          v14 = v4;
31        goto LABEL_12;
32      }
33    }
34    v8 = a[v5];
35    if ( v8 )
36    {
37      ++v7;
38      v4 ^= v8;
39      v6 = 1;
40      goto LABEL_9;
41    }
42    if ( v6 )
43      v14 = v4;
44    a[(int)v5] = v4;
45LABEL_12:
46    if ( v7 <= 1 && v3 > 1 )
47    {
```

```
LABEL_12:
    if ( v7 <= 1 && v3 > 1 )
    {
LABEL_18:
      v12 = "Try again!";
      goto LABEL_19;
    }
    ++v3;
  }
  while ( v3 != 5 );
  v10 = a;
  v11 = 0;
  do
    v11 += *v10++;
  while ( v10 != &a[31] );
  v12 = "Congrats!";
  if ( v11 != 1073840184 )
    goto LABEL_18;
LABEL_19:
  puts(v12);
  return 0;
}
```

Chúng ta cần in ra chuỗi “Congrats!”. Điều đó có nghĩa tổng tất cả các phần tử trong mảng a = 1073840184. Mảng a ban đầu toàn bộ bằng 0.

Đến với vòng lặp do while, vòng này lặp lại tối đa 5 lần, mỗi lần lấy 1 số nhập vào để xử lí.

Với số d nhập vào, đầu tiên chương trình gán v4 = d, tìm ra vị trí i của bit 1 đầu tiên từ trái sang của v4. Sau đó kiểm tra:

Với a[i] == 0 (a chưa được gán, => gán a[i] = v4, kiểm tra điều kiện)

Với a[i] != 0 (a đã được gán, tăng v7 lên 1, v4 ^= a[i], trở lại vòng lặp và tìm bit 1 tiếp theo của v4 (sau khi xor)).

Yêu cầu: Tổng bằng 1073840184

Từ số thứ 3 trở đi, v7 >= 2, tức điều kiện if (v8) xảy ra ít nhất 2 lần

Nhận xét: Với tính chất như vậy, mỗi lần lặp chúng ta chỉ có thể gán 1 vị trí tại a. Và các số phía sau cần có ít nhất 2 bit 1 trùng vị trí với vị trí đã gán lên mảng a trước đó. Giá trị số phía sau cần được tính bằng các phép xor với số trước đó.

Mình đã tính 5 số cần thiết như sau

```
>>> bin(1073840184)
'0b1000000000000011000000000111000'
>>> a = 0b10000000000000000000000000000000
>>> b = 0b10000000000000000
>>> c = a ^ b ^ 0b1000000000000000
>>> d = a ^ b ^ 0b100000
>>> e = a ^ b ^ 0b11000
>>> a
1073741824
>>> b
65536
>>> c
1073840128
>>> d
1073807392
>>> e
1073807384
```

Vì số bit của a > b > c > d > e, nên phép xor sẽ đảm bảo bit 1 đầu tiên của c, d, e trùng với bit 1 đầu tiên của a, sau khi xor với a thì bit 1 đầu tiên của c, d, e sẽ trùng với bit 1 đầu tiên của b, và sau khi xor lần thứ 2 thì giá trị c, d, e trở về giá trị cần gán và gán vào các vị trí khác nhau trên mảng a (do vị trí bit 1 lớn nhất khác nhau) tổng của a + b + c + d + e = 1073840184

```
┌──(kali㉿ DESKTOP-6TCCJEL)-[/mnt/c/Users/BILL/Desktop/Release_3/XBS]
└─$ ./a.out
1073741824
65536
1073840128
1073807392
1073807384
Congrats!
```