3kCTF_2021

3k_chall_Microscopic Revenge

`Challenge's Description`

```
The 2021 version is here! YAY \o/
NOTE: flag format is flag{}
```

`Link source:` `Microscopic Revenge`

TL;DR

- Các ký tự trong input flag lần lượt sẽ được xử lý bằng hàm **sub_4012D0()**
- mình không hiểu tại sao nhưng debug mình thấy được **sub_4012D0()** sẽ gọi đến code tại **.text:00182622** để thực hiện, chính nhờ đó mình cũng suy ra được `len flag bằng` `0x28`

> - Hàm **sub_4012D0()** sẽ chứa *switch case*, khi debug chúng ta sẽ thấy được có *case 2, case 3* dùng để xuất lần lượt thông báo đến màn hình(case 2) và case 3 sẽ get flag input chúng ta nhập vào.
> - *Cũng có thêm **case 5,** để gấp đôi từng input[i] rồi **& 0xff***
> - *Sau khi xử lý ở case 5 cho từng input xong, sẽ đến **case 7,** một con trỏ hàm trỏ đến hàm tại **sub_402550**, tham số sẽ chứa input của chúng ta và một vài tham số rác khác nữa*

- Hàm **sub_402550()** đóng vai trò như một `hàm xử lý` từng phần tử input, nó khá **rối** trong cả *disassembly* cũng như *decompile* code của IDA. Mình không hiểu được khi static analysis
- Chỉ debug kỹ lại hàm cũng như xem cách các giá trị tham số thay đổi, mình mới hiểu được nó sẽ `tính toán` và có 2 giá trị trả về(giá trị sau hầu như bằng 0) được lưu tại **.data:00406658 unk_406658**
- Như vậy ta đã có một mảng mới là *unk_406658* sau khi xử lý tất cả các ký tự(mảng gồm 40 ký tự, bỏ thành phần trả về thứ 2 đi, vì nó luôn bằng 0)

- **[Quan trọng]** - Chương trình so sánh từng giá trị trong mảng mới tạo từ input với mảng được gửi sẵn tại **.data:00406028** Nếu thỏa tất cả thì pass challenge

---

Solve script

Cơ bản để mình có thể hiểu được solve script là phải hiểu được hàm **sub_402550(),** sau sau đó reimplement lại nó rồi brute force xem ký tự nào thỏa mãn với mảng được cho sẵn, như vậy là xong

solve_microscopic_revenge.py

```
target = [161370727, 487875157, 348326941, 390132824, 354474512, 60097899,

divide_number = 0x1DB038C5
def findChar(i):
  cnt = 0
  possible = []
  for xchar in range(33, 126):
    esi = 5
    res = 0
    ecx = 0
    flag_i = (xchar * 2) & 0xff
    #====================[[ stage init    ]]
    eax = flag_i % divide_number
    while esi != 0:
      #====================[[ stage esi = 5 ]]
      if esi == 5:
        eax = 1 * flag_i
        eax = flag_i % divide_number

        ecx = flag_i *flag_i
        eax = ecx

        eax = eax % divide_number

      #====================[[ stage esi = 2 ]]
      elif esi == 2:
        ecx = ecx**2        #(flag_i**4)
        eax  = ecx & 0xffffffff
        eax = eax % divide_number
      elif esi == 1:
        ecx = flag_i * ecx
        eax = ecx
```

```
        eax = eax % divide_number
        res = eax
      esi = esi >> 1
    #end while loop
    if res == target[i]:
      cnt +=1
      possible.append(chr(xchar))
  print('so ky tu phu hop: ', cnt)
  print(possible)
  return possible[0]

def main():
  flag = ''
  for i in range(0, 40):
    char = findChar(i)
    flag += char
  print(flag)

if __name__ == '__main__':
    main()

```