**Reverse 9 класс — 2**

**Описание**

Дан бинарь redstone-lock , он выводит флаги если пройти проверки

**Структура:**

Пользователь вводит строку:

Проверки, которые нужно пройти чтобы получить первый флаг:

input[0] = 'a'

input[1] - четная цифра

ord(input[2])^3 = 132651 ( input[2]=='3' )

input[3] = '7'

Проверки, которые нужно пройти чтобы получить второй флаг:

ord(input[4])^3 - 132651(51^3) = 24813

сумма цифр с позиции 5 = 100

**Задание:**

1. **ФЛАГ 1:** Найти XOR ключ в памяти программы

( vsosh{0k4y\_h3r3\_1$\_7H3\_f1r$7\_f14g\_d1d\_y0u\_g3t\_3n0ugh} ) 2. **ФЛАГ 2:** Расшифровать файл output.txt с помощью найденного ключа( vsosh{4444444gh\_0n3\_m0r3\_u\_g37\_m3} )

**Решение**

Запустим файл:

Enter the code: aaaaaa

:(

Файл требует какой-то код.

Откроем бинарь в ghidra, посмотрим в main:



Видим fgets и то, что дальше input подвергается некоторым проверкам, успешное прохождение которых приводит нас к выполнению print\_obfuscated\_flag1 и print\_obfuscated\_flag2 . Посмотрев внутрь этих функций мы видим что там сложное обфусцированное вычисление флага + проверки ret адреса, что мешает нам запустить функцию просто из дебагера поменяв rip.

Последовательно пройдемся по всем проверкам. Для начала переименуем очевидные переменные для удоства анализа.

local\_138 в input .

local\_10 в len\_input .

local\_24 в ord\_input\_2

local\_28 в cube\_ord\_input\_2

Сразу видим, что:

убирается \n у ввода

ввод должен быть >= 5 и меньше чем 0xc9(201)

**первый символ должен быть** a

второй символ должен быть любой четной цифрой

ascii номер третьего символа должен быть кубическим корнем числа 0x2062b(132651)

перебираем цифры и получается что это 51 или цифра 3

четвертый символ должен быть 7

Видим, что прохождения этих проверок достаточно для получения флага, но так как у нас всего 4 символа, а выше есть условие на >= 5, дополним несколькими символами чтобы получить первый флаг.

$ echo "a237aa" | ./redstone-lock

Enter the code: FLAG 1:

vsosh{0k4y\_h3r3\_1$\_7H3\_f1r$7\_f14g\_d1d\_y0u\_g3t\_3n0ugh}

:(

Теперь посмотрим на дальнейшие проверки чтобы получить флаг,

local\_134 может ввести в заблуждение, но это вспомнив что input до переименования назывался local\_138 несложно догадаться что это пятый элемент нашего ввода:

base\_address = rbp - 0x138 # Начало массива input

element\_5 = base\_address + 4 # input[4]

element\_5 = (rbp - 0x138) + 4

element\_5 = rbp - 0x134 # local\_134

Так же для удобства переименуем:

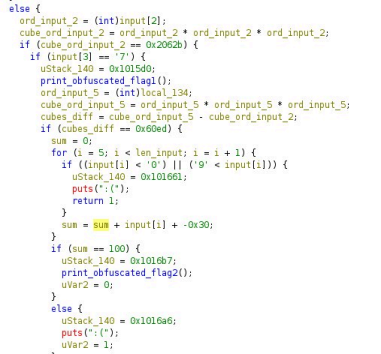
local\_2c -> ord\_input\_5

local\_30 -> cube\_ord\_input\_2

local\_34 -> cube\_diff

local\_20 -> i

local\_14 -> sum

Теперь у нас есть вводные для того, чтобы получить второй флаг:

разница кубов четвертого элемента и пятого(ascii представлений) должна равняться 0x60ed(24813), несложно понять что правильный символ это 6 (54) сумма всех последующих элементов должна равняться ста

Дополняем прошлый ввод шестеркой и цифрами, которые в сумме дают 100 и получаем второй флаг:

$ echo "a2376999999999991" | ./redstone-lock

Enter the code: FLAG 1:

vsosh{0k4y\_h3r3\_1$\_7H3\_f1r$7\_f14g\_d1d\_y0u\_g3t\_3n0ugh}

FLAG 2: vsosh{4444444gh\_0n3\_m0r3\_u\_g37\_m3}