

Летняя научно-исследовательская практика Injection (Zh3r0 CTF V2)

Воробьёв Денис

10.05.01 "Компьютерная безопасность" 3 курс

8 июля 2023 г.

Оглавление

- 1 Постановка задачи
- 2 Математическое вступление
- 3 Программирование
- 4 Доказательство выполнения работы

Постановка задачи

Условие

Файл `chall.py` содержит только одну функцию и только один зашифрованный флаг

```
def nk2n(nk):  
    l = len(nk)  
    if l == 1:  
        return nk[0]  
    elif l==2:  
        i,j = nk  
        return ((i+j)*(i+j+1))//2 +j  
    return nk2n([nk2n(nk[:l-l//2]),  
nk2n(nk[l-l//2:])])
```

Анализ кода

Приведённая выше функция шифрует флаг, до которого мы хотим добраться.

На вход функцией принимается байтовая последовательность (символы ASCII), на выходе натуральное число. Функция представляет собой рекурсивный алгоритм и возвращает на последнем слое рекурсии:

$$r = \frac{((i+j)*(i+j+1))}{2} + j. \quad (1)$$

где i и j есть результаты вызова функции для левой и правой частей входной строки соответственно.

Для захвата флага нам необходимо научиться обращать эту функцию, и мы обязательно сделаем это после элементарного математического вступления.

Математическое вступление

Основные формулы

Из (1) сразу следует:

$$2r - 2j = (i + j) * (i + j + 1). \quad (2)$$

Кроме того, имеем:

$$(i + j) * (i + j + 1) \approx (i + j)^2. \quad (3)$$

$$\sqrt{2r - 2j} \approx (i + j). \quad (4)$$

Алгоритм (I)

Если внимательно присмотреться к данной в `chall.py` функции, становится ясно, что до финального слоя рекурсии доходит последовательность из двух чисел. С этого мы и начнём дешифрование.

Положим $enc := r$.

Заметим $\sqrt{2enc} = \sqrt{(i+j) * (i+j+1) + 2j}$.

Сейчас мы будем аппроксимировать $i+j$ с помощью $\sqrt{2enc}$ следующим способом.

Из формулы (2): $j = \frac{2r - (i+j)^2 - (i+j)}{2} = \frac{2enc - (i+j)^2 - (i+j)}{2}$.

Перейдём, положив $e = \lfloor \sqrt{2enc} \rfloor$, к $j_k = \frac{2enc - (e-k) * ((e-k)+1)}{2}$.

Начиная с некоторого k выполняется: $j_k > 0$.

Алгоритм (II)

Теперь положим $e' = \sqrt{2 * enc - 2 * j_k}$.

Если $e' = e_k$, то из формулы (4) имеем

$e' = \sqrt{2 * enc - 2 * j_k} = e_k \approx (i + j)$, полагая теперь $i = e_k - j_k$ проверяем формулу (1).

Если (1) не выполняется, возвращаемся в начало цикла, инкрементируя $k = k + 1$.

Программирование

Основной цикл

```
e, nul = gmpy2.iroot(2 * enc, 2)
e = int(e)
while True:
    j = (2 * enc - (e * (e + 1))) // 2
    if j < 0:
        e = e - 1
        continue
    e1, nul = gmpy2.iroot(2 * enc - 2 * j, 2)
    if e == e1:
        i = e - j
        assert ((i + j) * (i + j + 1)) // 2 + j == enc
        break
    e = e - 1
```


Пара слов о программе

Во-первых, нетрудно заметить, что "существующего в теории" счётчика k в основном цикле нет. Дело в том, что на практике он оказался просто не нужен. Вычитать из e единицу на каждом шаге можно и без него, а хранить последовательность j_k или даже сам счётчик k – бесполезно.

Во-вторых, появилась странная переменная `nil` в двух местах. Спешу успокоить читателя: она нужна лишь потому, что функция `iroot()` возвращает два значения – собственно корень нужной степени и значение типа `bool`, истинность которого говорит о существовании вещественного корня.

В-третьих, в презентации не представлен полный код программы. На это мы пошли умышленно: для лаконичности, связности повествования, а ещё потому, что остальной код является сугубо служебным и не представляет большого интереса с точки зрения криптографии.

Доказательство выполнения работы

 C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64
zh3r0{wh0_th0ugh7_b1j3c710n5_fr0m_n^k_t0_n_c0uld_b3_
Press any key to continue . . .



1n_jection (Zh3r0 CTF V2)

"COVID: *exists* vaccine jokes: *challenge_name*"

Challenge contributed by **deuterium**

Challenge files:

- **challenge.py**

You have solved this challenge!

Код программы и этот документ находятся в GitHub:
--> нажмите на меня <--

Спасибо за внимание!