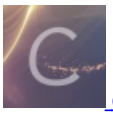
攻防世界Reverse进阶区-parallel-comparator-200-writeup

原创

y4ung 于 2020-09-21 21:50:04 发布 287 收藏

分类专栏： ctf 文章标签： ctf



1. 介绍

本题是xctf攻防世界中Reverse的进阶区的题parallel-comparator-200
题目来源： school-ctf-winter-2015

2. 分析

文件下载下来是一个c文件，直接打开查看源代码。

main函数中读取用户输入保存到变量user_string，然后传入highly_optimized_parallel_comparsion函数中判断，如果函数返回值is_ok为1，则打印出 `You win!` 。

```
int main()
{
    char *user_string = (char *)calloc(FLAG_LEN+1, sizeof(char));
    fgets(user_string, FLAG_LEN+1, stdin);
    int is_ok = highly_optimized_parallel_comparsion(user_string);
    if (is_ok)
        printf("You win!\n");
    else
        printf("Wrong!\n");
    return 0;
}
```

highly_optimized_parallel_comparsion函数中，在最后一个for循环里通过判断generated_string是否与just_a_string相等，若相等，则返回1。ok，看看generated_string怎么生成的。
通过initialization_number得到first_letter，范围在[a~z]。然后在第一个for循环中创建线程，函数是checking，参数为arguments。
在for循环中先对arguments这个字符串数组的每个字符串赋值。
在checking函数中，result返回值等于 `(argument[0]+argument[1]) ^ argument[2]`

```
arguments[i][0] = first_letter;
arguments[i][1] = differences[i];
arguments[i][2] = user_string[i];
```

第二个for循环将每个线程的返回值与just_a_string[i]相加（ASCII码相加）： `generated_string[i] = *(char *)result + just_a_string[i];` ，最终得到generated_string。

```
int highly_optimized_parallel_comparsion(char *user_string)
{
    int initialization_number;
    int i;
    char generated_string[FLAG_LEN + 1];
    generated_string[FLAG_LEN] = '\0';
    while ((initialization_number = random()) >= 64);  // initialization_number必须小于64

    int first_letter;  // 第一个字母 = (随机初始化的数字 % 26) + 97  [a~z]
    first_letter = (initialization_number % 26) + 97;  // 11
    printf("first_letter is: %d\n", first_letter);  // 108
    pthread_t thread[FLAG_LEN];
    char differences[FLAG_LEN] = {0, 9, -9, -1, 13, -13, -4, -11, -9, -1, -7, 6, -13, 13, 3, 9, -13, -11, 6, -7}
;
    char *arguments[20];  // 二维数组
    for (i = 0; i < FLAG_LEN; i++) {
        arguments[i] = (char *)malloc(3*sizeof(char));
        arguments[i][0] = first_letter;
        arguments[i][1] = differences[i];
        arguments[i][2] = user_string[i];
        pthread_create((pthread_t*)(thread+i), NULL, checking, arguments[i]);  // 创建线程
    }
    void *result;
    int just_a_string[FLAG_LEN] = {115, 116, 114, 97, 110, 103, 101, 95, 115, 116, 114, 105, 110, 103, 95, 105,
116, 95, 105, 115};  // "strange_string_it_is"
    for (i = 0; i < FLAG_LEN; i++) {
        pthread_join(*(thread+i), &result);  // 用来等待一个线程的结束，result为线程的返回值
        generated_string[i] = *(char *)result + just_a_string[i];  // generated_string 赋值
        free(result);
        free(arguments[i]);
    }
    int is_ok = 1;  // 这个变量未被使用
    for (i = 0; i < FLAG_LEN; i++) {
        if (generated_string[i] != just_a_string[i])  // *(char *)result + just_a_string[i] == just_a_string[i]
  =>  *(char *)result == 0
            return 0;
    }
    return 1;
}
```

等等，第三个for循环是判断generated_string与just_a_string的相等关系，也就是说每个线程的返回值result都必须为0，即(argument[0]+argument[1]) ^ argument[2]必须为0，即 `(first_letter + differences[i]) ^ user_string[i] == 0 => user_string[i] = 0 ^ (first_letter + differences[i])` 。那么现在的问题是first_letter等于多少。 但initialization_number是在用户输入以后才计算的。。

不过既然已经有了源代码，那直接打印出来看一下？发现每次运行，first_letter的值都为108，也就是rand()的值都是11。

写脚本算一下，得到flag: `lucky_hacker_you_are`

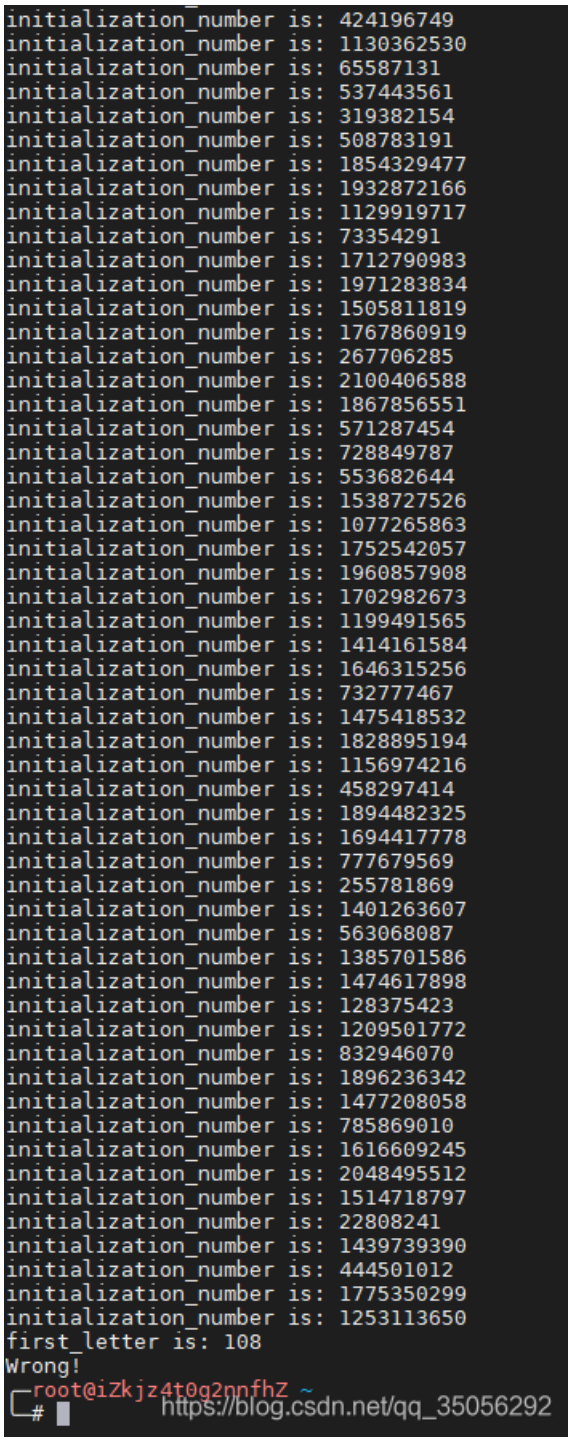
```python
res = ""
differences = [0, 9, -9, -1, 13, -13, -4, -11, -9, -1, -7, 6, -13, 13, 3, 9, -13, -11, 6, -7]
just_a_string = [115, 116, 114, 97, 110, 103, 101, 95, 115, 116, 114, 105, 110, 103, 95, 105, 116, 95, 105, 115]

for i, each in enumerate(just_a_string):
    # (argument[0]+argument[1]) ^ argument[2] 必须为0
    curr_val = 0 ^ (108 + differences[i])
    res += chr(curr_val)

print("flag: ", res)
```

为什么那个while循环中最后退出循环时，rand()的值都是11呢。。
这与srand()函数有关。在调用rand()函数产生随机数前，必须先利用srand()设好随机数种子（seed）, 如果未设随机数种子, rand()在调用时会自动设随机数种子为1。

```
initialization_number is: 182579937
initialization_number is: 424196749
initialization_number is: 1130362530
initialization_number is: 65587131
initialization_number is: 537443561
initialization_number is: 319382154
initialization_number is: 508783191
initialization_number is: 1854329477
initialization_number is: 1932872166
initialization_number is: 1129919717
initialization_number is: 73354291
initialization_number is: 1712790983
initialization_number is: 1971283834
initialization_number is: 1505811819
initialization_number is: 1767860919
initialization_number is: 267706285
initialization_number is: 2100406588
initialization_number is: 1867856551
initialization_number is: 571287454
initialization_number is: 728849787
initialization_number is: 553682644
initialization_number is: 1538727526
initialization_number is: 1077265863
initialization_number is: 1752542057
initialization_number is: 1960857908
initialization_number is: 1702982673
initialization_number is: 1199491565
initialization_number is: 1414161584
initialization_number is: 1646315256
initialization_number is: 732777467
initialization_number is: 1475418532
initialization_number is: 1828895194
initialization_number is: 1156974216
initialization_number is: 458297414
initialization_number is: 1894482325
initialization_number is: 1694417778
initialization_number is: 777679569
initialization_number is: 255781869
initialization_number is: 1401263607
initialization_number is: 563068087
initialization_number is: 1385701586
initialization_number is: 1474617898
initialization_number is: 128375423
initialization_number is: 1209501772
initialization_number is: 832946070
initialization_number is: 1896236342
initialization_number is: 1477208058
initialization_number is: 785869010
initialization_number is: 1616609245
initialization_number is: 2048495512
initialization_number is: 1514718797
initialization_number is: 22808241
initialization_number is: 1439739390
initialization_number is: 444501012
initialization_number is: 1775350299
initialization_number is: 1253113650
first_letter is: 108
Wrong!
root@iZkjz4t0g2nnfhZ ~
# 
```



```
initialization_number is: 182579937
```



可以看到跑了两次的结果一模一样，就是因为种子未设置，采用了默认值1的缘故