攻防世界reverse进阶parallel-comparator-200 writeup（#伪随机、#异或变换）

原创

漫小牛 于 2021-05-09 09:51:39 发布 130 收藏 1

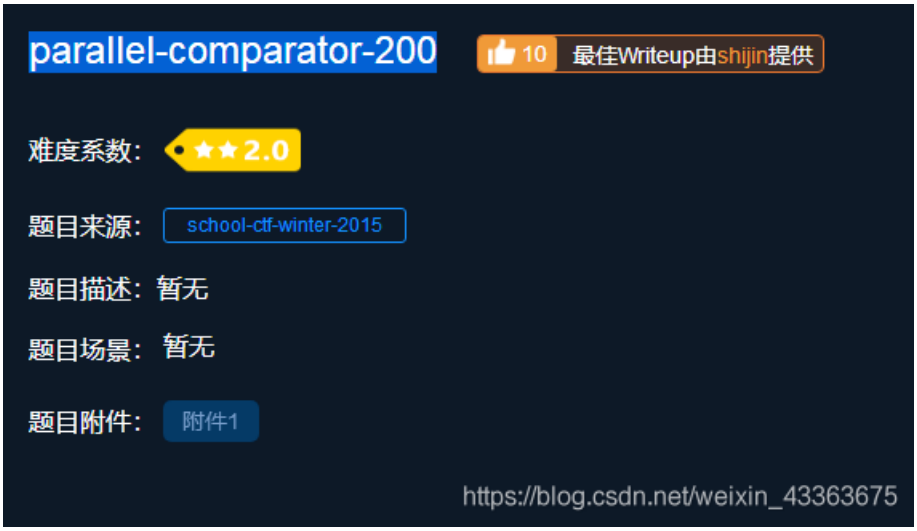
分类专栏： CTF Writeup 文章标签： 经验分享

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/weixin_43363675/article/details/116560581
版权

CTF Writeup 专栏收录该内容
5 篇文章 0 订阅
订阅专栏

文章目录

第一步 概貌分析



这是攻防世界reverse中的一道进阶题。下载附件1后，可看到这是一个.c的文件，用dev c++打开该文件，可查看到文件的源代码。

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

#define FLAG_LEN 20

void * checking(void *arg) {
    char *result = malloc(sizeof(char));
    char *argument = (char *)arg;
    *result = (argument[0]+argument[1]) ^ argument[2];
    return result;
}

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);

    int first_letter;
    first_letter = (initialization_number % 26) + 97;  //第一个字母是一个随机的小写字母?

    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];
    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]); //起20个线程
    }

    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};
    for (i = 0; i < FLAG_LEN; i++) {
        pthread_join(*(thread+i), &result);
        generated_string[i] = *(char *)result + just_a_string[i];
        free(result);
        free(arguments[i]);
    }

    int is_ok = 1;
    for (i = 0; i < FLAG_LEN; i++) {
        if (generated_string[i] != just_a_string[i])
            return 0;
    }

    return 1;
}

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;
}
```

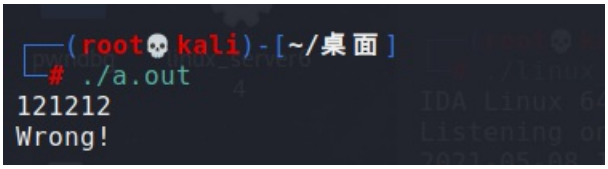
pthread是在linux平台下广泛使用的线程库，因此，应在linux平台下编译执行该文件。

第二步 程序的编译和执行

编译命令为：

```
gcc 2e554730887e4ebca5a8e9d04658b2c8.c -pthread -g
```

注意，需添加-g选项，以便使用gdb进行调试；同时，由于程序中使用了多线程库，需添加-pthread编译选项，否则程序会报错。
在命令行中执行该文件，随便输入一些字符后，提示错误“Wrong”。



第三步 静态人工分析

1、主函数分析

主函数main在L60调用了highly_optimized_parallel_comparsion函数，从函数名可知，该函数进行了并行优化，该函数返回值is_ok为1时，即找到了flag。

2、highly_optimized_parallel_comparsion分析

进入highly_optimized_parallel_comparsion函数分析其逻辑：

- L21产生一个随机数，并根据该随机数在L24产生一个随机的小写字母。
- L29-L35的循环创建了20个线程，线程返回的结构在L41通过pthread_join函数返回到变量result中。

pthread_join()函数，以阻塞的方式等待thread指定的线程结束。当函数返回时，被等待线程的资源被收回。如果线程已经结束，那么该函数会立即返回。并且thread指定的线程必须是joinable的。

- L42用于计算generated_string数组：

```
generated_string[i] = *(char *)result + just_a_string[i];
```

- L49对两数组generated_string和just_a_string逐字节进行比较，若不相等，则返回false，若为真，则返回True。

3、线程函数体分析

进入到线程函数checking进行分析，关键代码为

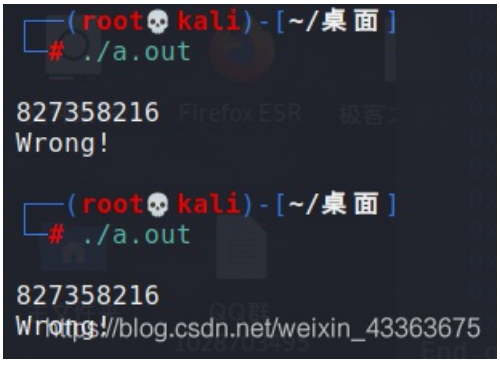
```
*result = (argument[0]+argument[1]) ^ argument[2];
```

4、综合分析

- L49可知generated_string和just_a_string必须相等，即：
  generated_string[i] = just_a_string[i]。
- 同时，依据L42的赋值语句generated_string[i] = *(char *)*result* +*just_a_string[i]可知*(char *)result为0，即：
  (argument[0]+argument[1]) ^ argument[2] = 0。两边同时异或argument[2]，得到等式：
  argument[0]+argument[1]=argument[2]。
- 在这里，argument[1]为differences数组，argument[2]为用户的输入字符串，关键是argument[0]是一个随机产生的小写字母，经不同的小写字母会计算出不同的输入字符串，这显然是不合常理的。在这种情况下，可以进行爆破，跑出来26个结果，看看哪个像flag，也可从random伪随机数发生器分析。

第四步 动态执行验证

实际上，random虽然是一个随机函数，但属于伪随机，每次运行程序时，第一次调用该函数产生的结果是相同的，都是827358216。根据该值计算的first_letter为108。



第五步 编写exp

在dev c++中编写exp代码：

```c
#include <stdio.h>
char differences[20] = {0, 9, -9, -1, 13, -13, -4, -11, -9, -1, -7, 6, -13, 13, 3, 9, -13, -11, 6, -7};
main()
{
 int i=0;
 char flag[20] = {0};
 for(;i<20;i++)
 {
  flag[i] = differences[i]+108;
 }
 printf("flag is %s\n", flag);

}
```

编译并执行，得到flag：

```
flag is lucky_hacker_you_are

--------------------------------
Process exited after 0.2719 seconds with return value 0
请按任意键继续. . .
```