# GoogleCTF 2021 CPP write up

原创

AliceNCsyuk 于 2021-07-19 10:01:19 发布 273 收藏

分类专栏： CTF 文章标签： 网络安全 google

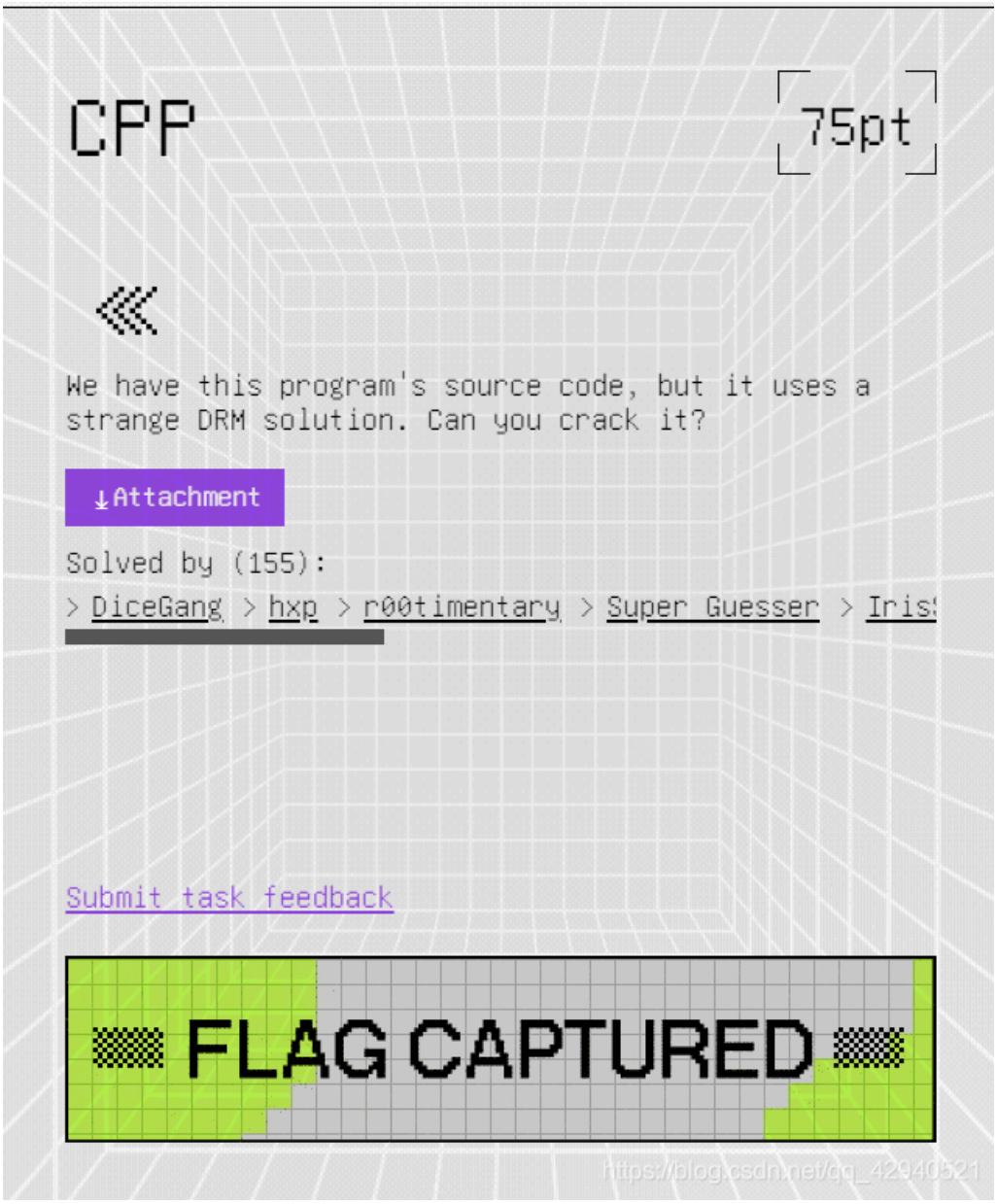
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/qq_42940521/article/details/118888781
版权

CTF 专栏收录该内容
4 篇文章 0 订阅
订阅专栏

## Google CTF 2021 Reversing CPP

googlectf
谷歌举行的CTF，今天上午8点结束
本场比赛过题人数第二多的题目，别问我为啥过题人数最多的Filestore我没解出来，问就是不会写交互脚本导致时间不够

题目描述如下：




感觉题目描述没啥意义，确实用不上
下载后只有一个cpp.c文件
发现里面是宏编程

```
// Copyright 2021 Google LLC
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     https://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
#if __INCLUDE_LEVEL__ == 0
// Please type the flag:
#define FLAG_0 CHAR_C
#define FLAG_1 CHAR_T
```

```
#define FLAG_1 CHAR_T
#define FLAG_2 CHAR_F
#define FLAG_3 CHAR_LBRACE
#define FLAG_4 CHAR_w
#define FLAG_5 CHAR_r
#define FLAG_6 CHAR_i
#define FLAG_7 CHAR_t
#define FLAG_8 CHAR_e
#define FLAG_9 CHAR_UNDERSCORE
#define FLAG_10 CHAR_f
#define FLAG_11 CHAR_l
#define FLAG_12 CHAR_a
#define FLAG_13 CHAR_g
#define FLAG_14 CHAR_UNDERSCORE
#define FLAG_15 CHAR_h
#define FLAG_16 CHAR_e
#define FLAG_17 CHAR_r
#define FLAG_18 CHAR_e
#define FLAG_19 CHAR_UNDERSCORE
#define FLAG_20 CHAR_p
#define FLAG_21 CHAR_l
#define FLAG_22 CHAR_e
#define FLAG_23 CHAR_a
#define FLAG_24 CHAR_s
#define FLAG_25 CHAR_e
#define FLAG_26 CHAR_RBRACE

// No need to change anything below this line
#define CHAR_a 97
#define CHAR_b 98
#define CHAR_c 99
#define CHAR_d 100
#define CHAR_e 101
#define CHAR_f 102
#define CHAR_g 103
#define CHAR_h 104
#define CHAR_i 105
#define CHAR_j 106
#define CHAR_k 107
#define CHAR_l 108
#define CHAR_m 109
#define CHAR_n 110
#define CHAR_o 111
#define CHAR_p 112
#define CHAR_q 113
#define CHAR_r 114
#define CHAR_s 115
#define CHAR_t 116
#define CHAR_u 117
#define CHAR_v 118
#define CHAR_w 119
#define CHAR_x 120
#define CHAR_y 121
#define CHAR_z 122
#define CHAR_A 65
#define CHAR_B 66
#define CHAR_C 67
#define CHAR_D 68
#define CHAR_E 69
#define CHAR_F 70
```

```
#define CHAR_G 71
#define CHAR_H 72
#define CHAR_I 73
#define CHAR_J 74
#define CHAR_K 75
#define CHAR_L 76
#define CHAR_M 77
#define CHAR_N 78
#define CHAR_O 79
#define CHAR_P 80
#define CHAR_Q 81
#define CHAR_R 82
#define CHAR_S 83
#define CHAR_T 84
#define CHAR_U 85
#define CHAR_V 86
#define CHAR_W 87
#define CHAR_X 88
#define CHAR_Y 89
#define CHAR_Z 90
#define CHAR_0 48
#define CHAR_1 49
#define CHAR_2 50
#define CHAR_3 51
#define CHAR_4 52
#define CHAR_5 53
#define CHAR_6 54
#define CHAR_7 55
#define CHAR_8 56
#define CHAR_9 57
#define CHAR_LBRACE 123
#define CHAR_RBRACE 125
#define CHAR_UNDERSCORE 95
#warning "Please wait a few seconds while your flag is validated."
#define S 0
#define ROM_00000000_0 1
```

类似这样的，定义了flag的值并使用宏进行运算，尝试运行发现出错，也就是flag正确才能出运行到main函数

```
6217  #if S != -1
6218  #include "cpp.c"
6219  #endif
6220  #if S != -1
6221  #include "cpp.c"
6222  #endif
6223  #endif
6224  #if __INCLUDE_LEVEL__ == 0
6225  #if S != -1
6226  #error "Failed to execute program"
6227  #endif
6228  #include <stdio.h>
6229  int main() {
6230      printf("Key valid. Enjoy your program!\n");
6231      printf("2+2 = %d\n", 2+2);
6232  }
6233  #endif
```

https://blog.csdn.net/qq_42940521

先用python处理好缩进

链接：https://pan.baidu.com/s/1dwSViPCw8AkbkfoGmy0a9A

提取码：c1p7

–来自百度网盘超级会员V5的分享

以下为可以在编译时打印宏的值的语句

```
#define PRINT_MACRO_HELPER(x) #x
#define PRINT_MACRO(x) #x"="PRINT_MACRO_HELPER(x)
#pragma message(PRINT_MACRO(__INCLUDE_LEVEL__))
#pragma message(PRINT_MACRO(S))
```

尝试使用fuzz test模糊测试判断程序控制流，但是并没有什么效果，按道理修改FLAG的值应该会造成程序控制流的改变，则编译打印内容应该会改变，但是并没有出现这种情况

通过fuzz判断只要能绕过#error "INVALID_FLAG"就是flag，推测为使用递归进行检测flag是否正确，并加入了防编译时间或者编译信息攻击的机制，也就是判断最后S是否为-1就结束整个flag的判断过程

flag长度为27，过检测块的次数为108，为27的4倍数，估计为一轮校验flag的一位

分支跳转语句只有8个，其他大量的S块都是顺序执行，可能这是突破口

分析宏编程过程，发现四个校验S块连续执行，完美

涉及flag相关的S的分支块是34，35，45，50

初步判断整体程序为递归+双层for循环，递归可以忽略不计，因为到达13层后直接进行flag全部判断，判断结束后直接退栈

发现I寄存器始终保持递增，经过分析得到I为外层for循环的i，这也是为什么之前fuzz时没有变化的原因

flag的取值点位在34，剩下几个都是辅助计算，不进行flag正确与否判别程序首先从S=34按flag编号为0-26的顺序，在每个循环中读取flag的值到A寄存器中，而在S块25中每次读入到B寄存器中的B的初始值是确定的

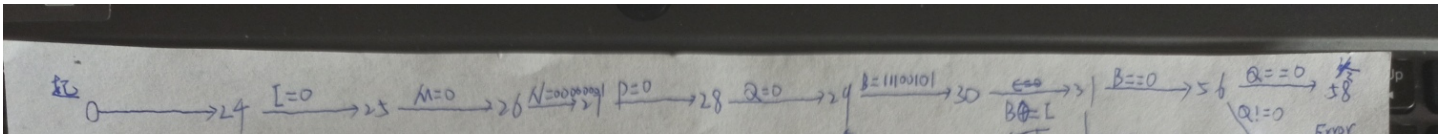
确定S块5的R值一定为0

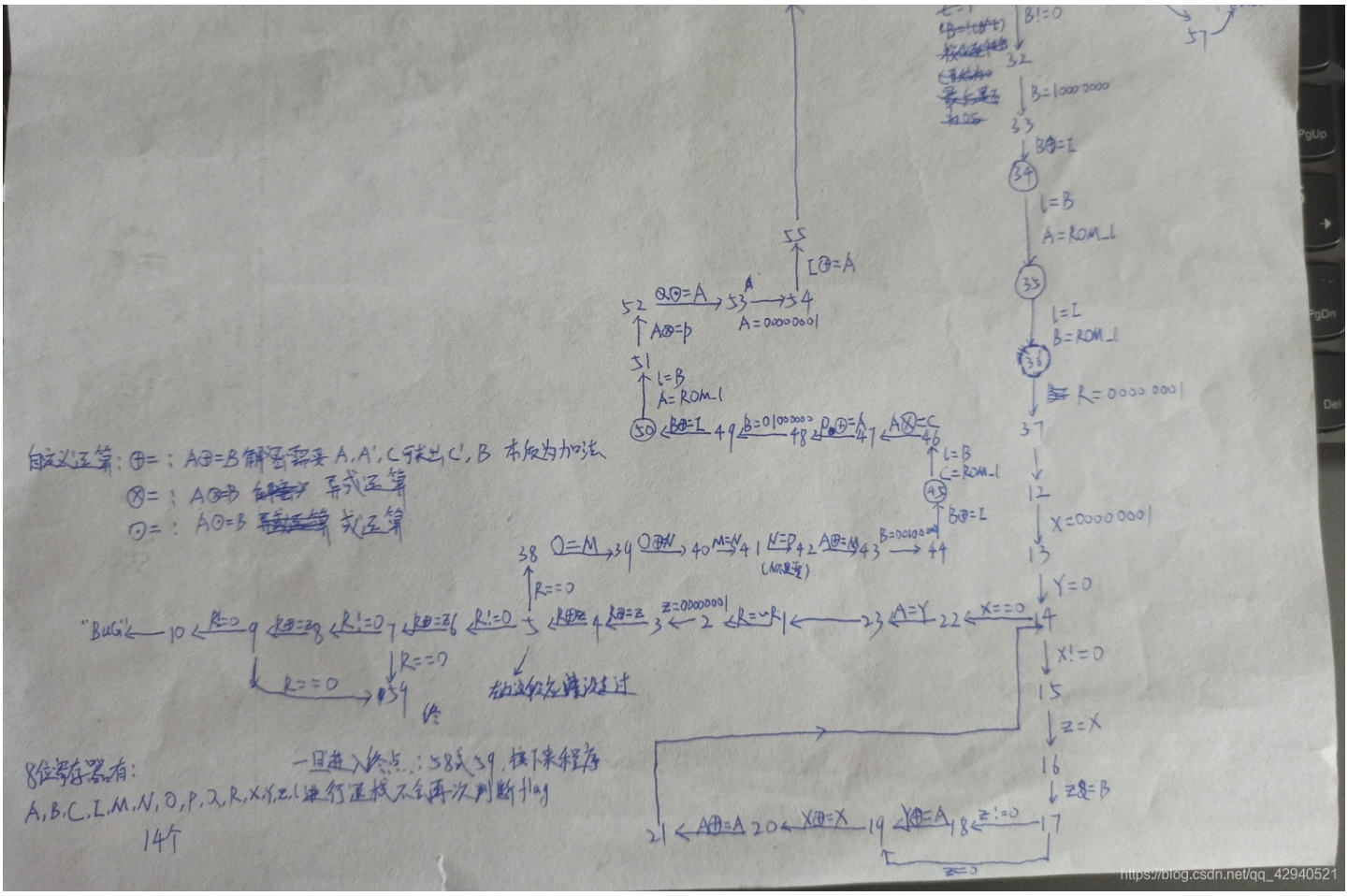
寄存器O,N,M中的值是斐波那契数列，是确定的

小循环的过程也是确定的

只要解析完最后两个特殊运算符号的含义，就可以写简化的逆向脚本了

手绘的程序流程图如下






圈叉是异或，圈加是无符号加法，圈点是或运算，解析完成，开始设计新的解密脚本

每一轮中，只有Q和flag中某一位的值不确定，最后一轮结束后Q的值为0，所以可以倒flag序爆破求解flag。根据程序，P最后的值可以直接算出，反向解密的时候根据程序我们需要把P减去一个临时的A寄存器的值，我们只需要在程序中提前存储好这个值，在下次解密时让P减去就可以了。递归编写解密脚本

链接：https://pan.baidu.com/s/1xR0j3WltX52_h5ZzLXtrDA

提取码：q5ue

–来自百度网盘超级会员V5的分享