REVERSE-PRACTICE-BUUCTF-13

原创

P1umH0 于 2021-02-26 23:10:13 发布 102 收藏

分类专栏： Reverse-BUUCTF

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_45582916/article/details/114155843

版权

Reverse-BUUCTF 专栏收录该内容

32 篇文章 3 订阅

订阅专栏

REVERSE-PRACTICE-BUUCTF-13

- firmware
- [ACTF新生赛2020]Oruga
- [Zer0pts2020]easy strcmp
- [GXYCTF2019]simple CPP

firmware

.bin（二进制）文件，由题目提示知是路由器固件逆向

参考：逆向路由器固件之解包 Part1

linux安装好binwalk和firmware-mod-kit

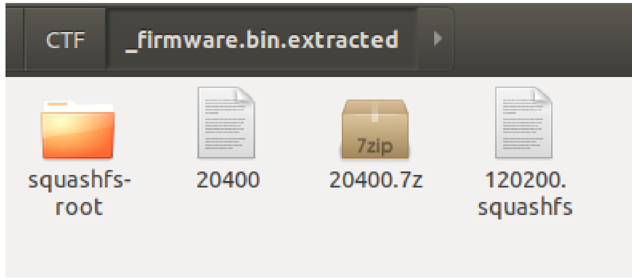
binwalk会分析二进制文件中可能的固件头或者文件系统，然后输出识别出的每个部分以及对应的偏移量

binwalk该二进制文件，发现包含了squashfs文件系统（squashfs是linux下的一种只读压缩文件系统类型）

```
zihaoli@zihaoliworld:~/CTF$ binwalk firmware.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             TP-Link firmware header, firmware version: 1.-20432.3, image version: "", product ID: 0x0, product version: 155254791, kernel load address: 0x0, kernel entry point: 0x80002000, kernel offset: 4063744, kernel length: 512, rootfs offset: 772784, rootfs length: 1048576, bootloader offset: 2883584, bootloader length: 0
69424         0x10F30         Certificate in DER format (x509 v3), header length: 4, sequence length: 64
94080         0x16F80         U-Boot version string, "U-Boot 1.1.4 (Aug 26 2013 - 09:07:51)"
94256         0x17030         CRC32 polynomial table, big endian
131584        0x20200         TP-Link firmware header, firmware version: 0.0.3, image version: "", product ID: 0x0, product version: 155254791, kernel load address: 0x0, kernel entry point: 0x80002000, kernel offset: 3932160, kernel length: 512, rootfs offset: 772784, rootfs length: 1048576, bootloader offset: 2883584, bootloader length: 0
132096        0x20400         LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 2203728 bytes
1180160       0x120200        Squashfs filesystem, little endian, version 4.0, compression:lzma, size: 2774624 bytes, 519 inodes, blocksize: 131072 bytes, created: 2015-04-13 09:35:04
```

https://blog.csdn.net/weixin_45582916

binwalk -e 该二进制文件，提取出各部分数据到各个文件，相当于解压固件，解压出来的文件放在当前目录的/_firmware.bin.extracted文件夹下（注意此时的suqashfs-root文件夹是否为空）



使用firmware-mod-kit解包提取出来的squashfs文件

/***

firmware-mod-kit的一些用法：

extract-firmware.sh 解包固件

build-firmware.sh 重新封包

check_for_upgrade.sh 检查更新

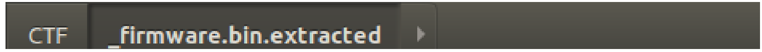
unsquashfs_all.sh 解包提取出来的squashfs文件

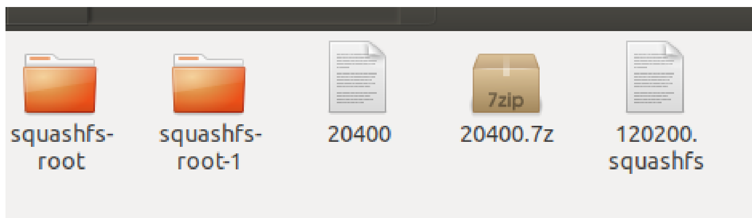
***/

```
zihaoli@zihaoliworld:~/CTF/_firmware.bin.extracted$ /home/zihaoli/firmware-mod-kit/unsquashfs_all.sh 120200.squashfs
/home/zihaoli/firmware-mod-kit/unsquashfs_all.sh: 行 85: ./src/binwalk: 没有那个文件或目录
Attempting to extract SquashFS .X file system...


Trying ./src/squashfs-2.1-r2/unsquashfs...
Trying ./src/squashfs-2.1-r2/unsquashfs-lzma...
Trying ./src/squashfs-3.0/unsquashfs...
Trying ./src/squashfs-3.0/unsquashfs-lzma...
Trying ./src/squashfs-3.0-lzma-damn-small-variant/unsquashfs-lzma...
Trying ./src/others/squashfs-2.0-nb4/unsquashfs...
Trying ./src/others/squashfs-3.0-e2100/unsquashfs...
Trying ./src/others/squashfs-3.0-e2100/unsquashfs-lzma...
Trying ./src/others/squashfs-3.2-r2/unsquashfs...
Trying ./src/others/squashfs-3.2-r2-lzma/squashfs3.2-r2/squashfs-tools/unsquashfs...
Trying ./src/others/squashfs-3.2-r2-hg612-lzma/unsquashfs...
Trying ./src/others/squashfs-3.2-r2-wnr1000/unsquashfs...
Trying ./src/others/squashfs-3.2-r2-rtn12/unsquashfs...
Trying ./src/others/squashfs-3.3/unsquashfs...
Trying ./src/others/squashfs-3.3-lzma/squashfs3.3/squashfs-tools/unsquashfs...
Trying ./src/others/squashfs-3.3-grml-lzma/squashfs3.3/squashfs-tools/unsquashfs...
Trying ./src/others/squashfs-3.4-cisco/unsquashfs...
Trying ./src/others/squashfs-3.4-nb4/unsquashfs...
Trying ./src/others/squashfs-3.4-nb4/unsquashfs-lzma...
Trying ./src/others/squashfs-4.2-official/unsquashfs... Parallel unsquashfs: Using 1 processor

Trying ./src/others/squashfs-4.2/unsquashfs... Parallel unsquashfs: Using 1 processor

Trying ./src/others/squashfs-4.0-lzma/unsquashfs-lzma... Parallel unsquashfs: Using 1 processor
480 inodes (523 blocks) to write

[=======================================================\         ] 454/523  86%
created 341 files
created 39 directories
created 70 symlinks
created 0 devices
created 0 fifos
File system sucessfully extracted!
MKFS="./src/others/squashfs-4.0-lzma/mksquashfs-lzma"
zihaoli@zihaoliworld:~/CTF/_firmware.bin.extracted$
```
https://blog.csdn.net/weixin_45582916

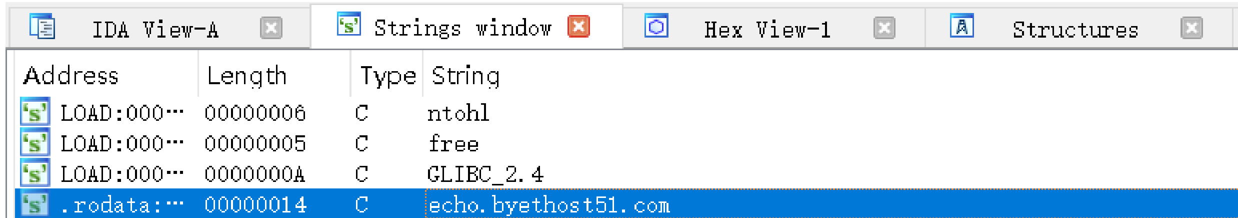
解包squashfs文件出来的文件放在了当前目录的squashfs-root-1文件夹中

（实际上，在本人虚拟机binwalk -e 该二进制文件后，生成文件夹/_firmware.bin.extracted中的squashfs-root文件夹，已经是解包squashfs文件出来的文件夹了，即squashfs-root文件夹和squashfs-root-1文件夹中包含的文件是相同的，如果squashfs-root文件夹下内容为空，是由于sasquatch安装有问题导致的，可以通过重新安装sasquatch解决，参考：dir815_FW_102.bin路由器固件解压碰到的坑）





在/_firmware.bin.extracted/squashfs-root/tmp目录或/_firmware.bin.extracted/squashfs-root-1/tmp目录下有一个backdoor文件
backdoor查壳发现有upx壳，脱壳后拖入ida分析
在字符串窗口找到网址



交叉引用网址，在initConnection函数中找到端口

```
 1 bool initConnection()
 2 {
 3   char *v0; // r0
 4   char s; // [sp+4h] [bp-208h]
 5   int v3; // [sp+204h] [bp-8h]
 6 
 7   memset(&s, 0, 0x200u);
 8   if ( mainCommSock )
 9   {
10     close(mainCommSock);
11     mainCommSock = 0;
12   }
13   if ( currentServer )
14     ++currentServer;
15   else
16     currentServer = 0;
17   strcpy(&s, (&commServer)[currentServer]);
18   v3 = 36667;
19   if ( strchr(&s, 58) )
20   {
21     v0 = strchr(&s, 58);
22     v3 = atoi(v0 + 1);
23     *strchr(&s, 58) = 0;
24   }
25   mainCommSock = socket(2, 1, 0);
26   return connectTimeout(mainCommSock, &s, v3, 30) == 0;
27 }
```
https://blog.csdn.net/weixin_45582916

[ACTF新生赛2020]Oruga

elf文件，无壳，ida分析

main函数逻辑清晰，获取输入，检验输入是否为“actf{”开头，验证输入内容，以及最后一个字符是否为“}”

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    __int64 result; // rax
4    __int64 v4; // [rsp+0h] [rbp-40h]
5    char v5; // [rsp+9h] [rbp-37h]
6    char s2[4]; // [rsp+Ah] [rbp-36h]
7    char input[40]; // [rsp+10h] [rbp-30h]
8    unsigned __int64 v8; // [rsp+38h] [rbp-8h]
9
10   v8 = __readfsqword(0x28u);
11   memset(input, 0, 0x19uLL);
12   printf("Tell me the flag:", 0LL);
13   scanf("%s", input);
14   strcpy(s2, "actf{");                        // s2="actf{"
15   LODWORD(v4) = 0;
16   while ( (signed int)v4 <= 4 )               // input的前5个字符放入v4中
17   {
18     *((_BYTE *)&v4 + (signed int)v4 + 4) = input[(signed int)v4];
19     LODWORD(v4) = v4 + 1;
20   }
21   v5 = 0;
22   if ( !strcmp((const char *)&v4 + 4, s2) )   // s2="actf{"，该if在验证input的前5个字符是否与“actf{”相同
23   {
24     if ( (unsigned __int8)check((__int64)input) )// 验证input内容，以及最后一个字符是否为“}”
25       printf("That's True Flag!", s2, v4);
26     else
27       printf("don't stop trying...", s2, v4);
28     result = 0LL;
29   }
30   else
31   {
32     printf("Format false!", s2, v4);
33     result = 0LL;
34   }
35   return result;
36 }
```

https://blog.csdn.net/weixin_45582916

进入check函数，分析可知是一个16x16的迷宫，和常规迷宫的按一次方向键则往该方向前进一格的机制不同，该迷宫的前进机制为，按一次方向键，则往该方向一直前进，直到撞到墙，具体分析知道，在map为零的位置时，可以持续前进，到达第一个非零的位置就停下来，然后减一个步长回到前一个零的位置，再读input的内容更新步长，更新步长实际上就是换方向

```c
_BOOL8 __fastcall check(__int64 input)
{
  int index; // [rsp+Ch] [rbp-Ch]
  signed int input_index; // [rsp+10h] [rbp-8h]
  signed int step; // [rsp+14h] [rbp-4h]

  index = 0;
  input_index = 5;
  step = 0;
  while ( map[index] != '!' )                   // 终点为"!"
  {
    index -= step;                              // 由第48行代码可知，由于index叠加走到了非零的位置，这时需要减一个步长
，回到前一个零位置
    if ( *(_BYTE *)(input_index + input) != 'W' || step == -16 )// 由input的内容决定步长step
    {
      if ( *(_BYTE *)(input_index + input) != 'E' || step == 1 )
      {
        if ( *(_BYTE *)(input_index + input) != 'M' || step == 16 )
        {
          if ( *(_BYTE *)(input_index + input) != 'J' || step == -1 )
            return 0LL;
          step = -1;                            // J-左
        }
        else
        {
          step = 16;                            // M-下
        }
      }
      else
      {
        step = 1;                               // E-右
      }
    }
    else
    {
      step = -16;                               // W-上
    }
    ++input_index;                              // 读下一个input的字符
    while ( !map[index] )                       // 该while循环体只能在map[index]=='0'时执行
    {
      if ( step == -1 && !(index & 0xF) )       // 判断边界的四个if
        return 0LL;
      if ( step == 1 && index % 16 == 15 )
        return 0LL;
      if ( step == 16 && (unsigned int)(index - 240) <= 0xF )
        return 0LL;
      if ( step == -16 && (unsigned int)(index + 15) <= 0x1E )
        return 0LL;
      index += step;                            // index按照当前的步长循环叠加，即按下一个方向键，就会沿着这个方向一直走
，直到第一个非零的位置停下
    }
  }
  return *(_BYTE *)(input_index + input) == '}';
}
```

把map提取出来，制成16x16的迷宫，按照前进机制走完迷宫即可，起始点为左上角的[0,0]，终止点为“！”（0x21），W-上，M-下，J-左，E-右，路线即为flag

```
0x00, 0x00, 0x00, 0x00, 0x23, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x23, 0x23, 0x23, 0x23, \
0x00, 0x00, 0x00, 0x23, 0x23, 0x00, 0x00, 0x00, 0x4F, 0x4F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4F, 0x4F, 0x00, 0x50, 0x50, 0x00, 0x00, 0x00, \
0x00, 0x00, 0x00, 0x4C, 0x00, 0x4F, 0x4F, 0x00, 0x4F, 0x4F, 0x00, 0x50, 0x50, 0x00, 0x00, 0x00, \
0x00, 0x00, 0x00, 0x4C, 0x00, 0x4F, 0x4F, 0x00, 0x4F, 0x4F, 0x00, 0x50, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x4C, 0x4C, 0x00, 0x4F, 0x4F, 0x00, 0x00, 0x00, 0x00, 0x50, 0x00, 0x00, 0x00, 0x00, \
0x00, 0x00, 0x00, 0x00, 0x00, 0x4F, 0x4F, 0x00, 0x00, 0x00, 0x00, 0x50, 0x00, 0x00, 0x00, 0x00, \
0x23, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x23, 0x00, 0x00, 0x00, \
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4D, 0x4D, 0x4D, 0x00, 0x00, 0x00, 0x23, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4D, 0x4D, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x45, 0x45, \
0x00, 0x00, 0x00, 0x30, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x45, 0x00, \
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x45, 0x45, \
0x54, 0x54, 0x54, 0x49, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x45, 0x00, \
0x00, 0x54, 0x00, 0x49, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x45, 0x00,
0x00, 0x54, 0x00, 0x49, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D, 0x21, 0x00, 0x00, 0x00, 0x45, 0x45

W-上  M-下  J-左  E-右

MEWEMEWJMEWJM
```

https://blog.csdn.net/weixin_45582916

[Zer0pts2020]easy strcmp

elf文件，无壳，ida分析

main函数，有一个比较字符串的if语句决定输出的内容，其他什么也没有

```
 1 __int64 __fastcall main(signed int a1, char **a2, char **a3)
 2 {
 3   if ( a1 > 1 )
 4   {
 5     if ( !strcmp(a2[1], "zer0pts{********CENSORED********}") )
 6       puts("Correct!");
 7     else
 8       puts("Wrong!");
 9   }
10   else
11   {
12     printf("Usage: %s <FLAG>\n", *a2, a3, a2);
13   }
14   return 0LL;
15 }
```

https://blog.csdn.net/weixin_45582916

来到start函数，发现在调用main函数前，先调用了fini函数和init函数

```
.text:00000000000005E0 ; Attributes: noreturn
.text:00000000000005E0
.text:00000000000005E0                 public start
.text:00000000000005E0 start           proc near               ; DATA XREF: LOAD:0000000000000018↑o
.text:00000000000005E0 ; __unwind {
.text:00000000000005E0                 xor     ebp, ebp
.text:00000000000005E2                 mov     r9, rdx         ; rtld_fini
.text:00000000000005E5                 pop     rsi             ; argc
.text:00000000000005E6                 mov     rdx, rsp        ; ubp_av
.text:00000000000005E9                 and     rsp, 0FFFFFFFFFFFFFFF0h
```

```
.text:00000000000005ED                 push    rax
.text:00000000000005EE                 push    rsp             ; stack_end
.text:00000000000005EF                 lea     r8, fini        ; fini
.text:00000000000005F6                 lea     rcx, init       ; init
.text:00000000000005FD                 lea     rdi, main       ; main
.text:0000000000000604                 call    cs:__libc_start_main_ptr
.text:000000000000060A                 hlt
.text:000000000000060A ; } // starts at 5E0
.text:000000000000060A start           endp
.text:000000000000060A
```

https://blog.csdn.net/weixin_45582916

fini函数直接返回了，分析init函数，可以知道，在调用main函数前，程序将.init_array段地址到.fini_array段地址之间的函数全部执行一遍，命令行参数作为段之间函数的参数

```
 1 void __fastcall init(unsigned int a1, __int64 a2, __int64 a3)
 2 {
 3   __int64 v3; // r15
 4   signed __int64 v4; // rbp
 5   __int64 v5; // rbx
 6
 7   v3 = a3;
 8   v4 = &off_200DF0 - &off_200DE0;              // .fini_array段的地址-.init_array段的地址
 9   init_proc();
10   if ( v4 )
11   {
12     v5 = 0LL;
13     do
14       ((void (__fastcall *)(_QWORD, __int64, __int64))*(&off_200DE0 + v5++))(a1, a2, v3);// 从.init_array段开始地址执行到.fini_array段的地址
15     while ( v4 != v5 );
16   }
17 }
```

https://blog.csdn.net/weixin_45582916

这里可以看到，.init_array段和.fini_array段之间有3个函数，依次分析知道，重要的是sub_795函数

```
.init_array:0000000000200DE0 ; Segment alignment 'qword' can not be represented in assembly
.init_array:0000000000200DE0 _init_array     segment para public 'DATA' use64
.init_array:0000000000200DE0                 assume cs:_init_array
.init_array:0000000000200DE0                 ;org 200DE0h
.init_array:0000000000200DE0 off_200DE0      dq offset sub_6E0       ; DATA XREF: LOAD:00000000000000F8↑o
.init_array:0000000000200DE0                                         ; LOAD:0000000000000210↑o ...
.init_array:0000000000200DE8                 dq offset sub_795
.init_array:0000000000200DE8 _init_array     ends
.init_array:0000000000200DE8
.fini_array:0000000000200DF0 ; ELF Termination Function Table
.fini_array:0000000000200DF0 ; ===========================================================================
.fini_array:0000000000200DF0
.fini_array:0000000000200DF0 ; Segment type: Pure data
.fini_array:0000000000200DF0 ; Segment permissions: Read/Write
.fini_array:0000000000200DF0 ; Segment alignment 'qword' can not be represented in assembly
.fini_array:0000000000200DF0 _fini_array     segment para public 'DATA' use64
.fini_array:0000000000200DF0                 assume cs:_fini_array
.fini_array:0000000000200DF0                 ;org 200DF0h
.fini_array:0000000000200DF0 off_200DF0      dq offset sub_6A0       ; DATA XREF: init+13↑o
.fini_array:0000000000200DF0 _fini_array     ends
.fini_array:0000000000200DF0
```

https://blog.csdn.net/weixin_45582916

sub_795->sub_6EA，分析sub_6EA函数，计算输入的长度，将输入的内容顺序地与qword_201060数组的元素相减，然后去到main函数和那段字符串比较

```
 1 __int64 __fastcall sub_6EA(__int64 input, __int64 a2)
 2 {
 3   int i; // [rsp+18h] [rbp-8h]
 4   int input_len; // [rsp+18h] [rbp-8h]
 5   int j; // [rsp+1Ch] [rbp-4h]
 6
 7   for ( i = 0; *(_BYTE *)(i + input); ++i )
 8     ;
 9   input_len = (i >> 3) + 1;
```

```
10     for ( j = 0; j < input_len; ++j )
11       *(_QWORD *)(8 * j + input) -= qword_201060[j];
12     return qword_201090(input, a2);
13   }
```

https://blog.csdn.net/weixin_45582916

写脚本即可得到flag

```
arr=[0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x42, 0x09,
  0x4A, 0x49, 0x35, 0x43, 0x0A, 0x41, 0xF0, 0x19, 0xE6, 0x0B,
  0xF5, 0xF2, 0x0E, 0x0B, 0x2B, 0x28, 0x35, 0x4A, 0x06, 0x3A,
  0x0A, 0x4F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
str="zer0pts{********CENSORED********}"
flag=""
for i in range(len(str)):
    flag+=chr((ord(str[i])+arr[i])%128)
print(flag)
```

```
test1
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
zer0pts{l3ts_m4k3_4_DDSOUR_t0d4y}
```

https://blog.csdn.net/weixin_45582916

[GXYCTF2019]simple CPP

exe程序，运行后提示输入flag，输入错误退出程序，无壳，ida分析
交叉引用字符串来到sub_140001290函数

```
__int64 sub_140001290()
{
  bool v0; // si
  __int64 v1; // rax
  __int64 v2; // r8
  unsigned __int8 *v3; // rax
  unsigned __int8 *v4; // rbx
  int v5; // er10
  __int64 v6; // r11
  _BYTE *input_copy; // r9
  void **v8; // r8
  __int64 arr[3]; // rdi
  __int64 arr[2]; // r15
  __int64 arr[1]; // r12
  __int64 arr[0]; // rbp
  signed int v13; // ecx
  unsigned __int8 *v14; // rdx
  __int64 v15; // rdi
  __int64 *v16; // r14
  __int64 v17; // rbp
  __int64 v18; // r13
  _QWORD *v19; // rdi
  __int64 v20; // r12
  __int64 v21; // r15
  __int64 v22; // rbp
  __int64 v23; // rdx
  __int64 v24; // rbp
  __int64 v25; // rbp
  __int64 v26; // r10
  __int64 v27; // rdi
  __int64 v28; // r8
```

```
bool v29; // dl
__int64 v30; // rax
void *v31; // rdx
const char *v32; // rax
__int64 v33; // rax
_BYTE *v34; // rcx
__int64 v36; // [rsp+20h] [rbp-68h]
void *input; // [rsp+30h] [rbp-58h]
unsigned __int64 input_len; // [rsp+40h] [rbp-48h]
unsigned __int64 v39; // [rsp+48h] [rbp-40h]

v0 = 0;
input_len = 0i64;
v39 = 15i64;
LOBYTE(input) = 0;
LODWORD(v1) = printf(std::cout, "I'm a first timer of Logic algebra , how about you?");
std::basic_ostream<char,std::char_traits<char>>::operator<<(v1, sub_140001B90);
printf(std::cout, "Let's start our game,Please input your flag:");
sub_140001DE0(std::cin, &input, v2);            // 读取input
std::basic_ostream<char,std::char_traits<char>>::operator<<(std::cout, sub_140001B90);
if ( input_len - 5 > 25 )                       // input不需要GXY{}包住
{
  LODWORD(v33) = printf(std::cout, "Wrong input ,no GXY{} in input words");
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v33, sub_140001B90);
  goto LABEL_45;
}
v3 = sub_1400024C8(32ui64);                     // 开辟一块大小为32的，元素类型为unsigned int8的地址空间给v3
v4 = v3;
if ( v3 )                                       // 新开辟的地址空间全部赋0值
{
  *v3 = 0i64;
  *(v3 + 1) = 0i64;
  *(v3 + 2) = 0i64;
  *(v3 + 3) = 0i64;
}
else
{
  v4 = 0i64;
}
v5 = 0;
if ( input_len > 0 )
{
  v6 = 0i64;
  do
  {
    input_copy = &input;
    if ( v39 >= 16 )
      input_copy = input;
    v8 = &Dst;
    if ( qword_140006060 >= 0x10 )
      v8 = Dst;                                 // Dst="i_will_check_is_debug_or_not"，长度为28
    v4[v6] = input_copy[v6] ^ *(v8 + v5++ % 27);// input和Dst异或，结果放入v4
    ++v6;
  }
  while ( v5 < input_len );
}
arr[3] = 0i64;
arr[2] = 0i64;
arr[1] = 0i64;
arr[0] = 0i64;
```

```
  if ( input_len > 30 )                          // input长度验证
    goto LABEL_28;
  v13 = 0;
  if ( input_len <= 0 )
    goto LABEL_28;
  v14 = v4;                                      // v14=v4，是input和Dst异或的结果
  do                                             // do循环体，实际上是把v14分成四段，前三段长度为8，分别放入v12，v11，v
10中，剩下的放在v9
                                                 // 依次记为arr[0]，arr[1]，arr[2]，arr[3]，重要的是算出arr[0~3]

  {
    v15 = *v14 + arr[3];
    ++v13;
    ++v14;
    switch ( v13 )
    {
      case 8:
        arr[0] = v15;
        goto LABEL_24;
      case 16:
        arr[1] = v15;
        goto LABEL_24;
      case 24:
        arr[2] = v15;
LABEL_24:
        v15 = 0i64;
        break;
      case 32:
        printf(std::cout, "ERRO,out of range");
        exit(1);
        break;
    }
    arr[3] = v15 << 8;
  }
  while ( v13 < input_len );
  if ( arr[0] )
  {
    v16 = sub_1400024C8(32ui64);
    *v16 = arr[0];                               // arr[0~3]放入v16[0~3]中
    v16[1] = arr[1];
    v16[2] = arr[2];
    v16[3] = arr[3];
    goto LABEL_29;
  }
LABEL_28:
  v16 = 0i64;
LABEL_29:
  v36 = v16[2];                                  // v36=arr[2]
  v17 = v16[1];                                  // v17=arr[1]
  v18 = *v16;                                    // v18=arr[0]
  v19 = sub_14000223C(32ui64);                   // 开辟一块大小为32的，元素类型为unsigned int8的地址空间给v19
  if ( IsDebuggerPresent() )                     // 反调试
  {
    printf(std::cout, "Hi , DO not debug me !");
    Sleep(0x7D0u);
    exit(0);
  }
  v20 = v17 & v18;                               // v20=arr[1]&arr[0]
  *v19 = v17 & v18;                              // v19[0]=arr[1]&arr[0]
  v21 = v36 & ~v18;                              // v21=arr[2]&(~arr[0])
```

```
  v19[1] = v21;                                   // v19[1]=arr[2]&(~arr[0])
  v22 = ~v17;                                     // v22=~arr[1]
  v23 = v36 & v22;                                // v23=arr[2]&(~arr[1])
  v19[2] = v36 & v22;                             // v19[2]=arr[2]&(~arr[1])
  v24 = v18 & v22;                                // v24=arr[0]&(~arr[1])
  v19[3] = v24;                                   // v19[3]=arr[0]&(~arr[1])
  if ( v21 != 0x11204161012i64 )                  // 验证v19[1]==0x11204161012，即arr[2]&(~arr[0])==0x11204161012
  {
    v19[1] = 0i64;
    v21 = 0i64;
  }
  v25 = v21 | v20 | v23 | v24;                    // 需v25==0x3E3A4717373E7F1F成立，即(arr[2]&(~arr[0])) | (arr[1]&
arr[0]) | (arr[2]&(~arr[1])) | (arr[0]&(~arr[1]))==0x3E3A4717373E7F1F
  v26 = v16[1];                                   // v26=arr[1]
  v27 = v16[2];                                   // v27=arr[2]
  v28 = v23 & *v16 | v27 & (v20 | v26 & ~*v16 | ~(v26 | *v16));// (arr[2]&(~arr[1])) & (arr[0]) | (arr[2]) & ((a
rr[1]&arr[0]) | (arr[1]) & ~(arr[0]) | ~((arr[1]) | (arr[0])))==0x8020717153E3013

  v29 = 0;
  if ( v28 == 0x8020717153E3013i64 )              // 由第171行代码可知，v0需要为1，v28==0x8020717153E3013成立
    v29 = v25 == 0x3E3A4717373E7F1Fi64;           // 验证v25==0x3E3A4717373E7F1F，相等返回1，不等返回0
  if ( (v25 ^ v16[3]) == 0x3E3A4717050F791Fi64 )// v16[3]=arr[3]，即arr[3]^0x3E3A4717373E7F1F==0x3E3A4717050F791
F
    v0 = v29;                                     // v0为v25==0x3E3A4717373E7F1F的返回值，等式成立返回1，不等返回0
  if ( (v21 | v20 | v26 & v27) != (~*v16 & v27 | 0xC00020130082C0Ci64) || v0 != 1 )// 需要v0为1，意味着v25==0x3E3
A4717373E7F1F成立且((arr[2]&(~arr[0])) |(arr[1]&arr[0]) | (arr[1]) & (arr[2])) == (~(arr[0])& (arr[2]) | 0xC0002
0130082C0C)
  {
    printf(std::cout, "Wrong answer!try again");
    j_j_free(v4);
  }
  else
  {
    LODWORD(v30) = printf(std::cout, "Congratulations!flag is GXY{");// input不需要GXY{}包住
    v31 = &input;
    if ( v39 >= 16 )
      v31 = input;
    v32 = sub_140001FD0(v30, v31, input_len);
    printf(v32, "}");
    j_j_free(v4);
  }
LABEL_45:
  if ( v39 >= 0x10 )
  {
    v34 = input;
    if ( v39 + 1 >= 0x1000 )
    {
      v34 = *(input - 1);
      if ( (input - v34 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v34);
  }
  return 0i64;
}
```

写解arr[0~3]的脚本

```
from z3 import *
arr=[BitVec('arr[%d]'%i,64) for i in range(4)]
s=Solver()
s.add(arr[2]&(~arr[0])==0x11204161012)
s.add(arr[3]^0x3E3A4717373E7F1F==0x3E3A4717050F791F)
s.add(((arr[2]&(~arr[0])) |(arr[1]&arr[0]) | (arr[1]) & (arr[2])) == (~(arr[0])& (arr[2]) | 0xC00020130082C0C))
s.add((arr[2]&(~arr[1])) & (arr[0]) | (arr[2]) & ((arr[1]&arr[0]) | (arr[1]) & ~(arr[0]) | ~((arr[1]) | (arr[0])))==0x8020717153E3013)
s.add((arr[2]&(~arr[0])) | (arr[1]&arr[0]) | (arr[2]&(~arr[1])) | (arr[0]&(~arr[1]))==0x3E3A4717373E7F1F)
if s.check():
    print(s.model())
```

```
test1
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
[arr[2] = 577031497978884115,
 arr[0] = 4483973367147818765,
 arr[1] = 864693332579200012,
 arr[3] = 842073600]
```

https://blog.csdn.net/weixin_45582916

写逆异或运算得到flag的脚本，由flag的明文字符串可知，arr[1]的结果错误，原因是原方程组有多组解，参考别的师傅的wp，比赛给出了第二部分的flag，e!P0or_a，替换掉错误的8个字符，结果为We1l_D0ne!P0or_algebra_am_i

```
Dst="i_will_check_is_debug_or_not"
arr=[0]*4
arr[2] = 577031497978884115
arr[0] = 4483973367147818765
arr[1] = 864693332579200012
arr[3] = 842073600
for i in range(4):
    print(hex(arr[i]).replace('0x','').replace('L','').zfill(16))
flag=[0x3e,0x3a,0x46,0x05,0x33,0x28,0x6f,0x0d,  #3e3a460533286f0d
      0x0c,0x00,0x02,0x01,0x30,0x08,0x2c,0x0c,  #0c00020130082c0c
      0x08,0x02,0x07,0x17,0x15,0x3e,0x30,0x13,  #08020717153e3013
      0x32,0x31,0x06]# 0000000032310600
s=""
for i in range(len(flag)):
    s+=chr(flag[i]^ord(Dst[i%27]))
print(s)
```

```
test2
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test2.py
3e3a460533286f0d
0c00020130082c0c
08020717153e3013
0000000032310600
We1l_D0ndeajoa_Slgebra_am_i
```

https://blog.csdn.net/weixin_45582916