攻防世界逆向高手题的BABYRE

原创

沐一 · 林 于 2021-08-21 18:04:49 发布 154 收藏 1

分类专栏： CTF 逆向 文章标签： unctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/119838441

CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏

逆向

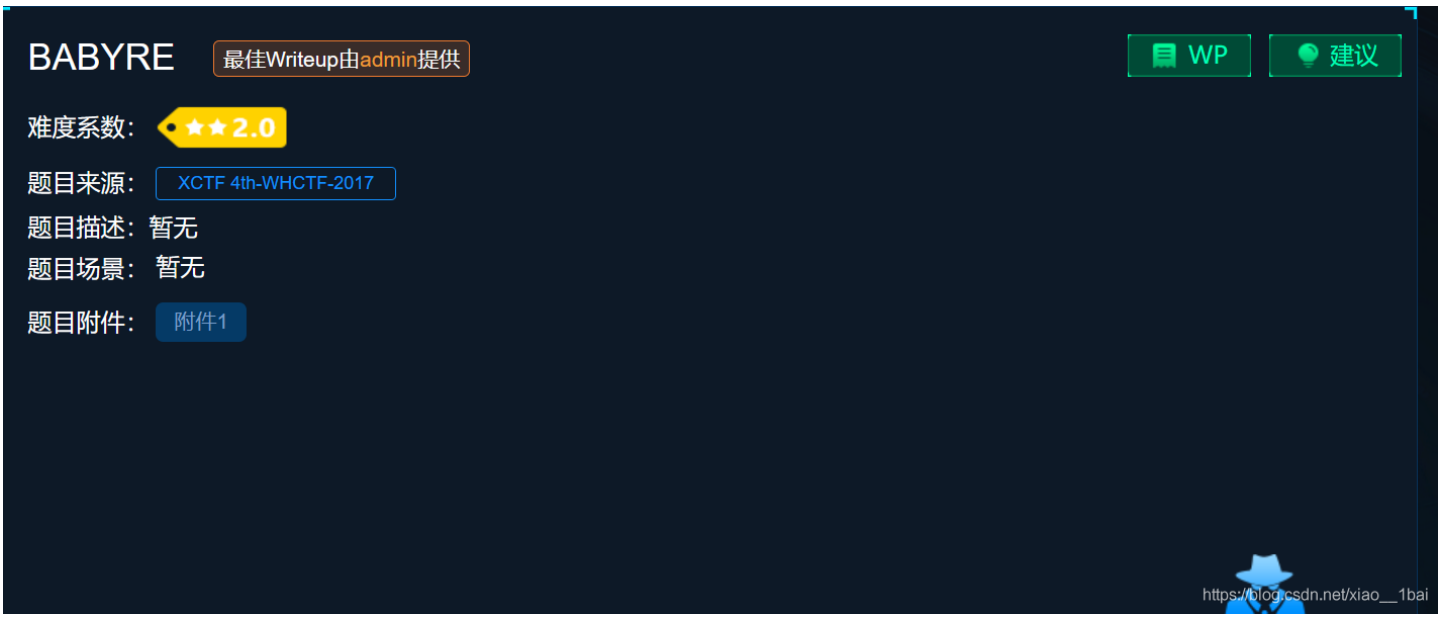
95 篇文章 6 订阅

订阅专栏

攻防世界逆向高手题之BABYRE

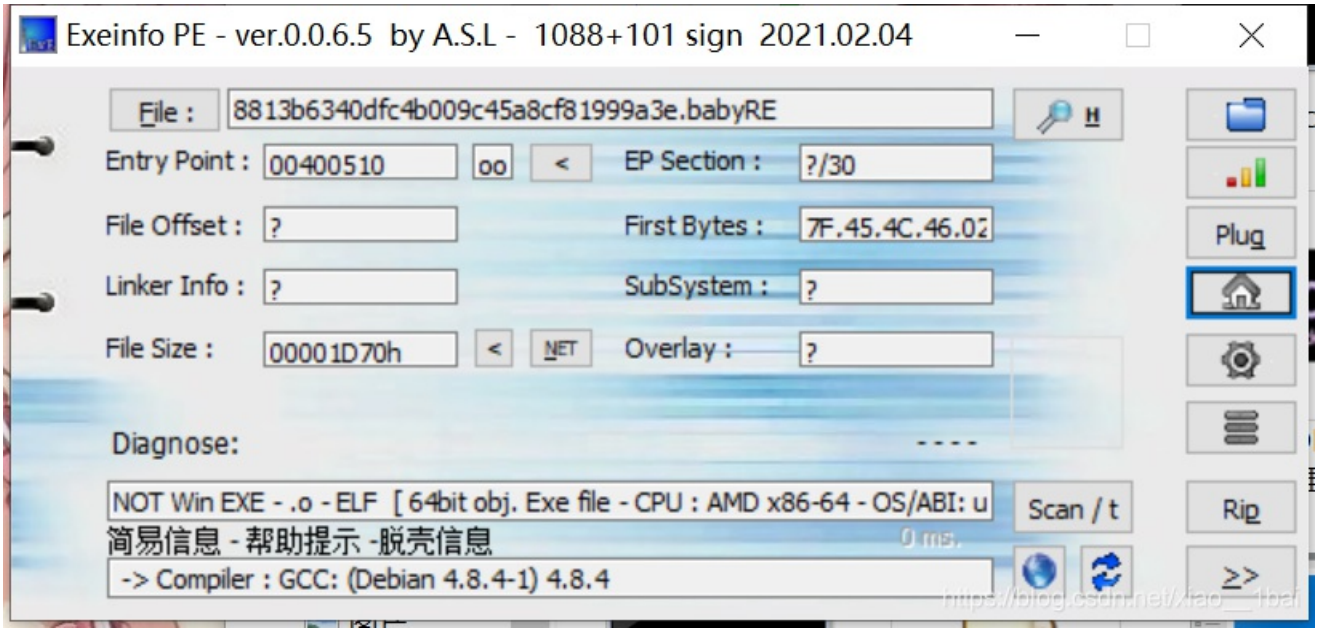
继续开启全栈梦想之逆向之旅~

这题是攻防世界逆向高手题的BABYRE



这题应该是我第一次接触的花指令题，怪兴奋的！目前我的理解是代码和数据混淆，让IDA误认为是数据而反汇编出错。

好了，开始分析：
照例扔入exeinpefo中查看信息：



64位ELF文件，无壳，扔入IDA64位中查看伪代码：

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char input_flag[24]; // [rsp+0h] [rbp-20h] BYREF
4   int v5; // [rsp+18h] [rbp-8h]
5   int i; // [rsp+1Ch] [rbp-4h]
6
7   for ( i = 0; i <= 181; ++i )
8     judge[i] ^= 0xCu;
9   printf("Please input flag:");
10  __isoc99_scanf("%20s", input_flag);
11  v5 = strlen(input_flag);
12  if ( v5 == 14 && (*(unsigned int (__fastcall **)(char *))judge)(input_flag) )
13    puts("Right!");
14  else
15    puts("Wrong!");
16  return 0;
17 }
```

这里犯下第一个错误：
由于是第一次接触花指令，对 (*(unsigned int (__fastcall **)(char *))judge)(input_flag)) 这一行代码我看了好久愣是没看懂，input_flag是我们输入的字符串,judge中文是判断，根据以前做题这种中文类名字的确是暗示，可前面显示的judge[i] 这摆明是个数组啊，judge(input_flag) 不就变成了数组首地址加Input_flag了吗？还有这种玩法？？？

然后我就去看wp了(笑~)：
他们大概意思了judge是一个函数名，这个judge函数名的函数呢是通过前面逻辑用数据生成出来的。
想起C语言好像有通过字符串拼接生成新命令的技巧，突然就恍然大悟了。
所以这里 (*(unsigned int (__fastcall **)(char *))judge)(input_flag)) 应该这样分析，unsigned int 是judge函数的返回类型，(__fastcall **)是函数的调用约定，(char *)这个只是提取judge的数组头的一连串字符串做函数名而已。

然后看他们WP中显示judge是双击跟踪跟踪不了的，会报错，我的倒是跟踪得了，长这样：

```
.data:0000000000600B00 ; char judge[182]
.data:0000000000600B00 judge           db 59h                  ; CODE XREF: main+80↑p
.data:0000000000600B00                                         ; DATA XREF: main+16↑r ...
.data:0000000000600B01                 db  44h ; D
.data:0000000000600B02                 db  85h
.data:0000000000600B03                 db 0E9h
.data:0000000000600B04                 db  44h ; D
.data:0000000000600B05                 db  85h
.data:0000000000600B06                 db  71h ; q
.data:0000000000600B07                 db 0D4h
.data:0000000000600B08                 db 0CAh
.data:0000000000600B09                 db  49h ; I
.data:0000000000600B0A                 db 0ECh
.data:0000000000600B0B                 db  6Ah ; j
.data:0000000000600B0C                 db 0CAh
.data:0000000000600B0D                 db  49h ; I
.data:0000000000600B0E                 db 0EDh
.data:0000000000600B0F                 db  61h ; a
.data:0000000000600B10                 db 0CAh
.data:0000000000600B11                 db  49h ; I
.data:0000000000600B12                 db 0EEh
.data:0000000000600B13                 db  6Fh ; o
.data:0000000000600B14                 db 0CAh
.data:0000000000600B15                 db  49h ; I
.data:0000000000600B16                 db 0EFh
00000B00 0000000000600B00: .data:judge (Synchronized with Hex View-1)
```

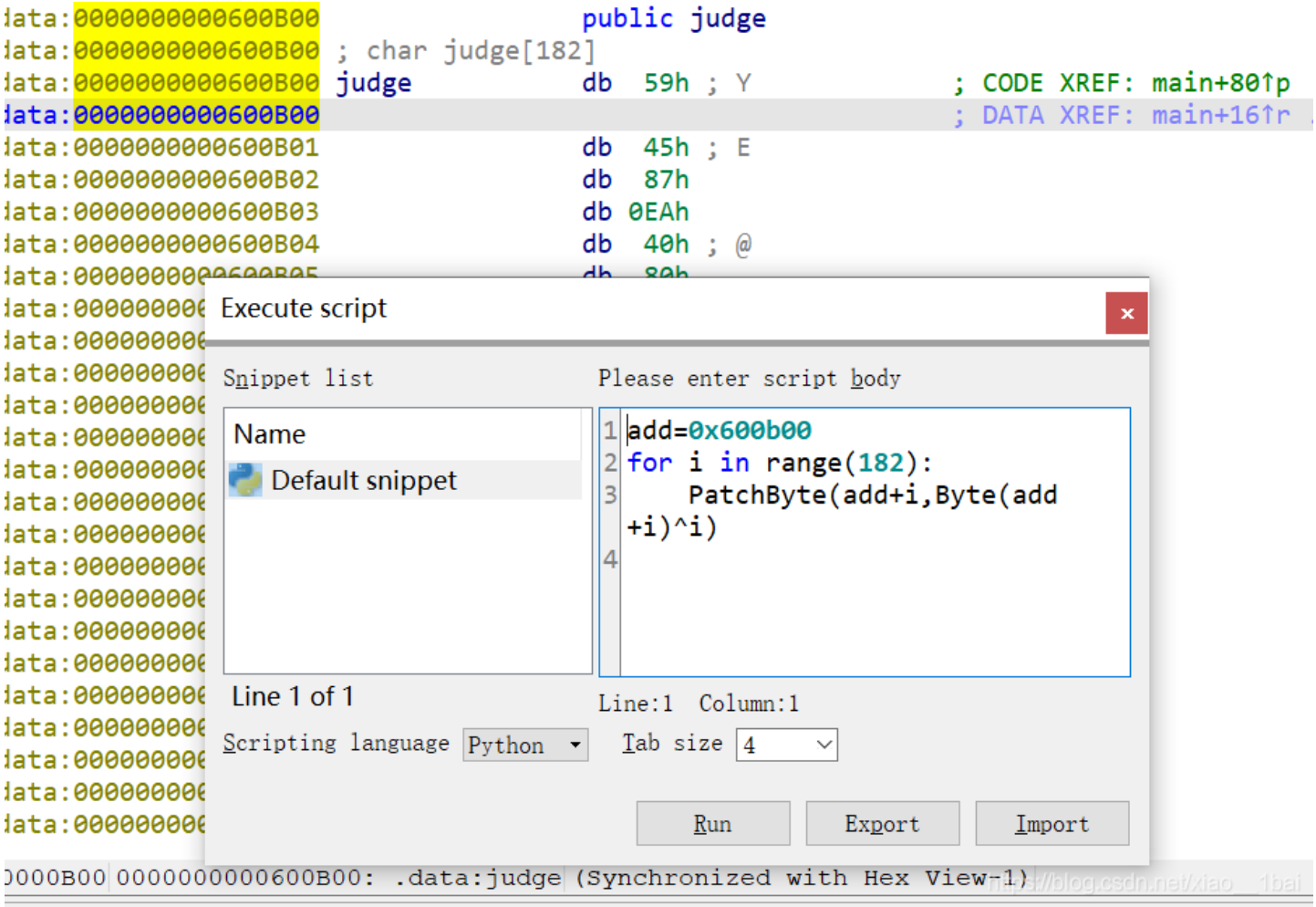
这里犯下第二个错误：
一开始这里也卡了我好一会，后来发现是我用了新版的IDA7.5,这里本来有个指针分析错误，IDA7.5直接把它当成数据去了，如果我用C（汇编），P（创建函数），在judge开头出按一下照样显示指针分析错误：(ps：不按C的话有时IDA会报错函数在指定地址具有未定义的指令/数据。您的请求已放入自动分析队列。按一下C可能是提醒它可以转成数据吧)

```
.data:0000000000600B00
.data:0000000000600B00 ; char judge[182]
.data:0000000000600B00                 public judge
.data:0000000000600B00 judge           proc far                ; CODE XREF: main+80↑p
.data:0000000000600B00                                         ; DATA XREF: main+16↑r ...
.data:0000000000600B00                 pop     rcx
.data:0000000000600B01                 test    ecx, r13d
.data:0000000000600B04                 test    [rcx-2Ch], r14d
.data:0000000000600B08                 retf    0EC49h
.data:0000000000600B08 judge           endp ; sp-analysis failed
.data:0000000000600B08
.data:0000000000600B08 ; ---------------------------------------------------------------------------
.data:0000000000600B0B                 db  6Ah ; j
.data:0000000000600B0C                 db 0CAh
.data:0000000000600B0D                 db  49h ; I
.data:0000000000600B0E                 db 0EDh
```

知道原理后我们可以开始做题了，顺带说一下这题型是用户输入相关的生成型flag，所以要逆向逻辑。

第一种方法：直接嵌入脚本在让他把加密流程跑一遍得出真正judge代码逻辑，根据逻辑解密：

```
for ( i = 0; i <= 181; ++i )
  judge[i] ^= 0xCu;
```



代码如下：(记住按一遍即可，按两遍就归回原样了)

```
add=0x600b00
for i in range(182):
    PatchByte(add+i,Byte(add+i)^0xC)
```

然后C，P键扫描生成函数，如果有红色的行就用U设为未定义再C，P键即可：

```
.data:0000000000600B00 ; ---------------------------------------------------------------------------
.data:0000000000600B00
.data:0000000000600B00 ; char judge[182]
.data:0000000000600B00                 public judge
.data:0000000000600B00 judge:                              ; CODE XREF: main+80↑p
.data:0000000000600B00                                     ; DATA XREF: main+16↑r ...
.data:0000000000600B00                 pop     rcx
.data:0000000000600B01                 xchg    r13d, r10d
.data:0000000000600B04                 xor     byte ptr [rdi-2Dh], 0C2h
.data:0000000000600B09                 db      40h             ; PC/XT PPI port B bits:
.data:0000000000600B09                 out     61h, al         ; 0: Tmr 2 gate ┐┌► OR 03H=spkr ON
.data:0000000000600B09                                         ; 1: Tmr 2 data ┘└  AND 0fcH=spkr OFF
.data:0000000000600B09                                         ; 3: 1=read high switches
.data:0000000000600B09                                         ; 4: 0=enable RAM parity checking
.data:0000000000600B09                                         ; 5: 0=enable I/O channel check
.data:0000000000600B09                                         ; 6: 0=hold keyboard clock low
.data:0000000000600B09                                         ; 7: 0=enable kbrd
.data:0000000000600B0C                 mov     byte ptr [rbx+6Eh], 0DAh
.data:0000000000600B11                 pop     rax
.data:0000000000600B12                 cld
.data:0000000000600B13                 jl      short near ptr unk_600AF3
.data:0000000000600B15                 pop     rsp
.data:0000000000600B16                 stc
.data:0000000000600B17                 jg      short near ptr unk_600AEB
.data:0000000000600B19                 push    rax
.data:0000000000600B1A                 repne push 78F754D6h
00000B00 0000000000600B00: .data:judge (Synchronized with Hex View-1)
```

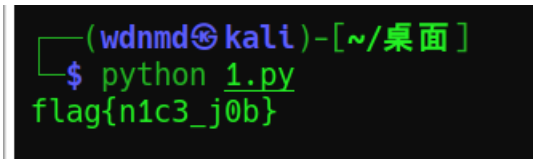


```
1  __int64 __fastcall judge(__int64 a1)
2  {
3    char v2[5]; // [rsp+8h] [rbp-20h] BYREF
4    char v3[9]; // [rsp+Dh] [rbp-1Bh] BYREF
5    int i; // [rsp+24h] [rbp-4h]
6
7    qmemcpy(v2, "fmcd", 4);
8    v2[4] = 127;
9    qmemcpy(v3, "k7d;V`;np", sizeof(v3));
10   for ( i = 0; i <= 13; ++i )
11     *(_BYTE *)(i + a1) ^= i;
12   for ( i = 0; i <= 13; ++i )
13   {
14     if ( *(_BYTE *)(i + a1) != v2[i] )
15       return 0LL;
16   }
17   return 1LL;
18 }
```


函数逻辑很简单，flag是用户输入有关的生成型flag，对输入的简单的异或然后比较每个字符与预先给的字符串是否相等，等就返回1(true)，所以我们直接用预先给的字符串逆逻辑异或即可：

```python
#key1="fmcd"
#key2=chr(0x7f)
#key3="k7d;V`;np"
#key4=key1+key2+key3
key="fmcd\x7Fk7d;V`;np"
flag=""
for i in range(14):
 flag+=chr(ord(key[i])^i)
print(flag)
```

代码如上，犯下的第 3 个错误就是我一开始以为python会吧\x7F判断成4个字符，结果是我多虑了，直接就是python自己会解析\x类型，太妙了，结果如图：



第二种方法：

直接远程调试，在第12行 if (v5 == 14 && (unsigned int)judge((__int64)s))处断下代码，断在这里的话IDA已经把judge解密完了，直接双击跟踪生成函数即可：

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char s[24]; // [rsp+0h] [rbp-20h] BYREF
4    int v5; // [rsp+18h] [rbp-8h]
5    int i; // [rsp+1Ch] [rbp-4h]
6
7    for ( i = 0; i <= 181; ++i )
8      *((_BYTE *)judge + i) ^= 0xCu;
9    printf("Please input flag:");
10   __isoc99_scanf("%20s", s);
11   v5 = strlen(s);
12   if ( v5 == 14 && (unsigned int)judge((__int64)s) )
13     puts("Right!");
14   else
15     puts("Wrong!");
16   return 0;
17 }
```



```
1  __int64 __fastcall judge(__int64 a1)
2  {
3    char v2[5]; // [rsp+8h] [rbp-20h] BYREF
4    char v3[9]; // [rsp+Dh] [rbp-1Bh] BYREF
5    int i; // [rsp+24h] [rbp-4h]
6
7    qmemcpy(v2, "fmcd", 4);
8    v2[4] = 127;
9    qmemcpy(v3, "k7d;V`;np", sizeof(v3));
10   for ( i = 0; i <= 13; ++i )
11     *(_BYTE *)(i + a1) ^= i;
12   for ( i = 0; i <= 13; ++i )
13   {
14     if ( *(_BYTE *)(i + a1) != v2[i] )
15       return 0LL;
16   }
17   return 1LL;
18 }
```


最后：

由于使用新的IDA7.5，一开始遇到了Byte等IDApython函数未定义的问题，在以下博客中找到了解决方法,非常感谢！：(IDA Pro 7.5版本使用IDAPython)

https://blog.csdn.net/HaiLanLin/article/details/115585340

总结：

> 1：犯下第一个错误
> 由于是第一次接触花指令，对 (*(unsigned int (__fastcall **)(char *))judge)(input_flag)) *这一行代码我看了好久愣是没看懂。*
> *(*(unsigned int (__fastcall **)(char *))judge)(input_flag)) 应该这样分析，unsigned int 是judge函数的返回类型，(__fastcall **)是函数的调用约定，(char *)这个只是提取judge的数组头的一连串字符串做函数名而已。*

> 2：犯下第二个错误：
> 我用了新版的IDA7.5,这里本来有个指针分析错误，IDA7.5直接把它当成数据去了，如果我用C（汇编），P（创建函数），在judge开头出按一下照样显示指针分析错误：(ps：不按C的话有时IDA会报错函数在指定地址具有未定义的指令/数据。您的请求已放入自动分析队列。按一下C可能是提醒它可以转成数据吧)

> 3：犯下的第 3 个错误就是我一开始以为python会吧\x7F判断成4个字符，结果是我多虑了，直接就是python自己会解析\x类型，太妙了。

解毕！敬礼！