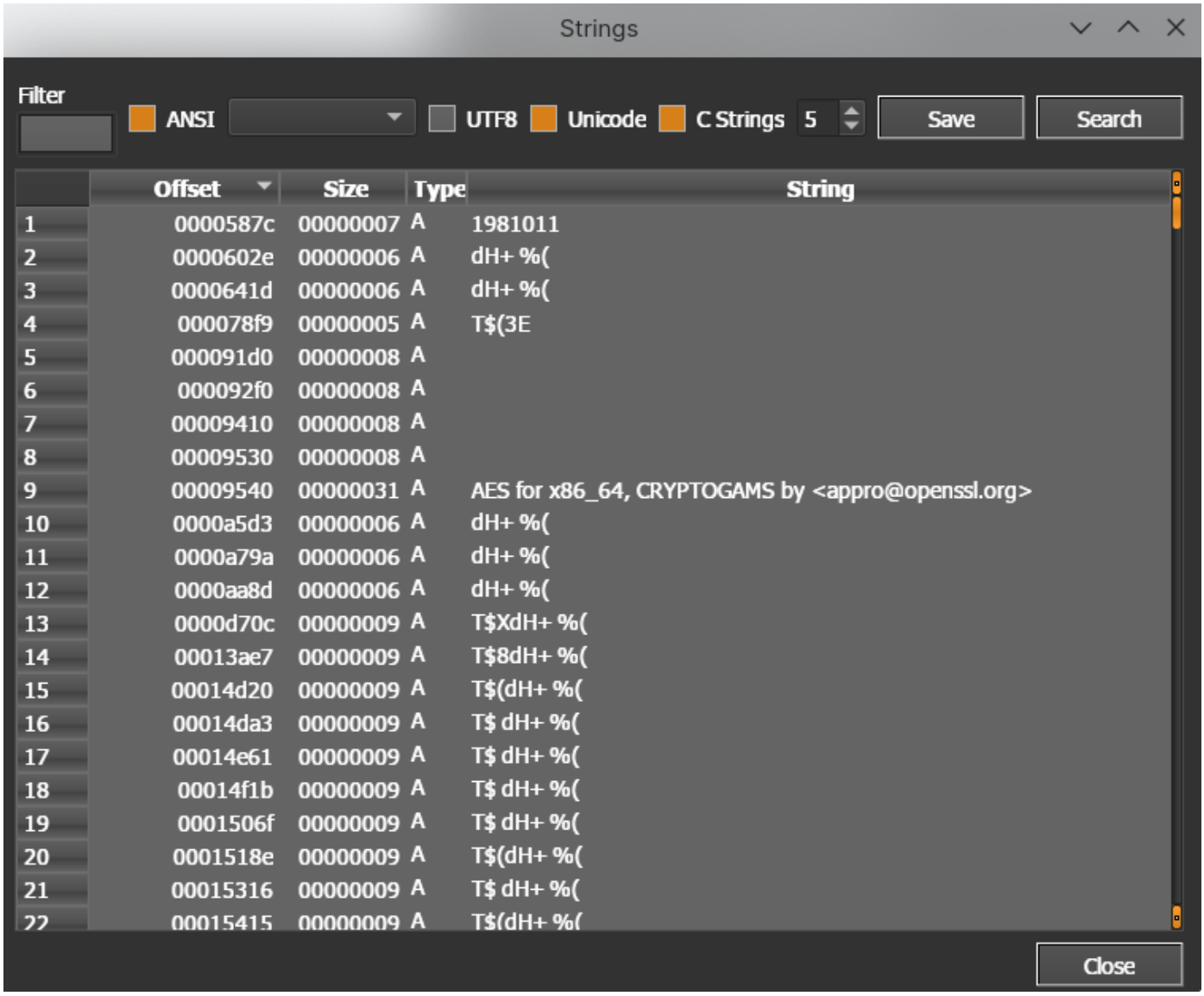
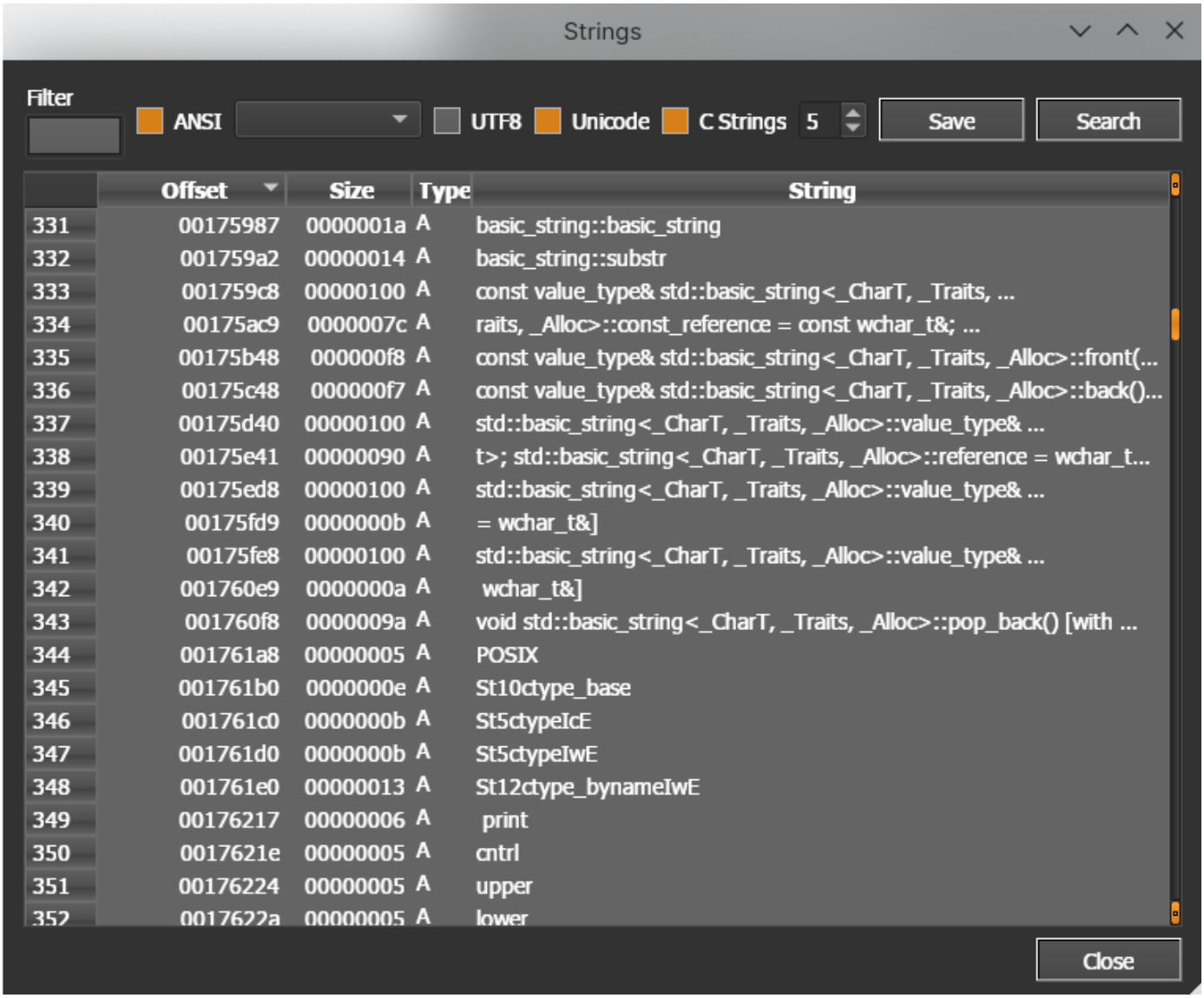
1. 用file命令查看文件

```
$ file eMecarT
eMecarT: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, BuildID[sha1]=608ec34aea51
a751eb80a14c6b24163bd2ba9ecf, for GNU/Linux 4.4.0, stripped
```

2. 用die分析文件，查看文件内字符串



发现openssl中的aes相关字符串



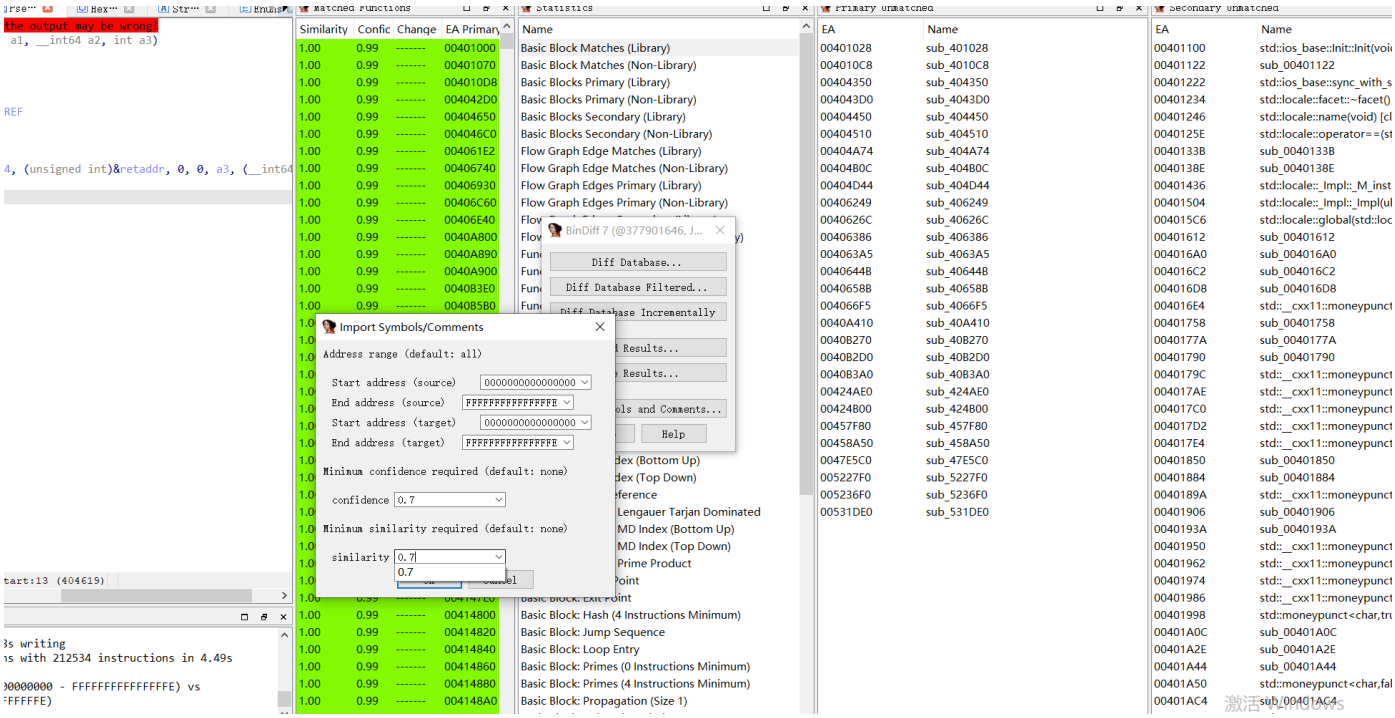
发现c++相关字符串

3. 通过ida的bindiff插件恢复符号

编写带有C++库和openssl库的文件

```cpp
#include <string>
#include <iostream>
#include <openssl/aes.h>
#include <stdio.h>
using namespace std;
void aes(const unsigned char* in,unsigned char *out,size_t length,const AES_KEY *key,unsigned char *ivec,const int enc){
    AES_cbc_encrypt(in,out,length,key,ivec,enc);
}
int main(){
    cout << "hello";
    printf("hello\n");
}
```

静态编译之后用ida打开，保存idb。用ida的bindiff插件比较题目的idb和保存的idb，将Import Symbols/Comments中的confidence和similarity设置为0.7



点击ok之后恢复部分符号

4. 分析逻辑

程序先读取一个字符串输入和一个数字输入，并将其转化为二进制字符串

```
  sub_40ACB0(&v7);
  _get_child_max(v16, "12345678901234561234567890123456", &v7);
  sub_40ACD0(&v7);
  strcpy(v28, "114514191981011");
  qmemcpy(v29, "0000000000000000", 16);
  v20[0] = 0x840AFF76F4F1CCBELL;
  v20[1] = 0x1B083D71A351AC45LL;
  v20[2] = 0x69CF04C27D1399DALL;
  v20[3] = 0xB6BE2817D3D5EF90LL;
  v20[4] = 0x7F3ABD64FEB9C098LL;
  v20[5] = 0x840AFF76F4F1CDEFLL;
  v20[6] = 0x840AFF76F4F1CCBFLL;
  v20[7] = 0x3E73E9FD8980088FLL;
  v20[8] = 0x172027BB1C2BC602LL;
  v20[9] = 0x8F90BC5D9EB292DFLL;
  v20[10] = 0xF1B278478FA40BD9LL;
  v20[11] = 0xCEF7E011F14C1F16LL;
  v20[12] = 0xB624A77659D338EELL;
  v20[13] = 0xDBA645049979E494LL;
  v20[14] = 0x733F2A4FC68A41D8LL;
  v20[15] = 0xAA1AF3F874687D6FLL;
  v20[16] = 0xE94DD34D794EB328LL;
  v20[17] = 0x67660C4D5C776B7DLL;
  v20[18] = 0x62CC0BC98430941DLL;
  v20[19] = 0x8E5B2123B36EAF6DLL;
  v20[20] = 0x8C278CD7CDB53F46LL;
  v20[21] = 0x4A72CB62B39CA3F7LL;
  v20[22] = 0x8119D366E317E644LL;
  v20[23] = 0xC79D8779FDBEA99ALL;
  v20[24] = 0x64D46DD5C7CD2EF5LL;
  v20[25] = 0x8C0E6750C44282B1LL;
  v20[26] = 0xE0F88A1CC54DF68ALL;
  v20[27] = 0xB69BD60F0D5AC4D9LL;
  stringinit((__int64)input);
  v0 = ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKa(&cout, "input a string:");
  sub_476160(v0, ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
  sub_40B920(&cin, input);
  v1 = ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKa(&cout, "input a number:");
  sub_476160(v1, ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
  ZNSt13basic_istreamIwSt11char_traitsIwEERsERi(&cin, &num);
  stringinit((__int64)bin_input);
  input_tmp = string2char((__int64)input);
  tobinstring((__int64)bin_input_tmp, input_tmp);
  ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEaSEOS4_(bin_input, bin_input_tmp);
  ZNSt12__sso_stringD2Ev(bin_input_tmp);
```

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
__int64 __fastcall tobinstring(__int64 a1, __int64 a2)
{
  char v3; // [rsp+16h] [rbp-1Ah]
  char v4; // [rsp+17h] [rbp-19h]
  int i; // [rsp+18h] [rbp-18h]
  int j; // [rsp+1Ch] [rbp-14h]

  stringinit(a1);
  v4 = j_strlen_ifunc(a2);
  for ( i = 0; i < v4; ++i )
  {
    v3 = *(_BYTE *)(i + a2);
    for ( j = 0; j <= 7; ++j )
    {
      if ( v3 >= 0 )
        ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEpLEPKc(a1, "0");
      else
        ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEpLEPKc(a1, "1");
      v3 *= 2;
    }
  }
  return a1;
}
```

之后fork()出一个子进程，在父进程里等待子进程，对输入进行处理，并杀死子进程

```
if ( pid )
{
  bin_input_char = (_DWORD *)string2char((__int64)bin_input);
  bin_input_char_copy = bin_input_char;
  wait(pid, &v7, 0);
  bin_input_char_copy = sub_404AA8(bin_input_char_copy, num);
  ptrace(0x4200LL, pid, 0LL, 0x100000LL);     // ptrace(PTRACE_SETOPTIONS, pid, 0, PTRACE_O_EXITKILL);
```

可以看出对输入进行处理的函数为xxtea，输入的数字是待加密数据中的dword个数（输入长度除以4），key为字符串“itisabeautyfulda”

```
{
  _QWORD *v2; // rax
  __int64 v3; // rax
  unsigned int *v4; // rax
  unsigned int *v5; // rax
  _DWORD *v7; // [rsp+8h] [rbp-38h]
  unsigned int v8; // [rsp+10h] [rbp-30h]
  unsigned int v9; // [rsp+14h] [rbp-2Ch]
  unsigned int i; // [rsp+18h] [rbp-28h]
  int v11; // [rsp+1Ch] [rbp-24h]
  int v12; // [rsp+20h] [rbp-20h]
  unsigned int v13; // [rsp+24h] [rbp-1Ch]
  __int64 v14; // [rsp+30h] [rbp-10h]

  v7 = a1;
  if ( !a2 )
  {
    v2 = (_QWORD *)sub_409630(8LL);
    *v2 = "[1]Fatal error!";
    sub_40A6A0(v2, &off_5D1038, 0LL);
  }
  if ( j_strlen_ifunc(a1) < (unsigned __int64)(4 * a2) )
  {
    v14 = malloc(4 * a2);
    j_memset_ifunc(v14, 0LL, 4 * a2);
    v3 = j_strlen_ifunc(a1);
    j_memcpy(v14, a1, v3);
    v7 = (_DWORD *)v14;
  }
  v11 = 52 / a2 + 6;
  v9 = 0;
  v8 = v7[a2 - 1];
  do
  {
    v9 -= 1640531527;
    v12 = (v9 >> 2) & 3;
    for ( i = 0; i < a2 - 1; ++i )
    {
      v13 = v7[i + 1];
      v4 = &v7[i];
      *v4 += ((v13 ^ v9) + (v8 ^ *(_DWORD *)&aItisabeautyful[4 * (v12 ^ i & 3)])) ^ (((4 * v13) ^ (v8 >> 5))
                                                                                   + ((v13 >> 3) ^ (16 * v8)));
      v8 = *v4;
    }
    v5 = &v7[a2 - 1];
    *v5 += ((*v7 ^ v9) + (v8 ^ *(_DWORD *)&aItisabeautyful[4 * (v12 ^ i & 3)])) ^ (((4 * *v7) ^ (v8 >> 5))
```

如过输入长度为232就把加密后的输入与一组数据进行比较，两组数据一致就输出字符`:)`。

用xxtea脚本解密比对的数据

```
#include <stdio.h>
#include <stdint.h>
#define DELTA 0x9e3779b9
unsigned char k[17]="itisabeautyfulda";
```

```
void btea(uint32_t *v, int n)
{
    uint32_t y, z, sum;
    unsigned p, rounds, e;
    unsigned int *key=(unsigned int *)k;
    if (n > 1)            /* Coding Part */
    {
        rounds = 6 + 52/n;
        sum = 0;
        z = v[n-1];
        do
        {
            sum += DELTA;
            e = (sum >> 2) & 3;
            for (p=0; p<n-1; p++)
            {
                y = v[p+1];
                z = v[p] += (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^
z)));
            }
            y = v[0];
            z = v[n-1] += (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^
z)));
        }
        while (--rounds);
    }
    else if (n < -1)      /* Decoding Part */
    {
        n = -n;
        rounds = 6 + 52/n;
        sum = rounds*DELTA;
        y = v[0];
        do
        {
            e = (sum >> 2) & 3;
            for (p=n-1; p>0; p--)
            {
                z = v[p-1];
                y = v[p] -= (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^
z)));
            }
            z = v[n-1];
            y = v[0] -= (((z>>5^y<<2) + (y>>3^z<<4)) ^ ((sum^y) + (key[(p&3)^e] ^ z)));
            sum -= DELTA;
        }
        while (--rounds);
    }
}
```

```
int main()
{
    int n= 58;
    uint32_t v[]=
{0x6ac68723,0xbc7123a7,0x64f3d87d,0x3e8c216,0x737dd747,0xb010868f,0x33030511,0x83453d34
,0x508e9921,0x2bfa017a,0x824aba3e,0xb426bc88,0xe2b6bfbc,0x10e2caf7,0x4fc41d21,0x67c588f
0,0xdbd13516,0x67cb17db,0x54e01fbd,0xc5b682d5,0xdccbe585,0xc51ec321,0xbb7cc296,0x158cb0
e8,0x6946bfd7,0xa70879ac,0x6b1b0108,0x6996f44e,0x37b754e7,0xb3a7607,0x62c425d4,0x34f5f4
09,0xc2c0d008,0xe8ed6971,0xa7c57884,0x8563eaa5,0x825dea33,0xb0605bb3,0x63319346,0x21771
47b,0x689899a1,0x7ba4a9cd,0x803c46d2,0xcffedda6,0xefb452d4,0x74e3dde,0xfb146cf2,0x1c014
40b,0xd3f5bccc,0x6f8a379e,0x7e46972d,0x64749b9b,0xeceacb7e,0x780fe61b,0xbcaf308e,0x641f
40b,0x8f0897bd,0xd5b3c7db};
    btea(v, -n);
    printf("\ndecrypted:\n");
    for(int i=0;i<232;i++){
        printf("%c",((char*)v)[i]);
    }
    return 0;
}
```

```
decrypted:
01100110011011000110000101100111011110110111001100101011101100110010101110010010111110011011011110110111001101110011000010101111101100111011010010111011001100101010111110111001
01101111011101010101111101110101011100000111110100
```

用python将二进制编码转为字符串

```
data="01100110011011000110000101100111011110110111001100101011101100110010101110010010111110011011011110110111001101110011000010101111101100111011010010111011001100101010111110111001
01101111011101010101111101110101011100000111110100"
num=len(data)//8
for i in range(num):
    tmp=data[i*8:i*8+8]
    tmp=int(tmp,2)
    tmp=chr(tmp)
    print(tmp,end='')
```

```
❯ /bin/python /home/lotke/Desktop/CTF/WorkSpace/test.py
flag{never_gonna_give_you_up}%
```

得到了假的flag，所以真正的判断逻辑应该在子进程，查看子进程的逻辑

```
  ptrace(0LL, 0LL, 0LL, 0LL);                   // ptrace(PTRACE_TRACEME,0,0,0);
  raise(17LL);                                  // raise(SIGCHLD);
  aes_cbc((__int64)resstring, (__int64)v19, bin_input, v30);
  reschar = (_QWORD *)string2char((__int64)resstring);
  *(_QWORD *)data2 = 0x4C455543525A5E72LL;
  *(_DWORD *)&data2[8] = 0x584C585D;
  *(_WORD *)&data2[12] = 0x5F5F;
  v28 = 0x10;
  *(_DWORD *)data1 = 0x595D5057;
  v26 = 0x55;
  for ( j = 0; j <= 27; ++j )
  {
    if ( reschar[j] != v23[j] )
    {
      for ( k = 0; k <= 3; ++k )
        output((unsigned __int8)(xorkey[k] ^ data1[k]));// 输出字符串'fail'
      sub_476160(&cout, ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
      exit(0LL);
    }
  }
  for ( m = 0; m <= 13; ++m )
    output((unsigned __int8)(xorkey[m] ^ data2[m]));// 输出字符串'congratulation'
  sub_476160(&cout, ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_);
  ZNSt12__sso_stringD2Ev(resstring);
  ZNSt12__sso_stringD2Ev(bin_input);
  ZNSt12__sso_stringD2Ev(input);
  ZNSt12__sso_stringD2Ev(v19);
  return 0;
}
```

首先子进程发送PTRACE_TRACEME的信号让父进程进行跟踪，然后向父进程发送SIGCHLD信号并暂停自身的执行。如父进程发送继续执行的信号，子进程会对输入进行CBC模式的AES加密，并与已有的数据进行比较。根据结果输出字符串'fail'或者字符串‘congratulation’（字符串数据进行过异或操作，所以要通过异或解回来，可以通过调试判断）。

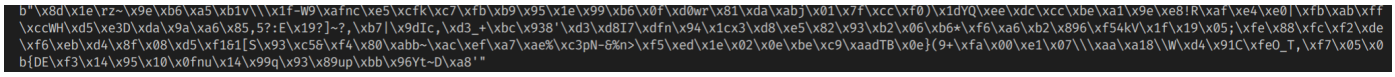
AES的key和iv、用来比较的数据在main函数中都能找到明文

```
sub_4059B0(key, "12345678901234561234567890123456", &v10);
sub_40ACD0(&v10);
strcpy(xorkey, "114514191981011");
qmemcpy(iv, "0000000000000000", 16);
finaldata[0] = 0x840AFF76F4F1CCBELL;
finaldata[1] = 0x1B083D71A351AC45LL;
finaldata[2] = 0x69CF04C27D1399DALL;
finaldata[3] = 0xB6BE2817D3D5EF90LL;
finaldata[4] = 0x7F3ABD64FEB9C098LL;
finaldata[5] = 0x840AFF76F4F1CDEFLL;
finaldata[6] = 0x840AFF76F4F1CCBFLL;
finaldata[7] = 0x3E73E9FD8980088FLL;
finaldata[8] = 0x172027BB1C2BC602LL;
finaldata[9] = 0x8F90BC5D9EB292DFLL;
finaldata[10] = 0xF1B278478FA40BD9LL;
finaldata[11] = 0xCEF7E011F14C1F16LL;
finaldata[12] = 0xB624A77659D338EELL;
finaldata[13] = 0xDBA645049979E494LL;
finaldata[14] = 0x733F2A4FC68A41D8LL;
finaldata[15] = 0xAA1AF3F874687D6FLL;
finaldata[16] = 0xE94DD34D794EB328LL;
finaldata[17] = 0x67660C4D5C776B7DLL;
finaldata[18] = 0x62CC0BC98430941DLL;
finaldata[19] = 0x8E5B2123B36EAF6DLL;
finaldata[20] = 0x8C278CD7CDB53F46LL;
finaldata[21] = 0x4A72CB62B39CA3F7LL;
finaldata[22] = 0x8119D366E317E644LL;
finaldata[23] = 0xC79D8779FDBEA99ALL;
finaldata[24] = 0x64D46DD5C7CD2EF5LL;
finaldata[25] = 0x8C0E6750C44282B1LL;
finaldata[26] = 0xE0F88A1CC54DF68ALL;
finaldata[27] = 0xB69BD60F0D5AC4D9LL;
```

尝试直接解密，解出来结果为不可见字符，显然不正确。

```
from Crypto.Cipher import AES
from pwn import *
key=b"12345678901234561234567890123456"
value=['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0']
iv=b'\x00\x00\x81\x19\x19\x14\x45\x11\x30\x30\x30\x30\x30\x30\x30\x30'
en_data=[b'\xbe\xcc\xf1\xf4v\xff\n\x84E\xacQ\xa3q=\x08\x1b',
b'\xda\x99\x13}\xc2\x04\xcfi\x90\xef\xd5\xd3\x17(\xbe\xb6',
b'\x98\xc0\xb9\xfed\xbd:\x7f\xef\xcd\xf1\xf4v\xff\n\x84',
b'\xbf\xcc\xf1\xf4v\xff\n\x84\x8f\x08\x80\x89\xfd\xe9s>',
b"\x02\xc6+\x1c\xbb' \x17\xdf\x92\xb2\x9e]\xbc\x90\x8f",
b'\xd9\x0b\xa4\x8fGx\xb2\xf1\x16\x1fL\xf1\x11\xe0\xf7\xce',
b'\xee8\xd3Yv\xa7$\xb6\x94\xe4y\x99\x04E\xa6\xdb',
b'\xd8A\x8a\xc6O*?so}ht\xf8\xf3\x1a\xaa',
b'(\xb3NyM\xd3M\xe9}kw\\M\x0cfg',
b'\x1d\x940\x84\xc9\x0b\xccbm\xafn\xb3#![\x8e',
```

```
b"F?\xb5\xcd\xd7\x8c'\x8c\xf7\xa3\x9c\xb3b\xcbrJ",
b'D\xe6\x17\xe3f\xd3\x19\x81\x9a\xa9\xbe\xfdy\x87\x9d\xc7',
b'\xf5.\xcd\xc7\xd5m\xd4d\xb1\x82B\xc4Pg\x0e\x8c',
b'\x8a\xf6M\xc5\x1c\x8a\xf8\xe0\xd9\xc4Z\r\x0f\xd6\x9b\xb6',]
aes=AES.new(key,AES.MODE_CBC,iv)
data=b''
for i in range(14):
    data+=aes.decrypt(en_data[i])
print(data)
```



可以猜测是父进程对子进程数据进行了修改，需要调试父进程。将fork函数patch掉，并将变量pid的值改为1即可调试父进程。

```
.text:0000000000405133                call    _ZNSt12__sso_stringD2Ev
.text:0000000000405138                mov     eax, 1          ; Keypatch modified this from:
.text:0000000000405138                                        ;   call fork
.text:0000000000405138                                        ; Keypatch padded NOP to next boundary: 4 bytes
.text:0000000000405138                                        ; Keypatch modified this from:
.text:0000000000405138                                        ;   nop
.text:0000000000405138                                        ;   nop
.text:0000000000405138                                        ;   nop
.text:0000000000405138                                        ;   nop
.text:0000000000405138                                        ;   nop
.text:0000000000405138                                        ; Keypatch modified this from:
.text:0000000000405138                                        ;   mov eax, 0
.text:000000000040513D                mov     [rbp+pid], eax
```

在父进程的xxtea处理函数里，如果num为0会进行特殊处理，反编译处理函数可以看出来是throw了一个exception，打上断点进行调试

```
if ( !num )
{
  v2 = (_QWORD *)sub_409630(8LL);
  *v2 = "[1]Fatal error!";
  sub_40A6A0((__int64)v2, (__int64)&off_5D1038, 0LL);
}
```

```
void __fastcall __noreturn sub_40A6A0(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 globals; // rax
  _DWORD *inited; // rax
  _DWORD *v6; // rbp
  int v7; // edx
  int v8; // ecx
  int v9; // er8
  int v10; // er9

  globals = _cxa_get_globals();
  ++*(_DWORD *)(globals + 8);
  inited = (_DWORD *)_cxa_init_primary_exception(a1, a2, a3);
  v6 = inited + 24;
  *inited = 1;
  Unwind_RaiseException((_DWORD)inited + 96, a2, v7, v8, v9, v10);
  sub_4097A0(v6);
  sub_40A570();
  sub_40A6F0();
}
```

最后可以跟踪到catch的逻辑，先用ptrace的PTRACE_POKEDATA请求修改子进程里的数据，最后用ptrace的PTRACE_CONT请求使子进程继续运行。

```
.text:000000000040570F mov     rdi, rax
.text:0000000000405712 call    sub_4097A0
.text:0000000000405717 lea     rdx, [rbp+iv]   ; a3
.text:000000000040571B mov     eax, [rbp+pid]
.text:0000000000405721 mov     rcx, 1145141919810000h ; a4
.text:000000000040572B mov     esi, eax        ; a2
.text:000000000040572D mov     edi, 5          ; a1
.text:0000000000405732 mov     eax, 0
.text:0000000000405737 call    ptrace          ; PTRACE_POKEDATA
.text:000000000040573C lea     rax, [rbp+key]
.text:0000000000405743 mov     esi, 0
.text:0000000000405748 mov     rdi, rax
.text:000000000040574B call    sub_47E460
.text:0000000000405750 mov     rsi, rax
.text:0000000000405753 mov     eax, [rbp+pid]
.text:0000000000405759 mov     rdx, 6572617364726962h
.text:0000000000405763 mov     rcx, rdx        ; a4
.text:0000000000405766 mov     rdx, rsi        ; a3
.text:0000000000405769 mov     esi, eax        ; a2
.text:000000000040576B mov     edi, 5          ; a1
.text:0000000000405770 mov     eax, 0
.text:0000000000405775 call    ptrace          ; PTRACE_POKEDATA
.text:000000000040577A lea     rdx, [rbp+finaldata] ; a3
```

```
.text:0000000000405781 mov     eax, [rbp+pid]
.text:0000000000405787 mov     rcx, 5E873CDC9EBE8011h ; a4
.text:0000000000405791 mov     esi, eax        ; a2
.text:0000000000405793 mov     edi, 5          ; a1
.text:0000000000405798 mov     eax, 0
.text:000000000040579D call    ptrace          ; PTRACE_POKEDATA
.text:00000000004057A2 lea     rax, [rbp+finaldata]
.text:00000000004057A9 lea     rdx, [rax+28h]  ; a3
.text:00000000004057AD mov     eax, [rbp+pid]
.text:00000000004057B3 mov     rcx, 0BC74073357E6A5DBh ; a4
.text:00000000004057BD mov     esi, eax        ; a2
.text:00000000004057BF mov     edi, 5          ; a1
.text:00000000004057C4 mov     eax, 0
.text:00000000004057C9 call    ptrace          ; PTRACE_POKEDATA
.text:00000000004057CE mov     eax, [rbp+pid]
.text:00000000004057D4 mov     ecx, 0          ; a4
.text:00000000004057D9 mov     edx, 0          ; a3
.text:00000000004057DE mov     esi, eax        ; a2
.text:00000000004057E0 mov     edi, 7          ; a1
.text:00000000004057E5 mov     eax, 0
.text:00000000004057EA call    ptrace          ; PTRACE_CONT
.text:00000000004057EF mov     ebx, 0
.text:00000000004057F4 call    __cxa_end_catch
.text:00000000004057F9 jmp     loc_4056AB
```

自行修改key、iv和加密后的数据为正确的值之后进行解密，得出正确的flag

```
from Crypto.Cipher import AES
from pwn import *
key=b"birdsare901234561234567890123456"
value=[0 , 0, 0x81, 0x19, 0x19, 0x14, 0x45, 0x11, '0', '0', '0', '0', '0', '0', '0',
'0']
iv=b'\x00\x00\x81\x19\x19\x14\x45\x11\x30\x30\x30\x30\x30\x30\x30\x30'
en_data=[
b"\x11\x80\xbe\x9e\xdc\x3c\x87\x5e\x45\xac\x51\xa3\x71\x3d\x08\x1b",
b"\xda\x99\x13\x7d\xc2\x04\xcf\x69\x90\xef\xd5\xd3\x17\x28\xbe\xb6",
b"\x98\xc0\xb9\xfe\x64\xbd\x3a\x7f\xdb\xa5\xe6\x57\x33\x07\x74\xbc",
b"\xbf\xcc\xf1\xf4\x76\xff\x0a\x84\x8f\x08\x80\x89\xfd\xe9\x73\x3e",
b"\x02\xc6\x2b\x1c\xbb\x27\x20\x17\xdf\x92\xb2\x9e\x5d\xbc\x90\x8f",
b"\xd9\x0b\xa4\x8f\x47\x78\xb2\xf1\x16\x1f\x4c\xf1\x11\xe0\xf7\xce",
b"\xee\x38\xd3\x59\x76\xa7\x24\xb6\x94\xe4\x79\x99\x04\x45\xa6\xdb",
b"\xd8\x41\x8a\xc6\x4f\x2a\x3f\x73\x6f\x7d\x68\x74\xf8\xf3\x1a\xaa",
b"\x28\xb3\x4e\x79\x4d\xd3\x4d\xe9\x7d\x6b\x77\x5c\x4d\x0c\x66\x67",
b"\x1d\x94\x30\x84\xc9\x0b\xcc\x62\x6d\xaf\x6e\xb3\x23\x21\x5b\x8e",
b"\x46\x3f\xb5\xcd\xd7\x8c\x27\x8c\xf7\xa3\x9c\xb3\x62\xcb\x72\x4a",
b"\x44\xe6\x17\xe3\x66\xd3\x19\x81\x9a\xa9\xbe\xfd\x79\x87\x9d\xc7",
b"\xf5\x2e\xcd\xc7\xd5\x6d\xd4\x64\xb1\x82\x42\xc4\x50\x67\x0e\x8c",
b"\x8a\xf6\x4d\xc5\x1c\x8a\xf8\xe0\xd9\xc4\x5a\x0d\x0f\xd6\x9b\xb6"]
```

```
aes=AES.new(key,AES.MODE_CBC,iv)
data=''
for i in range(14):
    data+=aes.decrypt(en_data[i]).decode()
num=len(data)//8
for i in range(num):
    tmp=data[i*8:i*8+8]
    tmp=int(tmp,2)
    tmp=chr(tmp)
    print(tmp,end='')
```