i春秋训练营 2017全国大学生网络安全竞赛 欢迎来到加基森 wp

原创

n00bzx 于 2021-08-23 21:00:04 发布 602 收藏

本文链接：https://blog.csdn.net/qq_35623926/article/details/119877237

这题磨了一周。。。做完我要认真准备高考了,学渣一个。。。

题目如图...

打开,32位elf文件,定位到关键函数

```
memset(v3, 0, sizeof(v3));
v4 = 0;
sub_8051000("%16s", v3);
if ( _strlen_ia32(v3) == 16 )
{
  sub_804A310(v3);
  sub_804A5A0(v3);
}
else
{
  sub_80515D0("Key length error!");
```

key长度要为16.

加密函数为第三个,第二个是密钥扩展.

```
v2 = a1[1];
v8[31] = -72;
dword_80EFB24 = 8;
dword_80EFB28 = 14;
v6[1] = v2;
v6[2] = a1[2];
v6[3] = a1[3];
```

```
v3 = sub_805B3F0(60 * dword_80EE084);
sub_8049F00(v8, v3);
sub_804A1A0(v6, v7, v3);
*a1 = v7[0];
a1[1] = v7[1];
a1[2] = v7[2];
v4 = __readgsdword(0x14u) ^ v9;
a1[3] = v7[3];
if ( v4 )
  (sub_8071890)(v4);
return 0;
```


这是第一步加密,第一步加密是一个变异的aes,变异了三个部分,分别是s盒内容,查表调换索引的高低四位,列混合的混淆矩阵.还有每一轮的最后,使用一个含有smc的vm加密一次.vm分析如下.

```
//省略循环判断结束的opcode
65 0 smc_dword((i1),heap[0])
7e 3 (i1) heap[3]=str[i1]
65 1 smc_dword((i2),heap[1])
7e 4 (16+i2) heap[4]=str[i2+16]
64 3 4 heap[3]^=heap[4]
65 0 smc_dword((i3),heap[0])
77 3 (i3) str[i3]=heap[3]
d9 15 unk=15
bf 3 heap[3]=unk;unk+=16
d9 14 unk=14
bf 4 heap[4]=unk;unk+=16
79 3 0 heap[3]-=heap[0]
79 4 0 heap[4]-=heap[0]//heap3=15-i,heap4=14-i
65 3 smc_dword((i4),heap[3])
7e 3 (i4) heap[3]=str[i4]//i4=15-i
65 4 smc_dword((i5),heap[4])
7e 5 (i5) heap[5]=str[i5]//i5=14-i
64 5 3 heap[5]^=heap[3]//str[14-i]^=str[15-i]
65 4 smc_dword((i6),heap[4])
77 5 (i6) str[i6]=heap[5]//str[14-i]^=str[15-i]
0 0 ++heap[0]
0 1 ++heap[1]
a0 170 循环结束跳转
```

加密代码如下.解密也写在一起了.解密里要变的是逆s盒,逆查表,列混合逆矩阵,以及逆vm.
(此处aes是根据网上大神的代码改的)

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#define BLOCKSIZE 16  //AES-128分组长度为16字节

// uint8_t y[4] -> uint32_t x
#define LOAD32H(x, y) \
  do { (x) = ((uint32_t)((y)[0] & 0xff)<<24) | ((uint32_t)((y)[1] & 0xff)<<16) | \
             ((uint32_t)((y)[2] & 0xff)<<8)  | ((uint32_t)((y)[3] & 0xff));} while(0)

// uint32_t x -> uint8_t y[4]
#define STORE32H(x, y) \
  do { (y)[0] = (uint8_t)(((x)>>24) & 0xff); (y)[1] = (uint8_t)(((x)>>16) & 0xff);   \
       (y)[2] = (uint8_t)(((x)>>8) & 0xff); (y)[3] = (uint8_t)((x) & 0xff); } while(0)
```

```
// 从uint32_t x中提取从低位开始的第n个字节
#define BYTE(x, n) (((x) >> (8 * (n))) & 0xff)

/* used for keyExpansion */
// 字节替换然后循环左移1位
#define MIX(x) (((S[BYTE(x, 2)] << 24) & 0xff000000) ^ ((S[BYTE(x, 1)] << 16) & 0xff0000) ^ \
               ((S[BYTE(x, 0)] << 8) & 0xff00) ^ (S[BYTE(x, 3)] & 0xff))

// uint32_t x循环左移n位
#define ROF32(x, n)  (((x) << (n)) | ((x) >> (32-(n))))
// uint32_t x循环右移n位
#define ROR32(x, n)  (((x) >> (n)) | ((x) << (32-(n))))
// S盒
unsigned char S[256] = {
        54, 79, 98, 216, 181, 132, 205, 246, 220, 42, 230, 237, 171, 82, 1, 175, 208, 10, 104, 20, 39, 161, 219,
 135, 156, 231, 41, 102, 53, 233, 180, 145, 139, 206, 243, 52, 86, 94, 35, 97, 112, 195, 167, 50, 45, 128, 12, 1
96, 245, 44, 114, 164, 201, 6, 185, 110, 25, 18, 7, 34, 214, 211, 0, 113, 152, 14, 107, 202, 100, 61, 186, 254,
136, 2, 227, 70, 239, 31, 47, 143, 46, 58, 49, 155, 240, 226, 123, 153, 187, 168, 72, 99, 212, 141, 244, 105, 19
1, 232, 75, 222, 218, 118, 176, 16, 184, 151, 198, 159, 22, 247, 172, 221, 69, 188, 197, 225, 173, 124, 91, 252,
 204, 174, 89, 154, 111, 229, 103, 163, 88, 199, 140, 101, 129, 73, 83, 131, 64, 57, 93, 55, 177, 116, 142, 77,
207, 51, 248, 162, 117, 115, 250, 215, 74, 203, 130, 120, 23, 224, 149, 125, 8, 241, 189, 65, 183, 4, 15, 138, 2
09, 81, 62, 56, 147, 194, 137, 210, 150, 148, 26, 251, 95, 126, 96, 36, 84, 234, 127, 11, 27, 67, 38, 43, 28, 17
9, 78, 133, 63, 90, 192, 30, 80, 119, 255, 9, 228, 158, 37, 157, 5, 19, 238, 213, 48, 170, 40, 121, 134, 76, 249
, 87, 109, 235, 71, 178, 217, 24, 182, 190, 122, 193, 160, 13, 92, 200, 165, 68, 106, 60, 32, 166, 3, 17, 59, 21
, 144, 253, 146, 236, 169, 33, 85, 29, 223, 108, 66, 242
};

//逆S盒
unsigned char inv_S[256] = {
        62, 14, 73, 240, 165, 208, 53, 58, 160, 203, 17, 187, 46, 231, 65, 166, 103, 241, 57, 209, 19, 243, 108,
 156, 225, 56, 178, 188, 192, 251, 199, 77, 238, 249, 59, 38, 183, 206, 190, 20, 214, 26, 9, 191, 49, 44, 80, 78
, 212, 82, 43, 145, 35, 28, 0, 139, 171, 137, 81, 242, 237, 69, 170, 196, 136, 163, 254, 189, 235, 112, 75, 222,
 90, 133, 152, 98, 217, 143, 194, 1, 200, 169, 13, 134, 184, 250, 36, 219, 128, 122, 197, 118, 232, 138, 37, 180
, 182, 39, 2, 91, 68, 131, 27, 126, 18, 95, 236, 66, 253, 220, 55, 124, 40, 63, 50, 149, 141, 148, 101, 201, 155
, 215, 228, 86, 117, 159, 181, 186, 45, 132, 154, 135, 5, 195, 216, 23, 72, 174, 167, 32, 130, 93, 142, 79, 244,
 31, 246, 172, 177, 158, 176, 105, 64, 87, 123, 83, 24, 207, 205, 107, 230, 21, 147, 127, 51, 234, 239, 42, 89,
248, 213, 12, 110, 116, 121, 15, 102, 140, 223, 193, 30, 4, 226, 164, 104, 54, 70, 88, 113, 162, 227, 96, 198, 2
29, 173, 41, 47, 114, 106, 129, 233, 52, 67, 153, 120, 6, 33, 144, 16, 168, 175, 61, 92, 211, 60, 151, 3, 224, 1
00, 22, 8, 111, 99, 252, 157, 115, 85, 74, 204, 125, 10, 25, 97, 29, 185, 221, 247, 11, 210, 76, 84, 161, 255, 3
4, 94, 48, 7, 109, 146, 218, 150, 179, 119, 245, 71, 202
};

/* copy in[16] to state[4][4] */
int loadStateArray(uint8_t (*state)[4], const uint8_t *in) {
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            state[j][i] = *in++;
        }
    }
    return 0;
}

/* copy state[4][4] to out[16] */
int storeStateArray(uint8_t (*state)[4], uint8_t *out) {
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            *out++ = state[i][j];//[j][i]
        }
    }
    return 0;
```

```
    return 0;
}
// 轮秘钥加
int addRoundKey(uint8_t (*state)[4], unsigned char *key) {
    uint8_t k[4][4];

    /* i: row, j: col */
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            k[i][j] = key[j*4+i];  /* 把 uint32 key[4] 先转换为矩阵 uint8 k[4][4] */
            state[i][j] ^= k[i][j];
        }
    }

    return 0;
}

//字节替换
int subBytes(uint8_t (*state)[4]) {
    /* i: row, j: col */
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            state[i][j] = S[((state[i][j]&0xf)<<4)|(state[i][j]>>4)]; //直接使用原始字节作为S盒数据下标
        }
    }

    return 0;
}

//逆字节替换
int invSubBytes(uint8_t (*state)[4]) {
    /* i: row, j: col */
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
         uint8_t tmp=inv_S[state[i][j]];
            state[i][j] = ((tmp&0xf)<<4)|(tmp>>4);
        }
    }
    return 0;
}

//行移位
int shiftRows(uint8_t (*state)[4]) {
    uint32_t block[4] = {0};

    /* i: row */
    for (int i = 0; i < 4; ++i) {
    //便于行循环移位，先把一行4字节拼成uint_32结构，移位后再转成独立的4个字节uint8_t
        LOAD32H(block[i], state[i]);
        block[i] = ROF32(block[i], 8*i);
        STORE32H(block[i], state[i]);
    }

    return 0;
}

//逆行移位
int invShiftRows(uint8_t (*state)[4]) {
    uint32_t block[4] = {0};
```

```
    /* i: row */
    for (int i = 0; i < 4; ++i) {
        LOAD32H(block[i], state[i]);
        block[i] = ROR32(block[i], 8*i);
        STORE32H(block[i], state[i]);
    }

    return 0;
}

/* Galois Field (256) Multiplication of two Bytes */
// 两字节的伽罗华域乘法运算
uint8_t GMul(uint8_t u, uint8_t v) {
    uint8_t p = 0;

    for (int i = 0; i < 8; ++i) {
        if (u & 0x01) {    //
            p ^= v;
        }

        int flag = (v & 0x80);
        v <<= 1;
        if (flag) {
            v ^= 0x1B; /* x^8 + x^4 + x^3 + x + 1 */
        }

        u >>= 1;
    }

    return p;
}

// 列混合
int mixColumns(uint8_t (*state)[4]) {
    uint8_t tmp[4][4];
    uint8_t M[4][4] = {{0xd,0xd,0xf,0xe},{0xe,0xd,0xd,0xf},{0xf,0xe,0xd,0xd},{0xd,0xf,0xe,0xd}};

    /* copy state[4][4] to tmp[4][4] */
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j){
            tmp[i][j] = state[i][j];
        }
    }

    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {  //伽罗华域加法和乘法
            state[i][j] = GMul(M[i][0], tmp[0][j]) ^ GMul(M[i][1], tmp[1][j])
                        ^ GMul(M[i][2], tmp[2][j]) ^ GMul(M[i][3], tmp[3][j]);
        }
    }

    return 0;
}

int invMixColumns(uint8_t (*state)[4]) {
    uint8_t tmp[4][4];
    uint8_t M[4][4] ={{7,2,5,1},{1,7,2,5},{5,1,7,2},{2,5,1,7}};
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j){
```

```
            tmp[i][j] = state[i][j];
        }
    }

    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            state[i][j] = GMul(M[i][0], tmp[0][j]) ^ GMul(M[i][1], tmp[1][j])
                        ^ GMul(M[i][2], tmp[2][j]) ^ GMul(M[i][3], tmp[3][j]);
        }
    }

    return 0;
}

#include <stdio.h>
#include<windows.h>
void vm(uint8_t (*state)[4])
{
 BYTE pos[16]={0};
for(int i=0;i<4;i++)
{
 for(int j=0;j<4;j++)
 {

 pos[i*4+j]=state[i][j];
}
}
 const char* ichunqiu="ichunqiuichunqiu";
for(int i=0;i<16;i++)
{
    if(14-i>=0)
    {
        pos[i]^=(BYTE)ichunqiu[i];
       pos[14-i]^=pos[15-i];
    }
 }
 for(int i=0;i<4;i++)
{
 for(int j=0;j<4;j++)
 {

 state[i][j]=pos[i*4+j];
}
}


}
void invvm(uint8_t (*state)[4])
{
 BYTE pos[16]={0};
for(int i=0;i<4;i++)
{
 for(int j=0;j<4;j++)
 {

 pos[i*4+j]=state[i][j];
}
}
 const char* ichunqiu="ichunqiuichunqiu";
 pos[0]^=pos[1];
 pos[14]^=ichunqiu[14];
```

```
 pos[1]^=pos[2];
 pos[13]^=ichunqiu[13];
 pos[2]^=pos[3];
 pos[12]^=ichunqiu[12];
 pos[3]^=pos[4];
 pos[11]^=ichunqiu[11];
 pos[4]^=pos[5];
 pos[10]^=ichunqiu[10];
 pos[5]^=pos[6];
 pos[9]^=ichunqiu[9];
 pos[6]^=pos[7];
 pos[8]^=ichunqiu[8];
 pos[7]^=pos[8];
 pos[7]^=ichunqiu[7];
 pos[8]^=pos[9];
 pos[6]^=ichunqiu[6];
 pos[9]^=pos[10];
 pos[5]^=ichunqiu[5];
 pos[10]^=pos[11];
 pos[4]^=ichunqiu[4];
 pos[11]^=pos[12];
 pos[3]^=ichunqiu[3];
 pos[12]^=pos[13];
 pos[2]^=ichunqiu[2];
 pos[13]^=pos[14];
 pos[1]^=ichunqiu[1];
 pos[14]^=pos[15];
 pos[0]^=ichunqiu[0];
 for(int i=0;i<4;i++)
{
 for(int j=0;j<4;j++)
 {

 state[i][j]=pos[i*4+j];
}
}

}

// 方便输出16进制数据
void printHex(uint8_t *ptr, int len) {
    for (int i = 0; i < len; ++i) {
        printf("%.2X ", *ptr++);
    }
    printf("\n");
}
#include<iostream>
using namespace std;
int main() {
FILE *fphzk;
fphzk=fopen("./bin","rb");//扩展后密钥
BYTE eke2y[256];
BYTE* ekey=eke2y;
fread(ekey,1,240,fphzk);
uint8_t state[4][4] = {0};
uint8_t pos[17]={0};
ekey+=208;
    /*const uint8_t *data = (uint8_t*)"1122334455667788";
    loadStateArray(state, data);
```

```
        addRoundKey(state, ekey);

        for(int i=1;i<0xe;i++)
        {
  ekey+=16;
     subBytes(state);
            shiftRows(state);  // 行移位
             mixColumns(state);

             addRoundKey(state, ekey);
     vm(state);
         }
          subBytes(state);
            shiftRows(state);  // 行移位
             addRoundKey(state, ekey+16);
  // storeStateArray(state, pos);
  // printHex(pos, 16); */
    BYTE opp[]={202, 211, 229, 67, 242, 60, 63, 236, 51, 169, 63, 149, 221, 140, 76, 42};
      loadStateArray(state,opp);
    addRoundKey(state, ekey+16);
     invShiftRows(state);
     invSubBytes(state);
      for(int i=1;i<0xe;i++)
        {
  invvm(state);
  addRoundKey(state, ekey);
  invMixColumns(state);
   invShiftRows(state);
     invSubBytes(state);
     ekey-=16;
         }
         addRoundKey(state, ekey);
         storeStateArray(state, pos);
         for(int i=0;i<4;i++)
             for(int j=0;j<4;j++)
                 std::cout<<pos[j*4+i];
         //std::cout<<pos;
         //printHex(pos, 16);
    return 0;
}
```

说下列混淆逆矩阵的求法.
进入matlab,输入原矩阵,再输入x=gf(原矩阵,8),(8为GF2^8),再输入inv(x)即可获得逆矩阵.(高中生没学过大学数学。。。
第二段加密不可逆,所以用z3求解,代码如下:

```
from z3 import *
flen=16
def read_word(arr,index):
    return arr[index]|(arr[index+1]<<8)
solver=Solver()
flag=[BitVec('flag%d'%i,32) for i in range(flen)]
#flag=[0, 16, 65, 64, 0, 48, 1, 64, 0, 0, 16, 0, 192, 0, 0, 0]
#flag=[0xaf,0x42,0xb2,0x89,0x3b,0x65,0xea,0x67,0xa4,0x4e,5,0xcc,0xde,0xba,0x1f,0xea]
for j in range(flen):
    solver.add(flag[j]>=0x0)
    solver.add(flag[j]<=0xff)
```

```
v20=[0]*256
f=open("./Gadgetzan",'rb')
q=f.read()
f.close()
r=q[0x75e60:0x75e60+256]
g=q[0x75c42:0x75c60]
t=q[0x75c60:0x75c60+512]
v18=0
for i in range(16):
    a=flag[i]
    b=flag[15-i]
    for j in range(8):
        c=(a>>j)&1
        d=(b>>j)&1
        v20[r[v18*8+j]]=c
        v20[r[v18*8+j+128]]=d
    v18+=1
v8=0x8ebc
tocmp=0x86c1
k=0
v7=0
debug=100
for i in range(16):
    v10=0
    j=0
    for j_ in range(0,32,2):
        z=v20[j_*8+k]
        if i==debug:
            print(hex(z))
            print(hex(v8))
        j+=2
        v10^=z*v8
        if j==32:
            break
        v8=read_word(t,2*(v7+(j>>1)))
    solver.add(tocmp==v10)
    if i==debug:
        print(hex(v10))
    v7+=16
    if k==15:
        break
    tocmp=read_word(g,k*2)
    v8=read_word(t,v7*2)
    k+=1
if solver.check()==sat:
    flag2=[0]*flen
    m=solver.model()
    for i in range(flen):
        flag2[i]=m[flag[i]].as_long()
    flag22=[]
    for i in range(len(flag2)):
        flag22.append(hex(flag2[i]))
    print(flag2)
```

解出第二段,[202, 211, 229, 67, 242, 60, 63, 236, 51, 169, 63, 149, 221, 140, 76, 42],带进第一段解密程序里解密,
解密得到flag:flag{mo4a3Ov2r5qlYgF8},但是不知为啥提交不对。。。
后面发现一个二维码,但是也扫不出来,网上说是能扫出来的。。。