[原创][原创]2019KCTF总决赛 第七题：东北奇闻 - 脱机算法 优

贝a塔 3

大牛

2019-12-19 14:12

大致流程，脱机算法搞出来，然后分析

1-题目改了androidx.appcompat.app.AppCmpatActivity; 在里面找到showAssist native
2-SO解密了2部分内存数据块，存放起来
3-手动抠出算法，内存加密函数2个，func1和func2 ，函数1直接可看，函数2 肉眼解一下 ollvm就可以找出算法
4-第二个函数func2是char加密，实际是个映射表，跑一遍255个字节就出来了
5-最后的比较函数根据第三步的表倒推出最原始的输入数据的一组地址

/**********找到对应的判断函数**********/
registernative 对应的函数地址是 0x19f81

```
75B90  65 6E 74 2F 43 6F 6E 74  65 78 74 3B 29 56 00 6F  ent/Content;)V.o
75BA0  6E 43 72 65 61 74 65 00  73 68 6F 77 41 73 73 69  nCreate.showAssi
75BB0  73 74 00 28 4C 61 6E 64  72 6F 69 64 2F 6F 73 2F  st.(Landroid/os/
75BC0  42 75 6E 64 6C 65 3B 29  5A 00 63 61 6E 6E 6F 74  Bundle;)Z.cannot
```

ida修改下sp，可以看到函数了
之后就是 通过 jni调用GetText获取输入值,这里假设先输入 gabcdefghijklmn来测试
最终根据JNI函数来找到内存字符串

```
AEB93470  70 47 E8 B6 01 00 00 00  00 00 00 00 00 00 00 00  pG..............
AEB93480  67 61 62 63 64 65 66 67  68 69 6A 6B 6C 6D 6E 00  gabcdefghijklmn.
AEB93490  70 6F 73 69 74 69 6F 6E  00 A5 B0 9F 00 00 00 00  position........
```

跟着字符串走到 0x9ce4处，这里是把输入的字符 每隔8个传入，开头是解密一块内存，但是后边的运算是要用到这4个数组，

比较运气好的是 是连续的 大概0xC00长度位一块
所以直接dump保存下来到文件,后面解析用到

```
R0  AEE7B010 .data:dword_AEE7B010
R1  AEE7B410 .data:dword_AEE7B410
R2  AEE7BC10 .data:dword_AEE7BC10
R3  AEE7B810 .data:dword_AEE7B810
R4  04FD24D8
```

```python
import idaapi
start_address = 0xAEE7B010
data_length = 0xc00 * 4
data = idaapi.dbg_read_memory(start_address , data_length)
fp = open('/Users/beita/tmp/pediy/ctfq4/barr_dump_bin', 'wb')
fp.write(data)
fp.close()
```

这部分代码用来第一次进行计算， 大致是前4个字节和猴子哥字节分别保存起来，然后用 字节对应到上面的4个数组元素来进行 加减异或运算 ， 然后ROR4 再运算， 大概是4组数据

处理完之后 ， 把加密后的2个数据， v37和v35的值传入到 DF18进行第二个加密

第二个加密函数分析　　代码 在附件 abc.c文件里

这部分代码在附件里　像是被ollvm的，， 不过肉眼过一下流程看看还好

```
824行     先保存参数到v70   params1=0x5d
  ================================================此时的param1是v104  也就是0x5d ================================================
314行     然后算出v111   v111  = params1 >> 6 = 1
                      v112 = params1 & 3 = 1

1024行      v118 = (params1 >> 4) & 3 ^ v116;  此处的116是上一个的112   此处v118=0
            v119 = (params1 >> 2) & 3 ^ v115;  此处的115是上一个的111   此处v119=2
                v29 = -1811835405;
484行     计算     v101
        LODWORD(v101) = 2 * params & 0xAA;
        HIDWORD(v101) = ((unsigned int)params >> 1) & 0x55;
        v101= 400000aa   64位的
781行   算出v20  =  0xae  然后0xae保存起来、
         v20 = HIDWORD(v101) | v101;
        根据v20 算出v103  v102
       v103 = v20 & 0xCC;           0x8c
       v102 = 4 * v20 & 0xCC;      0x88

1191行  根据v103 v102算出  V104  v105
    v104 = v102 | (v103 >> 2);    0xab
    v105 = o * (o - 1) & 1;      0


967行   对一开始的一个传入的0x5d的那个参数值重新赋值
后面是几个保存传入参数的变量都进行赋值
  ================================================此时的param1是v104  也就是0xab ================================================
314行     然后算出v111   v111  = params1 >> 6 = 2
                      v112 = params1 & 3 = 3
                     v116 = v112
                     v117=v111 ^ v112

1024行      v118 = (params1 >> 4) & 3 ^ v116;  此处的116是上一个的112   此处v118=1
            v119 = (params1 >> 2) & 3 ^ v115;  此处的115是上一个的111   此处v119=2

1141行       算出   v124
           LODWORD(v124) = v117 << 6;
            HIDWORD(v124) = 16 * v119 & 0x30;

354行     算出v125
        v125 = v124 | HIDWORD(v124);

终于来到了 1207行   计算返回结果
  v53 = v125 | 4 * v116 & 0xFFFFFFC;
   v58 = v53 & v118;
  v59 = v53 ^ (unsigned __int8)v118;
  v60 = v59 | v58;
  然后反转 v60 也就是 0xd4反转0x4d
```

继续往下跟 最终结果是通过 strcmp来比较的

可以看到只要前半部分能对应上就可以通过

```
68dd8a0f7065609e3106fb2bb1059423e80fb1347318ffeb83b8a074a7e6c9cf d9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b
426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b4269
19547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfc

d4ac1062d8474ae693be4254066e4b6cd9b426919547bcfcd9b426919547bcfc d9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b
426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b4269
19547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfcd9b426919547bcfc
```

然后 68dd8a0f7065609e3106fb2bb1059423e80fb1347318ffeb83b8a074a7e6c9cf

找出这个字符串

因为第二个加密算法已经分析过， 传入是 uint8_t类型 ， 输出也是uint_8t类型，所以，这里可以搞一个映射表

```
typedef struct {
    uint8_t orig;
    uint8_t enc;
} CHAR_TABLE;
void create_table() {
    char sss[2048];
    memset(sss, 0, 2048);

    char *key = "68dd8a0f7065609e3106fb2bb1059423e80fb1347318ffeb83b8a074a7e6c9cfd9\n";
    size_t right_key_len = strlen(key) / 2 + 1;
    uint8_t after_key_elem[right_key_len];          // 加密后的key数组
    uint8_t before_key_elem[right_key_len];                          // 输入之前的值
    memset(after_key_elem, 0, right_key_len);
    memset(before_key_elem, 0, right_key_len);
    char *p = key;
    for (int i = 0; i < right_key_len; i++) {
        char k_tmp[3];
        memset(k_tmp, 0, 3);
        memcpy(k_tmp, p, 2);
        after_key_elem[i] = strtol(k_tmp, NULL, 16);
        p += 2;
    }


    CHAR_TABLE enc_table[256];
    for (int i = 0; i < 256; i++) {
        enc_table[i].orig = i;
        enc_table[i].enc = encrypt2(i);
    }

    for (int i = 0; i < right_key_len; i++) {
        uint8_t v1 = after_key_elem[i];
        // 用加密后的每个元素取在映射表里遍历查找
        for (int k = 0; k < 256; k++) {
            CHAR_TABLE c = enc_table[k];
            if (v1 == c.enc) {
                before_key_elem[i] = c.orig;
                break;
            }
        }
    }

    printf("gogogo\n");
    printf("\n%s\n", sss);
}
```

然后得到 对应的表

68dd8a0f7065609e3106fb2bb1059423e80fb1347318ffeb83b8a074a7e6c9cfd9 字符串对应原始表在这里查找

运行一下看看脱机效果

比如这里我输入的是 gabcdefghijklmn 在手机上看 第一部分加密函数的结果

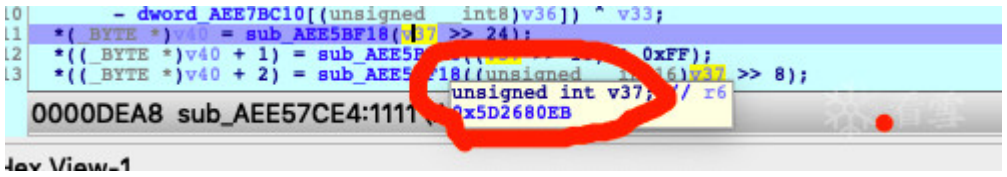
用两组数据

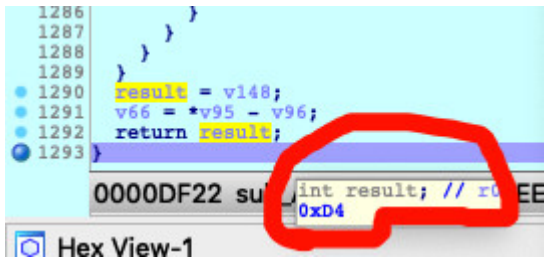
**********手机上**********

输入 后先截取前8字节 gabcdefg ， 看结果下图, 传入 df18的是 0x5d2680eb >> 24 那么就是说

gabcdefg被加密成 0x5d2680eb 然后传入高8位字节 也就是 0x5d

结果是输入0x5d 输出0xd4



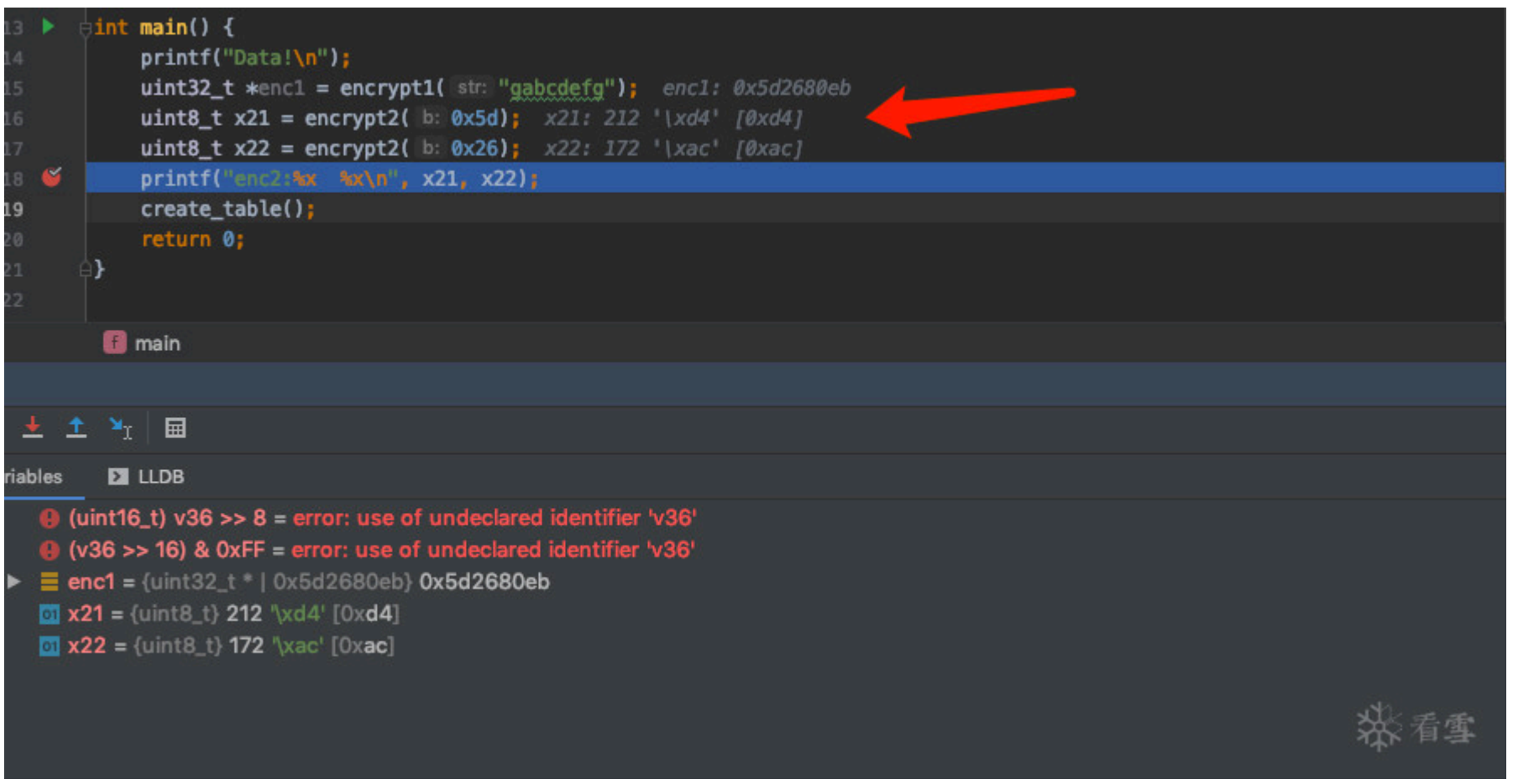


********** 脱机代码在附件**********

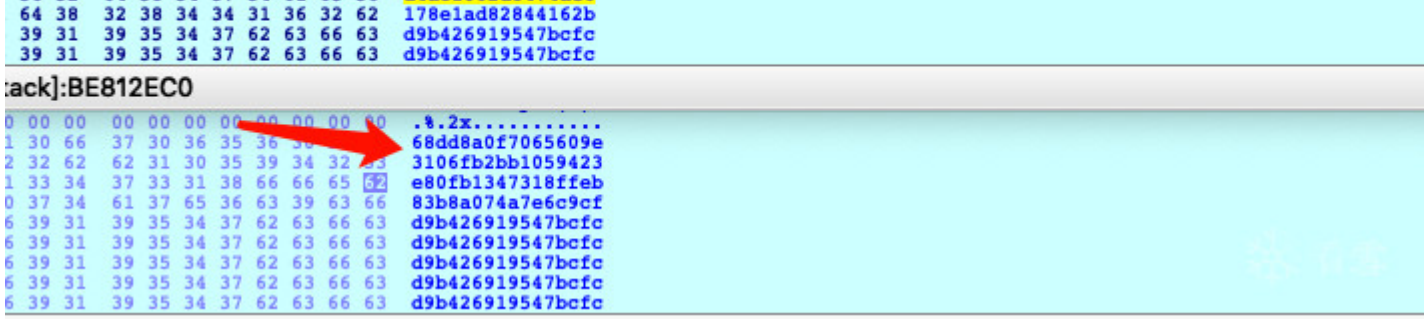
看到输入gabcdefg 测试，然后

加密0x5d 输出0xd4,

加密0x26输出 0xac



最后加密的结果也是在这里计算结束后后和 strcmp的2个参数 就是用的这里的第二个加密函数的结果来进行比较的，说明脱机加密是对的



这里证是对脱机的传入的值进行比较，说明算法是对的

****

脱机算法 如下 ， 具体内容在附件里， 代码可以运行，改一下文件路径就可以运行

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <zconf.h>

uint32_t *encrypt1(const char *str);

uint8_t encrypt2(uint8_t b);

#define ROTL(val, n) (((val)<<n) | ((val)>>(sizeof(val)*8-n)))
#define __ROL__(x, y) ROTL(x, y)      // Rotate left


#define LODWORD(x)  (*((uint32_t *)&(x)))
#define HIDWORD(x)  (*((uint32_t *)&(x)+1))

#define BYTEn(x, n)   (*((uint8_t*)&(x)+n))
#define BYTE4(x)   BYTEn(x,  4)                   // 取第4个字节

typedef struct {
    uint8_t orig;
    uint8_t enc;
} CHAR_TABLE;

char *readfile(const char *filename) {

    struct stat s;
    stat(filename, &s);
    int file_len = s.st_size;
    int fd = open(filename, O_RDONLY);
    char *data = malloc(file_len);
    lseek(fd, 0, SEEK_SET);
    memset(data, 0, file_len);
    int e = read(fd, data, file_len);
    return data;
}


uint32_t __ROR4__(uint32_t value, int count) {
    return __ROL__((uint32_t) value, -count);
}

uint32_t bswap32(uint32_t *val) {
    uint32_t v = *((uint32_t *) val);
    v = (
                (v & 0x000000FF) << 24) |
        ((v & 0x0000FF00) << 8) |
        ((v & 0x00FF0000) >> 8) |
        ((v & 0xFF000000) >> 24);
    return v;

}

uint32_t *encrypt1(const char *str) {
    char *data = readfile("/Users/beitaCLionProjects/untitled6/barr_dump_bin");
    char *v10 = readfile("/Users/beita/CLionProjects/untitled6/v10_bin");
    uint32_t *barr1 = data;
    uint32_t *barr2 = (data + 0x400);
    uint32_t *barr3 = (data + 0x800);
    uint32_t *barr4 = (data + 0xC00);


    uint32_t data1 = *(uint32_t *) str;
    uint32_t data2 = *(uint32_t *) (str + 4);
    uint32_t v12 = bswap32(&data1);
    uint32_t v13 = bswap32(&data2);
    uint32_t xx = 0xD573F1E4 + v13;
    uint32_t v14111 = __ROR4__(xx, 0x10);
    uint32_t v14 = __ROR4__(*(uint32_t *) v10 + v13, 0x20 - *(uint32_t *) (v10 + 0x40));
    uint32_t eee = barr1[88];
    // .text:0000DC3E 8B EA 04 0B                EOR.W           R11, R11, R4   R11存放的是这个值
    uint32_t v15 = v12 ^((barr2[(v14 >> 16) & 0xFF] ^ barr1[v14 >> 24])
                         - barr3[(uint16_t) v14 >> 8]
                         + barr4[(uint8_t) v14]);
    uint32_t v16 = __ROR4__(v15 ^ *(uint32_t *) (v10 + 4), 32 - *(uint32_t *) (v10 + 68));
    uint32_t v17 =
            (barr1[v16 >> 24] - barr2[(v16 >> 16) & 0xFF] + barr3[(uint16_t) v16 >> 8]) ^
            v13 ^barr4[(uint8_t) v16];
    uint32_t v18 = __ROR4__(*(uint32_t *) (v10 + 8) - v17, 32 - *(uint32_t *) (v10 + 72));
    uint32_t v19 =
            (((barr2[(v18 >> 16) & 0xFF] + barr1[v18 >> 24]) ^ barr3[(uint16_t) v18 >> 8])
             - barr4[(uint8_t) v18]) ^v15;
    uint32_t v20 = __ROR4__(v19 + *(uint32_t *) (v10 + 12), 32 - *(uint32_t *) (v10 + 76));
    uint32_t v21 = (barr4[(uint8_t) v20]
```

```
    uint32_t v22 = __ROR4__(*(uint32_t *) (v10 + 16) ^ v21, 32 - *(uint32_t *) (v10 + 80));
    uint32_t v23 = barr4[(uint8_t) v22] ^v19 ^(barr1[v22 >> 24]
                                             - barr2[(v22 >> 16) & 0xFF]
                                             + barr3[(uint16_t) v22 >> 8]);
    uint32_t v24 = __ROR4__(*(uint32_t *) (v10 + 20) - v23, 32 - *(uint32_t *) (v10 + 84));
    uint32_t v25 = v21 ^(((barr1[v24 >> 24] + barr2[(v24 >> 16) & 0xFF]) ^
                          barr3[(uint16_t) v24 >> 8])
                         - barr4[(uint8_t) v24]);
    uint32_t v26 = __ROR4__(*(uint32_t *) (v10 + 24) + v25, 32 - *(uint32_t *) (v10 + 88));
    uint32_t v27 = v23 ^((barr2[(v26 >> 16) & 0xFF] ^ barr1[v26 >> 24])
                         - barr3[(uint16_t) v26 >> 8]
                         + barr4[(uint8_t) v26]);
    uint32_t v28 = __ROR4__(*(uint32_t *) (v10 + 28) ^ v27, 32 - *(uint32_t *) (v10 + 92));
    uint32_t v29 = barr4[(uint8_t) v28] ^v25 ^(barr1[v28 >> 24]
                                             - barr2[(v28 >> 16) & 0xFF]
                                             + barr3[(uint16_t) v28 >> 8]);
    uint32_t v30 = __ROR4__(*(uint32_t *) (v10 + 32) - v29, 32 - *(uint32_t *) (v10 + 96));
    uint32_t v31 = v27 ^(((barr2[(v30 >> 16) & 0xFF] + barr1[v30 >> 24]) ^
                          barr3[(uint16_t) v30 >> 8])
                         - barr4[(uint8_t) v30]);
    uint32_t v32 = __ROR4__(*(uint32_t *) (v10 + 36) + v31, 32 - *(uint32_t *) (v10 + 100));
    uint32_t v33 = v29 ^((barr1[v32 >> 24] ^ barr2[(v32 >> 16) & 0xFF])
                         - barr3[(uint16_t) v32 >> 8]
                         + barr4[(uint8_t) v32]);
    uint32_t v34 = __ROR4__(*(uint32_t *) (v10 + 40) ^ v33, 32 - *(uint32_t *) (v10 + 104));
    uint32_t v35 =
            (barr1[v34 >> 24] - barr2[(v34 >> 16) & 0xFF] + barr3[(uint16_t) v34 >> 8]) ^
            barr4[(uint8_t) v34] ^v31;
    uint32_t v36 = __ROR4__(*(uint32_t *) (v10 + 44) - v35, 32 - *(uint8_t *) (v10 + 108));

    // 第三个字节和第一个字节异或 + 第二个字节 - 第4个字节
    uint32_t v37 =
            ((barr3[(uint16_t) v36 >> 8] ^ (barr1[v36 >> 24] + barr2[(v36 >> 16) & 0xFF]))
             - barr4[(uint8_t) v36]) ^v33;
    uint32_t *vv37 = v37;
    printf("v37: %x %x", v37, 123);
    return vv37;
}

uint8_t encrypt2(uint8_t b) {
    uint32_t params1 = b;
    uint8_t v111 = params1 >> 6;
    uint8_t v112 = params1 & 3;
    int v115 = v111;
    int v116 = v112;
    uint32_t v118 = (params1 >> 4) & 3 ^v116;
    uint32_t v119 = (params1 >> 2) & 3 ^v115;
    uint64_t v101 = 0;
    LODWORD(v101) = 2 * params1 & 0xAA;
    HIDWORD(v101) = ((uint32_t) params1 >> 1) & 0x55;
    int v20 = HIDWORD(v101) | v101;
    uint32_t v103 = v20 & 0xCC;
    uint32_t v102 = 4 * v20 & 0xCC;
    uint8_t v104 = v102 | (v103 >> 2);   // 0xab
    char v105 = 0 * (0 - 1) & 1;

    // params1被重新赋值  是v104
    params1 = v104;
    v111 = params1 >> 6;
    v112 = params1 & 3;
    v115 = v111;
    v116 = v112;
    int v117 = v111 ^v112;
    v118 = (params1 >> 4) & 3 ^ v116;
    v119 = (params1 >> 2) & 3 ^ v115;
    uint64_t v124 = 0;
    LODWORD(v124) = v117 << 6;
    HIDWORD(v124) = 16 * v119 & 0x30;

    int v125 = v124 | HIDWORD(v124);

    int v53 = v125 | 4 * v116 & 0xFFFFFFC;
    char v58 = v53 & v118;
    int v59 = v53 ^(uint8_t) v118;
    uint8_t v60 = v59 | v58;
    uint8_t return_val = (v60 >> 4) | 16 * v60;
    return return_val;
}

void create_table() {
    char sss[2048];
    memset(sss, 0, 2048);

    char *key = "68dd8a0f7065609e3106fb2bb1059423e80fb1347318ffeb83b8a074a7e6c9cfd9\n";
    size_t right_key_len = strlen(key) / 2 + 1;
    uint8_t after_key_elem[right_key_len];         // 加密后的key数组
    uint8_t before_key_elem[right_key_len];                         // 输入之前的值
```

```
    for (int i = 0; i < right_key_len; i++) {
        char k_tmp[3];
        memset(k_tmp, 0, 3);
        memcpy(k_tmp, p, 2);
        after_key_elem[i] = strtol(k_tmp, NULL, 16);
        p += 2;
    }


    CHAR_TABLE enc_table[256];
    for (int i = 0; i < 256; i++) {
        enc_table[i].orig = i;
        enc_table[i].enc = encrypt2(i);
        char tmp[8];
        memset(tmp, 0, 8);
        sprintf(tmp, "0x%02x, ", enc_table[i].enc);
        strcat(sss, tmp);
    }

    for (int i = 0; i < right_key_len; i++) {
        uint8_t v1 = after_key_elem[i];
        // 用加密后的每个元素取在映射表里遍历查找
        for (int k = 0; k < 256; k++) {
            CHAR_TABLE c = enc_table[k];
            if (v1 == c.enc) {
                before_key_elem[i] = c.orig;
                break;
            }
        }
    }

    printf("gogogo\n");
    printf("\n%s\n", sss);
}


int main() {
    printf("Data!\n");
    uint32_t *enc1 = encrypt1("gabcdefgh");
    uint8_t x21 = encrypt2(0x5d);
    uint8_t x22 = encrypt2(0x26);
    printf("enc2:%x  %x\n", x21, x22);
    create_table();
    return 0;
}
```

然后 根据脱机加密算法 电脑分析 一下 得到结果
flag:flag{9eca5de49470144c1694f6}

[注意]看雪招聘，专注安全领域的专业人才平台！

最后于 2019-12-26 19:39 被贝a塔编辑 ， 原因： 图搞错了

上传的附件：

barr_dump_bin （12.00kb，2次下载）
abc.c （126.57kb，3次下载）
v10_bin （0.20kb，2次下载）

收藏 · 1 支持 分享

赞赏记录

参与人	雪币	留言	时间
PLEBEE		为你点赞	2023-1-30 01:06

最新回复 (0)

贝a塔

3 年没回复的帖子已锁定，有问题请联系版主!

回帖 表情 高级回复

0

1

返回