看雪ctf晋级赛第二题wp

原创

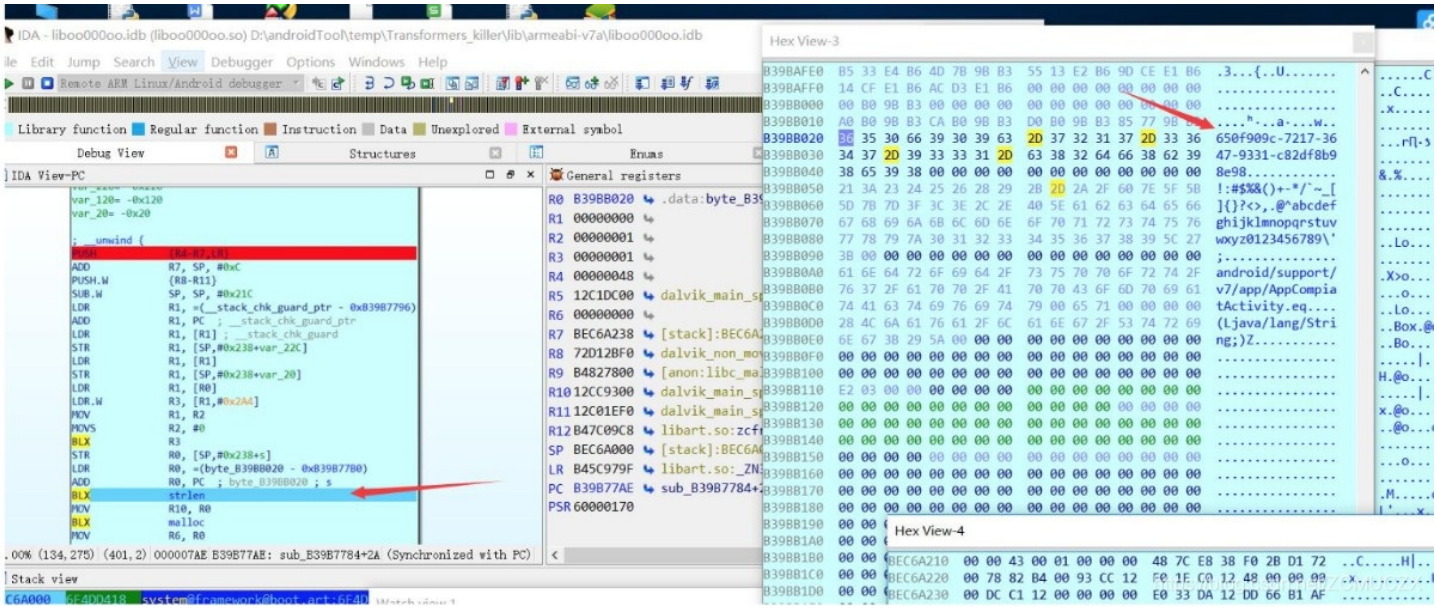
rv0p111 于 2019-10-08 13:19:24 发布 352 收藏

分类专栏： 安全相关

本文链接：https://blog.csdn.net/ZCMUCZX/article/details/102379335

安卓题目，应该就是有关动态库的题目了，所用对动态库进行分析，利用工具IDA、jeb、vs等

利用IDA动态调试，对sub_784进行分析，可以确定其实就是是rc4加密算法，以及下面的进行编码的业务逻辑组成



用到的其实就是RC4算法：

1、密钥流：RC4算法的关键是根据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是对应的，也就是说明文的长度是500字节，那么密钥流也是500字节。当然，加密生成的密文也是500字节，因为密文第i字节=明文第i字节^密钥流第i字节；

2、状态向量S：长度为256，S[0],S[1].....S[255]。每个单元都是一个字节，算法运行的任何时候，S都包括0-255的8比特数的排列组合，只不过值的位置发生了变换；

3、临时向量T：长度也为256，每个单元也是一个字节。如果密钥的长度是256字节，就直接把密钥的值赋给T，否则，轮转地将密钥的每个字节赋给T；

4、密钥K：长度为1-256字节，注意密钥的长度 keylen 与明文长度、密钥流的长度没有必然关系，通常密钥的长度趣味16字节（128比特）

密文第i字节=明文第i字节^密钥流第i字节；

明文第i字节=密文第i字节^密钥流第i字节；

密钥长度任意（1-256B），密文长度=密钥流长度=明文长度

二、本程序中的RC4算法应用

v36 = v46 ^ s

明文：s

密文：v36

向量S：unk_23E8256Bytes

密钥流：v46（来自于向量S）

三、作者剩余逻辑

v26 = byte_4050[v36的一些变换] ^ const

v26 == " {9*8gal*!Tn?@#fj'j$\g;;"

四、思路

1、从v26逆推出v36

2、从前面的计算里由unk_23E8推出v46

3、s = v36 ^ v46

所以我们依据这个思路，我们就是将题目当中的程序给逆推了下，查看ida的伪代码，我们应该输入的字符串长度应该是16位的，然后最后变成了24位，可以根据下面来判断

```
do
{
  v22 = (unsigned __int8)v46[v21];
  v20 = (v20 + (unsigned __int8)v47[v21] + v22) % 256;
  v46[v21++] = v46[v20];
  v46[v20] = v22;
}
while ( v21 != 256 );

v23 = strlen(s);
v24 = v23;
v25 = (unsigned __int8)v4[3];
v43 = 8 * (3 - -3 * (v23 / 3));
v42 = v25 + v43 / 6;
v26 = malloc(v42 + 1);
if ( v24 )
{
  v28 = 0;
  v29 = 0;
  v30 = 0;
  v44 = v25;
  do
  {
    v28 = (v28 + 1) % 256;
```


我们的{9*8gal*!Tn?@#fj'j$\g;;就是24位的，然后动态调试，输入16个字符串，看看是怎么变化的

v25 = 51 v42=75
0 a的时候 v26[51]=5B v36=0xFB
1 b v26[52]=67 v36=9

2 c v26[53]=61 v36=0xD9
v26[54]=5E

3 d v26[55]=5A v36=0x41
4 e v26[56]=7B v36=0x16

5 f v26[57]=6D v36=0xe4
v26[58]=6B

6 g v26[59]=6F v36=0x87
7 h v26[60]=31 v36=0x59

8 i v26[61]=61 v36=0xEF
v26[62]=76

9 j v26[63]=31

10 k v26[64]=70

11 l v26[65]=70
v26[66]=73

12 m v26[67]=5C

13 n v26[68]=76

14 o v26[69]= 31
v26[70]= 62

15 p v26[71]= 21

72 *(_BYTE *)(v17 + 1) = v38;
最后两个字符是自动给加上的

3B
3B

下面是分别取每一个字符进行爆破
空格{98gal!T?@#j'j\

0 空格

1 {

2 9
*

3 8

4 g

5 a
*

6 l

7 !

8 T
n

9 ?

10 @

11 #
f

12 j

13 ’

14 j
$

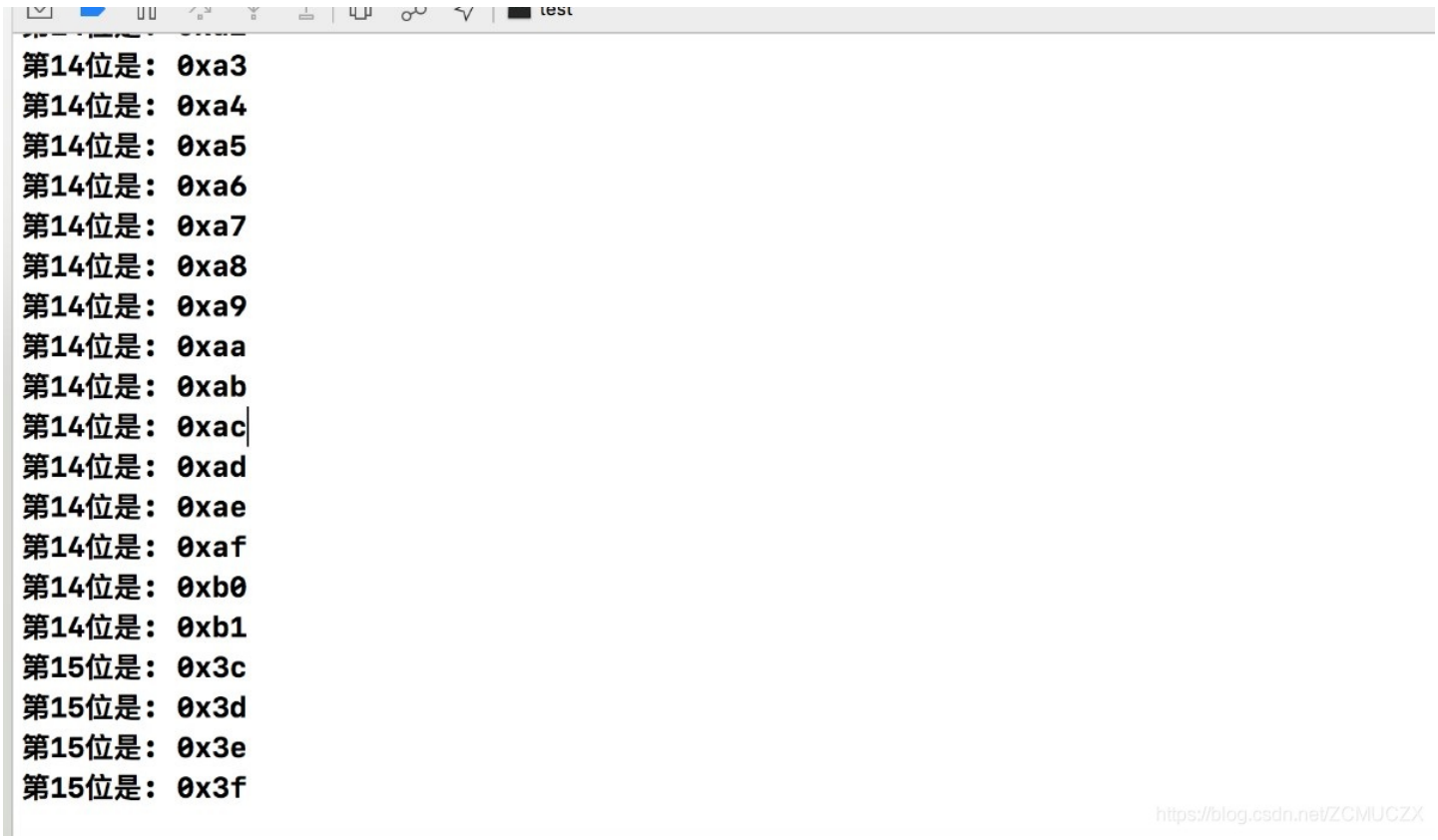
15 \

所以最终我们爆破只需要去爆破空格{98gal!T?@#j'j\这个就可以，倒推的函数，写出了下面的程序，可以成功的将v36给跑出来

```
398
399 #include "stdio.h"
400 #include "stdlib.h"
401
402 char byte_4050[] = "!:#$%&()+-*/`~_[]{}?
        <>,.@^abcdefghijklmnopqrstuvwxyz0123456789\\';";
403 char v26_real[17] = " {98gal!T?@#j'j\\";
404 unsigned char v36[16][100] = { 0 };
405 int flag = 0;
406
407 int main()
408 {
409     char *v26;
410     v26 = (char *)malloc(sizeof(char) * 17);
411     unsigned int temp[3][100] = { 0 };
412     int num[3] = { 0 };
413
414     for (int i = 0; i < 16; i++)
415     {
416         //0,3,6,9,12,15,18,21
417         if (i % 3 == 0)
418         {
419             num[0] = -1;
420             for (unsigned int j = 0; j < 255; j++)
421             {
422                 *(v26 + i) = byte_4050[j >> 2] ^ 7;                    //
                        v26[v44 + v29] = aAbcdefghijklmn[(unsigned int)v36 >> 2]
```



```
437             while (num[0] >= 0)
438             {
439                 flag = 0;
440                 for (unsigned int j = 0; j < 256; j++)
441                 {
442                     if ((temp[0][num[0]] | (j >> 4)) > 65)
443                         continue;
444
445                     *(v26 + i) = byte_4050[temp[0][num[0]] | (j >> 4)];
```



最后再可以求出多个v36，然后一个个尝试，根据伪代码当中的下面的语句，进行异或v46的值，我们的v46可以通过IDA动态调试进行导出，v36的值为char v36[17] = { 0xFD,
// 0x1E,
// 0x8A,
// 0x4E,
// 9,
// 0xCA,
// 0x90,
// 3,
// 0xE7,
// 0xF1,
// 0x85,
// 0x9F,
// 0x9B,
// 0xF7,
// 0x83,
// 0x3E };

```
156     v29 = 0;
157     v30 = 0;
158     v44 = v25;
159     do
160     {
161       v28 = (v28 + 1) % 256;
162       v35 = (unsigned __int8)v46[v28];
163       v30 = (v30 + v35) % 256;
164       v46[v28] = v46[v30];
165       v46[v30] = v35;
166       v17 = (unsigned __int8)v46[v28];
167       v36 = v46[(unsigned __int8)(v35 + v17)] ^ s[v29];
168       if ( v29 && (v27 = 2863311531u * (unsigned __int64)v29 >> 32, v37 = 3 * (v29 / 3), v37 != v29) )
169       {
170         v31 = v29 == 1;
171         if ( v29 != 1 )
172           v31 = v37 + 1 == v29;
173         if ( v31 )
174         {
175           v32 = byte_4050;
176           v26[v44 + v29] = byte_4050[(unsigned __int8)v26[v44 + v29] | ((unsigned int)v36 >> 4)];
177           v17 = (unsigned int)&v26[v44 + v29];
178           v27 = 4 * v36 & 0x3C;
179           *(_BYTE *)(v17 + 1) = v27;
180           if ( v29 + 1 >= v24 )
181             goto LABEL_53;
182         }
183         else
184         {
185           v33 = v29 == 2;
186           if ( v29 != 2 )
187             v33 = v37 + 2 == v29;
188           if ( v33 )
189           {
190             v17 = v36 & 0xC0;
191             v34 = v44++ + v29;
192             v26[v34] = byte_4050[(unsigned __int8)v26[v34] | (v17 >> 6)] ^ 0xF;
193             v27 = (int)&v26[v34];
194             *(_BYTE *)(v27 + 1) = byte_4050[v36 & 0x3F];
195           }
196         }
197       }
198       else
199       {
200         v26[v44 + v29] = byte_4050[(unsigned int)v36 >> 2] ^ 7;
201         v17 = (unsigned int)&v26[v44 + v29];
202         v27 = 16 * v36 & 0x30;
203         *(_BYTE *)(v17 + 1) = v27;
204         if ( v29 + 1 >= v24 )
```


就可以得到flag:fu0kzHp2aqtZAuY6