CISCN2021初赛WriteUp

原创

Hellsegamosken 于 2021-05-19 16:00:31 发布 509 收藏 1

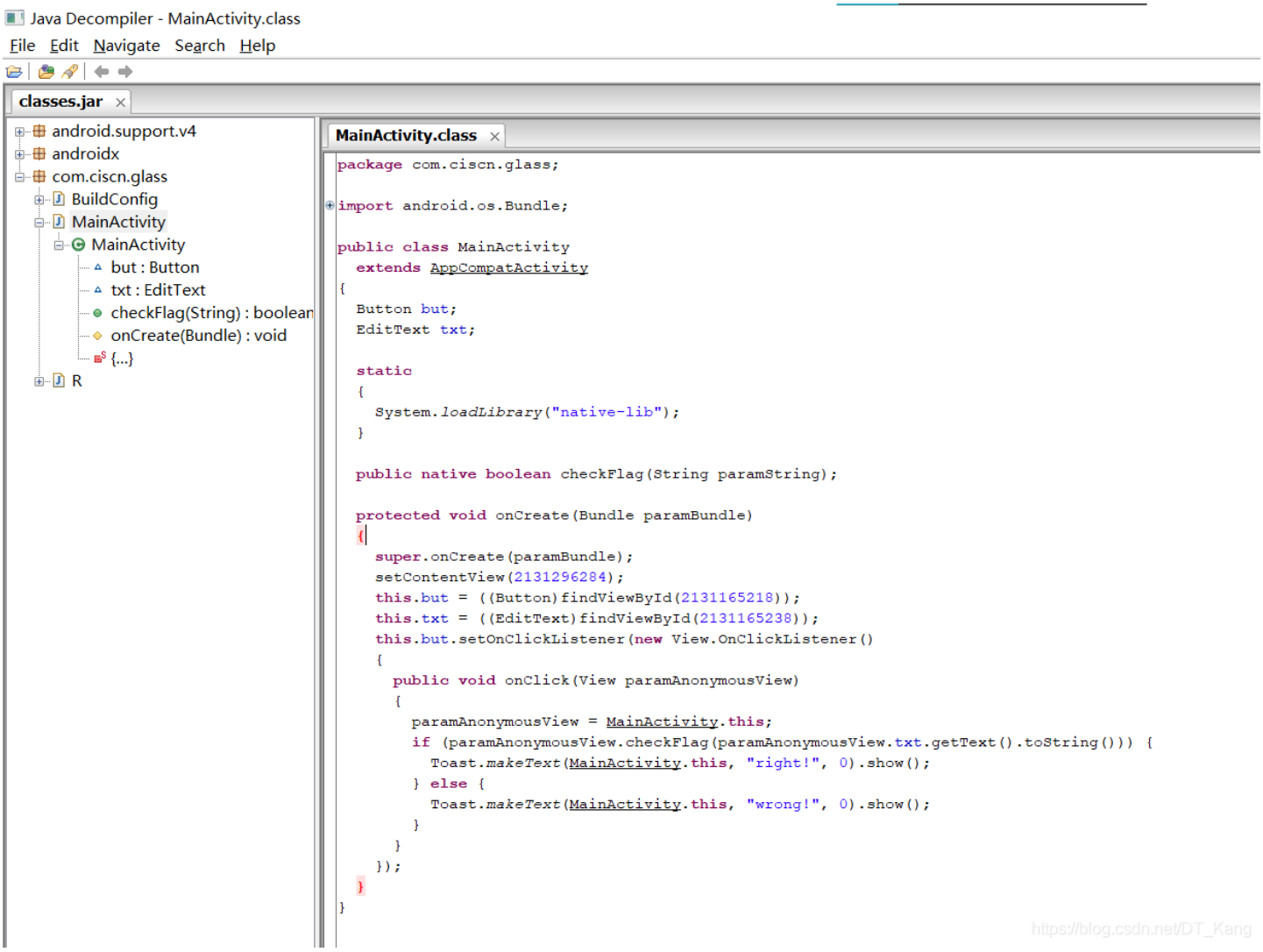
分类专栏： 杂题瞎记 文章标签： 逆向 ctf

本文链接：https://blog.csdn.net/DT_Kang/article/details/117031383

第一次打 ctf，啥也没学，赛前一道题没做过，了解了下 IDA 的使用就上了。

glass

200 分 reverse，安卓逆向，.apk 改 .zip 解压，classes.dex 转 classes.jar 后打开，找到 MainActivity 开始读代码。



发现调用了 `public native boolean checkFlag(String paramString);` 这样一个函数，但这个函数并没有写出来。搜了一下 `native` 关键字，发现是用来调用本地 C 代码的。然后在 glass\lib\armeabi-v7a 里 找到了 libnative-lib.so，直接拖进 IDA 打开。

```
1  bool __fastcall Java_com_ciscn_glass_MainActivity_checkFlag(int a1, int a2, int a3)
2  {
3    char *v3; // r4
4    int v4; // r5
5    char v6[256]; // [sp+0h] [bp-220h] BYREF
6    char v7[260]; // [sp+100h] [bp-120h] BYREF
7
8    v3 = sub_F0C(a1, a3);
9    if ( strlen(v3) != 39 )
10     return 0;
11   memset(v7, 0, 0x100u);
12   qmemcpy(v6, "12345678", sizeof(v6));
13   v4 = strlen(v6);
14   sub_FFC((int)v7, (int)v6, v4);
15   sub_1088((int)v7, v3, 39);
16   sub_10D4((int)v3, 39, (int)v6, v4);
17   return memcmp(v3, &unk_497C, 0x27u) == 0;
18 }
```


分析一波，发现 sub_F0C 是用来把接收的 Java String 转换成 char*。然后主要剩下 sub_FFC, sub_1088, sub_10D4, 这三个函数。进一步分析，猜测程序的主要逻辑是把 flag 经过这三个函数处理，和内存中的一段数据比较。

```
.data:0000497C unk_497C        DCB 0xA3              ; DATA XREF: Java_com_ciscn_glass_MainActivity_checkFlag+6C↑o
.data:0000497C                                       ; Java_com_ciscn_glass_MainActivity_checkFlag+72↑o ...
.data:0000497D                 DCB 0x1A
.data:0000497E                 DCB 0xE3
.data:0000497F                 DCB 0x69 ; i
.data:00004980                 DCB 0x2F ; /
.data:00004981                 DCB 0xBB
.data:00004982                 DCB 0x1A
.data:00004983                 DCB 0x84
.data:00004984                 DCB 0x65 ; e
.data:00004985                 DCB 0xC2
.data:00004986                 DCB 0xAD
.data:00004987                 DCB 0xAD
.data:00004988                 DCB 0x9E
.data:00004989                 DCB 0x96
.data:0000498A                 DCB    5
.data:0000498B                 DCB    2
.data:0000498C                 DCB 0x1F
.data:0000498D                 DCB 0x8E
.data:0000498E                 DCB 0x36 ; 6
.data:0000498F                 DCB 0x4F ; O
.data:00004990                 DCB 0xE1
.data:00004991                 DCB 0xEB
.data:00004992                 DCB 0xAF
.data:00004993                 DCB 0xF0
.data:00004994                 DCB 0xEA
.data:00004995                 DCB 0xC4
.data:00004996                 DCB 0xA8
```


因此只要把这段数据解密回去就行了。由于没有任何前置知识，于是只能进去一行一行读明白。

FFC：

```
1  int __fastcall sub_FFC(int a1, int a2, int a3)
2  {
3    int i; // r6
4    int v7; // r1
5    int ii; // r0
6    int jj; // r1
7    int v10; // r3
8    _BYTE v12[260]; // [sp+0h] [bp-120h] BYREF
9
10   memset(v12, 0, 0x100u);
11   for ( i = 0; i != 256; ++i )
12   {
13     *(_BYTE *)(a1 + i) = i;
14     sub_126C(i, a3);                          // of no use?
15     v12[i] = *(_BYTE *)(a2 + v7);             // all '1' or all 0?
16   }
17   ii = 0;
18   jj = 0;
19   while ( ii != 256 )
20   {
21     v10 = *(unsigned __int8 *)(a1 + ii);      // a1[ii]
22     jj = (jj + v10 + (unsigned __int8)v12[ii]) % 256;// jj += ii + '1' (or 0)
23     *(_BYTE *)(a1 + ii++) = *(_BYTE *)(a1 + jj);
24     *(_BYTE *)(a1 + jj) = v10;                // swap(a1[jj], a1[ii]), ii++
25   }
26   return _stack_chk_guard;
27 }
```


大概就是会得到 v7 这个数组。有几个疑点：

1. 第八行为什么是 260 个字节而不是 256。
2. 变量 v7 既没初始化也没有修改。（我就当做 v7 恒为零做的，不行）
3. 莫名其妙的有个 sub_126C，貌似没有任何影响。

赛后才知道，事实上这是在进行 RC4 加密，而 RC4 加密 v7 应该是不停自增的。

然后 10D4 是对原文进行可逆的异或运算，很好解密。1088 是 拿已知的 v7 数组和原文进行运算，也很好解密。但 1088 里面有一个问题：

```
// a1 is known
int __fastcall sub_1088(int a1, _BYTE *a2, int len2)
{
  int v3; // r3
  int v4; // r4
  int v5; // r5

  v3 = 0;
  v4 = 0;
  while ( len2 )
  {
    --len2;
    v4 = (v4 + 1) % 256;                    // v4++
    v5 = *(unsigned __int8 *)(a1 + v4);     // v5 = a1[v4]
    v3 = (v3 + v5) % 256;                   // v3 += a1[v4]
    *(_BYTE *)(a1 + v4) = *(_BYTE *)(a1 + v3);  // a1[v4] = a1[v3]
    *(_BYTE *)(a1 + v3) = v5;               // a1[v3] = v5
    *a2++ ^= *(_BYTE *)(a1 + (unsigned __int8)(v5 + *(_BYTE *)(a1 + v4)));// a1[v5 + a1[v4]]
  }
  return a1;
}
```


这里面 a1 就是主函数的 v7，值域是 [0, 255]，因此第十八行 v5 + a1[v4] 可能会越界。尝试对 256 取模，无果。赛后知道，这里确实应该取模，但因为比赛的时候上面 FFC 里面的 v7 没自增，于是没有解出。

我不知道为什么 IDA 里面会出现这种情况，里面的算法和 RC4 不完全一样，因此完全按照 IDA 里面的代码写的脚本跑不出 flag。别人都是百度 RC4 解密直接解出 flag。

插一嘴，我看别的 WP，1088 和 10D4 都是暴力枚举解密的，不是逆运算回去的。这样可能根本不需要每行代码都看明白？毕竟 RC4 应该是能一眼看出来的。

赛后修改过的脚本如下：

```
# include <bits/stdc++.h>
# define ll long long
# define db double
# define ld long double
# define pb push_back
# define fir first
# define sec second
# define rep(i, l, r) for (int i = l; i <= r; i++)
# define per(i, r, l) for (int i = r; i >= l; i--)
using namespace std;
typedef pair <int, int> P;

int read() {
    int x = 0; char c = getchar(), flag = '+';
    while (!isdigit(c)) flag = c, c = getchar();
    while (isdigit(c)) x = x * 10 + c - '0', c = getchar();
    return flag == '-' ? -x : x;
}
char v12[1000];

void FFC(unsigned a1[], unsigned a2[], int a3) {
    const int ty = 1;
    int ii = 0, jj = 0;
    unsigned v12[256];
    for (int i = 0; i < 256; i++) a1[i] = i, v12[i] = a2[i % 8];
```

```
    while (ii != 256) {
        int v10 = a1[ii];
        jj = (jj + v10 + v12[ii]) % 256;
        a1[ii++] = a1[jj];
        a1[jj] = v10;
    }
}

void rev(unsigned s[], int len1, unsigned a3[], int len3) {
    for (int j = 0; j < len1; j += len3) {
        for (int k = 0; k < len3 && j + k < len1; k++) {
            s[j + k] ^= a3[k];
        }
    }
    for (int i = 0; i < len1; i += 3) {
        unsigned a = s[i], b = s[i + 1], c = s[i + 2];
        s[i] = b ^ c;
        s[i + 1] = b ^ a;
        s[i + 2] = b ^ a ^ c;
    }
}
void sub_1088(unsigned result[], unsigned a2[], int a3) {
    int v3 = 0, v4 = 0, v5 = 0;
    int loop_39[39];
    memset(loop_39, 0, sizeof(loop_39));
    while(a3) {
        --a3;
        v4 = (v4 + 1) % 256;
        v5 = result[v4];

        v3 = (v3 + v5) % 256;
        result[v4] = result[v3];
        result[v3] = v5;
        //if (v5 + result[v4] > 255) cout<<"#";
        loop_39[38 - a3] = result[(v5 + result[v4]) % 256];
    }
    for (int i = 0; i < 39; i++) {
        a2[i] ^= loop_39[i];
    }
}
int main() {
    unsigned v6[256], v7[256];
    unsigned v3[] = {0xA3, 0x1A, 0xE3, 0x69, 0x2F, 0xBB, 0x1A, 0x84, 0x65, 0xC2, 0xAD, 0xAD, 0x9E, 0x96, 0x05, 0
x02, 0x1F, 0x8E, 0x36, 0x4F, 0xE1, 0xEB, 0xAF, 0xF0, 0xEA, 0xC4, 0xA8, 0x2D, 0x42, 0xC7, 0x6E, 0x3F, 0xB0, 0xD3,
 0xCC, 0x78, 0xF9, 0x98, 0x3F};

    memset(v7, 0, sizeof(unsigned) * 256);

    v6[0] = '1';
    v6[1] = '2';
    v6[2] = '3';
    v6[3] = '4';
    v6[4] = '5';
    v6[5] = '6';
    v6[6] = '7';
    v6[7] = '8';
    int v4 = 8;
    FFC(v7, v6, v4);
```

```
    rev(v3, 39, v6, v4);
    sub_1088(v7, v3, 39);
    // for (int i = 0; i < 39; i++) {
    //     printf("%d ", v3[i]);
    // }
    // cout << endl;
    for (int i = 0; i < 39; i++) {
        printf("%c", v3[i]);
    }
    return 0;
}
/* Hellsegamosken */
```