

逆向BSides SF CTF之flagstore.apk

原创

caiqi1qi 于 2017-08-21 21:20:51 发布 2340 收藏

分类专栏: [Android逆向](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/caiqi1qi/article/details/77460651>

版权



[Android逆向](#) 专栏收录该内容

52 篇文章 2 订阅

订阅专栏

看有一个writeup写了个装drozer, 然后尝试了一下并没有成功。感觉虽然功能不错, 但是有点麻烦。然后参考:

<https://ctf.rip/bsides-sf-ctf-2017-flag-receiver-mobile-reverse-engineering/>

<https://russtone.io/2017/02/14/bssidessf-2017-flagreceiver/>

在 `MainActivity` 中发现启动之后就向 `Send_to_Activity` 这个广播接收器注册了一个action 为 `com.flagstore.ctf.INCOMING_INTENT` 的广播,

```
public class MainActivity extends Activity {
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         TextView tv = new TextView(getApplicationContext());
15         tv.setText("To-do: UI pending");
16         setContentView(tv);
17         IntentFilter filter = new IntentFilter();
18         filter.addAction("com.flagstore.ctf.INCOMING_INTENT");
20         registerReceiver(new Send_to_Activity(), filter, permission._MSG, null);
    }
}
```

<http://blog.csdn.net/caiqi1qi>

而且界面只有一个 `to-do: UI pending`, 在 `Send_to_Activity` 的 `onReceive()` 方法中可以发现, 需要发送一个满足特定条件的广播才能激活进入下一个Activity。

```
public class Send_to_Activity extends BroadcastReceiver {
12     public void onReceive(Context context, Intent intent) {
15         if (intent.getStringExtra("msg").equalsIgnoreCase("OpenSesame")) {
16             Log.d("Here", "Intent");
18             context.startActivity(new Intent(context, CTFReceiver.class));
24             return;
        }
22     Toast.makeText(context, "Ah, ah, ah, you didn't say the magic word!", 1).show();
    }
}
```

<http://blog.csdn.net/caiqi1qi>

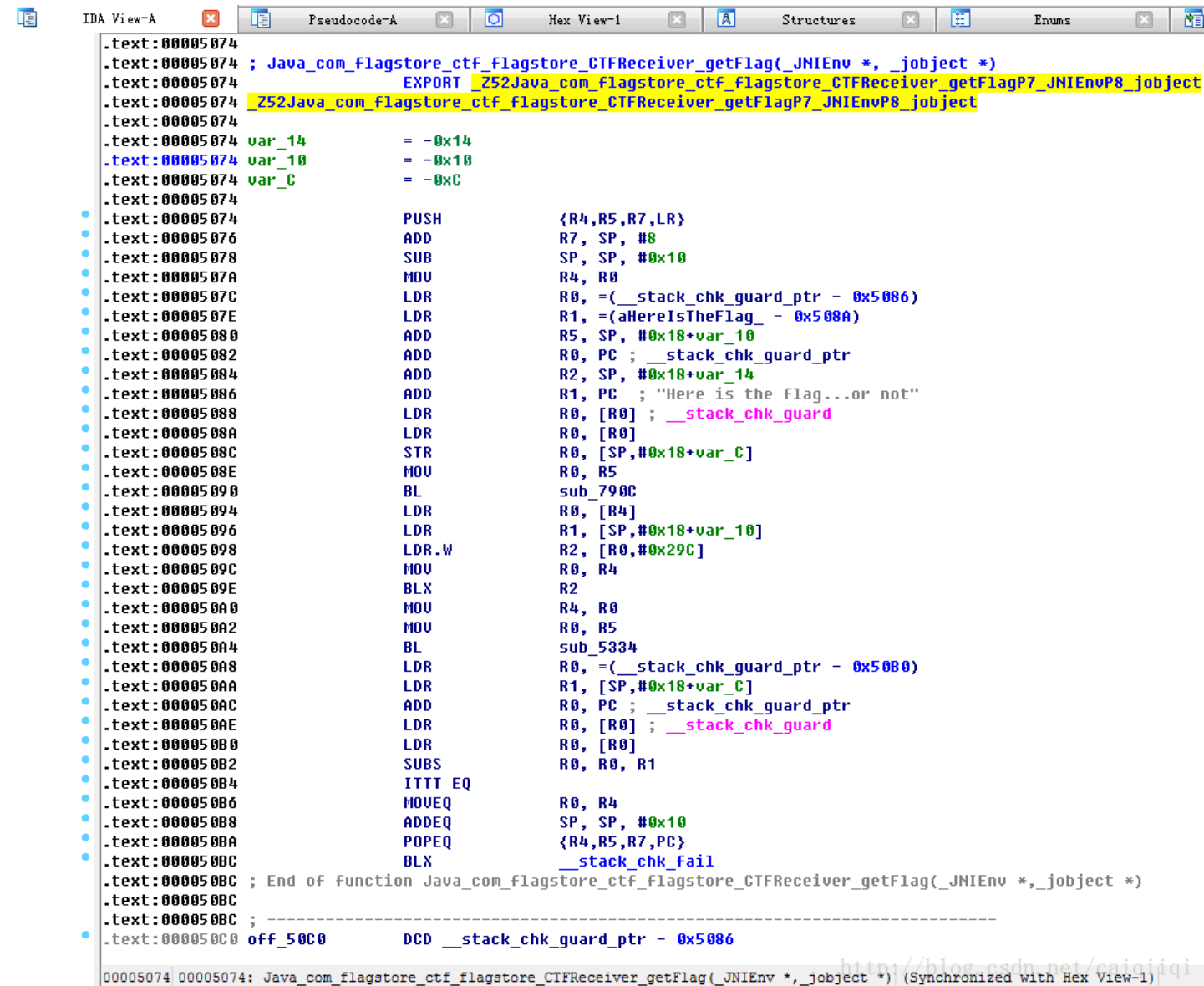
这个特定条件是:

extra中的msg值为: `OpenSesame`。然后调用android的 `am` 工具。不过记住要用root权限, 仅仅在adb shell `am...` 是不能触发的。

```
adb shell
su
am broadcast -a "com.flagstore.ctf.INCOMING_INTENT" --es msg "OpenSesame"
```

然后就可以看到界面跳转了。然后出现一个很大的按钮可以点击，然而点击之后，应用就崩溃了
于是静态分析源码。

把这个 `lib\armeabi-v7a\libnative-lib.so` 文件拖到IDA中去



```
IDA View-A Pseudocode-A Hex View-1 Structures Enums
.text:00005074 ; Java_com_flagstore_ctf_flagstore_CTFReceiver_getFlag(_JNIEnv *, _jobject *)
.text:00005074 EXPORT _Z52Java_com_flagstore_ctf_flagstore_CTFReceiver_getFlagP7_JNIEnvP8_jobject
.text:00005074 _Z52Java_com_flagstore_ctf_flagstore_CTFReceiver_getFlagP7_JNIEnvP8_jobject
.text:00005074 var_14 = -0x14
.text:00005074 var_10 = -0x10
.text:00005074 var_C = -0xC
.text:00005074 PUSH {R4,R5,R7,LR}
.text:00005076 ADD R7, SP, #8
.text:00005078 SUB SP, SP, #0x10
.text:0000507A MOV R4, R0
.text:0000507C LDR R0, =(__stack_chk_guard_ptr - 0x5086)
.text:0000507E LDR R1, =(aHereIsTheFlag_ - 0x508A)
.text:00005080 ADD R5, SP, #0x18+var_10
.text:00005082 ADD R0, PC ; __stack_chk_guard_ptr
.text:00005084 ADD R2, SP, #0x18+var_14
.text:00005086 ADD R1, PC ; "Here is the flag...or not"
.text:00005088 LDR R0, [R0] ; __stack_chk_guard
.text:0000508A LDR R0, [R0]
.text:0000508C STR R0, [SP,#0x18+var_C]
.text:0000508E MOV R0, R5
.text:00005090 BL sub_790C
.text:00005094 LDR R0, [R4]
.text:00005096 LDR R1, [SP,#0x18+var_10]
.text:00005098 LDR.W R2, [R0,#0x29C]
.text:0000509C MOV R0, R4
.text:0000509E BLX R2
.text:000050A0 MOV R4, R0
.text:000050A2 MOV R0, R5
.text:000050A4 BL sub_5334
.text:000050A8 LDR R0, =(__stack_chk_guard_ptr - 0x5080)
.text:000050AA LDR R1, [SP,#0x18+var_C]
.text:000050AC ADD R0, PC ; __stack_chk_guard_ptr
.text:000050AE LDR R0, [R0] ; __stack_chk_guard
.text:000050B0 LDR R0, [R0]
.text:000050B2 SUBS R0, R0, R1
.text:000050B4 ITTT EQ
.text:000050B6 MOVEQ R0, R4
.text:000050B8 ADDEQ SP, SP, #0x10
.text:000050BA POPEQ {R4,R5,R7,PC}
.text:000050BC BLX __stack_chk_fail
.text:000050BC ; End of function Java_com_flagstore_ctf_flagstore_CTFReceiver_getFlag(_JNIEnv *,_jobject *)
.text:000050BC ; -----
.text:000050C0 off_50C0 DCD __stack_chk_guard_ptr - 0x5086
00005074 00005074: Java_com_flagstore_ctf_flagstore_CTFReceiver_getFlag(_JNIEnv *,_jobject *) (Synchronized with Hex View-1)
```

找到 Exports 就可以找到一个函数 `getPhrase()`

然后双击，进入该函数，然后按F5是不行的，搜了一下才知道这时候需要点击右键，然后 `Create Function`。这样再按F5就可以得到C代码了。



```
IDA View-A Pseudocode-A Hex View-1 Structures Enums
1 int __fastcall Java_com_flagstore_ctf_flagstore_CTFReceiver_getPhrase(int a1, int a2, int a3, int a4, int a5)
2 {
3     int v5; // r8@1
4     int v6; // r4@1
5     int v7; // r6@1
6     const char *u8; // r10@1
7     const char *u9; // r11@1
8     char *src; // ST04_4@1
9     int v11; // r0@2
10    int result; // r0@3
11    char v13[76]; // [sp+8h] [bp-145h]@2
12    char v14; // [sp+57h] [bp-F9h]@3
13    int dest; // [sp+58h] [bp-F8h]@1
14    signed int v16; // [sp+5Ch] [bp-F4h]@1
```

```

15 signed int v17; // [sp+60h] [bp-F0h]@1
16 signed int v18; // [sp+64h] [bp-ECh]@1
17 signed int v19; // [sp+68h] [bp-E8h]@1
18 signed int v20; // [sp+6Ch] [bp-E4h]@1
19 char v21; // [sp+70h] [bp-E0h]@1
20 char v22[77]; // [sp+A6h] [bp-AAh]@1
21 char v23[77]; // [sp+F3h] [bp-5Dh]@1
22 int v24; // [sp+140h] [bp-10h]@1
23
24 v5 = a1;
25 v6 = a4;
26 v7 = 0;
27 v24 = __stack_chk_guard;
28 v8 = (const char *)((*int (**)(void))(*_DWORD *)a1 + 676))();
29 v9 = (const char *)((*int (__fastcall **)(int, int, _DWORD))(*_DWORD *)v5 + 676))(v5, v6, 0);
30 src = (char *)((*int (__fastcall **)(int, int, _DWORD))(*_DWORD *)v5 + 676))(v5, a5, 0);
31 dest = 1216233792;
32 v16 = 1313163366;
33 v17 = 2003059012;
34 v18 = 1700753218;
35 v19 = 1349085773;
36 v20 = 1987932780;
37 v21 = aAHfhendadwbo_e[24];
38 strcat((char *)&dest, v8, 0x33u);
39 strncpy(v23, v9, 0x4Cu);
40 strncpy(v22, src, 0x4Cu);
41 do
42 {
43     v11 = (unsigned __int8)v23[v7] ^ (unsigned __int8)v22[v7] ^ *((_BYTE *)&dest + v7);
44     v13[v7] = v11;
45     printf("%c\n", (unsigned __int8)v11);
46     ++v7;
47 }

```

00004F6C Java_com_flagstore_ctf_flagstore_CTFReceiver_getPhrase:1

<http://blog.csdn.net/caiqi1qi>

```

23
24 v5 = a1;
25 v6 = a4;
26 v7 = 0;
27 v24 = __stack_chk_guard;
28 v8 = (const char *)((*int (**)(void))(*_DWORD *)a1 + 676))();
29 v9 = (const char *)((*int (__fastcall **)(int, int, _DWORD))(*_DWORD *)v5 + 676))(v5, v6, 0);
30 src = (char *)((*int (__fastcall **)(int, int, _DWORD))(*_DWORD *)v5 + 676))(v5, a5, 0);
31 dest = 1216233792;
32 v16 = 1313163366;
33 v17 = 2003059012;
34 v18 = 1700753218;
35 v19 = 1349085773;
36 v20 = 1987932780;
37 v21 = aAHfhendadwbo_e[24];
38 strcat((char *)&dest, v8, 0x33u);
39 strncpy(v23, v9, 0x4Cu);
40 strncpy(v22, src, 0x4Cu);
41 do
42 {
43     v11 = (unsigned __int8)v23[v7] ^ (unsigned __int8)v22[v7] ^ *((_BYTE *)&dest + v7);
44     v13[v7] = v11;
45     printf("%c\n", (unsigned __int8)v11);
46     ++v7;
47 }
48 while ( v7 != 76 );
49 v14 = 0;
50 printf("Here is your Reply: %s", v13);
51 result = ((*int (__fastcall **)(int, char *))(*_DWORD *)v5 + 668))(v5, v13);
52 if ( __stack_chk_guard != v24 )
53     __stack_chk_fail(result, __stack_chk_guard - v24);
54 return result;
55 }

```

00004F6C Java_com_flagstore_ctf_flagstore_CTFReceiver_getPhrase:9

<http://blog.csdn.net/caiqi1qi>

其中 `__stack_chk_guard` 和 `__stack_chk_fail()` 是GCC的堆栈保护机制，其中 `__stack_chk_guard` 叫作CANARY(金丝雀)值。

我们发现这一随机值是放在了函数的局部变量和保存的指令指针（译注：此处指返回地址和EBP）之间。这个值被称作金丝雀(“canary”)值，指的是矿工曾利用金丝雀来确认是否有气体泄漏，如果金丝雀因为气体泄漏而中毒死亡，可以给矿工预警。

<http://www.freebuf.com/articles/system/24177.html>

在#27行中，将 `_stack_chk_guard` 保存在v24中，然后再在函数返回之前，#52，#53行将之前保存的值与现在的 `_stack_chk_guard` 比较，若两者不一致，金丝雀 (canary) 的值被修改了，栈溢出发生了，保存的指令指针可能也被修改了，因此不能安全返回，则执行 `_stack_chk_fail()`，然后会丢出一个错误，退出进程。

IDA小技巧

注意这里的dest, v16, v17, v18, v19, v20等是十进制，可以在IDA里选择将其改成16进制或者字符串。

```
31  dest = 'H^AQ';
32  v16 = 'NEHF';
33  v17 = 'vdAD';
34  v18 = 'e_oB';
35  v19 = 'PijM';
36  v20 = 0x767D726C;
37  v21 = aAHfhendadwbo_e[24];
38  strncat((char *)&dest, v8, 0x33u);
39  strncpy(v23, v9, 0x4Cu);
40  strncpy(v22, src, 0x4Cu);
41  do
```

关于 `strncat`

头文件: `#include <string.h>`

`strncat()`用于将n个字符追加到字符串的结尾，其原型为：

```
char * strncat(char *dest, const char *src, size_t n);
```

`strncat()`将会从字符串src的开头拷贝n个字符到dest字符串尾部，dest要有足够的空间来容纳要拷贝的字符串。如果n大于字符串src的长度，那么仅将src全部追加到dest的尾部。

`strncat()`会将dest字符串最后的'\0'覆盖掉，字符追加完成后，再追加'\0'。

返回字符串dest。

关于 `strncpy`

头文件: `#include <string.h>`

`strncpy()`用来复制字符串的前n个字符，其原型为：

```
char * strncpy(char *dest, const char *src, size_t n);
```

【参数说明】dest 为目标字符串指针，src 为源字符串指针。

【返回值】返回字符串dest。

关于跨平台的移植性

int类型比较特殊，具体的字节数同机器字长和编译器有关。如果要保证移植性，尽量用 `__int16` `__int32` `__int64`吧。`__int16`、`__int32`这种数据类型在所有平台下都分配相同的字节。所以在移植上不存在问题。

建议：在代码中尽量避免使用int类型，根据不同的需要可以用short,long,unsigned int 等代替。

<http://www.cppblog.com/xyjzsh/archive/2010/10/20/130554.html>

怪不得很多IDA出来的C有很多 `__int8`，`__int16`，`__int32`等等。

然后顺便反编译了 `getFlag`

```
IDA View-A Pseudocode-B Pseudocode-A Hex View-1
1 int __fastcall Java_com_flagstore_ctf_flagstore_CTFReceiver_getFlag(int a1)
2 {
3     int v1; // r4@1
4     int v2; // r4@1
5     char v4; // [sp+4h] [bp-14h]@1
6     int v5; // [sp+8h] [bp-10h]@1
7     int v6; // [sp+Ch] [bp-Ch]@1
8
9     v1 = a1;
10    v6 = _stack_chk_guard;
11    sub_798C(&v5, "Here is the flag...or not", &v4);
12    v2 = (*(int (__fastcall **)(int, int))(*(_DWORD *)v1 + 668))(v1, v5);
13    sub_5334(&v5);
14    if ( _stack_chk_guard != v6 )
15        _stack_chk_fail(_stack_chk_guard - v6, v6);
16    return v2;
17 }
```

<http://blog.csdn.net/caiqi1qi>