看雪3万课程笔记-FRIDA高级API实用方法：Frida Hook Native层

原创

kfyzjd2008 于 2022-03-02 14:39:36 发布 186 收藏
分类专栏： 安卓 文章标签： frida frida hook 看雪三万 fridahook 安卓逆向

本文链接：https://blog.csdn.net/kfyzjd2008/article/details/123211197

一、使用工具：

apk：攻防世界中CTF题，安装后图表显示黑客精神 xman.apk

工具：IDA，jadx

二、知识点：

1、Module.findBaseAddress("libmyjni.so");//获取基地址

2、Module.findExportByName("libmyjni.so","n2");//获取函数地址

3、Interceptor.attach //根据地址 hook native层函数

4、Process.findModuleByName("libart.so");//返回一个模块

5、enumerateSymbols //枚举模块符号（API）

三、分析：

1、首先jadx打开app分析java代码

```
MyApp myApp = (MyApp) getApplication();
int m = MyApp.m;
if (m == 0) {
    str2 = "未注册";
} else if (m == 1) {
    str2 = "已注册";
} else {
    str2 = "已混乱";
}
setTitle("Xman" + str2);
this.btn1 = (Button) findViewById(R.id.button1);
this.btn1.setOnClickListener(new View.OnClickListener() {
    /* class com.gdufs.xman.MainActivity.AnonymousClass1 */

    public void onClick(View v) {
        MyApp myApp = (MyApp) MainActivity.this.getApplication();
        if (MyApp.m == 0) {
            MainActivity.this.doRegister();
            return;
        }
        ((MyApp) MainActivity.this.getApplication()).work();
        Toast.makeText(MainActivity.this.getApplicationContext(), MainActivity.workString, 0).show();
    }
});
}
```

关键判断点m的值
为1则通过
为0则调用注册

```java
public class MyApp extends Application {
    public static int m = 0;

    public native void initSN();

    public native void saveSN(String str);

    public native void work();

    static {
        System.loadLibrary("myjni");
    }

    public void onCreate() {
        initSN();
        Log.d("com.gdufs.xman m=", String.valueOf(m));
        super.onCreate();
    }
}
```


从上图可以看出关键判断点为MyApp中m的值，从图二可以看出MyApp的方法注册在native层。

有initSN、saveSN、work三个函数。

2、解压apk用IDA打开libmyjni.so文件。导出函数中不能直接搜索到上面三个函数，我们直接双击JNI_ONLOAD，F5查看伪代码

```
jint JNI_OnLoad(JavaVM *vm, void *reserved)
{
  if ( !(*vm)->GetEnv(vm, (void **)&g_env, 65542) )
  {
    j___android_log_print(2, "com.gdufs.xman", "JNI_OnLoad()");
    native_class = (*(int (__fastcall **)(int, const char *))(*(_DWORD *)g_env + 24))(g_env, "com/gdufs/xman/MyApp");
    if ( !(*(int (__fastcall **)(int, int, char **, int))(*(_DWORD *)g_env + 860))(g_env, native_class, off_5004, 3) )
    {
      j___android_log_print(2, "com.gdufs.xman", "RegisterNatives() --> nativeMethod() ok");
      return 65542;
    }
    j___android_log_print(6, "com.gdufs.xman", "RegisterNatives() --> nativeMethod() failed");
  }
  return -1;
}
```

GetEnv第二个参数为JNIEnv* 我们手动改一下


```
jint JNI_OnLoad(JavaVM *vm, void *reserved)
{
  if ( !(*vm)->GetEnv(vm, (void **)&g_env, 65542) )
  {
    j___android_log_print(2, "com.gdufs.xman", "JNI_OnLoad()");
    native_class = (int)(*g_env)->FindClass(g_env, "com/gdufs/xman/MyApp");
    if ( !(*g_env)->RegisterNatives(g_env, (jclass)native_class, (const JNINativeMethod *)off_5004, 3) )
    {
      j___android_log_print(2, "com.gdufs.xman", "RegisterNatives() --> nativeMethod() ok");
      return 65542;
    }
    j___android_log_print(6, "com.gdufs.xman", "RegisterNatives() --> nativeMethod() failed");
  }
  return -1;
}
```


当我们手动更改GetEnv中的第二个参数之后，IDA自动识别出了另外两个API，FindClass、RegisterNatives

图中RegisterNatives的第三个参数是注册的方法数组，我们双击进入。

```
.data:00005000                                          ; sub_11BC+4↑o ...
.data:00005001                  DCB    0
.data:00005002                  DCB    0
.data:00005003                  DCB    0
.data:00005004 off_5004         DCD aInitsn             ; DATA XREF: JNI_OnLoad+46↑o
.data:00005004                                          ; JNI_OnLoad+4A↑o ...
.data:00005004                                          ; "initSN"
.data:00005008                  DCD aV                  ; "()V"
.data:0000500C                  DCD n1+1
.data:00005010                  DCD aSavesn             ; "saveSN"
.data:00005014                  DCD aLjavaLangStrin     ; "(Ljava/lang/String;)V"
.data:00005018                  DCD n2+1
.data:0000501C                  DCD aWork               ; "work"
.data:00005020                  DCD aV                  ; "()V"
.data:00005024                  DCD n3+1
.data:00005024 ; .data          ends
.data:00005024
```


此处注册了我们上面说的三个参数。双击进入saveSN我们查看此处代码。

```
1  int __fastcall n2(int a1, int a2, int a3)
2  {
3    int v5; // r7
4    const char *v7; // r6
5    const char *v8; // r5
6    int v9; // r4
7    int v10; // r0
8    char v11; // r2
9    signed int v12; // [sp+8h] [bp-38h]
0    char v13[20]; // [sp+10h] [bp-30h] BYREF
1
2    v5 = j_fopen("/sdcard/reg.dat", "w+");
3    if ( !v5 )
4      return j___android_log_print(3, "com.gdufs.xman", byte_2E5A);
5    strcpy(v13, "W3_arE_whO_we_ARE");
6    v7 = (const char *)(*(int (__fastcall **)(int, int, _DWORD))(*(_DWORD *)a1 + 676))(a1, a3, 0);
7    v8 = v7;
8    v12 = j_strlen(v7);
9    v9 = 2016;
0    while ( 1 )
1    {
2      v10 = v8 - v7;
3      if ( v8 - v7 >= v12 )
4        break;
5      if ( v10 % 3 == 1 )
6      {
7        v9 = (v9 + 5) % 16;
8        v11 = v13[v9 + 1];
9      }
0      else if ( v10 % 3 == 2 )
1      {
2        v9 = (v9 + 7) % 15;
3        v11 = v13[v9 + 2];
4      }
5      else
6      {
7        v9 = (v9 + 3) % 13;
8        v11 = v13[v9 + 3];
9      }
0      *v8++ ^= v11;
1    }
2    j_fputs(v7, v5);
3    return j_fclose(v5);
4  }
```

打开文件

写入文件、关闭文件

我们将函数的第一个参数a1按Y修改为JNIEnv* a1（native层的函数第一个参数都是JNIEnv*）

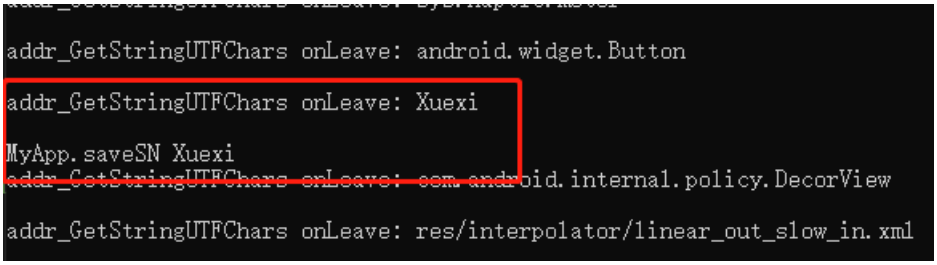
此时识别出来了一个API GetStringUTFChars

此处我们hook这个函数和这个API分别看看他们传入的参数

```
function hook_native(){
    var base_myjni = Module.findBaseAddress("libmyjni.so");//获取基地址

    if (base_myjni){
        var n2 = Module.findExportByName("libmyjni.so","n2");//获取函数地址 函数地址 - 基地址 = 静态偏移地址(ida
        console.log("base_myjni:",base_myjni,"n2:",n2);//thumb模式函数,0x000011F8,实际地址：0xcc2691f9 ,thumb
        Interceptor.attach(n2,{//根据地址 hook native层函数
            onEnter: function(args){
                console.log("n2 onEnter:",args[0],args[1],args[2]);
            },onLeave:function(retval){

            }
        });
    }
}
```

```
function hook_libart(){
    //获取模块地址
    var module_libart = Process.findModuleByName("libart.so");//android常用API都在此文件中
    var symbols = module_libart.enumerateSymbols();//枚举模块中的所有API，返回一个对象数组

    var addr_GetStringUTFChars = null;
    for (var i = 0; i < symbols.length; i++){
        // console.log(symbols[i].name);
        var name = symbols[i].name;
        if (name.indexOf("art") >= 0){
            if((name.indexOf("CheckJNI") == -1) && (name.indexOf("JNI") >= 0)){
                if(name.indexOf("GetStringUTFChars") >= 0){
                    console.log(name);
                    addr_GetStringUTFChars = symbols[i].address;//获取API地址
                }
            }
        }
    }

    if(addr_GetStringUTFChars){
        //hook函数地址
        Interceptor.attach(addr_GetStringUTFChars,{
            onEnter:function(args){

            },onLeave:function(retval){
                console.log("addr_GetStringUTFChars onLeave:",ptr(retval).readCString() + "\r\n");//将地址用
            }
        })
    }
}
```

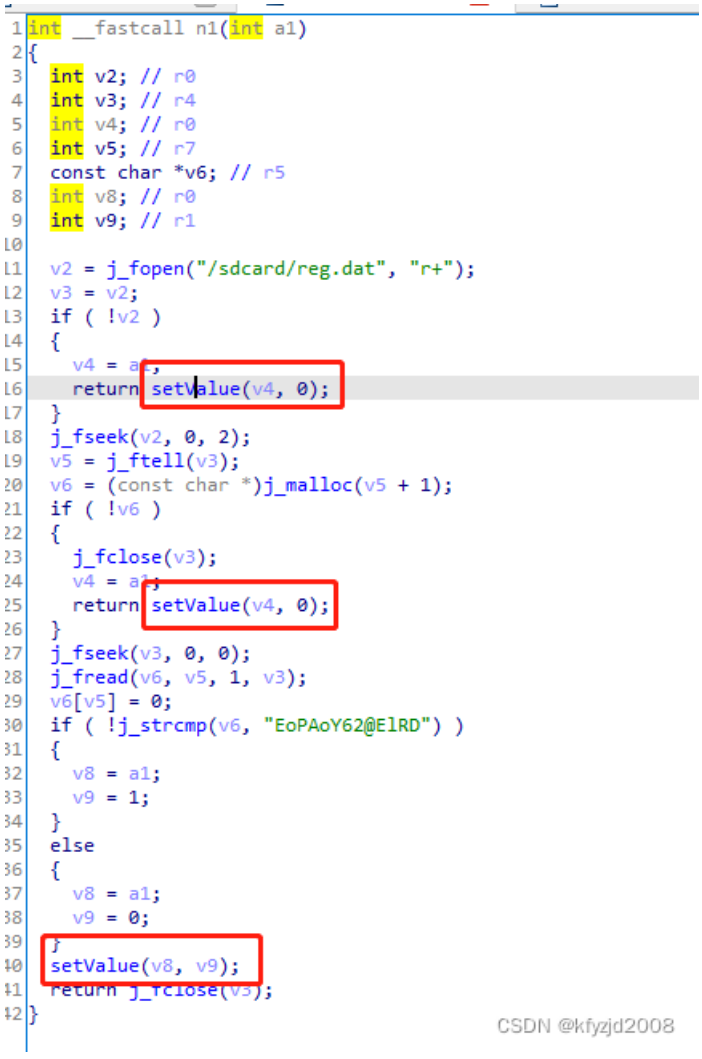
重新输入内容查看控制台输出：



此处值与我们输入的值相同，但是并没有什么卵用，后面还有一点查看堆栈的代码：

```
    if(addr_GetStringUTFChars){
        //hook函数地址
        Interceptor.attach(addr_GetStringUTFChars,{
            onEnter:function(args){
                //查看堆栈调用
                console.log("addr_GetStringUTFChars onEnter called from:\n" +
                    Thread.backtrace(this.context, Backtracer.FUZZY)
                        .map(DebugSymbol.fromAddress).join('\n') + '\n');

            },onLeave:function(retval){
                console.log("addr_GetStringUTFChars onLeave:",ptr(retval).readCString() + "\r\n");//将地址用
            }
        })
    }
}
```

上面的分析基本没啥信息，我们再次来到IDA中的三个函数，这次我们进入initSN中查看代码：



此处有多个setValue函数，我们双击进入查看：

```
int __fastcall setValue(int a1, int a2)
{
  int v4; // r5
  int v5; // r0

  v4 = (*(int (__fastcall **)(int, const char *))(*(_DWORD *)a1 + 24))(a1, "com/gdufs/xman/MyApp");
  v5 = (*(int (__fastcall **)(int, int, const char *, const char *))(*(_DWORD *)a1 + 576))(a1, v4, "m", "I");
  return (*(int (__fastcall **)(int, int, int, int))(*(_DWORD *)a1 + 636))(a1, v4, v5, a2);
}
```

此处将第一个参数改为JNIEnv*

```
int __fastcall setValue(JNIEnv *a1, int a2)
{
  jclass v4; // r5
  jfieldID v5; // r0

  v4 = (*a1)->FindClass(a1, "com/gdufs/xman/MyApp");
  v5 = (*a1)->GetStaticFieldID(a1, v4, "m", "I");
  return ((int (__fastcall *)(JNIEnv *, jclass, jfieldID, int))(*a1)->SetStaticIntField)(a1, v4, v5, a2);
}
```

将第一个参数更改为JNIEnv*

此时自动识别出函数FindClass，GetStaticFieldID，SetStaticIntField。看API名称大致意思为获取一个类，获取静态值，赋值静态值。

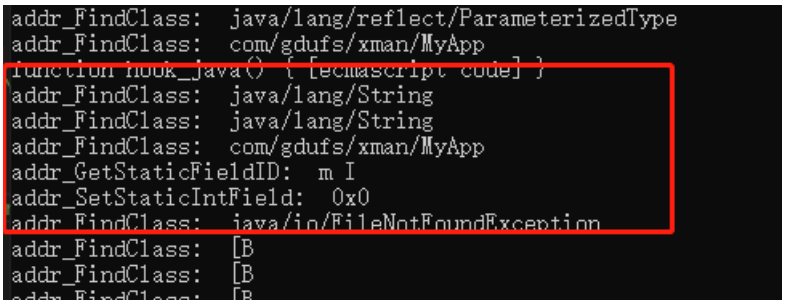
现在我们对这三个函数进行hook看看他们的参数。在 hook_libart 函数中加入：

```
            if(name.indexOf("FindClass") >= 0){
                console.log(name);
                addr_FindClass = symbols[i].address;
            }

            if(name.indexOf("GetStaticFieldID") >= 0){
                console.log(name);
                addr_GetStaticFieldID = symbols[i].address;
            }

            if(name.indexOf("SetStaticIntField") >= 0){
                console.log(name);
                addr_SetStaticIntField = symbols[i].address;
            }
```

循环后加入

```
    if(addr_GetStringUTFChars){
        //hook函数地址
        Interceptor.attach(addr_GetStringUTFChars,{
            onEnter:function(args){
                //查看堆栈调用
                // console.log("addr_GetStringUTFChars onEnter called from:\n" +
                //     Thread.backtrace(this.context, Backtracer.FUZZY)
                //         .map(DebugSymbol.fromAddress).join('\n') + '\n');

            },onLeave:function(retval){
                // console.log("addr_GetStringUTFChars onLeave:",ptr(retval).readCString() + "\r\n");
            }
        })
    }

    if(addr_FindClass){
        //hook函数地址
        Interceptor.attach(addr_FindClass,{
            onEnter:function(args){
                console.log("addr_FindClass: ",ptr(args[1]).readCString());
            },onLeave:function(retval){

            }
        })
    }

    if(addr_GetStaticFieldID){
        //hook函数地址
        Interceptor.attach(addr_GetStaticFieldID,{
            onEnter:function(args){
                console.log("addr_GetStaticFieldID: ",ptr(args[2]).readCString(),ptr(args[3]).readCString()
            },onLeave:function(retval){

            }
        })
    }

    if(addr_SetStaticIntField){
        //hook函数地址
        Interceptor.attach(addr_SetStaticIntField,{
            onEnter:function(args){
                console.log("addr_SetStaticIntField: ");
                console.log("addr_SetStaticIntField: ",args[3]);
            },onLeave:function(retval){

            }
        })
    }
}
```

再次执行：



从图中我们可以看到m被赋值了0

此时视频给出了两个解决方案：

方案1：直接更改/sdcard/reg.dat文件内容为："EoPAoY62@ElRD"

方案2：使用frida代码写入/sdcard/reg.dat文件内容为"EoPAoY62@ElRD"

```
function write_reg_dat(){
    var file = new File("/sdcard/reg.dat","w");
    file.write("EoPAoY62@ElRD");
    file.flush();
    file.close();
}
```

但是这两种方法我测试好像不行。不知道为什么。

正确的输入值应该是：201608Am!2333