XCTF_MOBILE11_黑客精神

原创

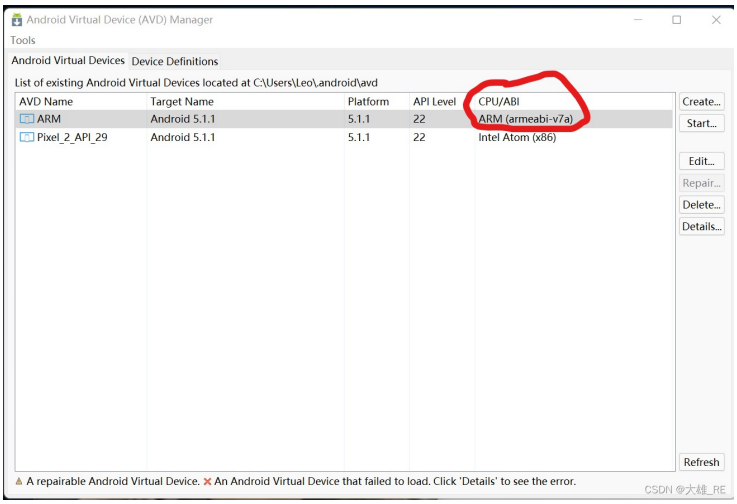
大雄_RE 于 2022-02-21 21:12:51 发布 1237 收藏 2
分类专栏： CTF 文章标签： android 逆向 ctf

本文链接：https://blog.csdn.net/shadow20080578/article/details/123054321

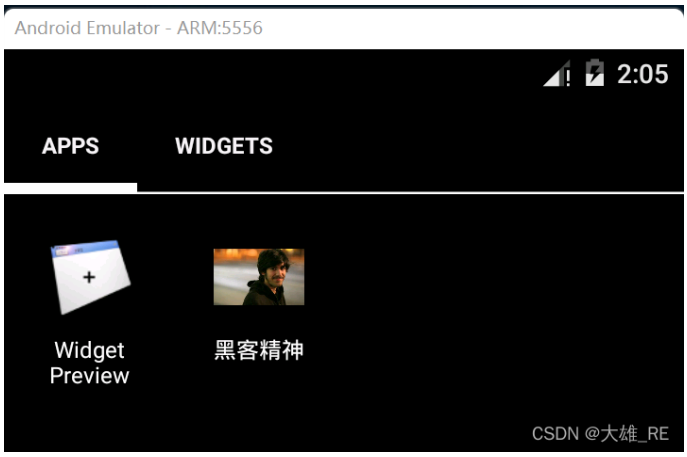
初见

附件为一个apk，先到模拟器中运行一下。

必须选择CPU为ARM的模拟器，因为app里面的native库只提供了ARM指令集的版本，没有提供x86指令集的版本：

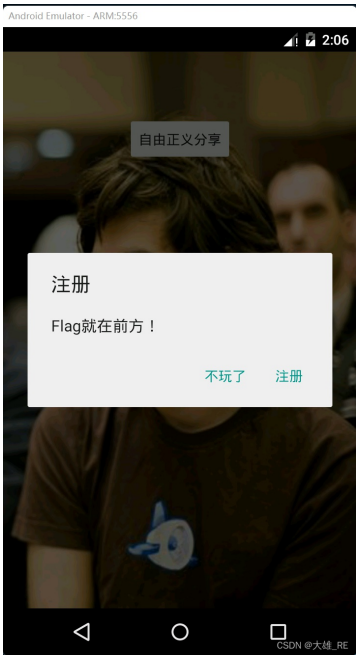


模拟器启动后，将apk拖到模拟器中进行安装，安装好后，列表中app的图标是一个骚年：



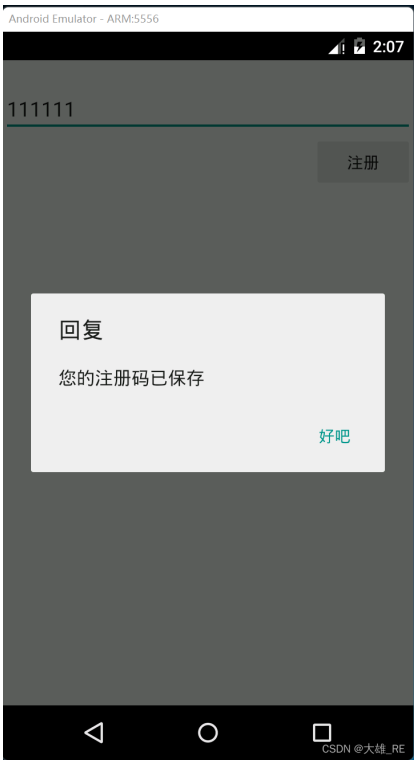
运行app，又见一个骚年：

除了一个按钮啥也没有，点击这个按钮，弹出一个两按钮的对话框：



点击“不玩了”，app直接退出。

点击“注册”，进入新的界面。在新界面中，可以输入一串字符串，并有一个注册按钮。随便输入一个串，点击注册，弹框显示“您的注册码已保存”：



弹框后，进程立即退出。从行为上暂时看不出这个注册码的作用，反编译看看代码。

MainActivity分析

使用jadx打开apk，先看一下MainActivity的代码，在onCreate函数内可以找到，“自由定义分享”按钮的响应函数：

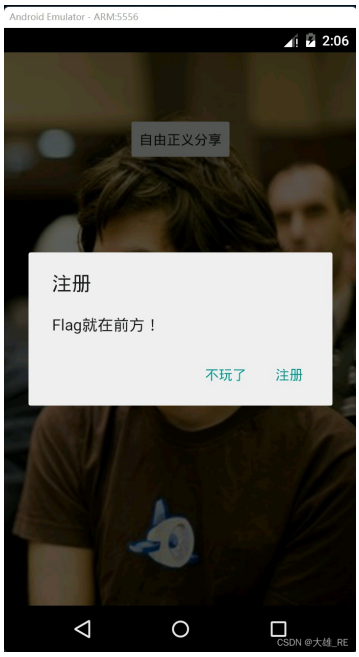
```
this.btn1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                MyApp myApp2 = (MyApp) MainActivity.this.getApplication();
                if (MyApp.m == 0) {
                    MainActivity.this.doRegister();
                    return;
                }
                ((MyApp) MainActivity.this.getApplication()).work();
                Toast.makeText(MainActivity.this.getApplicationContext(), MainActivity.workString, 0).show(
            }
});
```

核心就是，MyApp.m为空时，调用doRegister()。随后调用work()。

先看看doRegister()函数：

```
public void doRegister() {
    new AlertDialog.Builder(this).setTitle("注册").setMessage("Flag就在前方！").setPositiveButton("注册",
        public void onClick(DialogInterface dialog, int which) {
            Intent intent = new Intent();
            intent.setComponent(new ComponentName(BuildConfig.APPLICATION_ID, "com.gdufs.xman.RegActivi
            MainActivity.this.startActivity(intent);
            MainActivity.this.finish();
        }
    }).setNegativeButton("不玩了", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            Process.killProcess(Process.myPid());
        }
    }).show();
}
```

可见，这个doRegister()函数就是弹出下图对话框并处理按钮点击：



点击“不玩了”，直接killProcess。

点击“注册”，以类"com.gdufs.xman.RegActivity"创建新Activity并启动。

总结来看就是：

如果MyApp.m为空，创建RegActivity进行注册，使得MyApp.m不为空。然后调用work()。

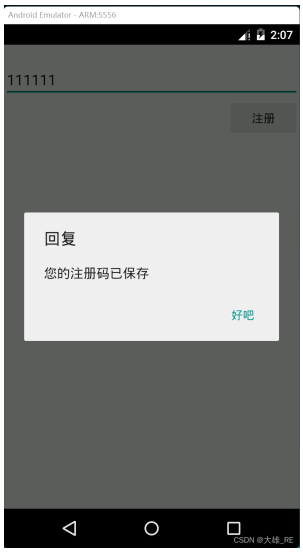
如果MyApp.m不为空，直接调用work()。

而这个work是个native函数：

```
public native void work();
```

RegActivity

RegActivity对应如下界面：



重点在“注册”按钮的响应函数：

```
public void onClick(View v) {
     String sn = RegActivity.this.edit_sn.getText().toString().trim();
     if (sn == null || sn.length() == 0) {
           Toast.makeText(RegActivity.this, "您的输入为空", 0).show();
           return;
     }
     ((MyApp) RegActivity.this.getApplication()).saveSN(sn);
     new AlertDialog.Builder(RegActivity.this).setTitle("回复").setMessage("您的注册码已保存").setPositiveButt
            public void onClick(DialogInterface dialog, int which) {
                  Process.killProcess(Process.myPid());
            }
     }).show();
}
```

可见，点击“注册”按钮后：

1. 调用saveSN()函数进行注册
2. 弹框提示“您的注册码已保存”，如上图
3. killProcess

到这里，已经分析出来的逻辑是：

如果MyApp.m为空，调用saveSN()进行注册，然后调用work()。

如果MyApp.m不为空，直接调用work()。

这个saveSN()也是native函数：

```
public native void saveSN(String str);
```

下面我们就分析分析这个两个native函数。

native库分析

将apk后缀名改为zip并解压，在解压后的lib目录下能得到native库文件：libmyjni.so。

用IDA加载libmyjni.so。

但在导出函数中没有找到work函数和saveSN函数。但能看到导出了JNI_OnLoad函数，这让人联想起JNI函数的动态注册方式：

利用结构体 JNINativeMethod 数组记录 java 方法与 JNI 函数的对应关系；

实现 JNI_OnLoad 方法，在加载动态库后，执行动态注册；

调用 FindClass 方法，获取 java 对象；

调用 RegisterNatives 方法，传入 java 对象，以及 JNINativeMethod 数组，以及注册数目完成注册；

这和该题的JNI_OnLoad函数过程一致：

```
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/xma
    if ( !(*(int (__fastcall **)(int, int, char **, int))(*(_DWORD *)g_env + 860))(g_env, native_class, off
    {
      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;
}
```

可见，off_5004就是java函数和JNI函数的对应关系，据此得到对应关系为：

- InitSN----n1
- SaveSN----n2
- Work----n3

下面分析native函数。

SaveSN

核心反编译代码为：

```
int __fastcall n2(int a1, int a2, int a3)
{
  v5 = j_fopen("/sdcard/reg.dat", "w+");
  strcpy(v13, "W3_arE_whO_we_ARE");
  v7 = input_string;
  v8 = v7;
  v12 = j_strlen(v7);
  v9 = 2016;
  while ( 1 )
  {
    v10 = v8 - v7;
    if ( v8 - v7 >= v12 )
      break;
    if ( v10 % 3 == 1 )
    {
      v9 = (v9 + 5) % 16;
      v11 = v13[v9 + 1];
    }
    else if ( v10 % 3 == 2 )
    {
      v9 = (v9 + 7) % 15;
      v11 = v13[v9 + 2];
    }
    else
    {
      v9 = (v9 + 3) % 13;
      v11 = v13[v9 + 3];
    }
    *v8++ ^= v11;
  }
  j_fputs(v7, v5);
  return j_fclose(v5);
}
```

这里的伪代码是我简化之后的，方便理解。这个里核心逻辑为：

1. 在循环中，依据"W3_arE_whO_we_ARE"对输入字符串进行变形
2. 将变形后的字符串存入文件"/sdcard/reg.dat"

InitSN

核心反编译代码为：

```
int __fastcall n1(int a1)
{
  v2 = j_fopen("/sdcard/reg.dat", "r+");
  j_fread(v6, v5, 1, v2);
  if ( !j_strcmp(v6, "EoPAoY62@ElRD") )
  {
    v8 = a1;
    v9 = 1;
  }
  else
  {
    v8 = a1;
    v9 = 0;
  }
  return j_fclose(v3);
}
```

这里的伪代码也是我简化过得，核心功能逻辑为：

1. 读"/sdcard/reg.dat"文件的内容
2. 比较内容是不是"EoPAoY62@ElRD"

到这里就清楚了，这道题就是要我们的输入字符串，在SaveSN中经过变形后为"EoPAoY62@ElRD"。

解题

SaveSN中对输入字符串进行变形的逻辑也很简单：

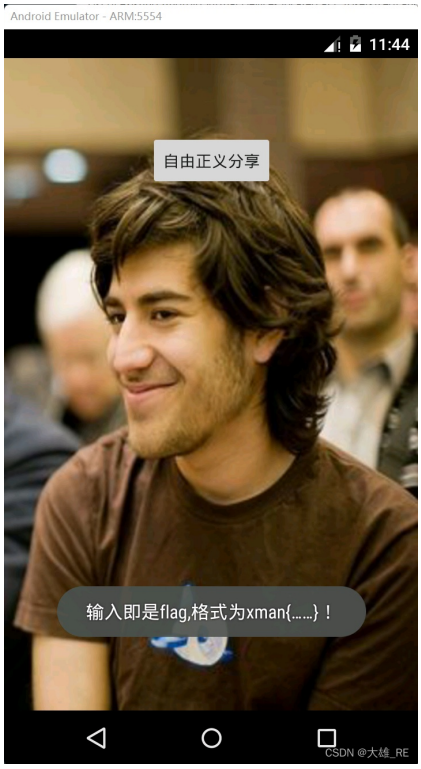
1. 取输入字符串里的每个字符a
2. 根据该字符在输入字符串里的下标，计算出一个值i
3. 取"W3_arE_whO_we_ARE"中下标为i的字符b
4. 计算a异或b的值，写入文件

根据变形过程，使用python实现解密代码：

```
seedstr = bytes('W3_arE_whO_we_ARE','utf-8')
seed = 2016;
result = bytes('EoPAoY62@ElRD','utf-8')
for i in range(13):
 if(i % 3 == 1):
  seed = (seed + 5) % 16
  tmp = seedstr[seed + 1]
  tmp = tmp ^ result[i]
  print(chr(tmp), end = '')
 elif(i % 3 == 2):
  seed = (seed + 7) % 15
  tmp = seedstr[seed + 2]
  tmp = tmp ^ result[i]
  print(chr(tmp), end = '')
 else:
  seed = (seed + 3) % 13
  tmp = seedstr[seed + 3]
  tmp = tmp ^ result[i]
  print(chr(tmp), end = '')
```

得到字符串：201608Am!2333。

输入该字符串进行注册，之后再次打开app，点击“自由正义分享”，提示：



所以flag为：xman{201608Am!2333}

欢迎关注我的微博：大雄_RE。专注软件逆向，分享最新的好文章、好工具，追踪行业大佬的研究成果。