Android安全攻防战，反编译与混淆技术完全解析（上）

2016-03-27 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：郭霖
> 链接：http://blog.csdn.net/guolin_blog/article/details/49738023

之前一直有犹豫过要不要写这篇文章，毕竟去反编译人家的程序并不是什么值得骄傲的事情。不过单纯从技术角度上来讲，掌握反编译功能确实是一项非常有用的技能，可能平常不太会用得到，但是一旦真的需要用到的了，而你却不会的话，那就非常头疼了。另外既然别人可以反编译程序，我们当然有理由应该对程序进行一定程度的保护，因此代码混淆也是我们必须要掌握的一项技术。那么最近的两篇文章我们就围绕反编译和混淆这两个主题来进行一次完全解析。

反编译

我们都知道，Android程序打完包之后得到的是一个APK文件，这个文件是可以直接安装到任何Android手机上的，我们反编译其实也就是对这个APK文件进行反编译。Android的反编译主要又分为两个部分，一个是对代码的反编译，一个是对资源的反编译，我们马上来逐个学习一下。

在开始学习之前，首先我们需要准备一个APK文件，为了尊重所有开发者，我就不拿任何一个市面上的软件来演示了，而是自己写一个Demo用来测试。
这里我希望代码越简单越好，因此我们建立一个新项目，在Activity里加入一个按钮，当点击按钮时弹出一个Toast，就这么简单，代码如下所示：

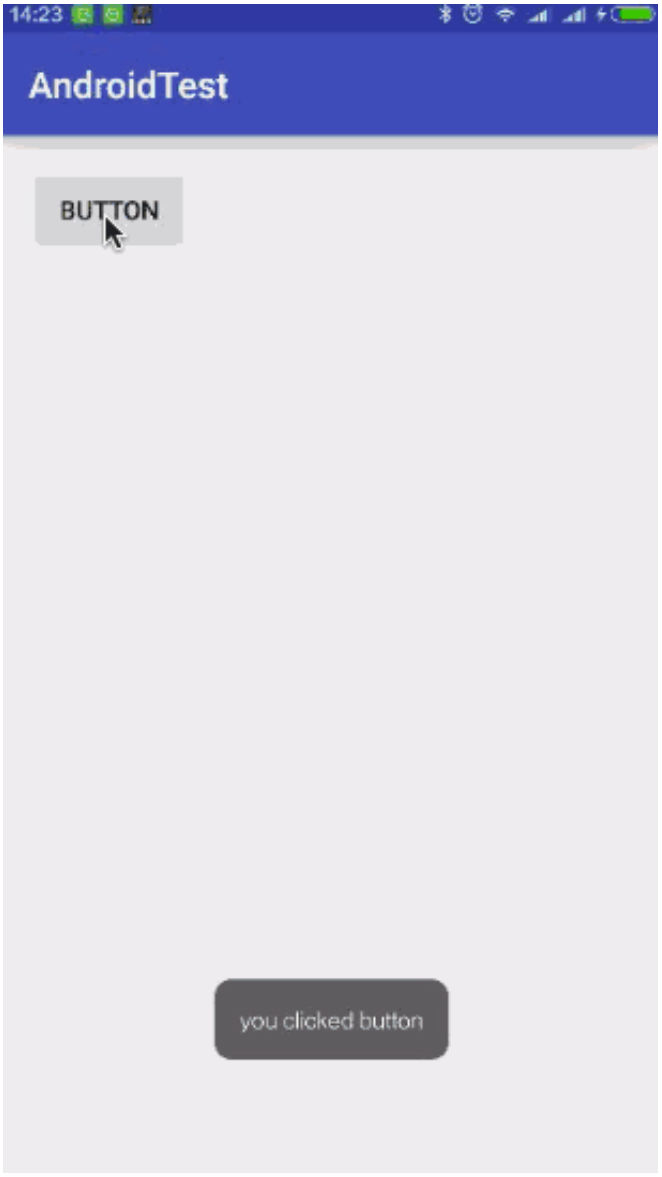
```
public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
            Toast.makeText(MainActivity.this, "you clicked button", Toast.LENGTH_SHORT).show();
        }
      });
  }

}
```

activity_main.xml中的资源如下所示：

```
RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin">

  Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"/>

RelativeLayout>
```

然后我们将代码打成一个APK包，并命名成Demo.apk，再把它安装到手机上，结果如下所示：



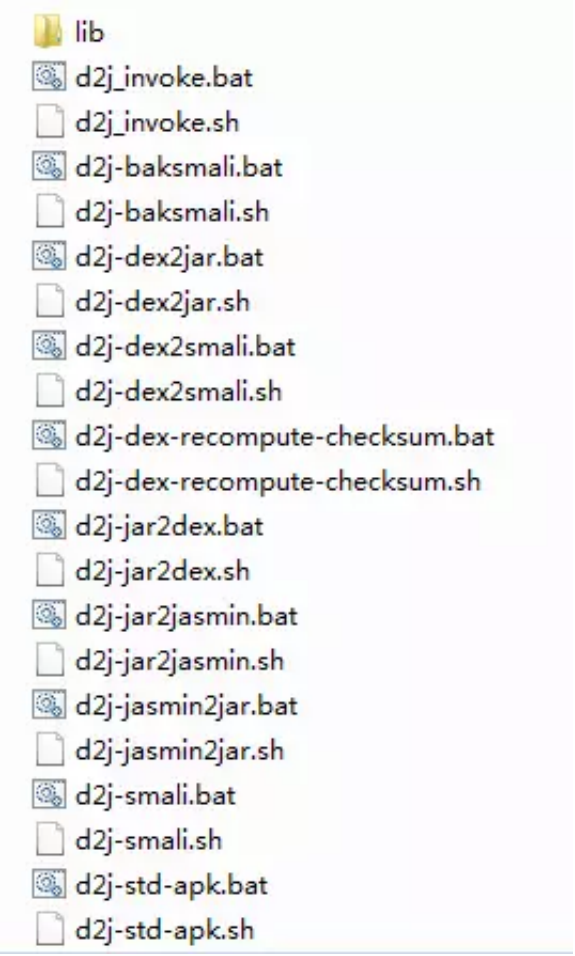
好的，到这里准备工作就已经基本完成了，接下来就让我们开始对这个Demo程序进行反编译吧。

反编译代码

要想将APK文件中的代码反编译出来，我们需要用到以下两款工具：

- dex2jar 这个工具用于将dex文件转换成jar文件下载地址：http://sourceforge.net/projects/dex2jar/files/
- jd-gui 这个工具用于将jar文件转换成java代码下载地址：http://jd.benow.ca/

将这两个工具都下载好并解压，然后我们就开始对Demo程序进行反编译。解压dex2jar压缩包后，你会发现有很多个文件，如下图所示：



其中我们要用到的是d2j-dex2jar.bat这个文件，当然如果你是linux或mac系统的话就要用d2j-dex2jar.sh这个文件。

然后我们将Demo.apk文件也进行解压，如果不知道怎么直接解压的可以先将文件重命名成Demo.zip，然后用解压软件打开。解压之后你会发现里面有一个classes.dex文件，如下图所示：

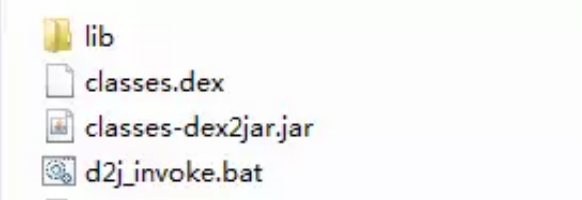
这个classes.dex文件就是存放所有java代码的地方了，我们将它拷贝到dex2jar解压后的目录下，并在cmd中也进入到同样的目录，然后执行：

```
d2j-dex2jar classes.dex
```

执行结果如下图所示：

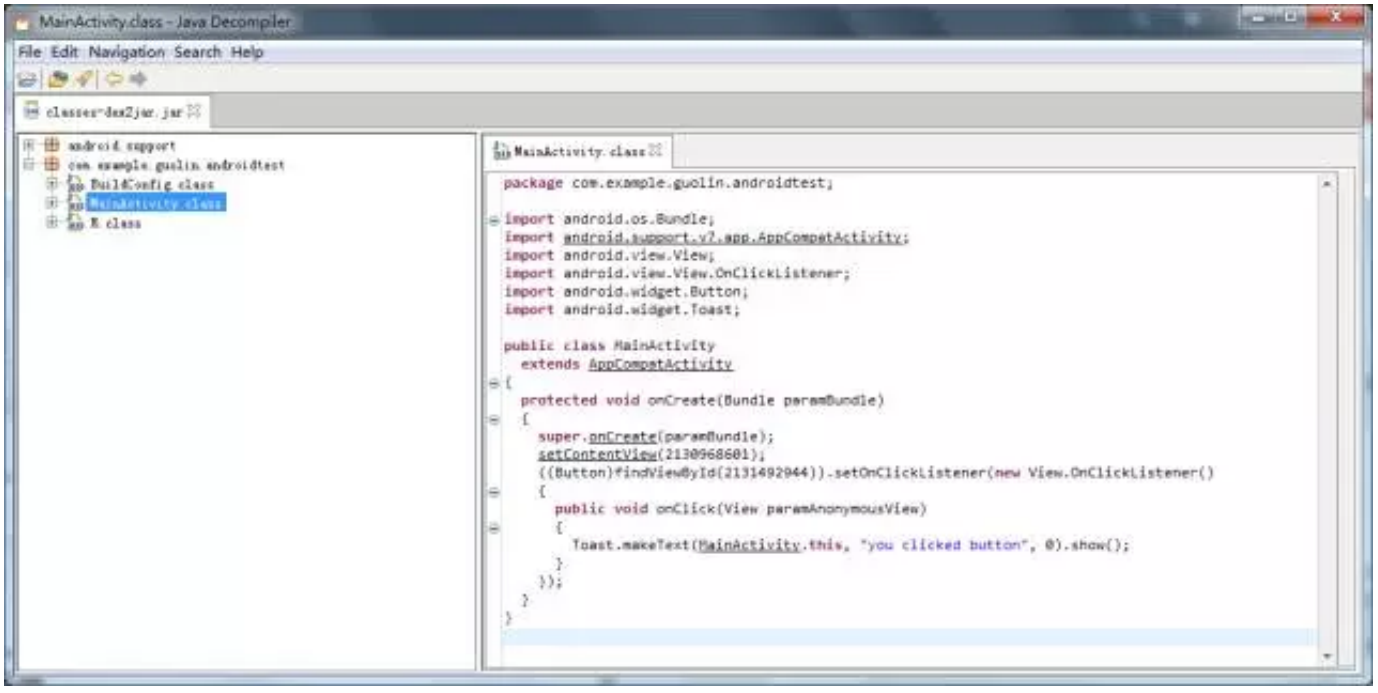
```
E:\googleDownload\dex2jar-2.0>d2j-dex2jar.bat classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar

E:\googleDownload\dex2jar-2.0>_
```

没有报任何错误，这就说明我们已经转换成功了。现在观察dex2jar目录，你会发现多了一个文件，如下图所示：



可以看到，classes-dex2jar.jar这个文件就是我们借助工具之后成功转换出来的jar文件了。但是对于我们而言，jar文件也不是可读的，因此这里还需要再借助一下jd-gui这个工具来将jar文件转换成java代码。

下面就很简单了，使用jd-gui工具打开classes-dex2jar.jar这个文件，结果如下图所示：



OK，由此可见，我们的代码反编译工作已经成功了，MainActivity中的代码非常清晰，基本已经做到了90%以上的还原工作。但是如果想要做到100%的代码还原还是非常有难度的，因为像setContentView()方法传入的参数，其实就是一个资源的id值而已，那么这里反编译也就只能将相应的id值进行还原，而无法变成像R.layout.activity_main这样直观的代码展示。

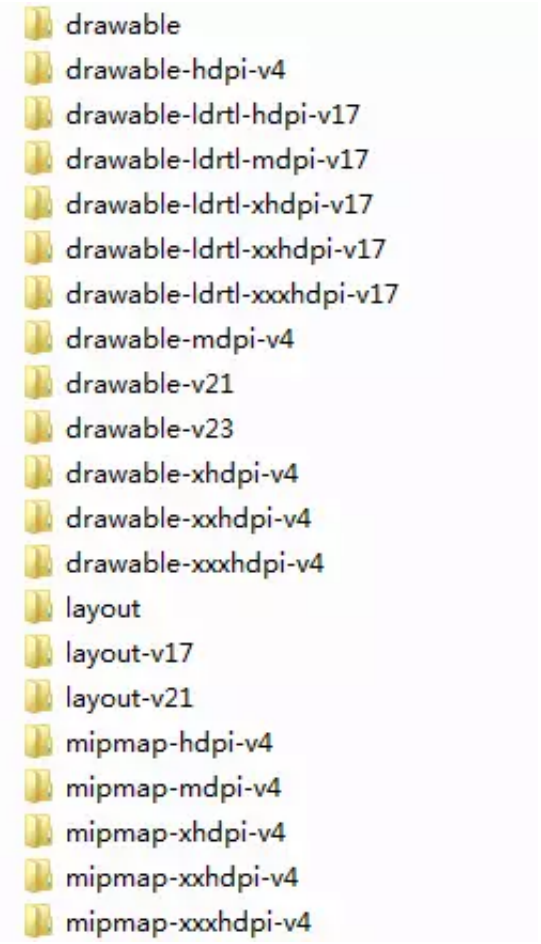
另外，除了MainActivity之外，还有很多其它的代码也被反编译出来了，因为当前项目有引

用support-v4和support-v7的包，这些引用的library也会作为代码的一部分被打包到classes.dex文件当中，因此反编译的时候这些代码也会一起被还原。

好的，学完了反编译代码，接下来我们看一下如何反编译资源。

反编译资源

其实细心的朋友可能已经观察到了，刚才Demo.apk的解压目录当中不是已经有资源文件了吗，有AndroidManifest.xml文件，也有res目录。进入res目录当中，内容如下图所示：



这不是所有资源文件都在这里了么？其实这些资源文件都是在打包的时候被编译过了，我们直接打开的话是看不到明文的，不信的话我们打开AndroidManifest.xml文件来瞧一瞧，内容如下图所示：

可以看到，这代码是完全没法阅读的。当然如果你去打开activity_main.xml看看，结果也不会好到哪儿去：

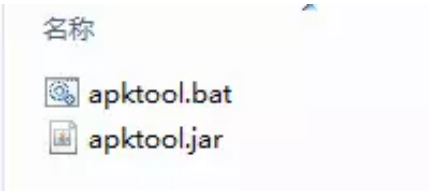
由此可见，直接对APK包进行解压是无法得到它的原始资源文件的，因此我们还需要对资源进行反编译才行。

要想将APK文件中的资源反编译出来，又要用到另外一个工具了：

- apktool 这个工具用于最大幅度地还原APK文件中的9-patch图片、布局、字符串等等一

系列的资源。下载地址：http://ibotpeaches.github.io/Apktool/install/

关于这个工具的下载我还要再补充几句，我们需要的就是apktool.bat和apktool.jar这两个文件。目前apktool.jar的最新版本是2.0.3，这里我就下载最新的了，然后将apktool_2.0.3.jar重命名成apktool.jar，并将它们放到同一个文件夹下就可以了，如下图所示：



接下来的工作就很简单了，我们将Demo.apk拷贝到和这两个文件同样的目录当中，然后cmd也进入到这个目录下，并在cmd中执行如下命令：

```
apktool d Demo.apk
```

其中d是decode的意思，表示我们要对Demo.apk这个文件进行解码。那除了这个基本用法之外，我们还可以再加上一些附加参数来控制decode的更多行为：

- -f 如果目标文件夹已存在，则强制删除现有文件夹（默认如果目标文件夹已存在，则解码失败）。
- -o 指定解码目标文件夹的名称（默认使用APK文件的名字来命名目标文件夹）。
- -s 不反编译dex文件，也就是说classes.dex文件会被保留（默认会将dex文件解码成smali文件）。
- -r 不反编译资源文件，也就是说resources.arsc文件会被保留（默认会将resources.arsc解码成具体的资源文件）。

常用用法就这么多了，那么上述命令的执行结果如下图所示：

```
E:\apktool>apktool d Demo.apk
I: Using Apktool 2.0.3 on Demo.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Administrator\apktool\framework\1.
apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

E:\apktool>_
```

这就说明反编译资源已经成功了。

当然即使你在和我执行一模一样的操作，也有可能会在这里反编译失败，比如说会报如下错误：

```
E:\apktool>apktool d Demo.apk
I: Using Apktool 2.0.3 on Demo.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\Administrator\apktool\framework\1.
apk
I: Regular manifest package...
I: Decoding file-resources...
W: Could not decode attr value, using undecoded value instead: ns=android, name=
color, value=0x0101042c
W: Could not decode attr value, using undecoded value instead: ns=android, name=
color, value=0x0101042c
W: Could not decode attr value, using undecoded value instead: ns=android, name=
color, value=0x0101042b
W: Could not decode attr value, using undecoded value instead: ns=android, name=
color, value=0x01010435
W: Could not decode attr value, using undecoded value instead: ns=android, name=
color, value=0x0101042a
W: Could not decode attr value, using undecoded value instead: ns=android, name=
color, value=0x0101042c
I: Decoding values */* XMLs...
Exception in thread "main" brut.androlib.err.UndefinedResObject: resource spec:
0x01030284
        at brut.androlib.res.data.ResPackage.getResSpec(ResPackage.java:59)
        at brut.androlib.res.data.ResTable.getResSpec(ResTable.java:65)
        at brut.androlib.res.data.ResTable.getResSpec(ResTable.java:61)
        at brut.androlib.res.data.value.ResReferenceValue.getReferent(ResReferen
ceValue.java:57)
        at brut.androlib.res.data.value.ResReferenceValue.encodeAsResXml(ResRefe
renceValue.java:47)
        at brut.androlib.res.data.value.ResScalarValue.encodeAsResXmlAttr(ResSca
larValue.java:46)
```

出现这个错误的原因很有可能是你之前使用过apktool的老版本进行过反编译操作，然后

apktool就会在你系统的C:UsersAdministratorapktoolframework这个目录下生成一个名字为1.apk的缓存文件，将这个缓存文件删除掉，然后再重新执行反编译命令应该就可以成功了。

现在你会发现在当前目录下多了一个Demo文件夹，这个文件夹中存放的就是反编译的结果了。我们可以打开AndroidManifest.xml来瞧一瞧，如下图所示：

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.guolin.androidtest" platformBuildVersionCode="23"
platformBuildVersionName="6.0-2438415">
    <application android:allowBackup="true" android:debuggable="true"
    android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
    android:supportsRtl="true" android:theme="@style/AppTheme">
        <activity android:name="com.example.guolin.androidtest.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>

```

怎么样？这样就完全能看得懂了吧，然后可以再到res/layout中看一下activity_main.xml文件，如下图所示：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
android:layout_width="fill_parent" android:layout_height="fill_parent"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <Button android:id="@id/button" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:text="Button" />
</RelativeLayout>
```

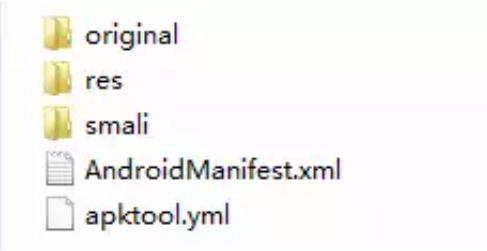
可以看到，activity_main.xml中的内容基本和源代码中的内容是一致的，外层是一个RelativeLayout，里面则是一个Button。你可以再到其它目录中去看一看别的资源，基本上都是可以正常还原的，这样我们就把反编译资源的方法也已经掌握了。

重新打包

那么对于反编译出来的文件夹，我们能不能重新把它打包成APK文件呢？答案是肯定的，只不过我实在想不出有什么义正言辞的理由可以让我们这么做。有的人会说汉化，没错，汉化的方式确实就是将一个APK进行反编译，然后翻译其中的资源再重新打包，但是不管怎么说这仍然是将别人的程序进行破解，所以我并不认为这是什么光荣的事情。那么我们就不去讨

论本身这件事情的对或错，这里只是站在技术的角度来学习一下重新打包的相关知识。

首先我们来看一下通过apktool反编译后的包目录情况，如下图所示：



其中，original文件夹下存放的是未经反编译过、原始的AndroidManifest.xml文件，res文件夹下存放的是反编译出来的所有资源，smali文件夹下存放的是反编译出来的所有代码，AndroidManifest.xml则是经过反编译还原后的manifest文件。这里值得一提的是smali文件夹，如果你进入到这个文件夹中你会发现它的目录结构和我们源码中src的目录结构是几乎一样的，主要的区别就是所有的java文件都变成了smali文件。smali文件其实也是真正的源代码，只不过它的语法和java完全不同，它有点类似于汇编的语法，是Android虚拟机所使用的寄存器语言，语法结构大概如下所示：

```
.class public Lcom/example/guolin/androidtest/MainActivity;
.super Landroid/support/v7/app/AppCompatActivity;
.source "MainActivity.java"


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 9
    invoke-direct {p0}, Landroid/support/v7/app/AppCompatActivity;-><init>()V

    return-void
.end method


# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 2
    .param p1, "savedInstanceState"    # Landroid/os/Bundle;

    .prologue
    .line 13
    invoke-super {p0, p1}, Landroid/support/v7/app/AppCompatActivity;->onCreate(Landroid/os/Bundle;)V

    .line 14
    const v1, 0x7f040019

    invoke-virtual {p0, v1}, Lcom/example/guolin/androidtest/MainActivity;->setContentView(I)V
```

看上去有点晕头转向是吗？但是如果你一旦能够看得懂smali文件的话，那么你就可以做很恐怖的事情了——你可以随意修改应用程序内的逻辑，将其进行破解！

不过我对这种黑技术并没有什么太大的兴趣，因此我也没有去做具体研究，但即使是这样，

也已经可以对程序的逻辑做一定程度的修改了。比如说当我们点击按钮时会弹出you clicked button这样一句Toast，逻辑是写在MainActivity按钮点击事件的匿名类当中的，因此这段代码反编译之后一定就会在MainActivity$1.smali这个文件当中，让我们打开瞧一瞧，部分代码如下所示：

```
38  # virtual methods
39 .method public onClick(Landroid/view/View;)V
40      .locals 3
41      .param p1, "v"    # Landroid/view/View;
42
43      .prologue
44      .line 19
45      iget-object v0, p0, Lcom/example/guolin/androidtest/MainActivity$1;->this$0:Lcom/example/guolin/androidtest/MainActivity;
46
47      const-string v1, "you clicked button"
48
49      const/4 v2, 0x0
50
51      invoke-static {v0, v1, v2}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
52
53      move-result-object v0
54
55      invoke-virtual {v0}, Landroid/widget/Toast;->show()V
56
57      .line 20
58      return-void
59  .end method
```

虽说多数的代码我是看不懂的，但其中第47行实在太明显了，Toast显示的内容不就是在这里定义的么，那么如果我们想把Demo程序hack掉，就可以将这段字符串给改掉，比如说我把它改成Your app is been hacked。

关于smali的语法，网上的资料也非常多，如果你对这门技术十分感兴趣的话可以直接上网去搜，这里我只是简单介绍一下，就不再深入讲解相关知识了。

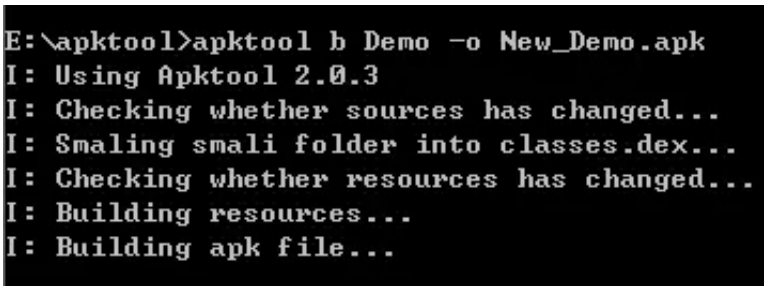
改了一处代码后我们再来改一处资源吧，比如这里想要把Demo的应用图标给换掉，那么首先我们要准备好一张新的图片，如下图所示：

然后从AndroidManifest.xml文件中可以看出，应用图标使用的是ic_launcher.png这张图片，那么我们将上面篮球这张图片命名成ic_launcher.png，然后拷贝到所有以res/mipmap开头的文件夹当中完成替换操作。

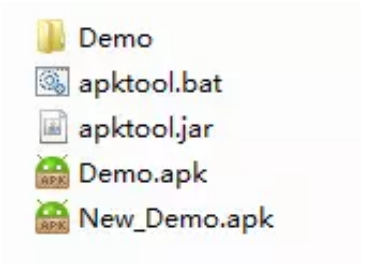
在做了两处改动之后，我们现在来把反编译后的Demo文件夹重新打包成APK吧，其实非常简单，只需要在cmd中执行如下命令：

> apktool b Demo -o New_Demo.apk

其中b是build的意思，表示我们要将Demo文件夹打包成APK文件，-o用于指定新生成的APK文件名，这里新的文件叫作New_Demo.apk。执行结果如下图所示：



现在你会发现在同级目录下面生成了一个新的APK文件：



不过不要高兴得太早了，目前这个New_Demo.apk还是不能安装的，因为它还没有进行签名。那么如果这是别人的程序的话，我们从哪儿能拿到它原来的签名文件呢？很显然，这是根本没有办法拿到的，因此我们只能拿自己的签名文件来对这个APK文件重新进行签名，但同时也表明我们重新打包出来的软件就是个十足的盗版软件。这里大家学学技术就好了，希望不要有任何人去做什么坏事情。

那么这里我就用一个之前生成好的签名文件了，使用Android Studio或者Eclipse都可以非常简单地生成一个签名文件。

有了签名文件之后在cmd中执行签名命令就可以进行签名了，命令格式如下：

> jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore 签名文件名 -storepass 签名密码 待签名的APK文件名 签名的别名

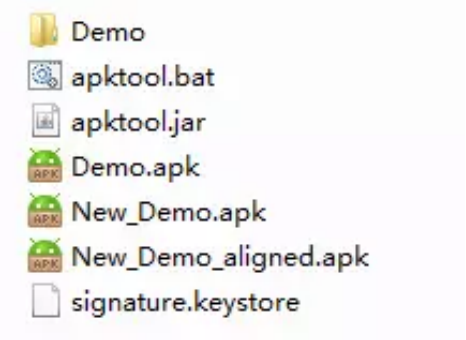
其中jarsigner命令文件是存放在jdk的bin目录下的，需要将bin目录配置在系统的环境变量当中才可以在任何位置执行此命令。

签名之后的APK文件现在已经可以安装到手机上了，不过在此之前Android还极度建议我们对签名后的APK文件进行一次对齐操作，因为这样可以使得我们的程序在Android系统中运

行得更快。对齐操作使用的是zipalign工具，该工具存放于/build-tools/目录下，将这个目录配置到系统环境变量当中就可以在任何位置执行此命令了。命令格式如下：

```
zipalign 4 New_Demo.apk New_Demo_aligned.apk
```

其中4是固定值不能改变，后面指定待对齐的APK文件名和对齐后的APK文件名。运行这段命令之后就会生成一个New_Demo_aligned.apk文件，如下所示：



这个New_Demo_aligned.apk就是我们重新打包签名对齐后的文件了，现在把它安装到手机上，效果如下图所示：

可以看到，应用图标已经成功改成了篮球，另外点击按钮后弹出的Toast的提示也变成了我们修改后的文字，说明重新打包操作确实已经成功了。

好的，我们把反编译代码、反编译资源、重新打包这三大主题的内容都已经掌握了，关于反编译相关的内容就到这里。