7. 智能手机安全

授课教师：游伟 副教授

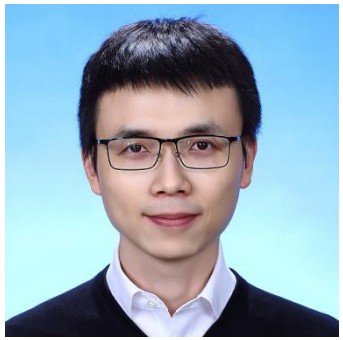
授课时间：周五10:00 – 11:30（立德楼909）

课程主页：https://www.youwei.site/course/cybersecurity

引子：电影《倒忌时》


2:35
倒忌时
取消
黑暗倒计时 一倒一忌一时一
死亡计算器你敢来试试吗
获取
App内购买
00 年
00 天
00 时
01 分
04 秒
02 57
19 56 53
倒计时应用 Countdown App
Death? There's an app for that
获取
04 YRS
44 DAY
13 HRS
52 MIN
19 SEC
45 YRS
154 DAY
26 MIN
8 SEC
大聪明说电影
bilibili
大聪明说电影
在国外有一款手机软件

嘉宾简介



黄建军，中国人民大学“杰出青年学者”，

荣获ACM SIGSOFT杰出论文奖。

- 当前研究方向：通过自然语言处理与静态程序分析相结合进行Android应用程序分析，包括恶意行为发现与隐私泄露检测等
- 未来研究兴趣：深度利用自然语言处理技术进行程序分析与相关知识的发现，以及通过机器学习的方法增强程序分析技术。

Android安全入门与应用重打包

黄建军
中国人民大学信息学院计算机系

首先介绍基本的Android应用基础与安全知识，然后介绍一种被广泛使用的攻击方法——应用重打包，及其基础的防御方法和绕开方法。

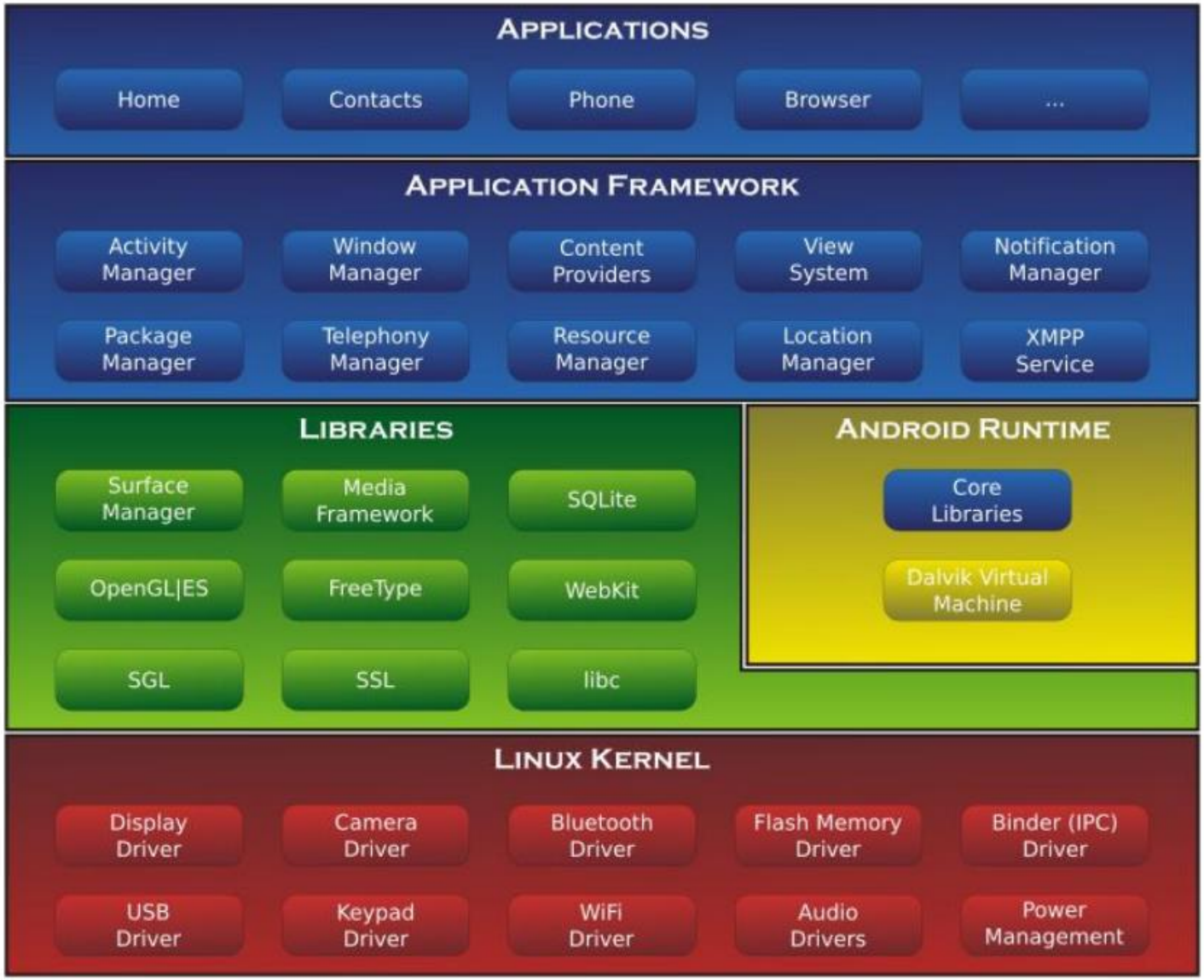
介绍Android应用重打包的基本概念和工具，并展示一个身边的重打包实例，然后简要说明进行重打包的基本流程（修改、重打包、签名），然后介绍最常用的基于应用签名验证的反重打包机制并简述相应的绕开方法。

内容

- Android入门及安全基础
- 应用重打包
- 身边的实例
- 重打包入门示例
- 反重打包
- 反“反重打包”
- 课后思考

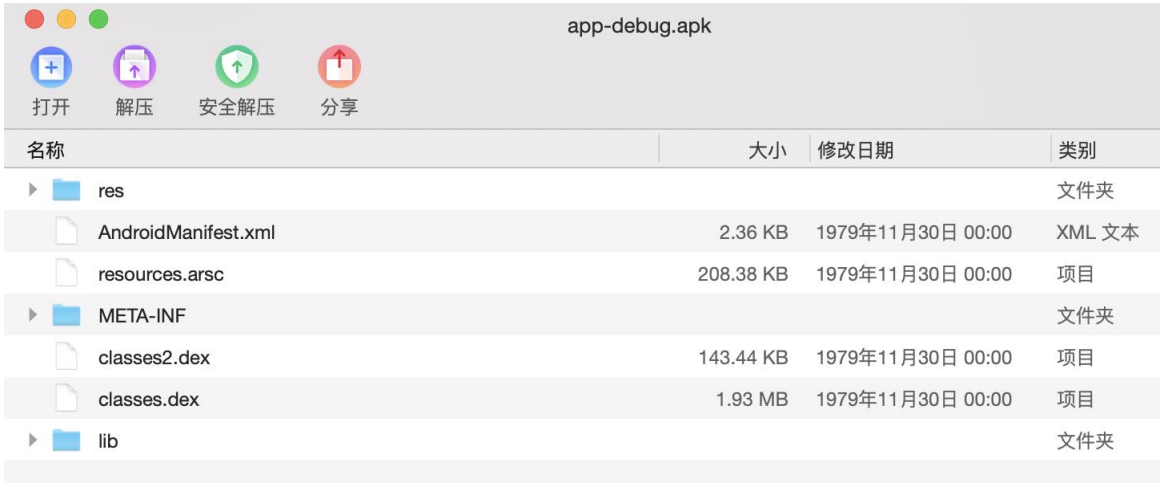
1 Android入门及安全基础

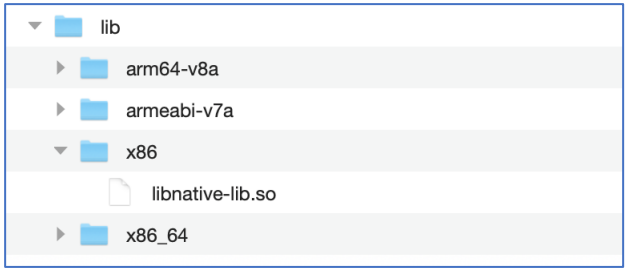
系统架构



Android应用的基本构造

- Android应用的安装包是APK格式（Android Application Package），是一种zip压缩格式，可通过常用的解压软件（如7-zip）解压
- 一个APK通常包括





Android应用的组件

- Android应用通常包含四大组件
  - Activity：为用户提供了一个可以通过与之交互来完成某些操作的界面。通常来说，Activity是Android应用中最重要的组件。对绝大多数应用来说，与用户交互是其最基本的功能
  - Service：可以在后台长时间运行，且不提供用户界面，如E-Mail更新检查服务
  - Broadcast Receiver：能在设备系统中接收广播通知的组件，能接受诸如低电量、启动完成、耳机连接、网络更换等消息以及其他应用程序发出的消息
  - Content Provider：以表格的形式提供数据，可以用于与其他应用共享数据。所有的Content Provider使用content://开头的特殊格式的URI，如content://sms/inbox提供了内置短信应用收件箱的接口

AndroidManifest.xml

- AndroidManifest文件声明了一个Android应用所需要的权限、组件（Activity、Service等）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:installLocation="auto" package="com.tencent.mm"
    <permission android:name="com.tencent.mm.plugin.permission.WRITE" android:protectionLevel="signature"/>
    <permission android:name="com.tencent.mm.plugin.permission.READ" android:protectionLevel="signature"/>
    <permission android:name="com.tencent.mm.permission.MM_MESSAGE" android:protectionLevel="signature"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
    <uses-permission android:name="com.tencent.mm.plugin.permission.READ"/>
    <uses-permission android:name="com.tencent.mm.plugin.permission.WRITE"/>
    <application android:allowBackup="false" android:appComponentFactory="android.support.v4.app.C
     android:largeHeap="true" android:name="com.tencent.mm.app.Application" android:persistent="fals
        <activity android:launchMode="standard" android:name="com.tencent.mm.app.WeChatSplashActi
        <activity android:name="com.tencent.mm.app.WeChatSplashFallbackActivity" android:process=":fa
        <service android:name="com.tencent.mm.splash.DexOptService" android:process=":dexopt"/>
<activity android:configChanges="keyboardHidden|orientation|screenSize" android:icon="@drawable/icon" android:name="com.tencent.mm.ui.tools.ShareImgUI"
  <intent-filter android:label="@string/v1">
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="image/*"/>
    <data android:mimeType="video/*"/>
    <data android:mimeType="text/*"/>
    <data android:mimeType="application/*"/>
  </intent-filter>
  <intent-filter android:label="@string/v1">
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="image/*"/>
  </intent-filter>
</activity>
```

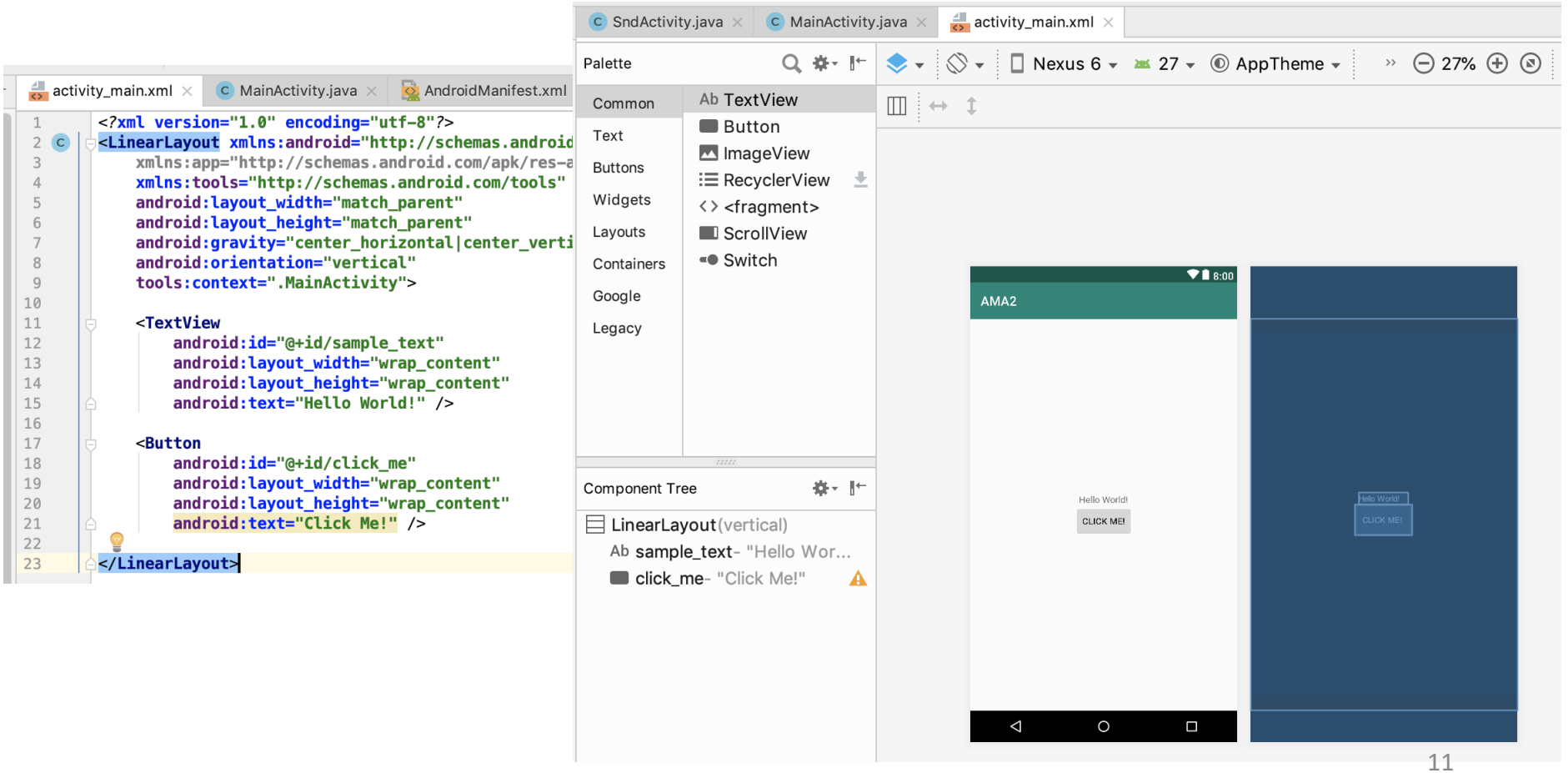
自定义权限、声明权限

自定义Application组件、声明activity等

声明支持的Intent

布局文件（Layout XML）

- XML格式的布局文件（layout）用于静态定义用户界面。应用程序代码加载布局文件之后，系统渲染生成相应的界面
  - res/layout/activity_main.xml



应用沙箱与UID模型

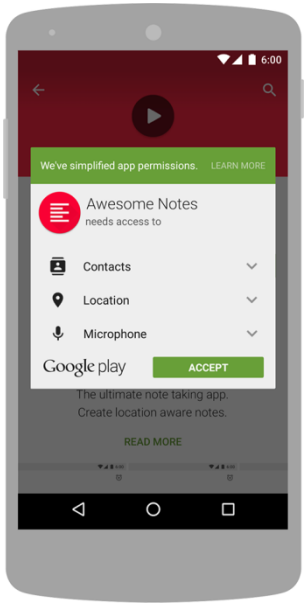
- 沙箱，对使用者来说可以理解为一种安全环境，对恶意访问者来说是一种限制
- 在Android系统中，应用（通常）都在一个独立的沙箱中运行，即**每一个Android应用程序都在它自己的进程中运行，都拥有一个独立的Dalvik虚拟机实例**
- Android系统沿用了Linux的UID权限模型和GID权限模型
- 一般情况下，Android为每一个应用程序分配一个唯一的UID
- 在Linux中，一个UID识别一个给定用户；在Android上，一个UID则识别一个应用程序。在安装应用程序时向其分配UID

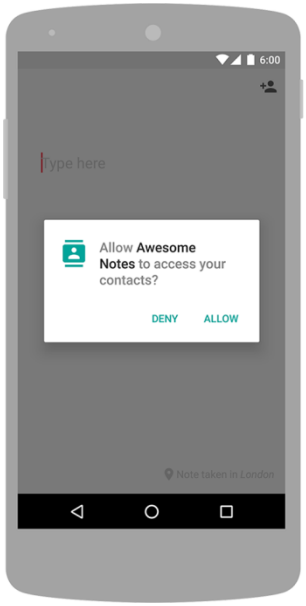
安全权限机制

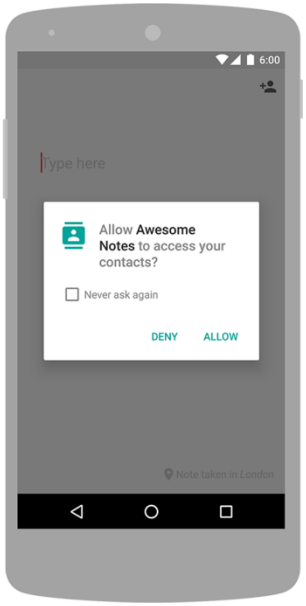
- Android继承了Linux的安全机制，并发展出了**权限**（Permission）机制，系统更多的安全功能通过该机制提供
- 权限可以限制某个特定进程的特定操作，也可以限制每个URI权限对特定数据段的访问
- Android安全架构的核心思想：默认设置下，所有应用都没有权限对其他应用、系统或用户进行较大影响的操作，包括读写用户隐私数据（联系人或电子邮件）、读写其他应用文件、访问网络、发送短信等
- 常见的系统原生权限包括：发送短信/彩信、拨打电话、修改SD卡上的内容、读取联系人、获得精确的基于GPS的地理位置、振动控制、访问网络等

权限批准

- 如果应用程序包含Normal权限申请，系统自动地赋予应用程序相应的权限
- 如果应用包含Dangerous权限申请，需要用户显式地同意并批准相应权限
  - 安装时权限请求（<= Android 5.1）
  - 运行时权限请求（>= Android 6）







应用签名

摘要

- 对原APK中每个文件计算摘要
- 将数据保存至MANIFEST.MF文件

二次摘要

- 对MANIFEST.MF文件本身及其中每一条记录的摘要再计算摘要
- 将数据保存至*.SF文件

签名

- 对ANDROID.SF文件计算摘要，利用私钥对该摘要加密
- 将公钥证书、上述加密结果等存入*.RSA

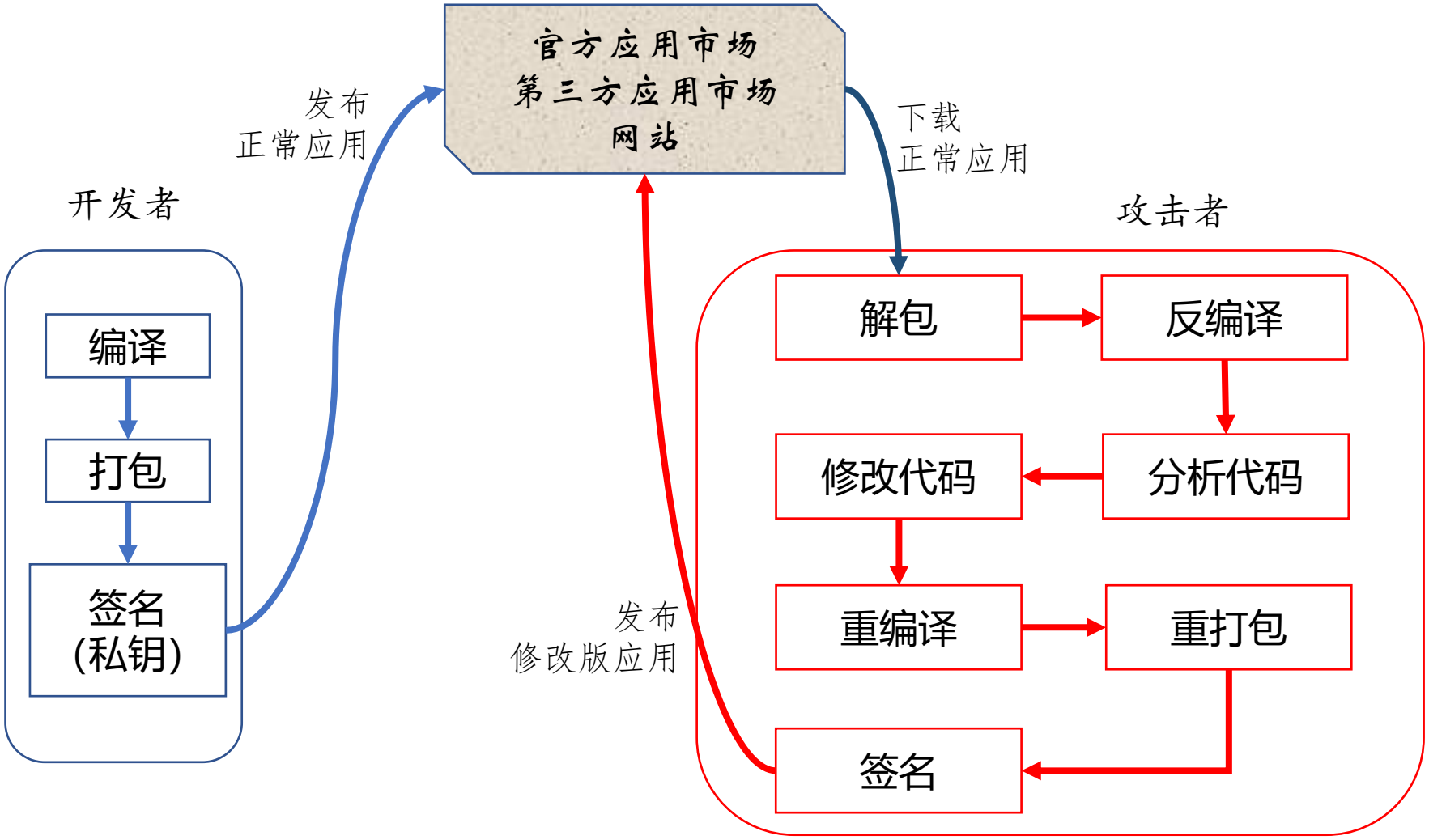
2 应用重打包

- 重打包（Repackaging）是Android平台上常见的攻击
- 攻击者通过修改应用市场下载的、流行的应用程序，对其逆向工程，添加恶意的代码（payload），再上传到应用市场或其他网站吸引用户下载安装
  - 多年前，研究人员发现，DroidDream木马被植入到Google Play上超过50个应用
  - **有研究指出80%的Android恶意应用是通过重打包实现的**
- 用户也可能想要去除（绕开）目标应用中的付费过程，对应用重打包

> 作为一名最强大脑节目的铁粉，十分崇拜里面大脑们的精彩表现，所以在大学的时候就下载了一款名字叫做疯狂大脑的应用，该应用可以构建自己的记忆宫殿、训练记忆编码等等，对训练记忆十分方便。可是最近更换手机重新安装的时候发现，该应用的许多功能都被设为付费项目，需要支付一定金额才可以打开，这引起了我的极大兴趣，很想看看它是怎么做的。于是就有了下面的过程。

- 有一定能力的开发者想要让更多的应用能嵌入自己的广告代码以获利

重打包的基本流程


官方应用市场
第三方应用市场
网站
发布
正常应用
下载
正常应用
开发者
攻击者
编译
打包
签名
(私钥)
解包
反编译
修改代码
分析代码
重编译
重打包
签名
发布
修改版应用

重打包的基本工具

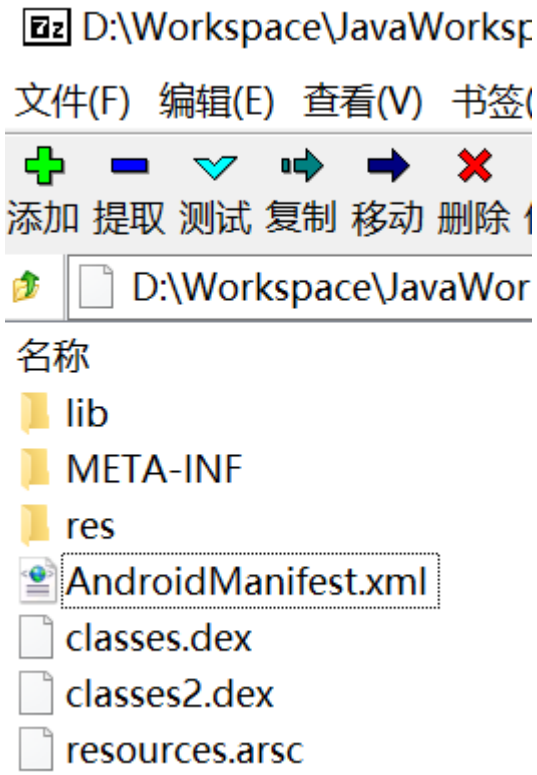
- 主要关注针对Java/kotlin代码编译而成的DEX字节码、应用开发过程中构建的XML文件进行修改
  - 不关注针对本地代码（Native Code）的修改
- 常用工具：ApkTool
  - 常用的APK反编译工具，可以将编译后的XML文件（二进制）还原为人可读的形式（几乎与开发时的状态一致），将编译后的DEX字节码反编译为一种smali格式的代码
  - 对无法处理的.so文件等保持原样
  - 攻击者修改XML或smali代码之后，ApkTool能够将所有文件重编译为APK包
  - 官网：`https://ibotpeaches.github.io/Apktool`
- 其他工具：包括dex2jar，JDGUI等

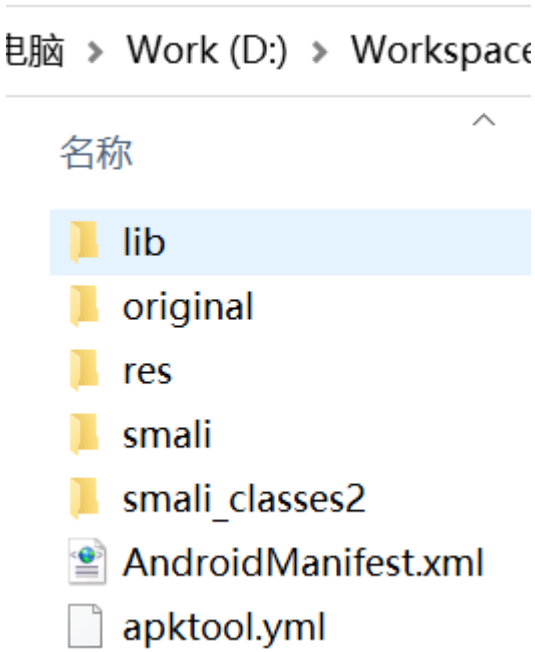
ApkTool反编译

- 以一个简单的示例应用为例
  - 使用Android Studio开发，默认情况下生成的APK是debug模式，位于<project>/app/build/outputs/apk/debug目录下
  - 该APK未被签名（APK中META-INF目录下没有*.SF及*.RSA），在Android Studio将其安装到模拟器时使用自带的私钥进行签名，该私钥位于C:/Users/<user>/.android/debug.keystore
  - 执行命令：`apktool d app-debug.apk`
    - `apktool`的命令行选项请自行查阅相关资料

```
I: Using Apktool 2.3.4 on app-debug.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\HUANG\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

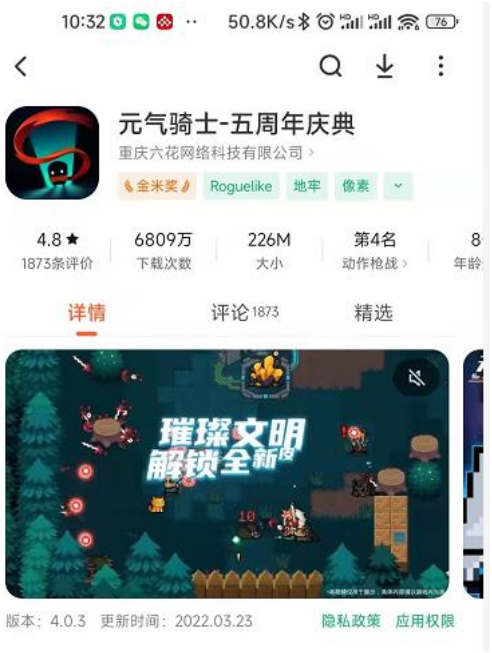
- 当前目录下生成app-debug目录（与APK文件名一致），其中包含反编译后的smali代码、xml文件（AndroidManifest、layout等），以及来自于原APK包中未做修改的文件（原始的二进制格式的AndroidManifest以及META-INF目录）
- 下方左图是APK包结构；右图是反编译后的目录结构，其中smali与smali_classes2分别对应原APK中的classes.dex与classes2.dex





3 身边的实例

- “元气骑士”的内购破解
  - 元气骑士是一款由凉屋游戏工作室研发的一款角色扮演类游戏，于2017年4月18日发布
  - 用户可以通过应用内支付的方式购买道具，价格不等，最低1元
  - 一位接触Android逆向工程不久的同学想要通过**重打包**的方式使得自己能够少花钱、畅快玩



- 获取有效信息

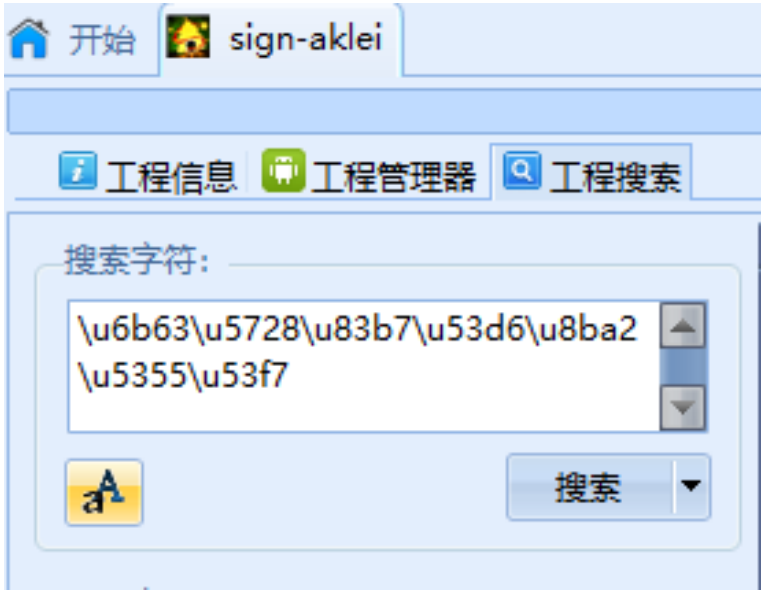

选择角色
0
Bai du 经验
jingyan.baidu.com

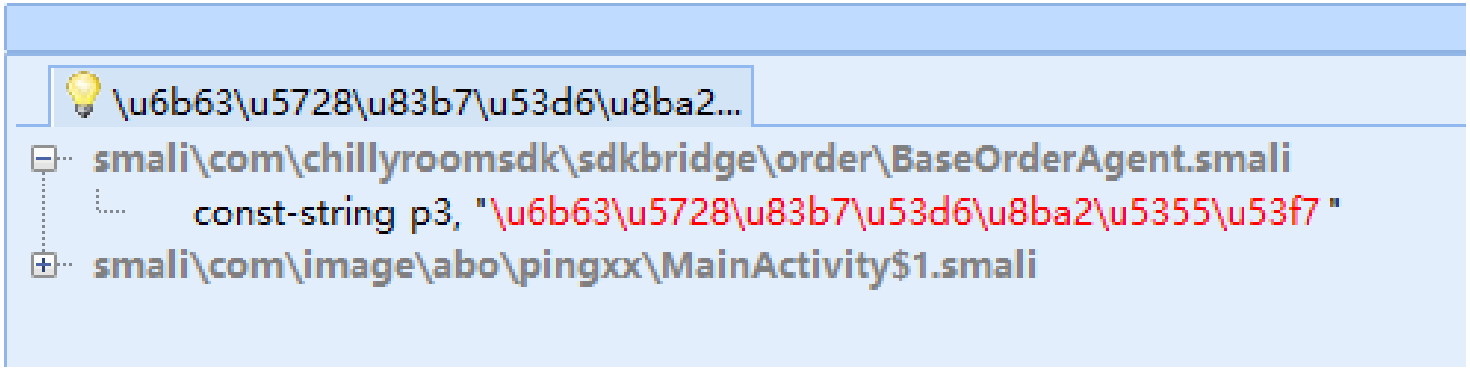

宝石商店
元气骑士
正在获取订单号
null 校验订单失败:unpaid
返回


宝石商店
元气骑士
正在检查支付结果
订单取消user_cancelled,
返回

- 定位有效信息


开始
sign-aklei
工程信息
工程管理器
工程搜索
搜索字符:
正在获取订单号
搜索


开始
sign-aklei
工程信息
工程管理器
工程搜索
搜索字符:
\u6b63\u5728\u83b7\u53d6\u8ba2
\u5355\u53f7
搜索


\u6b63\u5728\u83b7\u53d6\u8ba2...
smali\com\chillyroomsdk\sdkbridge\order\BaseOrderAgent.smali
const-string p3, "\u6b63\u5728\u83b7\u53d6\u8ba2\u5355\u53f7"
smali\com\image\abo\pingxx\MainActivity$1.smali

- 发现与价格相关的变量和订单相关的类（在反编译得到的Java代码上查看）

BaseOrderAgent.class

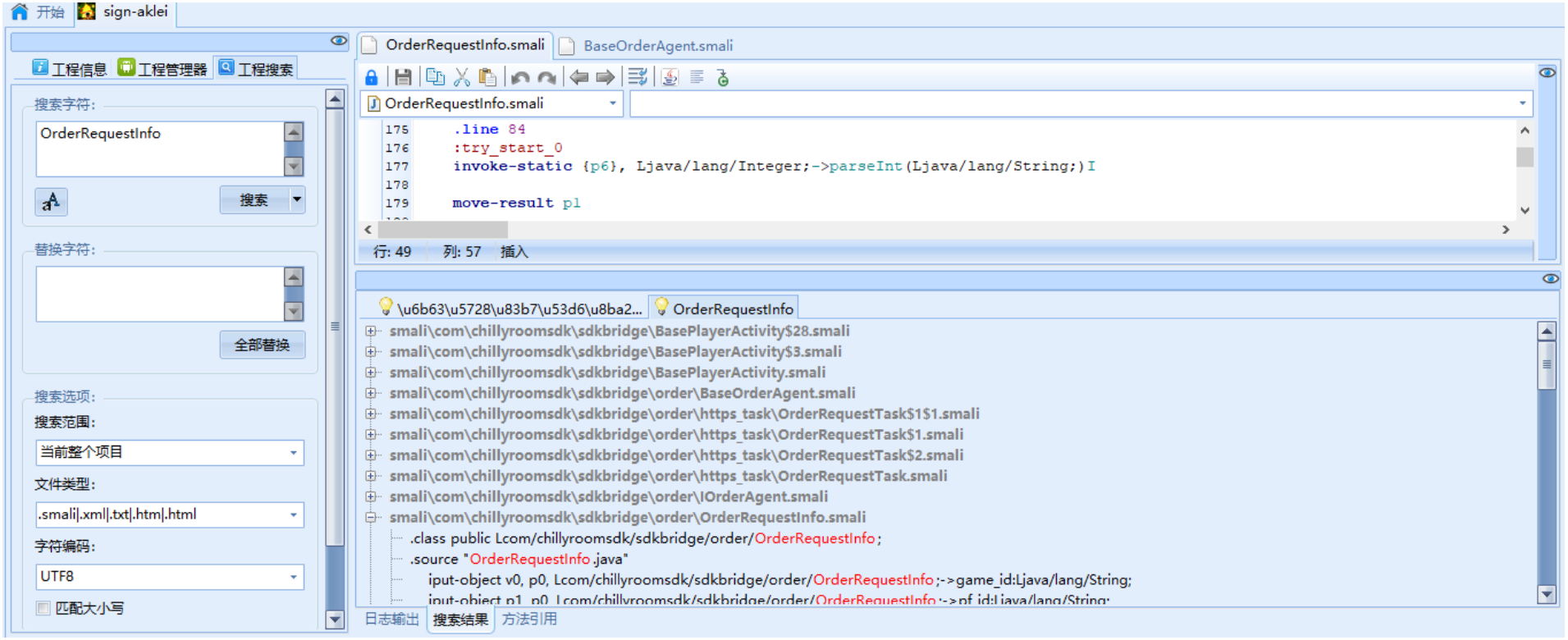
```
public void doRequestOrder(String paramString1, String paramString2, String paramString3, String paramString4, String paramString5)
{
  paramString5 = SdkConfig.getInstance().getAppParam("appId");
  String str = SdkConfig.getInstance().getAppParam("packageId");
  int i = Integer.parseInt(SdkConfig.getInstance().channelId);
  paramString1 = new OrderRequestInfo(SdkConfig.getInstance().channelName, i, paramString5, paramString1, paramString2, paramString3, str, param
  onRequestOrderInfo(paramString1);
  paramString2 = new StringBuilder();
  paramString2.append("order info price:");
  paramString2.append(paramString1.total_price);
  Log.i("Order", paramString2.toString());
  OrderRequestTask.StartTaskOnMainThread(ProgressDialog.show(UnityPlayer.currentActivity, SdkConfig.getInstance().getAppName(), "正在获取订单号"
}

public void doRestoreOrder(Activity paramActivity, ProgressDialog paramProgressDialog, String paramString, boolean paramBoolean)
{
  OrderRestoreTask.StartTaskOnMainThread(paramActivity, paramProgressDialog, paramString, paramBoolean);
}

public IOrderAgent getOrderAgent()
{
  return this;
}

public void onRestoreOrder(String paramString1, String paramString2, String paramString3, double paramDouble, String paramString4)
{
  if ((UnityPlayer.currentActivity instanceof IStatisticsAgent))
  {
    String str = SdkConfig.getInstance().channelId;
    ((IStatisticsAgent)UnityPlayer.currentActivity).getStatisticsAgent().logPay(paramString1, paramString2, paramString3, paramDouble, str, para
  }
  paramString4 = new StringBuilder();
  paramString4.append(paramString3);
  paramString4.append(" 订单已恢复");
  paramString4.toString();
```

- 搜索相关类


开始
sign-aklei
工程信息
工程管理器
工程搜索
搜索字符:
OrderRequestInfo
搜索
替换字符:
全部替换
搜索选项:
搜索范围:
当前整个项目
文件类型:
.smali|.xml|.txt|.htm|.html
字符编码:
UTF8
匹配大小写
OrderRequestInfo.smali
BaseOrderAgent.smali
OrderRequestInfo.smali
175 .line 84
176 :try_start_0
177 invoke-static {p6}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
178
179 move-result p1
行: 49
列: 57
插入
\u6b63\u5728\u83b7\u53d6\u8ba2...
OrderRequestInfo
smali\com\chillyroomsdk\sdkbridge\BasePlayerActivity$28.smali
smali\com\chillyroomsdk\sdkbridge\BasePlayerActivity$3.smali
smali\com\chillyroomsdk\sdkbridge\BasePlayerActivity.smali
smali\com\chillyroomsdk\sdkbridge\order\BaseOrderAgent.smali
smali\com\chillyroomsdk\sdkbridge\order\https_task\OrderRequestTask$1$1.smali
smali\com\chillyroomsdk\sdkbridge\order\https_task\OrderRequestTask$1.smali
smali\com\chillyroomsdk\sdkbridge\order\https_task\OrderRequestTask$2.smali
smali\com\chillyroomsdk\sdkbridge\order\https_task\OrderRequestTask.smali
smali\com\chillyroomsdk\sdkbridge\order\IOrderAgent.smali
smali\com\chillyroomsdk\sdkbridge\order\OrderRequestInfo.smali
.class public Lcom/chillyroomsdk/sdkbridge/order/OrderRequestInfo;
.source "OrderRequestInfo.java"
iput-object v0, p0, Lcom/chillyroomsdk/sdkbridge/order/OrderRequestInfo;->game_id:Ljava/lang/String;
日志输出
搜索结果
方法引用

- 从Java代码上观察价格变量的赋值

OrderRequestInfo.class

```
public OrderRequestInfo(String paramString1, int paramInt, String paramString2, String paramString3, String paramString4, String paramString5,
{
  this.pf_id = paramString1.toLowerCase();
  this.pk_id = paramString6;
  this.pay_id = paramInt;
  this.op_id = 0;
  this.app_id = paramString2;
  this.server_id = 1;
  this.pay_channel = paramString7;
  this.account = "";
  if ((UnityPlayer.currentActivity instanceof ILoginAgent)) {
    paramString1 = ((ILoginAgent)UnityPlayer.currentActivity).getUid("");
  } else {
    paramString1 = "";
  }
  this.third_account = paramString1;
  paramString1 = new StringBuilder();
  paramString1.append(OrderManager.getOrderIndex());
  paramString1.append("");
  this.role_id = paramString1.toString();
  this.item_id = paramString3;
  this.item_name = paramString4;
  this.item_count = 1;
  try
  {
    this.total_price = Integer.parseInt(paramString5);
  }
  catch (Exception paramString1)
  {
    label151:
    break label151;
  }
  this.total_price = Math.round(Float.parseFloat(paramString5));
  this.item_price = this.total_price;
  this.currency = "RMB";
  this.code = "";
```

- 修改价格

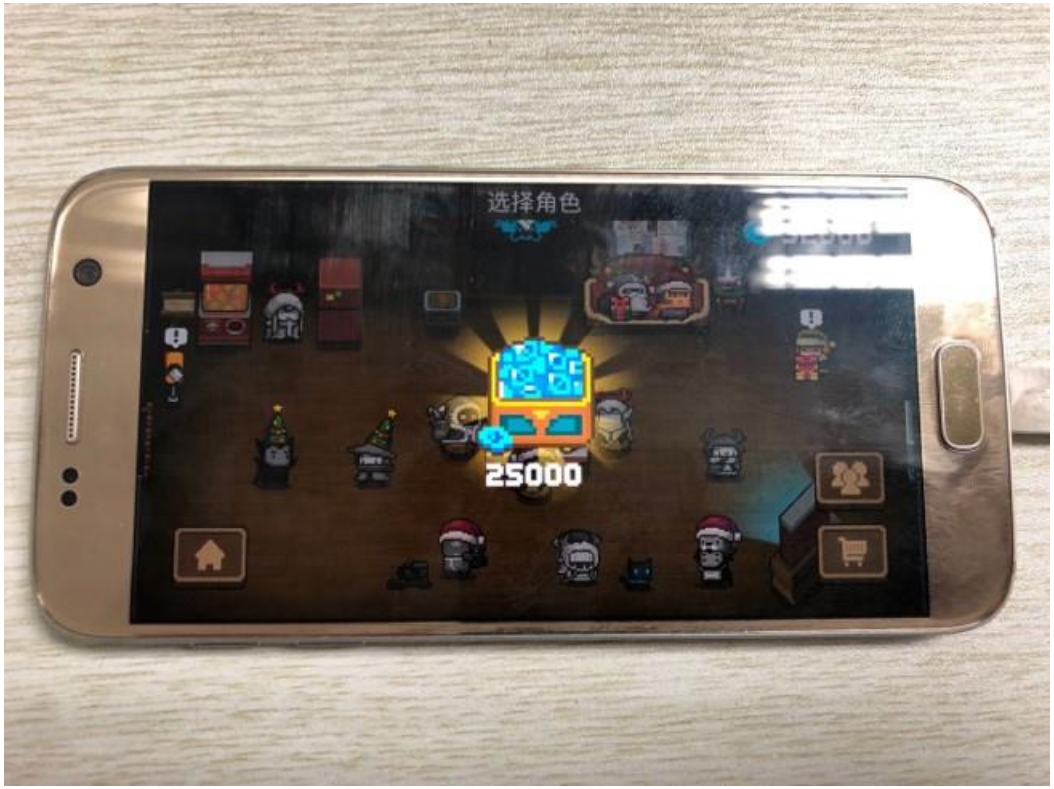
OrderRequestInfo.smali | BaseOrderAgent.smali

OrderRequestInfo.smali

```
175     .line 84
176     :try_start_0
177     invoke-static {p6}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
178
179     move-result p1
180
181     const/4 p1, 0x1
182
183     iput p1, p0, Lcom/chillyroomsdk/sdkbridge/order/OrderRequestInfo;->total_price:I
184     :try_end_0
185     .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
186
187     goto :goto_1
188
189     .line 87
190     :catch_0
191     invoke-static {p6}, Ljava/lang/Float;->parseFloat(Ljava/lang/String;)F
192
193     move-result p1
194
195     invoke-static {p1}, Ljava/lang/Math;->round(F)I
196
```

破解结果


宝石商
免费复活卡
800
5000
确认付款
¥0.01
订单信息
25000宝石
付款方式
账户余额
立即付款


选择角色
25000


宝石商店
免费复活卡
800
5000
12000
25000
¥10
¥1
¥5
¥10
¥15
麦克风
00:02

4 重打包入门示例

- 重打包任务：在应用初始界面中添加一个横条，用作模拟添加的广告
  - 假设目标APK是app-debug.apk
  - 实际场景中，添加广告除了要修改layout对应的XML之外，还需要添加相应的代码，使得应用能连接网络更新广告、响应点击等
  - 这里简化所有操作，仅仅添加一个界面文本元素
  - 右图展示了目标应用的原起始界面

- 根据对APK的分析，发现其实界面对应的layout是res/layout/activity_main.xml，如图所示
- 在Button下方加入下图所示的XML代码，创建一个文本标签，其内容是红色的“This is an injected banner!!”

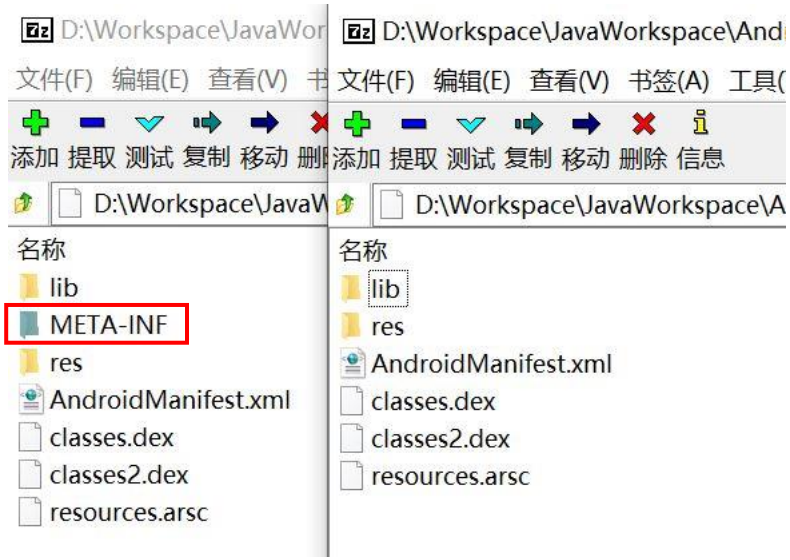
activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout android:gravity="center" android:
   orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent"
3     xmlns:android="http://schemas.android.com/apk/res/
      android" xmlns:app="http://schemas.android.com/apk/
      res-auto">
4       <TextView android:id="@id/sample_text" android:
        layout_width="wrap_content" android:layout_height="
        wrap_content" android:text="Hello World!" />
5       <Button android:id="@id/click_me" android:
        layout_width="wrap_content" android:layout_height="
        wrap_content" android:text="Click Me!" />
6  </LinearLayout>
```

```
<TextView android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAlignment="center"
    android:textColor="@android:color/holo_red_dark"
    android:text="This is an injected banner!!" />
```

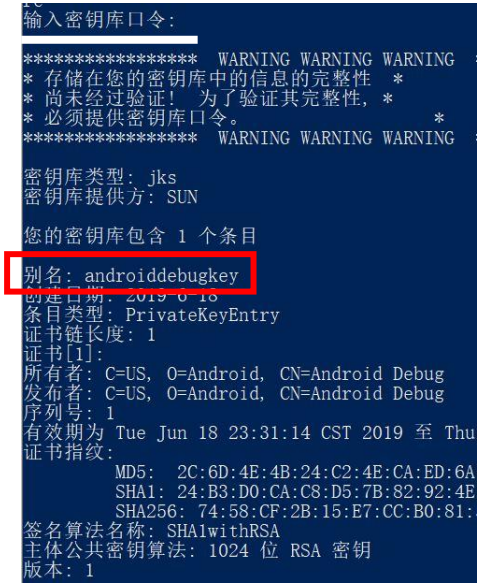
- 做出修改后，执行ApkTool命令，重编译打包应用程序
  - `apktool b app-debug`
  - 其中“app-debug”包含了反编译生成的所有内容
  - 无错时，出现下左图所示的重编译流程提示
  - 默认生成的APK文件在app-debug/dist目录下（未签名）
  - 对照可发现重打包生成的APK没有META-INF目录（下右图）

```
I: Using Apktool 2.3.4
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether sources has changed...
I: Smaling smali_classes2 folder into classes2.dex...
I: Checking whether resources has changed...
I: Building resources...
W: fakeLogOpen(/dev/log_crash) failed
W: fakeLogOpen(/dev/log_stats) failed
W: fakeLogOpen(/dev/log_stats) failed
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```



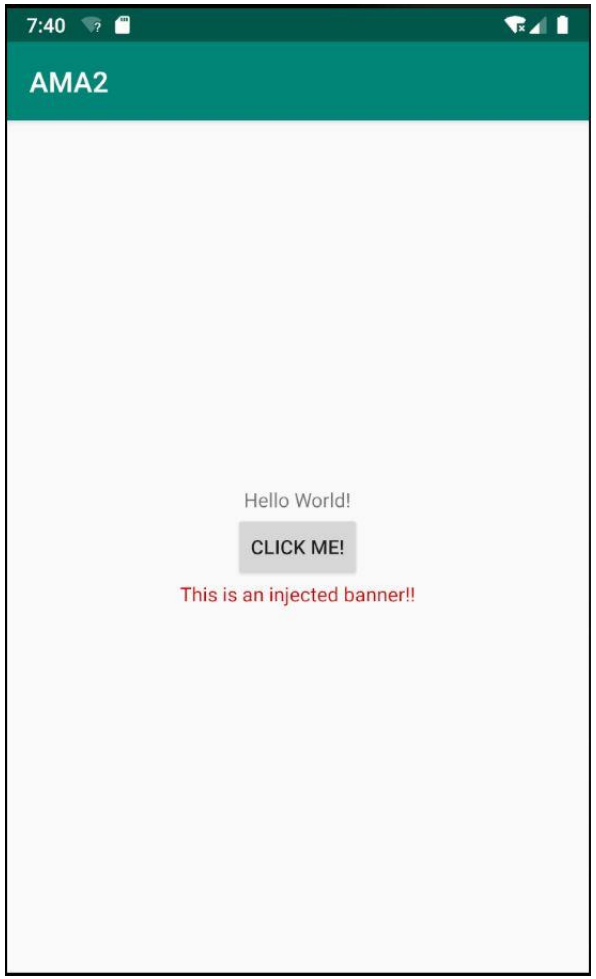
对应用签名

- 开发者通常是自己通过工具创建密钥对，使用私钥对应用签名
- 本例直接使用Android Studio提供的密钥进行签名
- 执行以下命令（jarsigner为JDK中的工具）
  - jarsigner -verbose -keystore <path-to>\debug.keystore -signedjar app-debug-signed.apk app-debug\dist\app-debug.apk androiddebugkey
    - 可以添加其它的命令行选项，如指定哈希算法
  - 其中，app-debug-signed.apk为签名后生成的文件，androiddebugkey为密钥库（debug.keystore）的别名，该别名可以通过JDK中keytool命令查看
    - 如：keytool -list -v -keystore <path-to>\debug.keystore，效果如右上图所示
  - 两个命令运行时，要求输入密钥库的口令（或密码短语），该密钥库口令为android
    - 自己创建密钥库需牢记口令，否则无法进行应用签名
  - 右下图展示了jarsigner的工作流程



安装运行重打包的应用

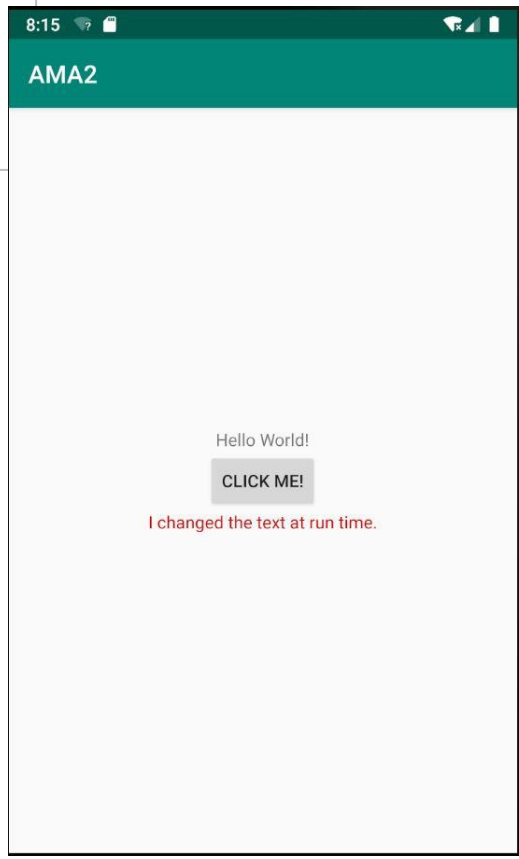
- 使用Android SDK提供的adb工具安装重打包签名后的应用
  - 执行命令：`<path-to>\adb install ap-debug-signed.apk`
  - 在Windows上，通过Android Studio安装的SDK中sdb默认位于C:\Users\<user>\AppData\Local\Android\platform-tools目录
  - 安装若遇到如下错误提示：[INSTALL_FAILED_TEST_ONLY: installPackageLI]，则需要将AndroidManifest.xml中的android:testOnly="true"去除，或改为“false”，然后利用ApkTool重新打包，再重新签名；或是在`install`之后加上“`-t`”选项
  - 出现“Success”提示表明安装成功（若系统已有同样的应用，需卸载后再安装）
  - 运行该应用出现如右图所示的效果，表明对原应用的修改成功完成



修改可执行代码（Java）

- 修改可执行代码（Java部分），需要修改ApkTool反编译得到的Smali代码，再重打包+签名

```
1. const v1, 0x7f070092
2. invoke-virtual {p0, v1}, …/MainActivity;->findViewById(I)…
3. move-result-object v1
4. check-cast v1, Landroid/widget/TextView;
5. const-string v0, "I changed the text at run time."
6. invoke-virtual {v1, v0}, …/TextView;->setText(…)V
```

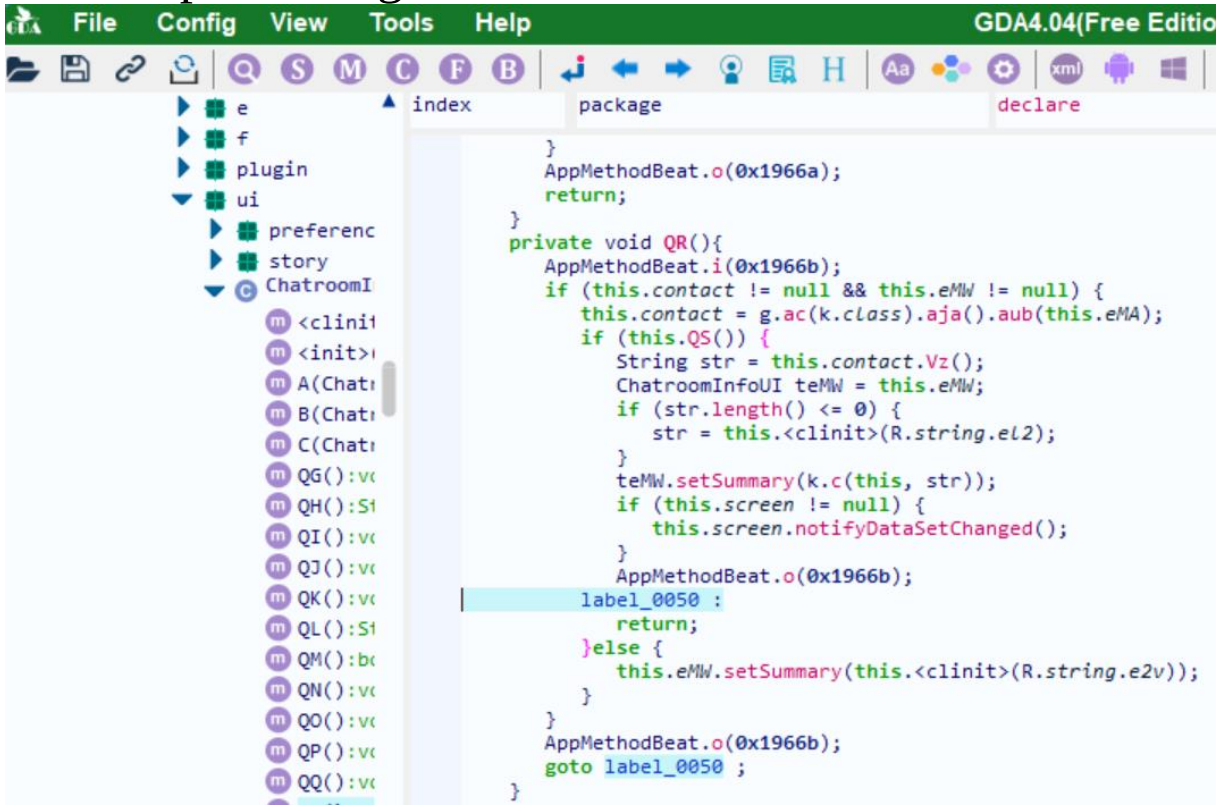


工具推荐：AndroidKiller

- AndroidKiller是可视化界面的一款反编译软件，集APK反编译、打包、签名（自带keystore）等功能于一身，是常用的smali代码阅读器
  - https://down.52pojie.cn/Tools/Android_Tools/AndroidKiller_v1.3.1.zip
  - **缺点：**该软件只有Windows版，且多年不再更新
  - **优点：**可以通过点击界面操作的方式完成编译（含编译、重打包、签名）、安装、运行

工具推荐：GDA

- GJoy Dex Analyzer(GDA)，亚洲第一款全交互式的现代反编译器，同时也是世界上最早实现的dalvik字节码反编译器。 GDA不只是一款反编译器，同时也是一款轻便且功能强大的综合性逆向分析利器，其不依赖java且支持apk, dex, odex, oat, jar, class, aar文件的反编译，支持python及java脚本自动化分析。
- http://www.gda.wiki:9090/



持续更新，可较为正确地（无法保证）将dex字节码反编译为Java源代码

5 反重打包

- 针对应用重打包攻击，Android系统本身及相关研究人员提出了各种各样的反重打包技术，本节将介绍Android系统提供的最基础的基于签名验证的反重打包技术
- 基本原理：
  - 攻击者对合法应用做出修改，可能会改动多个地方，如AndroidManifest中的元信息：package名称、版本号；此外，攻击者无法提供与原开发者一致的密钥库对重打包后的应用进行签名，因此修改过的APK文件中的签名信息也会发生改变
  - 通过在应用程序中检查目标值与当前运行时所获结果的不一致，可以即时地判断出应用被重打包，从而结束运行
- 实际场景中，多采用验证签名判断是否遭遇重打包

获取签名

- Android提供PackageManager.getPackageInfo()来获取应用程序的各种信息，如：package名称，应用中包含的Activity、Service、Provider、Receiver，所请求的权限，版本名称，签名信息等
  - 其调用方法为（以在onCreate()中为例）

```
1. PackageManager pm = getApplicationContext().getPackageManager();
2. PackageInfo pinfo = pm.getPackageInfo(getPackageName(), X);
```

  - 其中第2行的X为该调用将获取的目标，取值包括：PackageManager.GET_SIGNATURES，PackageManager.GET_PERMISSIONS等
  - 在一个简单示例程序中获取签名数据，以字符串（十六进制）输入这个签名，并获取其摘要信息（哈希），代码如下。其中digest()方法是根据摘要算法计算哈希值并输出的一个方法，其中利用JDK中的MessageDigest计算哈希

```
1. Signature sig = pinfo.signatures[0];
2. System.err.println("MA: sig = " + sig.toCharsString().length()
3.                     + " " + sig.toCharsString());
4. digest(“MD5”, sig.toByteArray());
5. digest("SHA1", sig.toByteArray());
6. digest("SHA-256", sig.toByteArray());
```

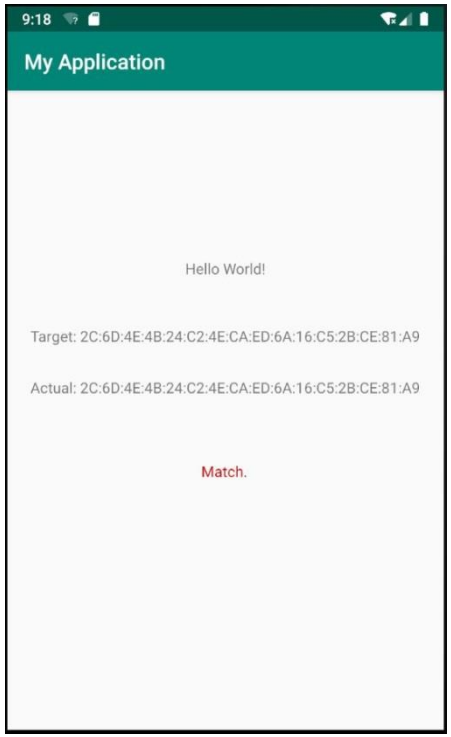
- 运行结果如下所示(从logcat中获得的运行时日志信息）
  - 第1行输出了所获取的签名信息转化为字符串后的长度，以及对应的字符串（太长因此后续使用“…”代替
  - 第2~4行显示了该签名信息分别使用三种摘要算法计算得到的哈希值

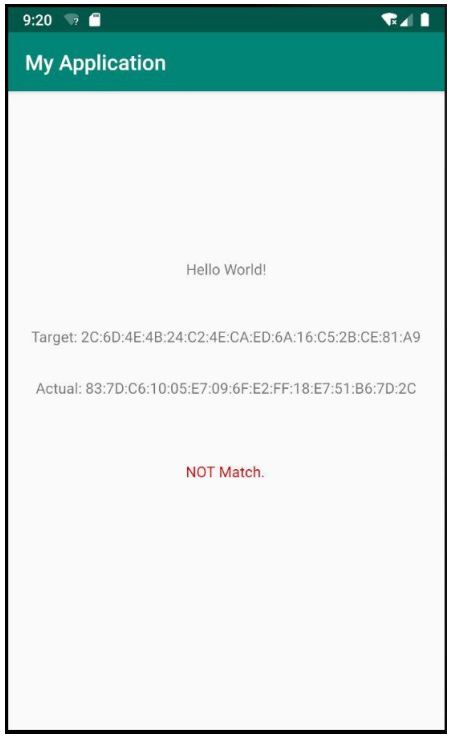
```
1. MA: sig = 962 308201dd30820146020101300d06092a864886f70d…
2. MD5 : 2C:6D:4E:4B:24:C2:4E:CA:ED:6A:16:C5:2B:CE:81:A9
3. SHA1 : 24:B3:D0:CA:C8:D5:7B:82:92:4E:F0:22:60:69:1E:CD:72:ED:E9:6B
4. SHA-256 : 74:58:CF:2B:15:E7:CC:B0:81:5E:20:39:13:07:1B:EE:
             2A:A4:55:29:09:63:D2:71:87:FD:D2:2D:CA:E5:87:C5
```

  - 下图展示了使用keytool从Android Studio自带的密钥库获取的信息，对比可发现与上述计算得到的各类摘要信息完全一致

```
别名: androiddebugkey
创建日期: 2019-6-18
条目类型: PrivateKeyEntry
证书链长度: 1
证书[1]:
所有者: C=US, O=Android, CN=Android Debug
发布者: C=US, O=Android, CN=Android Debug
序列号: 1
有效期为 Tue Jun 18 23:31:14 CST 2019 至 Thu Jun 10 23:31:14 CST 2049
证书指纹:
         MD5:  2C:6D:4E:4B:24:C2:4E:CA:ED:6A:16:C5:2B:CE:81:A9
         SHA1: 24:B3:D0:CA:C8:D5:7B:82:92:4E:F0:22:60:69:1E:CD:72:ED:E9:6B
         SHA256: 74:58:CF:2B:15:E7:CC:B0:81:5E:20:39:13:07:1B:EE:2A:A4:55:29:09:63:D2:71:87:FD:D2:2D:CA:E5:87:C5
签名算法名称: SHA1withRSA
主体公共密钥算法: 1024 位 RSA 密钥
```

- 将运行时提取到的签名信息与应用开发者密钥库中的签名信息进行对比，若不一致，则判定应用被重打包
- 以一个简单应用为例，其界面显示正确的签名信息的MD5值、当前运行时获取到的签名信息的MD5值、两者是否匹配的结果
- 直接从Android Studio运行应用程序得到如下左图的结果
- 使用ApkTool解包，再重打包，利用一个新建的密钥库签名之后运行的结果如下右图所示



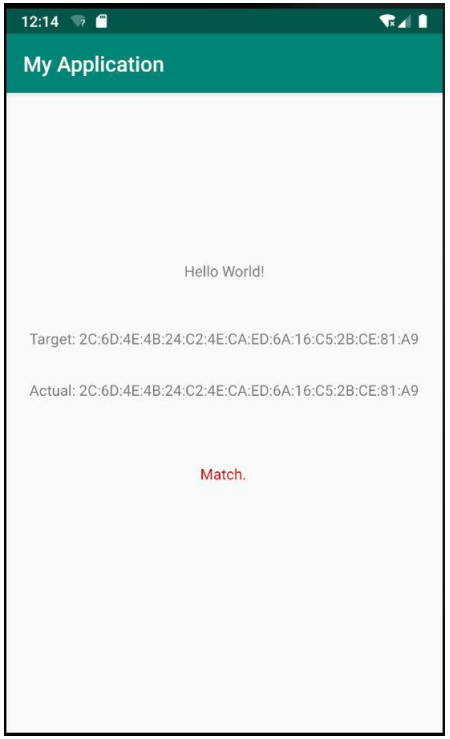


6 反“反重打包”

- 前述的反重打包技术是一种通用的、简单易行的技术
- 攻击者针对这种技术开发了新的攻击方法，除了深入分析反编译后的代码去除相应的签名验证外，还可以通过劫持相关函数、提供原始签名的方式绕过验证
- 具体而言：
  - 在应用运行启动时，劫持getPackageInfo()方法，并提供一个伪造的版本
  - 应用运行过程中调用getPackageInfo()实际执行了伪造的版本
  - 伪造的版本将原始APK中的签名信息提供给调用者
  - 调用者验证签名信息，发现运行时获取的签名与目标签名一致，因此认为当前应用没有被重打包
- 使用到的主要技术：通过Java提供的反射机制劫持目标方法

运行结果

- 针对前面具有反重打包支持的应用，劫持getPackageInfo()，使其返回正确的签名，然后重新打包，使用非Android Studio提供的密钥库签名，然后安装运行，结果如右图所示
  - 结果显示重打包后的应用获取到的签名信息与期望值匹配
  - 而实际上该APK对应的证书信息如下图所示，签名的MD5值与期望签名的MD5（右图所示）不一致




```
证书指纹:
        MD5:  83:7D:C6:10:05:E7:09:6F:E2:FF:18:E7:51:B6:7D:2C
        SHA1: 03:C0:EA:1D:7A:38:44:AA:D1:63:51:D6:52:97:52:AE:C9:47:3E:FA
        SHA256: FC:46:1C:03:14:15:BE:C0:90:B3:16:93:EF:79:6E:66:34:EE:70:41:0F:70:A9:A3:51:59:4D:0D:CE:F8:DF:C8
签名算法名称: SHA256withRSA
主体公共密钥算法: 2048 位 RSA 密钥
```

7 课后思考

- 找一个最新的“元气骑士”或其他需要购买道具的Android游戏，尝试通过重打包的方式节省开支
- 思考1：
  - 收集相关资料，探索面对一个复杂的Android应用时，通过“劫持”实现“反•反重打包”操作的劫持时机在哪？
- 思考2：
  - 如何进一步防御此类攻击？

- Android Studio下载地址：
  - https://developer.android.google.cn/studio
- Android开发文档
  - https://developer.android.google.cn/reference/packages