Android逆向入门篇--编译、打包、安装

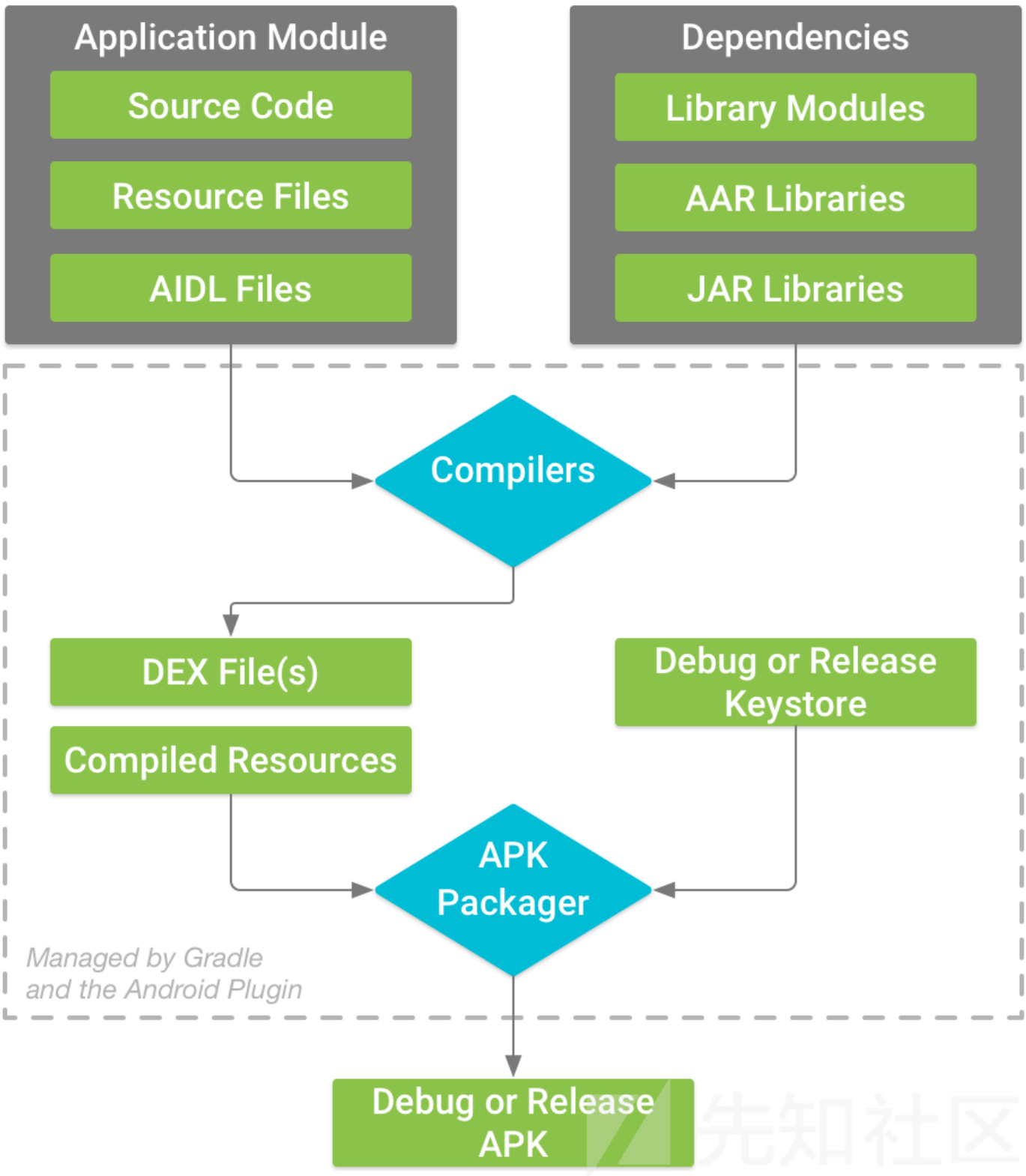
yong夜 / 2019-06-24 09:23:00 / 浏览数 6688 安全技术 移动安全 顶(0) 踩(0)

[TOC]

引言

本片作为Android逆向入门篇，只有先了解APK包的信息，才可以进一步来逆向它

APK打包



APK打包的内容主要有：应用模块也就是自己开发的用到的源代码、资源文件、aidl接口文件，还有就是依赖模块即源代码用到的第三方依赖库如：aar、jar、so文件。

从图中可以看出主要分为以下几步：

第一步：编译，打包

目录结构类似下面所示：

```
android-project/
├── AndroidManifest.xml
├── gen/
├── lib/
│   └── android-support-v4.jar
├── out/
├── res/
│   ├── drawable-xhdpi/
│   │   └── icon.png
│   ├── drawable-xxhdpi/
│   │   └── icon.png
│   ├── drawable-xxxhdpi/
│   │   └── icon.png
│   ├── layout/
│   └── activity_main.xml
└── src/
    └── cn/
        └── androidblog/
            └── testbuild/
                └── MainActivity.java
```

流程

1. 打包资源文件生成R.java，编译aidl生成java接口文件
2. 将源代码编译成DEX（Dalvik Executable) 文件（其中包括 Android 设备上运行的字节码）。
3. 将编译后的文件打包成一个APK压缩包

工具

aapt.exe/aapt2.exe：资源打包工具

javac.exe：将java转成class

dx.jar：将class转成dex文件

资源打包

aapt

通过aapt工具生成R.java和打包到压缩包中的各种编译好的xml文件、未编译的文件、arsc文件。

资源文件中values文件夹中的文件生成了resource.arsc和R.java

```
aapt.exe p -M AndroidManifest.xml -S ./main/res -I android.jar -J ./ -F ./out.apk
```

- p：打包
- -M：AndroidManifest.xml文件路径
- -S：res目录路径
- -A：assets目录路径
- -I：android.jar路径，会用到的一些系统库
- -J 指定生成的R.java的输出目录
- -F 具体指定apk文件的输出

aapt2

编译

从Android Studio
3.0开始，google默认开启了aapt2作为资源编译的编译器，aapt2的出现，为资源的增量编译提供了支持。当然使用过程中也会遇到一些问题，我们可以通过在gradle.prop

- 编译整个目录中的所有资源文件到一个压缩包中，并且全部编译成flat文件

```
aapt2.exe compile -o base.apk -dir E:\AndroidStudioProjects\TestJni\app\src\main\res
```

- 编译单个文件、多个文件到指定目录中

```
aapt2.exe compile -o E:\ E:\AndroidStudioProjects\TesttJni\app\src\main\res\mipmap-xxxhdpi\ic_launcher_round.png
```

链接

- -o：链接进指定压缩包内
- -I：指定android.jar路径
- --manifest：指定AndroidManifest.xml路径
- --java ：指定目录生成R.java（包含包路径，例如包名是com.test，则会生成到./com/test目录下）

需要把所有的flat文件加载后面

```
aapt2.exe link -o .\out.apk -I .\Sdk\platforms\android-28\android.jar --manifest E:\AndroidStudioProjects\TestJni\app\src\main
```

编译aidl文件

sdk\build-tools目录下的aidl.exe工具

- -I 指定import语句的搜索路径，注意-I与目录之间一定不要有空格
- -p 指定系统类的import语句路径，如果是要用到android.os.Bundle系统的类，一定要设置sdk的framework.aidl 路径
- -o 生成java文件的目录，注意-o与目录之间一定不要有空格，而且这设置项一定要在aidl文件路径之前设置

```
aidl -Iaidl -pD:/Android/Sdk/platforms/android-27/framework.aidl -obuild aidl/com/android/vending/billing/IInAppBillingService
```

编译源代码

toClass

javac：jdk自带工具

- -target：生成特定VM版本的class文件，也就是sdk版本
- -bootclasspath：表示编译需要用到的系统库
- -d：生成的class文件存放的目录位置

最后将需要编译的java文件放在文件末尾

```
javac -target 1.8 -bootclasspath platforms\android-28\android.jar -d e:/ java\com\testjni\*.java
```

todex

sdk\build-tools下的lib目录下的dx.jar工具

- --dex：将class文件转成dex文件
- --output：指定生成dex文件到具体位置

```
java -jar dx.jar --dex --ouput=.\classes.dex .\com\testjni\*.class
```

打包

由于apkbuilder工具被废弃了，我们可以手动将文件放到aapt生成的apk文件中

第二步：签名

使用签名工具对打包好的压缩包签名后才可以被android系统安装，签名后的证书文件放在META-INF目录下

公钥证书（也称为数字证书或身份证书）包含公钥/私钥对的公钥，对apk的签名也就是将公钥附加在apk上，充当指纹的作用，用来将APK唯一关联到开发者手上的私钥上。

密钥库是一种包含一个或多个私钥的二进制文件。我们先构建自己的密钥库

构建密钥库

- genkey：生成密钥库
- alias：密钥库别名
- keyalg：密钥算法RSA
- validity：证书有效期40000天
- keystore：密钥库名称

```
keytool -genkey -alias demo.keystore -keyalg RSA -validity 40000 -keystore demo.keystore
```

接着会让输入密钥密码，证书信息等，下面命令是查看自己这个密钥库中的详细信息的命令

```
keytool -list -keystore demo.keystore -v
```

apksigner

- sign：签署数字证书
- --ks：指定密钥库

```
java -jar apksigner.jar sign --ks demo.keystore demo.apk
```

jarsigner

jdk工具：jarsigner.exe

- keystore：指定密钥库文件置
- signedjar：签名后文件存储的位置

```
jarsigner.exe -verbose -keystore <■■■> -signedjar <■■■■apk■■> <■■■■■■apk■■> <■■■■■■>
```

第三步 zipalign对齐

为了减少RAM的使用，目的确保所有未压缩的数据在4字节边界上对其，根据不同签名工具，具体对齐时间不定

apksigner签名apk，在签名之前进行对齐，否则会致使签名无效

jarsigner签名apk，在签名之后进行对齐

- 4：表示4字节对齐

```
zipalign.exe 4 base.apk aligned.apk
```

APK安装

/system/app：系统自带的应用程序，需要ROOT权限方可删除

/data/app：用户安装应程序时，将apk文件复制到这里

/vendor/app：设备商的应用程序

/data/app-private：受DRM保护(数字版权管理)的app

/data/data：应用存放数据的地方

/data/dalvik-cache：将apk中的dex文件安装到这里

系统安装(放入就安装)

- 将安装包放入/system/app、/data/app、/data/app，/data/app-private目录中即可实现自动安装
- 如果删除/system/app、/data/app、/data/app、/data/app-private目录中的安装包，即可实现应用删除操作
- /system/app和/vendor/system下的应用需要ROOT权限方可删除

安装功能主要有systemServer的子类PackageManagerService来实现，如下面这里会注册一个观察者mSystemInstallObserver，监听/system/app内的应用安装包情况，

上面这些app安装目录的监听原理是大致相同的

```
// Collect ordinary system packages.
           File systemAppDir = new File(Environment.getRootDirectory(), "app");
           mSystemInstallObserver = new AppDirObserver(
               systemAppDir.getPath(), OBSERVER_EVENTS, true, false);
           mSystemInstallObserver.startWatching();
           scanDirLI(systemAppDir, PackageParser.PARSE_IS_SYSTEM
                   | PackageParser.PARSE_IS_SYSTEM_DIR, scanMode, 0);

private final class AppDirObserver extends FileObserver {
 ....
   if ((event&ADD_EVENTS) != 0) {
   ....
```

```
    p = scanPackageLI(fullPath, flags,
                              SCAN_MONITOR | SCAN_NO_PATHS | SCAN_UPDATE_TIME,
                              System.currentTimeMillis(), UserHandle.ALL);
```

在scanPackageLI方法中，具体在这里进行了安装操作

```
//invoke installer to do the actual installation
               int ret = mInstaller.install(pkgName, pkg.applicationInfo.uid,
                       pkg.applicationInfo.uid);
```

网络下载安装

从网络上下载下来apk安装包不管是手动点击安装还是应用检测安装，一般都会使用PackageManagerService的installPackage方法进行安装，这个方法向内部查看几层发现

```
/* Called when a downloaded package installation has been confirmed by the user */
   public void installPackage(
           final Uri packageURI, final IPackageInstallObserver observer, final int flags) {
       installPackage(packageURI, observer, flags, null);
   }
```

从截取的片段可以看到，这个方法出了实例化一个观察者外，主要通过传递安装参数给handle，下面我们看一下下handle的处理方法

```
public void installPackageWithVerificationAndEncryption(Uri packageURI,
           IPackageInstallObserver observer, int flags, String installerPackageName,
           VerificationParams verificationParams, ContainerEncryptionParams encryptionParams)
           .......
            observer.packageInstalled("", PackageManager.INSTALL_FAILED_USER_RESTRICTED);
            .......
final Message msg = mHandler.obtainMessage(INIT_COPY);
       msg.obj = new InstallParams(packageURI, observer, filteredFlags, installerPackageName,
               verificationParams, encryptionParams, user);
       mHandler.sendMessage(msg);
```

而处理message的hanle方法主要，将初始化安装参数，并接着发送标识为MCS_BOUND的message，而在处理这个message的时候，调用了startCopy，接着调用handle

```
void doHandleMessage(Message msg) {
           switch (msg.what) {
               case INIT_COPY: {
                   ......
                 mPendingInstalls.add(idx, params);
                       // Already bound to the service. Just make
                       // sure we trigger off processing the first request.
                       if (idx == 0) {
                           mHandler.sendEmptyMessage(MCS_BOUND);
                       }

case MCS_BOUND: {
.......
} else if (mPendingInstalls.size() > 0) {
                       HandlerParams params = mPendingInstalls.get(0);
                       if (params != null) {
                           if (params.startCopy()) {

private void installPackageLI(InstallArgs args,
           boolean newInstall, PackageInstalledInfo res) {
           ............
           final PackageParser.Package pkg = pp.parsePackage(tmpPackageFile,
               null, mMetrics, parseFlags);
```

adb安装

通过adb命令安装APK安装包，主要分为两步：

- adb push xxx.apk /data/local/tmp，先将apk文件传送到设备临时目录下
- pm install /data/local/tmp/xxx.apk

这里用到了pm类的runinstall方法，内部调用了installPackageWithVerification，通过下面这个跨进程接口调用了安装包服务来执行安装操作，也就回到了网络下载后的安

```
mPm = IPackageManager.Stub.asInterface(ServiceManager.getService("package"));
```

小结

【1】打包过程中，需要先把资源、aidl文件转成java文件，然后同所有java源码一起打包成.class再到dex

参考

【1】谷歌官方打包流程：https://developer.android.com/studio/build?hl=zh-cn

【2】apktool官网 https://ibotpeaches.github.io/Apktool/

【3】APK安装过程及原理详解 https://blog.csdn.net/hdhd588/article/details/6739281

上一篇：分析MS06-055 IE 栈溢出漏洞 下一篇：macOS恶意软件驻留技术分析

1. 2 条回复

132****0127 2019-09-27 11:39:41

你好，如果我想在android APP上调用dx.jar，该怎么调用？网上找的都是cmd执行bat的

0 回复Ta

yong夜 2019-09-30 20:50:45

@132****0127 我没调用过，不过这个好像是一个可执行jar包，你要是调用人家库，你的自己编译一个可导入的库，应该是这样的

0 回复Ta

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板