DANIELE ALTOMARE

Android bits

Android Reverse Engineering 101 – Part 1

DANIELE.ALTOMARE@GMAIL.COM   NOVEMBER 11, 2015   ANDROID

This is the first in a series of articles about reverse engineering Android applications.

In this series I will cover the anatomy of the **APK** and **AAR** package formats and few tools commonly used to reverse engineering or inspecting applications: **aapt**, **dex2jar**, **apktool** and **Androguard**.

Is this just for hackers?

Short answer is no.

There are many reasons why you would take into consideration these tools besides trying to hack or inject malicious code into an application, specially if you are a developer.

First, there may be an application with a particular layout or animation which you would like to replicate: using these tools you could access the XML resource files of interest.

Then, you may have a look at how specific business logic has been implemented by an application: it's not a matter of stealing third-party software, but simply double check if you can improve your own code or get any useful hint for improvements.

Last but not least, it is always a good practice to test your own application for security reasons: to check if the code or the resources have been effectively obfuscated or to be sure that unwanted files have not been packaged into the final release APK. You would be surprised about how many APK files are full of information like API keys, authentication tokens or unused resources.

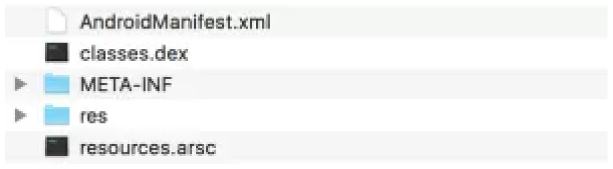
APK format

The **APK** bundle is the format used to package any application you develop or that you can get from Google Play Store or any other channel. In other words, for each application present on your device, there is a corresponding APK file (this is true also for pre-installed applications).

An APK file is essentially a ZIP file, so you can get one, rename it and then extract it to have access to its content.

Entry	Notes
AndroidManifest.xml	the manifest file in binary XML format.
classes.dex	application code compiled in the dex format.
resources.arsc	file containing precompiled application resources, in binary XML.
res/	folder containing resources not compiled into resources.arsc
assets/	optional folder containing applications assets, which can be retrieved by AssetManager.
lib/	optional folder containing compiled code - i.e. native code libraries.

Entry	Notes
META-INF/	folder containing the MANIFEST.MF file, which stores meta data about the contents of the JAR. The signature of the APK is also stored in this folder.


AndroidManifest.xml
classes.dex
META-INF
res
resources.arsc

APK file content example.

WHAT's DEX?

In short, DEX, or Dalvik Executable is a file format that contains compiled code written for Android and can be interpreted by the Dalvik virtual machine or by the Android Runtime (ART).

When an APK file is produced by the Android build system (like when you "run" your application project from Android Studio), the Java classes are first compiled into `.class` files and later a tool called **dx** will convert these files in the DEX format. dx is part of the Android toolchain, the Build Tools, and you can find it at this location:

`$ANDROID_SDK/build-tools`

Specific details about the layout and the contents of a DEX file can be found here.

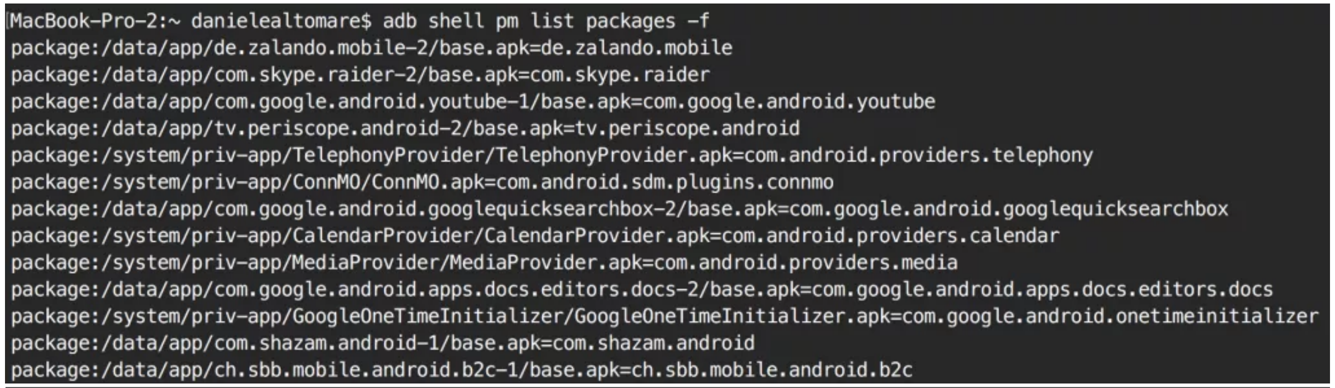
How can you get an apk file?

There are few ways to do that:

- if you need any arbitrary application, you can rely on online services like this one, getting the APK file directly from your desktop browser.
- if the application is installed on your device, you could simply rely on a backup utility like this one, and it will make a copy of the apk on a public folder on your device memory or on the SD card.
- inside folder `/system/app`, you can find pre-installed applications, such as: calculator, Chrome, camera, ... it depends on the particular ROM installed on your device.
- inside folder `/data/app`, you can find user installed applications.

In order to get an APK from your device, you can first list the available packages with the following command (remember to attach your device to the USB):

```
adb shell pm list packages -f
```



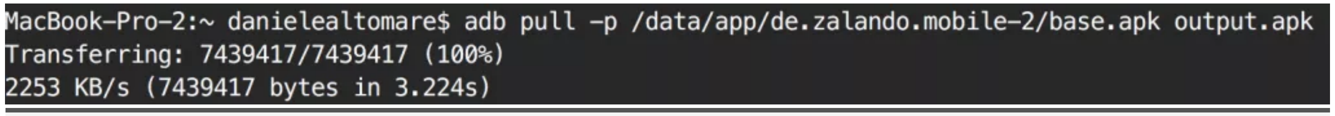
```
MacBook-Pro-2:~ danielealtomare$ adb shell pm list packages -f
package:/data/app/de.zalando.mobile-2/base.apk=de.zalando.mobile
package:/data/app/com.skype.raider-2/base.apk=com.skype.raider
package:/data/app/com.google.android.youtube-1/base.apk=com.google.android.youtube
package:/data/app/tv.periscope.android-2/base.apk=tv.periscope.android
package:/system/priv-app/TelephonyProvider/TelephonyProvider.apk=com.android.providers.telephony
package:/system/priv-app/ConnMO/ConnMO.apk=com.android.sdm.plugins.connmo
package:/data/app/com.google.android.googlequicksearchbox-2/base.apk=com.google.android.googlequicksearchbox
package:/system/priv-app/CalendarProvider/CalendarProvider.apk=com.android.providers.calendar
package:/system/priv-app/MediaProvider/MediaProvider.apk=com.android.providers.media
package:/data/app/com.google.android.apps.docs.editors.docs-2/base.apk=com.google.android.apps.docs.editors.docs
package:/system/priv-app/GoogleOneTimeInitializer/GoogleOneTimeInitializer.apk=com.google.android.onetimeinitializer
package:/data/app/com.shazam.android-1/base.apk=com.shazam.android
package:/data/app/ch.sbb.mobile.android.b2c-1/base.apk=ch.sbb.mobile.android.b2c
```

adb shell pm list packages -f

Having the path of the APK file, you can now pull it:

```
adb pull -p PATH/base.apk OUTPUT.apk
```

`-p` option simply displays the transfer progress, while the output filename is not mandatory, if omitted, `base.apk` is used instead.


```
MacBook-Pro-2:~ danielealtomare$ adb pull -p /data/app/de.zalando.mobile-2/base.apk output.apk
Transferring: 7439417/7439417 (100%)
2253 KB/s (7439417 bytes in 3.224s)
```

adb pull -p

You may wonder how backup applications are able to give you an APK: in fact, if you try to access the `/data/app` folder from any file manager application installed on your device, you can see that is not possible, access is denied.

But it's indeed true that programmatically you can have access to the APK of the user installed applications.

First you need to retrieve the list of the applications:

```
final Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

List<ResolveInfo> infos = getPackageManager().queryIntentActivities(mainIntent, 0);
```

Then, through the `ResolveInfo` , you can access the `publicSourceDir` field of the `ApplicationInfo` class, which is the full path to the publicly available parts of `sourceDir` , including resources and manifest.

```
File apkFile = new File(pkgInfo.activityInfo.applicationInfo.publicSourceDir);
if(apkFile.exists()) {
    ...
}
```

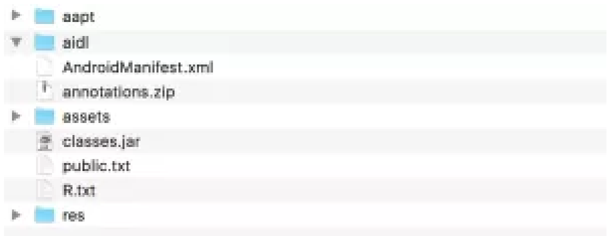
AAR format

The **AAR** bundle is the binary distribution of an Android Library Project: for example, the Android Support Library you may use in your application is packaged using this format. Also, if your Android project is set to release a library and not an application for the store, the format would this one as well.

An AAR file is essentially a ZIP file, so you can get one, rename it and then extract it to have access to its content.

These are the main entries you will find in an AAR file, even though there could be other files.

Entry	Required	Notes
AndroidManifest.xml	mandatory	the manifest file in plain XML.
classes.jar	mandatory	Java classes of the library.
res/	mandatory	folder containing resources used by the library.
R.txt	mandatory	output of aapt with `--output-text-symbols`. It's basically a list of all resources referenced by the library (strings, colors, dimens, attrs, layouts, ...).
assets/	optional	folder containing assets used by the library.
libs/*.jar	optional	folder containing any external library.
jni//*.so	optional	folder containing native libraries.
proguard.txt	optional	Proguard configuration file.
lint.jar	optional	custom Lint rules.


```
▸ aapt
▾ aidl
  AndroidManifest.xml
  annotations.zip
▸ assets
  classes.jar
  public.txt
  R.txt
▸ res
```

AppCompat-v7 23.1.0 aar file content.

One difference, compared to the apk, is the format of the `AndroidManifest.xml` file and the resources located in the `res` folder: in the aar package they are in plain XML, so you can easily open them.

Please note, for example, that the Support Library you usually declare as a dependency in your projects, come in the aar format and you can retrieve them at: `$ANDROID_SDK/extras/android/m2repository/com/android`

In the rest of this series, I will focus anyway on the APK format only, because it's the one used to package the applications installed on any device and distributed through Google Play Store or other channels.

In the next article, I will speak about **aapt** and **dex2jar** tools and how you can use them to retrieve important information about the apk file under analysis.

Related

Android Reverse Engineering 101 – Part 2
November 15, 2015
In "android"

Android Reverse Engineering 101 – Part 4
November 29, 2015
In "android"

Android Reverse Engineering 101 – Part 3
November 21, 2015
In "android"

ANDROID / REVERSE ENGINEERING / SECURITY

7 Comments

Philippe Banwarth 2 YEARS AGO Reply

You can also pull the apk with `adb` even from a non rooted device.
The `/data/app/` file access mode is `–x` : You cannot list the content but you can access it, as long as you know the filename, for example using a command like `adb shell pm list packages -f`. It also work with apks in `/system/app` and `/system/priv-app/`

daniele.altomare@gmail.com 2 YEARS AGO Reply

You're right, thanks for pointing it out: in fact I forgot there's this option. I've updated my post accordingly. Thanks again.

calmarj 2 YEARS AGO Reply

I like your wordpress theme, can you give me a link to it?

daniele.altomare@gmail.com 2 YEARS AGO Reply

It’s the default theme, Twenty Fifteen: https://wordpress.org/themes/twentyfifteen/

Xmodgames 1 YEAR AGO Reply

very informative in nature as its level of knowledge used in this article is a very important factor for me Thanks for sharing this article. All the points described here are very easy to understand.

Kirit Bhayani 11 MONTHS AGO Reply

how to get native code like jni folder including .mk & .cpp file?

daniele.altomare@gmail.com 11 MONTHS AGO Reply

Hello! You can’t get .cpp source code of the .so native libraries with apktool or other tools mentioned. You need for example IDA to decompile code (ARM, x86, ...): the are different versions based on the

architecture for which you have the .so files.
More info here: https://reverseengineering.stackexchange.com/questions/4624/how-do-i-reverse-engineer-so-files-found-in-android-apks

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Post Comment

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.