



jack mannino

SECURITY, FOOD, FUN!!

tuesday, september 28, 2010

Reversing Android Apps 101

First blog post in a long time, so I figured I should start back off with something I've been getting a lot of questions about lately, which is how to get started with learning more about Android application security.

I think the absolute best way to understand what's going on is to start looking at real world applications. This tutorial will show you how to download an Android application and convert it back into a format that we prefer, which is its original Java code. Hopefully this will be the first tutorial of many, so be sure to check back from time to time.

I hope this post can be of use to developers as well, in order to better understand that the code they allow to be distributed on mobile devices can easily be decompiled. If you are storing hardcoded credentials or reference other things you don't want the user (or bad guys) to know about, you should re-evaluate that design choice.

My disclaimer is that this should not be used for malicious purposes, etc etc and all of the other stuff no one will likely listen to.

To get started, you will need these applications on your box:

- Android SDK
- dex2jar
- apktool (JAR and dependencies)
- JD (or a similar Java decompiler)

The first step is to download the Android SDK (Software Development Kit). You will rely on this heavily when testing applications and interacting with Android devices. There are many useful utilities bundled with it including Android Debug Bridge (ADB) and the emulator.

The next step is to download an interesting application to rip open. Most of us are Twitter fans, so why don't we take a look at the Twitter application for Android?

At this point, you have 2 choices:

blog archive

- ▼ 2010 (13)
 - ▼ September (1)
 - [Reversing Android Apps 101](#)
 - ▶ May (1)
 - ▶ April (1)
 - ▶ March (2)
 - ▶ February (4)
 - ▶ January (4)
- ▶ 2009 (14)
- ▶ 2008 (2)

about me



Jack Mannino

CEO of nVisium
Security, die-hard
New York Mets
baseball fan, and all

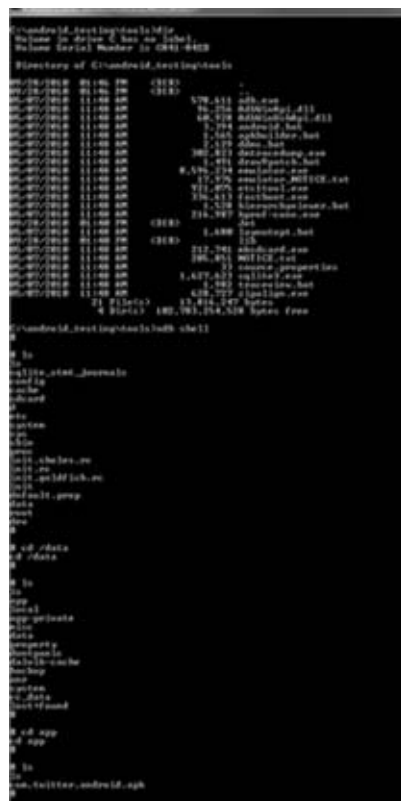
around great guy! =)

[View my complete profile](#)

- By default, the virtual devices available using the emulator that comes with the Android SDK do not have the Google Market application installed. You can either 1) spend time finding ways around this or 2) install an image that has the software pre-loaded. Here is an excellent tutorial on how to go that route, although you may want to use a more up to date image than the ones referenced. They use older versions of Android.

Assuming you have either plugged your phone in or started an Android virtual machine, in the tools folder of the Android SDK folder, you'll find several utilities including ADB. I recommend familiarizing yourself with it, as you will use it almost constantly if you are working with the Android platform.

ADB allows us to do many things including connect to the device and have shell access. We first need to figure out where the application we downloaded is stored. Under the `/data/app/` directory, we will find all of the APK files for applications we've downloaded. Under `/system/app` is all of the applications included with your Android distribution. For this tutorial, we are concerned with `/data/app` as this is where our Twitter application will be located.



Now that we know what we want to download, we need to transfer the file to our local system for further analysis. This can be achieved also by using ADB.

We will not be using the shell this time, but rather the "pull" command which allows us to select and retrieve files from the Android device. The general format for this command is: adb pull apk_file destination_directory

```
C:\android_testing\tools>adb pull /data/app/com.twitter.android.apk ../downloaded_apps
1158 KB/s (2183480 bytes in 1.840s)

C:\android_testing\tools>cd ../downloaded_apps

C:\android_testing\downloaded_apps>dir
Volume in drive C has no label.
Volume Serial Number is C841-84EB

Directory of C:\android_testing\downloaded_apps
09/28/2010  07:29 PM    <DIR>          .
09/28/2010  07:29 PM    <DIR>          ..
09/28/2010  07:29 PM             2,183,480 con.twitter.android.apk
               1 File(s)              2,183,480 bytes
               2 Dir(s)      102,812,626 bytes free

C:\android_testing\downloaded_apps>
```

Now that we have the APK file, we'll need to unzip it. The APK is simply in standard ZIP format, so any of your existing ZIP compatible utilities will work.

After unzipping the APK file, there are two things you should be primarily concerned with at this point: looking at the manifest file, and getting the .DEX file into a more desirable format to work with. The manifest contains juicy information like permissions, intent filters, and lots more. If you aren't familiar with the manifest file and how its used with Android applications, I highly recommend reading the documentation on the Android development site:

Manifest Intro

If you try to open the AndroidManifest.xml file that was just unzipped, you'll find that it isn't in plain text. We'll use apktool at this point to convert it into a format we are comfortable with. We'll use apktool to decode the entire APK, and then the manifest file should be much easier on the eyes:

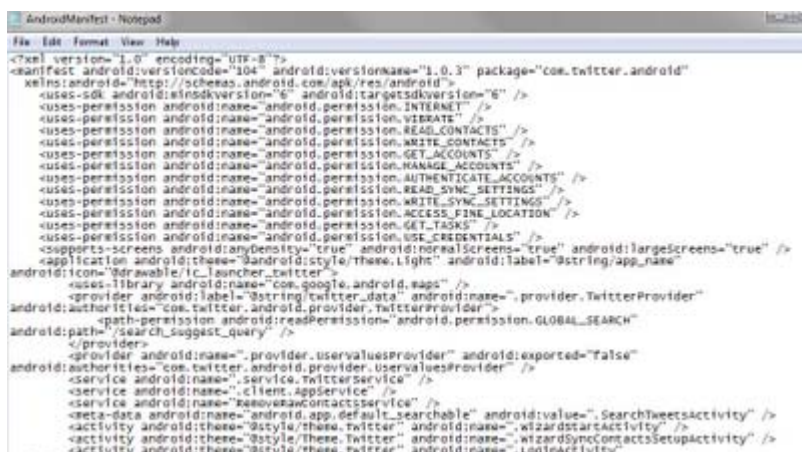
```
C:\android_testing\apktool>apktool.bat d c:\android_testing\downloaded_apps\con.
twitter.android.apk
I: Baksmaling...
I: Loading resource table...
I: Decoding resources...
I: Loading resource table from file: C:\Users\Jack Mannino\apktool\framework\1.a
pk
I: Copying assets and libs...

C:\android_testing\apktool>dir
Volume in drive C has no label.
Volume Serial Number is C841-84EB

Directory of C:\android_testing\apktool
09/28/2010  07:34 PM    <DIR>          .
09/28/2010  07:34 PM    <DIR>          ..
05/13/2010  01:28 PM             5,441,083 aapt.exe
09/03/2010  12:13 PM                69 apktool.bat
09/03/2010  09:58 AM             1,343,302 apktool.jar
09/28/2010  04:23 PM             1,204,866 apktool1.3.2.tar.gz2
09/28/2010  07:34 PM    <DIR>          con.twitter.android
               4 File(s)              7,989,320 bytes
               3 Dir(s)      102,807,003,136 bytes free

C:\android_testing\apktool>cd con.twitter.android
C:\android_testing\apktool\con.twitter.android>
```

After running apktool, this is what the AndroidManifest.xml file should look like when viewed in a text editor or Eclipse:



```

<?xml version="1.0" encoding="UTF-8"?>
<manifest android:versionCode="104" android:versionName="1.0.3" package="com.twitter.android">
  <uses-sdk android:minSdkVersion="6" android:targetSdkVersion="6" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.READ_CONTACTS" />
  <uses-permission android:name="android.permission.WRITE_CONTACTS" />
  <uses-permission android:name="android.permission.GET_ACCOUNTS" />
  <uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
  <uses-permission android:name="android.permission.READ_SYNC_SETTINGS" />
  <uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.GET_TASKS" />
  <uses-permission android:name="android.permission.USE_CREDENTIALS" />
  <supports-screens android:anyDensity="true" android:normalScreens="true" android:largeScreens="true" />
  <application android:theme="@android:style/Theme.Light" android:label="@string/app_name">
    <android:icon="@drawable/ic_launcher_twitter">
      <uses-library android:name="com.google.android.maps" />
    </android:icon>
    <provider android:label="@string/twitter_data" android:name=".provider.TwitterProvider">
      <path-permission android:readPermission="android.permission.GLOBAL_SEARCH" android:path="/search_suggest_query" />
    </provider>
    <provider android:name=".provider.UserValuesProvider" android:exported="false">
      <authorities="com.twitter.android.provider.UserValuesProvider" />
    </provider>
    <service android:name=".service.TwitterService" />
    <service android:name=".client.AppService" />
    <service android:name=".service.RemoveNewContactService" />
    <meta-data android:name="android.app.default_searchable" android:value=".SearchTweetsActivity" />
    <activity android:theme="@style/Theme.Twitter" android:name=".WizardStartActivity" />
    <activity android:theme="@style/Theme.Twitter" android:name=".WizardSyncContactsSetupActivity" />
    <activity android:theme="@style/Theme.Twitter" android:name=".LoginActivity" />
  </application>
</manifest>

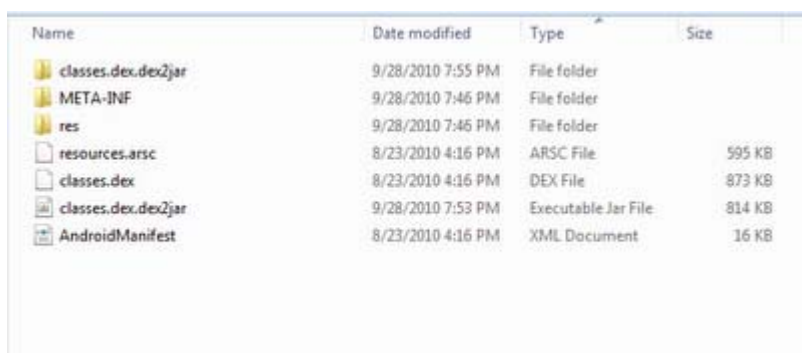
```

The classes.dex file in the originally unzipped APK file is the crown jewel here. It contains all of the application's code, but in the Dalvik Executable format.

There are tools such as dexdexer and apktool that will convert .DEX into smali assembly format, but again we are concerned with looking at the Java code. We can use dex2jar to achieve this:

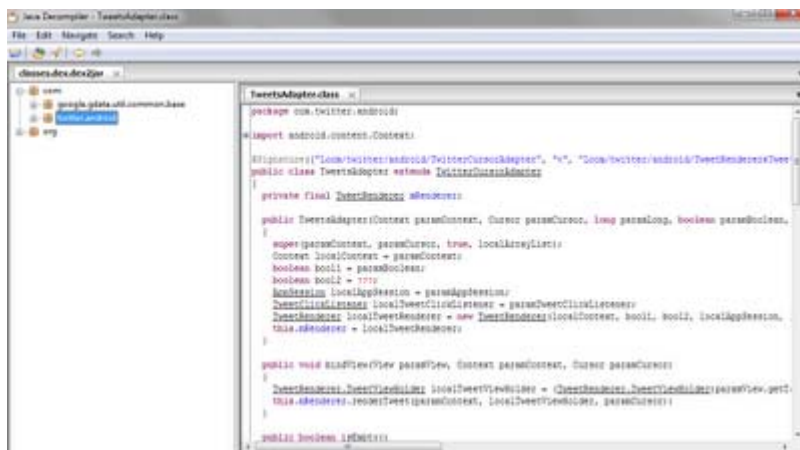
```
dex2jar.bat path_to_classes.dex
```

Once we've run dex2jar, there will be a classes.dex.dex2jar.jar file in the folder where the original .DEX file was located. Simply unzip this JAR file. Upon unzipping it, we see a bunch of .class files.



Name	Date modified	Type	Size
classes.dex.dex2jar	9/28/2010 7:55 PM	File folder	
META-INF	9/28/2010 7:46 PM	File folder	
res	9/28/2010 7:46 PM	File folder	
resources.arsc	8/23/2010 4:16 PM	ARSC File	595 KB
classes.dex	8/23/2010 4:16 PM	DEX File	873 KB
classes.dex.dex2jar	9/28/2010 7:53 PM	Executable Jar File	814 KB
AndroidManifest	8/23/2010 4:16 PM	XML Document	16 KB

Nothing to worry about, this is familiar territory at this point. Simply use your favorite Java decompiler to convert from bytecode to easily readable code.



We now have complete access to the entire client application. In the tutorials to come, I'll show you some things you should be paying close attention to when reviewing or designing an Android application.

posted by jack mannino at 5:03 pm

2 comments:

 **mephisto** said...

You can also use the AXMLPrinter2 utility (<http://code.google.com/p/android4me/downloads/detail?name=AXMLPrinter2.zip&can=2&q=>) to decode the Manifest.xml file and others that are encoded.

September 28, 2010 9:57 PM

 **jack mannino** said...

Thanks, I haven't used it but will take a look. Someone on Twitter also recommended undx as an alternative to dex2jar. Does essentially the same thing.

September 29, 2010 1:19 PM

Post a Comment

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)