Way of the Android Cracker

September, 2010, rev. 1, by lohan+

Lesson 0: Laying the Foundation

Includes: crackme0.apk, crackme0-solved.apk, smali.uew, testsign.jar, example.smali

cracking, v. - *understanding, broadly individuating, locating exactly and eliminating or suspending or deferring one or more protection schemes inside a software application you do not possess the source code of.*

Contents

Background

Mobile application development is the future. Every new generation of phones brings increased processing speed and power, better screens and longer battery life. At the dawn of the computing age, experts claimed computers would grow larger and smaller in number but the opposite trend has been true. Mobile platforms are a logical extension of this trend. Now you can have your MP3 player, e-mail, web, GPS, YouTube, social networks, etc. in your pocket. Mobile development is still young and if you begin your journey into the Way of the Cracker you can observe as protection methods are born, grow and evolve and crack them every step of the way.

This tutorial covers cracking on the Linux-based operating system called Android. It's open source and optimized for running on mobile platforms with their limitations on processing power and memory. Applications are contained in .apk files which are just special .zip files that have been signed with a certificate. The certificate is used as a fingerprint by the Market and Android to uniquely identify an App and Developer. Any modification of an .apk file invalidates the signature which means the .apk file won't install. Cracked or modified Apps must be re-signed. This is often done quite easily with a randomly generated certificate. There are special circumstances where a specifically generated certificate must be used and this will be covered in a later tutorial.

As of now there are two ways for developers to protect their mobile Apps. The main line of defense is the official App Market. Users pay for Apps through the Market which then allows the user to download the .apk file to their phone. Most of the time it is a full version .apk but sometimes it is merely a special Key or Donate version that the main App checks for to unlock features. The problem with this, of course, is that the .apk is left unprotected in the user's hands, literally. The only protection is the directory where the purchased .apk is stored requires root permissions which you do not have unless you *"root"* your phone. Rooting is a straightforward procedure, slightly different for each device. It is necessary to root your phone to install a custom Android ROM so instructions for rooting are widely available. On top of this, the Android Market allows for a full refund up to 24 hours from purchase. So ripping hundreds of Apps requires only an initial, refundable investment of ~5$, a rooted phone and time. This makes cracking any App or App Key that can simply be downloaded not worth your skills. You may be thinking "But lohan, if the Market is so vulnerable, why is it still viable?" My guess is people are lazy. Not everyone is willing to even find cracked versions of Apps, let alone root their phone and go through the trouble of getting a refund for an App that probably cost around 1$. Another reason might be that many Apps are made by one or a couple of developers and the App listing often has the developers full name which may help foster a more personal connection between users and devs. It sounds silly but I have been more than once criticized by a fan for cracking an App *on an App cracking forum*.

The alternative to the Market is exactly what you see with computer software: serials, licenses, server validation, etc. These methods will require The Way of the Cracker. These are Apps which you can usually download for free but have features that can be unlocked by some means. As of now, many programs rely entirely on Market protection but as mobile Apps begin to represent larger and larger investments of time and labor we will see increased protection measures and products offering code protection available to developers. Because the Dalvik supports reflection and the virtual machine has to be able to interpret the byte code no obfuscation can ever hope to be complete. It is the same case with .NET code. Obfuscation products like ProGuard may become more advanced with time but intense obfuscation will likely have a very negative impact on performance.

The program logic for an Android App is contained in a file called classes.dex in the root of the .apk file. It is a variant of Java's Jar format called the DEX (**D**alvik **EX**ecutable) format and can be disassembled to produce Dalvik byte code. It is similar to Java byte code but optimized for limited hardware. The two main tools for disassembling .apk files are Smali/Baksmali and Apktool. Smali/Baksmali is an assembler/disassembler which decompiles only the code while Apktool acts as a wrapper for Smali/Baksmali in addition to decoding resources files and images in addition to debugging without the original source, which will be covered in a later tutorial.

This first tutorial will hold your hand a lot more than I enjoy but I do it anyway because Android cracking is new and information is still somewhat sparse. I do, however, skip over a few details that are covered much better elsewhere such as how to use DOS or Linux. If you don't know your way around DOS or Linux then take a look at some cheat sheets with your good friend Google. Future tutorials will assume a higher skill level and will focus more on techniques and the specifics of Dalvik. These tutorials assume you are running Windows. If you are using Linux/Mac you will be smart/cool enough to figure out what to do differently.

Further reading:

- http://developer.android.com/guide/publishing/app-signing.html
- http://en.wikipedia.org/wiki/APK_(file_format)
- http://www.dalvikvm.com/
- http://en.wikipedia.org/wiki/Dalvik_virtual_machine
- http://en.wikipedia.org/wiki/Obfuscated_code

Setting up the Environment

First, you will need a working Java environment. If you are not sure if you have Java installed go here: http://www.java.com/en/download/index.jsp. There is a link called "Do I have Java?" that should sort you out. If you don't have it, install at least the JRE (**J**ava **R**untime **E**nvironment) but I would suggest also getting the JDK (**J**ava **D**evelopment **K**it) because it includes jarsigner. You can get the JDK here: http://www.oracle.com/technetwork/java/javase/downloads/index.html. When you are finished installing, test to make sure java is working by opening a command prompt and typing:

```
java –version
```

If you get an error, it could just be that your PATH variable was not set correctly. PATH contains a list of directories to search when you issue a command. So you type java and it looks in all the PATH directories for java and when it's not found results in an error. We just need to add either the JRE or JDK path. For the JRE version I have installed it is C:\Program Files\java\jre6\bin and for the JDK it is C:\Program Files\java\jdk1.6.0_21\bin. Make sure what yours is and follow these instructions for setting your path: http://www.java.com/en/download/help/path.xml

Next, you will need the android-sdk which contains various useful tools including the all-important ADB or **A**ndroid **D**ebug **B**ridge. This tool allows you to push/pull files, install/uninstall Apps or connect to the shell of the phone or emulated device. The android-sdk also comes with Android emulators so you do not even need a phone to begin cracking Apps.

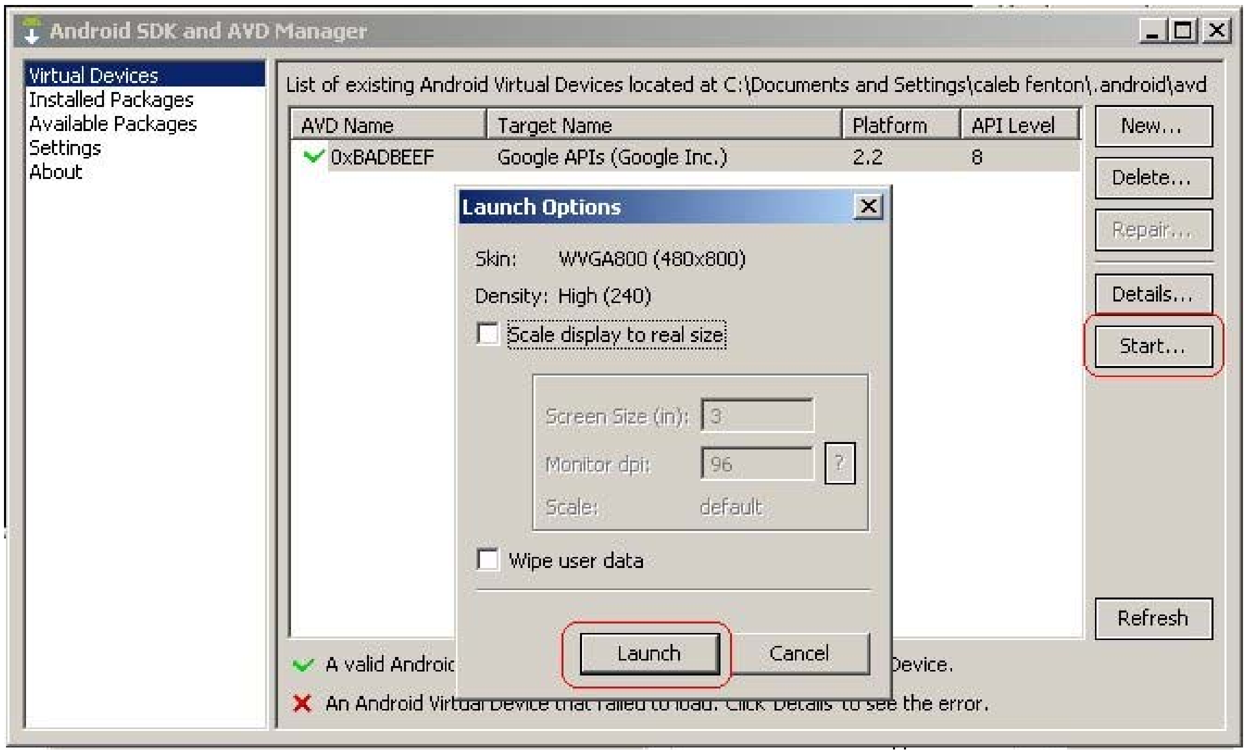
- Begin by downloading the SDK here: http://developer.android.com/sdk/index.html
  I suggest installing it to an easy-to-reach location such as C:\android-sdk or %HOMEDRIVE%\android-sdk
- Follow the well-written documentation for setting up the SDK here:
  http://developer.android.com/sdk/installing.html

- Download the *Usb Driver package*, which is required for ADB on Windows, and follow the steps provided. Instructions for installing the driver are at the Android Developer website.

You will also need at least one platform if you wish to run an emulator which is highly advisable considering you will want to test your cracked Apps in several environments. For more involved cracks everything might work fine on your phone but fail on another because your ROM or version is slightly different. Get the latest version of "*Google APIs by Google Inc*". The other choice, *"SDK Platform Android"*, is stripped down and does not have many of the necessary libraries your Apps will need and attempts to install will result in a missing library error from ADB.

Optional: Setting up an Emulator

- Open the android-sdk folder and start SDK Manager.exe. If you have an older SDK tools version it will be called SDK Setup.exe. It will run and show you a list of packages to install. Close that Window and proceed to the Virtual Devices selection in the list box on the left.
- Click the *New...* button.
  A new dialog will open and you can give it any name you wish.
  Something clever that makes you giggle every time you see it works well.
- In the Target drop down select the Android version you want the emulator to be.
- For SD Card, in the Size box put something like 100 MiB.
  You will only likely ever need a small amount of space on the emulated SD card for cracking and testing.
- For Skin, Built-in resolution can be left as default. If you prefer a large screen, as I do select WVGA800.
- For Hardware, this section allows you to setup the details of the hardware of your virtual phone. You can install GPS, increase RAM of cache size, etc. Be warned that some of these options can impair the emulated phone's ability to connect through your internet. One that is problematic is the GSM Modem support. If you wish to add all the options click New and then hold down enter until all of the different options are added.
- Click *Create AVD*.
  Once you have created this virtual device you will not be able to modify it easily but as you have just seen it is not difficult to simply create a new configuration.
- After the AVD is created a message box will appear and give you the results. Click *OK*.
- Select your new virtual device and click the *Start...* button.
- You have two options: Scale display and Wipe user data. I have never used scale but Wipe user data is useful as it will completely format the user data partition which is a complete factory reset.


Android SDK and AVD Manager

List of existing Android Virtual Devices located at C:\Documents and Settings\caleb fenton\.android\avd

AVD Name	Target Name	Platform	API Level
✓ 0xBADBEEF	Google APIs (Google Inc.)	2.2	8

Launch Options

Skin: WVGA800 (480x800)

Density: High (240)

[ ] Scale display to real size

Screen Size (in): 3

Monitor dpi: 96

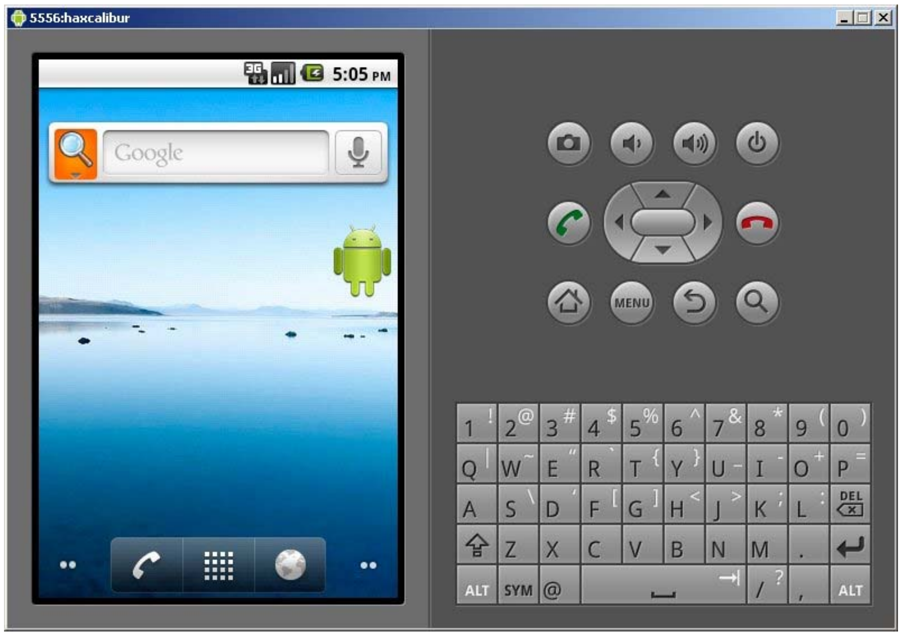
Scale: default

[ ] Wipe user data

✓ A valid Androic … Device.

✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

This step will take a few minutes. While it is loading we will proceed to the next step of testing ADB. If you do not have an actual phone then read on anyway so you will know what to do. When you're finished you should see something like this:

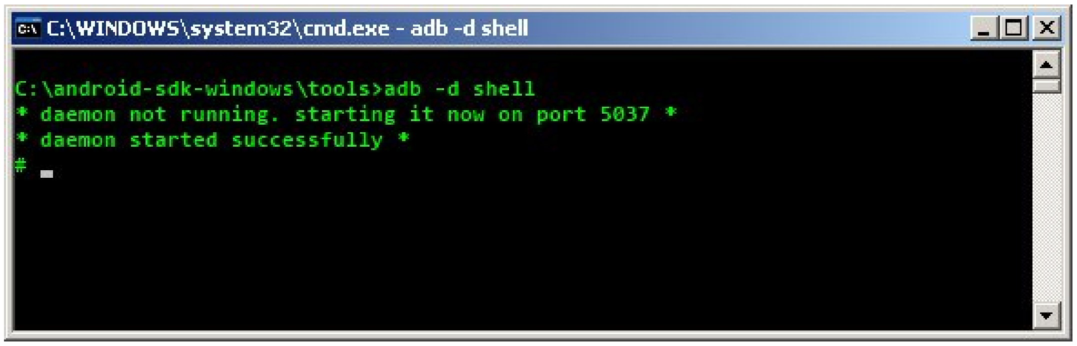


Testing ADB

- If you have a physical device, connect it to your PC with your USB cable.
- Open a command prompt and go to your android-sdk\tools folder
  A fast way to get to the prompt is *WindowsKey+R -> cmd -> Enter*.
  I prefer to create a batch file shortcut in any folder I access frequently which contains the command: *cmd*
- If you have your phone AND your emulator connected, you will have to tell ADB which one to use.
  For an actual phone use –d and for the emulator use –e.
  adb -d shell

  This will start the ADB daemon and connect you to your device's shell. If you see something like:

  ** daemon not running. starting it now on port 5037 **
  ** daemon started successfully **
  #



```
C:\WINDOWS\system32\cmd.exe - adb -d shell
C:\android-sdk-windows\tools>adb -d shell
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
#
```

… then you are good to go. Now you are on the Linux shell on your machine. It is not important but it is illuminating to poke around a bit and see how things are laid out. Type *exit* to quit the shell. Leave this window open for later. The ADB daemon will continue running in the background of your computer until you close the process, reboot or use the command *adb kill-server*.

Installing the App & Inspection

Now we know that ADB works we can install an .apk either from Market or from the PC. I'll assume you can work the Market. For this tutorial you already have the .apk on your PC so install it with this ADB command:

```
adb install \path\to\crackme0.apk
```

Once it is installed, run it and take a look around. In general you want to be looking for words like "trial", "expires", "license", "order", "validation", "serial" and so on. Don't rush this step, especially with more complex programs. It is sometimes easy to crack one part of the App so it shows you are running the full version but some features may still fail to work. Take notes of the exact strings you see on messages about needing to purchase or register. You will be searching the code for them later.

If it is a trial version then set the clock on your phone forward a few months so the trial will expire and see what it says. Then try uninstalling and reinstalling and see if it still knows it is expired. If it still expired then it is somehow communicating with a server and probably sending a unique identifier of your phone and time it was last used. Actually, this is more likely to happen if it expires naturally because the client will probably not be telling the server when a specific device should expire. This can be cracked the quick and dirty way perhaps by getting the trial to reset every time the App is launched or by completely bypassing trial checks.

If it requires a serial then try entering a random serial and make note of what the error message is. Look for strings like "Not Validated" or "Not Full Version" because when the App is building up that message it will be performing checks to see if it is registered or not and whatever functions it is calling can be cracked to always return that it is registered. Or it could be cracked so that any serial will count as a good serial.

License file protection sometimes has code to load the license, validate it and then create an object with the license information which is then accessed later. For example, maybe the App has a loading screen that shows "Licensed to Linus at ltorvalds@microsoft.com". The App loads the license and username and e-mail then validates the license. Then, when the welcome screen is created the information is displayed. For this the cracker could bypass the validation function completely and build up the object manually. It could also be that the App checks for a special "key" version. It may likely check the signature of the Key App with the main App to see if they are identical or it may check the size of the package with a known value. This can be easily circumvented by changing the main App to compare its signature with itself or find the jump that executes when it fails the check and circumvent it. Nearly every time you see code checking file sizes or especially signatures you are looking at protection code.

Web validation is more often employed than with PC programs because it is assumed that if you have a mobile you also have either wireless or a data plan. Many programs in the wild may employ web validation mixed with a license check or web validation of a serial. These are not by default more difficult to crack but the cracker must observe how the App handles different responses from the server and figure out what the proper response should be. More advanced debugging and watching web traffic are covered in a later tutorial. Once you know the proper response you can disable the “phoning home” and simply build up the proper server response manually so the rest of the App logic is unaffected.

The included crackme uses a serial check. Try entering a bogus serial and see what message it gives you. You can use that string as a starting point, though with crackme0 there are many obvious starting points.

No matter what protection techniques are used you must challenge yourself to crack the App in the cleanest way possible. This is the Way. There are often any number of different methods the cracker may employ but the difference between a script kiddie and a master is the master subverts protection at the source with quick, precise strikes. If you find you are changing many lines of code there is probably an easier way. Ask yourself: Could I do this any cleaner? Is the program going to run *exactly* as if it were legitimately purchased? Is it cracked such that it works for everyone? When was the last time I went outside? Days? Weeks? Do I even remember how to get there?

Getting the Apk

After you have run through the program for a time and have started thinking about your cracking strategy and have some notes on where to begin looking, we are ready to disassemble. For this tutorial we will use apktool. You can download it here: http://code.google.com/p/android-apktool/

For editing the Dalvik we will use UltraEdit with custom-made syntax highlighting. UltraEdit is a commercial program with more features than one could possibly ever learn which sometimes comes in handy. You can use any editor but I do not know of any

others that have syntax highlighting for Dalvik, which most helpful when staring at code for hours. If you want to give to the community, consider creating syntax highlighting configurations for your editor of choice. Feel free to take a look at the UltraEdit file included with this tutorial (smali.uew). If you get or have UltraEdit you will want to install the Wordfile to enable syntax highlighting. For info on how to do this, see here:
http://www.ultraedit.com/downloads/extras.html

This tutorial comes with crackme0.apk, so you can skip to the **Using Apktool** section, but if you did not have the .apk already you would pull it from your phone. Start ADB shell and we will look for the .apk file. It will likely be in one of the four places:

/data/app
/data/app-private
/sd-ext/app
/sd-ext/app-private

If your version of Android has Apps2Ext or Apps2SD, which allows you to install Apps to an external SD card ext partition, it will probably be in /sd-ext/app. Most apps are in app but Apps you pay for from market go into .../app-private and leave a .zip file behind in .../app. Non-rooted phones can not pull from app-private so if this is your situation you will need to either root your phone or download it with your computer somehow. Good information on how to root your phone is provided by CyanogenMod Wiki, which is a good quality custom Android ROM with many extra features and improvements. You can check out their Wiki here:
http://wiki.cyanogenmod.com/

- Once you know where the .apk file is, exit ADB shell and type:
  adb pull /sd-ext/app/name-of-app-package.apk
- For this tutorial try something like:
  adb pull /sd-ext/app/com.lohan.crackme0.apk
  The second path given tells ADB where to put the file it just pulled. If you do not give it this parameter it will just put it wherever adb.exe is running from.

Using Apktool

- Open a command prompt and go to the same directly where apktool and your .apk file are. To disassemble give it the command:
  apktool d com.lohan.crackme0.apk dump-crackme0
  And while you're at it, run apktool again with no parameters and familiarize yourself with the options.
- Apktool should have created a folder called dump-crackme0. This contains all of the source code and decoded resource files for window layouts, images, logos and so on.

Cracking

Starting Points

There are many different ways to begin. You could look at the names of all the .smali files and see if any of them look good. File names like LicenseCheck.smali or TrialController.smali are sometimes a good place to start. The strings you found during the inspection phase are either dynamically generated in the smali code or they exist in strings.xml. If you find them in strings.xml, look at the *name=* part of the line and copy the name. Example:
<string name="pro_version">Thanks for buying the Pro Version!</string>

For the above example copy "pro_version" and then search for that string. You should find it in a file called public.xml which contains the ID numbers for each string. Let's say you find it on the line:
<public type="string" name="pro_version" id="0x7a080001" />

Copy the string inside *id=* and then search the code for that number. The 0x in front is standard notation for a hexadecimal number. That’s not important here but it is helpful when looking at code because all numbers in the code are hex. Google around if you’re clueless as to what this means. When you find the code that uses that number it will probably be performing checks in that area on whether or not it is actually the pro version. Sometimes you may not find any code that references that number. If this is the case then it could be that you are dealing with a stripped down *"light"* version of an App that does not contain the code for the pro version. Or could just be a left-over string that is not used so don't give up immediately.

If you're unlucky enough to be dealing with boring, unobfuscated code you can probably start by looking for keywords in the code. Function names like *"IsLicensed"* or *"CheckDonateStatus"* are good places to start. Try some terms you picked up from your inspection.

Understanding

You will absolutely need a Dalvik opcode reference guide. Google provides this one here: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html. This is a list of all the Davik functions you will see and how they work. Programming, especially Java, and Assembly experience are helpful here. It's not possible to learn Dalvik by just reading; you will have to understand by looking at code. For this reason I've included example.smali with lots of comments to help you get started.

Sometimes working backward from your starting point helps. If the license check functions returns an integer, search in the code for where that function is called and see what return values it expects. Sometimes it may return different values for different levels of registration. Look for comparison operations. You will almost always see one right before a jump in the protection code. You can remove the jump or always make it jump or fix the values before so the comparison is always favorable.

When you make a change to a line leave a comment so you can keep track of your changes. It is very easy to forget where a change was made and it helps to have a little note why you did something later. Comments are stripped from the compiled version so you don't have to worry about someone else seeing them. Also, any lines you add should be kept together to separate them from normal code. This way your changes are very clear and easy to find. Here's an example:

```
const/v4 v0, 0x0

invoke-direct {v0}, Lcom/someapp/IsLegit(I);I

move-result v0

# cracknote: replace v0 with true and return that
const/4 v0, 0x1
return v0

…
```

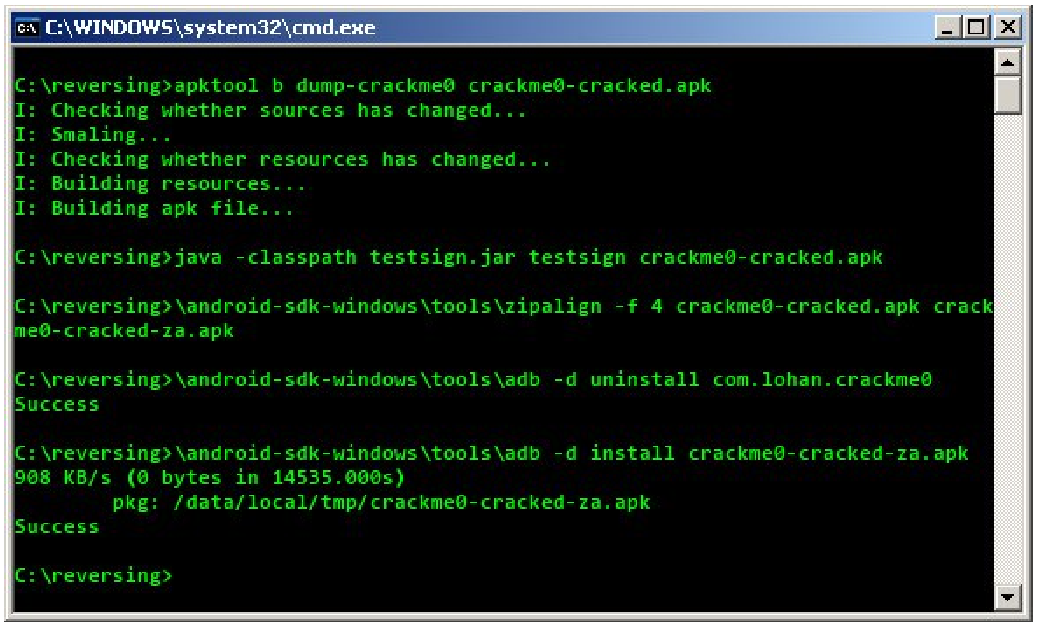
Compiling and Signing

After modifying the dump we simply recompile, resign, zipalign and install to the phone. Sometimes you will have many attempts at a crack before it works perfectly and these steps may be repeated many times so I suggest you create several batch scripts to help you save time. Zipaligning is an often ignored step in .apk optimization. To describe what it does, here is a quote from Android's own website:

> *"The resource-handling code in Android can efficiently access resources when they're aligned on 4-byte boundaries by memory-mapping them. But for resources that are not aligned (that is, when zipalign hasn't been run on an apk), it has to fall back to explicitly reading them — which is slower and consumes additional memory."*
> http://developer.android.com/resources/articles/zipalign.html

- To compile use *something like* this (your file names may be different):
  `apktool b dump-crackme0 crackme0-cracked.apk`
- Now use testsign.jar to sign with a random certificate:
  `java -classpath testsign.jar testsign crackme0-cracked.apk`
- To zipalign:
  `zipalign -f 4 crackme0-cracked.apk crackme0-cracked-za.apk`

- Use ADB to uninstall / install the .apk
  adb uninstall com.lohan.crackme0
  adb install crackme0-cracked-za.apk
  Uninstalling will automatically close the App if it is running so you don't need to worry about closing it before updating.



C:\WINDOWS\system32\cmd.exe

```
C:\reversing>apktool b dump-crackme0 crackme0-cracked.apk
I: Checking whether sources has changed...
I: Smaling...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...

C:\reversing>java -classpath testsign.jar testsign crackme0-cracked.apk

C:\reversing>\android-sdk-windows\tools\zipalign -f 4 crackme0-cracked.apk crack
me0-cracked-za.apk

C:\reversing>\android-sdk-windows\tools\adb -d uninstall com.lohan.crackme0
Success

C:\reversing>\android-sdk-windows\tools\adb -d install crackme0-cracked-za.apk
908 KB/s (0 bytes in 14535.000s)
        pkg: /data/local/tmp/crackme0-cracked-za.apk
Success

C:\reversing>
```

Now load up the new version on your phone and test it out. If it doesn't work the first time, don't be frustrated or intimidated. Just think of it as a puzzle. Hopefully you like puzzles and challenges or you truly have the wrong kind of hobby. After you crack a few Apps you will begin to see patterns and each new App will be easier and easier to crack until you are so bored you become excited when you finally meet a worthy adversary. If it gets too tough then take a break. Think about it during the day and sometimes a flash of inspiration will enter into you.

If you get stuck on the crackme0 check out the solution. It can be solved in one line. It makes it so any serial works for activation which is not bad. It would be better if the program required no validation at all. Since it is a simple App it would not be difficult to simulate this. See if you can figure out both ways on your own.

Final Notes

Be a little paranoid. It's all fun and games until someone decides to sue you. If you release your cracks do it under an alias that has nothing to do with your Facebook or Twitter or whatever. I shouldn't have to go into detail here just use your brain.

People *will* eventually steal your cracks and take credit for your stuff. If this bothers you then you are in it for the fame a little too much and that is not The Way; it is a weakness. Fame, recognition – a cracker seeks not these things. Do it because you like to know things you're not supposed to know. Do it because you enjoy being the first person to figure something out. Don't make enemies or engage in pointless flame wars. No one wins and you both look stupid. Just crack your Apps and set them free.

Comments, suggestions, corrections, criticisms and especially new or interesting Android cracking information are welcome if you can figure out how to reach me. Below is a hint.

Ykc5b1lXNHVjR3gxYzBCbmJXRnBiQzVqYjIwPQ==