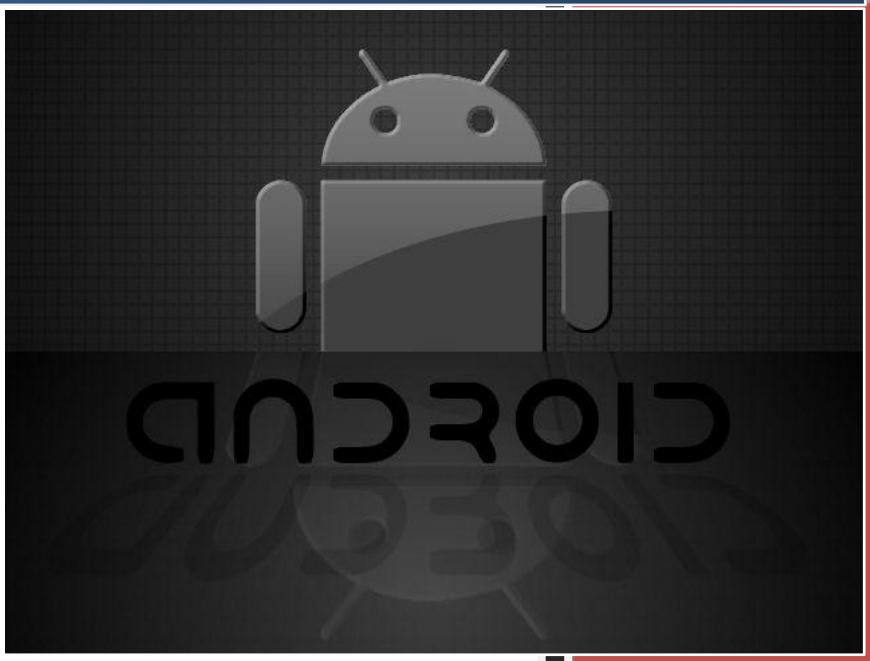
2011

Ingeniería inversa básica en Android Tomo II



Ghost

portalhacknet@gmail.com

17/07/2011

TABLA DE CONTENIDO

Dalvik y los OPCodes

Como muchos saben **Android está basado en Linux** y de forma nativa su el lenguaje oficial para desarrollar aplicaciones es Java. Esto porque se supone que implementa una plataforma que hace de máquina virtual, aunque no precisamente como la **máquina virtual de Java** (JVM) si no una modificada a la que llamaron **Dalvik Virtual Machine** creada por Dan Bornstein con la ayuda de algunos ingenieros de Google.

Dicho esto se puede definir que el código generado para las aplicaciones Android no es exactamente **"Java Bytecode"**, por eso los ejecutables para Android no contienen ficheros .class si no .dex que significa **Dalvik Executable.** De igual forma estos ficheros .dex se pueden "descompilar" y lo que veremos posteriormente será un lenguaje de bajo nivel (**Dalvik Bytecode**).

Aunque como tal con algunas aplicaciones específicas podemos **ver en un lenguaje de alto nivel** estos códigos (aplicaciones como Dex2Jar posteriormente descompilando con JD-GUI). Algunos enlaces interesantes sobre Dalvik y los OPCodes:

http://www.dalvikvm.com/
http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html
http://developer.android.com/reference/dalvik/bytecode/Opcodes.html

Herramientas a utilizar y preparación del entorno

Para la preparación del entorno tendremos en cuenta todo lo necesario para empezar casi "desde cero", los requisitos como siempre son saber la lógica de un lenguaje de programación (mejor si es **Java**) y bueno, lo demás se explicará poco a poco. Empecemos:

- Como trabajaremos con Android **necesitaremos usar la plataforma Java**, ojo, no basta con el entorno de ejecución (JRE) necesitaremos todas las herramientas para desarrolladores (JDK – Java Development Kit) y un IDE cualquiera, puede ser Eclipse. http://www.oracle.com/technetwork/java/javase/downloads/index.html

- También necesitaremos las **herramientas de desarrollo de Android** o SDK, en el portal oficial hay una guía de instalación http://developer.android.com/sdk/installing.html también debemos configurar un emulador o el **AVD**, que nos permitirá ejecutar una "máquina virtual" de Android en nuestros ordenadores (ver aquí http://developer.android.com/sdk/adding-components.html).

- **Smali/Baksmali:** se trata de un ensamblador/desemsamblador de ficheros apk que nos servirá para generar los fuentes en lenguaje de bajo nivel (aunque como tal no lo usaremos en esta guía). Se puede descargar de aquí: http://code.google.com/p/smali/

- **APKTool:** Esta herramienta sustituirá el Smali/Baksmali ya que los integra dentro de su paquete. Como su nombre lo indica es un kit de herramientas para trabajar con ficheros APK. http://code.google.com/p/android-apktool/ con el generaremos el código de bajo nivel y luego recompilaremos.

- **Testsign.jar**: Es una herramienta de terceros desarrollada para "firmar" o generar certificados aleatorios para indicarle al sistema que el fichero que hemos modificado se trata de una aplicación Android. Se descarga de aquí http://code.google.com/p/zen-droid/downloads/list (buscar el fichero testsign.jar).

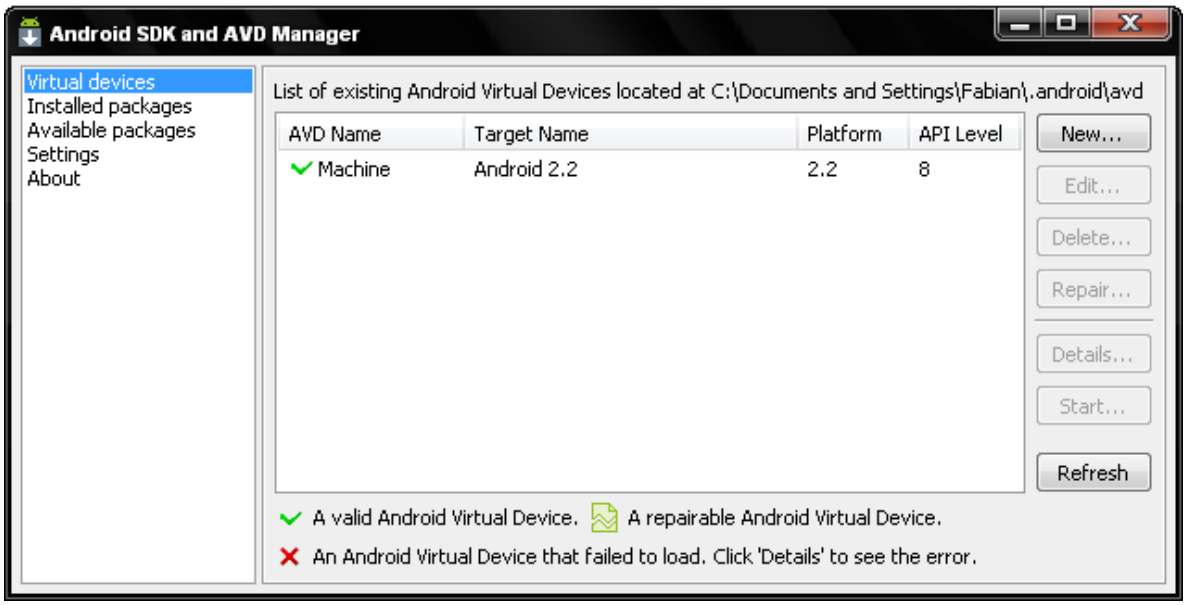
- **zipalign**: se trata de una herramienta que viene incluida en el SDK (**PATH\android-sdk-windows\tools\zipalign.exe**), que está hecha para preparar nuestro apk para el sistema Android, es decir, para que su rendimiento sea tal cual para dicho sistema ya que si no está "comprimida" con esta herramienta podría afectar el rendimiento de la misma en el dispositivo.

Instalar aplicaciones APK en el emulador de Android

El **AVD Manager Tool** es la herramienta que nos permite gestionar distintas "máquinas virtuales" de un sistema Android configuradas para un api específica (es decir, para distintas versiones del sistema operativo Android).

Supondré que ya tienen instalado el Android SDK que viene con el SDK Manager pero aún falta configurar dicho "emulador" o máquina virtual. Por lo que aquí explicaré como crear una. Lo que necesitaremos es ir al directorio en donde hemos instalado el Android SDK, allí veremos el ícono del logo o mascota de Android llamado SDK Manager, desde allí podremos instalar diversas apis y claro, configurar nuestra nueva máquina virtual.

Allí hacemos clic en la opción "**Virtual Devices**" y si es la primera vez que lo usamos de seguro no habrá ninguna máquina virtual configurada, por lo que **hacemos clic en el botón de la derecha que dice "New…"**.

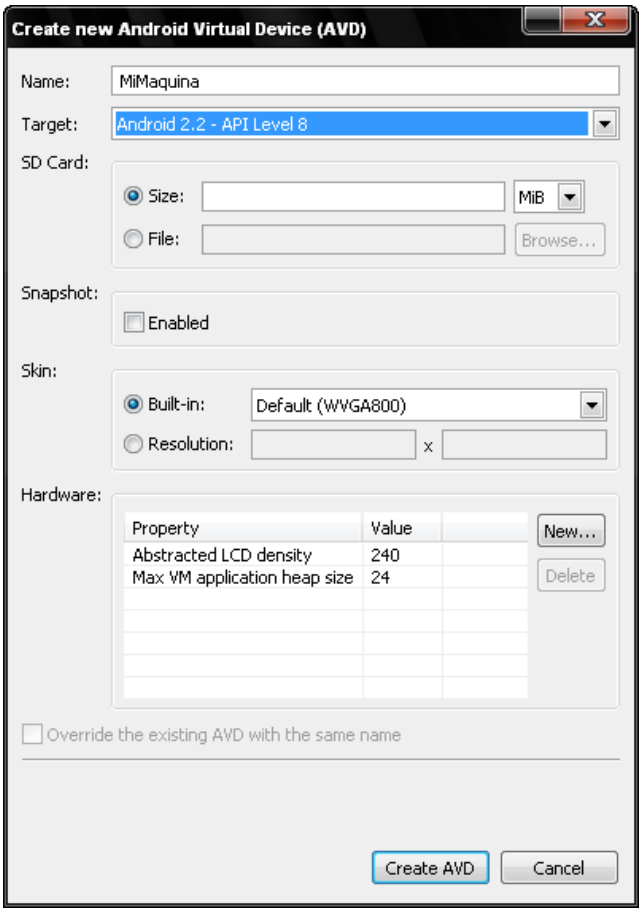


Luego tendremos que configurarla. Los parámetros son:

Name: El nombre de la máquina virtual. Esta no puede contener espacios.

Target: Es el api que utilizaremos para ejecutar la máquina virtual es decir, como si fuera la versión de Android. Recomiendo de momento la "API Level 8" o Android 2.2 que es lo mismo

Lo demás ya se configura una vez hemos seleccionado el api, pero si queremos parámetros adicionales podemos explorar la herramienta. Por último le damos a **Create AVD**.



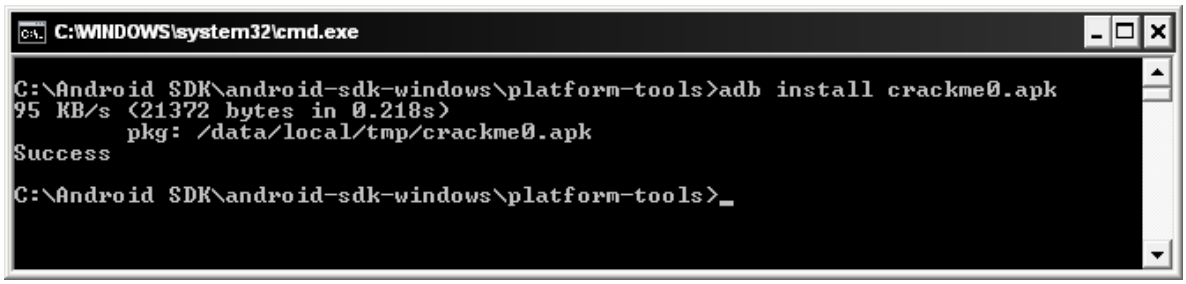
Listo ahora la seccionamos en el listado y en la parte derecha le damos "Start" y "Launch". Se empezará a cargar la nueva unidad virtual lo que puede tardar varios minutos así que debemos ser pacientes en este paso.

Cuando arranque veremos una pantalla como si fuese un móvil Android, con el sistema operativo completamente funcional. Pero como lo que vamos a hacer es un proceso de ingeniería inversa necesitaremos probar la aplicación, por lo que surge la necesidad de **instalar aplicaciones APK en esta nueva unidad virtual**.

Lo base para esto será saber que las aplicaciones Android se deben copiar a la carpeta "platform-tools" por ejemplo, la mía está en C:\Android SDK\android-sdk-windows\platform-tools, esto porque allí hay una herramienta llamada ADB (**Android Debug Bridge**) la cual es una herramienta que funciona por línea de comandos y que también sirve de conexión de nuestro móvil al ordenador y realizar tareas desde allí

En esta ocasión la utilizaremos para instalar APKs, por lo que una vez copiado el APK al directorio "Platform-tools" debemos ejecutar (con el emulador abierto):

adb install fichero.apk o para desinstalar **adb uninstall fichero**



Ahora si todo ha salido bien veremos la aplicación instalada en el emulador. Por cierto, algo importante a tener en cuenta es que **el emulador o unidad virtual debe estar ejecutándose al momento de realizar este proceso de instalación del APK**.

Instalar y usar ApkTool

Con el proceso anterior aprendimos a probar los APK. Bien, olvidemos por un rato eso y ahora iniciemos con el proceso de "ingeniería inversa" como tal. En esta ocasión usaremos **APKTool,** que básicamente sirve para cambiar algunas cosas de las aplicaciones Android pero además de ensamblar y desensamblarlas, que es nuestro uso principal en esta ocasión.

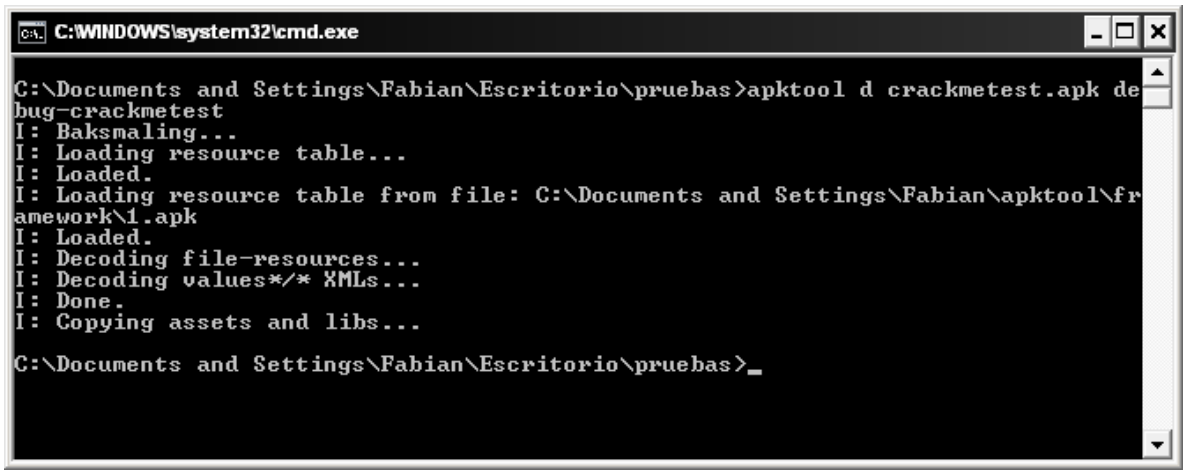
Para instalar APKTool deberemos bajar tanto el fichero apktool.jar (el principal) y si estamos en Windows debemos bajar las dependencias, veamos el proceso paso a paso mejor:

1. Descargar APKTool y dependencias (apktool1.4.1.tar.bz2 y apktool-install-windows-r04-brut1.tar.bz2).
2. Descomprimir todo en una sola carpeta.
3. Copiar estos ficheros al directorio principal de Windows, es decir, C:\Windows.

Listo ahora podemos ejecutar desde la consola de Windows el APKTool (Tecla Windows + R > cmd > Tecleamos **apktool** para ver todos sus parámetros). Para continuar con dicho proceso deberemos generar nuestro bytecode o fuentes en código de bajo nivel. Para "descompilar" nos ubicamos en la carpeta del APK y tecleamos en la consola:

apktool d fichero.apk nombre_nuevo_sin_extension

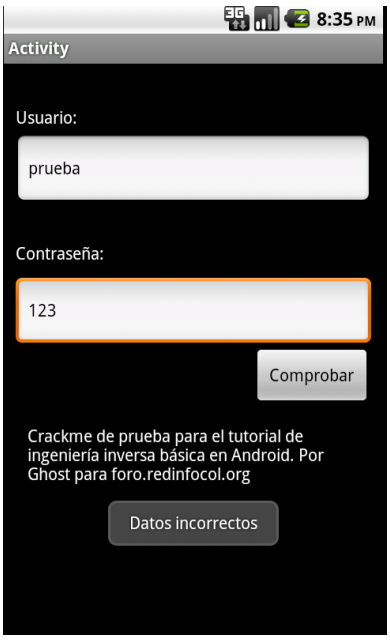
En la consola sería algo como esto:



Ya "desensamblamos" todo por lo que si revisamos la carpeta de nuestro APK hay un nuevo directorio el cual tiene el mismo nombre que le especificamos en la línea de comandos hace un momento. Lo que contiene son los ficheros fuente, el AndroidManifest.xml, entre otros recursos L**o que nos interesa es la carpeta smali**, donde están los fuentes.

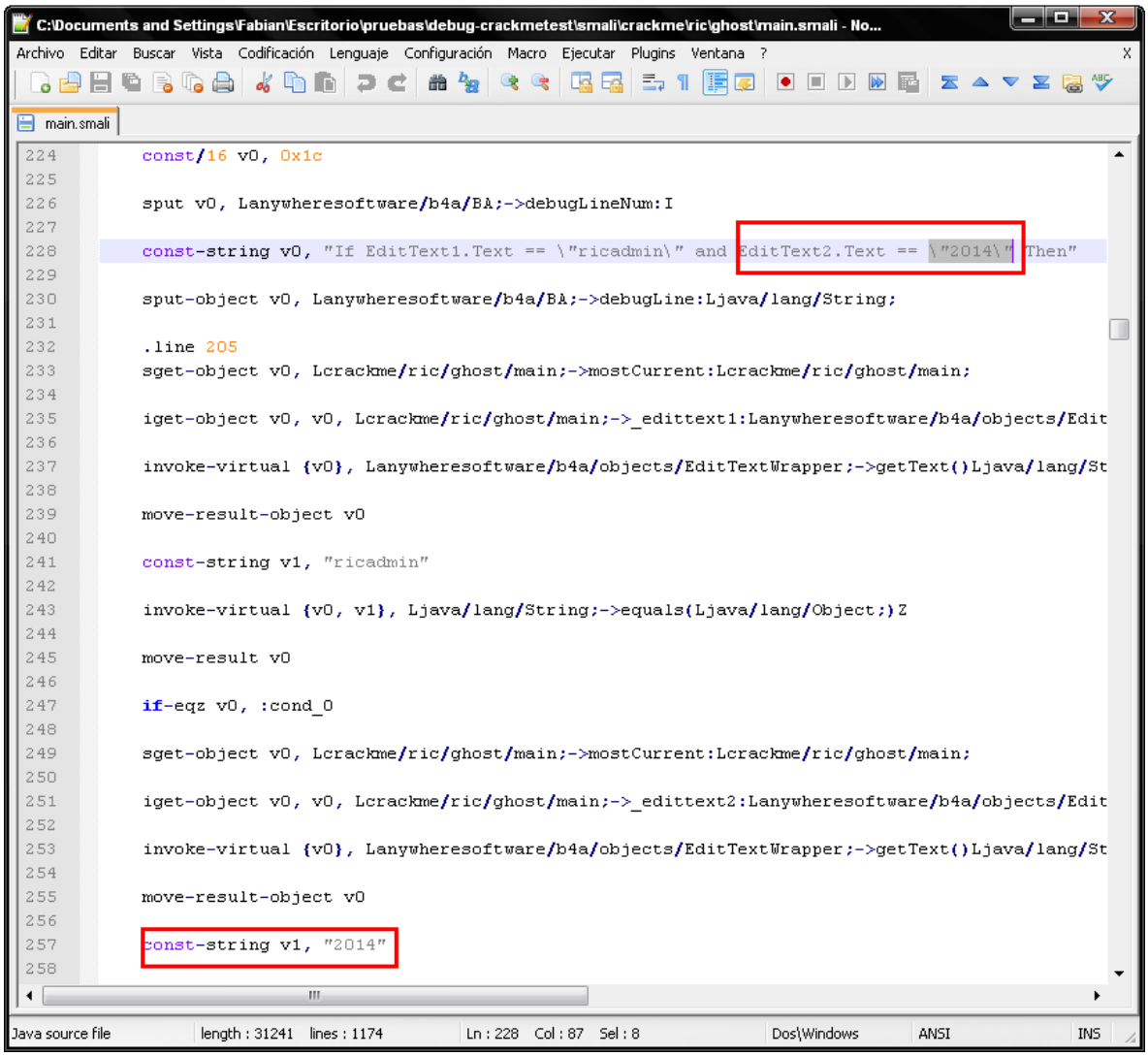
Si revisamos dicha carpeta y los ficheros generados, veremos unos archivos con extensión .smali, y lo primero que noto es un archivo llamado "**main.smali**", pues sí, el código en lenguaje de bajo nivel.

En el tomo nº 1 de esta guía básica compartí un "crackme" o una aplicación Android que pedia un usuario y contraseña, no era más que una comparación de strings. Supongamos ahora que estamos revisando dicha aplicación y nos ponemos a **buscar el mensaje de error que muestra cuando el usuario y contraseña son incorrectos**.



Usará ahora un editor de textos, de forma intuitiva uso el buscador para encontrar este mensaje y revisando un poco el código encuentro una comparación de strings, que no representan más que dicha comparación de usuario y contraseña. Pero estos string están declarados en un **OPCode** llamado const-string, por lo que ya sabemos la importancia de reconocerlos y que **para buscar cadenas está el OPCode const-string.**

Ahora, esto quiere decir que si cambio el **const-string** y la cadena encontrada en la comparación podré modificar los datos de acceso a la aplicación, si pongo por ejemplo "**123**" como contraseña podremos ver que pasa (sabiendo que la contraseña actual es "2014" y el nombre de usuario como se puede ver es "ricadmin"):

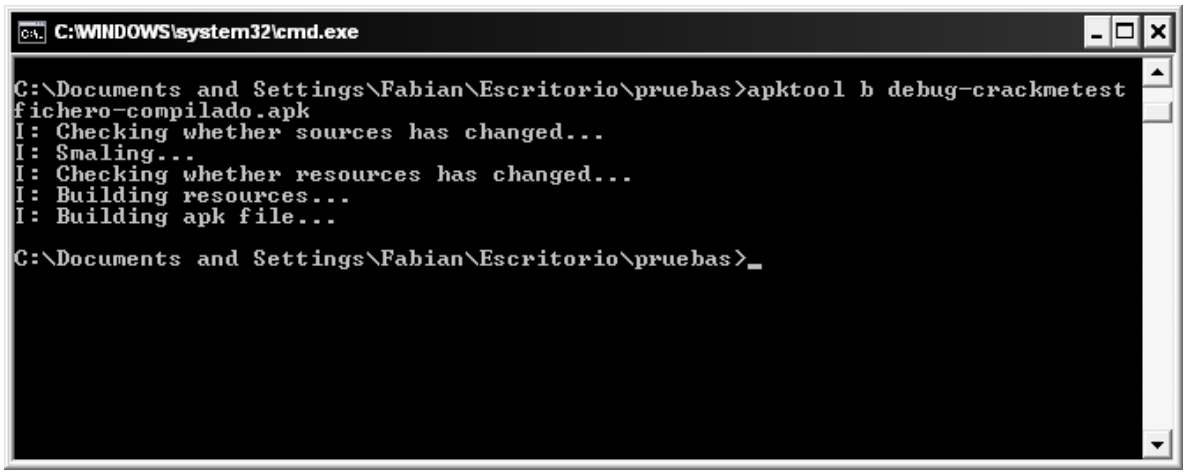


Una vez modificado el siguiente paso será empaquetar nuevamente nuestro fichero APK, por lo que abrimos una consola de comandos, nos ubicamos en la carpeta donde estábamos trabajando y tecleamos:

apktool b carpeta_de_trabajo ficherocompilado.apk

Nótese que anteriormente yo había "desensamblado" y le llamé a mi carpeta "debug-crackmetest". Tengan muy en cuenta los espacios que hay entre colocar el nombre de la carpeta y el del nuevo fichero compilado.

Y listo, con eso habremos generado un nuevo fichero APK con el nombre que le hemos acabado de poner (aparece en la misma carpeta de trabajo).



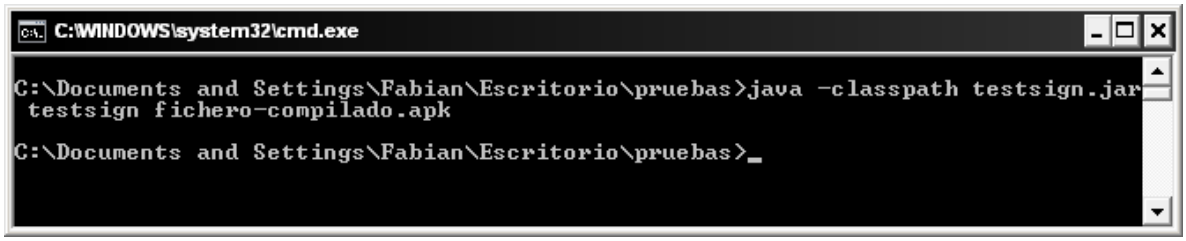
Firmando la aplicación o "generando certificados"

Es importante decirle a Android que nuestro APK en realidad si se trata de una aplicación para Android, pues estos tienen ciertas características que el sistema lee antes de su ejecución como tal.

Para esta tarea usaremos el "**testsign.jar**" una utilidad escrita para dicha finalidad y que podemos bajar en para empezar a usarla debemos colocarla en la misma carpeta de nuestro APK recién compilado y en la consola escribimos:

java -classpath testsign.jar testsign compilado-firmado.apk

Tener en cuenta que esto se le aplica a nuestro APK recién compilado. Si todo sale bien no saldrá nada por pantalla:

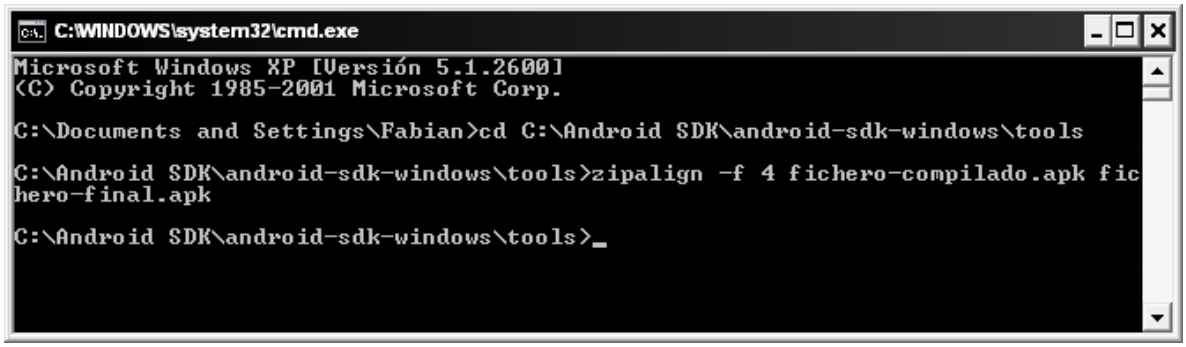


Generar un APK apto para Android

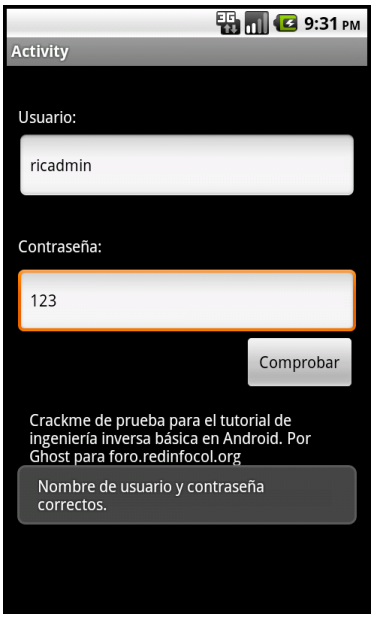
Listo, con el paso anterior "firmamos" la aplicación, ahora debemos optimizar el nuevo APK con una herramienta llamada "**zipalign**" que se encuentra en la carpeta "tools" de donde tenemos instalado el Android SDK.

El siguiente paso será copiar nuestro APK recién firmado a la carpeta "tools" que es donde se encuentra la aplicación "zipalign", abrimos una consola, nos ubicamos en el directorio donde está la aplicación y nuestro fichero recién copiado y tecleamos:

zipalign -f 4 fichero-compilado.apk fichero-final.pak



Sí todo sale bien tampoco saldrá nada. Ahora solo nos queda probar la aplicación y que los cambios se hayan realizado funcionen. Por lo que nuestro "fichero-final.apk" lo copiamos a "platform-tools" para instalarlo en nuestra unidad virtual que debe estar ejecutándose (por lo que en la consola escribimos "adb install fichero-final.apk"). Y el resultado es:



AGRADECIMIENTOS

Encontrar a gente a la que superar es importante para así ponernos metas que nos obliguen a avanzar en el camino del conocimiento, por eso le agradezco a todos estos degenerados porque algún día sabré tanto como ellos xDD:

UrbaN77, Radical (00000000), Monje (Clérigo), Casidiablo, [D-M-K], Hecky, SmartGenius.