# Dekompilierung von Android-Apps

Nachfolgend soll die Dekompilierung von Android Apps beschrieben werden. Dafür werden verschiedene Hilfsprogramme beschrieben. Es wird kein Anspruch auf Vollständigkeit erhoben.

## APK-Multi-Tool

Webseite: github.com

### Abhängigkeiten

Damit APK-Multi-Tool funktioniert, müssen das Android-SDK und die Platform-Tools installiert sein. Dies wird standardmäßig im Android Studio mit installiert, kann aber auch separat heruntergeladen und entpackt werden. Weiterhin werden mehrere Programme als Abhängigkeit benötigt und vorher installiert.

#### Android SDK

1. Download: android.com („`sdk-tools-linux-<Nummer>.zip`“)
2. Entpacken nach „`~/Android`“ (Verzeichnis muss eventuell erstellt werden)

#### Platform-Tools

1. Download android.com („`platform-tools-latest-linux.zip`“)
2. Entpacken nach „`~/Android`“ (Verzeichnis muss eventuell erstellt werden)

#### DEBIAN-Pakete

- Installation von: `optipng` `sudo` `aapt` `p7zip-full` `openjdk-8-jdk`

## Installation

Aus der originalen Datei „README“ im Archiv:

```
Installing APK Multi-Tool Itself
Instructions (Linux):
- Create a folder in your sdk called "APK-Multi-Tool" and put the contents
of the extracted APK-Multi-Tool into it or Rename the extracted APK-Multi-
Tool folder to just APK-Multi-Tool and put it into the sdk folder.
- Go to the the "sdk/APK-Multi-Tool" folder and rename "Script.sh" to
"script.sh".
- Go into the "other" folder, right click on one executable file at a time,
select properties, go to "permissions" in the new window and check the
"allow file to be run as a program" box (do this with all the
executables/.exe's).
- Open a terminal in the APK-Multi-Tool folder (or type in terminal: cd
"PATH TO THE script.sh"). Type in:
  chmod 755 script.sh
- chmod 755 all files inside other folder. (thanks for the tip bkmo  )
- Install "sox": Open the software center of the linux service and searched
for sox. Once installed you will have SOX working
- To add the path to your folder open up a terminal and type in:
  sudo su
  PATH=$PATH:/THE PATH TO YOUR "SCRIPT.SH"
  (for me this looks like the following)
  PATH=$PATH:/home/username/sdk/APK-Multi-Tool
- Export PATH:Open up a terminal in the APK-Multi-Tool folder (or type in
terminal: cd "PATH TO THE script.sh") and type in:
  export PATH={PATH}:/PATH TO Your SDK/sdk/platform-tools/adb
  (for me this looks like the following)
  export PATH={PATH}:/home/username/sdk/platform-tools
  ("username" is the user name that appears on your computer).
- Now open a terminal in the APK-Multi-Tool folder (or type in terminal: cd
"PATH TO THE script.sh") and type in:
  ./script.sh
- You should now have a running APK-Multi-Tool.
```

Meine Interpretation daraus:

1. Erstellen eines Unterverzeichnisses „`APK-Multi-Tool`“ im Verzeichnis „`~/Android/`“ und den Inhalt des heruntergeladenen Archivs dorthin kopieren
    - damit das funktioniert, muss „`Android SDK`“ heruntergeladen und entpackt werden (siehe Android-Abhängigkeiten)
2. Umbenennen der Datei „`~/Android/APK-Multi-Tool/Script.sh`“ in „`~/Android/APK-Multi-Tool/script.sh`“
    - im vorliegenden Fall heißt die Datei bereits „`script.sh`“
3. im Unterverzeichnis „`other`“ die Eigenschaften aller ausführbaren Dateien anpassen, sodass die Ausführung im Terminal durchgeführt wird
    - im vorliegenden Fall nicht möglich, da ohne grafische Oberfläche gearbeitet wird

4. Setzen der Dateiberechtigung 755 auf die Datei „`script.sh`“ („`chmod`“)
    - im vorliegenden Fall besitzt die Datei bereits die richtigen Rechte
5. Setzen der Dateiberechtigung 755 auf alle Dateien im Unterverzeichnis „`other`“ („`chmod`“)
    - obwohl JAR- und Zertifikatsdateien keine Ausführberechtigung benötigen, wurde das Skript „`perm`“ ausgeführt, welches die Rechte setzt
6. Installation von „`sox`“
7. Pfadumgebungsvariable um das Verzeichnis „`~/Android/APK-Multi-Tool/`“ erweitern
8. Pfadumgebungsvariable um das Verzeichnis „`~/Android/platform-tools/`“ erweitern
    - damit das funktioniert, müssen die „`Platform-Tools`“ heruntergeladen und entpackt werden (siehe Android-Abhängigkeiten)
9. Test des Skriptes „`script.sh`“
    - das Skript testet verschiedene Programme auf Existenz und gibt im Fehlerfall eine Meldung aus

Wird kein fehlendes Programm gefunden oder ein Fehler entdeckt, präsentiert sich ein Auswahlmenü:

```
################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ----------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

- Advanced Tasks (XML, Smali, etc.) -----------------------------------------
---
  9    Decompile APK                    10   Compile APK
  11   One-step Compile-Sign-Install

- Common Tasks --------------------------------------------------------------
---
  0    ADB Pull                         8    ADB Push
  4    Sign APK                         6    Install APK
  5    Zipalign APK

- Batch Operations ----------------------------------------------------------
---
  12   Batch Optimize APK               13   Batch Sign APK
  14   Batch Optimize OGG files

- Distribution & Update.zip Creation ----------------------------------------
---
  41   Create Update.zip                42   Push Update.zip to device

- Decompile Classes.dex & View Decompiled Code ------------------------------
---
  51   Decompile Classes.dex            52   View Decompiled Code
```

```
- ADB Device & APK Management ------------------------------------------
---
  30   Backup Device Installed APKs      31   Batch Rename APK
  32   Batch Install APK (apk-backup)

------------------------------------------------------------------------
---
  20   Set Active APK
  21   Import framework-res.apk  (Perform apktool.jar if framework-res.apk)
  22   Clear Project Files
  23   Set Compression Level     (Current compression level: 3)
  24   Create all missing directories
  25   Show APK package information
  99   Fix Tools permissions
  00   Quit
------------------------------------------------------------------------
---
  Active APK File: NONE
------------------------------------------------------------------------
---

Enter selection:
```

## Arbeitsumgebung

Es musste eine etwas komplexere Verzeichnisstruktur im Verzeichnis „`tmp/`“ erstellt, damit das dekompilieren funktionierte. Zusätzlich wurde das Verzeichnis „`~/Android/APK-Multi-Tool/other/`“ in das Verzeichnis „`/tmp/apk-multi-tool/decompile/`“ kopiert, weil im Skript an mehreren Stellen ein Wechsel darin erfolgt.

```
~$ mkdir /tmp/apk-multi-tool
~$ cd /tmp/apk-multi-tool
~$ mkdir -p decompile/place-apk-here-for-modding
~$ ln -s decompile/place-apk-here-for-modding
~$ cp -r ~/Android/APK-Multi-Tool/other decompile/
~$ cd decompile
```

Wenn das Skript das erste Mal gestartet wird, kann die Nummer „24“ eingegeben werden, damit die fehlenden Verzeichnisse zu erstellt werden.

## Dekompilierung

Für die Dekompilierung einer APK-Datei, diese einfach in das Verzeichnis „`/tmp/apk-multi-`

tool/decompile/place-apk-here-for-modding/“ ablegen und dann das Skript „script.sh“ starten (aufgrund der obigen Vorbereitung liegt das Skript im Suchpfad):

Als erstes die Nummer „9“ eingeben, um eine APK-Datei zu dekompilieren:

```
################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

- Advanced Tasks (XML, Smali, etc.) -------------------------------------
---
  9    Decompile APK                    10   Compile APK
  11   One-step Compile-Sign-Install

...

------------------------------------------------------------------------------
---
  Active APK File: NONE
------------------------------------------------------------------------------
---

Enter selection: 9
```

Das Skript stellt fest, dass noch keine APK-Datei gesetzt wurde, dies dann mit „Y“ umsetzen. Eine Liste mit vorhandenen APK-Datien wird angezeigt (in vorliegenden Fall ist es nur eine Datei), dort dann einfach die entsprechende Nummer angeben. Hier im Beispiel wird die APK-Datei vom „Total Commander“ entpackt:

```
No active APK file set.
Set active APK file? (Y/n): Y

Listing APK files:
------------------
1) tcandroid280.apk

Choose APK: 1
```

Die APK-Datei wird dekompiliert:

```
Selected: tcandroid280.apk
```

```
I: Using Apktool 2.2.1-64644a-SNAPSHOT on tcandroid280.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
/root/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

...

--------------------------------------------------------------------------
---
  Active APK File: tcandroid280.apk
--------------------------------------------------------------------------
---

Enter selection:
```

Im Verzeichnis „`/tmp/apk-multi-tool/decompile/`“ gibt es ein neues Unterverzeichnis „`projects/`“ mit einem weiteren Unterverzeichnis „`tcandroid280.apk/`“ (also so, wie die APK-Datei hieß), worin sich alle dekompilierten Dateien befinden.

## Kompilierung

Für die Kompilierung wird das Skript „`script.sh`“ aus dem Verzeichnis „`/tmp/apk-multi-tool/decompile/`“ gestartet.

Hier sollte im ersten Schritt die APK-Datei gesetzt werden, damit später eine (sonst doppelt erscheinende) Abfrage übersprungen wird:

```
################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

...

--------------------------------------------------------------------------
---
  20   Set Active APK

...

  00   Quit
--------------------------------------------------------------------------
---
  Active APK File: NONE
--------------------------------------------------------------------------
---

Enter selection: 20
```

Es wird wieder die APK-Datei vom „`Total Commander`“ ausgewählt:

```
Listing APK files:
------------------
1) tcandroid280.apk

Choose APK: 1
```

Jetzt kann die Nummer „`10`“ eingeben werden:

```
################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

- Advanced Tasks (XML, Smali, etc.) ------------------------------------
---
```

```
  9    Decompile APK                      10   Compile APK
  11   One-step Compile-Sign-Install

...

--------------------------------------------------------------------------
---
  Active APK File: tcandroid280.apk
--------------------------------------------------------------------------
---

Enter selection: 10
```

Es erfolgt die Abfrage, ob eine System-APK oder eine regulären (normalen) APK-Datei erstellt werden soll. Hier wird eine reguläre APK-Datei erstellt:

```
Enter APK type:
---------------
1) System APK
2) Regular APK

Enter selection: 2
```

Es erfolgt eine Überprüfung und eventuell auch die Meldung von Fehlern (bei JAVA sind das sogenannte „Exception“):

```
I: Using Apktool 2.2.1-64644a-SNAPSHOT
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
Exception in thread "main" brut.androlib.AndrolibException:
brut.androlib.AndrolibException: brut.common.BrutException: could not exec:
[/tmp/brut_util_Jar_8289407321027721594.tmp, p, --forced-package-id, 127, --
min-sdk-version, 3, --target-sdk-version, 23, --version-code, 196, --
version-name, 2.80, -F, /tmp/APKTOOL7002678223645764966.tmp, -0, arsc, -0,
arsc, -I, /root/.local/share/apktool/framework/1.apk, -S, /tmp/apk-multi-
tool/decompile/other/../projects/tcandroid280.apk/res, -M, /tmp/apk-multi-
tool/decompile/other/../projects/tcandroid280.apk/AndroidManifest.xml]

...

Copy unmodified files to compiled APK to reduce errors (Y/n)? Y

\\

Jetzt kann das Projekt nochmals bearbeitet werden (in einem anderen
Fenster). Soll die Kompilierung fortgeführt werden, einfach "''[ENTER]'"
```

```
drücken:

<code>7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov :
2016-05-21
p7zip Version 16.02 (locale=C,Utf16=off,HugeFiles=on,64 bits,8 CPUs Intel(R)
Core(TM) i7-6700 CPU @ 3.40GHz (506E3),ASM,AES-NI)

Scanning the drive for archives:
1 file, 2018373 bytes (1972 KiB)

Extracting archive: ../place-apk-here-for-modding/tcandroid280.apk
--
Path = ../place-apk-here-for-modding/tcandroid280.apk
Type = zip
Physical Size = 2018373

Everything is Ok

Files: 717
Size:       3851342
Compressed: 2018373

Delete all modified files in the /keep directory.
If you modified an XML file, delete the resources.arsc file.
Press Enter key to continue.
```

Es wird eine unsignierte APK-Datei erstellt:

```
7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C,Utf16=off,HugeFiles=on,64 bits,8 CPUs Intel(R)
Core(TM) i7-6700 CPU @ 3.40GHz (506E3),ASM,AES-NI)

Scanning the drive:
65 folders, 717 files, 3851342 bytes (3762 KiB)

Creating archive: ../place-apk-here-for-modding/tcandroid280-unsigned.apk
Items to compress: 782
Files read from disk: 717
Archive size: 1848665 bytes (1806 KiB)
Everything is Ok
Done
```

## Signierung

Wenn nicht bereits geschehen, sollte für die Signierung im ersten Schritt die oben genannte APK-Datei ausgewählt werden:

```
################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

...

--------------------------------------------------------------------------
---
  20   Set Active APK

...

  00   Quit
--------------------------------------------------------------------------
---
  Active APK File: NONE
--------------------------------------------------------------------------
---

Enter selection: 20
```

Ganz wichtig ist es hier, **NICHT** die unsignierte APK-Datei auswählen:

```
Listing APK files:
------------------
1) tcandroid280-unsigned.apk
2) ./tcandroid280.apk

Choose APK: 2
```

Die Signierung selbst wird dann über die Nummer „4“ durchgeführt:

```
################################ Apk Multi-Tools
################################

- Simple Tasks (Image editing, etc.) ------------------------------------
---
  1    Extract APK                      2    Optimize APK Images
  3    Zip APK
  7    One-step Zip-Sign-Install

- Advanced Tasks (XML, Smali, etc.) ------------------------------------
```

```
---
  9    Decompile APK                      10   Compile APK
  11   One-step Compile-Sign-Install

- Common Tasks -----------------------------------------------------------
---
  0    ADB Pull                           8    ADB Push
  4    Sign APK                           6    Install APK
  5    Zipalign APK

...

-------------------------------------------------------------------------
---
  Active APK File: ./tcandroid280.apk
-------------------------------------------------------------------------
---

Enter selection: 4
```

Es erfolgt keine Ausgabe. Die signierte APK-Datei liegt dann im Verzeichnis „`/tmp/apk-multi-tool-decompile/place-apk-here-for-modding/`“: „`tcandroid280-signed.apk`“.

# Dex2Jar

Webseite: sourceforge.net (aktuelle Version am im Februar 2018: „`dex2jar-2.0.zip`“)

## Installation

Die Datei ist ein ZIP-Archiv und wird in ein beliebiges Verzeichnis entpackt. Ähnlich dem anderen Programm („`APK Multi Tool`“) wird ein neues Verzeichnis „`~/Android/Dex2Jar/`“ erstellt und alle Dateien darin entpackt.

Die entpackten Shell-Skripte (zu erkennen an der Endung „`.sh`“ müssen unter Linux noch ausführbar gemacht werden:

```
~$ chmod 755 ~/Android/Dex2Jar/*.sh
```

Damit das Ausführen der Skripte an beliebiger Stelle funktioniert, kann das Verzeichnis ebenfalls in

die Pfadumgebungsvaraible aufgenommen werden (im vorliegenden Fall die Datei „`~/.bashrc`“):

```
export PATH="$HOME/Android/Dex2Jar:$PATH"
```

*Nicht vergessen, die Datei danach neu einzulesen, damit sich die Änderung auswirkt.*

## Arbeitsumgebung

Im Verzeichnis „`/tmp/`“ wird eine Arbeitsumgebung erstellt, in der das Dekompilieren stattfindet:

```
~$ mkdir -p /tmp/dex2jar/apk
~$ cd /tmp/dex2jar
```

## Dekompilierung

Die gewünschte APK-Datei wird in das Verzeichnis „`/tmp/dex2jar/apk/`“ kopiert. Im vorliegenden Fall ist das wieder die App „`Total Commander`“:

```
~$ cp tcandroid280.apk apk/
~$ d2j-dex2jar.sh apk/tcandroid280.apk
dex2jar apk/tcandroid280.apk -> ./tcandroid280-dex2jar.jar
```

Die neu erstellte JAR-Datei „`tcandroid280-dex2jar.jar` kann nun mit einem geeignetem Programm (JAVA Decompiler) angesehen werden.

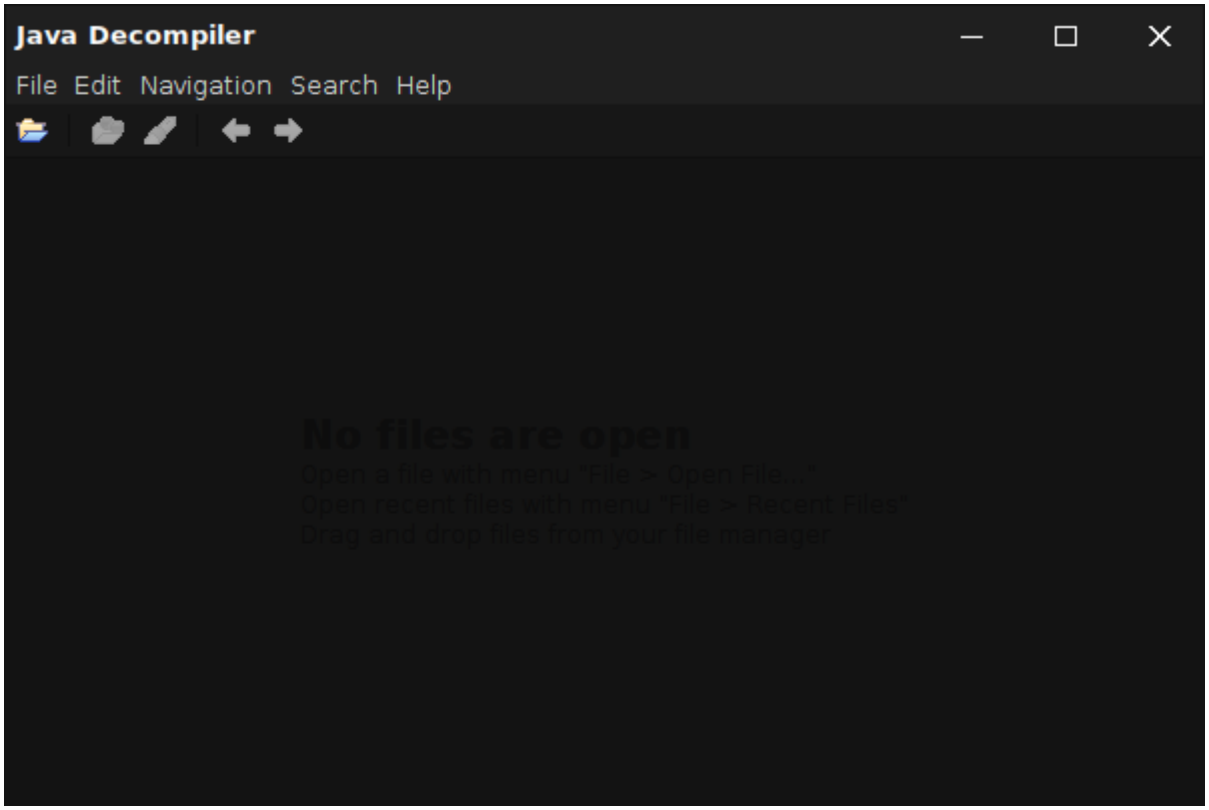
## JAVA Decompiler

Im Internet sind mehrere grafische Programme zum Anzeigen von JAR-Dateien zu finden.

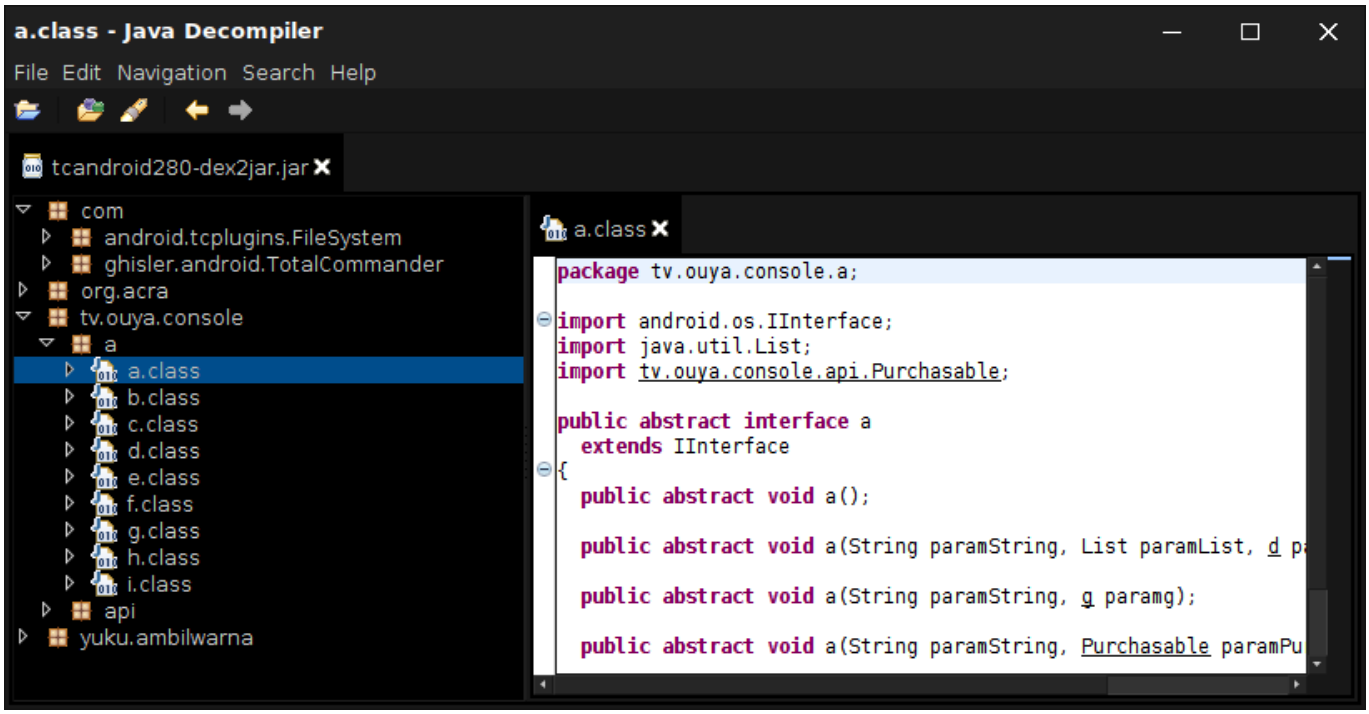
### JD-GUI

Download: benow.ca (aktuelle Version am im Februar 2018: “`jd-gui_1.4.0-0_all.deb`„)

Programmaufruf: `java -jar /opt/jd-gui/jd-gui.jar`




Unter „File“ → „Open File“ wird die JAR-Datei ausgewählt (hier „tcandroid280-dex2jar.jar“) und mit „OK“ geöffnet.

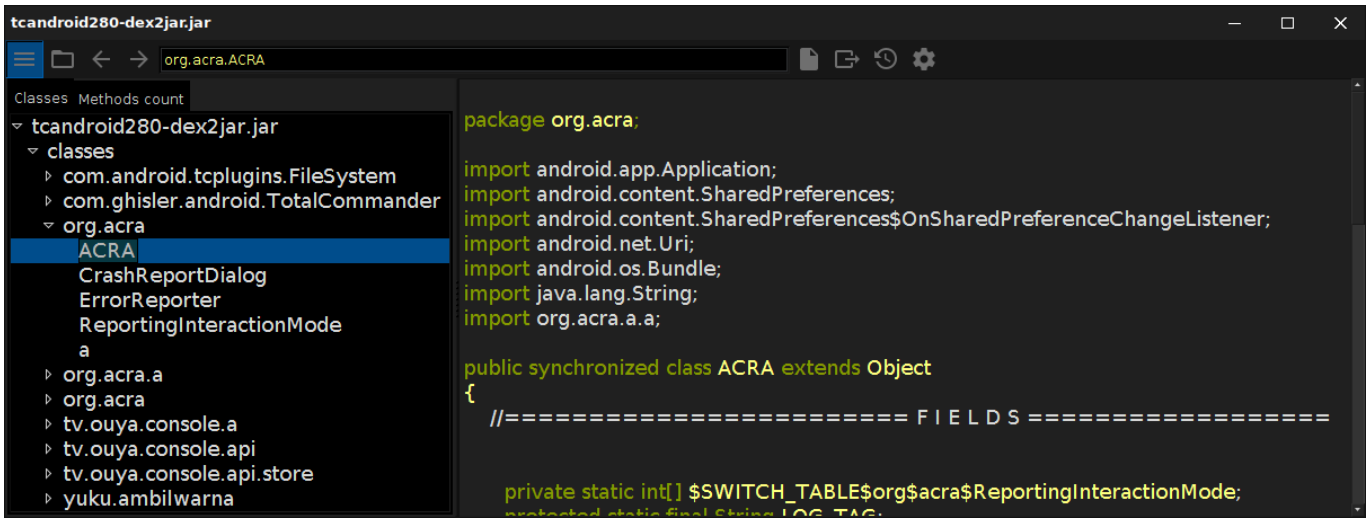



### ClassyShark

Download: github.com (aktuelle Version am im Februar 2018: „8.1“)

Programmaufruf: `java -jar ClassyShark.jar`

Auf das Zweite Symbol von links (der Ordner) klicken und im Dateidialog die APK-Datei suchen (hier "`tcandroid280-dex2jar.jar`„) und mit "OK„ öffnen.




### Weitere Programme

Die nachfolgenden Programme wurden im Internet gefunden, aber nicht näher betrachtet. Es sind Programme aufgelistet, wo eine Linux-Version existiert (oder zumindest ein JAVA-Programm):

- JAD Java Decompiler
- d4j
- Bytecode-Viewer
- Helios

## APKTool

Webseite: github.io

### Installation

Für Linux gibt es auf der oben genannten Webseite folgende Anleitung:

- Download Linux wrapper script (Right click, Save Link As apktool)
- Download apktool-2 (find newest here)
- Rename downloaded jar to apktool.jar
- Move both files (apktool.jar & apktool) to /usr/local/bin (root needed)
- Make sure both files are executable (chmod +x)
- Try running apktool via cli

Meine Interpretation dazu:

1. Herunterladen des Wrapper-Skriptes als Datei "`apktool`„ (mit rechter Maustaste auf Link klicken und "`Speichern unter`„ auswählen)
2. Herunterladen der aktuellsten JAR-Datei "`apktool_<Versionsnummer>.jar`„ von der Seite bitbucket.org
3. Umbenennen der soeben heruntergeladenen Datei zu "`apktool.jar`„
4. Verschieben der Datei in das lokale Binärverzeichnis "`/usr/local/bin/`„
    - dieser Schrott wird nachfolgend etwas anders gemacht
5. Dateirechte für beide Dateien auf ausführbar setzen
6. Skript auf der Konsole ausführen

Die beiden Dateien "`apktool`„ und "`apktool.jar`„ werden in das neu erstellte Verzeichnis "`~/Android/apktool`„ kopiert.

Damit das Ausführen des Skriptes an beliebiger Stelle funktioniert, kann das Verzeichnis ebenfalls in die Pfadumgebungsvaraible aufgenommen werden (im vorliegenden Fall die Datei „~/.bashrc"):

```
export PATH="$HOME/Android/apktool:$PATH"
```

*Nicht vergessen, die Datei danach neu einzulesen, damit sich die Änderung auswirkt.*

## Arbeitsumgebung

Im Verzeichnis "`/tmp/`„ wird eine Arbeitsumgebung erstellt, in der das Dekompilieren und Neukompilieren stattfindet:

```
~$ mkdir -p /tmp/apktool/apk
~$ cd /tmp/apktool
```

## Dekompilierung

Die gewünschte APK-Datei wird in das Verzeichnis "`/tmp/apktool/apk/`„ kopiert. Im vorliegenden

Fall ist das wieder die App “Total Commander„:

```
~$ cp tcandroid280.apk apk/
~$ apktool d apk/tcandroid280.apk
I: Using Apktool 2.3.1 on tcandroid280.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
/root/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Im aktuellen Verzeichnis gibt es jetzt ein neues Unterverzeichnis “tcandroid280„, welches die dekompilierten Daten enthält.

## Kompilierung

Das Neuerstellen der APK-Datei funktioniert so:

```
~$ apktool b tcandroid280
I: Using Apktool 2.3.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
```

Die neu erstellte APK-Datei befindet sich dann im Verzeichnis “tcandroid280/dist/„: “tcandroid280.apk„.

Alternativ kann beim Kompilieren auch eine neue APK-Datei angegeben werden:

```
~$ apktool b tcandroid280 -o tcandroid280.apk
I: Using Apktool 2.3.1
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
```

```
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
```

## Signierung

Die erstellten APK-Dateien müssen im letzten Schritt noch signiert werden, damit sie auf einem Android-Gerät installiert werden können.

Dafür wird im ersten Schritt mit dem Programm “`keytool`„ (gehört zum installierten JDK) ein privater Schlüssel erstellt:

```
~$ keytool -genkey -v -keystore mein-apk-key.jks -keyalg RSA -keysize 2048 -
validity 10000 -alias apk-schluessel
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Test Nutzer
What is the name of your organizational unit?
  [Unknown]:  Test
What is the name of your organization?
  [Unknown]:  Organisation
What is the name of your City or Locality?
  [Unknown]:  Niedersachsen
What is the name of your State or Province?
  [Unknown]:  Deutschland
What is the two-letter country code for this unit?
  [Unknown]:  de
Is CN=Test Nutzer, OU=Test, O=Organisation, L=Niedersachsen, ST=Deutschland,
C=de correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate
(SHA256withRSA) with a validity of 10,000 days
    for: CN=Test Nutzer, OU=Test, O=Organisation, L=Niedersachsen,
ST=Deutschland, C=de
Enter key password for <apk-schluessel>
    (RETURN if same as keystore password):
[Storing mein-apk-key.jks]

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to
PKCS12 which is an industry standard \
format using "keytool -importkeystore -srckeystore mein-apk-key.jks -
destkeystore mein-apk-key.jks -deststoretype pkcs12".
```

Die Werte müssen dann entsprechend den eigenen Vorstellungen angepasst werden.

Im aktuellen Verzeichnis liegt dann eine neue Schlüsseldatei “`mein-apk-key.jks`„.

Im zweiten Schritt wird mit dem Programm “`zipalign`„ die APK-Datei angepasst. Es kann bei Bedarf ganz einfach aus den Repositories nachinstalliert werden:

```
~# apt install zipalign
```

Jetzt kann die unsignierte APK-Datei angepasst werden:

```
~$ zipalign -v -p 4 tcandroid280.apk tcandroid280-unsigned-aligned.apk
Verifying alignment of tcandroid280-unsigned-aligned.apk (4)...
      49 AndroidManifest.xml (OK - compressed)
    6072 assets/favicon.ico (OK - compressed)
    9313 assets/help.htm (OK - compressed)
   21229 assets/help_cs.htm (OK - compressed)

...

 1619131 res/menu/playermenu21b.xml (OK - compressed)
 1619511 res/menu/playermenu_ouya.xml (OK - compressed)
 1619901 res/xml/mainpreferences.xml (OK - compressed)
 1621752 resources.arsc (OK)
Verification successful
```

Im letzten Schritt erfolgt die Signierung mit dem Programm “`apksigner`„. Das Programm gehört zum Android SDK und befindet sich in den “`build-tools`„. Es muss, wenn nicht bereits schon geschehen, mit dem SDK-Manager erst installiert werden. Das kann grafisch über Android Studio und wie hier nachfolgend beschrieben manuell auf der Konsole gemacht werden.

Auflisten aller vorhandenen Versionen:

```
~$ ~/Android/tools/bin/sdkmanager --list | grep build-tools
Warning: File /root/.android/repositories.cfg could not be loaded.
  build-tools;19.1.0                | 19.1.0       | Android SDK Build-Tools
19.1

...

  build-tools;26.0.3                | 26.0.3       | Android SDK Build-Tools
```

```
26.0.3
  build-tools;27.0.0                | 27.0.0       | Android SDK Build-Tools
27
  build-tools;27.0.1                | 27.0.1       | Android SDK Build-Tools
27.0.1
  build-tools;27.0.2                | 27.0.2       | Android SDK Build-Tools
27.0.2
  build-tools;27.0.3                | 27.0.3       | Android SDK Build-Tools
27.0.3
```

*Im Zweifelsfall einfach die aktuellste Version installieren.*

Installation des Paketes:

```
~$ ~/Android/tools/bin/sdkmanager "build-tools;27.0.3"
Warning: File /root/.android/repositories.cfg could not be loaded.
License android-sdk-license:
---------------------------------------
Terms and Conditions

...

14.7 The License Agreement, and your relationship with Google under the
License Agreement, \
shall be governed by the laws of the State of California without regard to
its conflict of \
laws provisions. You and Google agree to submit to the exclusive
jurisdiction of the courts \
located within the county of Santa Clara, California to resolve any legal
matter arising \
from the License Agreement. Notwithstanding this, you agree that Google
shall still be \
allowed to apply for injunctive remedies (or an equivalent type of urgent
legal relief) in \
any jurisdiction.


November 20, 2015
---------------------------------------
Accept? (y/N): y
done
```

*Der Download dauert je nach Anbindung einige Zeit.*

Damit das Programm “`apksigner`„ nicht jedes Mal mit dem vollen Pfad aufgerufen werden muss, könnte man das Verzeichnis wieder in die Pfadumgebungsvariable aufnehmen. Es ist es unter DEBIAN aber auch möglich eine Alternative dafür zu setzen:

```
~$ sudo update-alternatives --install /usr/bin/apksigner apksigner
~/Android/build-tools/27.0.3/apksigner 100
update-alternatives: using /root/Android/build-tools/27.0.3/apksigner to
provide /usr/bin/apksigner (apksigner) in auto mode
```

Für die Signierung der APK-Datei wird dann der vorher erstellte Schlüssel benötigt:

```
~$ apksigner sign --ks mein-apk-key.jks --out tcandroid280-signed-
aligned.apk tcandroid280-unsigned-aligned.apk
Keystore password for signer #1:
```

— *Steffen Bornemann 07.02.2018*

APK, Android, SDK, DEBIAN, Signierung, APK-Multi-Tool, Dex2Jar, JAVA-Decompiler, APKTool