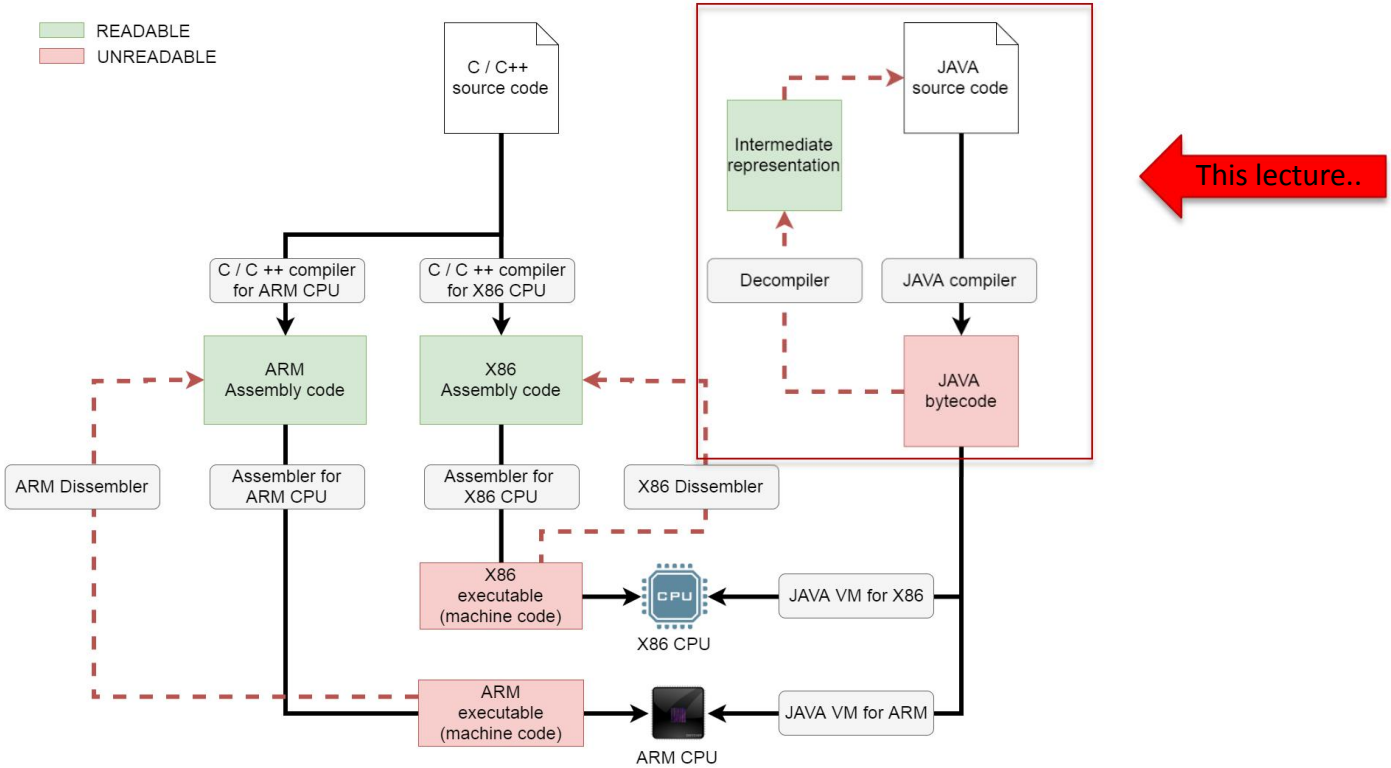
Rev. Eng. Android Apps

Lecture 1

CSCI 4450 & CSCI 6670

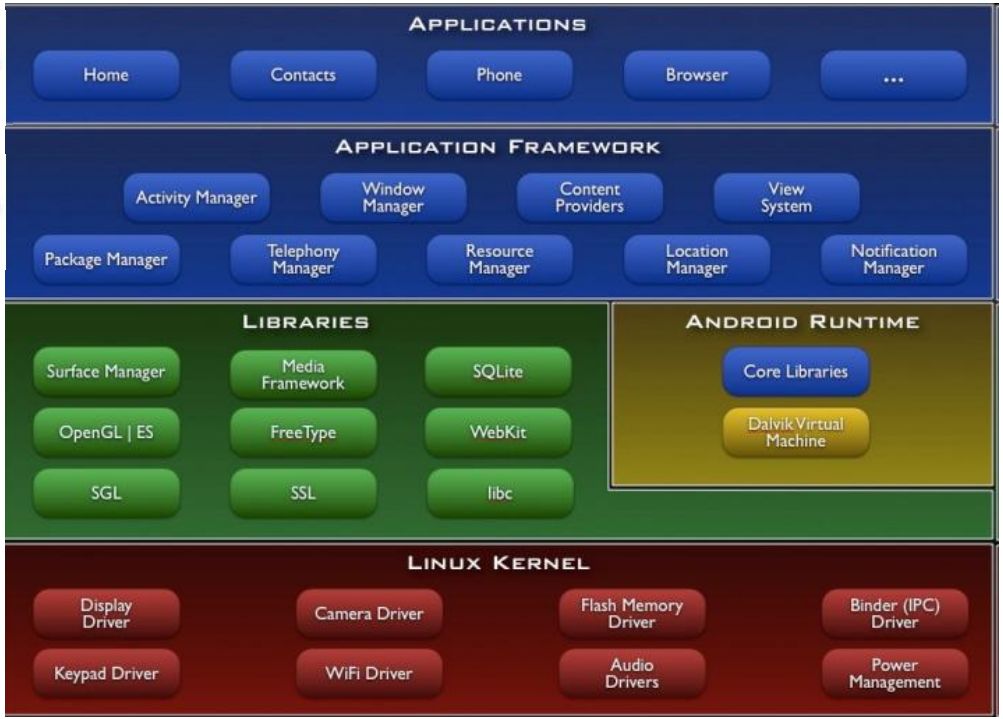
We are here


READABLE
UNREADABLE
C / C++ source code
JAVA source code
Intermediate representation
This lecture..
C / C ++ compiler for ARM CPU
C / C ++ compiler for X86 CPU
Decompiler
JAVA compiler
ARM Assembly code
X86 Assembly code
JAVA bytecode
ARM Dissembler
Assembler for ARM CPU
Assembler for X86 CPU
X86 Dissembler
X86 executable (machine code)
CPU
JAVA VM for X86
X86 CPU
ARM executable (machine code)
JAVA VM for ARM
ARM CPU

Why?

Cause of the Apps

and the framework



Objectives

- Memorizing the APK structure
- Define the AndroidManifest.xml structure
- Analyze DEX files

What is an APK file?

- APK is the file extension of Applications (Apps)
- Used by the Android operating system for distribution and installation of mobile apps and middleware (AppStore).
- *A ZIP* file that can be unziped by common tools, e.g., 7zip or winzip.
- Will be stored on the Android device after being installed.

Inside APK file

META-INF	
res	
assets	
lib	✔
AndroidManifest.xml	✔
resources.arsc	
classes.dex	✔

✔ items are most important for APK analysis

Inside APK file (con't)

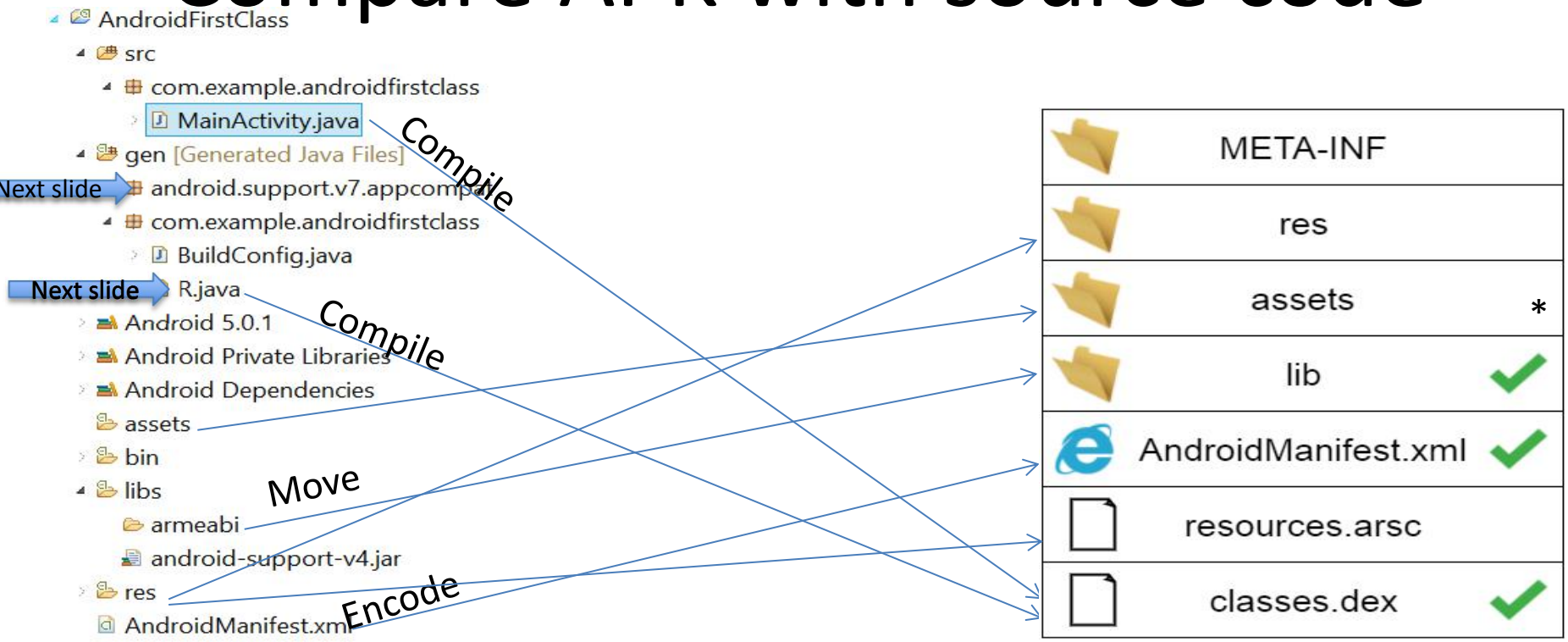
META-INF: files with hash values
res: .xml files, pictures etc.
assets: pictures
lib: .so files
AndroidManifest.xml: meaningless
Resources.arsc: meaningless
classes.dex: meaningless

✔ items are most important for APK analysis

APK content summary

- **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.
- **res directory** folder containing resources not compiled into resources.arsc.
- **lib** optional folder containing compiled code - i.e. native code libraries.
- **AndroidManifest.xml** is more or less the link between the two, providing some additional information about the application to the OS.
- **resource.arsc** file containing precompiled application resources, in binary XML.
  - note: when rev. eng. this might be a place where you find strings!
- **classes.dex** application code compiled in the dex format.

Compare APK with source code



* assets is optional. It stores raw asset files. The difference between res and assets is that each file in the res directory is given a pre-compiled ID registered in R.java

Android support libraries

- android.support.v7.appcompat is an Android support library.
- Android support libraries can provide newer features and classes to an earlier versions of Android.

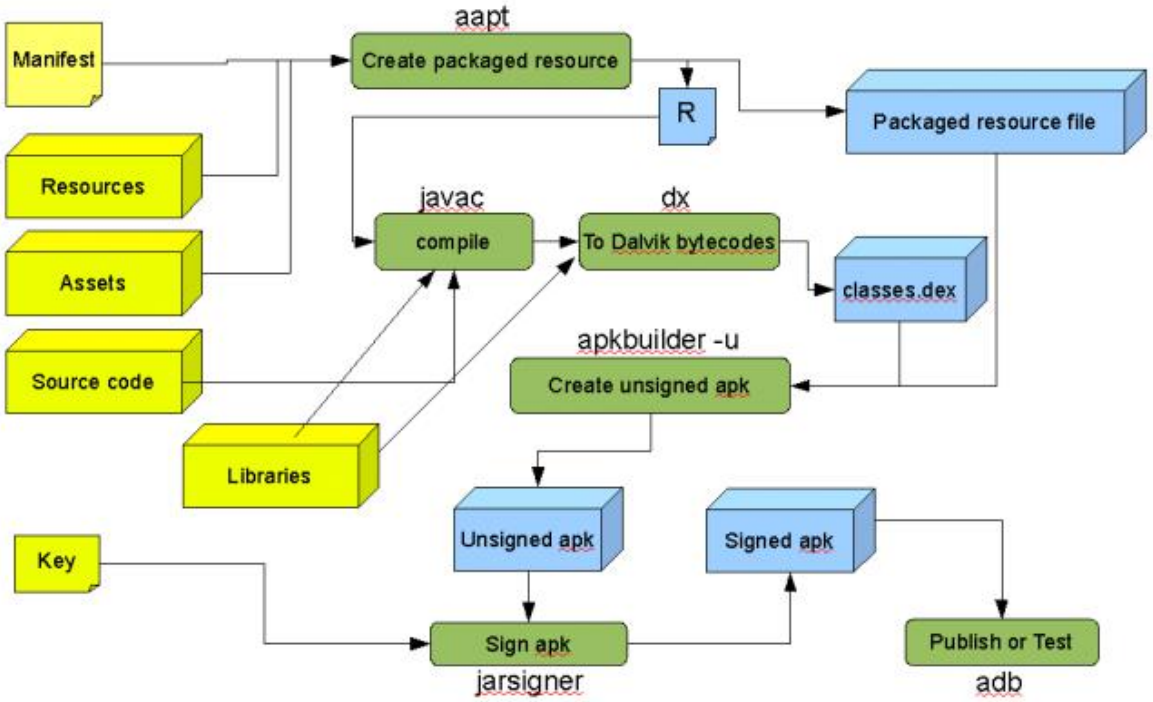
R.java

- Apps use resources such as strings, drawbles, layouts, and styles.
- Instead of hard coding such resources into an application, one externalizes them and refers to them by ID.
- The R.java file contains all those resource IDs, whether assigned by the programmer or generated by the sdk.

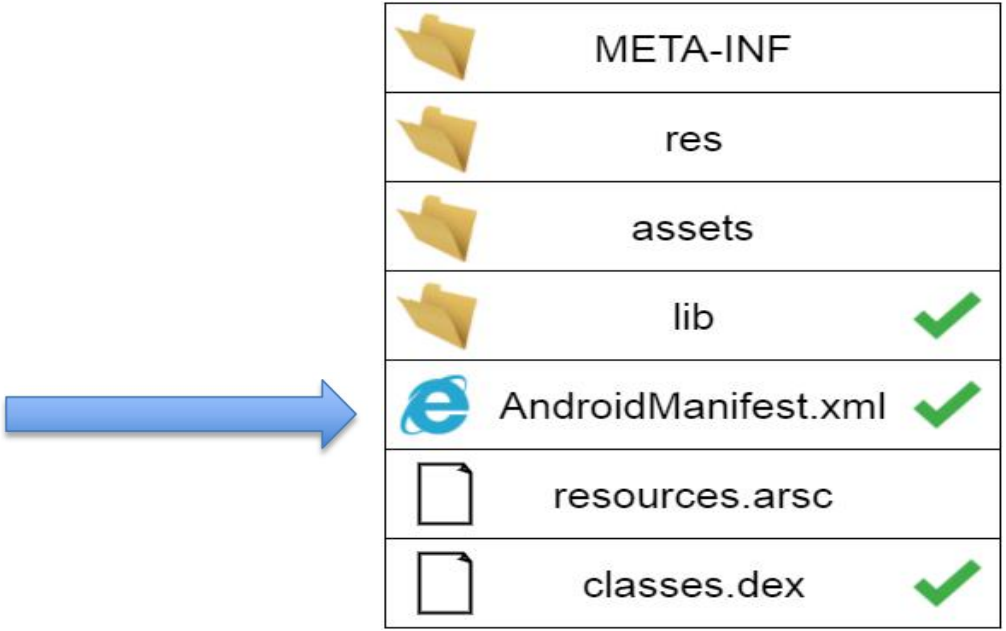
https://asantoso.wordpress.com/2009/09/15/how-to-build-android-application-package-apk-from-the-command-line-using-the-sdk-tools-continuously-integrated-using-cruisecontrol/

APK file generation

Input files
working steps
Output files
Tools



aapt = Android Asset Packaging Tool



ANDROIDMANIFEST.XML

AndroidManifest.xml

- An encoded XML (serialized) file that every application **must** have (with exactly that name) in its **root directory**. The manifest file provides **essential information** about your app to the Android system, which the system must have before it can run any of the app's code.
- *https://developer.android.com/guide/topics/manifest/manifest-intro.html*

What is provided by the manifest?

- Version of the Application
- Debugable or not
- Permissions
  - E.g., my App requires Internet, cam, phone contacts,...
- Unique name of the Application
- Main activity (programming entrance)
- Components
  - I.e., activities, services, content provider,..
- In short – it is a summary of the Application!
  - Metadata

How to decode the manifest file?

- ApkTool (recommended)
- AXML2xml
- AXMLPrinter2
- Android Asset Packaging Tool (aapt in Linux).

https://developer.android.com/guide/topics/manifest/manifest-intro.html

AndroidManifest structure

```
<manifest>      ------------------------------------1st layer
        <application>       -------------------------------------2nd layer
                        <activity>       --------------------------------------3rd layer
                        <service>
                        <receiver>
                        <provider>
                        <uses-library/>
        </application>
        <uses-permission/>
        <permission/>
        <permission-tree/>
        <permission-group/>
        <instrumentation/>
        <uses-sdk/>   ------------------------------API level
        <uses-configuration/>   -----------------Hardware/software features needed
        <uses-feature/>
        <supports-screens/>
</manifest>
```

Most important elements

- **manifest** is the root element. It has package attribute that describes the package name of the activity class.
- **application** is the subelement of the manifest, which mainly contains several subelements that declares the application component such as activity etc.
- **activity** is the subelement of application and represents an activity component that must be defined in the AndroidManifest.xml file.
- **intent-filter** is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.
- **permission** is a restriction limiting access to a part of the code or to data on the device.

Tag: <manifest>

```
<manifest
        ...
        package="com.woody.test"
        android:versionName="string"
        ...
    >
</manifest>
```

Tag: <application>

```
<application
    android:allowClearUserData=["true" | "false"]
    ...
    android:permission="string"
    android:debuggable=["true" | "false"]
    >...
 </application>
```

Tag: <activity > & <intent-filter>

```
<activity
      android:name=".TestActivity"                    \\First activity after the app is launched
      android:label="@string/app_name"
      android:permission="string"
      android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
      android:configChanges="keyboardHidden|orientation|keyboard" >
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

Permissions

android.permission.CALL_EMERGENCY_NUMBERS

android.permission.READ_OWNER_DATA

android.permission.SET_WALLPAPER

android.permission.DEVICE_POWER

...

<uses-permission/> & <permission/>

<uses-permission/>: If an application needs access to a feature protected by a permission, it must declare that it requires that permission with a <uses-permission> element in the manifest

<permission/>: An application can protect its own components (activities, services, broadcast receivers, and content providers) with permissions that are defined by Android, other applications or its own. A new permission is declared with the <permission> element.

Example: <uses-permission/> & <permission/>

If A dose not have the permission B can not receive the boradcast from A.

Application A:

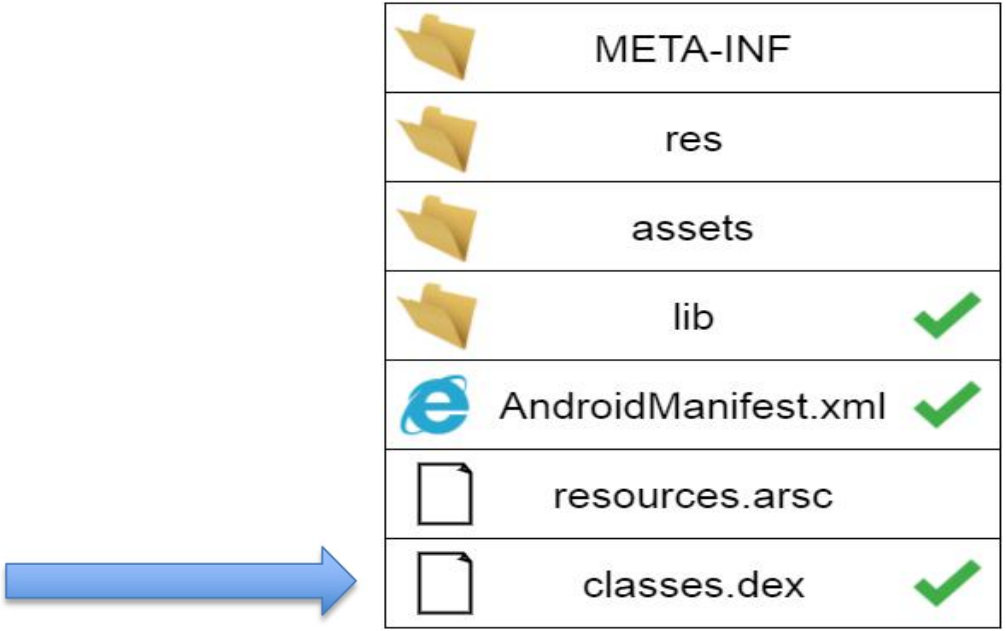
```
</manifest>
        <uses-permission android:name="com.example.testbutton.RECEIVE" />
        <application>
                <activity
                ...
                >
                </activity>
        </application>
</manifest>
```

Application B:

```
</manifest >
        <permission android:name="com.example.testbutton.RECEIVE"
/>
        <application >
                <receiver
                ...
                android:permission="com.example.testbutton.RECEIVE”
>
                </receiver>
        </application>
</manifest>
```

Answer questions

- 1. How many Activities dose the applicaiton have?
- 2. Which one is the main Activity?
- 3. What is the Class name of these Activities?
- 4. Any permissions have been declared?
- 5. What else information you find may help our analysis.


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

DEX FILES

What is Dalvik EXecutable (DEX) file

- Compiled executable Java code → bytecode file
- The structure is similar to *.class files in a JAR file*
  - Organized in a “certain” way
- Not readable out of the box
  - but can be **decompiled**!

What is included in a DEX file?

- Compiled source code written by developers.
- Android support libraries (e.g. Support Library v4 and appcompat library v7).
- 3rd party libraries (JAR files).
- Compiled R.java.

Overview decompiling DEX file

Possibilities to analyze DEX:

- Many tools that convert (details next slide)
  - DEX to Smali
    - Smali is an intermediate representation of the compiled code (details later).
  - DEX to JAVA
  - Other repersentations
  - Manual analysis tools

Summary for the decompilers

	Tool	Description
DEX to Smali		
1	ApkTool	decompiles APK file
2	Baksmali	disassembles DEX file to smali files
DEX to JAVA		
3	Dex2jar	converts DEX file to JAR file
4	Ded	converts DEX file to `.class` files
5	Dare	converts DEX file to `.class` files
3	JD-GUI	converts JAR file to JAVA source
4	JAD	converts JAR file to JAVA source
Manual analysis		
6	Androguard	reverse engineering APK file
7	IDA Pro	reverse engineering a wide range of binaries
8	JEB	reverse engineering APK file

Recommended tools [1]

- **ApkTool**
  - Decompiler for APK files (decode manifest & DEX to smali)
  - *Also allows to recompile applications!*
  - https://ibotpeaches.github.io/Apktool/

ApkTool - decompile

```
$ apktool d testapp.apk
I: Using Apktool 2.0.0 on testapp.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: 1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
$
```

ApkTool - recompile

```
$ apktool b test
I: Using Apktool 2.2.1 on test
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
```

Recommended tools [2]

- **Dex2jar**
  - Converts .dex file to .class files (zipped as jar)
  - https://github.com/pxb1988/dex2jar
- **JD-GUI**
  - is a standalone graphical utility that displays Java source codes of ".class" files.
  - http://jd.benow.ca/
- *Thus we can see Java Code!*
  - Drawback: we cannot recompile

JD-HUI

Java Decompiler - EntityCategory.class

File Edit Navigate Search Help

petstore-ejb3.0.jar

- META-INF
- petstore.ejb
  - dao
    - EJBCatalogDAO
    - EJBCatalogDAORemote
  - model
    - EntityCategory
    - EntityCategoryDetail
    - EntityItem
    - EntityItemDetail
    - EntityProduct
    - EntityProductDetail
  - util

EJBCatalogDAO.class | EntityCategory.class

```
@Entity
@Table(name="CATEGORY")
public class EntityCategory
{
  @PersistenceContext
  private EntityManager em;
  private int id;
  private Map<String, EntityCategoryDetail> detail

  public EntityCategory()
  {
31  this.id = 0;
32  this.details = new HashMap();
  }
```

Find: id Next Previous Case sensitive

Common tool usage

Often we use all 3 tools as Java Code is easier to understand than smali but smali code allows us to change things, repack it and run it again.

APK FILE MODIFICATION

APK file Repacking

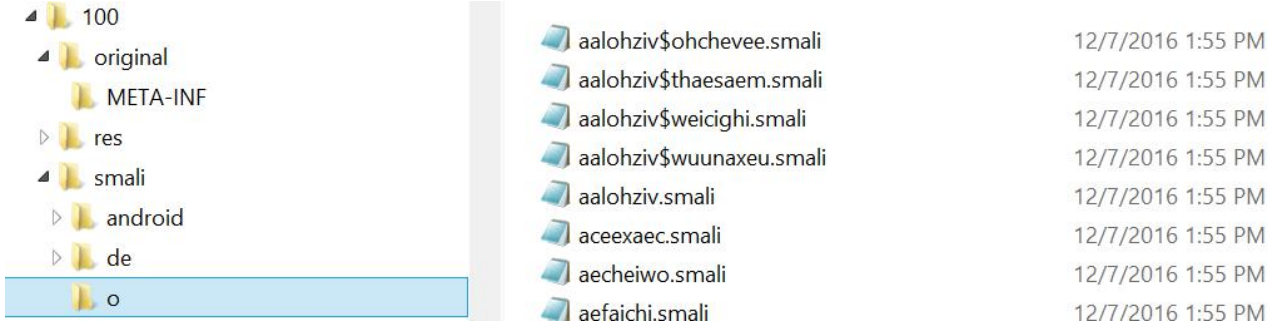
1. Decompile the APK file.
   - ApkTool (slide ApkTool – decompile)
2. Analyze and modify the Smali code.
   - TextEditor (next slides!)
3. Recompile the APK file.
   - APKTools (slide ApkTool – recompile)
4. Resign the APK file.
   - JarSigner

Once APK file is decompiled

- Dex file will be decompiled to smali files in folder 'smali’
  - Usually smali files are in folder 'smali/android' (the decompiled Android support libraries) can be ignored for the analysis.
- AndroidManifest.xml is decoded which then can be opened by TextEditor.
- XML files that define resources in 'res' folder are decoded (read with TextEditor).

Smali overview

- Smali files will be stored in folder “smali”
- Each individual smali file represents one java class
  - Any ‘$’ in the smali file’s name means it’s an inner class in Java



Smali code will be discussed next lecture in more detail!

Smali file

```
.class public Lcom/apkudo/util/Serializer;
.super Ljava/lang/Object;
.source "Serializer.java"

# static fields
.field public static final TAG:Ljava/lang/String; =
"ApkudoUtils"

# direct methods
.method public constructor <init>()V
    .registers 1

    .prologue
    .line 5
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
.end method
```

Class information

Static fields

Methods

Simple analysis of a DEX file

1. Locating the crucial code (breakthrough).
   1. Searching key word(s) in decompiled JAVA code or smali code (e.g. hints, key functions, APIs).
   2. Start from the code entrance.
      - can be found in the AndroidManifest.xml
   3. In case you find an interesting variable:
      - Add toast (=popup) in smali code, repack and run.
2. Changing the code flow.

Recommendation

1. Decompile DEX to JAVA code (DEX2JAR & JD-GUI) in order to locate the crucial code that you are willing to modify.
2. In the same class and function, locate the same code in smali file.
3. Modifying the smali file and repack.

Example for the recommendation

CFR Decompiler - Editable: false

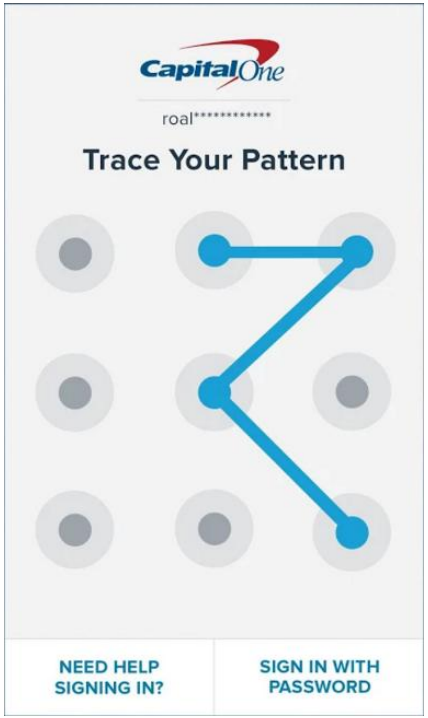
```java
/*
 * Decompiled with CFR 0_115.
 *
 * Could not load the following classes:
 *  aac
 *  com.callpod.android_apps.keeper.BaseFragmentActivity
 */
import com.callpod.android_apps.keeper.BaseFragmentActivity;
import java.lang.invoke.LambdaForm;

public final class aac
implements Runnable {
    private final BaseFragmentActivity a;

    private aac(BaseFragmentActivity baseFragmentActivity) {
        this.a = baseFragmentActivity;
    }

    public static Runnable a(BaseFragmentActivity baseFragmentActivity) {
        return new aac(baseFragmentActivity);
    }

    @LambdaForm.Hidden
    @Override
    public void run() {
        BaseFragmentActivity.a((BaseFragmentActivity)this.a);
    }
}
```

Smali Decompiler - Editable: false

```smali
.class public final synthetic Laac;
.super Ljava/lang/Object;

# interfaces
.implements Ljava/lang/Runnable;


# instance fields
.field private final a:Lcom/callpod/android_apps/keeper/BaseFragmentActivity;


# direct methods
.method private constructor <init>(Lcom/callpod/android_apps/keeper/BaseFragmentActivity;)V
    .registers 2

    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    iput-object p1, p0, Laac;->a:Lcom/callpod/android_apps/keeper/BaseFragmentActivity;

    return-void
.end method

.method public static a(Lcom/callpod/android_apps/keeper/BaseFragmentActivity;)Ljava/lang/Runnable;
    .registers 2

    new-instance v0, Laac;

    invoke-direct {v0, p0}, Laac;-><init>(Lcom/callpod/android_apps/keeper/BaseFragmentActivity;)V

    return-object v0
.end method


# virtual methods
```

(1.1) Locating crucial code

APP:



strings.xml

```
<string name="agree">AGREE</string>
<string name="alp_ELI_compare_title">Trace Your Pattern</string>
<string name="alp_ELI_confirm_name">Express Sign In Enabled</string>
<string name="alp_ELI_intro_name">Get Express Sign In</string>
<string name="alp_ELI_pattern_container_info">Express Login Pattern Grid</string>
<string name="alp_ELI_process_name">Create Your Pattern</string>
<string name="alp_cmd_next">Next</string>
<string name="alp_cmd_passwordsignin">Sign In With Password</string>
<string name="alp_cmd_patternhelp">Pattern Help</string>
<string name="alp_cmd_restart">Restart</string>
```

Smali file:

```
sget v1, Lgroup/pals/android/lib/ui/lockpattern/r$e;->alp_ELI_compare_title:I

invoke-virtual {v0, v1}, Landroid/widget/TextView;->setText(I)V

iget-object v0, p0, Lgroup/pals/android/lib/ui/lockpattern/a;->h:Landroid/widget/TextView;

new-instance v1, Ljava/lang/StringBuilder;

invoke-direct {v1}, Ljava/lang/StringBuilder;-><init>()V

sget v2, Lgroup/pals/android/lib/ui/lockpattern/r$e;->alp_msg_draw_pattern_to_unlock:I
```

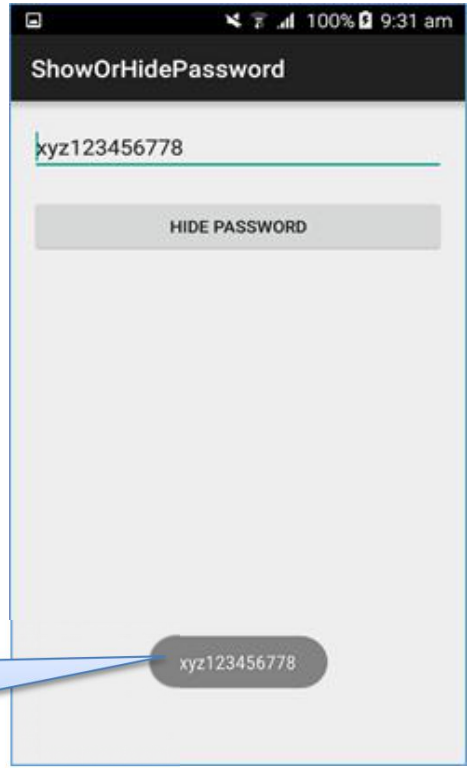
(1.2) Start from the code entrance

```
- <application android:name="com.capitalone.mobilebanking.MainApplication" android:theme="@style/MobileBankingTheme"
  android:largeHeap="true" android:label="@string/app_name" android:icon="@drawable/launcher_icon" android:allowBackup="true">
      <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
      <meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="AIzaSyDFzhuyjd2y4oK-PaWVMizbQL8f0TxB7no"/>
      <activity android:name="com.capitalone.mobilebanking.RestartAppDialogActivity" android:label="Restart App Dialog Activity"/>
      <activity android:name="com.capitalone.mobilebanking.SplashActivity" android:label="@string/app_name"
          android:finishOnTaskLaunch="true"/>
    - <activity-alias android:name="com.konylabs.capitalone.EnterpriseMobileBanking.LaunchActivity"
      android:targetActivity="com.capitalone.mobilebanking.SplashActivity">
        - <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
          </intent-filter>
      </activity-alias>
```

(1.3) add toast message

- const/4 v0, 0x1
- const-string v1, "YOUR MESSAGE"
- invoke-static {p0, v1, v0}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
- move-result-object v0
- invoke-virtual {v0}, Landroid/widget/Toast;->show()V

can be a variable



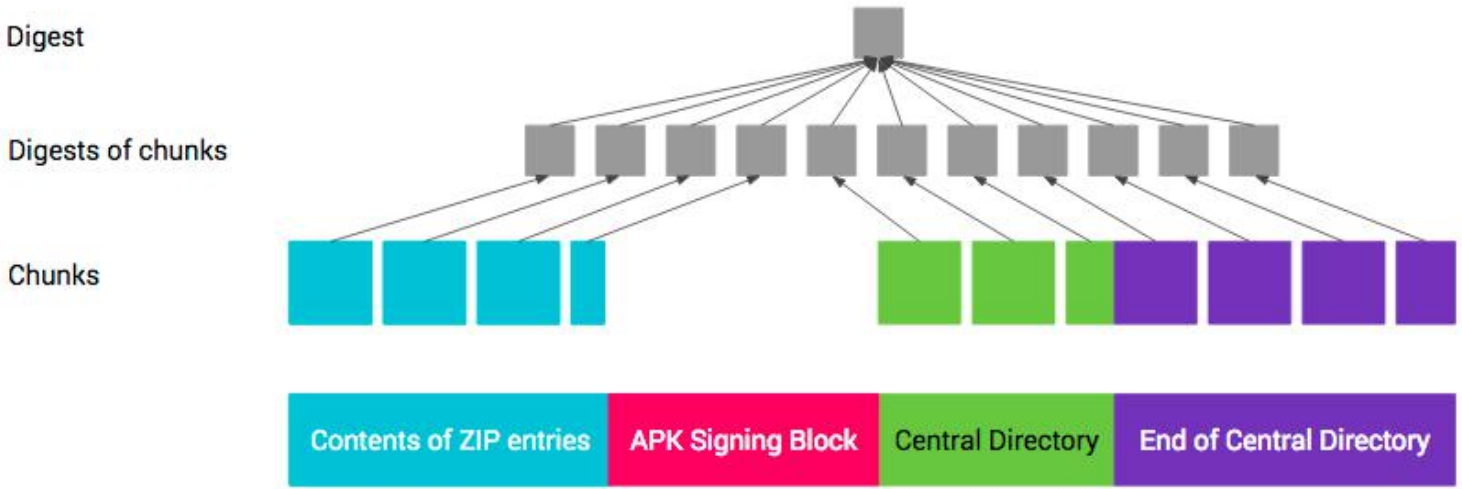
This is the toast do show a variable (in that case the password)

2. Changing the code flow

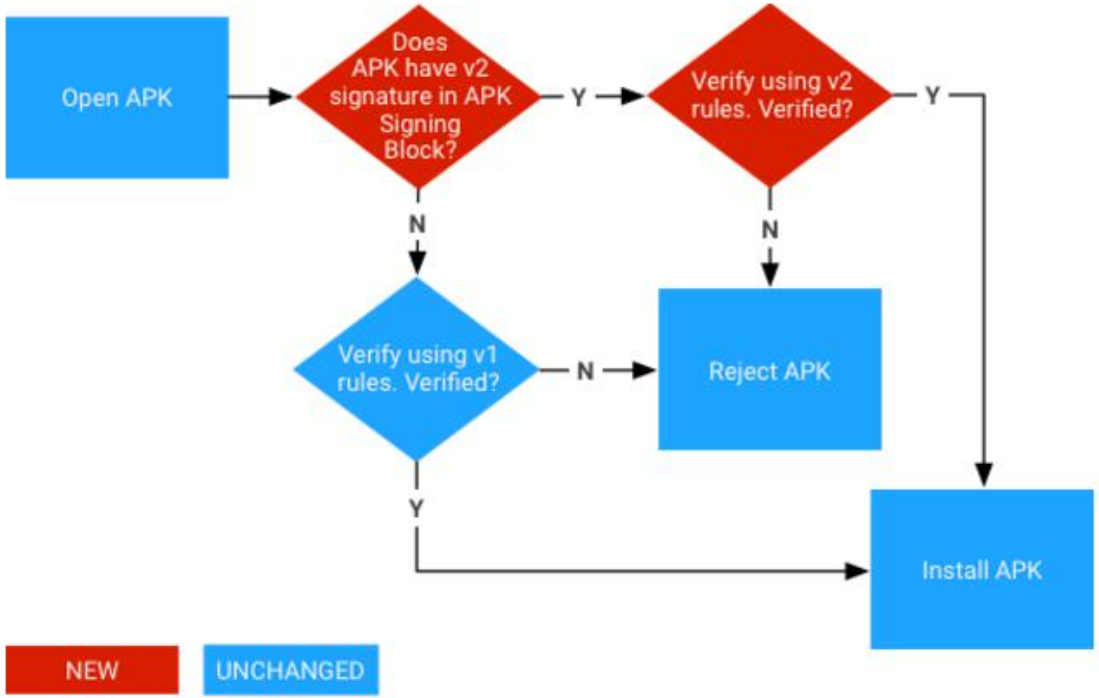
- If we delete this line the program will log in without verifying the password.
- If we change “if-eqz” to “if-nez”, we will log in when the password is incorrect.

```
if-eqz v1, cond_0
.line 35
iget-object v1, p0, Lcom/example/simplistexample/MainActivity$1;->this$0:Lcom/example/simplistexample/MainActivity;

const-string v2, "loged"

invoke-static {v1, v2, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

    .
    .
    .

:cond_0
iget-object v1, p0, Lcom/example/simplistexample/MainActivity$1;->this$0:Lcom/example/simplistexample/MainActivity;

const-string v2, "invalid password and you are stupid"

invoke-static {v1, v2, v3}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
```

APK Signature Scheme



APK Signature Verification

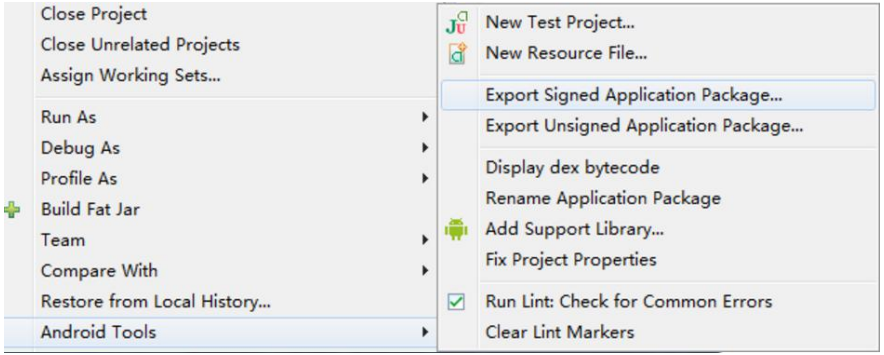


Tools for APK signing

- **jarsigner**:
  - default signing tool for java application
  - Using keystore for signing
- **signapk**:
  - default signing tool for Android application
  - Using pk8 and x509.pem for signing

How to creat a keystore (jarsigner)

- Eclipse:



- Keytool:
  - jdk_xx\jre\bin**keytool -genkey -alias mykey -keyalg RSA -validity 40000 -keystore demo.keystore**
  - If JDK version is higher than 1.7 add **-digestalg SHA1 -sigalg MD5withRSA**

jarsigner

Eclipse

Preferences

type filter text

- General
- Android
  - Build
  - DDMS
  - Editors
  - Launch
  - Lint Error Checking
  - LogCat
  - Usage Stats
- C/C++
- Fat Jar Preferences
- Help
- Install/Update
- Java
- Memory Analyzer
- Run/Debug
- Team
- XML

Build

Build Settings:

- [x] Automatically refresh Resources and Assets folder on build
- [x] Force error when external jars contain native libraries
- [x] Skip packaging and dexing until export or launch. (Speeds up automatic builds on file save)

Build output

- (•) Silent
- ( ) Normal
- ( ) Verbose

Default debug keystore: C:\Users\i\.android\debug.keystore

MD5 fingerprint: 90:99:63:B4:0A:23:E3:DA:08:00:85:96:CA:3A:89:F5

SHA1 fingerprint: D5:78:AE:52:4D:6C:68:28:B7:90:4B:59:11:C8:86:22:83:32:4C:34

Custom debug keystore: Browse...

MD5 fingerprint:

SHA1 fingerprint:

jarsigner command

- This is the command to sign the APK; afterwards it can de deployed

jarsigner -verbose -keystore demo.keystore -signedjar test_signed.apk test.apk mykey

  - demo.keystore: name of the keystore file
  - test.apk : name for the nsigned APK
  - test_signed.apk : name for the signed APK
  - mykey: alias

LAB 01

Setting up the environment

install

- ApkTool
- DEX2JAR
- JD-GUI
- jarsigner

Lab 01 – Task 1

Crack the Log-me-in app developed by another team

Objective:

1. Successfully log in and see the greeting page.
2. Retrieve the password (if possible, if not explain why).
3. Change the password (if possible).

Lab report:

Write a report and document the procedure with screen shots. The report should includes:

1. Explain what method was used for storing the password.
2. Provide the screen shot of the greeting page and explain the procedure.
3. Provide the old password if is able to be retrieved, if not expain why.
4. Provide the recompiled APK file (repacked_login.apk) and the new password (if it is changed).

Lab 01 – Task 2

Delete the advertisement of TicTacToe.apk

- Provide the executable repacked game APK.
- Sign the repacked APK by your own signature.
- Provide the report with the whole procedure and screenshots.