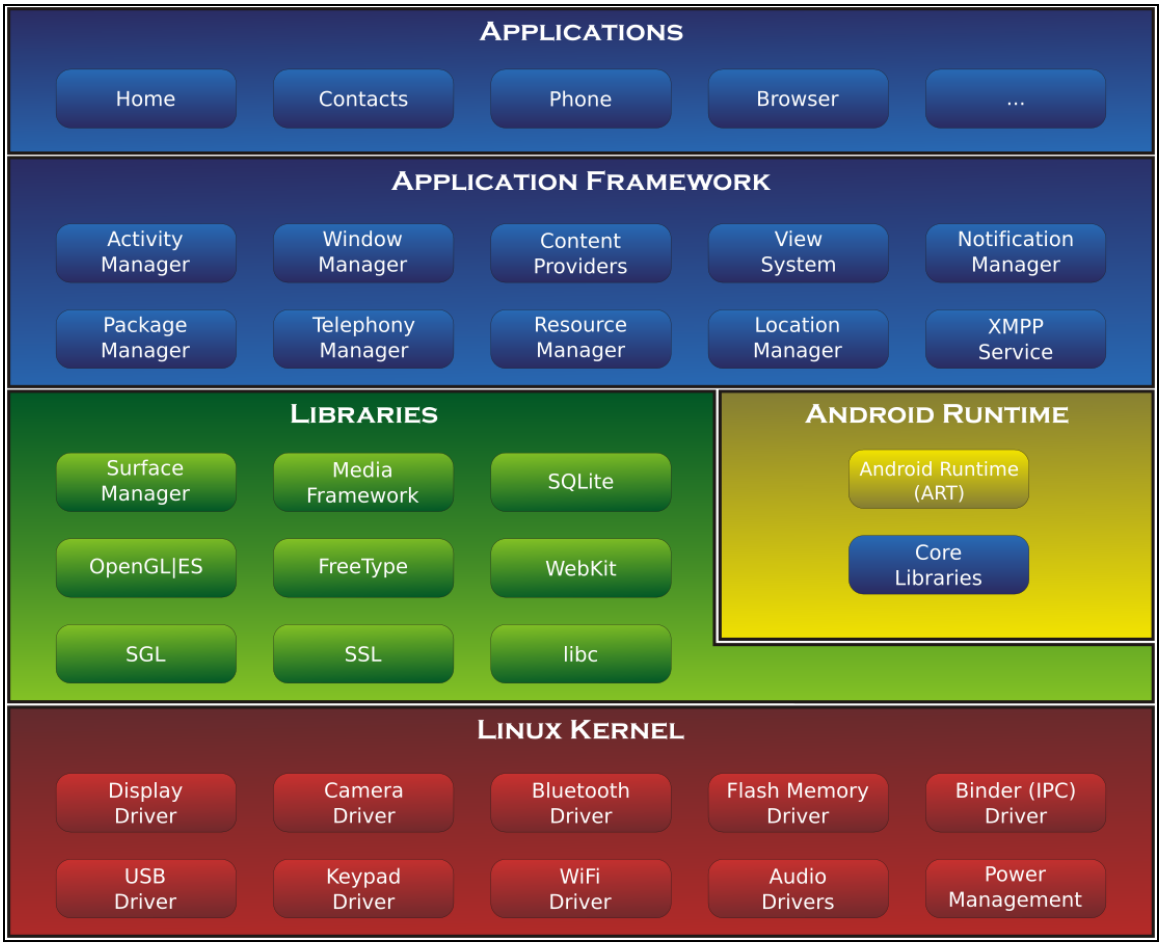
Android Application Penetration Testing Training Module

1. Android Architecture



The above diagram explains the different layers that are used between the Application and Hardware level to establish communication & perform different tasks.

- **Linux kernel** is the bottom layer and heart of the android architecture. It manages all the device drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc which are mainly required for the android device during the runtime. Linux kernel is responsible for power management, memory management, device management, resource access, etc.
- **Libraries** is a set of platform libraries such as WebKit, Surface Manager, OpenGL, SQLite, Media, SSL, etc which provides the device with a set of instructions that allow the android device to handle different types of data in an appropriate way.
- **Android Runtime** is an engine that powers the android applications along with the libraries which form the basis for the application framework. The Android Runtime contains components like core libraries and Dalvik Virtual Machine(DVM) which is responsible for running android applications.
  - **DVM** is a kind of Java Virtual Machine(JVM) specially designed and optimized for mobile devices. The DVM enables every Android application to run in its own

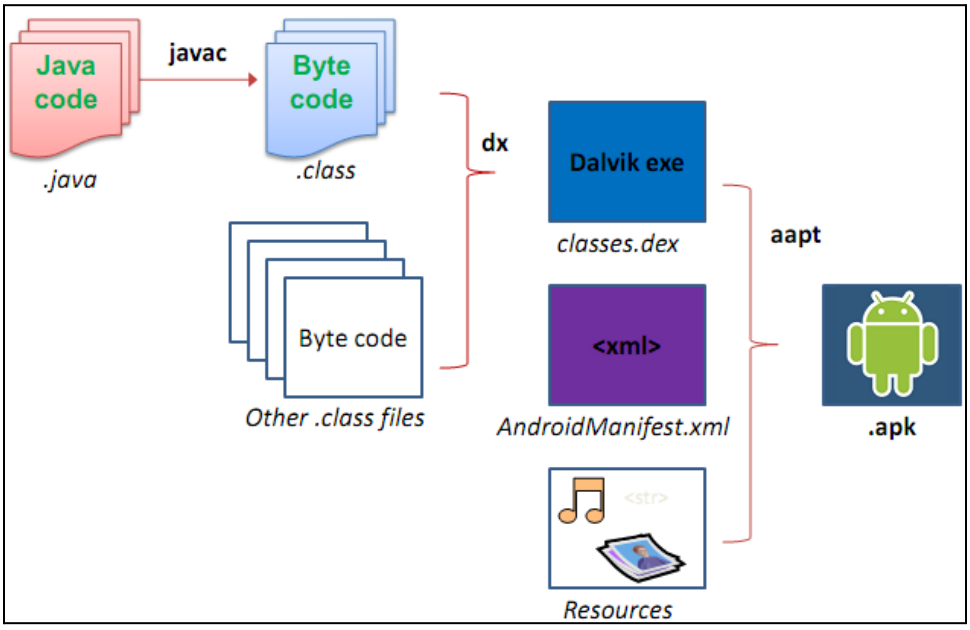
process, with its own instance by consuming less memory and providing fast performance.

- ➢ **Core libraries** in the Android runtime will enable us to implement android applications using the standard JAVA or Kotlin programming languages.

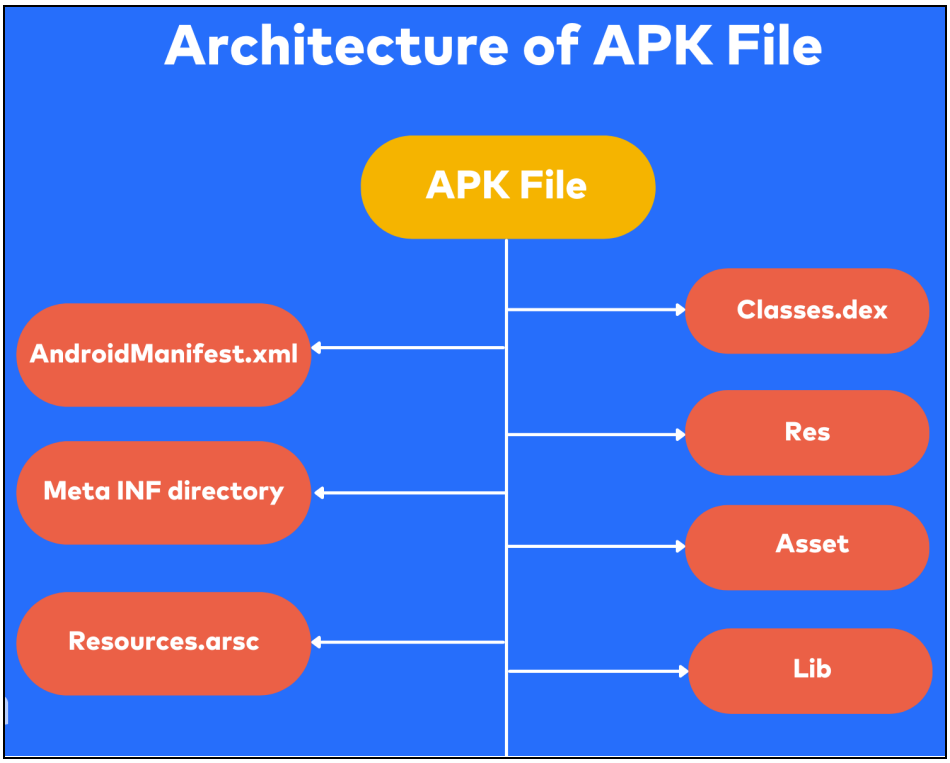
- ❖ **Application Framework** provides many higher-level services to applications in the form of Java classes. Application developers make use of these services when they develop applications. The application framework includes services like telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.
- ❖ **Applications:** All the native and third-party applications installed on the device reside in this layer and also typical user interaction happens in this layer. The application layer runs within the Android run time environment by utilizing the classes and services provided by the application framework. All applications such as home, contact, settings, games, browsers are using the android framework that uses android runtime and libraries.

2. Android Application Structure

- ❖ Android applications use "**.apk**" (Android Application Package/Android Package Kit) as a file extension, which is nothing but a Zip file containing a predefined structure that is used to store different components of an application.



- ❖ Android applications are written primarily in Java, Kotlin and C++ programming languages.



❖ An **.apk** file structure is as follows:

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

➔ **AndroidManifest.xml** file contains the meta-information about the application, such as the name of the application, package name, different activities and services, permissions required, supported version of Android, etc. It's like a blueprint of an android application.

➔ **META-INF/** folder is essentially a manifest of metadata information, including the developer certificate such as its signature.

➔ **classes.dex** file contains application code in the Dex file format which contains the Java libraries that the application uses.

➔ **lib/** is a directory with compiled native libraries used by the application.

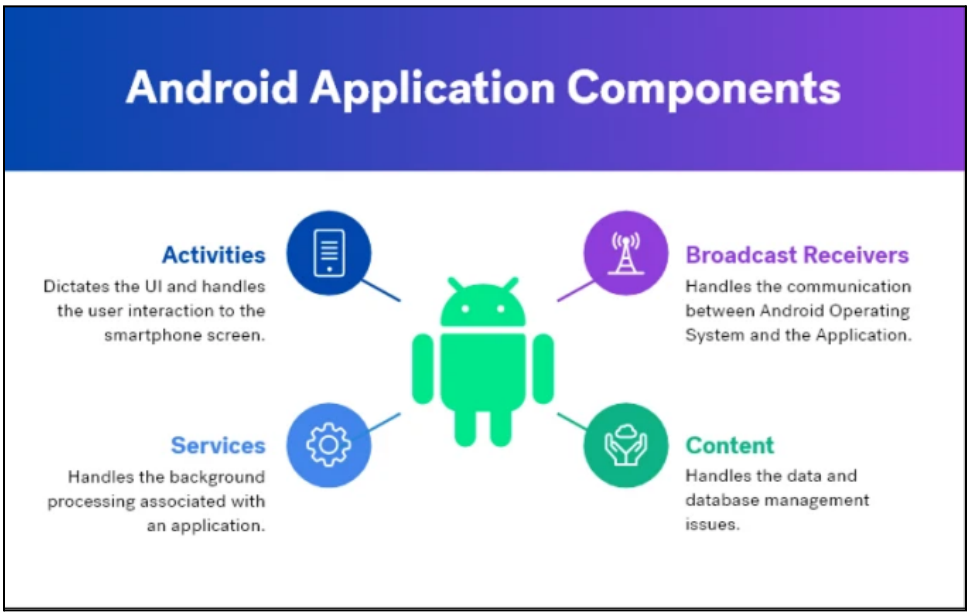
➔ **res/** The 'res' stands for Resource file. It can store resource files such as pictures, XML files, etc. It contains some additional folders such as Drawable, Layout, and Values.

➔ **assets/** A directory with application assets.

➔ **resources.arsc** Application resource file which contains compiled resources in a binary XML format. This may include images, strings or other data used by the application.

3. Android Application Components

Android Application components are the essential building blocks which are loosely coupled by the application manifest file 'AndroidManifest.xml' that describes each component of the application and how they interact.



❖ Typical Android application components include:

- ➢ **Activities** represent a single screen with a user interface(on-screen components).
- ➢ **Services** are components that run in the background to perform long-running operations(background tasks).
- ➢ **Broadcast Receivers** handles the communication between the android OS and the application(notifications).
- ➢ **Content Providers** supply data from one application to others on request. It handles the data and the data management issues.

4. Reverse Engineering Android Application

Reverse engineering refers to the process of taking something apart to see how it works, whether it's a physical object such as a lock or in this case, a mobile application. Decompiling is a form of reverse engineering in which a mobile app is analyzed by looking at its source code.

1. **Apktool** is a popular free tool used for reverse engineering closed, third-party, and binary Android applications. It can disassemble Java bytecode to the '.smali' format as well as extract and disassemble resources from APK archives. Also, we can use Apktool for patching and changing the manifest file. Click here to download Apktool.
    - Disassembling Android apk file => `apktool d <apk file>`
    - Once disassembled, check 'Manifest.xml' file for
        - Permissions needed for the application to run.
        - `android:debuggable="true"` => is application debuggable.
        - `android:allowBackup="true"` => is application backup enabled.
        - `android:usesCleartextTraffic="true"` => is cleartextTraffic enabled.
        - `android:exported="true"` => is application components exported.
        - `setFilterTouchesWhenObscured="true"` => is Tapjacking enabled.
2. **Analyzing .apk file**: To analyze the android application, rename the file extension '.apk' to '.zip' and extract the '.zip' file contents. Once extracted, look for '.dex' files(ie classes.dex).
2. **Dex2jar** is a free tool for converting bytecode from the '.dex' format into Java class files(.jar).
    - `d2j-dex2jar <classes/filename.dex>` => converts '.dex' file into '.jar' file.
3. **JD-GUI** is a graphical utility tool that makes Java source code readable, displaying it as Java class files. Click here to download Jd-GUI.
    - `jd-gui <filename.jar>` => to read java class code(.jar). Check for code obfuscation, hardcoded credential/sensitive datas, etc.
4. **Jadx** is a command-line+GUI tool used for producing Java source code from Android Dex and Apk files. Its main feature is to decompile Dalvik bytecode(.dex) to java classes. Click here to download Jadx-GUI
    - jadx-gui <filename.apk> => opens the '.apk' and shows java code.

5. Android Debug Bridge (ADB)

Android Debug Bridge (ADB) is a command-line tool that is used to communicate with devices. It has multiple device actions, such as installing the application, debugging, backup, and push or pull data from the device. Enable developer options and enable USB debugging in the emulator/rooted device. Click here to download ADB.

- `adb connect <device ip:port>` => to connect a device using adb.
- `adb devices` => Lists all adb connected devices.
- `adb kill-server` => to stop running adb server.
- `adb install <path/apk file/application_name.apk>` => to install an application using adb.
- `adb uninstall <apk file/application_name.apk/package_name>` => remove application package from the device using adb.
- `adb forward tcp:<port> tcp:<port>` => port forwarding using adb.

- adb pull <path_to_the_remote_file> => Copy a file/directory from the device.
- adb push <path_of_the_local_file> <path_to_the_remote_destination> => Copy a file/directory to the device
- adb shell => Initiate an adb shell

Click here to know more adb commands

6. Android Application Security Assessment with Drozer

Drozer is a comprehensive security and attack framework for Android developed by MWR Labs. It allows us to interact with the Dalvik VM, other apps IPC endpoints and the underlying OS. To use drozer, a drozer agent application(.apk) has to be installed on the emulator/rooted device. Click here to download Drozer.

To use drozer

- Connecting Drozer Client-Server
    - Open the drozer application in the device and turn on 'Embedded Server'.

    - adb forward tcp:31415 tcp:31415
    - drozer console connect

      Once connected we'll get a drozer shell, where drozer commands can be executed.

  Click here for Drozer Commands

7. Checking Insecure Logging using Logcat

- adb devices => First connect the device/emulator with adb.
- adb shell ps | findstr <package_name> or adb shell ps | grep <package_name> => Find the <process_id> of the application using <package_name>.
- adb logcat | findstr <process_id> or adb logcat | grep <process_id> => Run logcat with application <process_id> and look for sensitive information logged in while using the application.

8. Important Directories/Files to Look for in Android

- **/data/data** => This directory contains all the applications that are installed by the user.
- **/data/data/<package_name>** => This directory contains data stored for specific applications.
- **/data/data/<package_name>/shared_prefs** => SharedPreferences are objects that point to XML files in order to read and write on them. These XML files may contain sensitive data in a collection of key-value pairs.
- **/data/data/<package_name>/databases** => Locally stored database files. Once the database is identified, we can use the `sqlite3`/DB Browser tool to read its contents
- **/data/data/<package_name>/tmp** => Temporary directory.
- **/data/data/<package_name>/Cache** => Cache directory.
- Overall Check data stored inside '**/data/data/<package_name>/**' directory

9. Extracting an APK file

Once an android application is installed(ideally from playstore), its '.apk' file can be extracted from the device. For all android applications, its 'apk' file will be stored in **/data/app/<package_name>** directory with name '**base.apk**', where <package_name> differs depending on the applications.

- adb shell pm list packages | grep <application name> or adb shell pm list packages | findstr <application name> => To get application package name.
- adb shell pm path <package_name> => To get the path of the '.apk file' using package name.
- adb pull /data/app/com.example.myapp-1/base.apk => The following command retrieves the 'base.apk' file to our system.

10. Running Static Analysis with MobSF

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pentesting framework capable of performing static, dynamic, and malware analysis. It can be used for effective and fast security analysis of Android, iOS, and Windows mobile applications and support both binaries (APK, IPA, APPX) and zipped source code. MobSF can also perform dynamic testing of the application.
https://mobsf.github.io/docs/#/

11. Setting Up Proxy

Configuring proxy

12. Introduction to OWASP Mobile Top 10

1. **Improper Platform Usage:**
2. **Insecure Data Storage:**
3. **Insecure Communication:**
4. **Insecure Authentication:**
5. **Insufficient Cryptography:**
6. **Insecure Authorization:**
7. **Client Code Quality:**

8. **Code Tampering:**

9. **Reverse Engineering:**

10. **Extraneous Functionality:**

13. Tools Used

- Apktool
- Dex2jar
- JD-GUI
- Jadx
- ADB
- Drozer
- Logcat
- Root Browser
- XML viewer
- Sqlite3 db
- DB Browser (SQLite)
- Mobile Security Framework (MobSF)
- Burp Suite
- Frida
- Objection
- Xposed Framework(Rootcloak, SSL Unpinning, etc)

13. References

- https://book.hacktricks.xyz/mobile-pentesting/android-app-pentesting
- https://www.cobalt.io/blog/getting-started-with-android-application-security
- https://www.hackthebox.com/blog/intro-to-mobile-pentesting