Saathi

Date __/__/__ Mobile Application Penetration Testing

① Android Architecture :

Layer 1 → Linux Kernel : The first layer to interact with the hardware.
Provides fundamental software needed to boot, manage power & memory, device drivers, processes, applications, networking & security.

Layer 2 → Libraries : Transaction layer between the kernel & application framework, providing some of the common services available for apps and other programs.

Instructions for kernel to perform is given by libraries.

Android Runtime : Consists of DVM that interacts with the app & runs the app.

The libraries & DVM combines to become ART.

Layer 3: Application Framework : It contains the code compiled for the DVM & running on the DVM.

Layer 4: Application : The top layer where actually our apps are installed.

Page No.

Date __/__/____

Saathi

- Android application files has .apk extensions which is equivalent to .zip file. And unzip command could easily unzip an .apk file but it would be encoded.

- The apk file has — AndroidManifest.xml, classes.dex, resources.arsc, res, META-INF.

- AndroidManifest.xml — file which has all permissions, activities, contentproviders, services, intents required for the app to run is listed.

- classes.dex — Contains Java Byte Code in DEX i.e. Dalvik Exchange format. This .dex file can be decompiled and the application source code can be read.

- res — A folder which has device configurations, Bitmaps & Layouts.

- resources.arsc — A file containing binaries of compiled components which might include images, strings, or other data used by an app.

- META-INF : folder that contains the manifest information and other metadata about the java package carried by the jar file.

  It contains files like Manifest.mf, cert.sf & cert.rsa.

Page No.

Date __/__/____

Saathi

① Basic Terms:

1. **Activity**: The screen of the app. The webpage to a website is as same as an activity to an app.

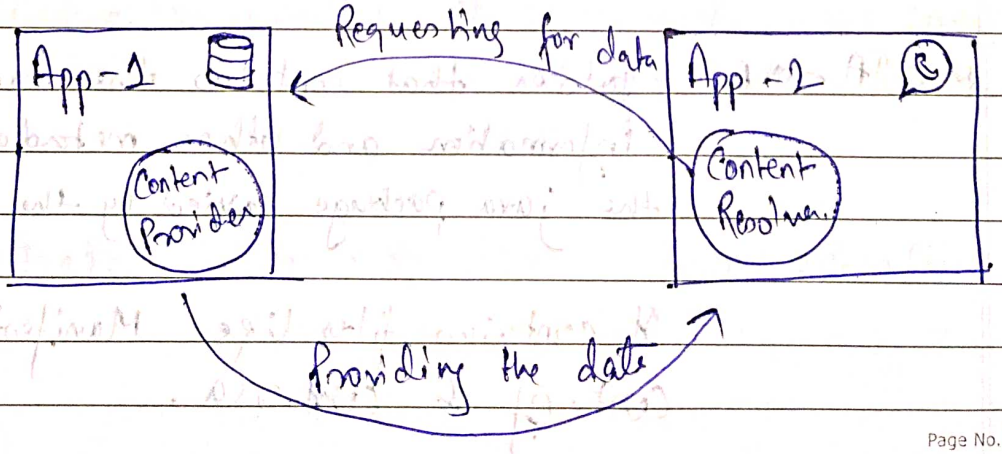
2. **Services**: Long running task that requires no UI and has to explicitly stopped by the user. Background tasks like API is an example.

3. **Broadcast Recievers**: Messages that are sent to the phone are called broadcasts. So components that listens for these notifications are called broadcast recievers.
   charging notification is an example.

4. **Content Providers**: Data sharing link between two apps

5. **Content Resolver**: Component present in an app that requests for data.

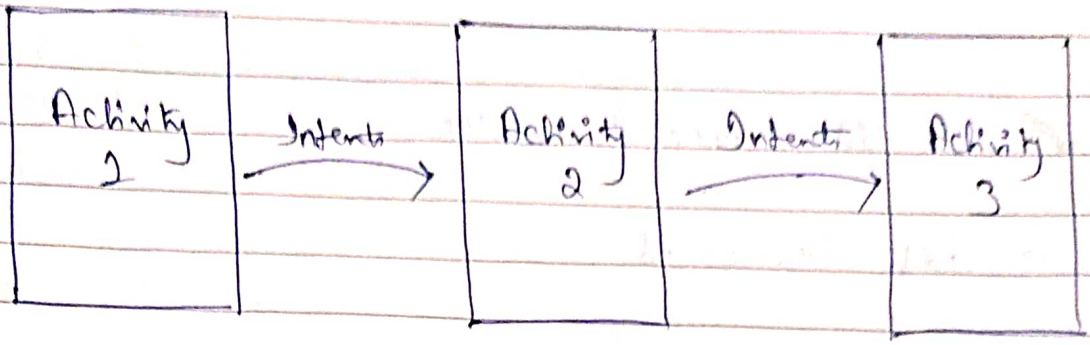
Diagram



Date __/__/____

Saathi

6. **Intent** : To communicate between activities.



1. Explicit : Intent that communicate between 2 or more activities of similar app
   e.g: clicking on Buy button leads to checkout page
2. Implicit : Intent that communicates between 2 or more activities of different app.
   e.g: clicking on link in mail leads to browser.

7. **Intent Resolution** : Process which checks which implicit intent needs to call which activity.

Tools : apktool → To decompile apps & make it readable
jadx → To view the content of apk files
MOb-SF → For static & dynamic analysis of an apk
Burp → Proxy tool to intercept/modify requests
Android Studio → IDE for app dev & also for AVD
adb → To get shell & access the terminal of android
Frida → For dynamic analysis and many more
Drozer → For dynamic analysis and many other different stuff.
jarsigner → To verify if the app is signed or not & to sign it.
keytool → To generate keys for app to sign it.
hexdump & dexdump → Convert byte code to hex format
objection → To use frida

Date __/__/____

Saathi

ADB Commands:

✓. To list connected devices :

→ adb devices -l

✓. To connect to a device :

→ adb connect <phone_ip>

✓. To disconnect a device.

→ adb ~~devices~~ disconnect

✓. To get shell of a device

→ adb shell.

✓. To start an activity through an intent

→ am start -a android.intent.action.VIEW http://www.google.com

✓. Reset adb host connection.

→ adb kill-server , adb start-server

✓. To send commands to a specific device

→ adb -s <device_name> install helloworld.apk

Date __/__/____

Saathi

✓. Port forwarding in adb

→ adb forward tcp:6100 tcp:7100

✓. Copying file to/from a device

→ adb pull <app.name> , adb push <app.menu>

✓## To verify whether an application is signed or not.

→ jarsigner -verify -verbose filename.apk

✓. To install an application

→ adb install filename.apk

✓. To uninstall an application

→ adb uninstall <package.name>

✓. To list the processes

→ adb shell ps

✓. To get logs of the android

→ adb logcat

✓. To see packages & to see the path of the package

→ pm list packages , pm path <package.name>

Saathi

Date __/__/____

Logging based Vulnerabilities

Step 1: Decompile the app using apktool or jadx.

Step 2: Search of Log.d function or method into the source code and carefully search for any strings or api keys or any other sensitive information.

Reversing an application

Step 1: Decompile the apk using apktool or jadx.
Step 2: Look for smali files and make necessary changes
Step 3: After recompile it using apktool and the sign the app with jarsigner.

Signing an Application

Step 1: keytool -genkey -v -keystore custom.keystore -alias mykeyname -keyalg RSA -keysize 2048 -validity 10000
(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m)

a - The toolname
b - generate key
c - verbose
d - repo where keys are stored
e - keystore repo name, anything would do
f - nickname for keystore
g - name of the alias
h - algorithm to be used for the key
i - the algorithm itself
j - size of your key
k - the size
l - validity of your key
m - no. of days

Page No.

Date ___/___/_____

Saathi

Step2: jarsigner -sigalg SHA1withRSA -digestalg SHA1 -keystore mycustomkeystore -storepass mystorepass repackaged.apk mykeyaliasname

1 2 3 4 5 6 7 8 9 10 11

1 - the tool
2 - signing algorithm
3 - the algorithm
4 - digest algorithm
5 - The algorithm
6 - the keystore
7 - name of the keystore we created
8 - the store password
9 - the password given in prev. step
10 - The apk to be signed
11 - The alias name we given in prev. step.

Step 3: jarsigner -verify repackaged.apk

→ To verify the signature of the app

Step 4: zipalign 4 repackaged.apk repackaged_final.apk

The signed apk — The final apk

alignment in bytes, 4 means 32-bit alignment.

zipalign is a zip archive alignment tool. It ensures that all uncompressed files in the archive are aligned relative to the start of the file. It should be used to optimize your APK file before distributing it to end-users.

Page No.

Date __/__/____

saathi

SSL Pinning Bypass

Step 1: Configure Burp & Emulator

Step 2: Install Xposed Installer apk in the emulator from the internet.

Step 3: Open the app and click on Install & Update and click on Install

Step 4: Then on the top-left corner you will get 3 dashes so click on it and go to download.

Step 5: Search for SSL & install both the 2 modules.

Step 6: Go to the modules then click on versions and then click downloads

Step 7: Then on the 3 dash menu, click on modules. Once there check on both the modules to start auto download.

Step 8: Minimize all the apps & on the menus of android, click on SSLUnpinning to open the app.

Step 9: After opening the app, select the app you would like to Bypass SSL.

Step 10: It is done & you can intercept the request in Burp.

* SSL Certificate Pinning is the process of associating a host with its certificate or public key to verify the authenticity. Bypassing it is known as SSL Pinning Bypass.

Page No.

Date __/__/____

Saathi

Some vulnerabilities:

1. Insecure logging
2. Hardcoded string
3. Insecure Data Storage
4. Input Validation Issues
5. Insecure Activity Access
6. Content Provider Injections
7. Insecure transmission of data
8. Lack of binary protections
9. Client Side Vulnerabilities

Problems with data on different layers:

1. Network layer: Data travelling from mobile applications from the device over wifi and data service.
2. Hardware layer: Baseband attacks, broadband attacks, and RF range attacks that can affect mobile features
3. Operating System Layer: Jailbreaking & rooting vulnerability in mobile platforms
4. Application Layer: API of the device without administrative permission.

Methodologies:

Discovery	> Analysis	> Exploitation	> Reporting
• OWASP, Understanding the platform, Client Side vs Server side Scenarios	Static, local file, archive, network & web traffic, Rev. Engg., Inter proces Comm.	Attempt to exploit the vuln., Priv. Escalation	Risk assessment Reporting

Saathi

Date __/__/____

OWASP Mobile Top 10 :

1. **Weak Server Side Controls :** Compromising application servers that forms the backbone of these application must be secured on their own. So vulns like Injections, IDOR, insecure communication may lead to complete compromise of it and attackers who have gained control over the ~~users~~ compromised server can push malicious content to all the application users and compromise user devices as well.

2. **Insecure Data Storage :** Storing sensitive information in log files, XML files, databases in an insecure manner.

3. **Insufficient Transport Layer Protection :** Since most of the data are prone to tampering so, SSL/TLS controls, which enforces confidentiality and integrity of data must be verified for correct implementations.

4. **Unintended Data Leakage :** Certain functionalities which are unintentionally place for better performance and user experience might leak data ~~which~~ or the data might be accessible to all or by a malware.

Date __/__/____

Saathi

5. Poor Authorization & Authentication: Since mobile devices are personal devices, so developers tends to keep sensitive data in the phone itself via some safety mechanisms and only authorized users could access it. If that mechanism is poorly built then all the authorized will be available to the non-authorized adversary.

6. Broken Cryptography: Algorithms that are meant to keep the data protected, if they are not implemented properly or if the public key & private key are not managed carefully then anyone could access the keys and could access the data.

7. Client Side Injections: Injecting queries of SQL, or commands of bash or any other programming based commands to alter, fetch, read, delete or accum data and compromise it.

8. Security Decisions via Untrusted Input: The implementation of certain functionalities such as use of hidden variables to check the authorization status can be bypassed by tampering them during transit via web service calls or interprocess communication calls. This may lead to privesc or unintended behavior of app.

Page No.

Saathi

Date __/__/____

9. Improper Session Handling: Obtaining of session tokens that are active for longer time may lead to account takeover if stolen.

10. Lack of binary protections: Since a mobile application can be reverse engineered. So, if an attacker inserts malicious code and ships to the user then the phone can be compromised. So when building an app, if the signing & checksum is not done correctly then the above havoc could occur.

Vulnerable Apps to Practice:

1. iMAS
2. Insecure Bank
3. Injured Android
4. DIVA & DVIA
5. Goat Droid
6. iGoat
7. Mobisec

Page No. ____