Android Testing Cheat Sheet

Source: https://www.owasp.org/index.php/Android_Testing_Cheat_Sheet

OWASP Step-by-step Approach

(For each of the standards below, there shall be multiple steps for the tester to follow])

M1 - Improper Platform Usage [Client Attacks]

- Check AndroidManifest.xml permissions configurations certain permissions can be dangerous
- If application uses fingerprinting, test against different vulnerabilities related to this feature

M2 - Insecure Data storage [Client Attacks]

This Section should be ideally tested after using the application for some time. This way application has time to store some data on the disk. It will probably require the use of a rooted Android device in order to access files in Android such as '/sdcard' Commonplaces to look at

- /data/data/app_folder
- /sdcard/
- /sdcard1/

Android applications need to store data locally in sqlite files or XML structures and hence need to performs either SQL/XML Queries or file I/O.

This gives rise to 2 major issues.

1. SQL / XML injection, and if the reading intent is publicly exposed another application could read this.
2. Local file read which can allow other application to read files of the application in question and if they contain sensitive data then data leakage via this media.

If the application is a HTML5 hybrid application then Cross Site Scripting (XSS) should also be considered. XSS will expose the entire application to the attacker as HTML5 applications will have the ability to call native functionality and hence control over the entire application. (WebViews)

Additionally a backup of the application could be made using `adb backup` option and that can be analyzed to identify what the application stores and leaks when the client interacts with it.

M3 - Insufficient Transport Layer [Network/Traffic attacks]

Multiple layer of checks to be performed here

1. On Server side

- Identify all ssl endpoints.
- Perform SSL Cipher Scan using (sslscan)[1] or similar software.
- SSLv2, SSLv3 is disabled

- TLS 1.2 and 1.1 is supported (1.2 is essential to ensure highest possible secure connection)
- RC4 and CBC Based Ciphers are disabled
- DH Params are >2048 Bits
- SSL Certificate is signed with atleast sha2 / sha256
- ECDHE Ciphers / Ciphers supporting Perfect forward secrecy are preferred
- SSL Certificate is from Trusted RootCA
- SSL Certificate is not expired
- Verify Interprocess communication implementation

2. On Device Side

- Ensure application is working correctly by navigating around.
- Put a proxy in between the application and remote server. If application fails to load. Application might be doing cert validation. Refer logcat if any message is printed.
- Place Proxy RootCA in trusted root CA list in device. (Burp)[2] (OWASP-ZAP)[3]
- Try using application again. If application still doesn't connect, application might be doing cert pinning.

You could bypass the certification pinning by Hooking or changing the Smali code:

Using Xposed:

- Install (Xposed Framework)[4] and (Just Trust Me)[5], enable JustTrustMe and then reboot device.
- Try again if everything works we have a application which employs certification pinning and we have bypassed it using the xposed module.
- Android Security provider has been properly updated against SSL Exploits

Changing SMALI:

- Identify/search for the methods where the certificate pinning is implemented (keywords like 'sha256/' followed by a certificate value such as "sha256/wl0L/C04Advn5NQ/xefY1aCEHOref7f/Q+sScuDcvbg="
- Change the value of the certificates being used by the one used by you generated BURP certificate

M4 - Insecure Authentication [Client/Server attacks]

For this part some of the tools necessary in order to carry on the assessment are,

- attack proxies such as ZAP, BURP or Charles
- Wireshark for Traffic analysis

By analyzing the traffic (HTTP request/response) between client and server, check the following items

- Analyze session management and workflow
- Analyze API authentication using an attack proxy
- Insecure WebViews
- Check if Credentials are being stored in the Datastore or Server side
- Misuse or access to Account Manager
- Authenticating Callers of Components

Improper Session Handling typically results in the same outcomes as poor authentication. Once you are authenticated and given a session, that session allows one access to the mobile application. There are multiple things to look at

- Check and validate Sessions on the Backend
- Check for session Timeout Protection
- Check for improper Cookies configuration
- Insecure Token Creation
- Insecure implementation of WebView

M5 - Insufficient Cryptography [Client/Network/Server attacks]

For this part , you will need to perform a overall analysis where you can enumerate where encryption has been used. For example:

- Type SSL/TLS encryption used
- Retrieving files securely using HTTPS URI or a secure tunnel such as implementing HttpsURLConnection or SSLSocket
- Authentication Session Tokens
- Data storage containing sensitive information in clear text
- Access to encryption keys or improper key management
- Usage of known weak crypto algo's like Rot13, MD4, MD5, RC2, RC4, SHA1
- Do it Yourself / let me design my own algo for encryption
- Secret key hard coded in the application code itself.
- Implementation of own protocol
- Insecure use of Random Generators

M6 - Insecure Authorization [Client/Server attacks]

After a proper Application mapping and understanding the data flow, you can verify the authorization mechanism for the following:

- Handling credentials: does the application make use of authorization tokens instead of asking Credentials all the time?
- Verify that the application allows access only to the allowed roles
- Storing username and password in the data storage instead of using AccountManager

M7 - Client Code Quality [Client Attacks]

There are 2 approaches for this part:

- If you have access to the source code, doing code review on the Client app and the server API.
- If you don't, you could still check the code by decompiling the APK

We strongly recommend a Code Review in this case. This will definitely extract many potential vulnerabilities due to bad implementation

M8 - Code Tampering [Client Attacks]

For this part you will need a rooted device and reverse engineering techniques

- Decompile APK using tools such as apktool, dex2jar / enjarify, Bytecodeviewer or commercial tools such as JEB
- Analyze code using decompiler such as JD-GUI or Bytecodeviewer. Commercial version such as JEB allows you to even debug the decompiled application but not in all cases

- After analyzing the code, attempt to bypass functionalities whether by changing the Smali code or Hooking methods using Xposed or Frida frameworks
- Verify if the application has been obfuscated and verify the level of obfuscation searching for specific strings.
- Decompile APK and change Smali, (check this tool which automates the process of decompiling , compiling and signing the application: https://github.com/voider1/a2scomp)
- Android Binaries are basically dex classes, which if not protected can result in an easy decompilation of source code. This could lead to code / logic leakage.

Following controls need to be checked for and validated:

- Jailbreak, Device Rooted- Detection Controls
- Checksum Controls
- Certificate Pinning Controls
- Debugger Detection Controls
- Xposed Detection Controls
- Dynamic loading code
- use of Native code with Android NDK

References

(OWASP M10-2014)[6]

M9 -Reverse Engineering [Client attacks]

Reverse Engineering is an essential part of pen testing mobile applications. It also requires the use of rooted devices. For this part make sure that you have prepared the following tools :

- Android Studio with SDK tools installed
- A rooted Android Device or Emulator
- For a rooted emulator you can use CuckoDroid which has Xposed installed
- Different tools installed for decompiling the APK such as apktool,Dex2Jar / enjarify or if you are looking for a total one use Bytecodeviewer or JEB
- IDA pro (for analyzing code flow)
- Smali decompiler/compiler and signer : https://github.com/voider1/a2scomp

Verify that:

- Has the application be obfuscated?
- Search for key strings and keywords with tools like Bytecodeviewer or JEB
- Search for implantation of SSL pinning , Device rooted or connections to API's (search for words like 'TrustManager' , 'SHA256', X509 ,SHA, SSL , for more info see Android Security Guidelines

M10 - Extraneous Functionality

For testing this part, it will be necessary to do a code review or Reverse engineer the APK (if code is not available)