Mobile Application Security Test Case Handbook

1. Insecure Data Storage

Description:
This issue arises when sensitive information such as tokens, passwords, or personal data is stored locally on the device without encryption. Common storage methods include SharedPreferences, SQLite databases, and internal files.

Why it Happens:
Developers often use default storage mechanisms for convenience, forgetting to encrypt or restrict access.

How to Find:
Use tools like 'adb shell' and 'run-as' to inspect the app directory:

```
  adb shell
  run-as com.example.app
  cat shared_prefs/prefs.xml
```

Risks Involved:
Attackers on rooted devices or with backup access can extract and read these files, leading to token theft and impersonation.

Exploitation Example:
1. Pull SharedPreferences file using adb.
2. View stored auth tokens in plain text.

2. Hardcoded Secrets

Description:
Secrets like API keys, tokens, and credentials are embedded directly in the source code.

Why it Happens:
Often added during development and forgotten in production builds.

How to Find:
Decompile APK using JADX or MobSF, and search for keywords like:

```
  password, token, key, secret
```

Risks Involved:
Exposed keys may be used by attackers to impersonate users or abuse third-party services.

Exploitation Example:
1. Use JADX to open APK.
2. Locate `String apiKey = "123XYZ";`
3. Use the key to access backend services.

3. Insecure Communication (HTTP)

Description:
Sensitive data like login credentials are transmitted over HTTP instead of HTTPS.

Why it Happens:
Developers skip HTTPS due to certificate complexity or lack of awareness.

How to Find:
Configure a proxy (Burp Suite), intercept network requests, and check URL schemes.

Risks Involved:
Unencrypted data can be intercepted via man-in-the-middle (MITM) attacks.

Exploitation Example:
1. Connect app to Burp.
2. Capture login credentials sent via HTTP.
3. Replay or manipulate requests.

4. Broken SSL Pinning

Description:
SSL pinning ensures the app only communicates with a specific certificate. Without it, any trusted CA can be used, allowing MITM attacks.

Why it Happens:
SSL pinning is often skipped due to complexity in implementation and debugging during development.

How to Find:
Intercept HTTPS traffic using Burp. If interception works, SSL pinning is likely missing. You can also use Frida to hook and bypass SSL validation.

Risks Involved:
Attackers can intercept, manipulate, or steal data in-transit even if HTTPS is used.

Exploitation Example:
1. Run the app with Burp proxy.
2. If HTTPS requests are visible, SSL pinning is missing.
3. Use Frida script to bypass if pinning exists:

```
frida -U -n com.example.app -l ssl_bypass.js
```

5. Insecure WebView Usage

Description:
Enabling JavaScript in WebView and loading untrusted or dynamic content can lead to JavaScript injection and data theft.

Why it Happens:
Developers often enable JS (`setJavaScriptEnabled(true)`) and load URLs from user input.

How to Find:
Look for:

```
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl(userInput);
```

Risks Involved:
Can lead to XSS, phishing, or RCE attacks via malicious pages.

Exploitation Example:
1. Inject payload in URL param: <script>alert(1)</script>
2. If executed inside WebView, vulnerability is confirmed.

6. Improper Exported Components

Description:
Activities, services, or broadcast receivers marked as 'exported=true' without permission checks are accessible by other apps.

Why it Happens:
Developers forget to restrict exported components.

How to Find:
Review AndroidManifest.xml for exported components without permissions.

Risks Involved:
Component hijacking, execution of internal logic by a malicious app.

Exploitation Example:
1. Use Drozer to list exported components.
2. Start one using:

```
  adb shell am start -n com.example/.ExportedActivity
```

7. Backup Enabled

Description:
If the 'allowBackup' flag is enabled in the manifest, app data can be extracted using ADB even on non-rooted devices.

Why it Happens:
Developers forget to disable backup by default.

How to Find:
Check AndroidManifest.xml for:

```
  android:allowBackup="true"
```

Risks Involved:
Sensitive data like credentials, tokens, and preferences can be extracted.

Exploitation Example:
1. Run: adb backup -apk -shared -all -f backup.ab
2. Convert .ab to .tar and inspect extracted files.

8. Debug Mode Enabled

Description:
Enabling debug mode in production APKs allows attackers to attach debuggers and extract data.

Why it Happens:

Leftover debug configuration in release build.

How to Find:
Check AndroidManifest.xml for:

```
  android:debuggable="true"
```

Risks Involved:
Reverse engineering, runtime inspection, memory dumping.

Exploitation Example:
1. Run: adb shell ps | grep <app_package>
2. Use jdwp to attach debugger and inspect memory.

9. Logging Sensitive Data

Description:
Sensitive data (e.g., auth tokens, passwords) is written to logs during development.

Why it Happens:
Debug logs are not removed or filtered in production.

How to Find:
Search source for:

```
  Log.d, Log.e, System.out.println, NSLog
```

Risks Involved:
Logcat data leakage, privilege escalation.

Exploitation Example:
1. Run: adb logcat | grep "token"
2. Extract token from logs and reuse in API call.

10. Insecure Intent Handling

Description:
Unprotected intent receivers or activities allow malicious apps to trigger or send arbitrary intents.

Why it Happens:
No validation of intent sources or actions.

How to Find:
Review intent filters and intent usage in code, check for absence of permission checks.

Risks Involved:
Intent spoofing, privilege escalation, unauthorized actions.

Exploitation Example:
1. Send crafted intent with adb:

```
   adb shell am start -a com.example.ACTION
```

2. Observe app behavior.

11. Session Management Flaws

Description:
Session tokens are not invalidated on logout or reused across devices indefinitely.

Why it Happens:
Lack of token lifecycle control and session expiry mechanisms.

How to Find:
Inspect login and logout code, review token issuance and invalidation logic.

Risks Involved:
Session hijacking, account takeover.

Exploitation Example:
1. Capture token before logout.
2. Reuse the same token post-logout in API calls.

12. IDOR (Insecure Direct Object Reference)

Description:
APIs allow access to other users' data by changing the object ID (e.g., user_id=2).

Why it Happens:
Missing access control on object identifiers.

How to Find:
Test API calls in Burp Suite, increment IDs like:
  /api/user/1 -> /api/user/2

Risks Involved:
Data leakage, unauthorized access.

Exploitation Example:
1. Intercept API call.
2. Change ID param to access another user's resource.

13. Missing Input Validation

Description:
Lack of input sanitization leads to injection vulnerabilities like SQLi, XSS, or logic flaws.

Why it Happens:
Developers trust front-end validation only.

How to Find:
Look for unchecked inputs being passed into logic, queries, or WebView.

Risks Involved:
Injection, data corruption, XSS.

Exploitation Example:
1. Inject payload: ' OR 1=1 --
2. Bypass auth or break logic.

14. Missing Code Obfuscation

Description:
The app code is easily readable post-decompilation, exposing sensitive logic and keys.

Why it Happens:
ProGuard/R8 not enabled or misconfigured.

How to Find:
Decompile app with JADX, check if names are readable.

Risks Involved:
Reverse engineering, credential theft.

Exploitation Example:
1. Open APK in JADX.
2. Read logic, keys, URLs, and credentials.

15. Hidden Debug/Test Code

Description:
Leftover debug or test activities, endpoints or UIs are exposed in release builds.

Why it Happens:
Improper build configuration cleanup.

How to Find:
Search for BuildConfig.DEBUG, test class names, hidden activities.

Risks Involved:
Internal logic abuse, privilege escalation.

Exploitation Example:
1. Use Drozer to list activities.
2. Launch unlisted activity:

```
adb shell am start -n com.example/.TestActivity
```