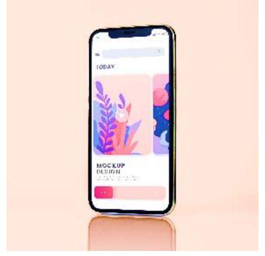
Mobile Hacking

ASSESSING MOBILE APPLICATIONS



CHEAT SHEET

V1.0

android

MAIN STEPS

- Decompile / Disassemble the APK
- Review the codebase
- Run the app
- Dynamic instrumentation
- Analyze network communications

OWASP MOBILE SECURITY PROJECTS

Mobile Security Testing Guide
- https://github.com/OWASP/owasp-mstg

Mobile Application Security Verification Standard
- https://github.com/OWASP/owasp-masvs

Mobile Security Checklist
- https://github.com/OWASP/owasp-mstg/tree/master/Checklists

TOOLS

- adb
- apktool
- jadx
- Frida
- BurpSuite

APK Structure

META-INF
- Files related to the signature scheme (v1 scheme only)

lib
- Folder containing native compiled code (ARM, MIPS, x86, x64)

assets
- Folder containing application specific files

res
- Folder containing all the resources (layouts, strings, etc.) of the app

classes.dex [classes2.dex] …
- Dalvik bytecode of the app

AndroidManifest.xml
- Manifest describing essential information about the app (permissions, components, etc.)

Package Name

The package name represents the app's unique identifier (e.g. for YouTube):
com.google.android.youtube

Data Storage

User applications
```
# /data/data/<package-name>/
```
Shared Preferences Files
```
# /data/data/<package-name>/shared_prefs/
```
SQLite Databases
```
# /data/data/<package-name>/databases/
```
Internal Storage
```
# /data/data/<package-name>/files/
```

Package Manager

List all packages on the device
```
# adb shell pm list packages
```
Find the path where the APK is stored for the selected package
```
# adb shell pm path <package-name>
```
List only installed apps (not system apps) and the associated path
```
# adb shell pm list packages -f -3
```
List packages having the specified pattern
```
# adb shell pm list packages -f -3 [pattern]
```

Application Signing

One-liner to create your own keystore
```
# keytool -genkeypair -dname "cn=John Doe, ou=Security, o=Randorisec, c=FR" -alias <alias_name>
-keystore <keystore_name> -storepass <keystore_password> -validity <days> -keyalg RSA -keysize 2048
-sigalg SHA1withRSA
```
Signing with **jarsigner** (Only supports v1 signature scheme)
```
# jarsigner -verbose -keystore <keystore_name> -storepass <keystore_password>  <APK_file>
<alias_name>
```
Signing with **apksigner** (Official tool from Android SDK which supports all signature schemes)
```
# apksigner sign --ks <keystore_name> --ks-pass pass:<keystore_password>  <APK_file>
```

Code Tampering

1. Disassemble and save the smali code into output directory
```
# apktool d <APK_file> -o <directory_output>
```
2. Modify the app (smali code or resource files)
3. Build the modified APK
```
# apktool b <directory_output> -o <APK_file>
```
4. Sign the APK (see Application Signing)
5. (Optional) Uses **zipalign** to provide optimization to the Android APK
```
# zipalign -fv 4 <input_APK> <output_APK>
```

Content Provider

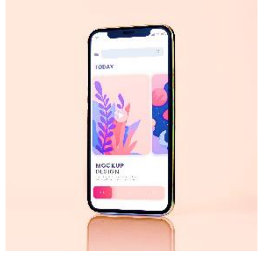
Query a Content Provider
```
# adb shell content query --uri content://<provider_authority_name>/<table_name>
```
Insert an element on a Content Provider
```
# adb shell content insert --uri content://<provider_authority_name>/<table_name>
--bind <param_name>:<param_type>:<param_value>
```
Delete a row on a Content Provider
```
# adb shell content delete --uri content://<provider_authority_name>/<table_name>
--where "<param_name>='<param_value>'"
```

Activity Manager

Start an Activity with the specified Intent
```
# adb shell am start -n <package_name/activity_name> -a <intent_action>
```
Start an Activity with the specified Intent and extra parameters
```
# adb shell am start -n <package_name/activity_name> -a <intent_action> --es <param_name>
<string_value> --ez <param_name> <boolean_value>  --ei <param_name> <int_value> …
```

Mobile Hacking

ASSESSING MOBILE APPLICATIONS



CHEAT SHEET

V1.0

android

MAIN STEPS

- Decompile / Disassemble the APK
- Review the codebase
- Run the app
- Dynamic instrumentation
- Analyze network communications

OWASP MOBILE SECURITY PROJECTS

Mobile Security Testing Guide
- https://github.com/OWASP/owasp-mstg

Mobile Application Security Verification Standard
- https://github.com/OWASP/owasp-masvs

Mobile Security Checklist
- https://github.com/OWASP/owasp-mstg/tree/master/Checklists

TOOLS

- adb
- apktool
- jadx
- Frida
- BurpSuite

SSL/TLS Interception with BurpSuite - Before Android 7

1. Launch Burp and modify Proxy settings in order to listen on “All interfaces” (or a specific interface)
2. Edit the Wireless network settings in your device or the emulator proxy settings
3. Export the CA certificate from Burp and save it with “.cer” extension
4. Push the exported certificate on the device with adb (into the SD card)
5. Go to “Settings->Security” and select “Install from device storage”
6. Select for “Credentials use” select “VPN and apps”

SSL/TLS Interception with BurpSuite - After Android 7

1. Install BurpSuite certificate on your device (see Before Android 7)
2. Disassemble the APK with apktool
3. Tamper the **network_security_config.xml** file by replacing the **<pin-set>** tag by the following

```
<trust-anchors>
      <certificates src="system" />
      <certificates src="user" />
</trust-anchors>
```

4. Build and sign the APK (see Code Tampering)

Bypass SSL Pinning using Frida

1. Install Burp certificate on your device (see SSL/TLS Interception with BurpSuite)
2. Install Frida (Frida – Installation)
3. Use “Universal Android SSL Pinning Bypass with Frida” as follow:

```
# frida -U --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida -f <package_name>
```

Objection - Inject Frida Gadget (non rooted device)

Steps to inject the Frida Gadget library inside an app:
1. Disassemble the app with apktool (see Code Tampering)
2. Add the lib-gadget library (https://github.com/frida/frida/releases) inside the app (lib folder)
3. Modify the smali code to load the lib-gadget (usually on the Main Activity

```
const-string v0, "frida-gadget"
invoke-static {v0}, Ljava/lang/System;->loadLibrary(Ljava/lang/String;)V
```

4. Add the INTERNET permission on the AndroidManifest.xml
5. Rebuild the app with apktool and sign it (see Code Tampering and Application Signing)

Inject Frida Gadget using Objection

```
objection patchapk --source UnCrackable-Level1.apk -V 12.10.4 --architecture x86_64
```

adb

Connect through USB
```
# adb -d shell
```
Connect though TCP/IP
```
# adb -e shell
```
Get a shell or execute the specified command
```
# adb shell [cmd]
```
List processes
```
# adb shell ps
```
List Android devices connected
```
# adb devices
```
Dump the log messages from Android
```
# adb logcat
```
Copy local file to device
```
# adb push <local> <device>
```
Copy file from device
```
# adb pull <remote> <local>
```
Install APK on the device
```
# adb install <APK_file>
```
Install an App Bundle
```
# adb install-multiple <APK_file_1> <APK_file_2> [APK_file_3] …
```
Set-up port forwarding using TCP protocol from host to Android device
```
# adb forward tcp:<local_port> tcp:<remote_port>
```

Frida – Installation

Install Frida on your system with Python bindings
```
# pip install frida frida-tools
```
Download the Frida server binary (check your architecture): https://github.com/frida/frida/releases)
```
# adb shell getprop ro.product.cpu.abi
```
Upload and execute the Frida server binary (adb service should run with root privileges)
```
# adb root
# adb push <frida-server-binary> /data/local/tmp/frida
# adb shell “chmod 755 /data/local/tmp/frida”
# adb shell “/data/local/tmp/frida”
```

Frida – Tools

List running processes (emulators or devices connected through USB)
```
# frida-ps -U
```
List only installed applications
```
# frida-ps -U -i
```
Attach Frida to the specified application
```
# frida -U <package_name>
```
Spawn the specified application without any pause
```
# frida -U -f <package_name> --no-pause
```
Load a script
```
# frida -U -l <script_file> <package_name>
```