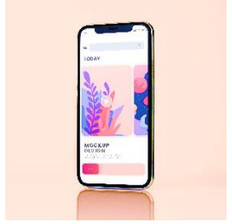
Mobile Hacking

ASSESSING MOBILE APPLICATIONS



CHEAT SHEET

V0.1

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 of the app

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

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

Data Storage

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

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>'"

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 created with the debug keystore provided by the Android SDK
 # jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore <Android_SDK_path>/debug.keystore -storepass android <APK_file> androiddebugkey
5. (Optional) Uses **zipalign** to provide optimization to the Android APK
 # zipalign -fv 4 <input_APK> <output_APK>

Keystore Creation

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 2000 -keyalg RSA -keysize 2048
-sigalg SHA1withRSA

Package Manager

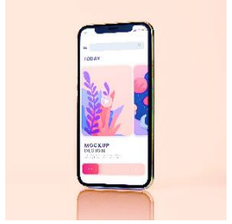
List all packages on the device
 # adb shell pm list packages
Find the path where the APK is stored for the selected package
 # adb shell pm path <package-name>
List only installed apps (not system apps) and the associated path
 # adb shell pm list packages -f -3
List packages having the specified pattern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

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

V0.1

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 Interception with BurpSuite

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”

Bypass SSL Pinning using Network Security Config

1. Install Burp certificate on your device (SSL Interception with BurpSuite)
2. Decompile 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 (Code Tampering)

Bypass SSL Pinning using Frida

1. Install Burp certificate on your device (SSL 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>

Native Libraries

Native libraries are loaded using the following function:
System.loadLibrary("native-lib");
Native functions are used with the **native** keyword:
public native String myNativeFunction();
To reverse native libraries, the common tools can be used such as:
IDA Pro, Radare2/Cutter, Ghidra and Hopper
Intercept native functions and set callbacks with Frida using the **Interceptor** module

```
Interceptor.attach (Module.findExportByName ( “<native-library>", “<function_name>"), {
      onEnter: function (args) { <your_code>},
      onLeave: function (retval) {<your_code>} });
```

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 device
 # adb forward tcp:<local_port> tcp:<remote_port>

Frida – Installation

Install Frida on your system
 # pip install frida frida-tools (Python bindings)
Download the Frida server binary (https://github.com/frida/frida/releases) regarding your architecture:
 # adb shell getprop ro.product.cpu.abi
Upload and execute the Frida server binary
 # 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>

Version 0.1 • Updated: 2020-01