OWASP German Chapter
Stammtisch Initiative/Ruhrpott

Android App Pentest Workshop 101
About

• What we will try to cover in the first session:
  – Setup of a Mobile Application Pentest Environment
  – Basics of Mobile Application Pentests
  – Common issues in Mobile Applications’

• What we try to cover in the second session:
  – Advanced Mobile Application Pentesting
    • Removing smali code
    • Adding smali code
Setup

• You will need the following:
  – A laptop or any hardware that can run a VM
  – VM: Ubuntu 16.10 Yakkety(64bit).vdi
  – Android VM: Android-x86.5.1 rc1.vdi
  – Virtualbox (recommended)
  – Slides: https://docs.google.com/presentation/d/1owwDCtehvEZ4trdKoE7zPFeX52tNhhKOhsypr6Zcow8/edit?usp=sharing (goo.gl/e2rlzy)
  – Internet connection to google up things
VM Configuration - Network

Android WorkShop Pentest - Settings

Network

Adapter 1  Adapter 2  Adapter 3  Adapter 4

- Enable Network Adapter

Attached to: NAT

Name:

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Deny

MAC Address: 08002721B7BF

- Cable Connected
- Port Forwarding

Cancel  OK
Virtualbox - Internal Network

• You might need to run the following on your host machine:

Android Internals

- Various versions by vendors
- 3rd party markets, self-install apps
- Sandboxing
- ASLR, DEP, Stack Canaries
- On demand permission model
- Security services
  - Keystore, Fingerprint, Smartlock
  - Device and storage encryption
SANBOXING

Access to
- App data
- Restricted filesystem
- Temp
- Security Services

No access to
- Other users’ data
- Other apps’ data
- System files
- Hardware

App 1
- App Data
- User Data
- Binary / VM
- Security Services
- Cloud / Content

App 2
- App Data
- User Data
- Binary / VM
- Security Services
- Cloud / Content

Mobile OS
- System Files
- Hardware

OWASP
Open Web Application Security Project
Inter-APP Communication

- Intents
  - ACTIVITY
  - SERVICE
  - BROADCAST
- Content providers
Virtual Machines

- **Android Runtime (ART)**
  - Replaced Dalvik VM
  - Apps have codes for both

- **Xamarin Studio**
  - Mono based VM for C#
  - Runs on iOS, Android & Win

- **Apache Cordova**
  - Framework for HTML & JS
  - Runs on iOS, Android & Win
OWASP Top 10 Mobile Risks

1. Weak Server Side Controls
2. Insecure Data Storage
3. Insufficient Transport Layer Protection
4. Unintended Data Leakage
5. Poor Authorization and Authentication
6. Broken Cryptography
7. Client Side Injection
8. Security Decisions Via Untrusted Inputs
9. Improper Session Handling
10. Lack of Binary Protections
OWASP – Threat Model for Mobile
Common tools for android

- Latest Android SDK
  - Compilers and debugging tools
  - Viewers and analysers
  - Android virtual devices
- Androguard Assessment Tool
  - Anthony Desnos
  - https://github.com/androguard
- Drozer by MWR Labs
  - https://labs.mwrinfosecurity.com/tools/drozer
- Androbugs by Yu-Cheng
  - http://www.androbugs.com
Further OWASP recommended Tools
The Workshop VM

• Your VM comes with several pre-installed tools such as:
  – Android Studio
  – apktool
  – dex2-jar
  – JD-GUI
  – Jarsigner
  – drozer
  – and others
The Mobile Application

- GIT: https://github.com/OWASP-Ruhrpott/owasp-workshop-android-pentest
- Android
  - applicationId "ruhrpott.owasp.com.vuln_app_1"
  - compileSdkVersion 23
  - minSdkVersion 22
- 10 challenges/vulnerabilities (so far)
About Android Applications

- Android apps are compiled into .dex (Dalvik Executable) files which are then packed (archive)
- It might be possible to „reverse“ dex into Java code
  => However, you cannot recompile it back to an app (normally)

- AndroidManifest.xml: Contains information about the application such as needed permissions, needed android version and others

- /res: Resources (e.g. Images) and certain XML configurations can be placed in this directory
Task 1 – „Get used to the tools“

• Power On your VMs
• Open „OWASP Ruhrpott Workshop App“ in your Android VM and open the „Get used to the tools page“

1. Download the application from the device [adb]
2. Get the smali code [apktool]
3. Try to decompile the source code [dex2jar, JD-GUI]

Side note: run “adb connect <android IP>” first
[10 min]
## Task 1 - Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb shell</td>
<td>Android Debug Bridge (adb) is a command line tool that lets you communicate with an emulator or connected Android device. ‘shell’ is used to spawn a shell for further actions</td>
</tr>
<tr>
<td>pm list packages</td>
<td>list installed packages</td>
</tr>
<tr>
<td>adb pull /data/app/ruhrpott.owasp.com.vuln_app_1/base.apk .</td>
<td>download APK file to current folder</td>
</tr>
<tr>
<td>d2j-dex2jar.sh base.apk</td>
<td>retrieve dex files from apk</td>
</tr>
<tr>
<td>java –jar jd-gui-1.4.0.jar base-dex2jar.jar</td>
<td>graphical interface to browse source code</td>
</tr>
<tr>
<td>java –jar apktool_2.2.0.jar d base.apk</td>
<td>retrieve dex files and decode resources</td>
</tr>
</tbody>
</table>
How to proceed

• You will notice that the code is obfuscated – unfortunately this is very common and a default configuration in Android Studio
• Task: Try to identify the MainActivity Class and how „fragments“ are loaded
How to proceed

• You will notice that the code is obfuscated – unfortunately this is very common and a default configuration in Android Studio

• Task: Try to identify the MainActivity Class and how „fragments“ are loaded

• You will notice that the MainActivity Class uses fragments and that each page of the application is labelled with a number (0-X). Now you know which class belongs to which page

Side Note: Feel free to have a look at the source code (https://github.com/OWASP-Ruhrpott/owasp-workshop-android-pentest)
Task 2 – „Hidden Things“

- Strings are not always referenced in a class
- Your task is to identify the difference between dex2-jar and apktool
Task 2 – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>java –jar apktool_2.2.0.jar d base.apk</td>
<td>retrieve dex files and decode resources</td>
</tr>
<tr>
<td>Open values.xml in AppFolder/res/values/strings.xml</td>
<td></td>
</tr>
<tr>
<td>Look for “superhiddenstring”</td>
<td></td>
</tr>
</tbody>
</table>

- Apps used this technique as part of obfuscation and/or to hide encryption keys
Task 4 – „Logcat Output“

“Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the Log class. This page is about the command-line logcat tool, but you can also view log messages from the Logcat window in Android Studio.”

• Developer often use this feature to retrieve debug output
  – Side Note: Sometimes you can activate the „debug“ privilege within the AndroidManifest.xml to retrieve logcat messages, as this is just deactivated in production releases

• Your task: Get familiar with logcat and use its filter feature to find the „specific“ logcat message

[5 min]
## Task 4 – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb logcat -s &quot;owasp-key&quot;</td>
<td>filters for logcat messages with the tagname &quot;owasp-key&quot;</td>
</tr>
</tbody>
</table>
Task 3 – „Basic HTTP Request“

- Please change the network settings of the Android VM to 1 active adapter (NAT)

[5 min]
# Task 3 – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Android proxy to burp</td>
<td></td>
</tr>
<tr>
<td>Sniff traffic via burp</td>
<td></td>
</tr>
</tbody>
</table>
Task 5 – „Basic HTTPS Request“

• Please change the network settings of the Android VM to 1 active adapter (NAT)

[5 min]
## Task 5 – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Android proxy to burp</td>
<td></td>
</tr>
<tr>
<td>Install burp root CA</td>
<td></td>
</tr>
<tr>
<td>Sniff traffic via burp</td>
<td></td>
</tr>
</tbody>
</table>
Undo Network configuration
Android - Intents

“An intent is an abstract description of an operation to be performed. It can be used with `startActivity` to launch an `Activity`, `broadcastIntent` to send it to any interested `BroadcastReceiver` components, and `startService(Intent)` or `bindService(Intent, ServiceConnection, int)` to communicate with a background `Service`. An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.”
drozer allows you to search for security vulnerabilities in apps and devices by assuming the role of an app and interacting with the Dalvik VM, other apps' IPC endpoints and the underlying OS.

drozer provides tools to help you use, share and understand public Android exploits. It helps you to deploy a drozer Agent to a device through exploitation or social engineering. Using weasel (MWR's advanced exploitation payload) drozer is able to maximise the permissions available to it by installing a full agent, injecting a limited agent into a running process, or connecting a reverse shell to act as a Remote Access Tool (RAT).

• [https://github.com/mwrlabs/drozer](https://github.com/mwrlabs/drozer)
Task 6 – „Authorised Area“

- You will be asked for a password on this page
- Task: Circumvent the password check in order to view the „authorised“ Area of the application

- Side Note: There are several ways to solve this – In this case you should try to use drozer

[10 mins]
## Task 6 – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute drozer.apk</td>
<td>Starts drozer Agent</td>
</tr>
<tr>
<td>adb forward tcp:31415 tcp:31415</td>
<td>Forwards tcp traffic between emulator/device and your system</td>
</tr>
<tr>
<td>drozer console connect</td>
<td>Connect to drozer interface</td>
</tr>
<tr>
<td>run app.package.list</td>
<td>List all installed packages</td>
</tr>
<tr>
<td>run app.package.info –ruhrpott.owasp.com.vuln_app_1</td>
<td>General Information about the app</td>
</tr>
<tr>
<td>run app.package.manifest ruhrpott.owasp.com.vuln_app_1</td>
<td>Leaks manifest and available intents</td>
</tr>
<tr>
<td>adb shell</td>
<td></td>
</tr>
<tr>
<td>am start –a „ruhrpott.owasp.com.vuln_app_1.loggeddin“ –t „text/plain“</td>
<td></td>
</tr>
</tbody>
</table>
Task 7 – „Auth Brute“

- You have probably noticed in the last task that another intent is exposed by the application
- Task: Brute force the login via the intent
## Task 7 – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>adb shell</td>
<td>start shell</td>
</tr>
<tr>
<td><code>am start -a &quot;ruhrpott.owasp.com.vuln_app_1.auth&quot; -e &quot;x&quot; &quot;91337&quot; -t &quot;text/plain&quot;</code></td>
<td>Launch intent with extras (parameter)</td>
</tr>
</tbody>
</table>
END OF SESSION 1