Mobile App Security
Mobile Devices and Security

Part of the materials from Manuel Egele, Adam C. Champion, Dong Xuan, Michael Ernst, Amir Houmansadr
Mobile Devices are Ubiquitous

- More than 2 billion Android users
- More than 1 billion iOS users

1 Billion!

http://www.usatoday.com/story/tech/2016/05/16/android-could-top-2-billion-users/84464690/
We Spend More Time on Mobile Devices

Internet Usage (Engagement) Growth Solid
+11% Y/Y = Mobile @ 3 Hours / Day per User vs. <1 Five Years Ago, USA

Time Spent per Adult User per Day with Digital Media, USA, 2008 – 2015YTD

KPCB mobile technology trends
Significant App Usage

130,000,000,000

App downloads
How is it different from PC security?

- **Limited computing resources**
  - Battery, CPU, memory, storage, bandwidth
  - Therefore, we can not use PC solutions right out of the box

- **User education**
  - Broader range of users compared to PCs
  - More prone to social attacks

- **Ubiquitous**
  - Less expensive, outnumber PCs

- **More interfaces for connectivity**
  - Bluetooth, infrared, WiFi, cellular, USB, tethering, NFC
How is it different from PC security?

- **Portable**
  - Subject to loss, search and seizure
  - Subject to short range breaches (WiFi, near field communication)

- **Carry sensitive content**
  - Your location
  - Your identity (MAC, Bluetooth address, IMEI, IMSI)
  - Payment apps, financial apps, credit cards

- **Wider range of software (apps) than PCs**
  - “There’s an app for it”
  - Untrusted third-party vendors
Are All Apps Good?
Mobile App Analysis

Configuration

Application

App analysis

No security risk

Security risk detected

Alert + Report

1. Configuration
2. Application
3. App analysis
4. No security risk
5. Security risk detected
6. Alert + Report
Dynamic vs. Static Analysis

Static Analysis

- Examine program without execution
- Build a model of program state
  - An abstraction of the run-time state
- Reason over possible behaviors
  - E.g., “run” the program over the abstract state

- Hard to define the “abstraction”
- Conservative: high false positives
- Fast to run, good coverage 😊
Dynamic Analysis

- Execute program (over some inputs)
  - The compiler provides the semantics
- Observe executions
  - Requires instrumentation infrastructure
- Must choose what to measure, and what test runs

  - Dependent to the testing cases
  - Slow to run, limited coverage
  - More precise 😊
TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones, William Enck, et. al, OSDI’10
Problem

- Apps can be installed on a smartphone from Google Play, App Store, etc.

- All of the apps that don’t come with the OS are suspicious towards information leakage

- Monitor when sensitive data is being leaked in real time from the system through third-party applications
Mobile Device Information Leakage

- Types of mobile device information sources:
  - Internal to device (e.g., GPS location, IMEI, etc.)
  - External sources (e.g., CNN, Chase Bank, etc.)

- Third-party mobile apps can leak info
  - Send out device ID (IMEI/EID), contacts, location, etc.
  - Apps can intercept info sent from a source, send to different destination

- Motives:
  - Monitor activity,
  - Advertisement, Market research
  - Identity theft
Challenges for Monitoring Privacy Info

- Resource constraints
  - Extra network traffic (cellular data plan was once expensive)
  - Battery consumption

- Third-party apps are entrusted with several types of private information

- Sensitive information can be difficult to identify
  - Even when it’s sent in clear format
  - Geo-location data is a pair of floating point numbers
Dynamic Taint Analysis

- Dynamic taint analysis is a technique that tracks information dependencies from an origin.

- Conceptual idea:
  - Taint source
  - Taint propagation
  - Taint sink

```plaintext
\[ c = \text{taint\_source()} \]
\[ \ldots \]
\[ a = b + c \]
\[ \ldots \]
\[ \text{network\_send}(a) \]
Dynamic Taint Analysis

i = get_input();
two = 2;
if(i%2 == 0){
j = i+two;
l = j;
} else {
k = two*two;
l = k;
}
jmp l;

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Taint Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Taint Analysis

```c
i = get_input();
two = 2;
if(i%2 == 0){
    j = i+two;
    l = j;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Taint Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>true</td>
</tr>
</tbody>
</table>
Dynamic Taint Analysis

```c
i = get_input();
two = 2;
if(i%2 == 0){
    j = i+two;
    l = j;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Taint Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>true</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
<td>false</td>
</tr>
</tbody>
</table>
i = get_input();
two = 2;
if(i%2 == 0){
    j = i+two;
    l = j;
} else {
    k = two*two;
    l = k;
}
jmp l;

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Taint Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>true</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
<td>false</td>
</tr>
<tr>
<td>j</td>
<td>8</td>
<td>true</td>
</tr>
</tbody>
</table>
Dynamic Taint Analysis

```c
i = get_input();
two = 2;
if(i%2 == 0){
    j = i+two;
    l = j;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Taint Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>true</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
<td>false</td>
</tr>
<tr>
<td>j</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>l</td>
<td>8</td>
<td>true</td>
</tr>
</tbody>
</table>
Architecture of Android System
Android background

- Android is a Linux-based OS
- All of the core functionality has been written in Java and C/C++
- Applications are written in Java and then are converted into Dalvik Executable (DEX) byte code
- DEX code is executed in Dalvik Virtual Machine (DVM)
- Applications communicate via binder IPC interface
Android background

- Binder IPC (Inter-Process Communication)
  - Android apps communicate with each other using IPC binder
  - "Parcels" are fundamental components of IPC framework which serialize data objects before sending it out of VM
  - Binder kernel module passes parcel messages between processes
Android background

- Permissions
- Some apps don’t request for permissions to perform some action, they delegate their job to other apps

[applications.androidxiphone.com]
Android background

- Each app on an android phone is run inside new Dalvik VM sandbox

- Each DVM is assigned a unique user id (uid)

- All the permissions requested by an app to access phone resources are assigned to uid

- Uid remains same when an app is run/updated but pid can be different
Approach to Taint Tracking

System-wide, multiple-marking, taint tracking design

- **Variable-level**: VM interpreter provide variable-level tracking within untrusted app code.
- **Method-level**: System-provide native libraries.
- **Message-level**: Message-level tracking between applications.
- **File-level**: persistent information conservatively retains its taint markings.
TaintDroid architecture within Android.
Taint Adaptors

- Taint sources and sinks must be carefully integrated into existing architectural framework

- Depends on type of information
  - Low-bandwidth sensors: location, accelerometer
  - High-bandwidth sensors: microphone, camera
  - Information databases: address book, SMS data
  - Device identifiers: IMER, IMSI, ICC-ID
  - Network taint sink
Application Study

- Selected 30 applications with bias on popularity and access to Internet, location, microphone, and camera.
- Out of 105 flagged connections, only 37 legitimate
Findings - Phone Identifiers

applications sent device (IMEI) and 2 apps sent phone info (Ph. #, IMSI*, ICC-ID) to a remote server without informing the user.

- One application transmits the phone information every time the phone boots.

- One application displays a privacy statement that clearly indicates that the application collects the device ID

- Other uses the hash of the IMEI instead of the number itself.
Findings - Location

- 15 of the 30 applications shared physical location with an ad server (admob.com, ad.qwapi.com, ads.mobclix.com, data.flurry.com)
- Exposure of location information occurred both in plaintext (11) and in binary format (4).
Limitations

- It is a firmware modification, not an app

- Only tracks explicit data flows
Security Risks in Inter-App Communication

Analyzing Inter-Application Communication in Android, Erika Chin, et. al, Mobisys’11  [http://www.comdroid.org/]
Example

- Open Maps App From Yelp, passing a GPS coordinate
Inter-App Communications

- Sandboxes application
  - “mutually distrusting principals”
  - Default access to only its own data
- Message passing between “components”

Threats:

- If sender does not correctly specify recipient
  - Attacker can intercept the message
- If receiver does not restrict from whom it receives messages
  - Attacker can inject code
Dangerous Examples

- **SMS Message Spy Pro**
  - User facing activity functions as a tip-calculator
  - Collects all SMS messages and sends them to third-party

- **MobiStealth**
  - Collects SMS messages, call, browser, and GPS histories and sends them to third-party
  - [http://www.mobistealth.com/](http://www.mobistealth.com/)
Inter-App Communication on Android

- Android message passing system can become an attack surface

- Intent
  - Intents can be used for both intra- and inter-application communication

- ComDroid
  - A tool analyzes Android apps to detect potential vulnerabilities
  - Personal data loss, corruption, phishing…
Application Structure

- **Activity**
  - Portions of the application’s user interface
    - Login window, registration interface, etc.

- **Service**
  - Performs background processing
    - Download a file, play music, etc.

- **Broadcast Receiver**
  - Handlers for global messages
    - Boot completed, power disconnected, etc.

- **Content Provider**
  - Manages access to structured data
    - User calendar, contacts, etc.
Android Intents

- Activities, Services, and Broadcast Receivers can send/receive Intents

- Intents (explicit & implicit) can:
  - Start Activities
  - Start, stop, and bind Services
  - Broadcast information to Broadcast Receivers

- To receive Intents, component must be declared in the manifest

- System broadcast Intents
  - Only can be sent by the OS
Two types of Intent

- **Explicit Intent**
  - Identify recipient based on the app’s package name

- **Implicit Intent**
  - Identify the recipient based on a general action that an app can handle

I am looking for "Google Maps"

I am looking for whoever can handle GPS coordinates
Explicit Intents

Yelp

To: MapActivity

Map App

Name: MapActivity

Only the specified destination receives this message
Implicit Intents

Yelp

Handles Action: VIEW

Map App

Handles Action: DISPLAYTIME

Clock App
Implicit Intents

Yelp

Implicit Intent
Action: VIEW

Handles Action: VIEW

Map App

Handles Action: VIEW

Browser App
Sending an Intent

- Intent is defined by
  - Component name, an action, a category, data, extra data
  - Certain actions that it only the system can send

- Intent filter constrains incoming Intents by
  - Action: a general operation to be performed
  - Data: specifies the type of data
  - Category: additional information about the action

No rule against multiple applications specifying the same Intent filter
Threat 1: Hijacking
Threat 2: Spoofing
There is no mechanism for apps to verify the intent sender
Attacks

- Only consider attacks on exported components
  - Consider non-exported components and exported components with a Signature (or higher) permission level to be private and not susceptible to the below attacks

- **Type 1: Intents sent to the wrong application**
  - Can leak data
  - Broadcast theft, activity hijacking, and service hijacking

- **Type 2: Receiving external Intents**
  - Data corruption, code injection
  - Malicious broadcast injection, malicious activity launch, malicious service launch
Common Developer Pattern: Unique Action Strings

IMDb App

Handles Actions:
- `willUpdateShowtimes`
- `showtimesNoLocationError`

Showtime Search

Implicit Intent Action:
- `willUpdateShowtimes`

Results UI
**Date & Location**

**Thursday, June 23**

**Current Location**

**New This Week**

**Bad Teacher (2011)**
Rated R, 1 hr 32 mins, 6.3/10
Showtimes from Century Richmond
Hilltop 16, Century San Francisco Centre
9 and XD, and 1 other...

**Cars 2 (2011)**
Rated G, 1 hr 53 mins, 6.9/10
Showtimes from AMC Bay Street 16, AMC
Bay Street 16, and Others
Common Developer Pattern: Unique Action Strings

**IMDb App**

Handles Actions: `willUpdateShowtimes`, `showtimesNoLocationError`

**Implicit Intent**
Action: `willUpdateShowtimes`

**Showtime Search**

**Results UI**
ATTACK #1: Eavesdropping

IMDb App

Showtime
Search

Implicit Intent
Action:
willUpdateShowtimes

Eavesdropping App

Handles Action:
willUpdateShowtimes,
showtimesNoLocationError

Malicious
Receiver

Sending Implicit Intents makes communication public
ATTACK #2: Intent Spoofing

Malicious Injection App

Malicious Component

Action: `showtimesNoLocationError`

IMDb App

Handles Action:
`willUpdateShowtimes`,
`showtimesNoLocationError`

Receiving Implicit Intents makes the component public
Typical case

- **IMDb Showtimes**
  - **Date & Location**
    - Thursday, June 23
  - **Current Location**
  - **New This Week**
    - **Bad Teacher (2011)**
      - Rated R, 1 hr 32 mins, 6.3/10
      - Showtimes from Century Richmond, Hilltop 16, Century San Francisco Centre, 9 and XD, and 1 other...
    - **Cars 2 (2011)**
      - Rated G, 1 hr 53 mins, 6.9/10
      - Showtimes from AMC Bay Street 16, AMC Bay Street 16 and Southern

Attack case

- **IMDb Showtimes**
  - **Date & Location**
    - Thursday, June 23
  - **Current Location**
  - **Please specify a location**
    - No showtimes were found for the selected date and location.
ATTACK #3: Man in the Middle

**IMDb App**

- Action: `willUpdateShowtimes`
- Handles Action: `willUpdateShowtimes`, `showtimesNoLocation`

**Malicious Receiver**

- Action: `showtimesNoLocationError`

**Man-in-the-Middle App**

- Action: `showtimesNoLocationError`
ATTACK #4: System Intent Spoofing

- Background – System Broadcast
  - Event notifications sent by the system
  - Some can only be sent by the system

- Receivers become accessible to all applications when listening for system broadcast
System Broadcast

- **System Notifier**
  - Action: *BootCompleted*

- **App 1**
  - Component
  - Handles Action: *BootCompleted*

- **App 2**
  - Component
  - Handles Action: *BootCompleted*

- **App 3**
  - Component
  - Handles Action: *BootCompleted*
ComDroid

- Disassemble application DEX files using Dedexer tool
- Parses the disassembled output and logs potential component and Intent vulnerabilities
  - Permission
    - Normal and Dangerous
  - Intent Analysis
    - Intents, IntentFilters, registers, sinks (e.g., sendBroadcast(), startActivity(), etc.) and components
ComDroid

- **Intent**
  - Whether it has been made explicit
  - Whether it has an action
  - Whether it has any flags set
  - Whether it has any extra data

- **Sinks**
  - Implicit or not?

- **Component Analysis**
  - Public or not?
  - With data / without data?
  - System broadcast?
Some Recommendations

- Use caution with implicit Intents and exporting Components
- Use explicit Intents to send private data
- Use explicit Intents for internal communication
- Returned results should be checked for authenticity
- Avoid exporting Components
- The same Component should not handle both internal and external Intents
- Intent filters are not security measures
OK ... Before we get to the next topic
Target Fragmentation

Slides from John Mitchell
### Android Versions

- Each version has new security features

- **Android 1.5, Cupcake**: April 27, 2009
- **Android 1.6, Donut**: September 15, 2009
- **Android 2.0-2.1, Eclair**: October 26, 2009 (initial release)
- **Android 2.2-2.2.3, Froyo**: May 20, 2010 (initial release)
- **Android 2.3-2.3.7, Gingerbread**: December 6, 2010 (initial release)
- **Android 3.0-3.2.6, Honeycomb**: February 22, 2011 (initial release)
- **Android 4.0-4.0.4, Ice Cream Sandwich**: October 18, 2011 (initial release)
- **Android 4.1-4.3.1, Jelly Bean**: July 9, 2012 (initial release)
- **Android 4.4-4.4.4, KitKat**: October 31, 2013 (initial release)
- **Android 5.0-5.1.1, Lollipop**: November 12, 2014 (initial release)
- **Android 6.0-6.0.1, Marshmallow**: October 5, 2015 (initial release)
- **Android 7.0-7.1.2, Nougat**: August 22, 2016 (initial release)
- **Android 8.0-8.1, Oreo**: August 21, 2017 (initial release)
- **Android 9.0, Pie**: August 6, 2018
Target Fragmentation

- Android apps written in the lower versions of SDK can run in the compatible mode in the new OS
  - Outdated security code invisibly permeates the app ecosystem
  - "Patched" security vulnerabilities still exist in the wild
  - "Risky by default" behavior is widespread
“If the device is running Android 6.0 or higher... [the app] must request each dangerous permission that it needs while the app is running.

- Android Developer Reference
“If the device is running Android 6.0 or higher and your app's target SDK is 6.0 or higher [the app] must request each dangerous permission that it needs while the app is running.

- Android Developer Reference
“If the [operating system version of the device] is higher than the version declared by your app’s targetSdkVersion, the system may enable compatibility behaviors to ensure that your app continues to work the way you expect.”

- Android Developer Reference
Dataset

- 1,232,696 Android Apps
- Popularity, Category, Update, and Developer metadata
- Collected between May 2012 and Dec 2015
- Broken into five datasets by collection date

Outdatedness

Android 5.0 Released
Android 5.1 Released
Android 6.0 Released
App Collected
App
Collected
Outdatedness

App
Updated
Negligent Outdatedness

Android
5.0
Released

Android
5.1
Released

App
Updated

Android
6.0
Released

App
Collected
Fragment Injection

- Malicious apps can send crafted messages to exported classes that inherit from PreferenceActivity.
- The messages are interpreted as code, allowing attackers to execute arbitrary code from the malicious app.
Fragment Injection

- Fixed in Android 4.4
- Developers implement `isValidFragment` to authorize fragments

```java
// Put this in your app
protected boolean isValidFragment(String fName){
    return MyFrag.class.getName().equals(fName);
}
```
**Fragment Injection**

Vulnerable if:

- Targets 4.3 or lower (31%)
- Some class inherits from `PreferenceActivity` (4.8%)
- That class is exported (1.1%)
- That class does not override `isValidFragment` (0.55%)

4.2% of apps vulnerable if no fix was ever implemented
Mixed Content in WebView

Mixed Content: The page at 'https://googlesamples.github.io/web-fundamentals/samples/discovery-and-distribution/avoid-mixed-content/simple-example.html' was loaded over HTTPS, but requested an insecure script 'http://googlesamples.github.io/web-fundamentals/samples/discovery-and-distribution/avoid-mixed-content/simple-example.js'. This request has been blocked; the content must be served over HTTPS.
Mixed Content in WebView

- Major web browsers block Mixed Content
- In Android 5.0, WebViews block Mixed Content by default
- Can override default with `setMixedContentMode()`
% Apps Allowing Mixed Content

Target Android Version

< 5.0

>= 5.0

% Apps

0

20

40

60

80

100