463.11.2 Android Security

Computer Security II
CS463/ECE424
University of Illinois
Reading

- **Advertising Attacks**

- **Permission Re-Delegation Attacks**

- **Update and Collusion Attacks**
Agenda

• Permission re-delegation attacks
• Update and collusion attacks
• Mobile Advertising
Privilege Escalation Attacks on Android

• Gaining elevated access to resources that are normally protected against an unauthorized application

• 3 major classes
  – Confused deputy attacks: leveraging unprotected interfaces of benign programs
    o Permission re-delegation attacks
  – Collusion attacks: malicious applications work together to achieve their goal
  – Update attacks: vulnerabilities in software update mechanisms
Permission Re-delegation Attacks

Why Could This Happen?

- App w/ permissions exposes a public interface
  - The “deputy” app may accidentally expose privileged functionality
  - Or intentionally expose it, but the attacker invokes it in a surprising context
    - Example: broadcast receivers in Android
  - Or intentionally expose it but fail to correctly reduce the invoker’s authority
    - Dynamic (programmatic) permission checks
      - checkCallingPermission(), checkCallingOrSelfPermission() etc.
Public Interfaces in Android Manifest

• Via exported tag
  – <service android:name=".WiFiService" android:exported="true" android:permission="com.app.MY_PERMISSION">

• Via intent filters
  – <receiver android:name=".WiFiBroadcastReceiver"> <intent-filter>
    <action android:name="android.intent.action.WIFI"/>
  </intent-filter> </receiver>

Component is still public if android:exported="false" AND it has an intent filter!
Prevalence of Public Interfaces

- Examine 872 apps and check their `AndroidManifest.xml`
  - 16 core system apps;
  - 756 most popular free; 100 most popular paid
- 320 of these (37%) have dangerous/signature permissions and at least one type of public component
- 9% of all apps perform **dynamic permission checks**
  - But typically to **protect content providers** and not services or broadcast receivers
  - Only 1 application in a random set w/ 50 apps does so to protect a service or broadcast receiver
- 11 of 16 system applications are at risk
Implementing the Attack

• Constructing the attack
  – Decompile the potentially vulnerable app
  – Build call graph of the app
  – Search the call graph to find paths from public entry points (sources) to protected system APIs (sinks)

• Likely to miss some viable paths
  – Cannot detect flow through callbacks

• Only construct attacks on API calls for verifiable side effects
Case Studies

• Build attacks for 5 system apps
  – Settings: phone’s primary control panel
    o Settings UI sends intent to **Settings receiver** on user’s button clicks
    o Unprivileged app can also send **Intents** to this broadcast receiver
    o Requires CHANGE_WIFI_STATE, BLUETOOTH_ADMIN, ACCESS_FINE_LOCATION permissions
  – DeskClock: time and alarm functionality
    o Public **service** that accepts directions to play alarms
    o Send **Intent** to indefinitely vibrate the phone
      o prevent phone from sleeping
    o Requires VIBRATE and WAKE_LOCK permissions
Defense: IPC Inspection

- Ideas borrowed from:
  - Stack inspection
    - When a privileged API call is made, system checks within a runtime whether the call stack includes any unprivileged application. Depends on runtime (Java vs C).
  - History-based access control (HBAC)
    - Reduces the permissions of trusted code after interactions with untrusted code. Relies on runtime mechanisms.
  - Mandatory access control (MAC)
    - Central flow control by OS enforced fixed info. flow policy
    - Apps cannot be strictly ordered in terms of integrity level

We need runtime independence and ability of reduction of privileges!
Defense: IPC Inspection

• When an app receives a message from another app, reduce the privileges of recipient to the intersection of requester’s and recipient’s permissions
  – Maintain a list of current permissions for each app
  – Build privilege reduction into system’s IPC mechanism
  – Allow apps to accept or reject messages
    o They can register a set of acceptable requesters
    o Requesters are identified based on their permissions

Agenda

• Permission re-delegation attacks
• Update and collusion attacks
• Mobile Advertising
Permission Model Revision

Install-time Permissions < version 6

Permission Types

- Normal
- Signature
- Dangerous
- SignatureOr System

[Tuncay, Güliz Seray., et al. 2018]
Permission Model Revision

Runtime Permissions

>= version 6

Permission Types

- Normal
- Signature
- Dangerous
Permission Model Revision

Permission Groups
Permission Model Revision

Permission Types
- Normal
- Signature
- Dangerous

Custom Permissions

Protect Exported App Components
Prevalence of Custom Permissions

1308

Google Play
Prevalence of Custom Permissions

1350 permissions

65% apps declare custom permissions

70% apps use custom permissions

Total # of custom permissions
Observation 1

No clear distinction between system permissions and custom permissions

[Tuncay, Güliz Seray., et al. 2018]
Observation 1

No clear distinction between system permissions and custom permissions

- declared by the system
- declared by 3rd party apps
Privilege (Permission) Escalation Attack

My_Permission

normal

Microphone Group
dangerous

My_Permission

record audio

Granted

Granted

Tuncay, Güliz Seray., et al. 2018
Observation 2

No distinction between custom permissions owners
Collusion + Confused Deputy Attack

Skype_Permission

App A

App B

signature

Collusion + Confused Deputy Attack

[Tuncay, Güliz Seray., et al. 2018]
Defense

**android**

- Decisions made by principals outside the framework’s Trusted Compute Base affect enforcement at runtime
  - → privilege escalation
- Custom permissions are claimed on a FCFS basis
  - → spoofing
- Software testing

**cusper**

- Systematically addresses the lack of separation of trust by decoupling system from custom permissions
- Provides a backward-compatible OS-level naming convention for tracking ownership of custom permissions
- Formally verified to be correct

*Tuncay, Güliz Seray., et al. 2018*
Agenda

• Permission re-delegation attacks
• Update and collusion attacks
• Mobile Advertising
Why In-app Advertising

Angry Birds on iPhone

- Paid $0.99
- Dec 2010
  - 12,000,000 downloads
  - $8,000,000 profit (total)
- ? July 2014

Angry Birds on Android

- Free
- Dec 2010
  - 8,000,000 downloads
  - $1,000,000 profit (per month)
- July 2014
  - 100,000,000 – 500,000,000 installations
## Why in-app advertising?

### Abs workout

<table>
<thead>
<tr>
<th>COST</th>
<th>DOWNLOADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1.49</td>
<td>10,000 – 50,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COST</th>
<th>DOWNLOADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>10,000,000 – 50,000,000</td>
</tr>
</tbody>
</table>
How does it work?

• App Developer registers with ad network or ad exchange
• Receives a dev id and the ad SDK
• Includes the ad library in the application
• Includes a UI element in the app’s layout
• Requests the permissions the ad network requires (Android)
Most free apps rely on it for profit

• Main app UI

* Responsible for 65%-75% of energy usage in free applications!

Ad component
AdRisk: Overview

• **Problem**: assessing security and privacy risks of third-party advertising libraries embedded into apps

• **Approach**: the authors collected 100,000 apps, identified 100 ad libraries and statically analyzed them to assess their potential risk

• **Contributions**: found that ad libraries send sensitive information to remote servers and fetch and run code dynamically

Ad Libraries Collection

- 100,000 apps from Google Play (March-May 2011)
  - Extract: permissions requested;
  - Extract: app Java class tree hierarchy
  - Candidate Set (CS) includes those apps with Internet permission; Ad Set (AS) is initially empty
    - Randomly select one app from CS and **disassemble**
    - If it contains a new ad library
      - add to AS; store its ad library class hierarchy
      - remove all apps in CS with this class hierarchy
    - Repeat until |AS| = 100
  - 100 ad libraries present in 52.1% of all apps

AdRisk

• Step 1
  – Identify dangerous APIs
  – Identify sinks

• Step 2
  – Identify possible risks
Identifying Dangerous APIs

- Analysis of
  - Android documentation
  - Android source code
- Annotate APIs with permissions they require
- ClassLoader APIs and use of java.lang.reflect package can also be dangerous
- Permissions found: 34 dangerous, 26 signature, 11 signatureOrSystem, 5 normal
Identifying possible risks (1/2)

• Dangerous behaviors
  – Can be triggered from one of many entry points
  – It is dangerous if:
    o There exists a path from an entry point to an API call that can cost the user money (e.g. sending an sms) or,
    o There exists a path from entry point to an API call that allows access to personal info AND there exists a path from that API call to a sink (e.g., sending data over the Internet)
Identifying possible risks (2/2)

If the API provides access to personal info

Dangerous API

Entry Point

Network Sink
AdRisk Output

• Potentially-feasible paths
• Use of reflection
  – Java.lang.reflect
• Dynamic code loading
  – Class Loader
• Permission probing
  – Ad networks opportunistically check for permissions
• JavaScript linkages
  – Wrap Android API’s with JS and expose it to rich-media apps
• Reading list of installed packages (apps)
Results

The distribution of ad libraries among host apps
## Results

<table>
<thead>
<tr>
<th>Ad Network</th>
<th>Included in Apps</th>
<th>Probes Permissions</th>
<th>Uses Obfuscation</th>
<th>Uses Reflection</th>
<th>Uses JavaScript</th>
<th>Read Installed Packages</th>
<th>Location Data</th>
<th>Place Phone Call</th>
<th>List Accounts</th>
<th>Read Calendar</th>
<th>Read Contact/Call Logs</th>
<th>Read Browser Bookmarks</th>
<th>Read Phone Information</th>
<th>Read Phone Number</th>
<th>Send SMS</th>
<th>Change Calendar</th>
<th>Change Contacts</th>
<th>Use Vibrator</th>
<th>ClassLoader</th>
</tr>
</thead>
<tbody>
<tr>
<td>admob/android/ads</td>
<td>27235</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>google/ads</td>
<td>16323</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>flurry</td>
<td>5152</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>google/./analytics</td>
<td>4551</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>millennialmedia</td>
<td>4228</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mobelix</td>
<td>4190</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>adwhirl</td>
<td>3915</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>qwapi</td>
<td>1745</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>youmi</td>
<td>1699</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mobfox</td>
<td>1524</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>zestadz</td>
<td>1514</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cauly</td>
<td>1249</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>inmobi</td>
<td>1229</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>wooboo</td>
<td>1183</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>admarvel</td>
<td>1101</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>smaato</td>
<td>1077</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mobclick</td>
<td>1058</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>jp/co/nobot</td>
<td>995</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>airpush</td>
<td>945</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Results

• Location and IMEI
  – Targeted advertising

• Place phone call, send text message, add event to calendar
  – Only through user interaction

• Other
  – Sosceo transmits call history through the Internet
  – Some of them upload the user’s phone number
  – WAPS uploads the list of all installed apps
  – Mobus reads through SMSs to determine the text-messaging service center they use
Results

• Categorization of ad libraries
  – Invasively collecting Personal Info
    o Usually employed by smaller ad networks; SMS, call logs, list of apps e.t.c.
  – Permissively disclosing data to running ads
    o JS wrapping of Android API (user interaction)
    o gpsStart(<callback>) (no user interaction)
  – Unsafely Fetching and Loading Dynamic Code
    o One ad network allows the host app to be remotely controlled!
Reading

• Advertising Attacks

• Permission Re-Delegation Attacks

• Update and Collusion Attacks
Firefox 86 Introduces Total Cookie Protection

Tim Huang, Johann Hofmann and Arthur Edelstein | February 23, 2021

Firefox replaces GA with a fake no-op GA (rather than outright blocking it) in strict tracking protection, to prevent websites from breaking:

"Google Analytics is being shimmed by Firefox. See bugzilla.mozilla.org/show_bug.cgi?id=1493602 for details."

Total Cookie Protection creates a separate cookie jar for each website you visit. (Illustration: Meghan Newell)

https://blog.mozilla.org/security/2021/02/23/total-cookie-protection/
Discussion Questions

• Which permission model do you prefer: Installation-Time vs Ask-On-First-Use vs something else?
• We’ve seen that most of mobile malware target Android phones? Why do you think this is happening? Is iOS more secure?
• What could a malware do on a mobile device vs a desktop machine?
• What can a malicious ad library do?
• How can malicious mobile applications disguise themselves?
Discussion Questions

• Why are we not making all app components private to protect apps from privilege escalation attacks?
• Does IPC inspection have an impact on application developers?
• What kind of apps would you be more comfortable sharing your data with? Are there any apps you are not comfortable sharing your data with?
• Other kinds of attacks on smartphones?