Announcements:
• Reaction paper was due today (and all classes)
• Feedback for reaction papers soon
• “Preference Proposal” Homework due 9/24

Learning Objectives:
• Background of mobile OSes
• Study the security fundamentals of Android OS
• Discuss privilege escalation and web attacks

Reminder: Please put away (backlit) devices at the start of class
Mobile Phone Evolution

• Basic Phone
• Phone with SIM Application Toolkit (STK)
  – Tiny programs stored on GSM SIM cards
  – Typically enable value-added features
• Feature Phones
  – Extra features on the phone firmware itself
  – Typically provided by the phone manufacturer
• Smart Phones
  – API available that enables third-party apps
Growth of Mobile OS

![Worldwide Smartphone OS Market Share (Share in Unit Shipments)](source: IDC, May 2017)
### Worldwide Device Shipments by Device Type, 2016-2019 (Millions of Units)

<table>
<thead>
<tr>
<th>Device Type</th>
<th>2016</th>
<th>2017</th>
<th>2018</th>
<th>2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional PCs (Desk-Based and Notebook)</td>
<td>220</td>
<td>204</td>
<td>193</td>
<td>187</td>
</tr>
<tr>
<td>Ultramobiles (Premium)</td>
<td>50</td>
<td>59</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>Total PC Market</td>
<td>270</td>
<td>262</td>
<td>264</td>
<td>267</td>
</tr>
<tr>
<td>Ultramobiles (Basic and Utility)</td>
<td>169</td>
<td>160</td>
<td>159</td>
<td>156</td>
</tr>
<tr>
<td>Computing Device Market</td>
<td>439</td>
<td>423</td>
<td>423</td>
<td>423</td>
</tr>
<tr>
<td>Mobile Phones</td>
<td>1,893</td>
<td>1,855</td>
<td>1,903</td>
<td>1,924</td>
</tr>
<tr>
<td>Total Device Market</td>
<td>2,332</td>
<td>2,278</td>
<td>2,326</td>
<td>2,347</td>
</tr>
</tbody>
</table>

Source: Gartner (January 2018)
PC versus Smart Phones

• Why worry specifically about mobile OS security?
  – Smart phones are computing platforms similar to desktop OS: why not use the same principles?

• PC versus Smart Phones
  – Users: Root privileges typically not given to user
  – Persistent Personal Data
    • Input is cumbersome, so credentials are frequently stored
  – Battery performance is an issue
    • Implementing some security features may drain battery
  – Network usage can be expensive
PC versus Smart Phones (cont’d)

- Unique features in Smart Phones
  - Location Data
    - GPS and Wifi-based tracking
  - Premium SMS Messages (expensive)
  - Making and recording phone calls
  - Logs of previously sent SMS
  - Different authentication mechanisms
    - Fingerprint reader (available across platform)
    - Face Unlock (Android 5.0)
    - Trusted Places, Devices, Voice (Android 5.0)
  - Specific markets for mobile apps
Mobile OS Security Frameworks

• Covered
  – Android Security Model

• Not Covered
  – Apple iOS
  – Windows OS
  – Blackberry
Android
Platform outline:

- Linux kernel
- Embedded Web Browser
- SQL-lite database
- Software for secure network communication
  - Open SSL, Bouncy Castle crypto API and Java library
- Java platform for running applications
- C language infrastructure
- Also: video APIs, Bluetooth, vibrate phone, etc.
Android Market (Google Play Store)

- Self-signed apps
- Open market
  - Not rigorously reviewed by Google (unlike Apple)
  - Bad applications may show up on market
  - Malware writers can get code onto platform: Self-signed applications are possible
    - shifts focus from remote exploit to privilege escalation
Android Application Structure

• 4 components
  – Activity – one-user task
    • E.g., scroll through your inbox
  – Service – Java daemon that runs in background
    • E.g., application that streams an mp3
  – Broadcast receiver
    • “mailboxes” for messages from other applications
  – Content provider
    • Store and share data using a relational database interface

• Activating components
  – Via Intents
• Message between components in same or different app
• Intent is a bundle of information, e.g.,
  – action to be taken
  – data to act on
  – category of component to handle the intent
  – instructions on how to launch a target activity

• Routing can be
  – Explicit: delivered only to a specific receiver
  – Implicit: all components that have registered to receive that action will get the message
Android Manifest File

- Declarations
  - Components
  - Component capabilities
    - Intent filters
    - Permissions etc.
- App requirements
  - Permissions
  - Sensors etc.

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.gulizseray.provider.infoprovender">
  <uses-permission
    android:name="android.permission.RECORD_AUDIO" />
  <uses-feature
    android:name="android.hardware.microphone"
    android:required="true"/>
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="InfoProvider"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
      android:name=".MainActivity"
      android:theme="@style/AppTheme.NoActionBar">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".InfoProviderService" android:exported="true"/>
  </application>
</manifest>
```
• Example of permissions provided by Android
  – “android.permission.INTernet”
  – “android.permission.READ_EXTERNAL_STORAGE”
  – “android.permission SEND_SMS”

• Protection levels
  – Dangerous, normal, signature

• Also possible to define custom permissions
  – To enable or disable other apps to call their features

• Used to be granted at installation but...
• Example of permissions provided by Android
  – “android.permission.INTERNET”
  – “android.permission.READ_EXTERNAL_STORAGE”
  – “android.permission.SEND_SMS”

• Protection levels
  – Dangerous
  – Normal
  – Signature

• Also possible to define custom permissions
  – To enable or disable other apps to call their features

• Used to be granted at installation but
Android Runtime Permissions

- Runtime permissions
  - Dangerous permissions granted at runtime
  - Normal and signature permissions still granted at installation
  - Only valid for modern apps (API >= 23), permissions are still install time for legacy app
  - Permissions granted based on a permission group basis
Q: Which permission model do you prefer: Installation-Time vs Ask-On-First-Use vs something else?
As a user? Complications for developers?
Isolation

• Multi-user Linux operating system
• Each application normally runs as a different user
  – Each app has its own VM
  – Traditional linux-based permissions apply (DAC)
• Applications announce permission requirement
  – Create a whitelist model – user grants access
  – ICC reference monitor checks permissions (MAC)
• Flexibility and reusability important for Android
  – Enable apps to work together to accomplish things
• Apps communicate through application framework
  – Intents based on Binder IPC
  – Implemented in kernel as a driver
Q: Heavily relying on IPC (Android) vs completely standalone apps (iOS, kind of)? Which one do you think is better?
Android malware

New Android malware samples (per year)

<table>
<thead>
<tr>
<th>Year</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>3,809</td>
</tr>
<tr>
<td>2012</td>
<td>214,327</td>
</tr>
<tr>
<td>2013</td>
<td>1,192,035</td>
</tr>
<tr>
<td>2014</td>
<td>1,548,129</td>
</tr>
<tr>
<td>2015</td>
<td>2,333,777</td>
</tr>
<tr>
<td>2016</td>
<td>3,246,284</td>
</tr>
<tr>
<td>2017</td>
<td>3,002,482</td>
</tr>
</tbody>
</table>

© DATA analysts are counting over 3 million new Android malware samples in 2017. 744,065 of these were discovered in the fourth quarter.
Android malware

850 million Android devices still at risk of being hijack by Stagefright bug

Security researchers say fragmented manner of Android operating system restricts protections.


Stagefright malware is back! 'Android bug in history' return third time and could infect a billion phones

- Stagefright bug lets attackers take control of older Android phones
- Notorious bug is back for a third time, security research firm
- Israel-based NorthBit security researchers seem to show the threat
- It only affects handsets running software older than Android being told to upgrade, and install anti-malware apps

By SARAH GRIFFITHS FOR MAILONLINE
PUBLISHED: 06:59 EDT, 21 March 2016 | UPDATED: 19:17 EDT, 21 March 20

Google Is Fighting A Massive Android Malware Outbreak -- Up To 21 Million Victims

Thomas Brewster Forbes Staff
Security
I cover crime, privacy and security in digital and physical forms.

Android has been hit by another big malware campaign, as criminals exploit Google's platform. Photographer: Chris Goodney/Bloomberg
Attacks on Android

- Privilege Escalation Attacks on Android
- Web Attacks on Android
- Phishing
- Clickjacking
- Side-channel etc.
Privilege Escalation Attacks on Android

- Gaining elevated access to resources that are normally protected from an application
- 2 major classes
  - confused deputy attacks: leveraging unprotected interfaces of benign applications
    - **Permission re-delegation attacks**
  - collusion attacks: malicious applications merge their permissions
- Other interesting ones: mobile OS update etc.
Permission Re-delegation Attacks

Wi-Fi Manager

Access Wi-Fi?

Wi-Fi Control App

confused deputy

Permission requested in advance

Access Wi-Fi?

Attack App

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, attempt to reduce the invoker’s authority, but do it incorrectly
    • Dynamic (programmatic) permission checks
      – checkCallingPermission(), checkCallingOrSelfPermission() etc.

Public Interfaces in Android Manifest

• Via exported tag
  – <service android:name=".WiFiService"
    android:exported="true"> </service>

• 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!

I have two similar applications (one free, one paid).

An activity is defined with `exported="false"

```xml
<activity>
    android:name=".MyActivity"
    android:exported="false"
    android:noHistory="true"
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="vnd.android.cursor.item/vnd.mine" />
    </intent-filter>
</activity>
```

When I call `startActivity` with the appropriate implicit intent from the free app, the activity picker appears.

I don't understand why the activity from the paid app appears, since it is `exported="false"`.

I suppose I can add an intent filter based on the URL, but my question is: why does the activity from the other app appear when the doc reads
If Your Activity Has an `<intent-filter>`, Export It

Slightly less than two years ago, I pointed out a problem in Android where an activity that has an `<intent-filter>`, but is marked as not being exported (`android:exported="false"`), screws up the chooser. The chooser ignores the exported flag and offers up the non-exported activity to the user... then promptly crashes if the user actually chooses it.

Dianne Hackborn specifically called this out as being a bug in the app:

would generally consider this a bug in the app – if you have an activity that you aren’t allowing other apps to launch, why the heck are you publishing an intent filter that they will match to try to launch?

What would be nice, of course, is if Google paid attention to its own advice.

The AOSP Music app has five activities that violate this rule:
- VideoBrowserActivity
- ArtistAlbumBrowserActivity
- AlbumBrowserActivity
Prevalence of Public Interfaces

- Examine 872 apps
- 320 of these (37%) have 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
  – Build call graph of the app
  – Search the call graph to find paths from public entry points to protected system APIs

• 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
    • Settings UI sends intent to Settings receiver on user’s button clicks
    • Unprivileged app can also send intents to this broadcast receiver
    • Requires CHANGE_WIFI_STATE, BLUETOOTH_ADMIN, ACCESS_FINE_LOCATION permissions
  – DeskClock: time and alarm functionality
    • Public service that accepts directions to play alarms
    • Send intent to indefinitely vibrate the phone (prevent phone from sleeping)
    • Requires VIBRATE and WAKE_LOCK permissions

Defense

- 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. Dependent on runtime.
  - 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!

IPC Inspection

• When an app receives a message from another app, reduce the privileges of recipient to the intersection of recipient’s and requester’s permissions
  – Implemented in the OS, not in the runtime.
  – Maintain a list of current permissions for each app
  – Build privilege reduction into system’s IPC mechanism
  – Allow apps to accept or reject messages
    • They can register a set of acceptable requesters
    • Requesters can be identified individually (domain) or based on their permissions

There can be multiple requesters

- Create new app instances for the deputy if privilege reduction is necessary
- Instance reuse
  - Primary instance can be reused in an install-time system
  - Not possible with time-of-use systems since deputy could dynamically request permissions and it isn’t clear which requester is responsible for the permission prompt

Singleton deputy apps will have their permissions repeatedly reduced until app exists
Attacks on Android

- Privilege Escalation Attacks on Android
- Web Attacks on Android
Embedded Web Browsers

• Web container for showing web pages within app context
• >90% of apps in Google Play Store use WebView
• Interesting use cases:
  – Displaying ads
  – Reuse of web code
  – Hybrid frameworks
WebView API

- **Web settings**
  - setJavaScriptEnabled()
  - setAllowFileAccess()

- **Navigation**
  - shouldOverrideUrlLoading()

- **App code access**
  - JavaScript interfaces
  - JavaScript event handlers
    - onJsAlert(), onJsPrompt(), onJsConfirm()

- **HTML5 API**
  - GeoLocation, getUserMedia
  - App should have permissions

- **Loading content**
  - loadUrl() etc.
  - “http://”, “https://”, “file://”, “javascript:”
Prevalence of WebView API

- WebView
- JavaScript Enabled
- JavaScript Interfaces
- Event Handler (prompt)
- Event Handler (alert)
- Event Handler (confirm)
- HTML5 Geolocation API
- HTML5 Media API

Percentage of apps in dataset
Web Attacks on Android

• Web to app attacks
  – Excess authorization
    • Web domains abuse app/device resources
  – File-based cross-zone scripting
    • loadUrl(“file://....html”)
    • File system access by third party domains

• App to web attacks
  – JavaScript injection
    • loadUrl(“javascript:.....”)
  – Event sniffing and hijacking
    • doUpdateVisitedHistory, onFormResubmission

App-Web Code Interaction
Threat Model for Excess Authorization

- Have iframes!
- Expected to be app’s own domain!
- Navigate to other sites!

Defense: Draco

- Fine grained and origin based access control for WebView
- Protect all parts of all three access channels for all types of apps uniformly
- 2 main components
  - Draconian Policy Language
  - Draco Runtime System

Draconian Policy Language

Policy rule example:

```
https://www.caremark.com WebViewJavascriptInterface <all> geolocation, camera decisionpoint<system>
```
Draco Architecture

Android App

Chromium library

Policy Manager

Enforcement Module

Parsing Module

Draconian policies

App developer

loadUrl ("policyrule:..."

Android System WebView

Chromium library

Static Analysis Module

 Decompiler

Static analyzer

App user

subject
class name
method list
permissions
decisionpoint

Mobile & Device Security: Looking Forward

• Where to look for literature: “Big 4” security conferences (IEEE S&P a.k.a. Oakland, USENIX Security, CCS, NDSS), WiSec, SPSM workshop (now merged with WiSec), MobiSys

• Hot topics in mobile & device security (not exhaustive):
  – Android permissions
  – Mobile advertising & third-party libraries
  – Side-channel attacks on mobile devices
  – IoT security