Mobile Device and Platform Security – Part II

John Mitchell
Guest Lecture Tuesday June 6

• **Diogo Mónica**, Director of security at Docker
• Topics include container security and what it is like working as a security engineer
• Diogo a very good speaker. Please show your appreciating and come to class.
• We will include topics on the final exam
Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture and defenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation
ANDROID

History and early decisions
Android history

- Android, Inc founded by Andy Rubin around 2005
  - Worked with HTC-built device with a physical keyboard
  - Scrapped Blackberry-like phone when iPhone came out
  - First Android phone HTC Dream, Oct 2008 (T-Mobile G1): touchscreen and keyboard

- Open-source software project
- Backed and acquired by Google
HTC Dream

• First phone had
  – Android 1.6 (Donut)
  – 3.15 megapixel rear camera with auto-focus
  – 3.2 inch touchscreen
  – Gmail, Google Maps, Search, Google Talk, YouTube, calendar, contacts, alarm
Android ecosystem

• Open-source software distributed by Google
  – Increase number of users and devices linked to core Google products

• Multiple hardware vendors
  – Can customize software for their products

• Open marketplace for apps
App market

- Self-signed apps
- App permissions
  - granted on user installation
- Open market
  - Bad apps may show up on market
  - Shifts focus from remote exploit to privilege escalation
ANDROID PLATFORM

Device locking and permissions
Device lock and unlock

- Similar PIN and fingerprint
- Fingerprint API lets users
  - Unlock device
  - Securely sign in to apps
  - Use Android Pay
  - Purchase on Play Store
Android permissions

• Example of permissions provided by Android
  – “android.permission.INTERNET”
  – “android.permission.READ_EXTERNAL_STORAGE”
  – “android.permission.SEND_SMS”
  – “android.permission.BLUETOOTH”

• Also possible to define custom permissions
Android permission model

```xml
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.NFC" />
<uses-permission android:name="android.permissionINTERNET" />
```

ANDROID PLATFORM

Platform security features
Android

• Platform outline:
  – Linux kernel, browser, SQL-lite database
  – Software for secure network communication
    • Open SSL, Bouncy Castle crypto API and Java library
  – C language infrastructure
  – Java platform for running applications
    • Dalvik bytecode, virtual machine
Exploit prevention

- Open source: public review, no obscurity
- Goals
  - Prevent remote attacks, privilege escalation
  - Secure drivers, media codecs, new and custom features
- Overflow prevention
  - ProPolice stack protection
    - First on the ARM architecture
  - Some heap overflow protections
    - Chunk consolidation in DL malloc (from OpenBSD)
- ASLR
  - Avoided in initial release
    - Many pre-linked images for performance
  - Later developed and contributed by Bojinov, Boneh
dlmalloc (Doug Lea)

• Stores meta data in band
• Heap consolidation attack
  – Heap overflow can overwrite pointers to previous and next unconsolidated chunks
  – Overwriting these pointers allows remote code execution
• Change to improve security
  – Check integrity of forward and backward pointers
    • Simply check that back-forward-back = back, f-b-f=f
  – Increases the difficulty of heap overflow
ANDROID PLATFORM

App execution environment
Application development concepts

• Activity – one-user task
  – Example: scroll through your inbox
  – Email client comprises many activities
• Service – Java daemon that runs in background
  – Example: application that streams an mp3 in background
• Intents – asynchronous messaging system
  – Fire an intent to switch from one activity to another
  – Example: email app has inbox, compose activity, viewer activity
    • User click on inbox entry fires an intent to the viewer activity, which then allows user to view that email
• Content provider
  – Store and share data using a relational database interface
• Broadcast receiver
  – “mailboxes” for messages from other applications
Security Features

• Isolation
  – Multi-user Linux operating system
  – Each application normally runs as a different user

• Communication between applications
  – May share same Linux user ID
    • Access files from each other
    • May share same Linux process and Dalvik VM
  – Communicate through application framework
    • “Intents,” based on Binder, discussed in a few slides

• Battery life
  – Developers must conserve power
  – Applications store state so they can be stopped (to save power) and restarted – helps with DoS
Application sandbox

• Application sandbox
  – Each application runs with its UID in its own Dalvik virtual machine
    • Provides CPU protection, memory protection
    • Authenticated communication protection using Unix domain sockets
    • Only ping, zygote (spawn another process) run as root
  – Applications announce permission requirement
    • Create a whitelist model – user grants access
      – Don’t interrupt user – all questions asked as install time
    • Inter-component communication reference monitor checks permissions
Android permission model

Android Intents

• Msg 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
• **Layers of security**
  – Each application executes as its own user identity
  – Android middleware has reference monitor that mediates the establishment of inter-component communication (ICC)

Source: Penn State group Android security paper
MAC Policy Enforcement in Android. This is how applications access components of other applications via the reference monitor. Component A can access components B and C if permission labels of application 1 are equal or dominate labels of application 2.
Security issues with intents

• Sender of an intent can verify that the recipient has a permission by specifying a permission with the method call
• Senders can use explicit intents to send the message to a single component (avoiding broadcasting)
• Receivers have to handle malicious intents
Attack: Permission redelegation

• Definition: an application without a permission gains additional privileges through another application
• Example of the “confused deputy” problem
Permission redelegation

WiFi Manager (Strict Sheriff) -> Wifi Control App
Access Wifi
Requested permission during install

Access Wifi
Attacker App
No permissions during install

Permission redelegation

WifiManager (Strict Sheriff) → WifiControlApp

WifiControlApp granted AttackerApp permission without checking

AttackerApp → WifiControlApp

Confused Deputy

How could this happen?

- App w/ permissions exposes a public interface
- Study in 2011
  - Examine 872 apps
  - 320 of these (37%) have permissions and at least one type of public component
  - Construct attacks using 15 vulnerabilities in 5 apps
- Reference
  - Permission Re-Delegation: Attacks and Defenses, Adrienne Felt, Helen Wang, Alexander Moshchuk, Steven Hanna, Erika Chin, Usenix 2011
Example: power control widget

• Default widgets provided by Android, present on all devices

• Can change Wi-fi, BT, GPS, Data Sync, Screen Brightness with only one click

• Uses Intent to communicate the event of switching settings

• A malicious app without permissions can send a fake Intent to the Power Control Widget, simulating click to switch settings
### Vulnerable versions (in red)

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6</td>
<td>Donut</td>
<td>4</td>
<td>0.10%</td>
</tr>
<tr>
<td>2.1</td>
<td>Eclair</td>
<td>7</td>
<td>1.50%</td>
</tr>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>8</td>
<td>3.20%</td>
</tr>
<tr>
<td>2.3 - 2.3.2</td>
<td>Gingerbread</td>
<td>9</td>
<td>0.10%</td>
</tr>
<tr>
<td>2.3.3 - 2.3.7</td>
<td>Gingerbread</td>
<td>10</td>
<td>36.40%</td>
</tr>
<tr>
<td>3.2</td>
<td>Honeycomb</td>
<td>13</td>
<td>0.10%</td>
</tr>
<tr>
<td>4.0.3 - 4.0.4</td>
<td>Ice Cream Sand</td>
<td>15</td>
<td>25.60%</td>
</tr>
<tr>
<td>4.1.x</td>
<td></td>
<td>16</td>
<td>29.00%</td>
</tr>
<tr>
<td>4.2.x</td>
<td>Jelly Bean</td>
<td>17</td>
<td>4.00%</td>
</tr>
</tbody>
</table>

- Principle of least privilege helps but is not a solution
- Apps with permissions need to manage security

Java Sandbox

• Four complementary mechanisms
  – Class loader
    • Separate namespaces for separate class loaders
    • Associates *protection domain* with each class
  – Verifier and JVM run-time tests
    • NO unchecked casts or other type errors, NO array overflow
    • Preserves private, protected visibility levels
  – Security Manager
    • Called by library functions to decide if request is allowed
    • Uses protection domain associated with code, user policy
Stack Inspection

- Permission depends on
  - Permission of calling method
  - Permission of all methods above it on stack
    - Up to method that is trusted and asserts this trust

Stories: Netscape font / passwd bug; Shockwave plug-in

Many details omitted here
ANDROID MALWARE
Android malware example

22 March 2013  World Uyghur Congress

In what was an unprecedented coming-together of leading Uyghur, Mongolian, Tibetan and Chinese activists, as well as other leading international experts, we were greatly humbled by the great enthusiasm, contribution and desire from all in attendance to make this occasion something meaningful, the outcome of which produced some concrete, action-orientated solutions to our shared grievances. The attachment is a letter on behalf of WUC, UNPO and STP.

Install malicious “conference app”

WUC’s Conference in Geneva

On behalf of all at the Word Uyghur Congress (WUC), the Unrepresented Nations and Peoples Organization (UNPO) and the Society for Threatened Peoples (STP), Human Rights in China: Implications for East Turkestan, Tibet and Southern Mongolia In what was an unprecedented
Malware behavior triggered by C&C server (Chuli)

WUC’s Conference in Geneva

On behalf of all at the Word Uyghur Congress (WUC), the Unrepresented Nations and Peoples Organization (UNPO) and the Society for Threatened Peoples (STP), Human Rights in China: Implications for East Turkestan, Tibet and Southern Mongolia. In what was an unprecedented
Chuli source-to-sink flows
ANDROID WEB APPS
A Large-Scale Study of Mobile Web App Security

Patrick Mutchler, Adam Doupe, John Mitchell, Chris Kruegel, Giovanni Vigna
Mobile Apps
Mobile Apps
Mobile Apps
Mobile web app: embeds a fully functional web browser as a UI element
Obj foo = new Object();
addJavascriptInterface(foo, 'f');
JavaScript Bridge

f.bar();
Why?

• Full-featured mobile web apps
• Expose phone functionality to JavaScript
Security Concerns

• Who can access the bridge?
  – Everyone
46 U.S. CRUISE MISSILES 'ONE OR TWO' KEY KHORASAN KILLED 'A DOZEN' CIVILIANS DEAD
No origin distinction in WebView

f.bar();
Static Analysis

• How many mobile web apps?
• How many use JavaScript Bridge?
• How many vulnerable?
Experimental Results

- 737,828 free apps from Google Play (Oct ’13)
- 563,109 apps embed a browser
- 219,404 use the JavaScript Bridge
- 107,974 have at least one security violation
Most significant vulnerabilities

1. Loading untrusted web content

2. Leaking URLs to foreign apps

3. Exposing state changing navigation to foreign apps
1. Loading untrusted web content

2. Leaking URLs to foreign apps

3. Exposing state changing navigation to foreign apps
“You should restrict the web-pages that can load inside your WebView with a whitelist.”

- Facebook
“…only loading content from trusted sources into WebView will help protect users.”

- Adrian Ludwig, Google
1. Navigate to untrusted content
// In app code
myWebView.loadUrl("foo.com");
// In app code
myWebView.load("foo.com");

<!-- In HTML -->
<a href="foo.com">click!</a>
// In app code
myWebView.load("foo.com");

<!-- In HTML -->
<a href="foo.com">click!</a>

<!-- More HTML -->
<iframe src="foo.com"/>
// In app code
myWebView.loadUrl("foo.com");

<!-- In HTML -->
<a href="foo.com">click!</a>

<!-- More HTML -->
<iframe src="foo.com"/>

// In JavaScript
window.location = "foo.com";
public boolean shouldOverrideUrlLoading(WebView view, String url) {

    // False -> Load URL in WebView
    // True  -> Prevent the URL load

}
public boolean shouldOverrideUrlLoading(
    WebView view, String url) {

    String host = new URL(url).getHost();
    if (host.equals("stanford.edu")) {
        return false;
    }
    log("Overrode URL: " + url);
    return true;
}
Reach Untrusted Content?

- 40,084 apps with full URLs and use JavaScript Bridge
- 13,683 apps (34%) can reach untrusted content
Use HTTPS?

- 152,706 apps with partially computed URLs
- 87,968 apps (57%) with HTTP URLs
Handling SSL Errors

onReceivedSslError

1. handler.proceed()
2. handler.cancel()
3. view.loadUrl(...
Mishandling SSL Errors

• 117,974 apps implement `onReceivedSslError`
• 29,652 apps (25%) **must** ignore errors

- 56% Vulnerable on all devices
- 37% No Bridge Object
- 8% Vulnerable on some devices
## Primary results

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>% Relevant</th>
<th>% Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsafe Nav</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>HTTP</td>
<td>40</td>
<td>56</td>
</tr>
<tr>
<td>Unsafe HTTPS</td>
<td>27</td>
<td>29</td>
</tr>
</tbody>
</table>
Popularity

Download Count (thousands)

- Blue: <1
- Red: 100-1,000
- Orange: 1-10
- Purple: >1,000
- Green: 10-100

Normalized Vulnerability Rate

Unsafe Nav

HTTP

Unsafe HTTPS
Outdated Apps

- Unsafe Nav
- HTTP
- Unsafe HTTPS
Libraries

- 29% unsafe nav
- 51% HTTP
- 53% unsafe HTTPS
Additional security issues

Based on 998,286 free web apps from June 2014

<table>
<thead>
<tr>
<th>Mobile Web App Feature</th>
<th>% Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript Enabled</td>
<td>97</td>
</tr>
<tr>
<td>JavaScript Bridge</td>
<td>36</td>
</tr>
<tr>
<td>shouldOverrideUrlLoading</td>
<td>94</td>
</tr>
<tr>
<td>shouldInterceptRequest</td>
<td>47</td>
</tr>
<tr>
<td>onReceivedSslError</td>
<td>27</td>
</tr>
<tr>
<td>postUrl</td>
<td>2</td>
</tr>
<tr>
<td>Custom URL Patterns</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vuln</th>
<th>% Relevant</th>
<th>% Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsafe Navigation</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>Unsafe Retrieval</td>
<td>40</td>
<td>56</td>
</tr>
<tr>
<td>Unsafe SSL</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>Exposed POST</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Leaky URL</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>
Takeaways

• Apps must not load untrusted content into WebViews

• Able to identify violating apps using static analysis

• Vulnerabilities are present in the entire app ecosystem
ANDROID VERSIONING
Target Fragmentation in Android Apps

Patrick Mutchler
John Mitchell

Yeganeh Safaei
Adam Doupe
Android apps can run using outdated OS behavior
  - The large majority of Android apps do this
  - Including popular and well maintained apps

Outdated security code invisibly permeates the app ecosystem
  - “Patched” security vulnerabilities still exist in the wild
  - “Risky by default” behavior is widespread
Roadmap

What is target fragmentation?
Target fragmentation statistics
Security consequences
Roadmap

What is target fragmentation?

Target fragmentation statistics

Security consequences
“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
Roadmap

What is target fragmentation?

Target fragmentation statistics

Security consequences
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
Negligent Outdatedness

Outdatedness

Android 5.0 Released
Android 5.1 Released
App Updated
Android 6.0 Released
App Collected
Roadmap

What is target fragmentation?

Target fragmentation statistics

Security consequences
A malicious application can invoke any exported `PreferenceActivity` class and supply it with an `:android:show_fragment` Intent extra in order to make it load an arbitrary class.
Fragment Injection

Fixed in Android 4.4

Developers implement isValidFragment to authorize fragments

// 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()`
**SOP for file:// URLs in WebView**

Android 4.1 separate file:// URLs are treated as unique origins

Can override with `setAllowFileAccessFromFileURLs()`
% Apps Using Unsafe file:// SOP

Target API Version

<16

>=16
Summary of Target Fragmentation

Android apps can run using outdated OS behavior
- The large majority of Android apps do this
- Including popular and well maintained apps

Outdated security code invisibly permeates the app ecosystem
- “Patched” security vulnerabilities still exist in the wild
- “Risky by default” behavior is widespread
Two lectures on mobile security

• Introduction: platforms and trends
• Threat categories
  – Physical, platform malware, malicious apps
• Defense against physical theft
• Malware threats
• System architecture and defenses
  – Apple iOS security features and app security model
  – Android security features and app security model
• Security app development
  – WebView – secure app and web interface dev
  – Device fragmentation
Comparison: iOS vs Android

• App approval process
  – Android apps from open app store
  – iOS vendor-controlled store of vetted apps

• Application permissions
  – Android permission based on install-time manifest
  – All iOS apps have same set of “sandbox” privileges

• App programming language
  – Android apps written in Java; no buffer overflow...
  – iOS apps written in Objective-C
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>iOS</th>
<th>Android</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unix</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open market</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Closed market</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vendor signed</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self-signed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User approval of permissions</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Managed code</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Native code</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>iOS</th>
<th>Android</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unix</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Open market</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Closed market</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Vendor signed</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self-signed</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>User approval of permissions</td>
<td>x</td>
<td>7-&gt; 8</td>
<td></td>
</tr>
<tr>
<td>Managed code</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Native code</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture and defenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation