Security of Mobile Applications

Vitaly Shmatikov
Structure of Android Applications

- This is a very brief and incomplete summary
  - See Enck et al. “Understanding Android Security”

- Applications include multiple components
  - Activities: user interface
  - Services: background processing
  - Content providers: data storage
  - Broadcast receivers for messages from other apps

- **Intent:** primary messaging mechanism for interaction between components
Explicit Intents

Yelp

To: MapActivity

Name: MapActivity

Only the specified destination receives this message
Implicit Intents

- Yelp
  - Implicit Intent
  - Action: VIEW

- Map App
  - Handles Action: VIEW

- Browser App
  - Handles Action: VIEW
Android Security Model

- Based on permission labels assigned to applications and components
- Every app runs as a separate user
  - Underlying Unix OS provides system-level isolation
- Reference monitor in Android middleware mediates inter-component communication

Access permitted if labels assigned to the invoked component are in the collection of invoking component
Mandatory Access Control

- Permission labels are set (via manifest) when app is installed and cannot be changed.
- Permission labels only restrict access to components, they do not control information flow – means what?
- Apps may contain “private” components that should never be accessed by another app (example?)

- If a public component doesn’t have explicit permissions listed, it can be accessed by any app.
System API Access

- System functionality (e.g., camera, networking) is accessed via Android API, not system components
- App must declare the corresponding permission label in its manifest + user must approve at the time of app installation
- Signature permissions are used to restrict access only to certain developers
  - Ex: Only Google apps can directly use telephony API
Refinements

◆ Permission labels on broadcast intents
  • Prevents unauthorized apps from receiving these intents – why is this important?

◆ Pending intents
  • Instead of directly performing an action via intent, create an object that can be passed to another app, thus enabling it to execute the action
  • Invocation involves RPC to the original app
  • Introduces delegation into Android’s MAC system
Unique Action Strings

Common developer pattern

IMDb App

Handles Actions:
willUpdateShowtimes,
showtimesNoLocationError

Showtime Search

Implicit Intent Action: willUpdateShowtimes

Results UI
Eavesdropping


**IMDb App**

- Showtime Search
- Implicit Intent Action: `willUpdateShowtimes`

**Eavesdropping App**

- Handles Action: `willUpdateShowtimes`, `showtimesNoLocationError`
- Malicious Receiver
Intent Spoofing

[Felt et al.]

**Malicious Injection App**

Malicious Component

Action: showtimesNoLocationError

**IMDb App**

Handles Action:
willUpdateShowtimes, showtimesNoLocationError

Results UI

Also man-in-the-middle
System Broadcast

Event notifications broadcast by the system (can’t be spoofed)

System Notifier
Action: BootCompleted

Component
App 1
Handles Action: BootCompleted

Component
App 2
Handles Action: BootCompleted

Component
App 3

Broadcast receivers make components publicly accessible

[Felt et al.]
Exploiting Broadcast Receivers

[Felt et al.]

Malicious App

To: App1.Component

Malicious Component

App 1
Handles Action: BootCompleted

Component
Real World Example: ICE

[Felt et al.]

Allows doctors access to medical information on phones

Contains a component that listens for the BootCompleted system broadcast

On receipt of this intent, exits the app and locks the screen
Permissions: Not Just Android

All mobile OSes, HTML5 apps, browser extensions...
Permission Re-Delegation

- An application with a permission performs a privileged task on behalf of an application without permission.

[Permission Re-Delegation: Attacks and Defenses. USENIX Security 2011]
Examples of Re-Delegation

- Permission re-delegation is an example of a “confused deputy” problem
- 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 and attempt to reduce the invoker’s authority, but do it incorrectly
  - Remember postMessage origin checks?
Mobile Apps in Web Languages

Woulda been nice if...

- Platform-independent
- Portable
- Reuse old Web apps
- Low maintenance

Hybrid App Model

Over 400,000 developers worldwide
Hybrid App Development

The World Of Hybrid Frameworks

WebView

Same Origin Policy

Java
C/C++/C#
Objective-C

appcelerator
appMobi
APPSPRESSO
COCOON
Sencha

bridge

bridge

OS Access Control Policy
WebView

◆ Embedded browser in smartphone apps
◆ Basic same origin policy inside the browser + holes in the browser sandbox allowing Web code to invoke native functionality
  • Camera, contacts, file system, etc.
◆ Multiple “bridges” between Web and local code
  • JavaScript interfaces to local objects
  • Interception of browser events (eg, special URLs)
  • Other custom and ad-hoc schemes

Invoking Java from JavaScript

[wv.addJavascriptInterface(new FileUtils(), "FUtil");
wv.addJavascriptInterface(new ContactManager(), "GC");

// The FileUtils class has the following methods:
public int write (String filename, String data, boolean append);
public String read (filename);

// The ContactManager class has the following methods:
public void searchPeople (String name, String number);
public ContactTriplet getContactData (String id);
Invoking JavaScript from Java

```java
String str = "<div><h2>Hello World</h2></div>";
webView.loadUrl("javascript:document.appendChild(str);");
webView.loadUrl("javascript:document.cookie='';");
```
The Hybrid Security Model
Attacks from Malicious App

JavaScript injection
Event sniffing and hijacking

[Luo et al.]
Attack from Malicious Web Content

[Luo et al.]
Frame Confusion

What is the origin of this JavaScript object?
It Gets Worse

Java Reflection API... accessible from Web side

[Android Java code]

[Luo et al.]
Simple Fixes Don’t Work

Most hybrid frameworks don’t even attempt to verify whether access request comes from an authorized Web origin

PhoneGap attempts to filter based on developer-provided whitelist

- Mediation either incomplete (does not catch iframe loads) or too strict (prohibits even loading of content from other origins, breaks look-and-feel)
- Incorrect origin checks
  - Broken regexes bite again – anchoring bugs, etc.

State of the Union

◆ Convergence of Web and mobile programming
◆ Complex, poorly understood software stacks with badly fitting security policies
◆ New classes of vulnerabilities
  • Worst case: Web advertiser gets to inject arbitrary code into mobile apps running on your phone!%
◆ Evolving defenses
  • Our capability-based NoFrak defense is being integrated into PhoneGap, but that’s just the first step...