Mobile Platform Security
[continued]

Spring 2016

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Admin

• Office hours: Wed 1:30 (not today)

• Lab 3 is out (due June 3, 8pm)
  – Android security
  – 3 parts (+1 extra credit)
  – You should not need to write a lot of code
  – Don’t procrastinate on getting an Android development environment set up!
Challenges with Isolated Apps

So mobile platforms isolate applications for security, but...

1. **Permissions**: How can applications access sensitive resources?

2. **Communication**: How can applications communicate with each other?
State of the Art

**Prompts** *(time-of-use)*

- Disruptive, which leads to prompt-fatigue.

**Manifests** *(install-time)*

- Out of context; not understood by users.

In practice, both are **overly permissive**: Once granted permissions, apps can misuse them.
Android 6.0: Prompts!

- First-use prompts for sensitive permission (like iOS).
- Big change! Now app developers need to check for permissions or catch exceptions.
Improving Permissions: AppFence

Today, ultimatums give app developers an unfair edge in obtaining permissions.

**The App that User Wishes to Install**

**THIS APPLICATION HAS ACCESS TO THE FOLLOWING:**

- NETWORK COMMUNICATION
  - full Internet access
- YOUR PERSONAL INFORMATION
  - read contact data
- PHONE CALLS
  - read phone state and identity

**OK**

I’d rather not share all that information just to try this app, but it looks like I have no choice.

**AppFence** can enable new interfaces that give users control over the use of their info.

**The App that User Wishes to Install**

**THIS APPLICATION HAS ACCESS TO THE FOLLOWING:**

- NETWORK COMMUNICATION
  - block Internet access if data will be sent to any servers known advertisers any third parties
- YOUR PERSONAL INFORMATION
  - allow access to all contact data
- DEVICE INFORMATION
  - allow access to anonymized device ID
  - report my phone # as 650 555 1212

**OK**

I’ll start by giving out only the information I think this app actually needs.
Improving Permissions: User-Driven Access Control

Insight:
A user’s natural UI actions within an application implicitly carry permission-granting semantics.
Let this application access my location now.

**Our study shows:**
Many users already believe (52% of 186) – and/or desire (68%) – that resource access follows the user-driven access control model.
New OS Primitive: Access Control Gadgets (ACGs)

**Approach:** Make resource-related UI elements first-class operating system objects (access control gadgets).

- To receive resource access, applications must embed a system-provided ACG.
- ACGs allow the OS to capture the user’s permission granting intent in application-agnostic way.
Challenges with Isolated Apps

So mobile platforms isolate applications for security, but...

1. **Permissions**: How can applications access sensitive resources?

2. **Communication**: How can applications communicate with each other?
Reminder: Android Applications

• **Activities** provide user interfaces.
• **Services** run in the background.
• **BroadcastReceivers** receive messages sent to multiple applications (e.g., BOOT_COMPLETED).
• **ContentProviders** are databases addressable by their application-defined URIs.

• **AndroidManifest.xml**
  – Specifies application components
  – Specifies required permissions
(2) Inter-Process Communication

• Primary mechanism in Android: **Intents**
  – Sent between application components
    • e.g., with `startActivity(intent)`

  – **Explicit**: specify component name
    • e.g., `com.example.testApp MainActivity`

  – **Implicit**: specify action (e.g., `ACTION_VIEW`) and/or data (URI and MIME type)
    • Apps specify **Intent Filters** for their components.
Unauthorized Intent Receipt

- **Attack #1**: Eavesdropping / Broadcast Thefts
  - Implicit intents make intra-app messages public.
- **Attack #2**: Activity Hijacking
  - May not always work:
- **Attack #3**: Service Hijacking
  - Android picks one at random upon conflict!
Intent Spoofing

• **Attack #1:** General intent spoofing
  – Receiving implicit intents makes component public.
  – Allows data injection.

• **Attack #2:** System intent spoofing
  – Can’t directly spoof, but victim apps often don’t check specific “action” in intent.
Permission Re-Delegation

- An application without a permission gains additional privileges through another application.
- **Demo video**
- Settings application is deputy: has permissions, and accidentally exposes APIs that use those permissions.
Aside: Incomplete Isolation

Embedded UIs and libraries always run with the host application’s permissions! (No same-origin policy here...)

[Shekhar et al.]

[Bar chart showing the number of apps installed vs the number of ad libraries installed.]

[Image of a map from the Google library, a social button from the Facebook library, and an ad from the ad library.]
More on Android...
Android Application Signing

• Apps are signed
  – Often with self-signed certificates
  – Signed application certificate defines which user ID is associated with which applications
  – Different apps run under different UIDs

• Shared UID feature
  – Shared Application Sandbox possible, where two or more apps signed with same developer key can declare a shared UID in their manifest
Shared UIDs

- App 1: Requests GPS / camera access
- App 2: Requests Network capabilities

Generally:
- First app can’t exfiltrate information
- Second app can’t exfiltrate anything interesting

With Shared UIDs (signed with same private key)
- Permissions are a superset of permissions for each app
- App 1 can now exfiltrate; App 2 can now access GPS / camera
File Permissions

• Files written by one application cannot be read by other applications
  – Previously, this wasn’t true for files stored on the SD card (world readable!) – Android cracked down on this

• It is possible to do full file system encryption
  – Key = Password/PIN combined with salt, hashed
Memory Management

- Address Space Layout Randomization to randomize addresses on stack
- Hardware-based No eXecute (NX) to prevent code execution on stack/heap
- Stack guard derivative
- Some defenses against double free bugs (based on OpenBSD’s dmalloc() function)
- etc.

[See http://source.android.com/tech/security/index.html]
Android Fragmentation

• Many different variants of Android (unlike iOS)
  – Motorola, HTC, Samsung, ...
• Less secure ecosystem
  – Inconsistent or incorrect implementations
  – Slow to propagate kernel updates and new versions

[https://developer.android.com/about/dashboards/index.html]
What about iOS?

• Apps are sandboxed
• Encrypted user data
  – See recent news...
• App Store review process is (maybe) stricter
  – But not infallible: e.g., see Wang et al. “Jekyll on iOS: When Benign Apps Become Evil” (USENIX Security 2013)

• No “sideloading” apps
  – Unless you jailbreak

Figure 1: High Level Intuition