CSE 484 / CSE M 584: Computer Security and Privacy

Mobile Platform Security [finish]

Spring 2017

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

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

- Feedback on project checkpoints
- Lab 3 is out (due June 2, 8pm)
  - Android security
  - 3 parts
  - You should not need to write a lot of code
  - Don’t procrastinate on getting an Android development environment set up!
    - (Took us ~15 minutes, but just in case...)

5/23/17
What’s Different about Mobile Platforms?

• Applications are isolated
  – Each runs in a separate execution context
  – No default access to file system, devices, etc.
  – **Different than traditional OSes** where multiple applications run with the same user permissions!

• **App Store**: approval process for applications
  – Market: Vendor controlled/Open
  – App signing: Vendor-issued/self-signed
  – User approval of permissions
More Details: Android

• Based on Linux
• Application sandboxes
  – Applications run as separate UIDs, in separate processes.
  – Memory corruption errors only lead to arbitrary code execution in the context of the particular application, not complete system compromise!
  – (Can still escape sandbox – but must compromise Linux kernel to do so.) ← allows rooting

Since 5.0: ART (Android runtime) replaces Dalvik VM to run apps natively

Enck et al.

Display
Bluetooth
GPS Receiver
Cellular Radio

Embedded Linux

Binder

Installed Applications

System Applications
Rooting and Jailbreaking

• Allows user to run applications with root privileges
  – e.g., modify/delete system files, app management, CPU
    management, network management, etc.

• Done by exploiting vulnerability in firmware to install su
  binary.

• Double-edged sword…

• Note: iOS is more restrictive than Android
  – Doesn’t allow “side-loading” apps, etc.
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
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?
(1) Permission Granting Problem

Smartphones (and other modern OSes) try to prevent such attacks by limiting applications’ access to:

– System Resources (clipboard, file system).
– Devices (camera, GPS, phone, ...).

How should operating system grant permissions to applications?

Standard approach: Ask the user.
State of the Art

Prompts (time-of-use)
State of the Art

Prompts (time-of-use)

Disruptive, which leads to prompt-fatigue.

Manifests (install-time)
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.
Are Manifests Usable?

Do users pay attention to permissions?

24 observed installations

- 17% looked at permissions
- 42% didn’t look, but aware
- 42% unaware of permissions

... but 88% of users looked at reviews.
Are Manifests Usable?

Do users understand the warnings?

<table>
<thead>
<tr>
<th>Permission</th>
<th>n</th>
<th>Correct Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Choice</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READCALENDAR</td>
<td>101</td>
<td>46</td>
</tr>
<tr>
<td>CHANGE_NETWORK_STATE</td>
<td>66</td>
<td>26</td>
</tr>
<tr>
<td>READ_SMS1</td>
<td>77</td>
<td>24</td>
</tr>
<tr>
<td>CALL_PHONE</td>
<td>83</td>
<td>16</td>
</tr>
<tr>
<td><strong>2 Choices</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAKE_LOCK</td>
<td>81</td>
<td>27</td>
</tr>
<tr>
<td>WRITE_EXTERNAL_STORAGE</td>
<td>92</td>
<td>14</td>
</tr>
<tr>
<td>READ_CONTACTS</td>
<td>86</td>
<td>11</td>
</tr>
<tr>
<td>INTERNET</td>
<td>109</td>
<td>12</td>
</tr>
<tr>
<td>READ_PHONE_STATE</td>
<td>85</td>
<td>4</td>
</tr>
<tr>
<td>READ_SMS2</td>
<td>54</td>
<td>12</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAMERA</td>
<td>72</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 4: The number of people who correctly answered a question. Questions are grouped by the number of correct choices. n is the number of respondents. (Internet Survey, n = 302)
Are Manifests Usable?

Do users act on permission information?

“Have you ever not installed an app because of permissions?”

25 interview responses

- Yes: 72%
- No: 8%
- Probably: 20%
Over-Permissioning

- Android permissions are badly documented.
- Researchers have mapped APIs → permissions.

www.android-permissions.org (Felt et al.), http://pscout.csl.toronto.edu (Au et al.)
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.

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
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.

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.
Let this application access my location now.

Insight:
A user’s natural UI actions within an application implicitly carry permission-granting semantics.

[our work]
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.
(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.]

Ad from ad library

Map from Google library

Social button from Facebook 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