CSE 484 / CSE M 584: Computer Security and Privacy

Mobile Platform Security

Spring 2015

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...
Roadmap

• Mobile malware
• Mobile platforms vs. traditional platforms
• Deep dive into Android
  – Continued next Wednesday
  – More details on iOS in section

• Guest lectures Wednesday and Friday
• Holiday on Monday!
Questions: Mobile Malware

**Q1:** How might malware authors get malware onto phones?

**Q2:** What are some goals that mobile device malware authors might have?

**Q3:** What technical things might malware authors do?
Smartphone (In)Security

Users accidentally install malicious applications.

Over 60% of Android malware steals your money via premium SMS, hides in fake forms of popular apps

By Emil Protalinski, Friday, 5 Oct '12, 05:50pm
Even legitimate applications exhibit questionable behavior.

_Hornyack et al._: 43 of 110 Android applications sent location or phone ID to third-party advertising/analytics servers.
Malware in the Wild

Android malware is growing.
Mobile Malware Attack Vectors

• Unique to phones:
  – Premium SMS messages
  – Identify location
  – Record phone calls
  – Log SMS

• Similar to desktop/PCs:
  – Connects to botmasters
  – Steal data
  – Phishing
  – Malvertising
Mobile Malware Examples

• **DroidDream** (Android)
  – Over 58 apps uploaded to Google app market
  – Conducts data theft; send credentials to attackers

• **Zitmo** (Symbian, BlackBerry, Windows, Android)
  – Poses as mobile banking application
  – Captures info from SMS – steal banking 2nd factors
  – Works with Zeus botnet

• **Ikee** (iOS)
  – Worm capabilities (targeted default ssh pwd)
  – Worked only on jailbroken phones with ssh installed
Mobile Malware Examples

“ikee is never going to give you up”
(Android) Malware in the Wild

What does it do?

<table>
<thead>
<tr>
<th>Root Exploit</th>
<th>Remote Control</th>
<th>Financial Charges</th>
<th>Information Stealing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Net</td>
<td>SMS</td>
<td>Phone Call</td>
</tr>
<tr>
<td># Families</td>
<td>20</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td># Samples</td>
<td>1204</td>
<td>1171</td>
<td>1</td>
</tr>
</tbody>
</table>
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.
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

[Enck et al.]
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
Android Malware Techniques

- Add background Service
- Modify existing application source code
- Component library replacement

- To avoid basic signature detection:
  - Dynamically download new Dalvik bytecode
  - Use DexClassLoader API to run the downloaded code

- Use exploit to obtain root access
- Many other techniques
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.

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

**Manifests (install-time)**

Out of context; not understood by users.
Are Manifests Usable?

Do users pay attention to permissions?

- 42% Didn’t look, but aware
- 17% Looked at permissions
- 42% Unaware of permissions

... but 88% of users looked at reviews.

[Felt et al.]
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>READ_CALENDAR</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>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>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: 8%
- No: 20%
- Probably: 72%
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.)
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.
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.

---

Network Communication
- full Internet access

Personal Information
- read contact data

Phone Calls
- read phone state and identity

Device Information
- allow access to device ID
- report my phone # as 650 555 1212

---

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

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

Let this application access my location now.

Insight:
A user’s natural UI actions within an application implicitly carry permission-granting semantics.
Access Control Gadgets (ACGs)

- Special UI elements that carry permission-granting semantics: When user clicks, grant access.
- ACGs are owned by system and embedded by apps: need to secure them!
  - No clickjacking, no programmatic clicking, etc.