Admin

• Lab #2 due today
• No class Monday (Memorial Day)
• Lab #3 out by early next week, due June 7
  – Tip: Read the entire lab description first
Roadmap

• Mobile malware
• Mobile platforms vs. traditional platforms
• Deep dive into Android
Mobile Malware: Threat Modeling

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

**Q2:** What are some goals that mobile device malware authors might have, or technical attacks they might attempt? *How does this different from desktop settings?*
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
Smartphone (In)Security

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.

Android flashlight app tracks users via GPS, FTC says hold on
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
Malware in the Wild

Android malware grew quickly!
Today: millions of samples.
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 2\textsuperscript{nd} factors
  - Works with Zeus botnet

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

“ikee is never going to give you up”
Why All These Problems?

Not because smartphone OS designers don’t care about security...
Background: Before Mobile Platforms

Assumptions in traditional OS (e.g., Unix) design:
1. There may be multiple users who don’t trust each other.
2. Once an application is installed, it’s (more or less) trusted.
Background: Before Mobile Platforms

Assumptions in traditional OS (e.g., Unix) design:
1. There may be multiple users who don’t trust each other.
2. Once an application is installed, it’s (more or less) trusted.

FranziBook:Desktop franzi$ whoami
franzi

FranziBook:Desktop franzi$ id
uid=501(franzi) gid=20(staff) groups=20(staff),401(com.apple.sharepoint.group.1),502(access_bpf),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),33(_appstore),100(_lproperator),204(_developer),395(com.apple.access_ftp),398(com.apple.access_screensharing),399(com.apple.access_ssh)

FranziBook:Desktop franzi$ ls -l hello.txt
-rw-r--r-- 1 franzi staff 0 Nov 29 10:08 hello.txt

FranziBook:Desktop franzi$ chmod 700 hello.txt
FranziBook:Desktop franzi$ ls -l hello.txt
-rwx------ 1 franzi staff 0 Nov 29 10:08 hello.txt
Background: Before Mobile Platforms

Assumptions in traditional OS (e.g., Unix) design:
1. There may be multiple users who don’t trust each other.
2. Once an application is installed, it’s (more or less) trusted.

Apps can do anything the UID they’re running under can do.
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.) \(\leftarrow\) allows rooting

[Enck et al.]
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.
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)

Manifests (install-time)
State of the Art

Prompts (time-of-use)

Disruptive, which leads to prompt-fatigue.

Manifests (install-time)

Storage
Modify/delete SD card contents

System tools
Prevent phone from sleeping, write sync settings

Your location
Fine (GPS) location

Network communication
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

- 42% Looked at permissions
- 42% Didn’t look, but aware
- 17% 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>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>1 Choice</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>2 Choices</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: 8%
- No: 72%
- Probably: 20%
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.
(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.
Eavesdropping and Spoofing

• Buggy apps might accidentally:
  – Expose their component-to-component messages publicly → eavesdropping
  – Act on unauthorized messages they receive → spoofing
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.]
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

CFG exhibited in vetting process. Red node represents the vulnerabilities. Dynamic CFG in victim’s device after exploiting vulnerabilities.