# CSE 484 / CSE M 584: Computer Security and Privacy

Autumn 2019

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

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

# Announcements

- My office hours
  - 12/4 (Wed), 11:30am, CSE1 Attrium?
- Final Project checkpoint 2 looked great!
- HW3 + Lab3: both "light", but please don't wait until Friday to start
- Friday: *Optional* opportunity to learn about Space + Security

# Roadmap


- History, How we got here

- Mobile malware
- Mobile platforms vs. traditional platforms
- Dive into Android

# Questions: Mobile Malware

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

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

# Smartphone (In)Security

Users accidentally install malicious applications.

# Smartphone (In)Security

Even legitimate applications exhibit questionable behavior.

**Top Mobile Apps Overwhelmingly Leak Private Data: Study**

By Robert Lemos | Posted 2013-07-31 Email Print

*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**

By Michael Kassner in IT Security, December 11, 2013, 9:49 PM PST

# And in the news this morning…

## Android 'spoofing' bug helps targets bank accounts

By Mark Ward
Technology correspondent, BBC News

3 hours ago

The loophole was found when a security firm probed how bad apps stole cash

**A "major" security weakness in Google's Android software has let cyber-thieves craft apps that can steal banking logins, a security firm has found.**

The bug lets attackers create fake login screens that can be inserted into legitimate apps to harvest data.

More than 60 financial institutions have been targeted by the technique, a survey of the Play store indicated.

"It targeted several banks in several countries and the malware successfully exploited end users to steal money," said Tom Hansen, chief technology officer of Norwegian mobile security firm Promon, which found the bug.

### Lurking threat

The problem emerged after Promon analysed malicious apps that had been spotted draining bank accounts.

Called Strandhogg, the vulnerability can be used to trick users into thinking they are using a legitimate app but are actually clicking on an overlay created by the attackers.

"We'd never seen this behaviour before," said Mr Hansen.

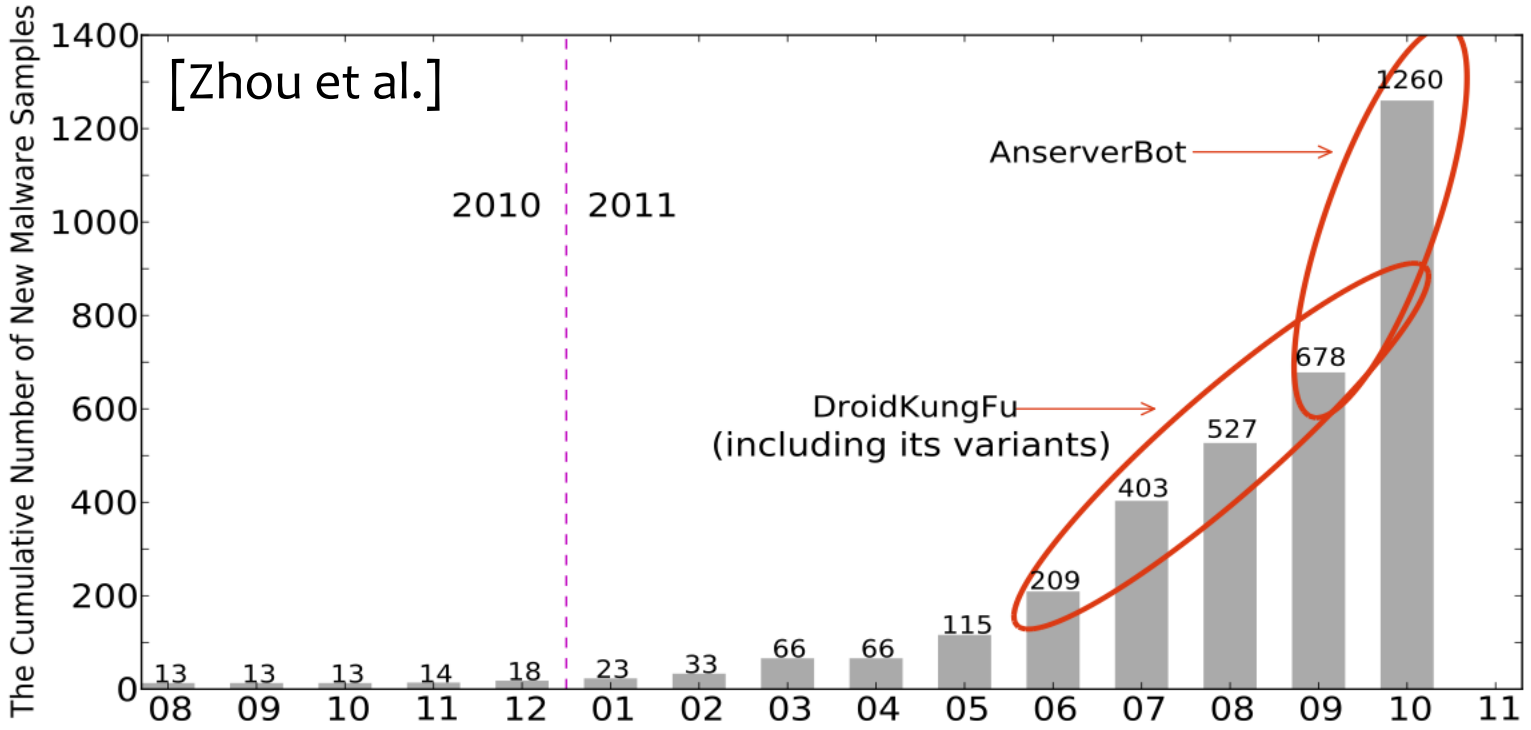
"As the operating system gets more complex it's hard to keep track of all its interactions," he said. "This looks like the kind of thing that gets lost in that complexity."

# Mobile Malware Goals

- "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.



# 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.

# 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.

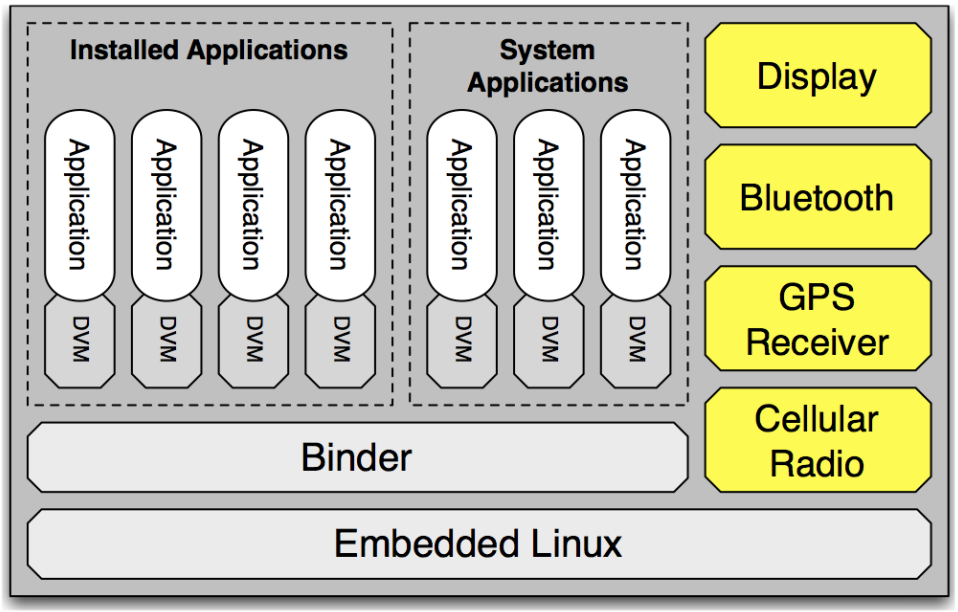
# What's Different about Mobile Platforms?

- **Isolation:** 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

[Enck et al.]




- 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

# Challenges with Isolated Apps

Mobile platforms isolate applications for security, but…

1. Permissions: How can applications access sensitive resources?
2. Communication: How can applications communicate with each other?

We've seen similar issues on the Web, and to some extent with IoT (see Lab3).

# 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, … ).

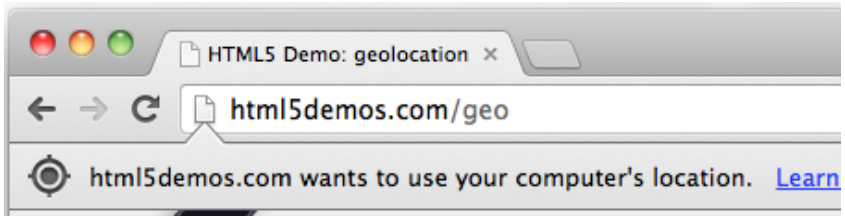

– (We've seen permission granting as a challenge with mobile devices, and expect to see them the future for other technologies as well… )

How should operating system grant permissions to applications?

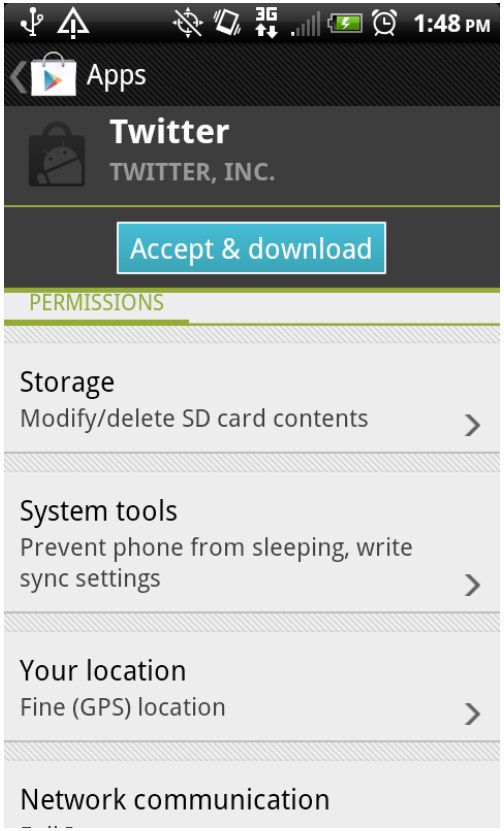
Standard approach: Ask the user.

# Two Ways to Ask the User

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


**Manifests** (install-time)

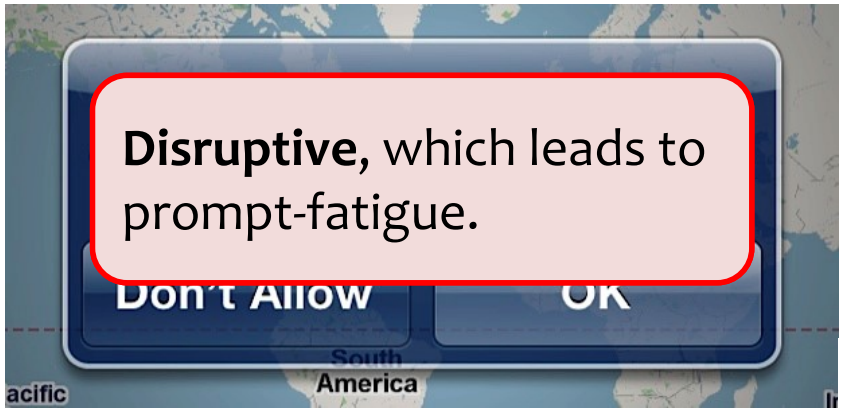


# Questions

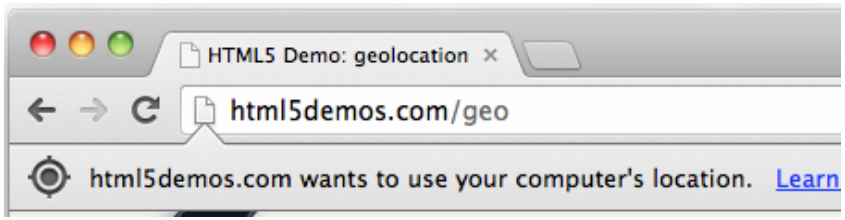
- Q: What are the pros and cons of the manifest-based permission model?
- Q: What are the pros and cons of the “ask each use” permission mode?

# Two Ways to Ask the User

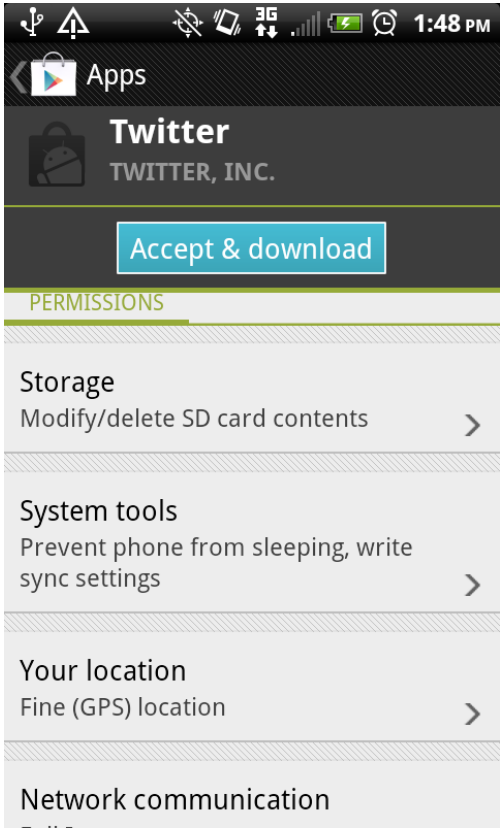
## **Prompts** (time-of-use)




**Disruptive,** which leads to prompt-fatigue.

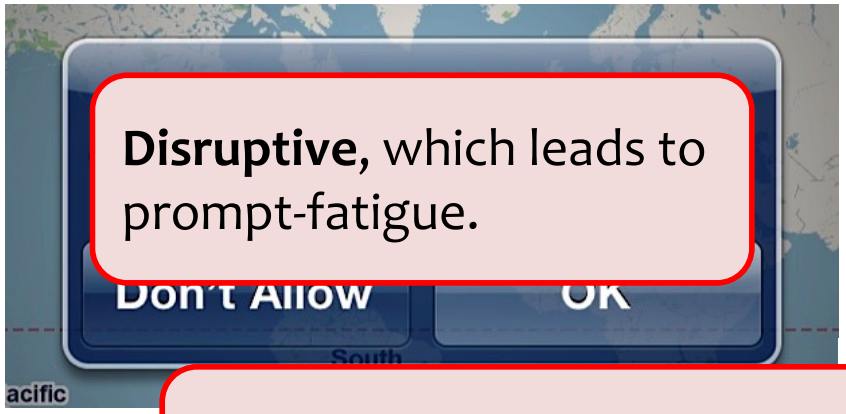



## **Manifests** (install-time)

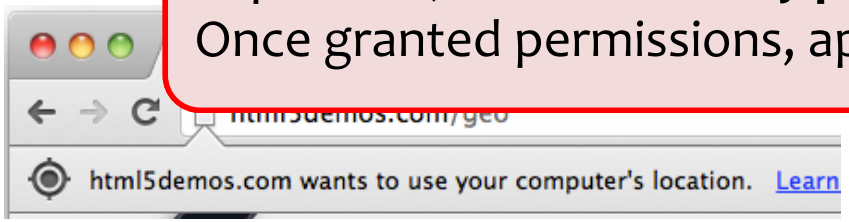


# Two Ways to Ask the User

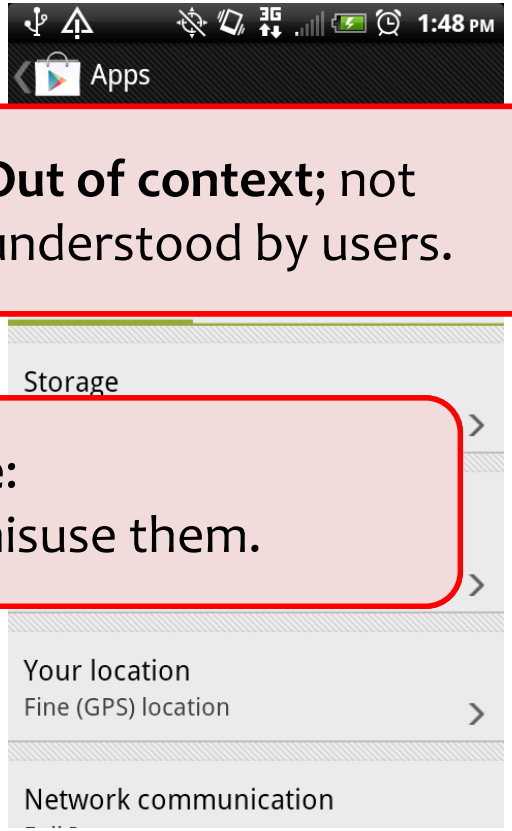
**Prompts** (time-of-use)




**Disruptive**, which leads to prompt-fatigue.




**Manifests** (install-time)



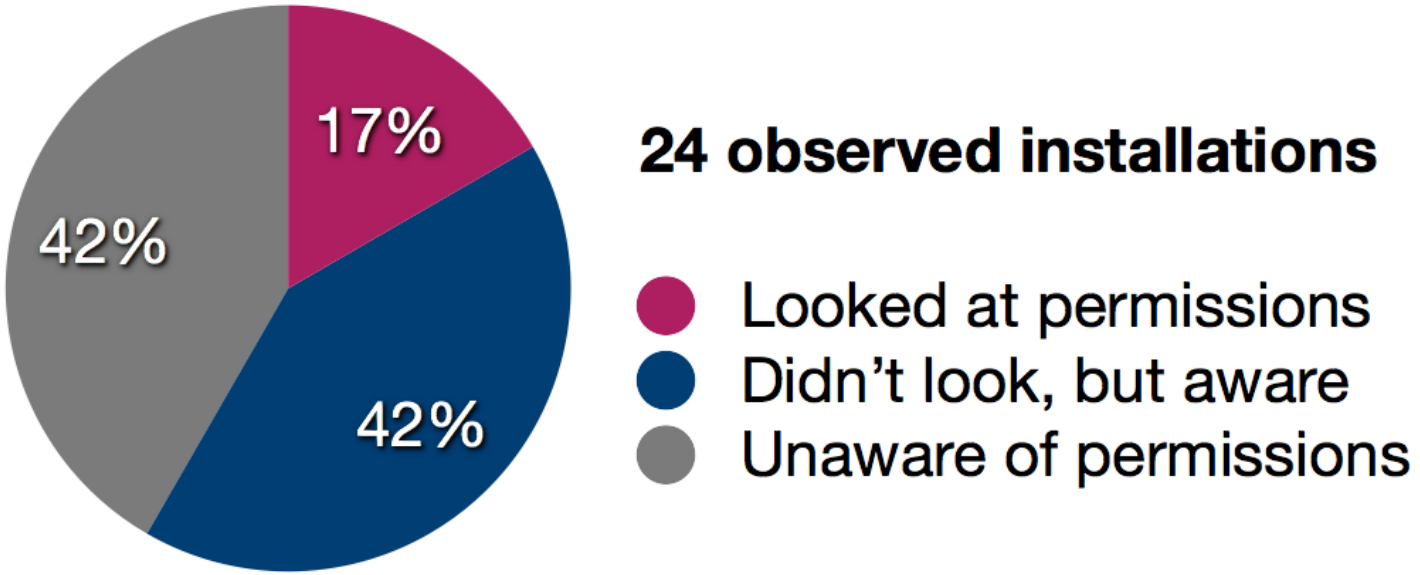

**Out of context**; not understood by users.

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

[Felt et al.]

# Are Manifests Usable?

Do users pay attention to permissions?




… but 88% of users looked at reviews.

[Felt et al.]

# Are Manifests Usable?

Do users understand the warnings?

| | Permission | $n$ | Correct Answers | |
|---|---|---|---|---|
| 1 Choice | READ_CALENDAR | 101 | 46 | 45.5% |
| | CHANGE_NETWORK_STATE | 66 | 26 | 39.4% |
| | READ_SMS$_1$ | 77 | 24 | 31.2% |
| | CALL_PHONE | 83 | 16 | 19.3% |
| 2 Choices | WAKE_LOCK | 81 | 27 | 33.3% |
| | WRITE_EXTERNAL_STORAGE | 92 | 14 | 15.2% |
| | READ_CONTACTS | 86 | 11 | 12.8% |
| | INTERNET | 109 | 12 | 11.0% |
| | READ_PHONE_STATE | 85 | 4 | 4.7% |
| | READ_SMS$_2$ | 54 | 12 | 22.2% |
| 4 | CAMERA | 72 | 7 | 9.7% |

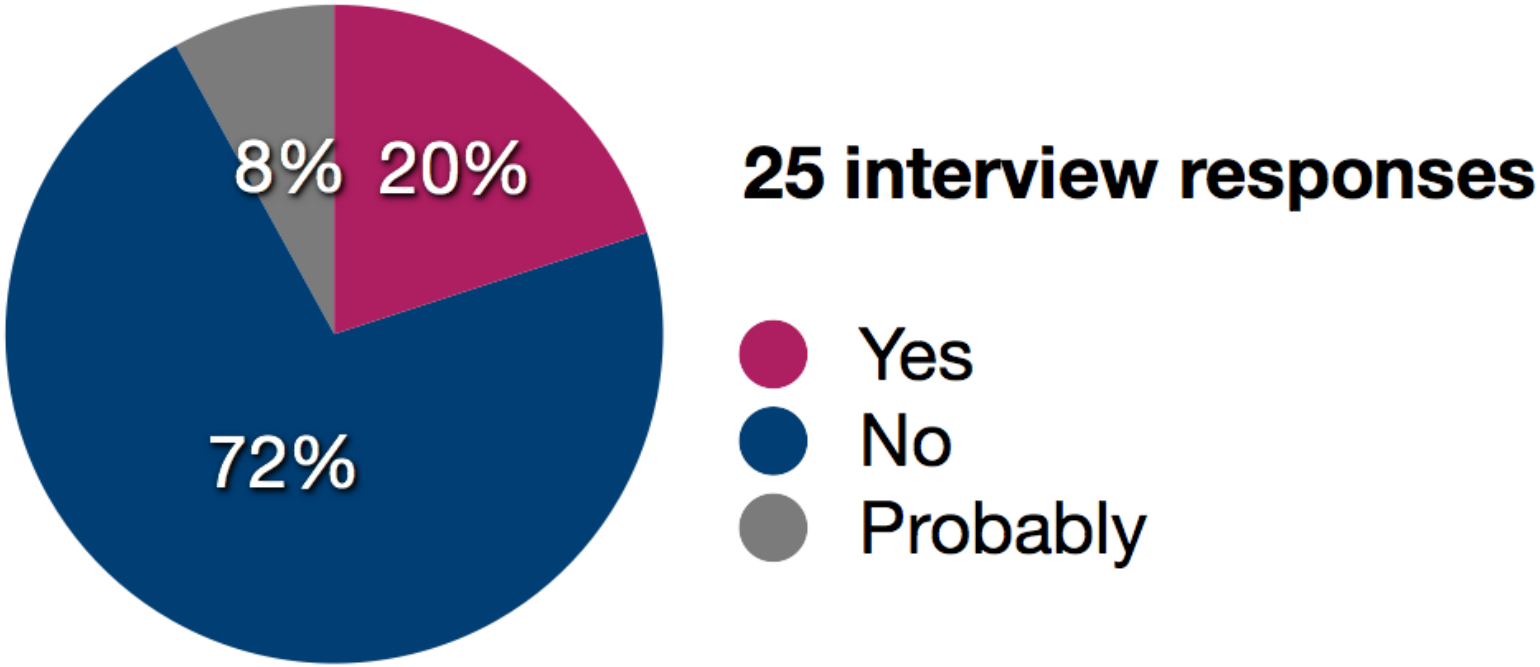
Table 4: The number of people who correctly answered a question. Questions are grouped by the number of correct choices. $n$ is the number of respondents. (Internet Survey, $n = 302$)

[Felt et al.]

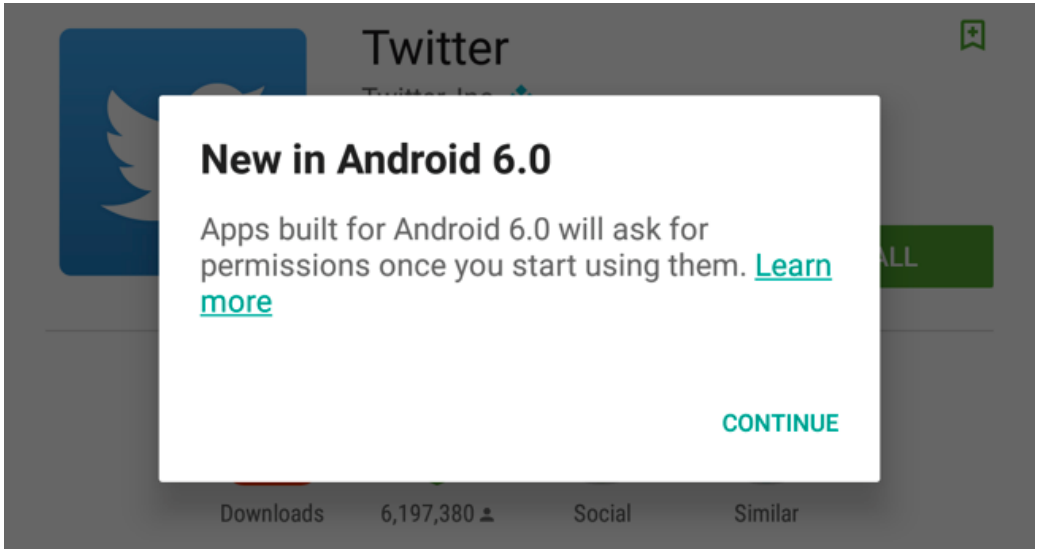
# Are Manifests Usable?

Do users act on permission information?

"Have you ever not installed an app because of permissions?"



# Android 6.0: Prompts!



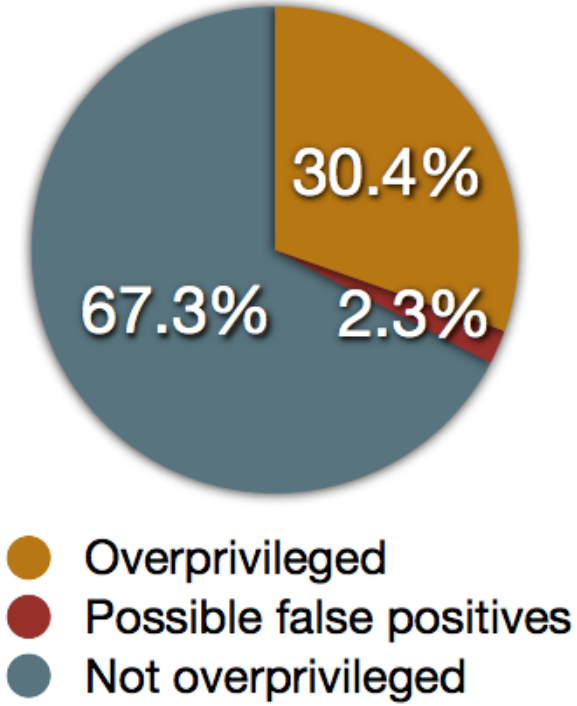

- First-use prompts for sensitive permission (like iOS).
- Big change. Now app developers need to check for permissions or catch exceptions.

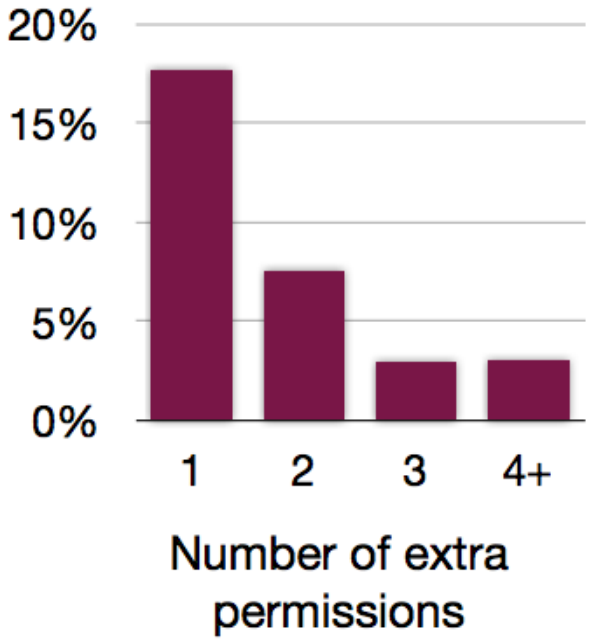
[Felt et al.]

# Over-Permissioning

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

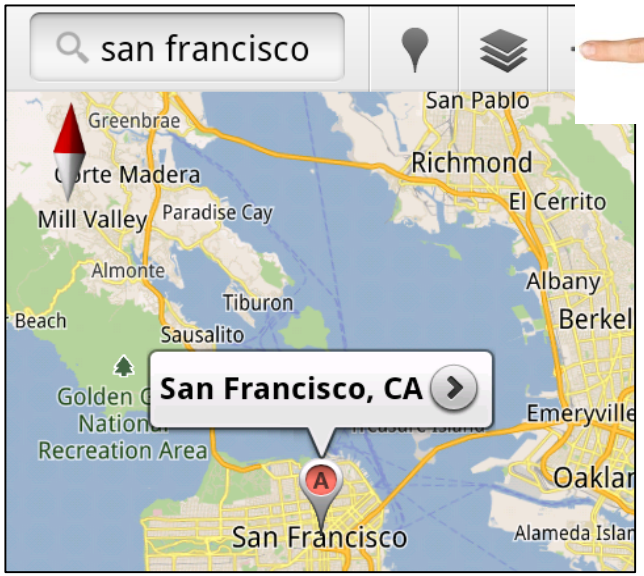
www.android-permissions.org (Felt et al.), http://pscout.csl.toronto.edu (Au et al.)

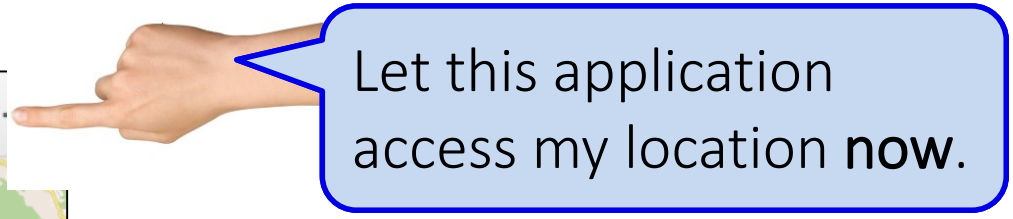


[Roesner et al]

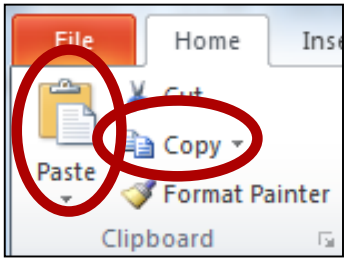
# Improving Permissions: User-Driven Access Control

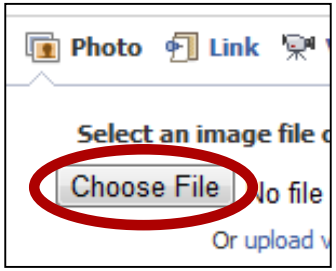



**Insight:**

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


[Roesner et al]

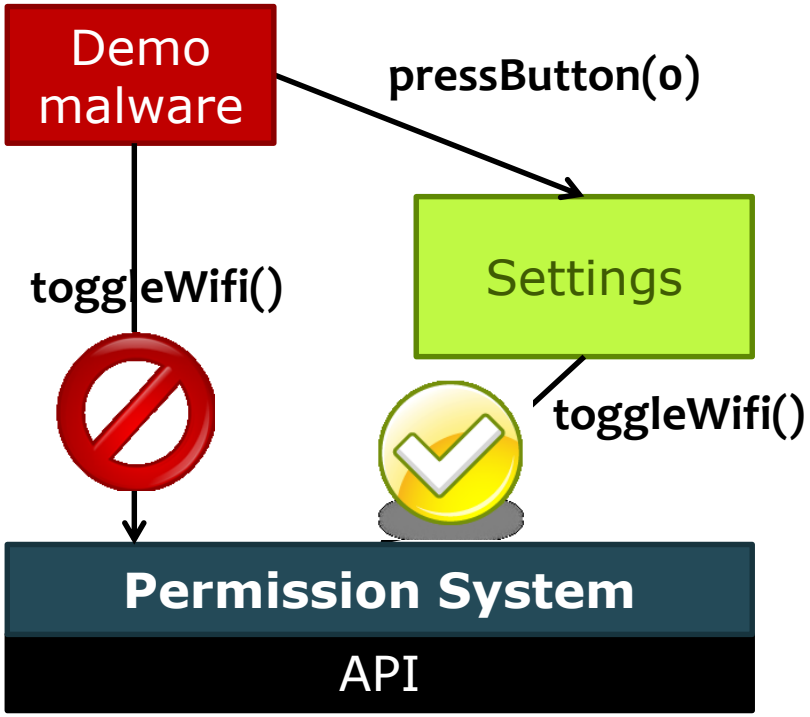
# Improving Permissions: User-Driven Access Control

***Study:***
Many users already believe (52% of 186) – and/or desire (68%) – that resource access follows the user-driven access control model.

[Felt et al.]

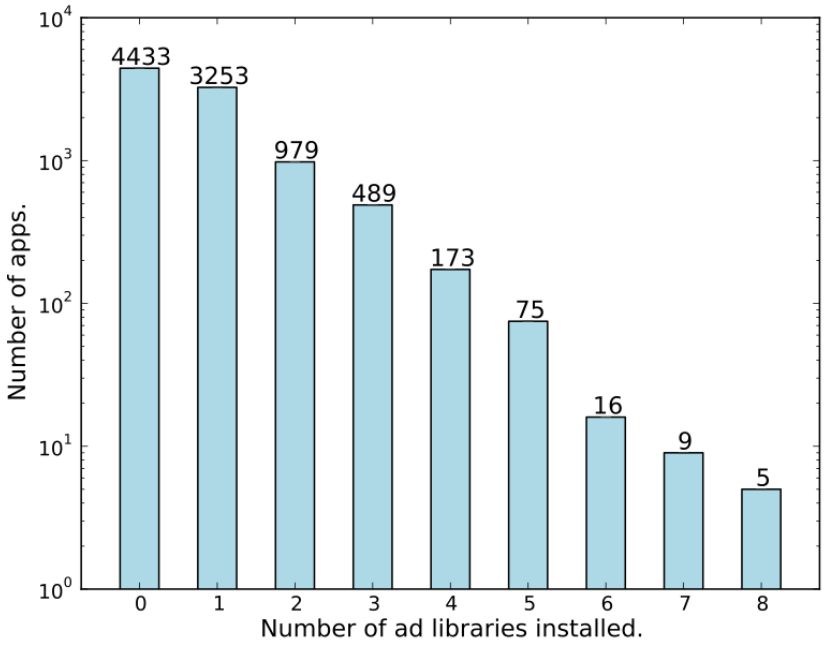
# Permission Re-Delegation

- An application without a permission gains additional privileges through another application.
- 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**

# Android Application Signing

- Apps are signed
  - 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
- Fact that these properties have changed over time speaks to
  - The historic challenges for security, even at large companies
  - The importance of considering security from the beginning

# Android Permission Recommendations

- Only use the permissions necessary for your app to work
- Pay attention to permissions required by libraries
- Be transparent
- Make system accesses explicit. Providing continuous indications when you access sensitive capabilities (for example, the camera or microphone) …

https://developer.android.com/training/permissions/usage-notes

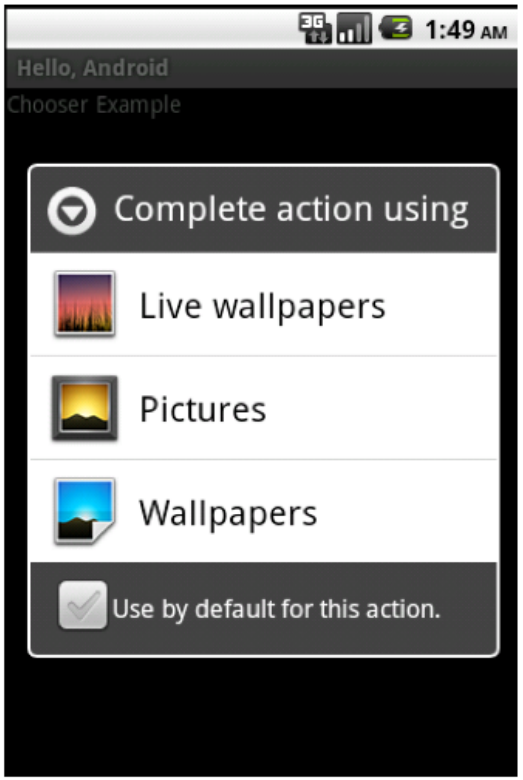
# (2) Inter-Process Communication

- 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.

[Chin et al.]

# 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!



[Chin et al.]

# 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.

# 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]

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 1.0% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.8% |
| 4.1.x | Jelly Bean | 16 | 3.2% |
| 4.2.x | | 17 | 4.6% |
| 4.3 | | 18 | 1.3% |
| 4.4 | KitKat | 19 | 18.8% |
| 5.0 | Lollipop | 21 | 8.7% |
| 5.1 | | 22 | 23.3% |
| 6.0 | Marshmallow | 23 | 31.2% |
| 7.0 | Nougat | 24 | 6.6% |
| 7.1 | | 25 | 0.5% |

*Data collected during a 7-day period ending on May 2, 2017.*
*Any versions with less than 0.1% distribution are not shown.*