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…**)

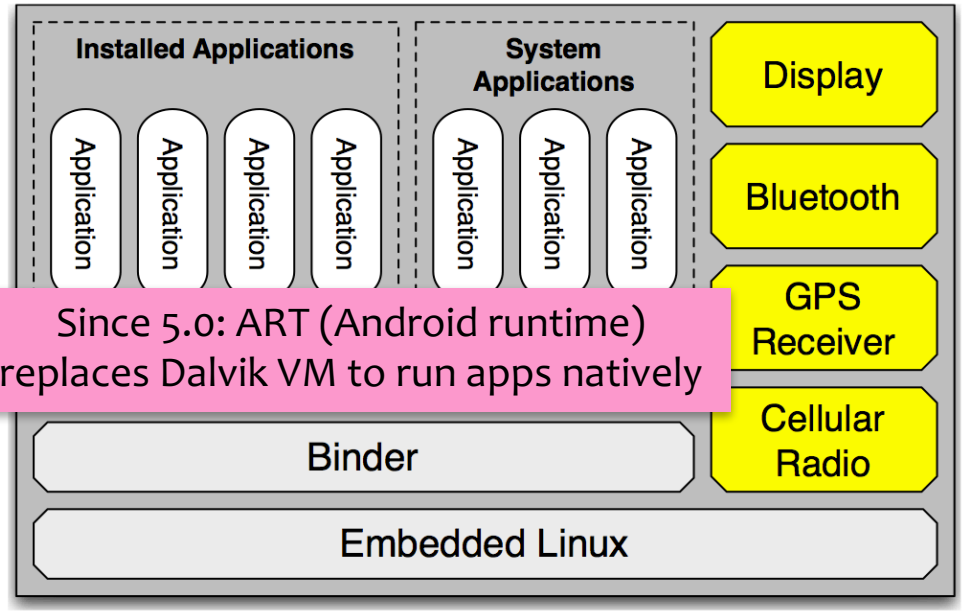
What's Different about Mobile Platforms?

- 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

[Enck et al.]



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

- 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

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:

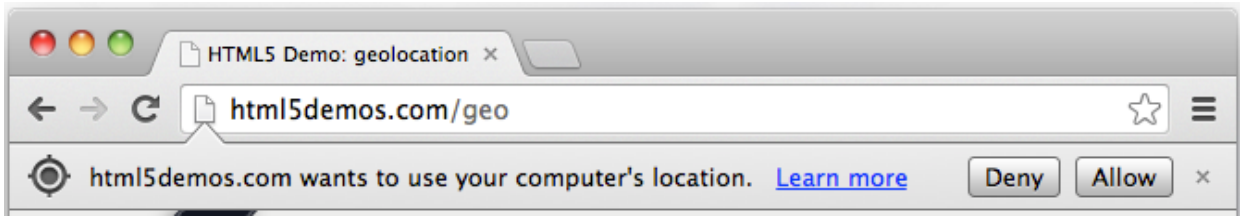
- System Resources (clipboard, file system).
- Devices (camera, GPS, phone, …).

How should operating system grant permissions to applications?

Standard approach: Ask the user.

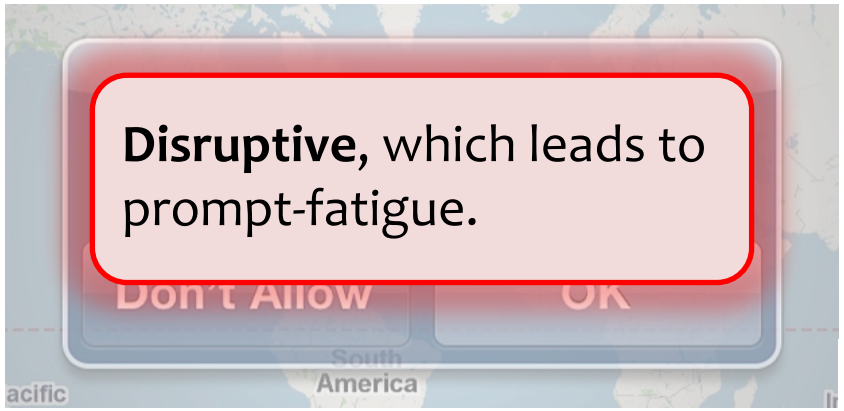
State of the Art

Prompts (time-of-use)

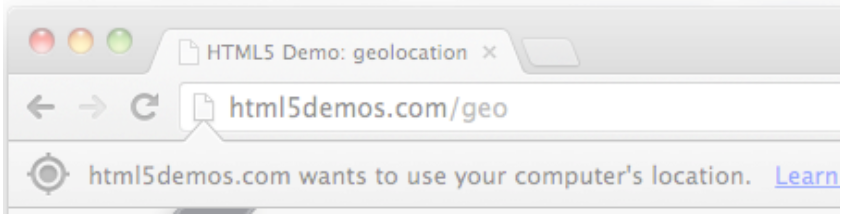


State of the Art

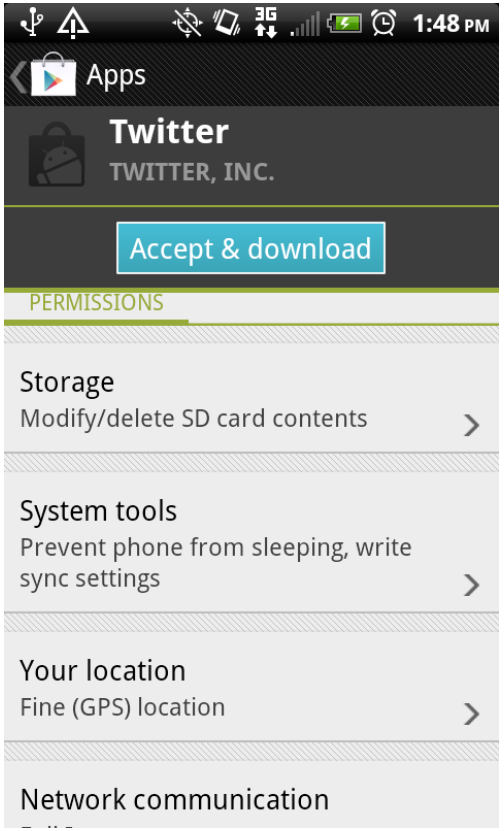
Prompts (time-of-use)



Disruptive, which leads to prompt-fatigue.



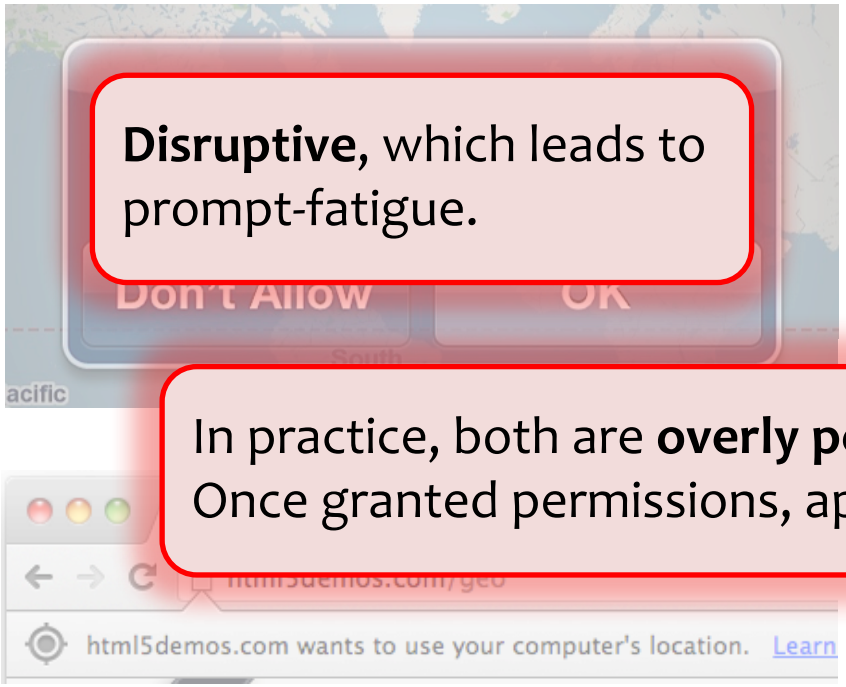
Manifests (install-time)

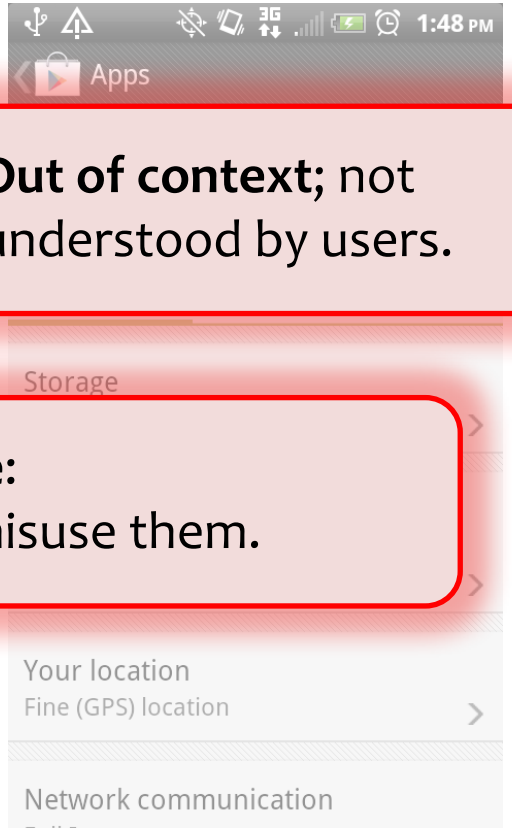


State of the Art

Prompts (time-of-use)

Manifests (install-time)





Disruptive, which leads to prompt-fatigue.

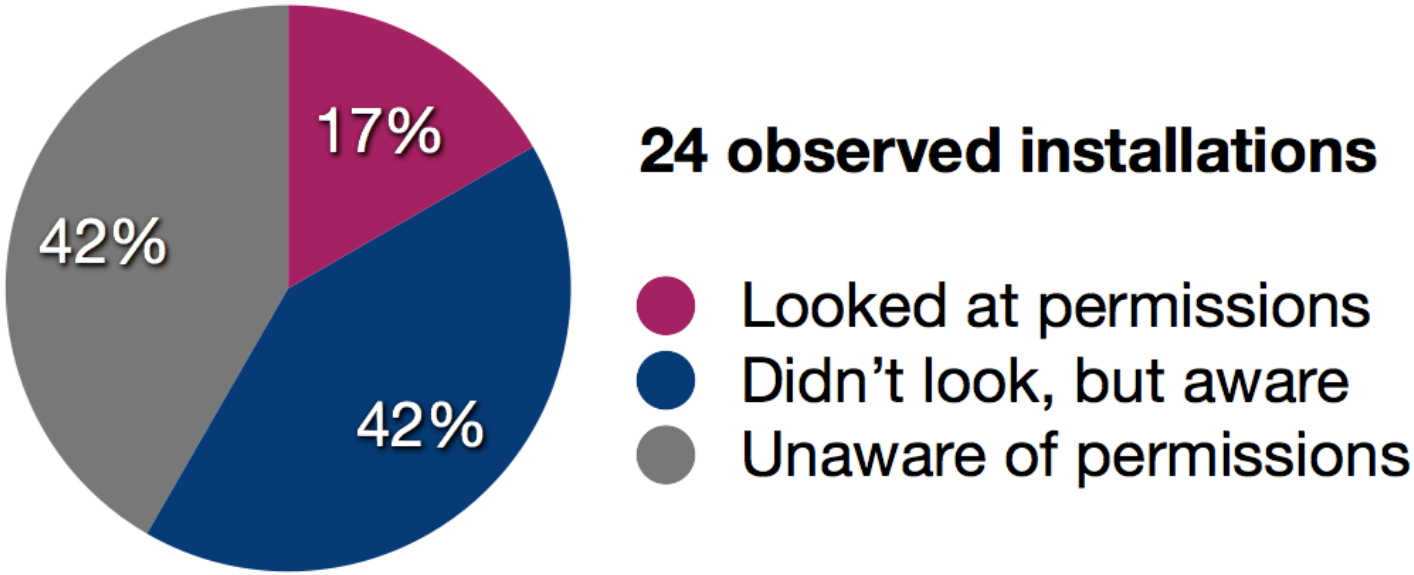
Out of context; not understood by users.

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%

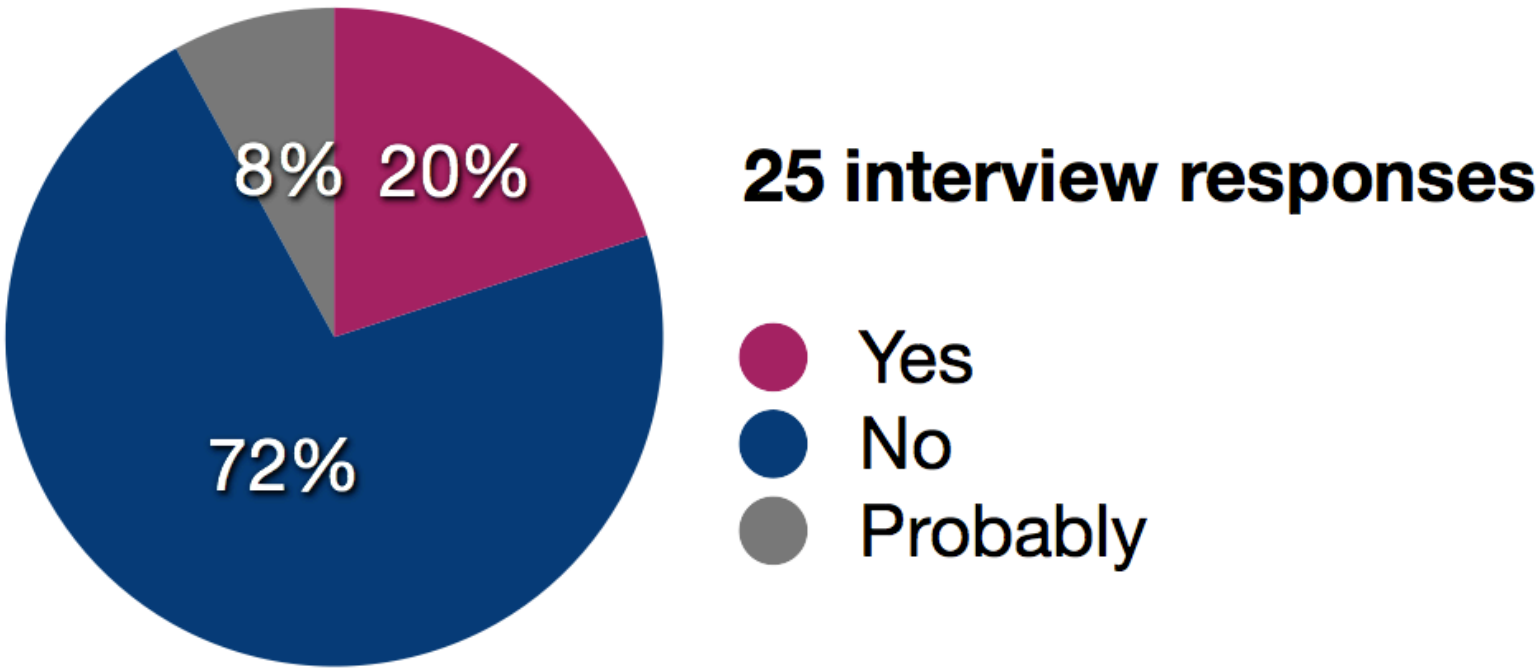
Table 4: The number of people who correctly answered a question. Questions are grouped by the number of correct choices. n is the number of respondents. (Internet Survey, $n = 302$)

[Felt et al.]

Are Manifests Usable?

Do users act on permission information?

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

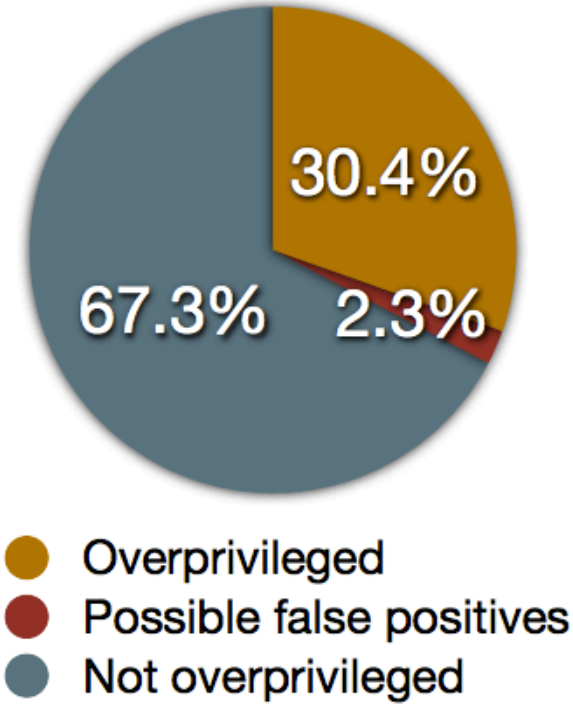


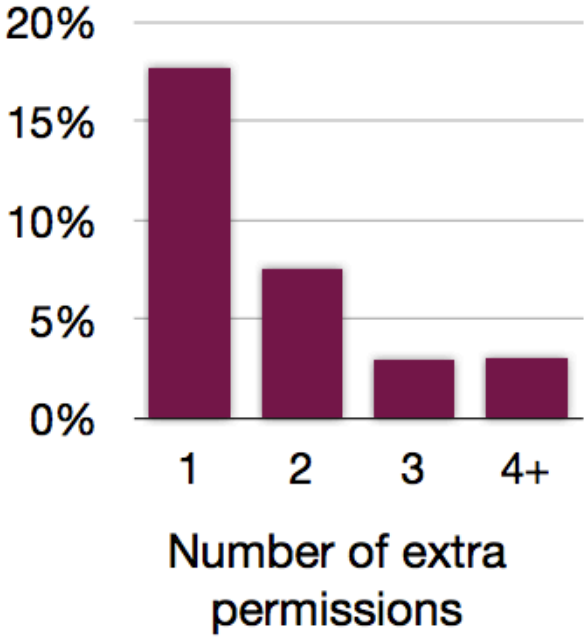
[Felt et al.]

Over-Permissioning

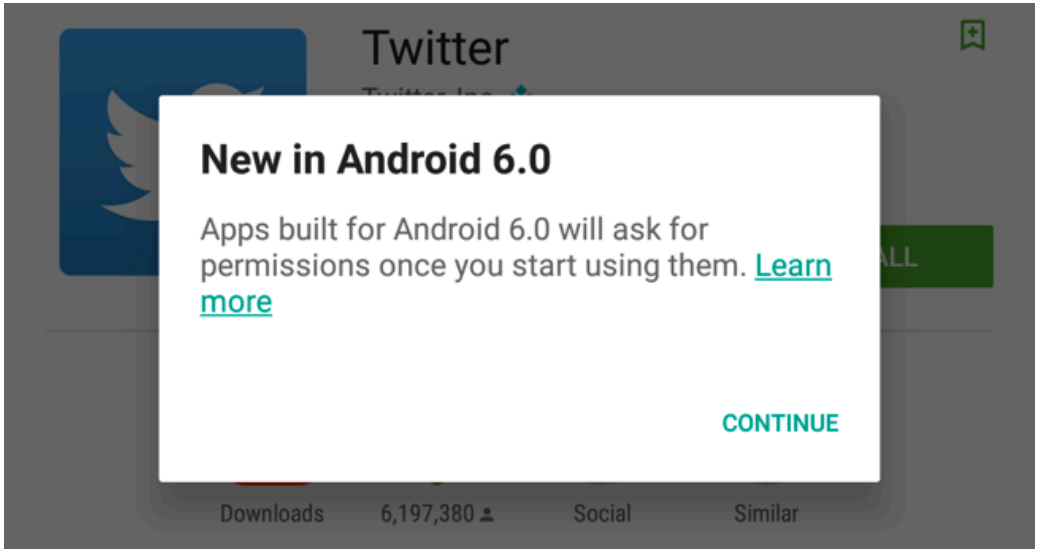
- Android permissions are badly documented.
- Researchers have mapped APIs → permissions.

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





Android 6.0: Prompts!

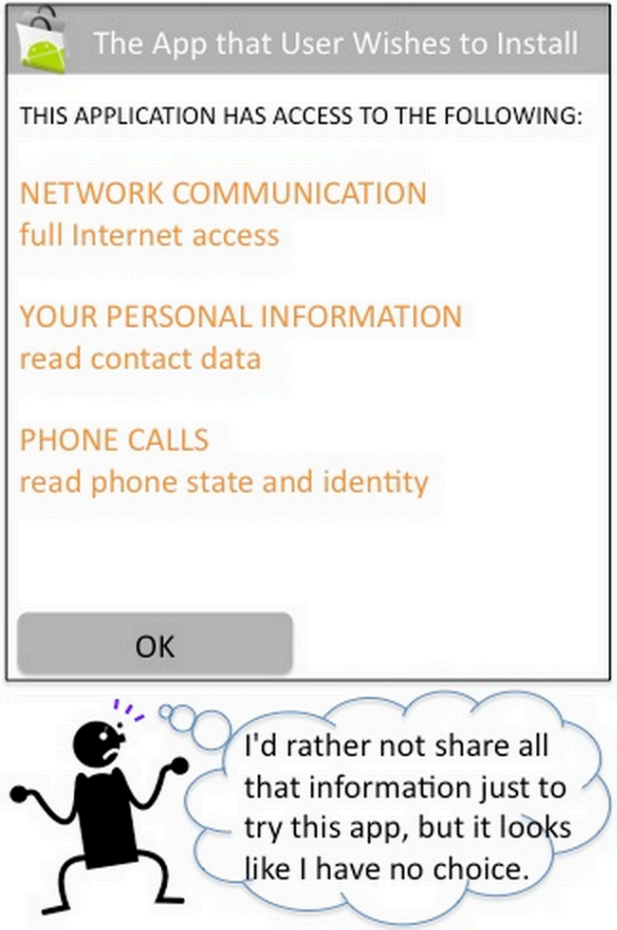


- First-use prompts for sensitive permission (like iOS).
- Big change! Now app developers need to check for permissions or catch exceptions.

[Hornyack et al.]

Improving Permissions: AppFence

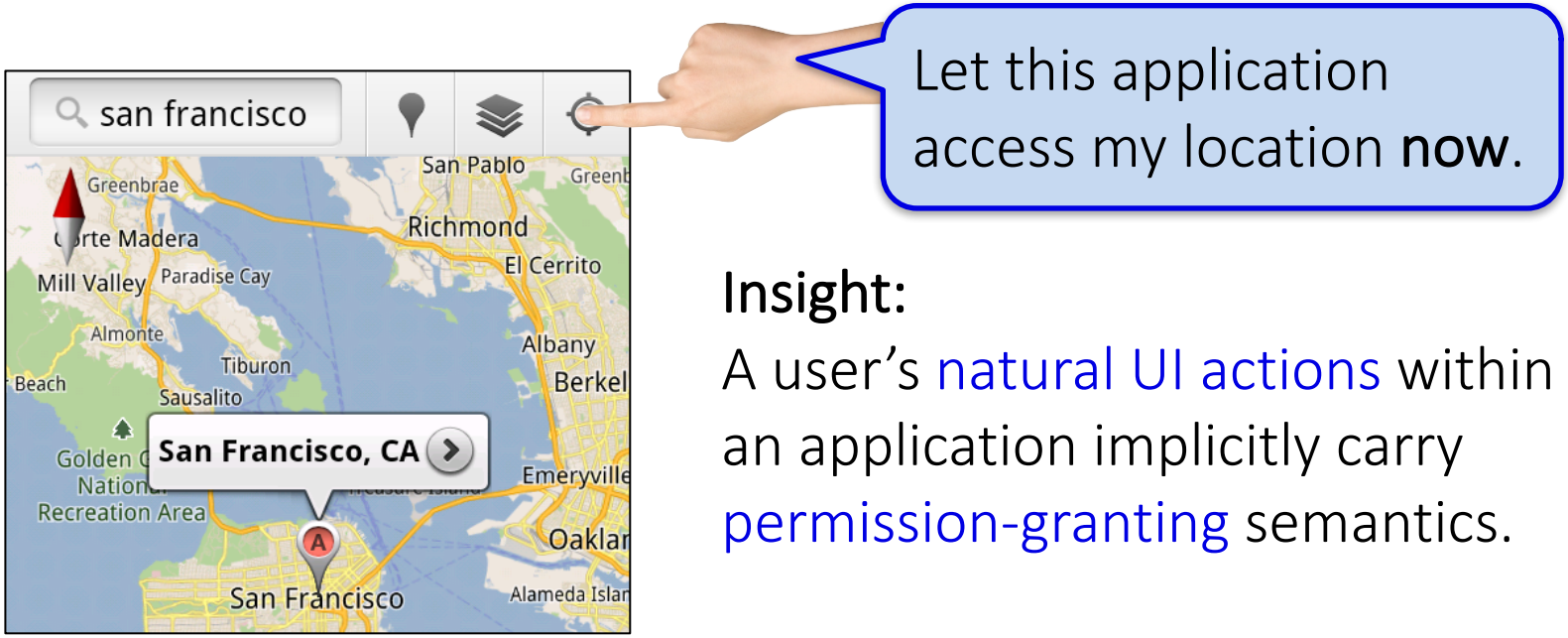
Today, ultimatums give app developers an unfair edge in obtaining permissions.



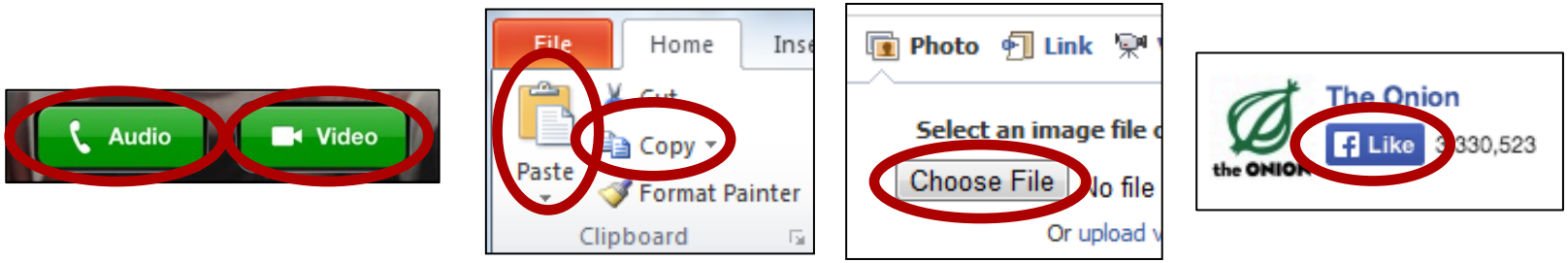
AppFence can enable new interfaces that give users control over the use of their info.

[our work]

Improving Permissions: User-Driven Access Control

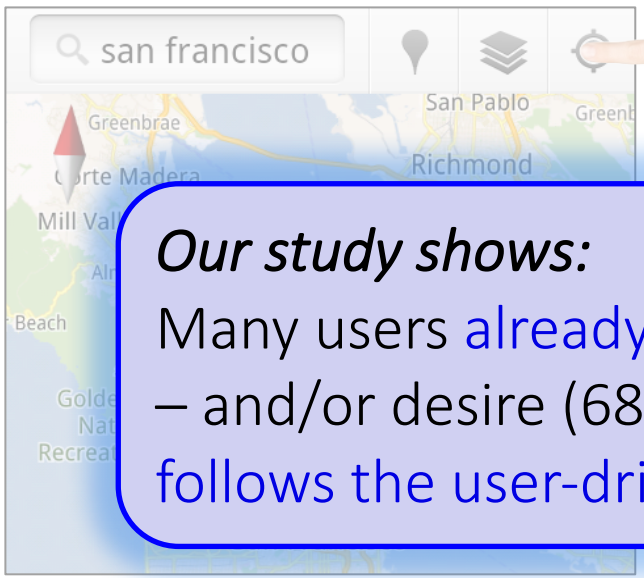


Insight:
A user's natural UI actions within an application implicitly carry permission-granting semantics.



[our work]

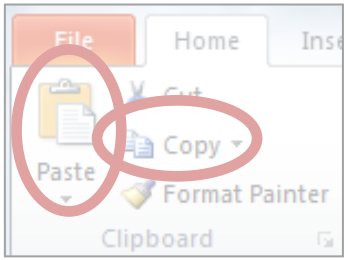
Improving Permissions: User-Driven Access Control

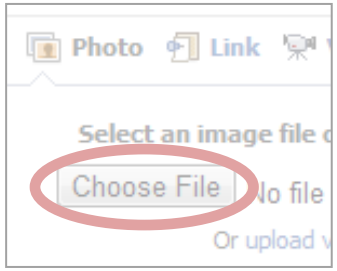


Let this application access my location **now**.

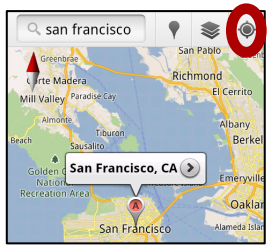
Our study shows:
Many users already believe (52% of 186) – and/or desire (68%) – that resource access follows the user-driven access control model.

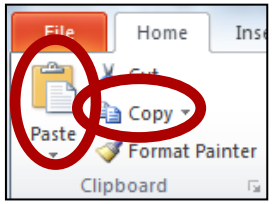
ns within
carry
antics.

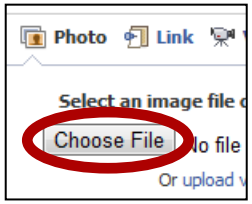




New OS Primitive: Access Control Gadgets (ACGs)







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.

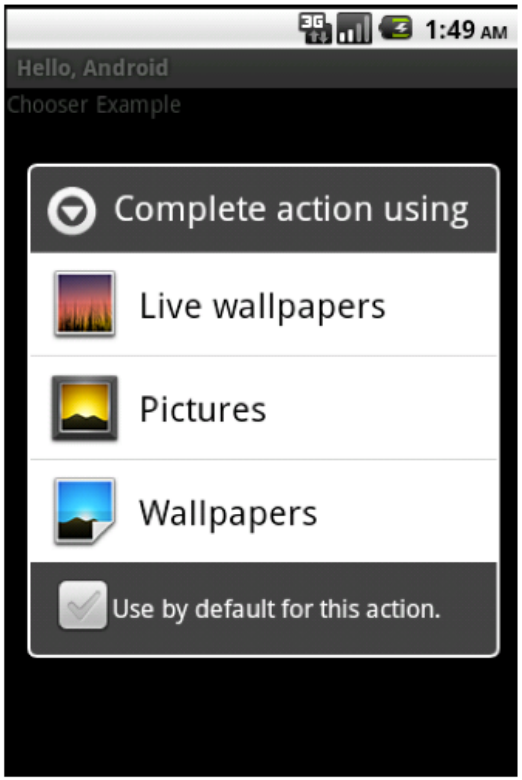
(2) Inter-Process Communication

- 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.

[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.]

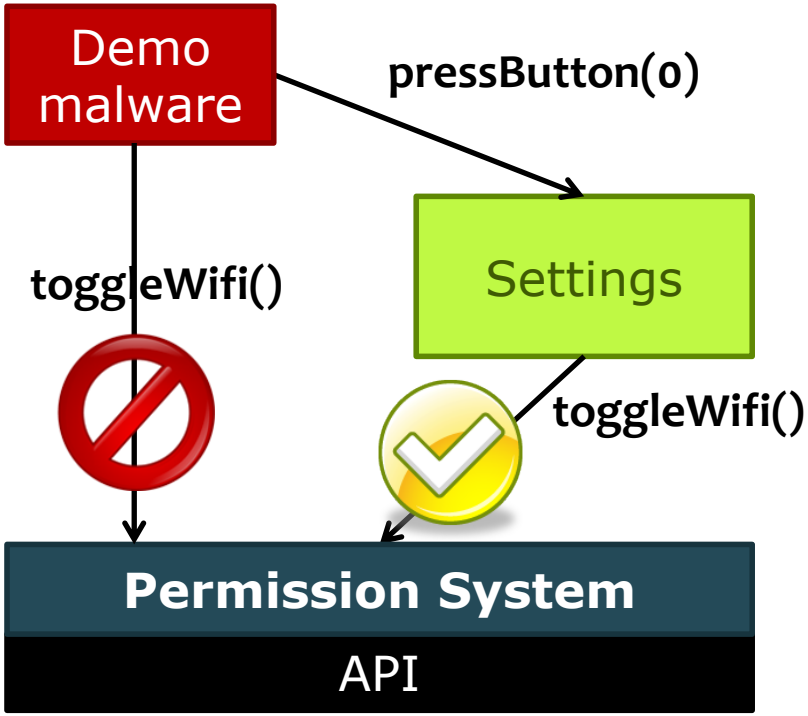
Intent Spoofing

- 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.

[Felt et al.]

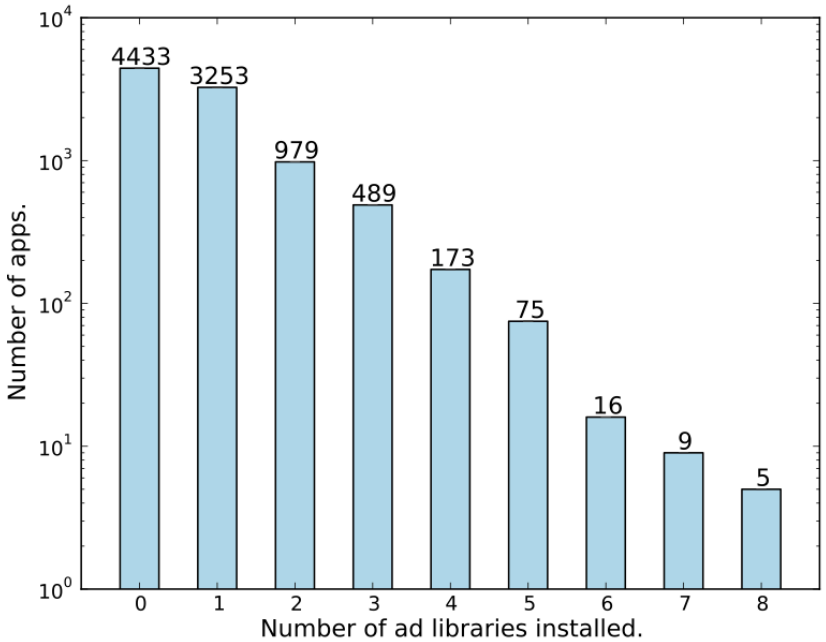
Permission Re-Delegation

- 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.



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

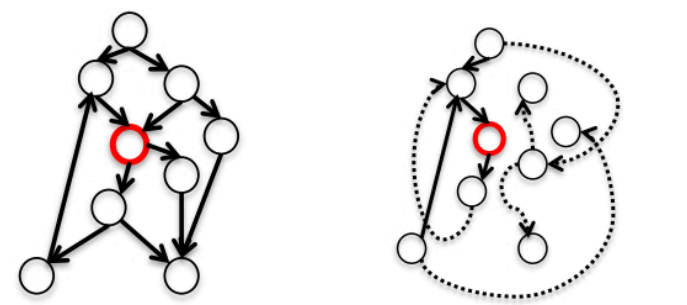
[https://developer.android.com/about/dashboards/index.html]

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.

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



CFG exhibited in vetting process. Red node represents the vulnerabilities.

Dynamic CFG in victim's device after exploiting vulnerabilities.

Figure 1: High Level Intuition