# CSE 484 / CSE M 584:
# Mobile Platform Security

Fall 2022

Franziska (Franzi) Roesner
franzi@cs

# Announcements

- **Lab 2:** Due tomorrow

- **Final Project:** Checkpoint #2 due Wednesday

- **Homework 3:** Out tomorrow, due December 9

- **Guest lecture** on Friday
  - Charlie Reis from Google Chrome

# 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 might this differ from desktop settings?

# What can go wrong?

*"Threat Model" 1: 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

Android flashlight app tracks users via GPS, FTC says hold on

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

# What can go wrong?

*Threat Model 1: Malicious applications*

Example attacks:

– Premium SMS messages

– Track location

– Record phone calls

– Log SMS

– Steal data

– Phishing

Some of these are unique to phones (SMS, rich sensor data)

# What can go wrong?

*Threat Model 2: Vulnerable applications*

Example concerns:

– User data is leaked or stolen

  • (on phone, on network, on server)

– Application is hijacked by an attacker

# Mobile Platform Security

- Mobile platform security is actually *really good*!
- Why?

# 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),5
02(access_bpf),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_apps
erveradm),98(_lpadmin),33(_appstore),100(_lpoperator),204(_developer),395(com.appl
e.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

# Why isolate on mobile devices and not PCs?

- Application isolation is *great!*

- **Phones** drew lessons from desktops

- **Desktops** draw lessons from phones
  - App Isolation sometimes available for PCs
    - Windows 10 Sandbox (May 2019)

- **Browsers** learning too
  - Site Isolation **[guest lecture on Friday!]**

# 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.)
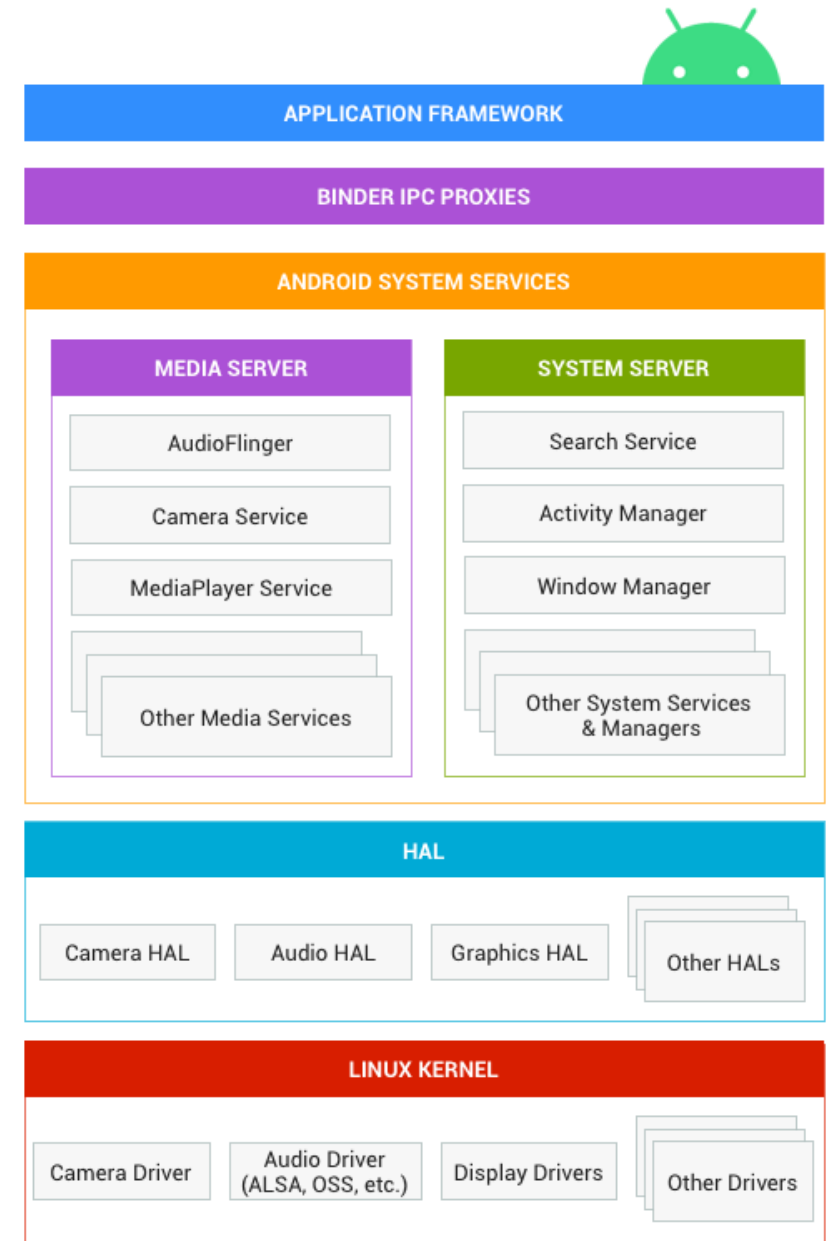    → allows rooting



**Figure 1.** Android system architecture

# 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).
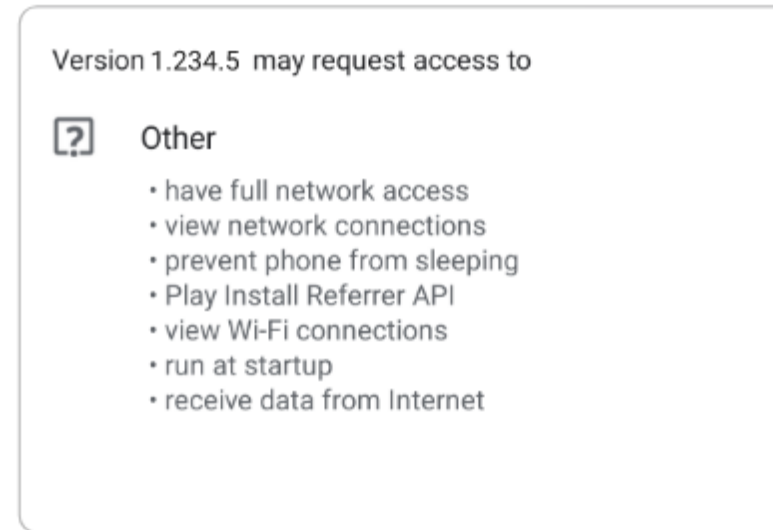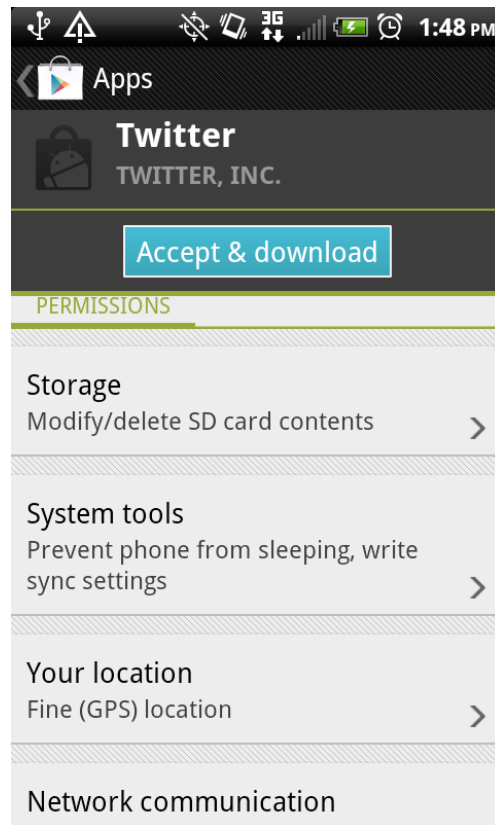- Devices (camera, GPS, phone, ... ).

> How should operating system grant permissions to applications?

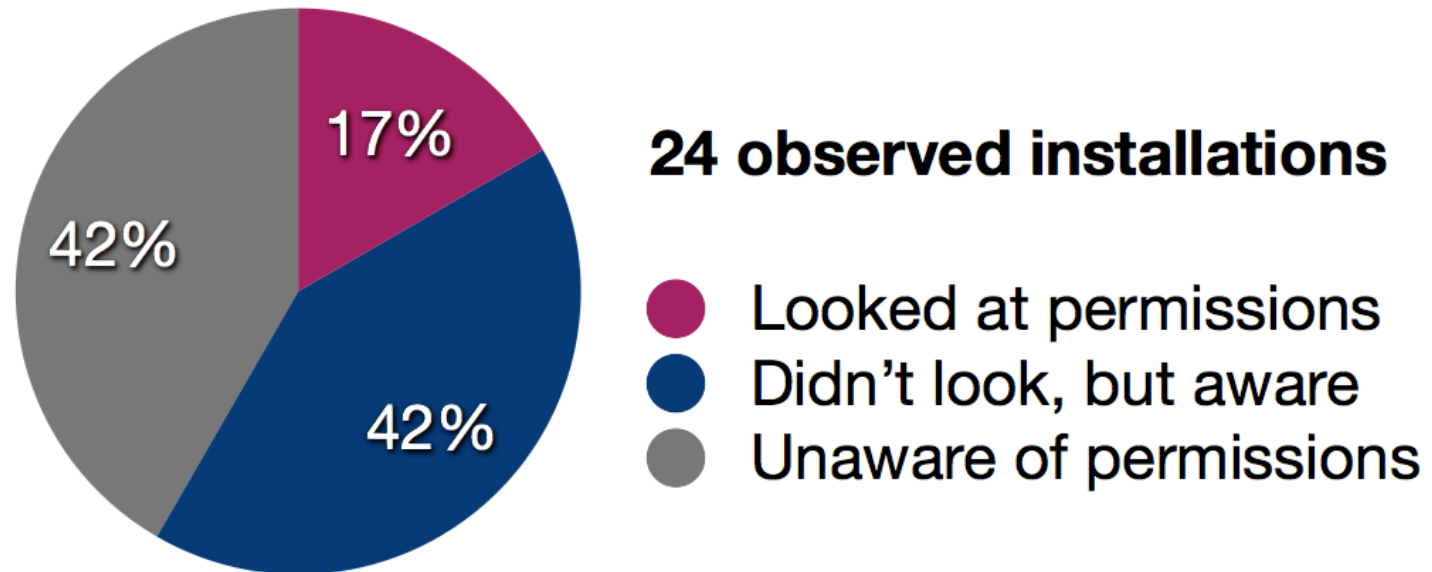Standard approach: Ask the user.

# Android's Old Approach: Manifests

- Big list of things the app wants at install time

# Are Manifests Usable?

Do users pay attention to permissions?



**24 observed installations**

- 🔴 Looked at permissions
- 🔵 Didn't look, but aware
- ⚪ Unaware of permissions

17% — 42% — 42%

… but 88% of users looked at reviews.

# 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, $n = 302$)
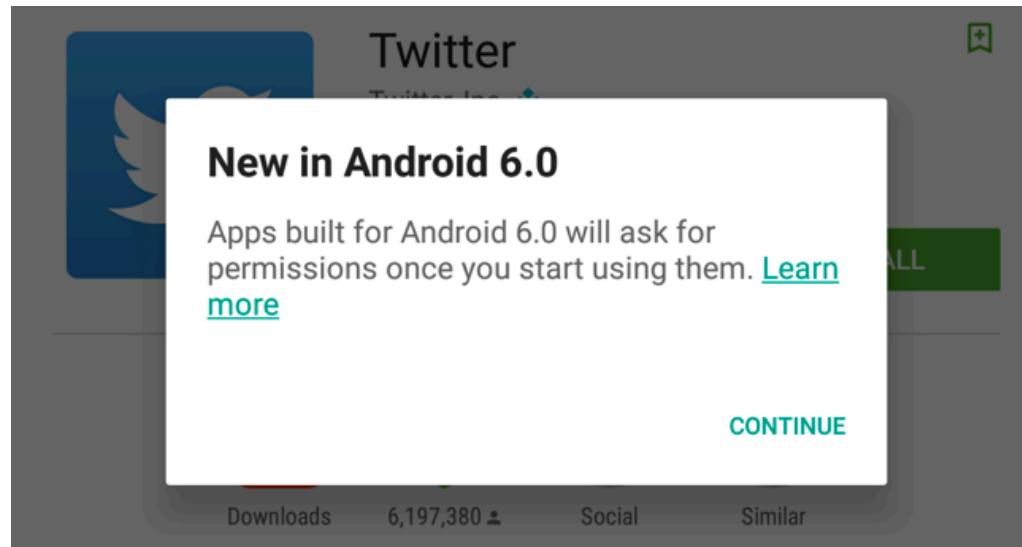
# Are Manifests Usable?

## Do users act on permission information?

"Have you ever not installed an app because of permissions?"
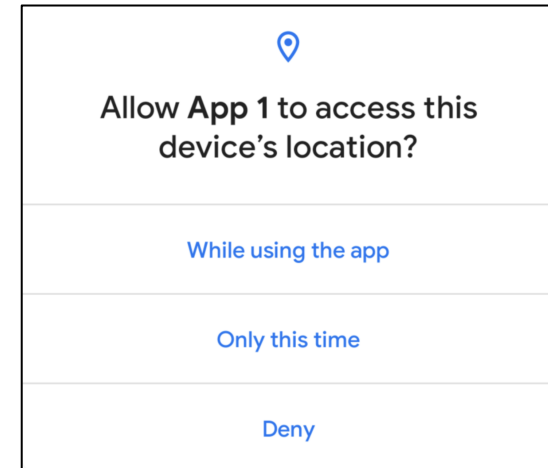
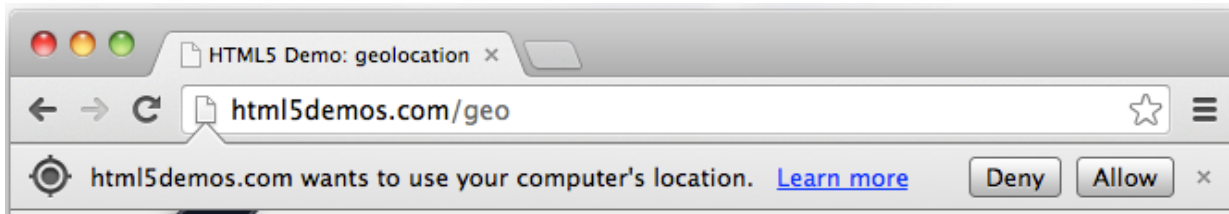8% 20%

**25 interview responses**

72%

- Yes
- No
- Probably
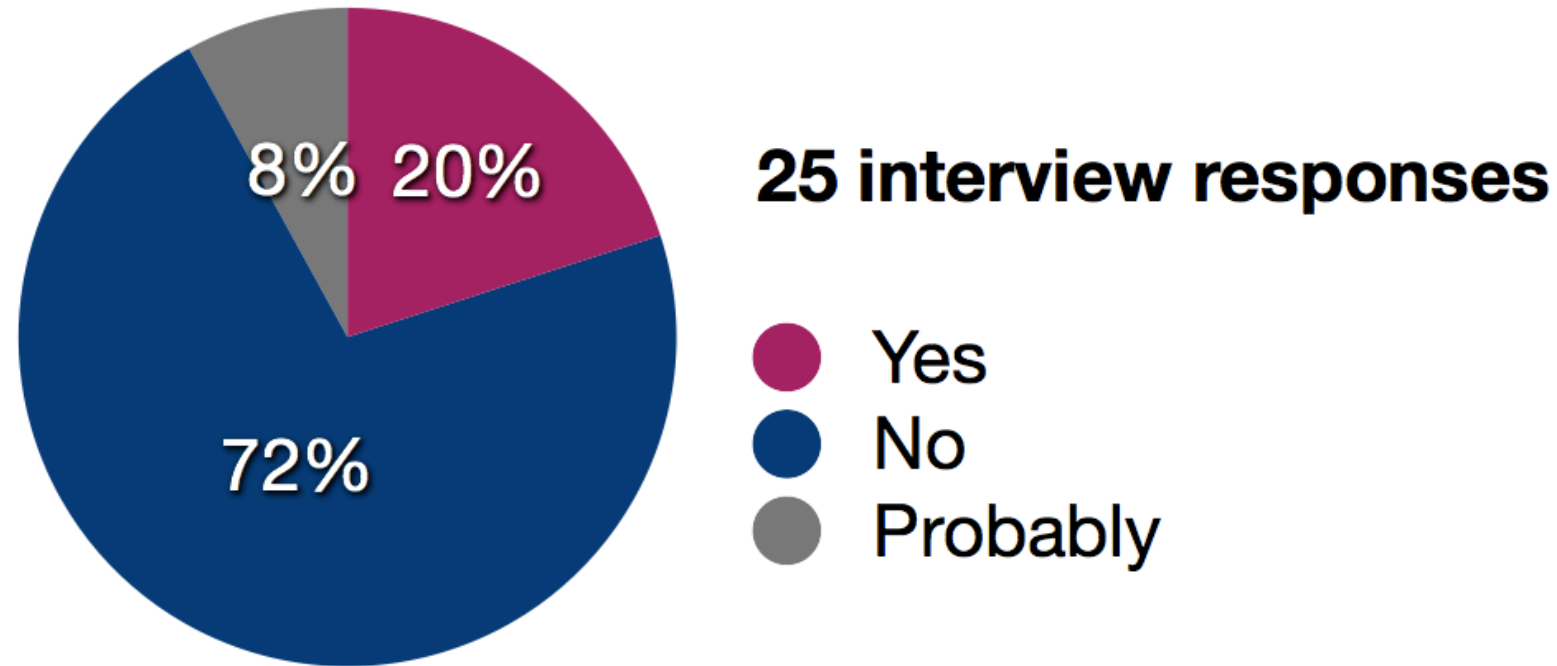
# Android 6.0: Prompts!
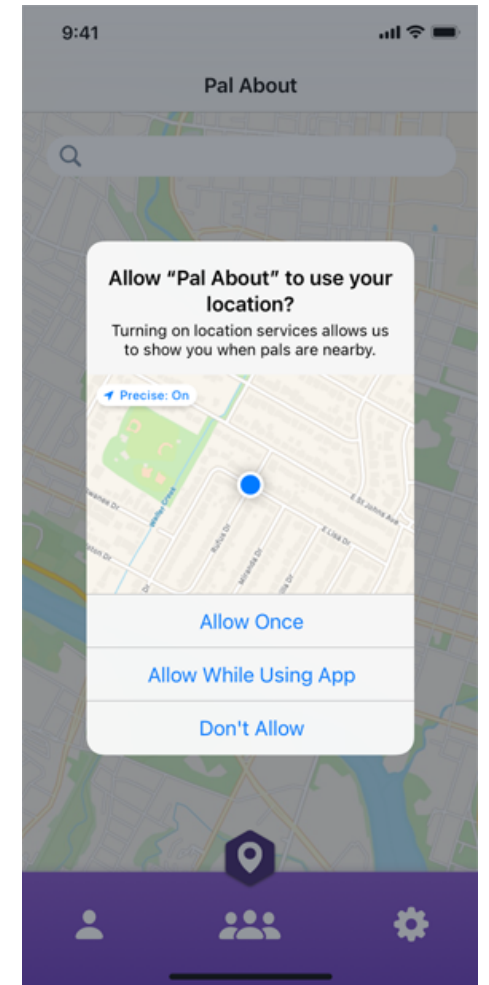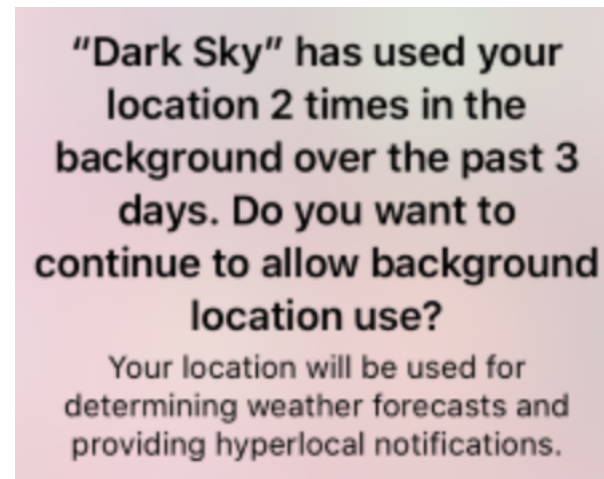


- First-use prompts for sensitive permission (like iOS always had).
- Big change! Now app developers needed to check for permissions or catch exceptions.

# State of the Art

- Time-of-use prompts



- More context in UIs

- Proactive monitoring of permissions usage

- Lots of privacy features+settings

Allow **App 1** to access this device's location?

While using the app

Only this time

Deny

"Dark Sky" has used your location 2 times in the background over the past 3 days. Do you want to continue to allow background location use? Your location will be used for determining weather forecasts and providing hyperlocal notifications.

Allow "Pal About" to use your location? Turning on location services allows us to show you when pals are nearby.

Allow Once

Allow While Using App

Don't Allow

# Still not a totally solved problem…

- Prompts are disruptive and lead to "prompt fatigue"
- In practice, prompts are still overly permissive
  - Apps can use permissions for other purposes and at other times

# (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.

- Settings application is deputy: has permissions, and accidentally exposes APIs that use those permissions.
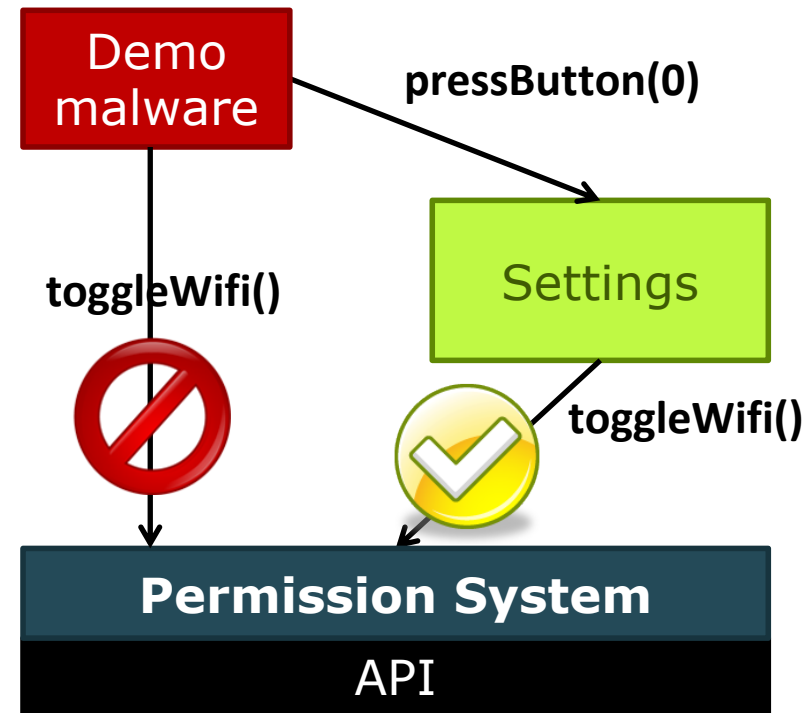
# Aside: Incomplete Isolation

Embedded UIs and libraries always run with the host application's permissions! (No same-origin policy here…)

## Thousands of Android apps have old security flaws lurking inside

Apps with millions of downloads are using code libraries with vulnerabilities in them, including some created by Facebook, Alibaba and Yahoo

# Other Android Security Features

- Secure hardware

- Full disk encryption

- Modern memory protections (e.g., ASLR, non-executable stack)

- Application signing

- App store review

# 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
  - Many changes made in past few years (e.g. Project Treble)

| Android Platform Version (API Level) | Distribution (as of April 10, 2020) |
|---|---|
| Android 4.0 "Ice Cream Sandwich" (15) | 0.2% |
| Android 4.1 "Jelly Bean" (16) | 0.6% |
| Android 4.2 "Jelly Bean" (17) | 0.8% |
| Android 4.3 "Jelly Bean" (18) | 0.3% |
| Android 4.4 "KitKat" (19) | 4% |
| Android 5.0 "Lollipop" (21) | 1.8% |
| Android 5.1 "Lollipop" (22) | 7.4% |
| Android 6.0 "Marshmallow" (23) | 11.2% |
| Android 7.0 "Nougat" (24) | 7.5% |
| Android 7.1 "Nougat" (25) | 5.4% |
| Android 8.0 "Oreo" (26) | 7.3% |
| Android 8.1 "Oreo" (27) | 14% |
| Android 9 "Pie" (28) | 31.3% |
| Android 10 (29) | 8.2% |

# What about iOS?

- Apps are sandboxed

- Encrypted user data
  - Often in the news…

- "App Tracking Transparency" prevents cross-app tracking

- App Store review process is (was? 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