Guest Lecture

Secure Android Application Development

March 2020

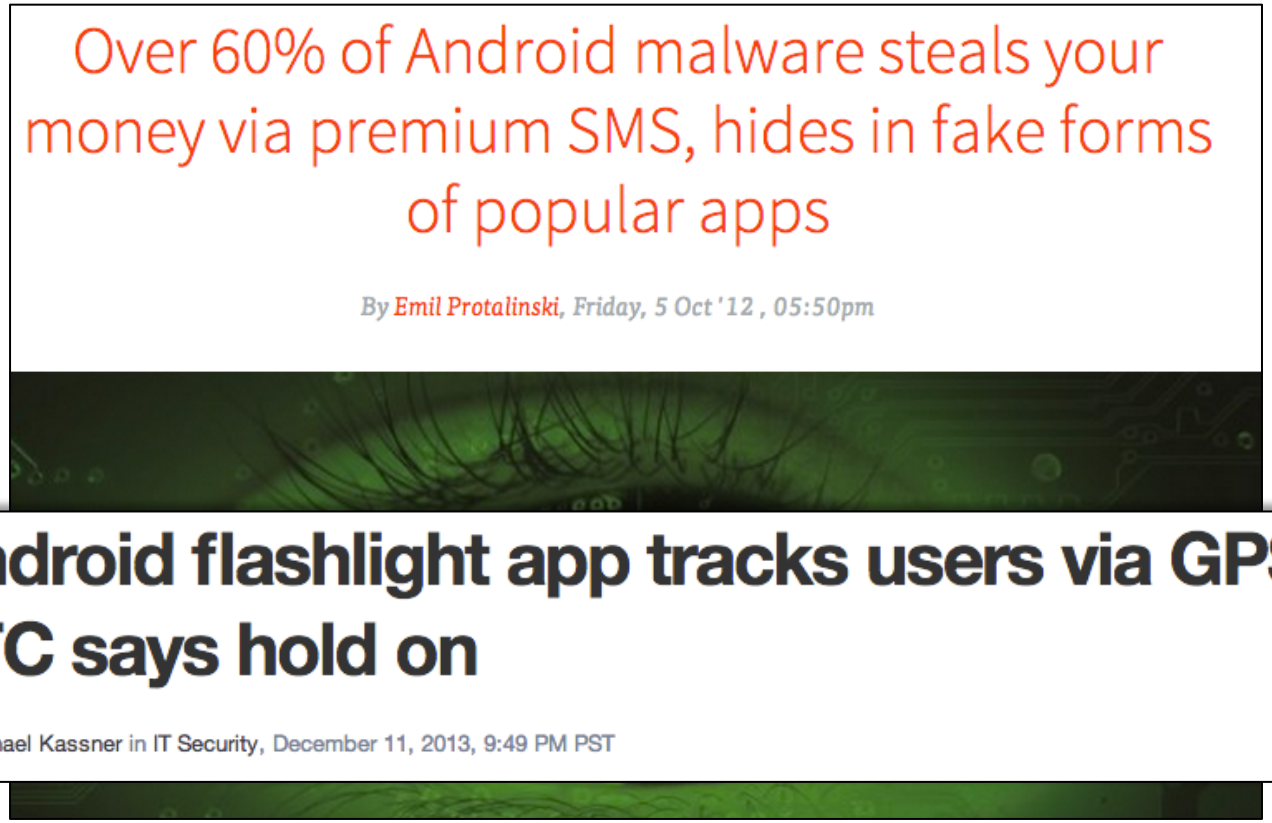
Franziska (Franzi) Roesner
Associate Professor, CSE
franzi@cs.washington.edu

Roadmap

- **Part 1:** What can go wrong?
- **Part 2:** How are mobile platforms (Android) designed for security?
- **Part 3:** Best practices for mobile (Android) app developers

What can go wrong?

"Threat Model" 1: Malicious applications



What can go wrong?

"Threat Model" 1: Malicious applications

Example attacks:

- Premium SMS messages
- Track location
- Record phone calls
- Log SMS
- Steal data
- Phishing

Tip: Don't do these things! :)

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 Features

Goal: Limit how much harm developers of malicious or buggy applications can do!

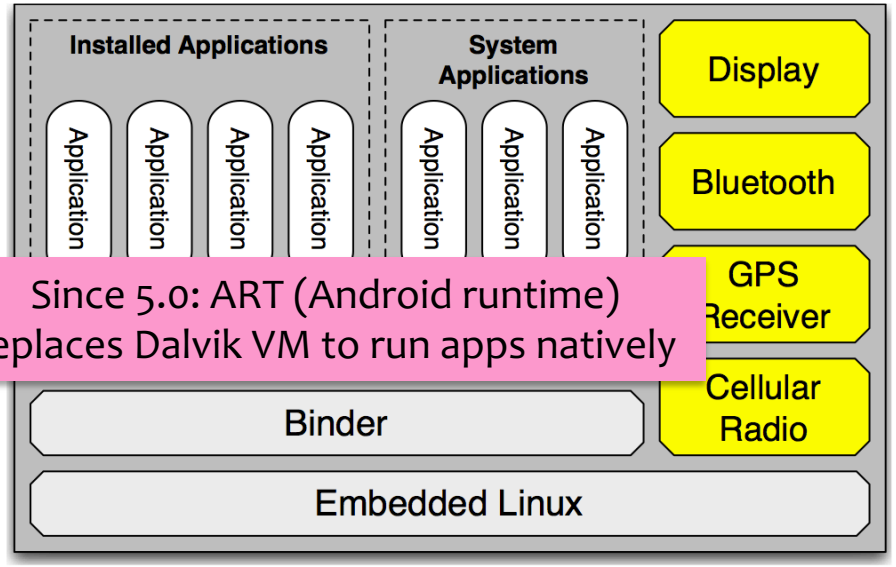
A key feature: Application isolation

Also:

- Secure hardware
- Full disk encryption
- Modern memory protections (e.g., ASLR)
- Application signing
- App store review

Applications are Isolated

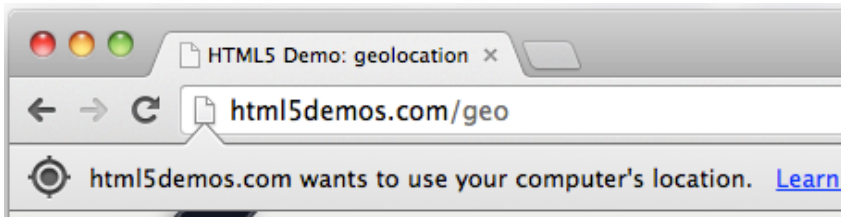
- **From each other**
  - Run in separate processes
  - With separate UIDs



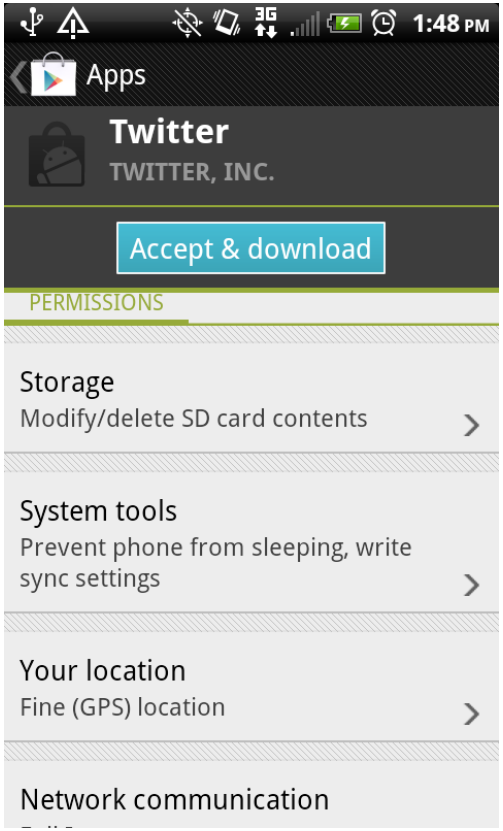
- **And from the system**
  - No shared accessible file system
  - No default access to devices

Permissions

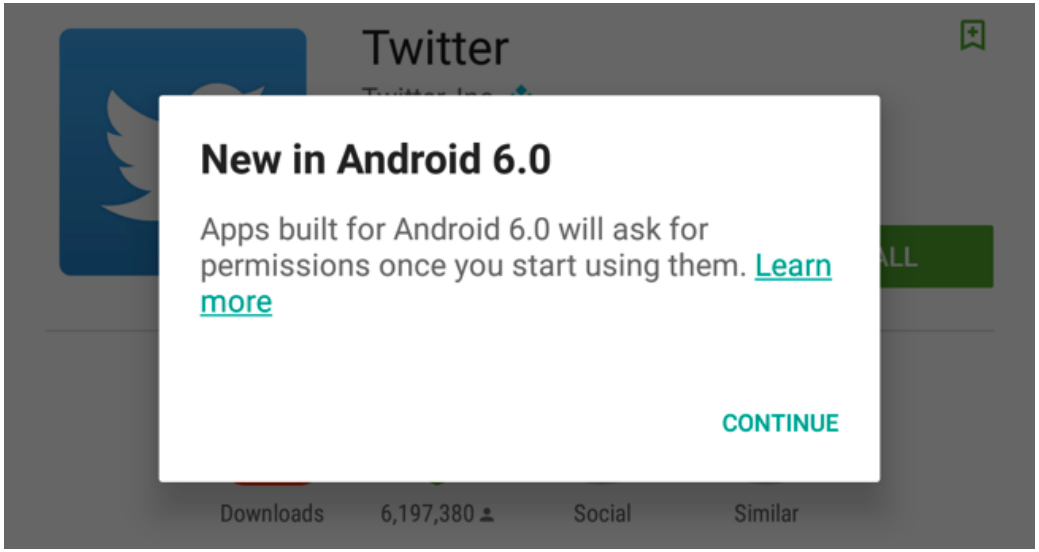
Prompts (time-of-use)



Manifests (install-time)



Android 6.0: Prompts!



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

Best Practices for Mobile App Developers

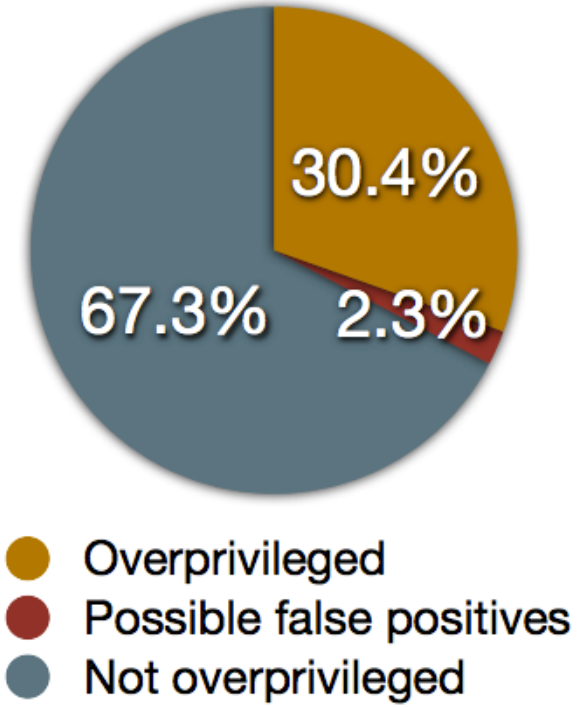
1. Only ask for the permissions you need
2. Use "internal storage" for sensitive data
3. Validate input from external sources (incl. users)
4. Encrypt network communications
5. Don't hard-code secrets in source code
6. Use existing cryptography support (carefully)
7. Be careful with inter-process communications
8. Be careful about libraries you include

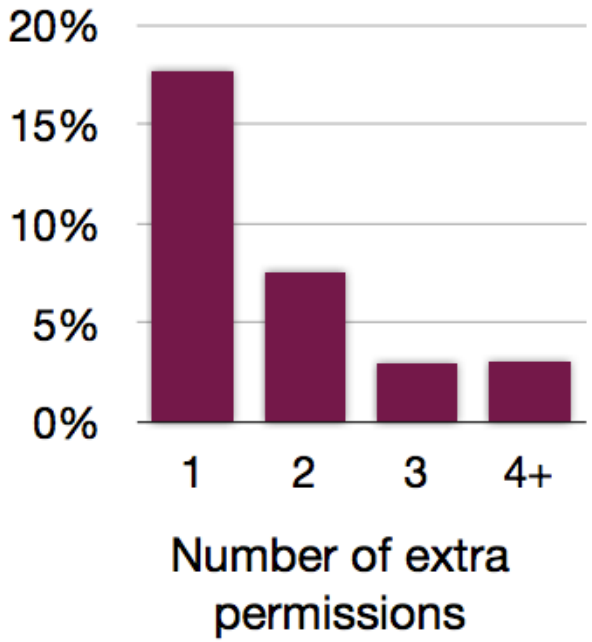
More: https://developer.android.com/training/articles/security-tips

[Felt et al.]

1. Only ask for permissions you need

Apps are often "overprivileged". Early (2011) study:





2. Use "internal storage" for sensitive data

Internal storage is:

- Not accessible to other apps
- Deleted when the app is installed

External storage (like SD cards) are globally readable and writable.

Even better, encrypt data (`EncryptedFile`).

3. Validate input from external sources (including users)

Check your assumptions about input!

- From users, from other apps, from web
- Length? Format?
- Can cause issues like buffer overflow attacks (in native code, not Java) or SQL injections (when sent to server)
- Be careful with WebViews

4. Encrypt network communications

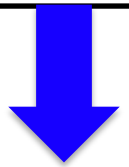
Use HTTPS! This means user data will not be readable over the network.

```
URL url = new URL("https://www.yourserver.com");
HttpsURLConnection urlConnection =
   (HttpsURLConnection) url.openConnection();
urlConnection.connect();
InputStream in = urlConnection.getInputStream();
```

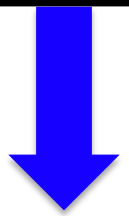
5. Don’t hard-code secrets in source code

CryptoHelper.java

```
private static String SECRET_KEY = "8badcafef00ddead";
```



GreatApp.apk



CryptoHelper.smali

```
const string v0, "8badcafef00ddead”
sput-object v0, Ledu/washington/cs/cse484/simplenotepad/
    CryptoHelper;->SECRET_KEY:Ljava/lang/String;
```

Instead, use Android Keystore

Allows you to create and store secret values.

Prevents extraction of keys by:

1. Making sure they never enter the application process
2. Use of secure hardware

6. Use existing cryptography libraries (carefully)

Do not write your own cryptography!

Unless you are a cryptographer ☺

Follow recommended best practices for parameter selections:
https://developer.android.com/guide/topics/security/cryptography

7. Be careful with inter-process communications

- 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.]

Eavesdropping and Spoofing

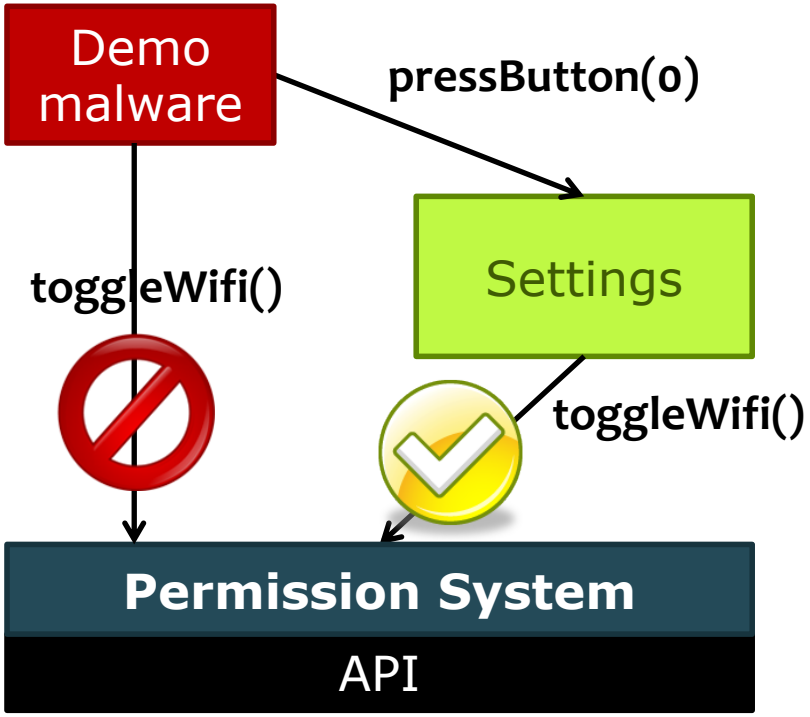
- Buggy apps might accidentally:
  - Expose their component-to-component messages publicly → eavesdropping
  - Act on unauthorized messages they receive → spoofing

[Felt et al.]

Permission Re-Delegation

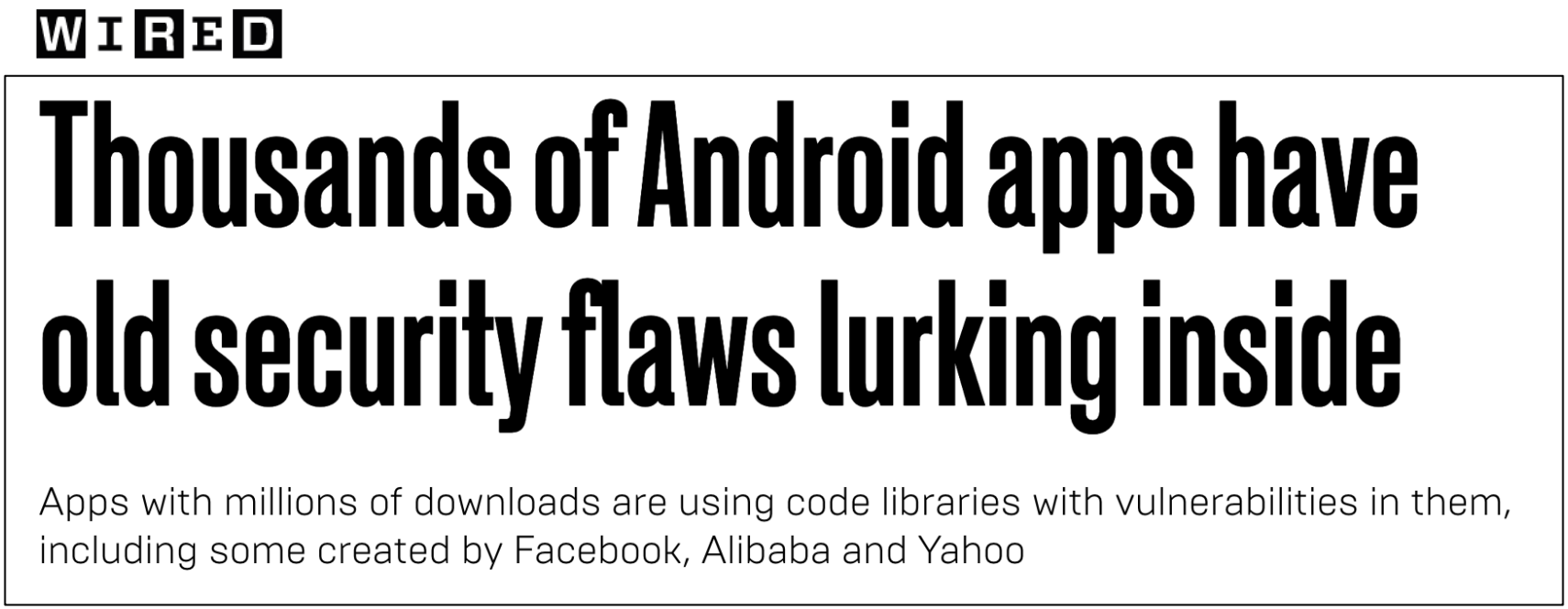
- An application without a permission gains additional privileges through another application.

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



8. Be careful about libraries you include

Libraries run in the context of the application
–> they can use all the same permissions!


WIRED

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

Best Practices for Mobile App Developers

Questions?

1. Only ask for the permissions you need
2. Use "internal storage" for sensitive data
3. Validate input from external sources (incl. users)
4. Encrypt network communications
5. Don't hard-code secrets in source code
6. Use existing cryptography libraries (carefully)
7. Be careful with inter-process communications
8. Be careful about libraries you include

More: https://developer.android.com/training/articles/security-tips