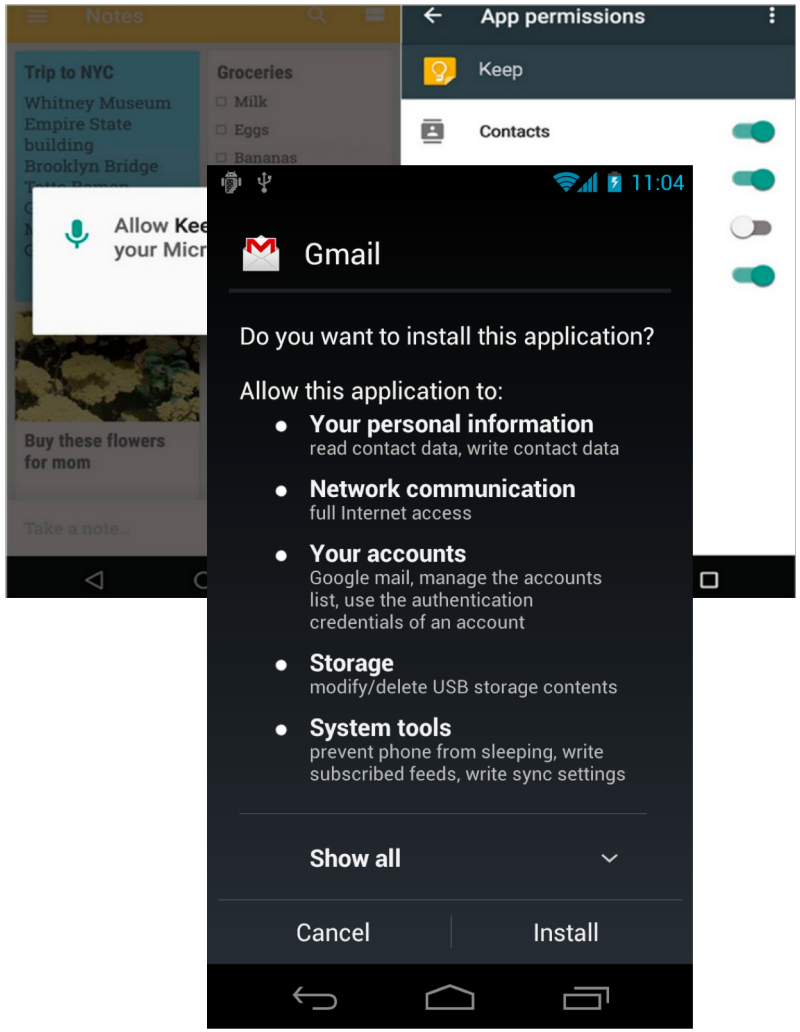
EECS 448 Smartphone Security
Android Permissions

Xusheng Xiao

Electrical Engineering and Computer Science
Case Western Reserve University

Last Lecture: Android Permissions

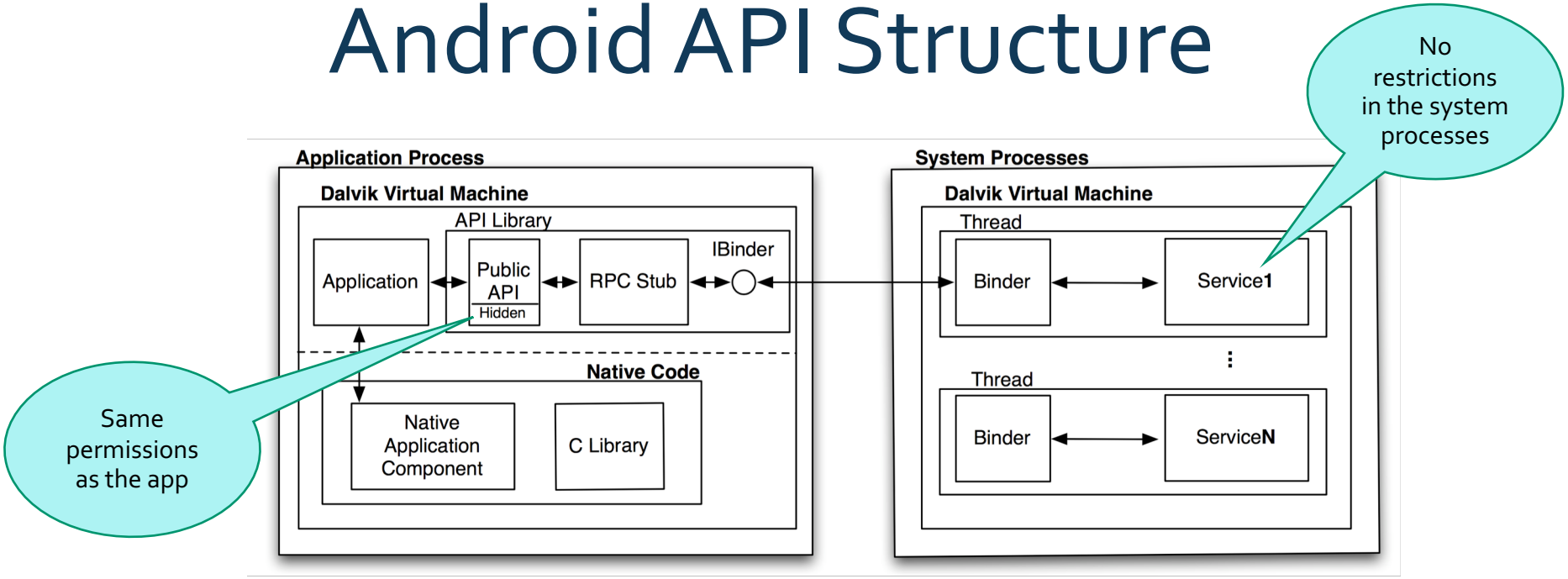
- Normal permissions
  - Automatically granted by the system
- Dangerous permissions
  - Version 6.0+ (23+): granted by users at run time
    - Revoked by any time in the setting.
  - Version <6.0 (<23): granted by users at installation time
    - New permissions granted during update
    - Revoked by uninstalling the apps



Categorize the System Permissions

- Normal Permissions
  - Permissions that are automatically granted by the system.
  - e.g., INTERNET, NFC, SET_ALARM
- Dangerous Permissions
  - Permissions that require users' confirmation. Managed as "Permission Groups"
  - e.g., CALL_PHONE, READ_EXTERNAL_STORAGE
- Signature/System Permissions
  - Permissions that will only grant to the developers identified with the same signatures
- Root (Linux), Google's Backdoors (will not consider them)

Android API Structure



- Public APIs → Private APIs → RPC Stub → System Processes
  - E.g., ClipboardManager.getText() → IClipboard$Stub$Proxy → ClipboardService
- Problem: Java reflection to access hidden and private classes, which may change between releases

Enforcement (APIs)

- No centralized permission checks
- API libraries check permissions (insecure)
- System processes check permissions (secure)
- Some permissions are enforced by Unix groups
  - E.g., INTER- NET, WRITE_EXTERNAL_STORAGE, or BLUETOOTH
- Native code is mediated by Linux permissions
  - Open files or sockets are checked by Linux permissions
  - Cannot directly communicate with the system API
  - Can use Java wrappers but subject to app permissions

Enforcement (Content Providers)

```
<uses-permission android:name="android.permission.READ_USER_DICTIONARY">
```

- They are protected by both static and dynamic permission checks
- They support fine granularity access controls
  - e.g., Requires default permission to visit A, public access to "content://A/b" and permission P_2 to visit part "content://A/c"

```
<provider …>
      <path-permission android:pathPrefix="/subpath1"
      android:readPermission="com.example.SUBPATH1_READ_PERMISSION"
      android:writePermission="com.example.SUBPATH1_WRITE_PERMISSION" />
      <path-permission android:pathPrefix="/subpath2"
      android:readPermission="com.example.SUBPATH2_READ_PERMISSION"
      android:writePermission="com.example.SUBPATH2_WRITE_PERMISSION" />
</provider>
```

Enforcement (Content Providers)

- Developers can do it programmatically (explicitly calls system validation mechanism for incoming queries)

```
if (ContextCompat.checkSelfPermission(thisActivity,
Manifest.permission.WRITE_CALENDAR)
        != PackageManager.PERMISSION_GRANTED) {
    // Permission is not granted
}
```

Enforcement (Intents)

Sender:

```
Intent broadcast = new Intent(this, MyBroadcastReceiver.class);
sendBroadcast(broadcast, "android.demo.mypermission");
```

Receiver:

```
<uses-permission android:name="andro.demo.mypermission"/>
```

- Intent helps pass messages and interact between the code in different applications
- Risk: application A has been granted permission *p*, application B uses intents to access resources protected by *p* from A.
- ActivityManagerService: pass all the intents and enforce restriction policy.
  - Some intents can be sent only by apps with permissions
  - System intents can only be sent via system processes (check UID)

Enforcement (Package Installer)

- Signature/System Level Security: granted at install-time
- Check if certificate matches (won't bother users)

Permission Testing – Randoop

[Felt et al. CCS'11]

- Randoop, a tool that can generate test cases randomly by tracing the chain of method calls, with little modifications
- Use the return value as the parameter for the other methods
- Modification: log the methods checked by permission system
- Limitations:
  - 1. Relies on the types in the input pool, but Android has too many types
  - 2. Does not handle order issues (Android expects this, sometimes)
  - 3. Native methods may early-terminate the test.

Android Permissions Demystified
Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner.
ACM CCS 2011.

Permission Testing - Discover the Android's Access Control Policy

- Customizable Test Case Generation
  - Tool: their own tool
  - How it works: automatically generate test cases from a list of methods and their corresponding parameters
    - ✓: high coverage, allows manual adjustment ✗: needs human effort
- Manual Verification
  - Solves: Asynchronous API call & Permission might be argument-dependent
  - HOWTO:
    - use Randoop and their tools to generate test cases (and permission maps)
    - manually verify the correctness of these maps
    - use cases from the customized tool to confirm

Permission Testing of Content Provider

- Perform four actions: query, insert, update and delete
- URI resource: android.provider package
- HOWTO: simple — execute each type of the database operation with or without permission and log the security exception.
- Run until no security exceptions appeared

Permission Testing of Intents

- Researchers build a pair of applications
- Documentation does not provide a comprehensive list of system intents (bad Android documentation)
- Test system intents:
  - System broadcasts WiFi, Bluetooth, Phone Calls, etc.
  - Check if permission check occurs when applications receive the broadcast and intents deliver are delivered successfully.
  - Run until no security exceptions appeared

Permission Map — Which API relates to which permission

- Why: Android documentation is unclear
- Coverage: 85% (out of 1665 classes with 16732 methods)
- Reveals: Android Secret Menu…
  - e.g. the documentation claims that a method is protected by a dangerous permission MANAGE_ACCOUNTS, but it can be accessed by a normal permission GET_ACCOUNTS.
  - or, the documentation claims some of the permission that do not exist

Permission	Usage
BLUETOOTH	85
BlUETOOTH_ADMIN	45
READ_CONTACTS	38
ACCESS_NETWORK_STATE	24
WAKE_LOCK	24
ACCESS_FINE_LOCATION	22
WRITE_SETTINGS	21
MODIFY_AUDIO_SETTINGS	21
ACCESS_COARSE_LOCATION	18
CHANGE_WIFI_STATE	16

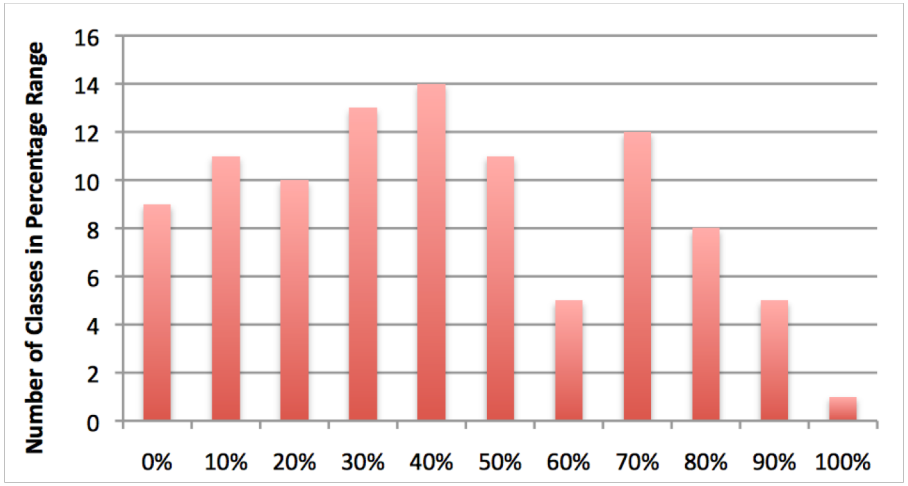
10 most checked permissions

Statistics — What does the Permission Map tell us

- 1244 (6.45%) API calls check permissions + 15 API calls customized by manufacturers.
- 12% API calls are protected by Signature Permission - A good signal
- Some permissions are unused
  - One killer permission — BRICK (permanently disable the device)

Statistics — What does the Permission Map tell us (Cont'd)

- Does hierarchical permission exists? Yes or No
  - Having CHANGE_WIFI_STATE will not grant you ACCESS_WIFI_STATE. This applies to all permissions
  - Except ACCESS_COARSE_LOCATION (automatically granted if you have ACCESS_FINE_LOCATION)
- Permission Granularity - A double edged sword
  - Pros: Provide a higher level security for users
  - Cons: Confuse developers (Two Bluetooth permissions are applied to 6 large classes)



Statistics — What does the Permission Map tell us (Cont'd)

- Content Providers: 18/62 methods are open for all four actions
- They all belong to content://media URI
- Some broadcasts are prohibited; some require permission before sending the intent; some will not have responses
- Most of the restricted receivers are protected by Bluetooth permission (14/23)
- 7 intents require permission for start Activity, 2 for Services

Stowaway — A Static Analysis Tool

- Why do we need it: Developer Errors & Security Audit
- What to feed Stowaway: disassembled Dalvik Executables (DEX), supported by Dedexer tool (open source)
- Handles API calls, Content Providers and Intents

Stowaway on API Calls

- Normally, Stowaway tracks the calls on API methods from disassembled codes. It is enough, but…
- Reflection
  - Tracks the Class obj and getters for Method, Constructors, etc.
- Internet & Storage
  - Storage permission cannot be mapped because it is enforced by Linux system and associated with many file interfaces
  - WebView needs Internet permission and can be initiated from XML
  - Thus Stowaway checks XML file (string matching)

Stowaway on Content Providers

- Stowaway checks the API methods that returns a string of URI (e.g., android.net.Uri.parse(String)).
- It checks all string literals that begin with content://
- It also checks the helper class
  - e.g., String *.BOOKMARKS_URI=content://browser/bookmarks
- Big Limitation: Does not know what action the application performs with the URI

Stowaway on Intents

- With the help from ComDroid (also developed by UC Berkeley people)
- ComDroid records the inter application communication. Stowaway reads its output and sees whether a permission is required for the intent
- Occasionally ComDroid does not work; Stowaway checks if the string literal is in the protected list
- Stowaway checks the permission to send and to receive intent

Evaluation - terms

- Overprivileged - Stowaway calculates the maximum set of the Android permissions needed (Question: How?)
- False Positive
  - Stowaway misses an API, Content Provider or Intent
  - Stowaway cannot find a permission map with the entry from the target
  - Non-system applications and their customized permission

Evaluation - Results

- Test 940 applications
  - 40 by manual analysis
  - 900 by Stowaway
- 7% of the false positive rate
  - Actually, it is only caused by the incompleteness of the map (Runtime.exec & web-embedded codes)
- 35% are overprivileged

Why Unnecessary Permissions

- ``Related'' Permission names
  - MOUNT_UNMOUNT_FILESYSTEMS for android.intent.action.MEDIA_MOUNTED Intent
  - ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE permissions
    - ACCESS_NETWORK_STATE for Internet connection, e.g., celluar network or wifi
    - ACCESS_WIFI_STATE for info about WIFI
- Deputy
  - One application asks the Android Market to install another application. The sender asks for INSTALL_PACKAGES
  - Ask for CAMERA while sending intent to Camera app

Permissions: https://developer.android.com/reference/android/Manifest.permission
Intents: https://developer.android.com/reference/android/content/Intent

Why Unnecessary Permissions

- Related methods
  - Setters in android.provider. Settings.Secure need WRITE_SETTING, but getters do not
- Copy and Paste
  - Registers to receive the android.net.wifi.STATE_ CHANGE Intent and requests the ACCESS_WIFI_STATE permission.
- Deprecated Permissions
  - ACCESS_GPS or ACCESS_LOCATION
- Testing Artifacts
  - ACCESS_MOCK_LOCATION
- Signature/System Permissions
  - Not working on standard handsets

Thank You !

Questions ?