PScout: Analyzing the Android Permission Specification

Kathy Au, Billy Zhou,
James Huang, David Lie

University of Toronto

ROGERS
DEPARTMENT OF
ELECTRICAL
& COMPUTER
ENGINEERING
UNIVERSITY of TORONTO

VELUT ARBOR ÆVO

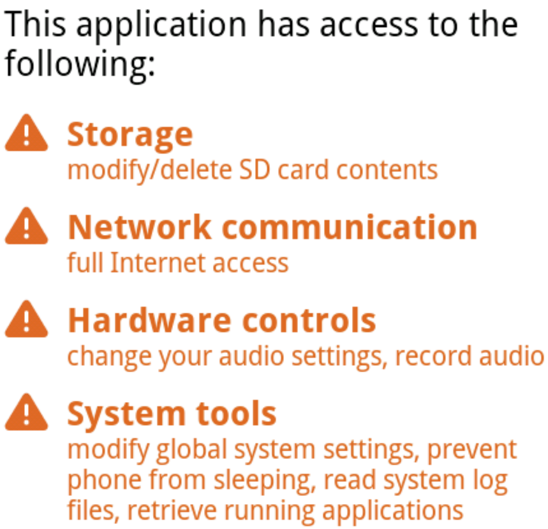
Smartphone Permission System

- Smartphones are loaded with sensors
  - GPS, camera, microphone, NFC, Wi-Fi radio, etc.
- Permission System
  - Access control to confine 3rd party applications
  - Implemented in **ALL** current major smartphone OSs
  - Android Permission System

A good understanding of permission systems is required to study smartphone security

Android Permission System

- Per-application access control policy
  - communicated at installation time
- 79 permission in Android 4.0
  - E.g. CHANGE_WIFI_STATE



This application has access to the following:

⚠ **Storage**
modify/delete SD card contents

⚠ **Network communication**
full Internet access

⚠ **Hardware controls**
change your audio settings, record audio

⚠ **System tools**
modify global system settings, prevent phone from sleeping, read system log files, retrieve running applications

Android Permission System

- API to Permission Mapping:
  - android.net.wifi.WifiManager.reassociate(); CHANGE_WIFI_STATE
  - android.telephony.TelephonyManager.getDeviceId(); READ_PHONE_STATE
- Complete mapping NOT available due to incomplete documentation

Key Questions

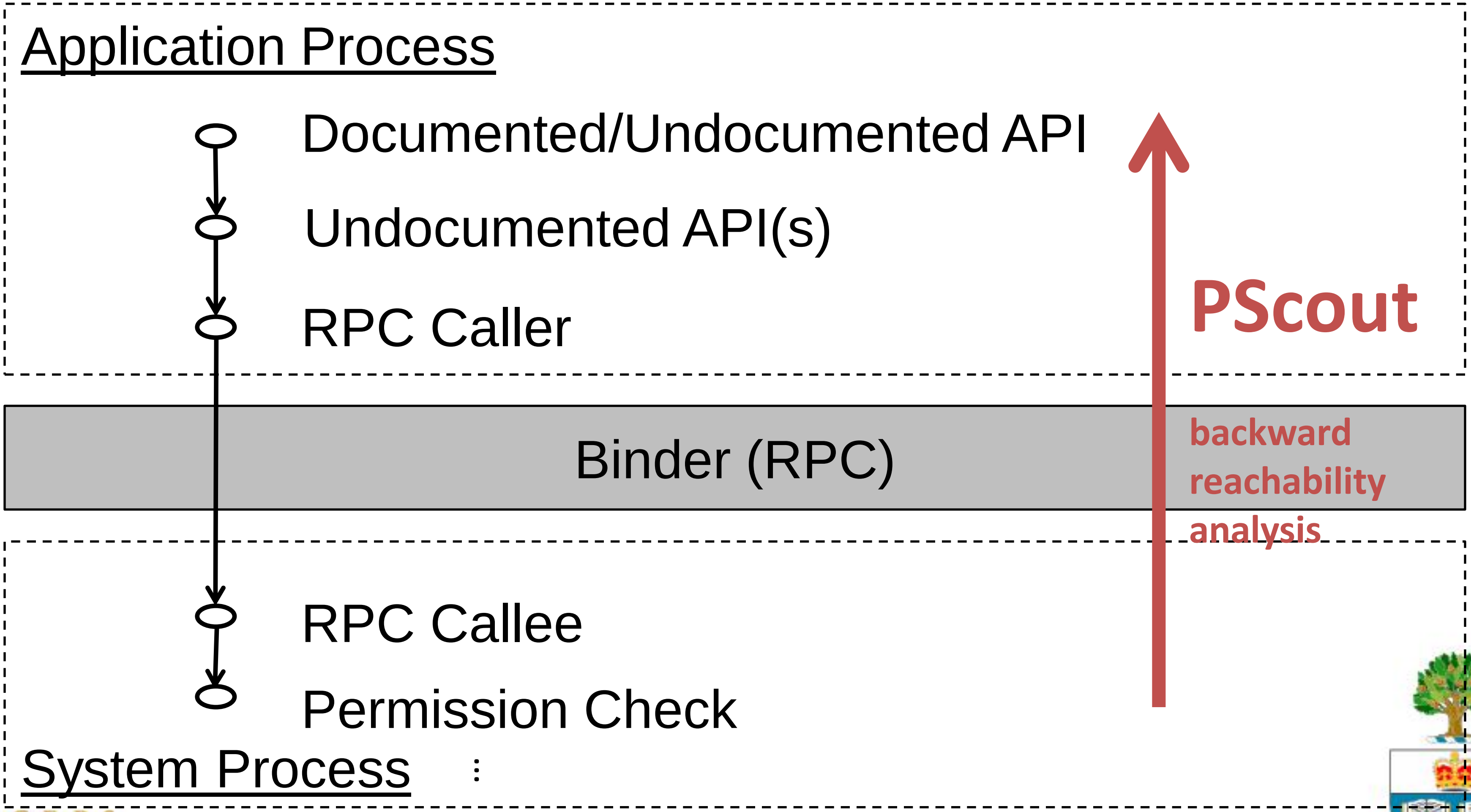
1. Are there any redundant permissions?
2. Are undocumented APIs used?
   - Undocumented APIs are APIs that are not listed in the Android API reference
3. How complex is the Android specification?
   - How are permission mappings interconnected?
4. How has it evolved over time?

API to Permission Mapping

- Most complete existing API to permission mapping [Felt et al., CCS 2011]
  - API fuzzing
  - Limitations: incomplete coverage, parameter generation, valid test sequences
- Difficult to reuse system for different Android versions due to manual effort required

Goal: A version-independent analysis tool that is more complete than existing tool

PScout: Overview



Application Process

Documented/Undocumented API

Undocumented API(s)

RPC Caller

Binder (RPC)

RPC Callee

Permission Check

System Process

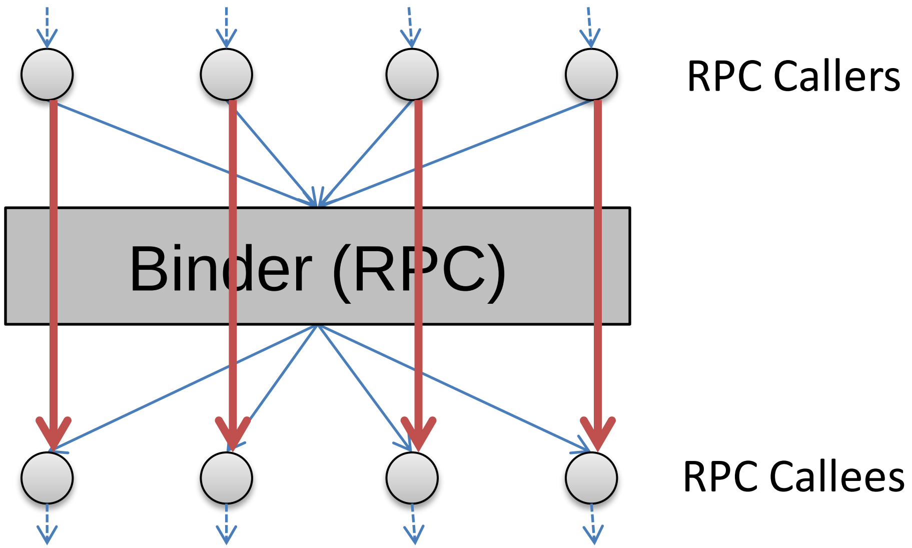
⋮

PScout

backward reachability analysis

PScout: Call Graph Generation

- Call Graph Generation
  - Entire Android framework
  - Refined with RPC/IPC information



Reachability: Starting Points

- *Permission Check* definition:
  - An execution point in the OS after which the calling application must have the required permission
- Three types:
  - Explicit calls to *checkPermission* functions
  - Accesses to specific content providers
  - Sending/receiving of specific intents

Reachability: Stopping Conditions

- Method caller ID is temporary cleared
  - Permission enforcement always pass when caller ID is cleared in system processes

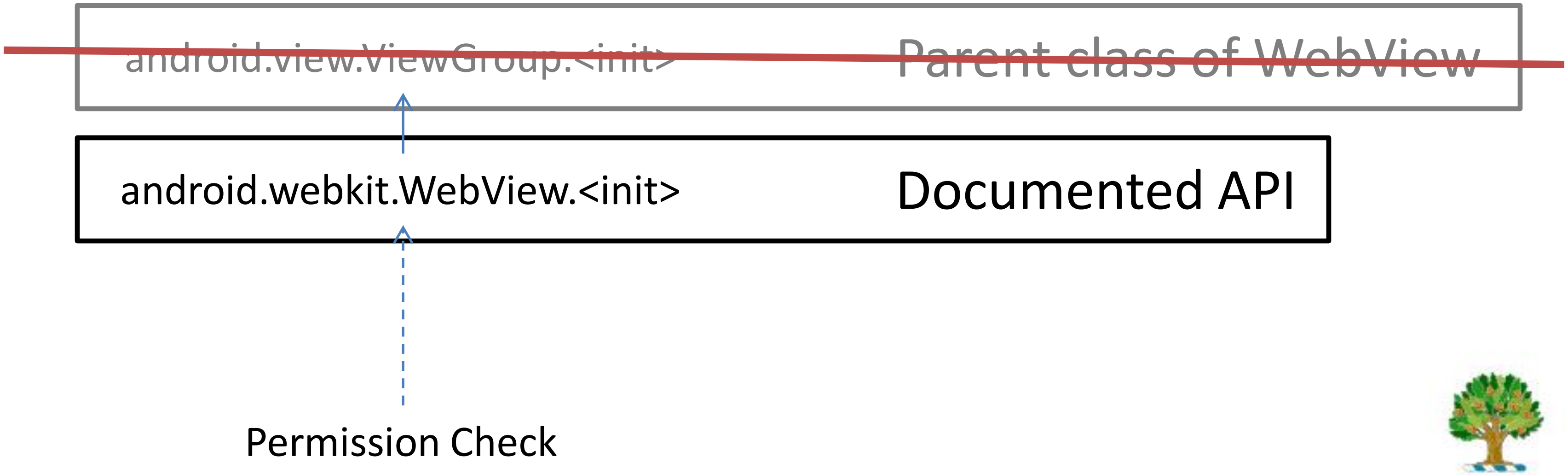
```
void Function() {
      clearCallingIdentity
      <enforce permission X>
      restoreCallingIdentity
}
```

Case 1:
Requires Permission X to proceed

Case 2:
Does not require permission to proceed

Reachability: Stopping Conditions

- Reached generic parent classes of documented APIs



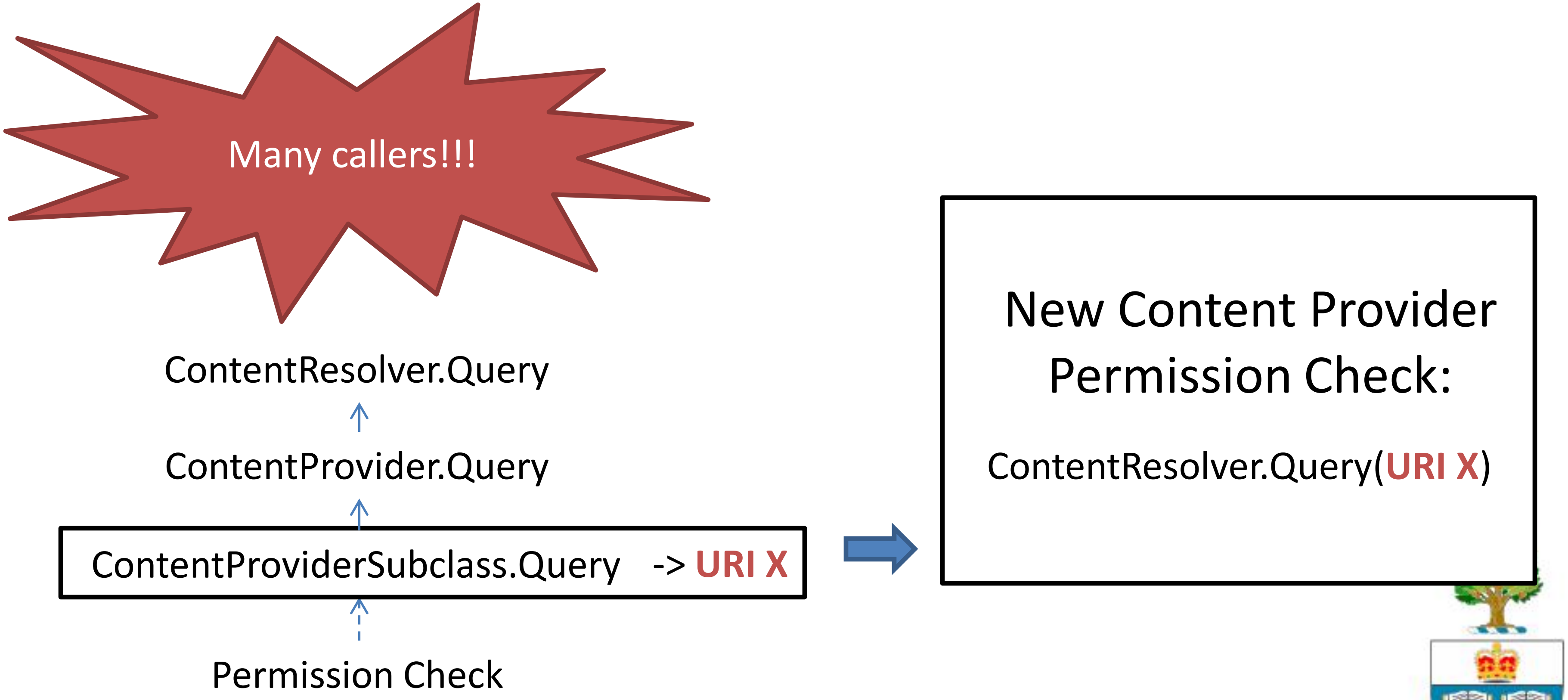
~~android.view.ViewGroup.<init>~~ ~~Parent class of WebView~~

android.webkit.WebView.<init> Documented API

Permission Check

Reachability: Stopping Conditions

- Reached Content Provider subclasses



Many callers!!!

ContentResolver.Query

ContentProvider.Query

ContentProviderSubclass.Query -> **URI X**

Permission Check

New Content Provider Permission Check:

ContentResolver.Query(**URI X**)

Key Questions

1. Are there any redundant permissions?
2. Are undocumented APIs used?
3. How complex is the Android specification?
4. How has it evolved over time?

Q1: Redundancy in Permissions?

- Conditional Probability
  - P(Y|X) = ?
  - Given an API that checks for permission X, what is the probability that the same API also check for permission Y?
  - 79 permissions -> 6162 pairs of permissions

Q1: Redundancy in Permissions?

- *Redundant Relationship*
  - Both permissions are always checked together
  - P(Y|X) = 100% and P(X|Y) = 100%
  - Only 1 pair found: KILL_BACKGROUND_PROCESSES and RESTART_PACKAGES
    - RESTART_PACKAGES is a deprecated permission

Q1: Redundancy in Permissions?

- *Implicative Relationship*
  - All APIs that check for permission X also checks for permission Y
  - P(Y|X) = 100% and P(X|Y) = ?
  - Found 13 pairs
  - Many write permissions imply read permissions for content providers
    - E.g. WRITE_CONTACTS implies READ_CONTACTS

Q1: Redundancy in Permissions?

- *Reciprocative Relationship*
  - The checking of either permission by an API means the other permission is also likely checked
  - P(Y|X) > 90% and P(X|Y) > 90%
  - Found 1 pair:
    ACCESS_COARSE_LOCATION vs.
    ACCESS_FINE_LOCATION
    - FINE is not a superset of COARSE permission
    - PhoneStateListener requires COARSE permission

Q1: Redundancy in Permissions?

- 15/6162 all possible pairs of permission demonstrates to have close correlation
- There is little redundancy in the Android permission system.

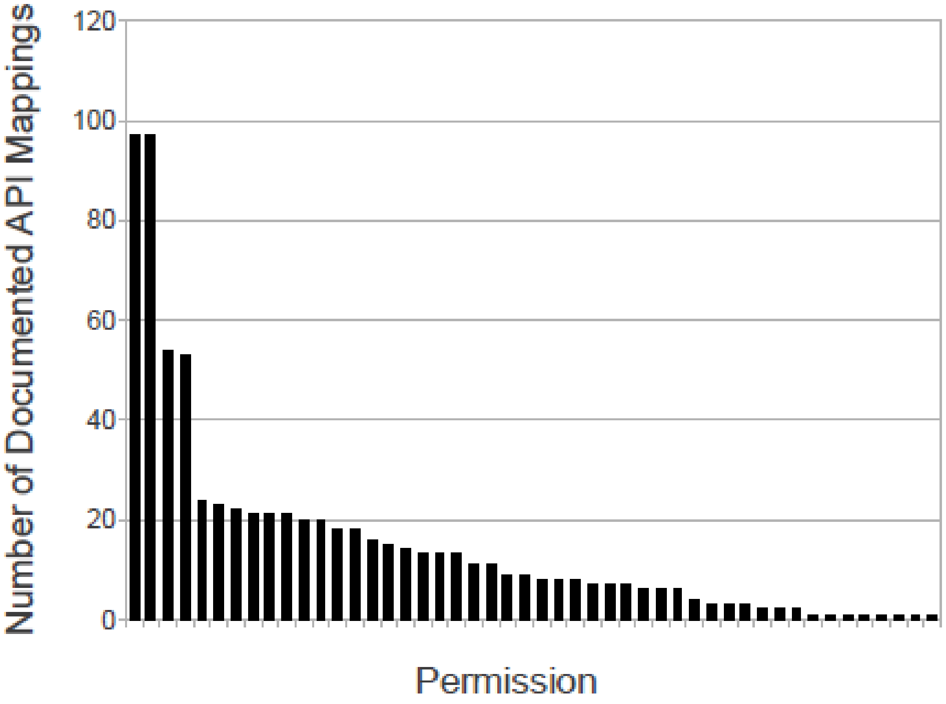
Q2: Undocumented API usage?

- 22-26% of the declared permissions are only checked through undocumented APIs
  - can be hidden from most developers
  - E.g. SET_ALWAYS_FINISH, SET_DEBUG_APP are moved to system level permission in Android 4.1

- 3.7% applications use undocumented APIs

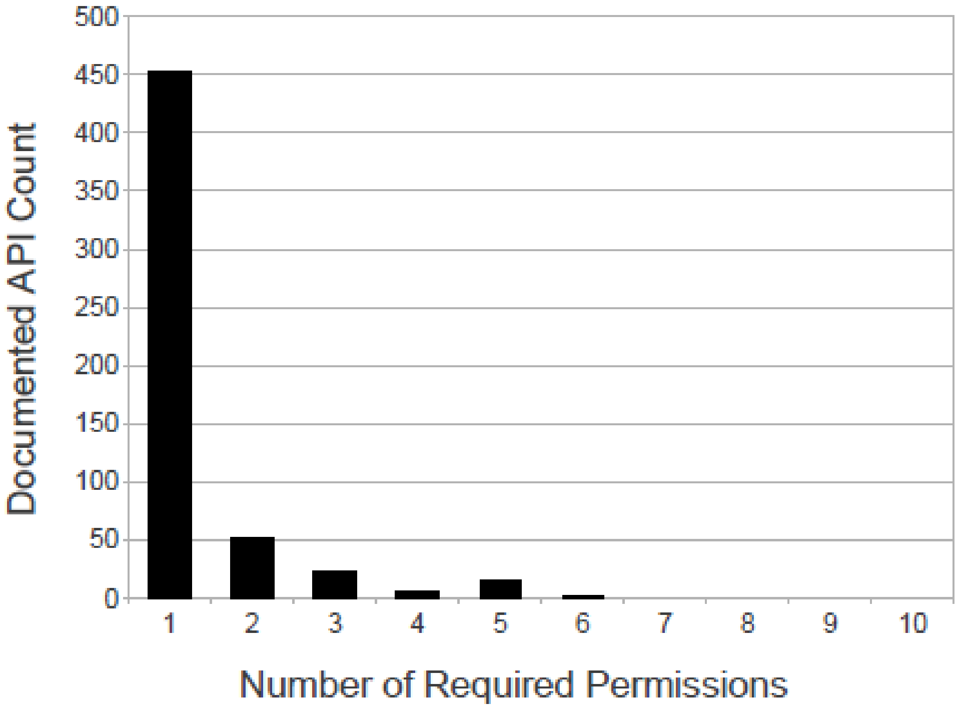
Undocumented APIs are rarely used in real applications, some permissions can be hidden.

Q3: Specification Complexity



- 75% of permission map to <20 API calls
- Permissions guards specific functionalities

Q3: Specification Complexity



- >80% APIs require only 1 permission, few need more than 3
- Sensitive APIs have relatively distinct functionality

Q3: Specification Complexity

- Few overlaps in the permission mapping
- Android permission specification is simple.

Q4: Changes over time?

- Permission checks grew proportionally with code sizes between 2.2 and 4.0
  - 2 KLOC per permission checks
- More sensitive functionality are exposed through documented APIs over time
  - New APIs introduced with permissions
  - Undocumented -> documented API mapping
  - Existing APIs + new permission requirements

Q4: Changes over time?

- Small changes can lead to permission changes
  - No fundamental changes in API functionality

CLASS: android.server.BluetoothService

```
public boolean startDiscovery() {
    if (getState() != STATE_ON) return false;
    try {
        return mService.startDiscovery();
    } catch (RemoteException e) {Log.e(TAG, "",
    return false;
}
```

Added in Android 2.3:
getState() also require
BLUETOOTH permission

Same between Android 2.2 and
Android 2.3:
startDiscovery() require
BLUETOOTH_ADMIN permission

Q4: Changes over time?

- Tradeoff between fine-grain permission and permission specification stability
  - – E.g. Combining the BLUETOOTH and BLUETOOTH_ADMIN permissions can prevent the permission change between 2.2 and 2.3 but reduces the least-privilege protection

Conclusion

- PScout extracts the Android permission specifications of multiple Android versions using static analysis.
  - Results show that the extracted specification is more complete than existing mappings
  - Error from static analysis imprecision is small
- There is little redundancy in the Android permission systems.
- Few application developers use undocumented APIs while some permissions are only required through undocumented APIs.
- There is a tradeoff between fine-grain permission and permission specification stability.

Getting PScout

PScout source code and the permission mappings for Android (2.2/2.3/3.2/4.0/4.1) are available for download at:

http://pscout.csl.toronto.edu