

Enabling Fine-Grained Access Control for Android Apps

Yao Guo
Peking University

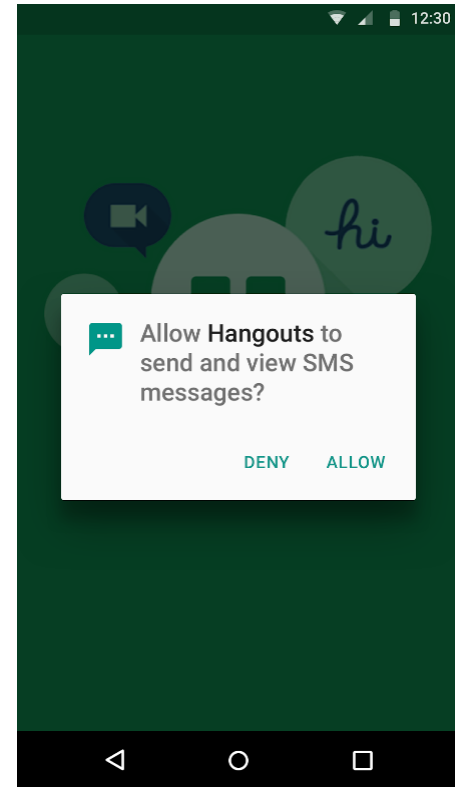
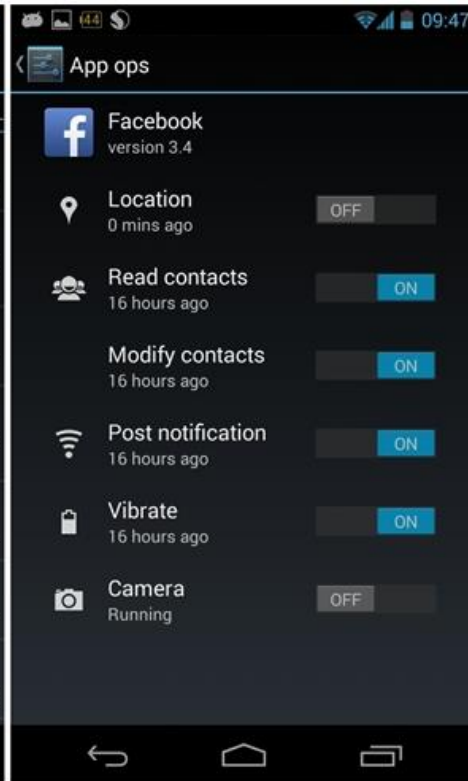
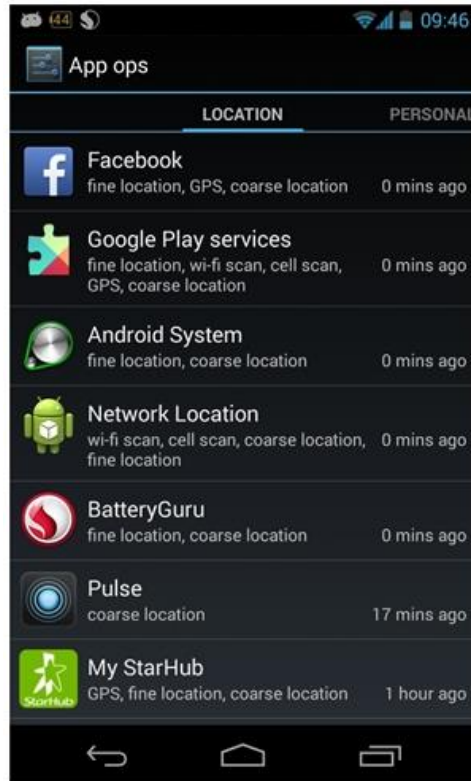
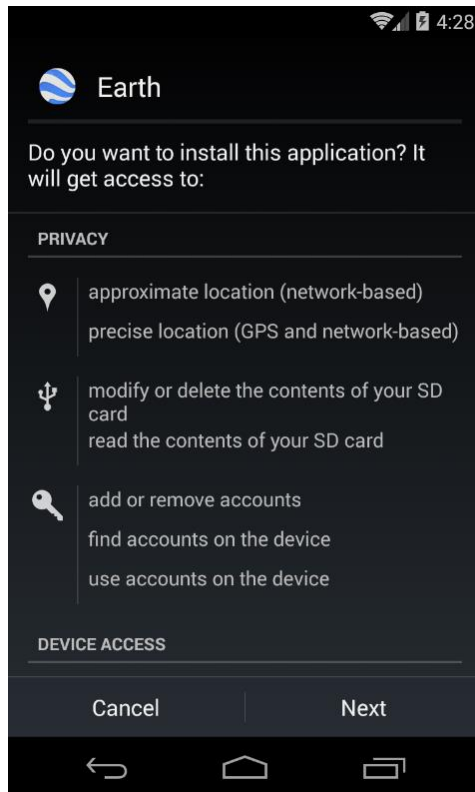
In Collaboration with:

Dr. Haoyu Wang (BUPT), Yuanchun Li, Dr. Xiangqun Chen: Peking University
Dr. Jason Hong, Dr. Yuvraj Agarwal: Carnegie Mellon University

*Helsinki-HKUST-Tsinghua Workshop on Mobile Services and Edge Computing,
Helsinki, Finland, July 27-29, 2016*



Permission-based Access Control



Android 1.6-5.1

All-or-nothing: Permissions requested at installation

Android 4.3-4.4.2

AppOps: Permission management

Android 6.0-

Permission requested at run-time

What are your apps really doing?

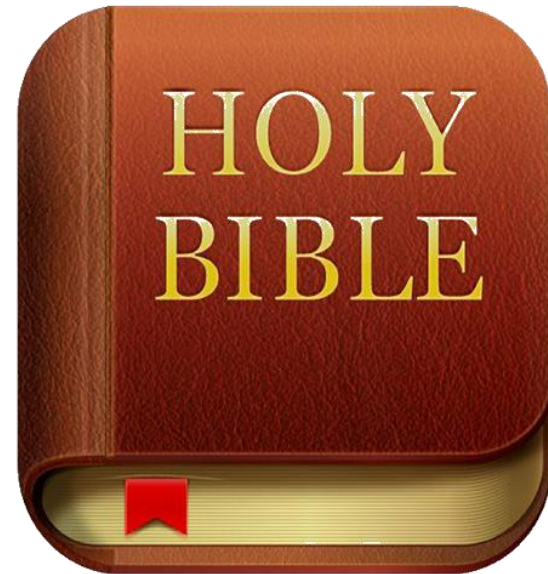
- Many apps have unusual permissions



Brightest Flashlight

Requests:

- Location Data
- Unique Device ID



Holy Bible

Requests:

- Location Data
- Unique Device ID
- Network State

Permission Requests vs. User Knowledge



May I access your location? →

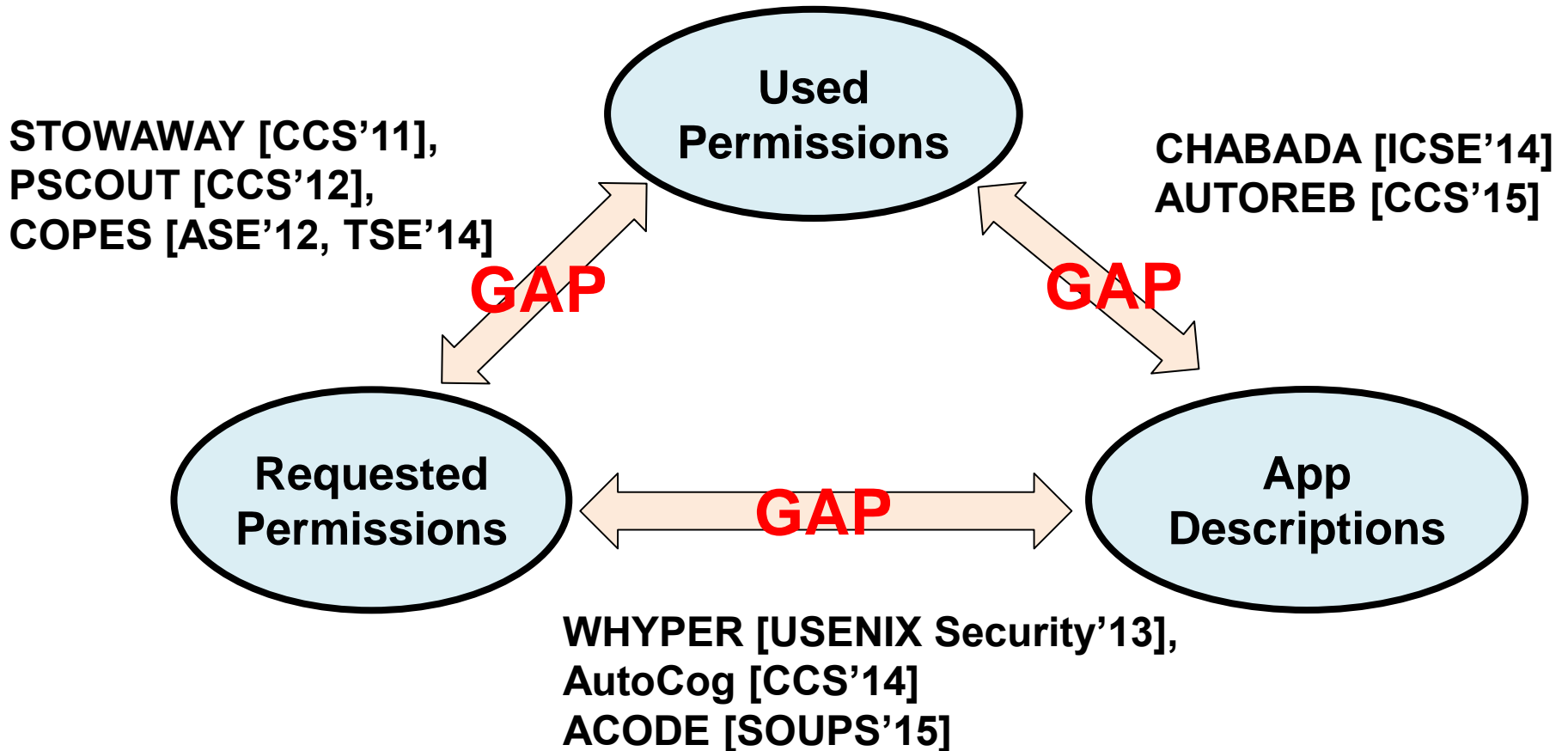
May I read your contact? →

May I ...? →



“Users often feel difficult to understand **why** each permission is required.”

Related Work – Understanding Permissions



Related Work - Improving Permission

- **Improving Android access control**
 - Fine-grained permission: Dr.Android and Mr.Hide [SPSM'12]
 - Context-aware access control: CRePE[ISC'10], [Bai,Securecomm'10]
- **Privilege separation of app and ad libraries**
 - System level: AdDroid [ASIACCS'12], AdSplit [Security'12],
 - App Instrumentation: PEDAL [MobiSys'15]



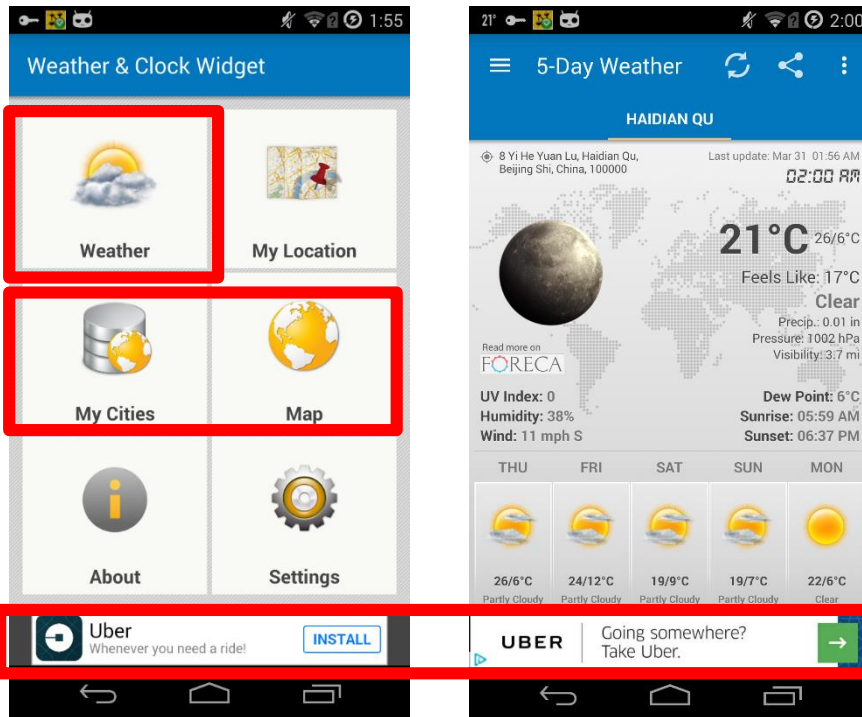
Challenges

- Observation: An app could use a permission in multiple ways

Weather & Clock Widget

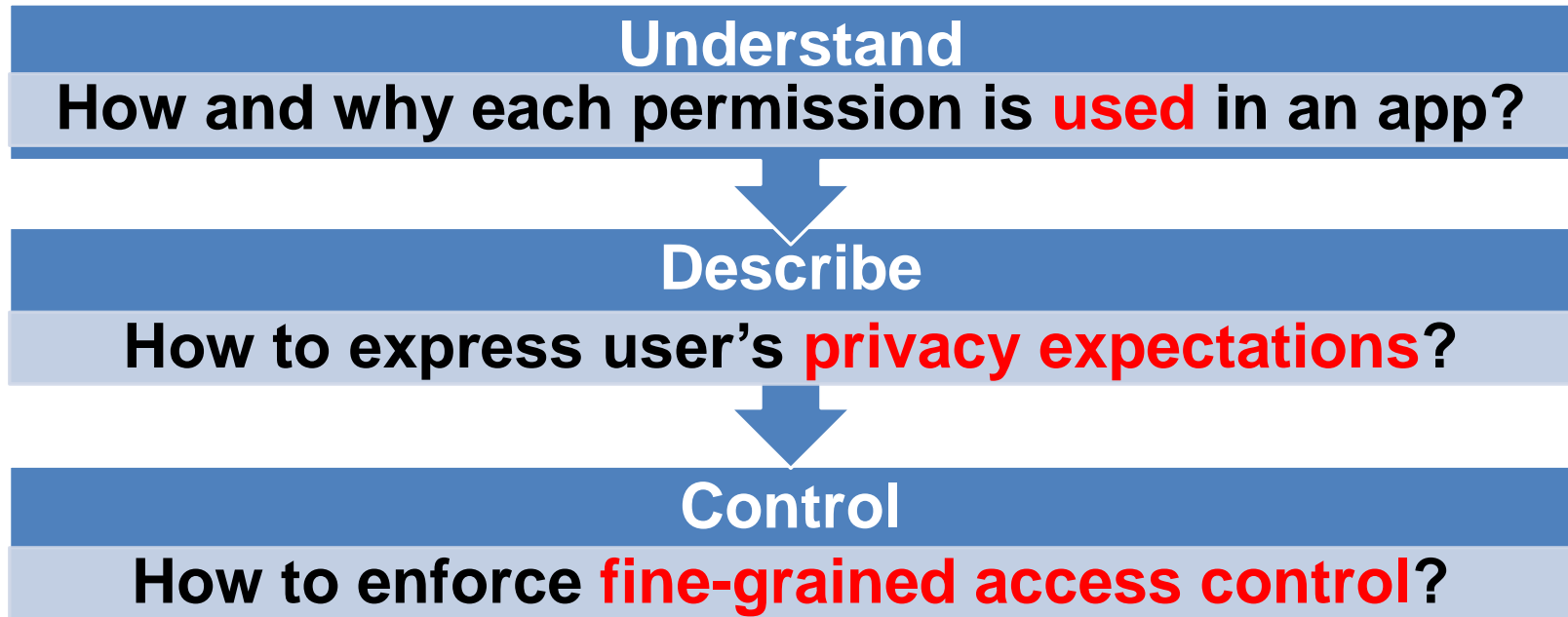
Uses location for:

- Advertising
- Map
- Weather



Our Approach

- Fine-grained access control for Android apps



- Two Examples
 - **Edgar**: Purpose-based access control
 - **PERUIM**: UI-based access control



EDGAR: PURPOSE-BASED ACCESS CONTROL FOR ANDROID



Motivation

- We introduce a new concept *purpose* to indicate the reason for accessing a sensitive data item.
- A permission can be used for *different purposes* within the same app
 - Use location for ads, nearby searching, weather ...
- ➔ **Edgar**: Purpose-based access control



A Taxonomy of Purposes

Location permission

Search nearby places

Location-based
customization

Transportation information

Recording

Map and navigation

Geosocial networking

Geotagging

Location spoofing

Alert and remind

Location-based game

Contacts permission

Backup and Synchronization

Contact management

Blacklist

Call and SMS

Contact-based customization

Email

Find friends

Record

Fake calls and SMS

Remind



Edgar Overview

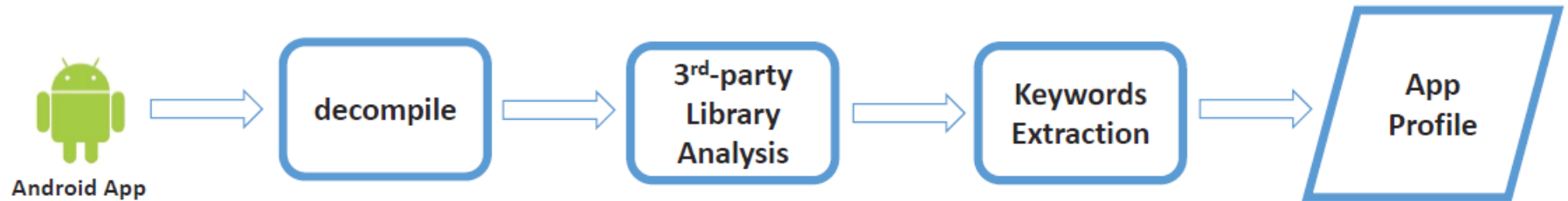
- Decompiled code still retain the text of many identifiers (class names, method names, and field names)
 - These strings offer a hint as to what the code is doing
- Approach:** use *dynamic taint analysis* at runtime to monitor privacy data flow, and determine the purpose of sensitive behavior based on *call stack traces*

```
libcore.os.send(192.44.68.6) received data with tag 0x11 data=[GET/m/ad?v=6&id=a0e6a927970d44468c4c1
897d0e1e0ae&nv=3.2.2&dn=LGE%2CAOSP%20on%20Mako%2Cfull_mako&udi]
java.net.PlainSocketImpl.write(PlainSocketImpl.java:507)
└_.....
└_com.mopub.mobileads.AdFetchTask.fetch(AdFetchTask.java:57)
└_com.mopub.mobileads.AdFetchTask.doInBackground(AdFetchTask.java:42)
└_com.mopub.mobileads.AdFetchTask.doInBackground(AdFetchTask.java:1)
└_android.os.AsyncTask$2.call(AsyncTask.java:287)
└_java.util.concurrent.FutureTask.run(FutureTask.java:234)
└_java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1080)
└_java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:573)
└_java.lang.Thread.run(Thread.java:848)
```

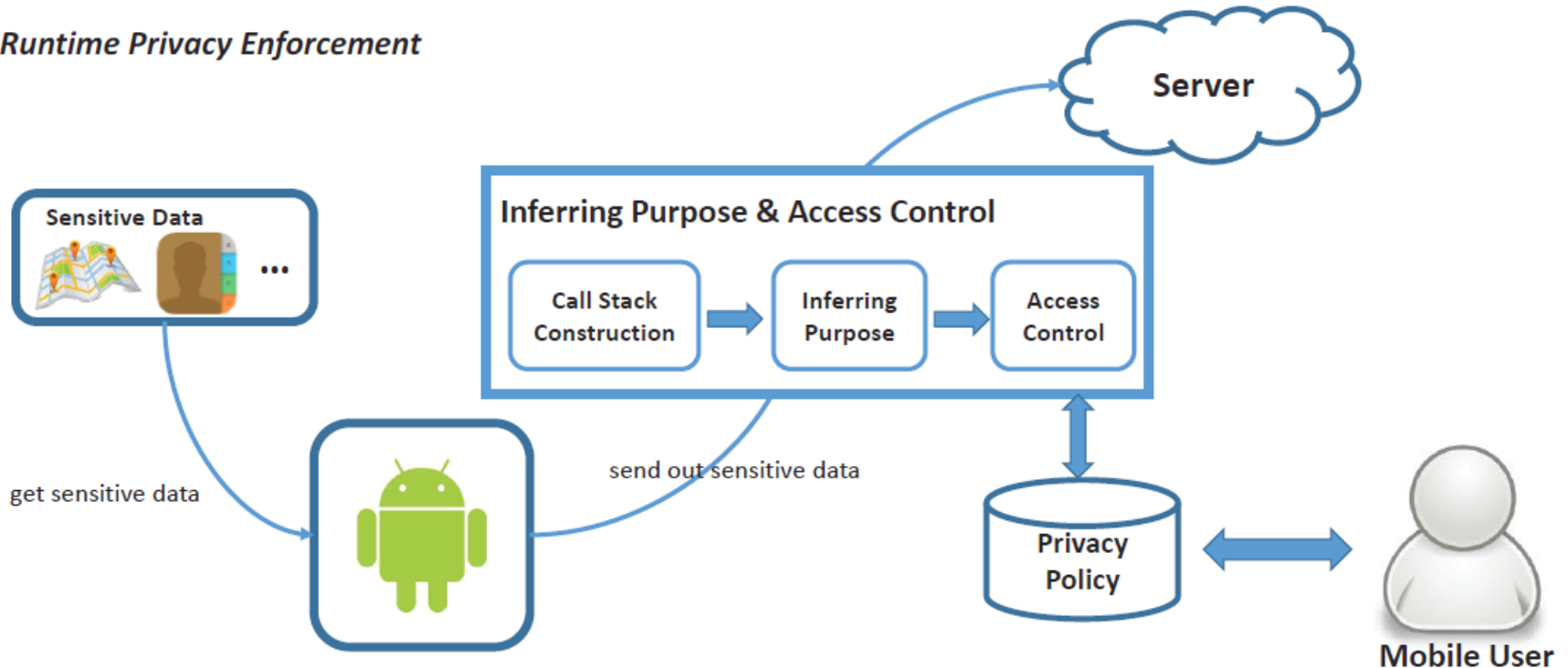


Purpose-based Access Control

Offline/Install Time Learning



Runtime Privacy Enforcement

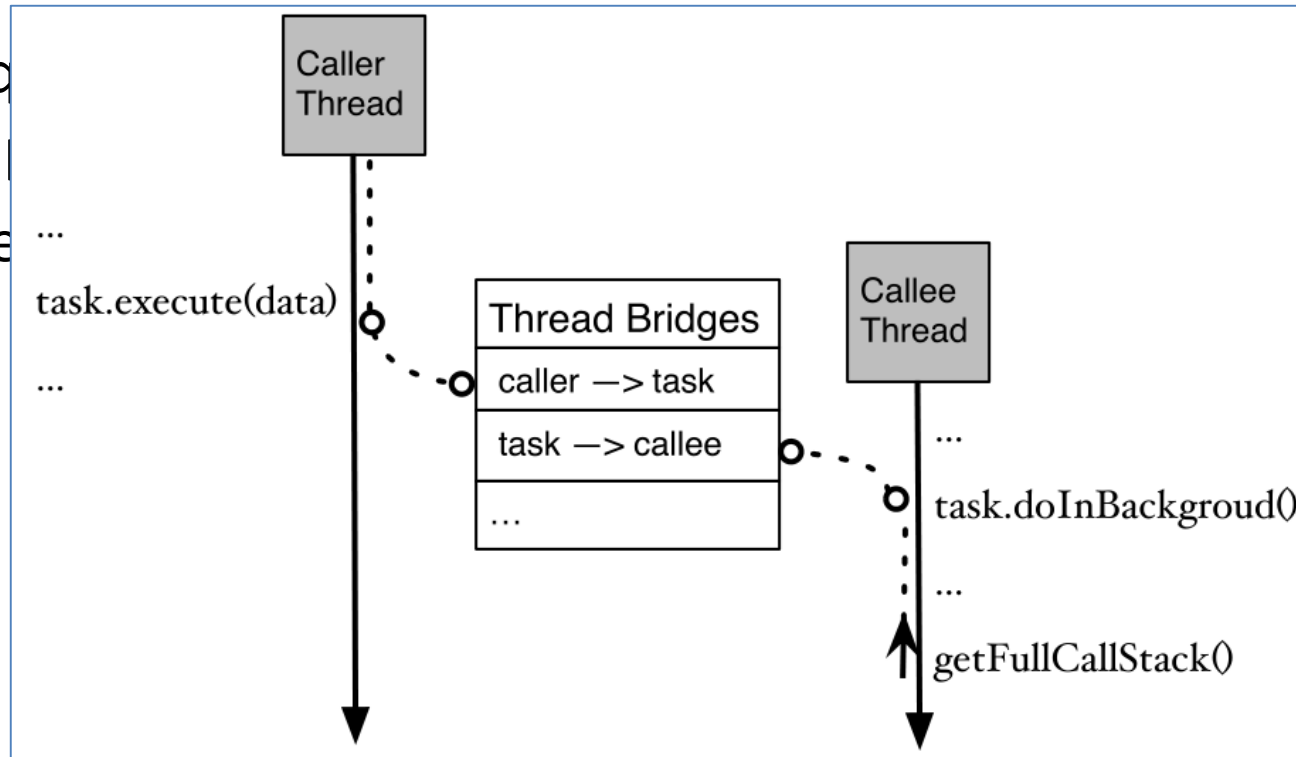


Inferring Purposes at Runtime

- Infer the purposes of sensitive permission uses at runtime based **call stack information**
 - Learn how the sensitive data is accessed and used, which offer a hint as to why sensitive data is being used.

- Call Stack Construction

- Frequent
- Thread

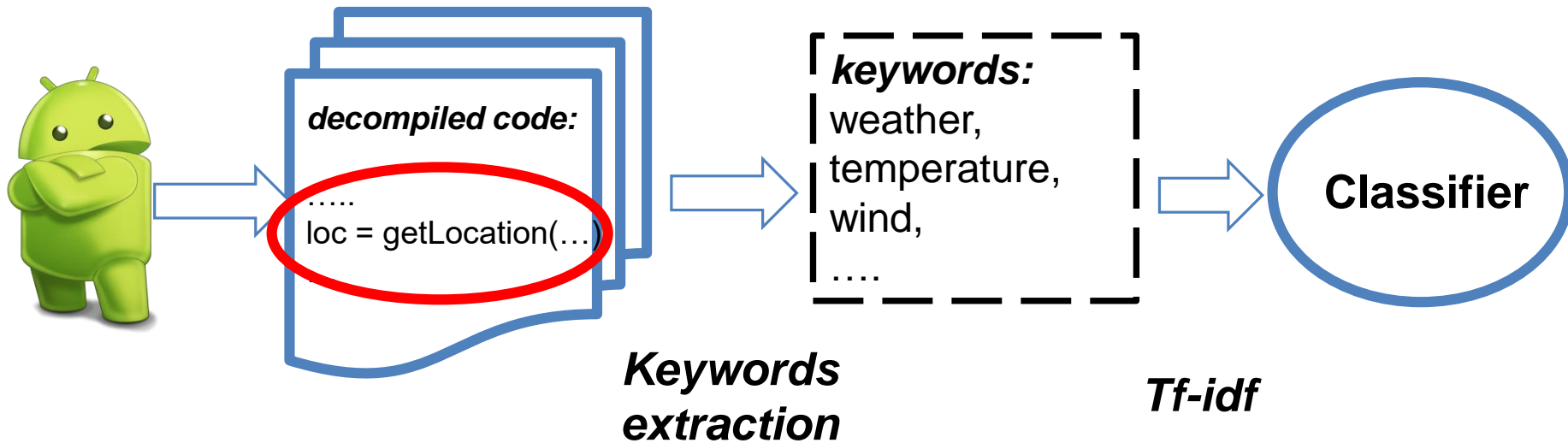


... difficult to infer
... head
... trace



Inferring Purposes at Runtime

- Uses two heuristic methods to infer purposes
 - Analyzing the call stack traces to see whether the sensitive data is used by *well-known third-party libraries*
 - Extracting *meaningful key words* from the methods and classes that related to the call stack



Policy Enforcement

- **Access Control Policies**

- Users can easily define global privacy policies for all the apps using a triple **<permission, purpose, action>**

| Privacy Policy | Description |
|------------------------------------|--|
| <location, ads, block> | disallow accurate location for advertisement |
| <location, nearbysearching, allow> | allow to use location for nearby searching |

- **Policy Enforcement**

- Modified TaintDroid such that at each sink point the app behavior is checked against user-defined policies
- If the sensitive behavior violates the policy, an exception would be thrown to block the data path.



Evaluation

- Dataset: performed experiments on 830 popular apps
 - 81 apps leak GPS location data
 - collect 480 call stack traces that leaks location, of which 171 are unique traces
- Accuracy

| | Accuracy (All Stack Traces) | Accuracy (Unique Stack Traces) |
|-------------------------------------|--------------------------------|-----------------------------------|
| 3 rd -party Libraries | 98.59% | 96% |
| Custom code | 81.19% | 79.25% |
| Total (w/o obfuscation) | 94.73% | 90.20% |
| Total (with obfuscation) | 89.80% | 80.70% |



Performance Evaluation

- Java Micro-benchmark
 - perform similar to TaintDroid, has a 27% overhead with respect to unmodified Android
- Performance overhead breakdown
 - The average performance overhead is about **258 ms** in total

| | Call Stack Construction | Library Comparison | TF-IDF Calculation | SVM Classification |
|--------------------|-------------------------|--------------------|--------------------|--------------------|
| Average time (ms) | 5.81 | 49.03 | 43.38 | 159.95 |
| Standard deviation | 0 | 80.53 | 29.7 | 18.38 |

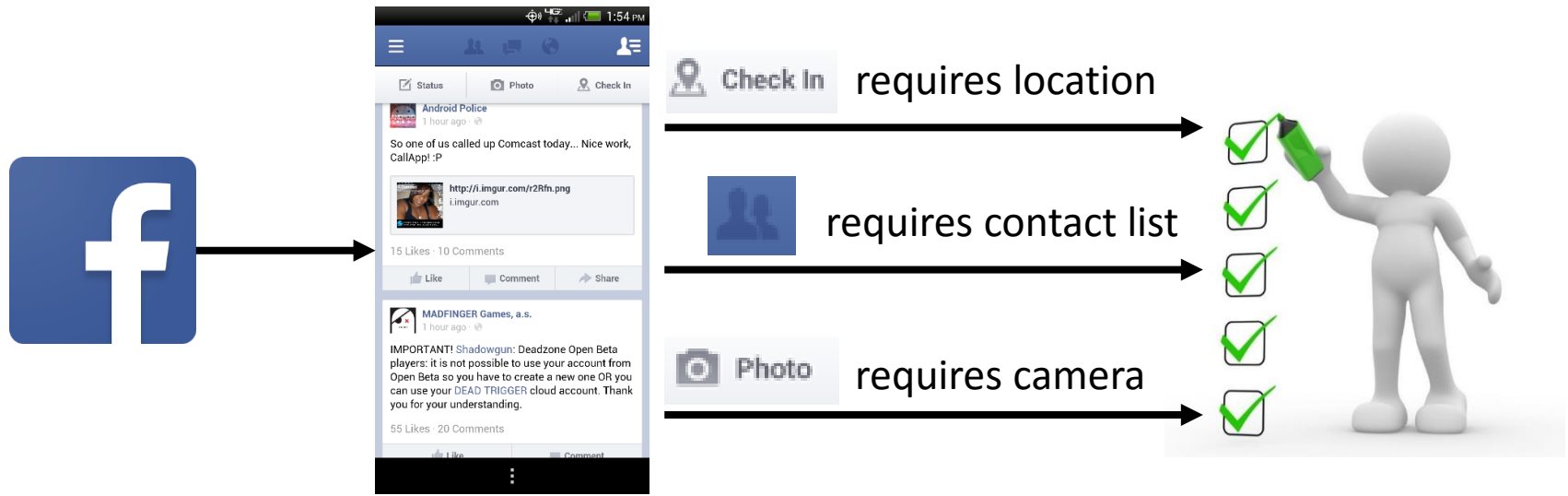


PERUIM: UI-BASED ACCESS CONTROL FOR ANDROID



Understanding Permissions based on UI

- **User interface:** the bridge between an app and users.
- Benefits of using UI to represent permissions
 - UI reflects an app's actual behavior
 - UI is easy for users to understand



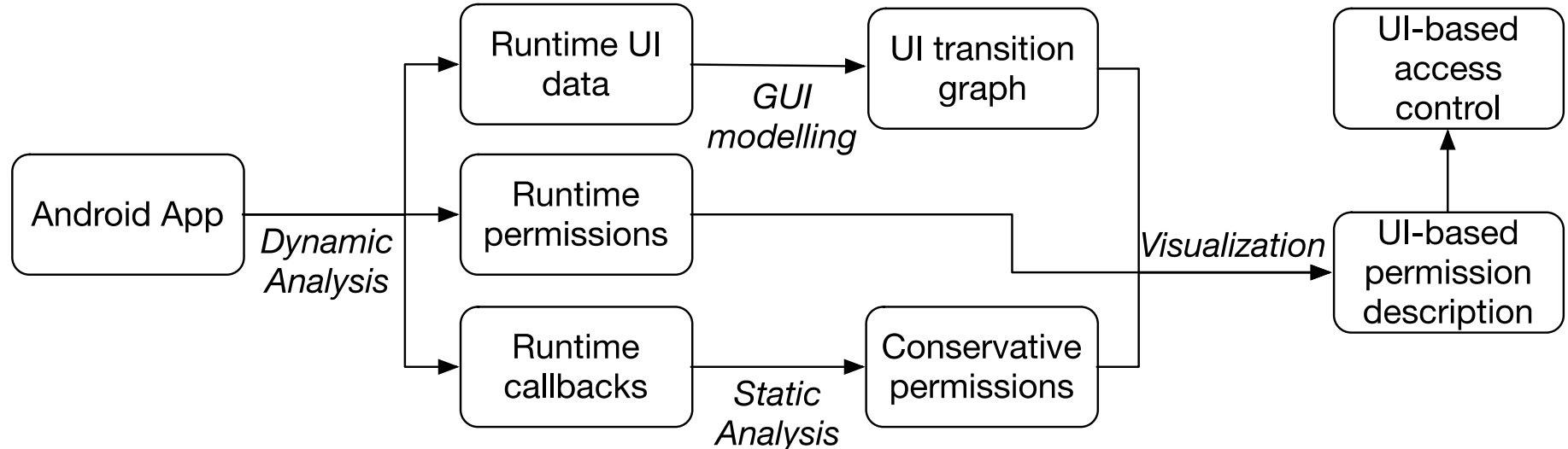
Our Approach: PERUIM

- We introduce a new concept *Permission-UI Mappings*, which includes:
 - {will-access}*: permission requests triggered by UI interaction;
 - {accessed}*: permissions used to render UI content.
- Example:

| UI-based Permission Description | | |
|---------------------------------|-------------------|--------------|
| Permission | Who will access | Who accessed |
| INTERNET | 1, 2, 4, 5, 7 | 7 |
| NETWORK_STATE | 1, 4 | 7 |
| LOCATION | 1, 4 | 7 |

Approach Overview

- UI-based permission visualization and access control based on Permission-UI Mappings (PERUIM)



UI Transition Graph Construction

1. Retrieve the UI hierarchy within an app
2. Construct a UI transition graph (UTG)



UTG = GUI model of Android App

- **Nodes:** UI states
 - Each UI state contains multiple components
- **Edges:** UI events
 - Press keys, click buttons, etc.

Constructed with **dynamic testing**



Extracting UI-related Permissions

Connecting permissions to a UI event:

Dynamic extraction

Logging UI events and permission requests at run-time.

```
...
0ms --- Button.onTouch() called
13ms -- LOCATION permission requested
15ms -- INTERNET permission requested
...
```

Precise

Static extraction

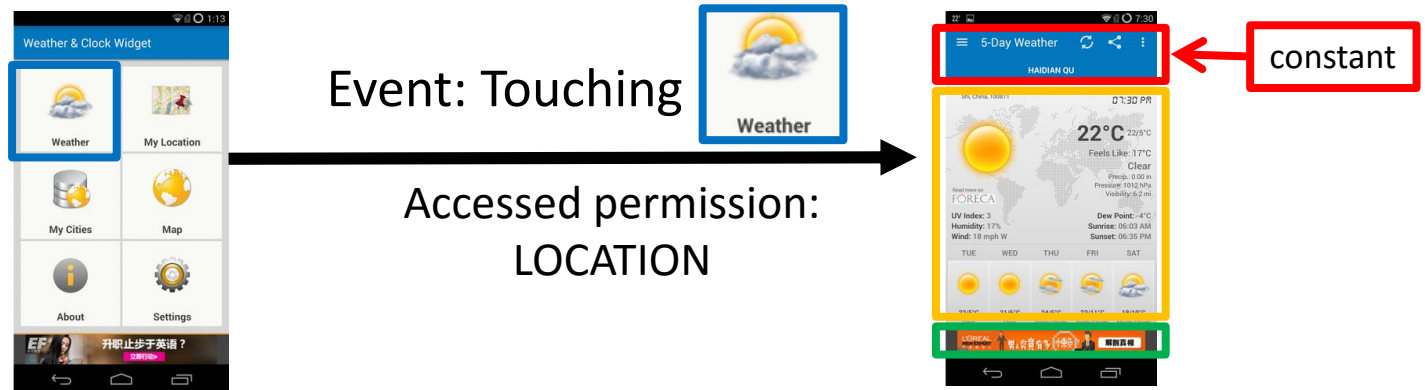
Building call-graph from event handler and mapping the APIs to permissions

```
public void onTouch() {
    l = getLocation(); // uses LOCATION API
    findRestaurants(l); // uses INTERNET API
    ...
}
```

Conservative



Mapping Permissions to UI



- {will-access}

mapping:

will access LOCATION



- {accessed} mappings:



































accessed LOCATION

Permission-UI Mapping Results

- With PERUIM, user can better understand the permission requested in an app.

UI-based Permission Description

| Permission | Who will access | | Who accessed | |
|---------------|---|---|---|---|
| FINE_LOCATION |  |  |  |  |
| |  |  |  |  |
| |  |  |  |  |
| |  |  |  |  |
| INTERNET |  |  |  |  |
| |  |  |  |  |
| |  |  |  |  |
| |  |  |  |  |

Overview of all mappings
(in HTML)



Interactive view of mappings,
grouped by UI components
(in Android)

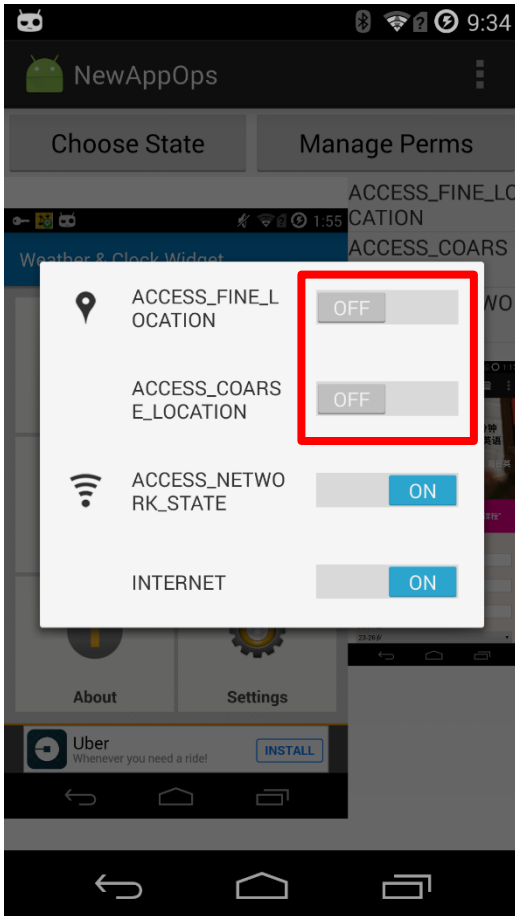
Evaluation – Mapping Accuracy

- Precision of PERUIM, evaluated with 10 popular apps from Google Play.
- The correct mappings are manually labelled.

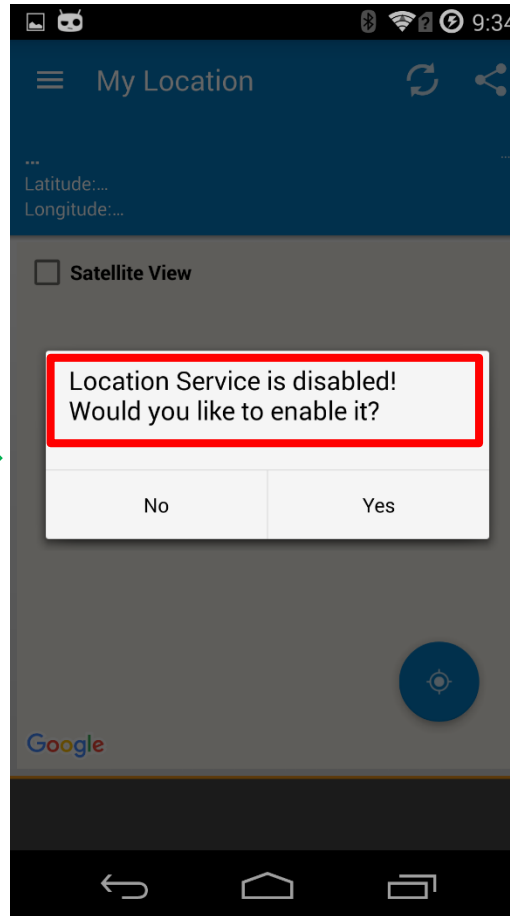
| | {will-access} set | {accessed} set | overall |
|-----------|-------------------|----------------|---------|
| # mapping | 141 | 74 | 215 |
| # correct | 123 | 42 | 165 |
| precision | 87.23% | 56.77% | 76.75% |

UI-based Access Control

- Access control interface



1. Policy configuration in PERUIM Android app.



2-a. App under control. (Permission denied.)



2-b. App under control. (Permission granted.)

Challenges and Future Work

- Understanding why and how each permission is exactly used is difficult
 - Many apps use permissions in unexpected/malicious ways
- Users' attitude is the key: make sure users understand!
 - User understanding is the first step in access control
 - Extensive user studies are needed
- It is a challenging problem: many issues remain unsolved
 - Edgar and PERUIM have only touched a tip of the iceberg
 - Any suggestion/collaborations are welcome!



Other Related Work

- Mobile app analysis
 - Analyzed 1 million Android apps (All free apps from Google Play at March 2015)
 - **LibRadar**: a method to **detect third-party libraries** in a given app
 - Detected over 16,000 third-party libraries from 1 million apps
 - Instant library detection by **LibRadar**: <http://radar.pkuos.org>
 - **WuKong**: a fast and accurate method to **detect app clones**
- Android energy modeling and optimization
 - Power modeling based on coarse-grained battery data
 - Analysis of background energy consumption
 - Automated fixing of sensor energy bugs based on app instrumentation



Related Publications

- **Fine-grained Access Control**

- Haoyu Wang, Jason I. Hong, Yao Guo, “*Using Text Mining to Infer the Purpose of Permission Use in Mobile Apps*”, **UbiComp 2015**, Osaka, Japan, Sep 2015.
- Yuanchun Li, Yao Guo, Xiangqun Chen, “*PERUIM: Understanding Mobile Application Privacy With Permission-UI Mapping*”, **UbiComp 2016**, Heidelberg, Germany, September 2016. (to appear)

- **Android App Analysis**

- Ziang Ma, Haoyu Wang, Yao Guo and Xiangqun Chen, “*LibRadar: Detecting Third-party Libraries in Android Apps*”, **ICSE 2016** (Demo Track), Austin, TX. May 2016.
- Haoyu Wang, Yao Guo, Ziang Ma, Xiangqun Chen, “*WuKong: A Scalable and Accurate Two-Phase Approach to Android App Clone Detection*”, **ACM ISSTA 2015**, Baltimore, MD, pp. 71-82.

- **Energy Optimization**

- Yuanchun Li, Yao Guo, Junjun Kong, Xiangqun Chen, “*Fixing Sensor-Related Energy Bugs through Automated Sensing Policy Instrumentation*”, **ACM/IEEE ISLPED 2015**, Rome, Italy.



Thank you!

Contact: yaoguo@pku.edu.cn

