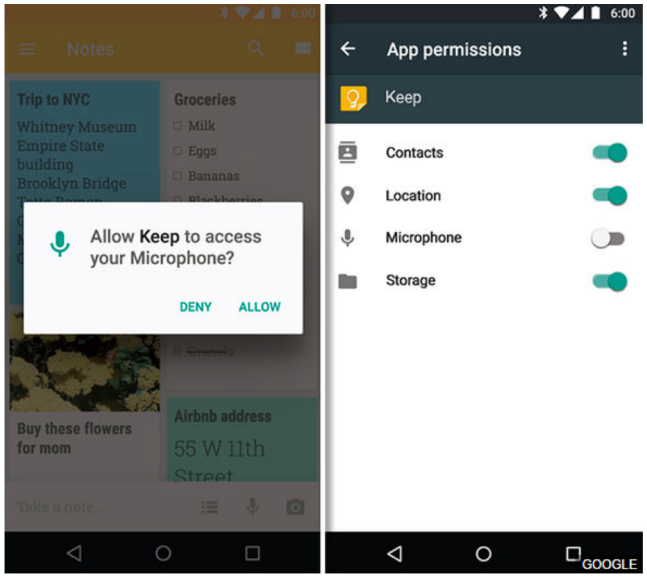
Automatic Identification of Sensitive UI Widgets based on Icon Classification for Android Apps

PI: Xusheng Xiao
Case Western Reserve University

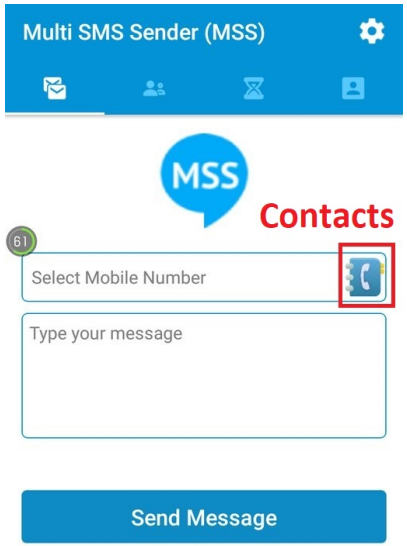
NSF Award #1755772
CRII: SaTC: Enhancing Mobile App Security by Detecting Icon-Behavior Contradiction

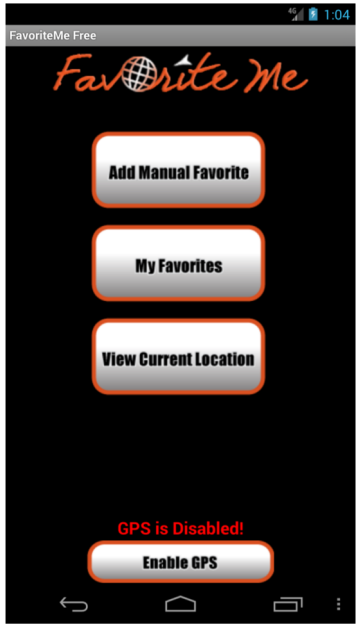
Privacy Issues of Mobile App

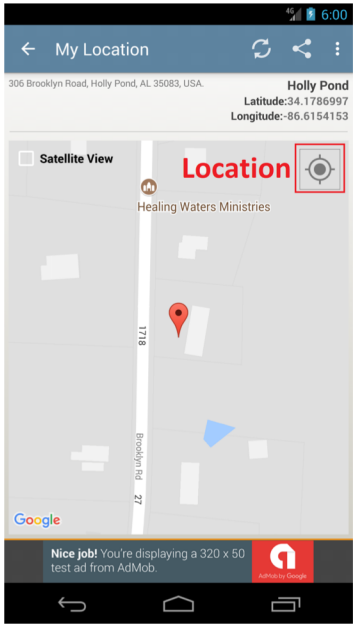


- Mobile apps have become an integral part of our life
  - E.g., business, transportation, education
- Many apps access sensitive data, raising privacy concerns
  - E.g., location, contacts, microphone

Sensitive UI Widgets



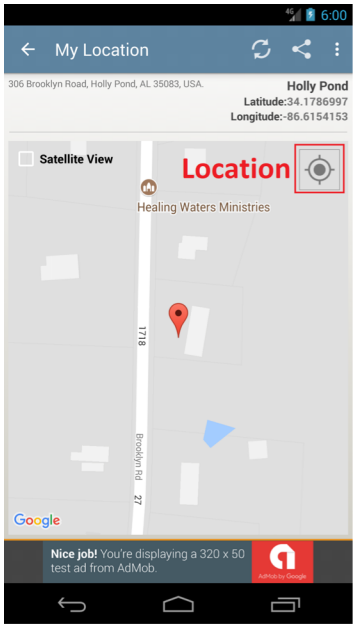




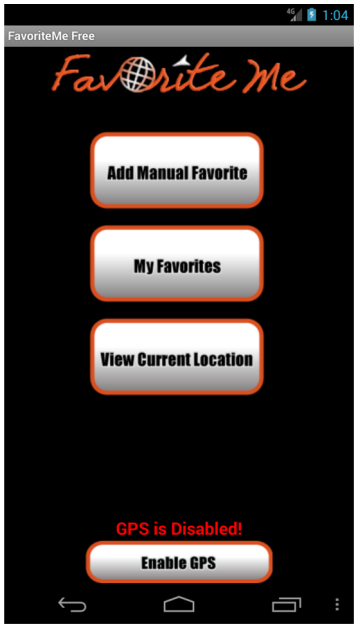
- It is crucial to understand the apps' intentions in using the sensitive information
  - E.g., Inspecting permissions and informing users about sensitive data
- Apps express their intentions to use or collect users' sensitive data via sensitive UI widgets, i.e., justifying the uses of the data
- App market needs an automatic approach to understand these intentions

Challenges: Understanding Intentions of UI Widgets

- UI widgets' intentions are expressed via ***texts*** and ***images***
  - Prior works focus on analyzing framework APIs (e.g., device identifiers, and contacts) or descriptive texts (e.g., text labels), not images



- Object icons: icons with specific shapes, no co-located texts
  - Different styles, scales, angles
- Text icons: icons embedded with texts
  - Diversified colors and opacities



Automatic Identification of Sensitive UI Widgets with Icons

Sensitive UI Widget Identification

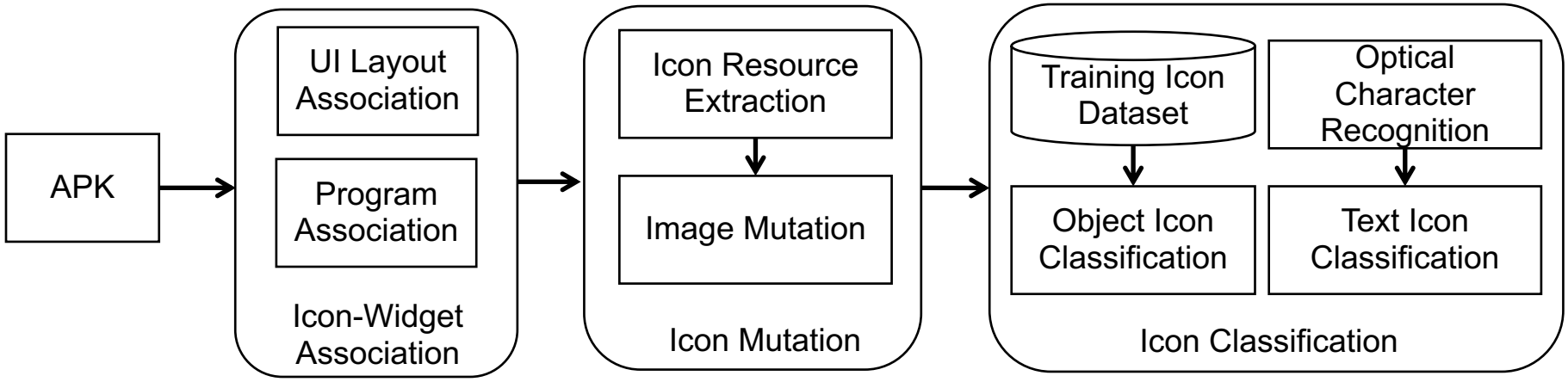
- Given an app, which UI widgets are associated with icons?

- Based on the icons, which sensitive data the UI widgets will use?

IconIntent

- Synergistically combine ***program analysis*** and ***icon classification***
- Associate icons with UI widgets via static analysis
- Classify the intentions of icons (both object icons and text icons) into eight pre-defined sensitive user input categories
  - Including Camera, Contacts, Email, Location, Phone, Photo, SMS, and Microphone

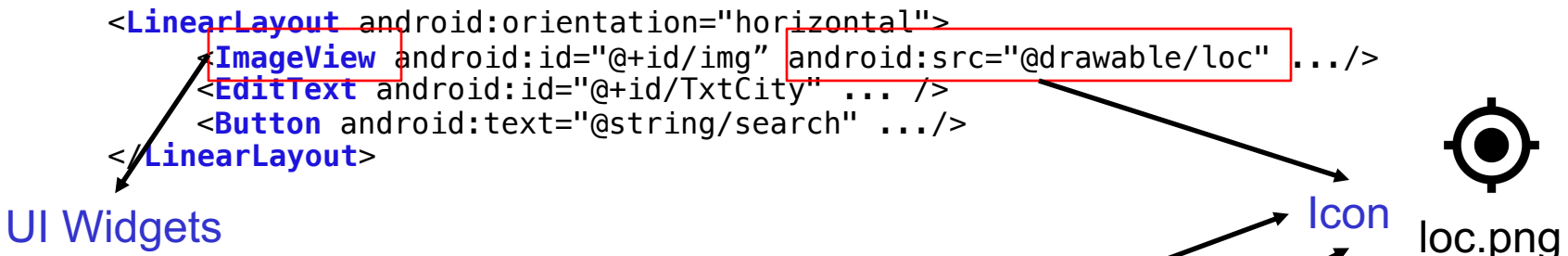
Overview of IconIntent



- Icon-Widget Association: static analysis on UI layout files and code
- Icon Mutation: image mutations on extracted icons
- Icon Classification: classification of icons into sensitive categories

Icon-Widget Association: UI Layout

- Static Analysis: XML parsing and resource resolution
- UI layouts: widgets and icons




```
<LinearLayout android:orientation="horizontal">
    <ImageView android:id="@+id/img" android:src="@drawable/loc" .../>
    <EditText android:id="@+id/TxtCity" ... />
    <Button android:text="@string/search" .../>
</LinearLayout>
```

- Drawable objects

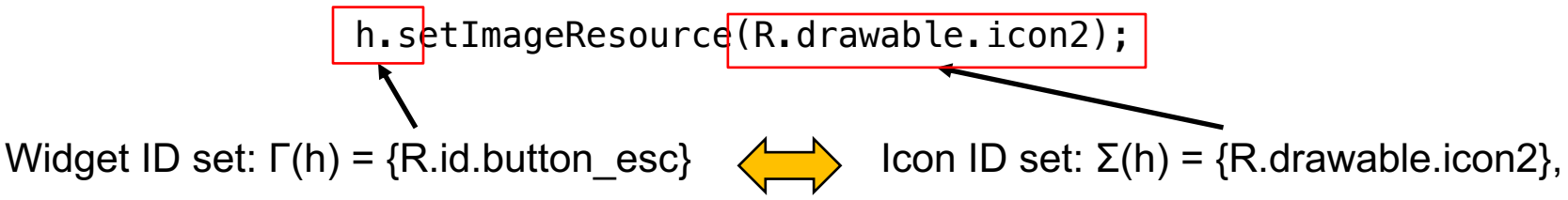
```
<selector>
    <item android:state_checked="true"
android:drawable="@drawable/btn_radio_to_on_mtrl_015" />
    <item android:drawable="@drawable/btn_radio_to_on_mtrl_000" />
</selector>
```

Icon-Widget Association: API Calls

- Life cycle methods: load layout files, bind variables to UI widgets, and associate icons to UI widgets

```
void onCreate(Bundle savedInstance) {
1    View g = this.findViewById(R.id.button_esc); // FindView ImageView
2    h = (ImageView) g; // cast to ImageView
3    h.setImageResource(R.drawable.icon2); // associate icon
     ...
}
```

- Static analysis: dataflow analysis with over-approximations to associate UI widgets and icons

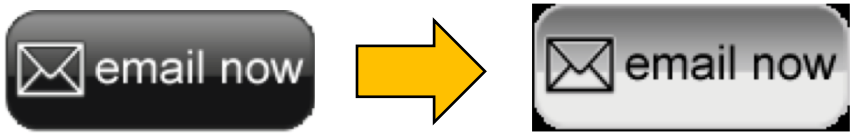


App Icon Varieties

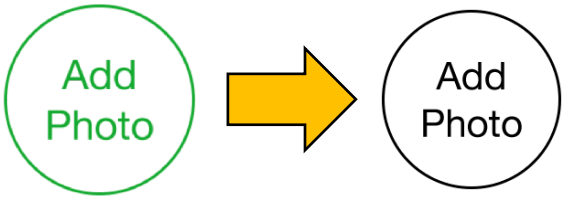
- Icons have different combinations of colors and transparencies in texts, backgrounds, and object shapes
- Challenges for computer vision techniques
  - Small: (a) and (b)
  - Low contrast: (c) and (d)
  - Bright color text and dark background: (e)
  - Opacity: (f) and (g)

Icon Mutation

- RGBA model <R,G,B,A> to represent an icon
  - R,G,B for red, green, blue, A for opacity
- Image mutations:
  - Image Scaling: enlarge pixel values using nearby pixels
  - Grayscale Conversion: convert an image to represent only the amount of light
  - Color Inversion: invert the colors of each pixel

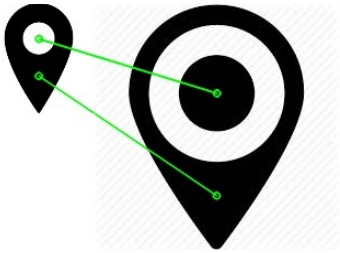


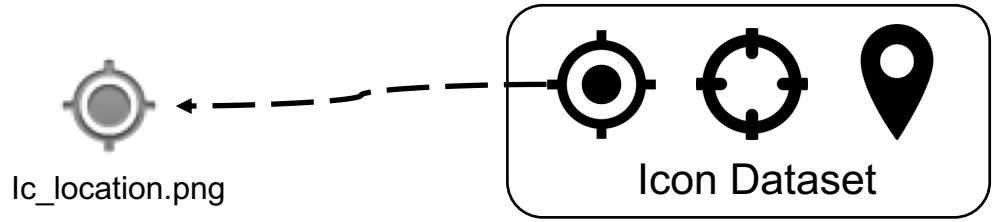
  - Contrast Adjustment: adjust the contrast of colors in the image
  - Opacity Conversion: convert the opacity differences to the color differences



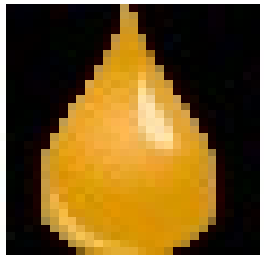
Icon Classification – Object Icon

- Object recognition to classify object icons based on a training icon set labeled with sensitive user-input categories





- Scale-Invariant-Feature-Transform (SIFT)
  - Identifying key locations that are invariant with respect to image translation, scaling, and rotation and matching key locations
  - Challenges:
    - Too few key locations
      - Enlarging icons and FAST
    - Lower tolerance for changes
      - Relative One-to-One Mapping



Icon Classification – Text Icon

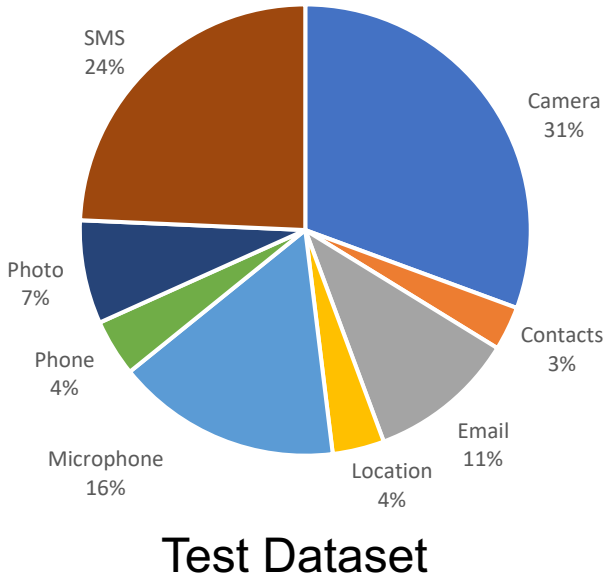
- Optical Character Recognition (OCR)
  - Working well for dark text and bright background

  - Still not perfect even with image mutations
    - E.g., location -> lcation or llocation, email -> emai
- Classification based on Keyword Similarity
  - 95+% of 300 text icons extracted from top Google Play apps containing 1 to 3 words
  - Edit distance-based similarity (considering keyword length)
    - $Sim_{w,k} = 1 - \frac{Edit\ Distance}{length(k)}$

Evaluation Setup

- Implementation:
  - Static analysis: Gator and Soot
  - Icon classification: OpenCV and Asprise OCR

- Subject:
  - Training dataset: 1,576 icons
    - Google image search: 800
    - Top 10,000 apps: 776
  - Test dataset: 150 apps with 5,791 icons
    - 539 sensitive object icons
    - 49 sensitive text icons
    - Total: 588 sensitive icons

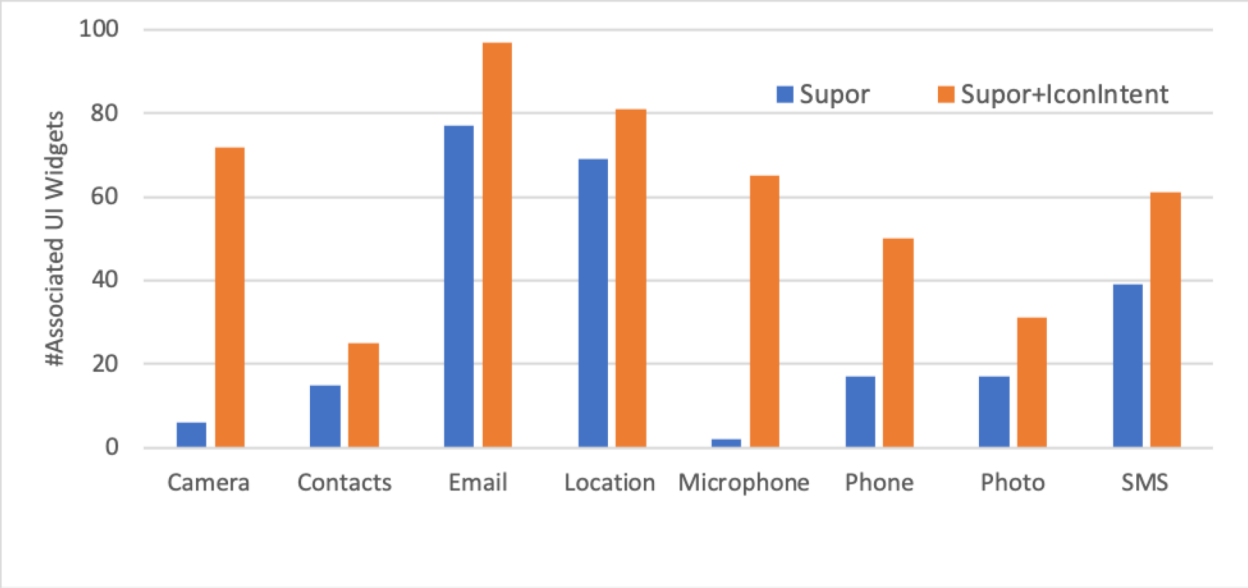


Identifying Sensitive UI Widgets

Category	#Detected SIs			#Apps with SIs	#Detected SWs	#Apps with SWs
	Object	Text	All			
Camera	148	1	149	47	65	35
Contacts	14	1	15	6	10	6
Email	44	5	49	16	25	12
Location	19	11	30	9	12	9
Microphone	75	3	78	26	65	19
Phone	20	1	21	6	38	4
Photo	41	12	53	13	19	13
SMS	125	11	136	23	24	10
All	486	44	530	135	248	97

- Detecting most sensitive icons (90.1%, 530 / 588) from most apps (135 out of 138 apps that contain sensitive icons)
- Prevalent sensitive UI widgets: 248 UI widgets from 97 apps (prec: 82.4%)
- Sensitive icons not always used in UI widgets
  - 125 SMS icons -> 24 UI widgets, 20 Phone icons -> 38 UI widgets

Combining with SUPOR



- SUPOR: text-based sensitive UI widget identification
  - Expand to include buttons, radio buttons, check boxes,
  - Leverage dex2jar to support custom widgets
- SUPOR: 242 SUI <-> SUPOR+IconIntent: 487 SUI
- Only 3 UI widgets are identified by both SUPOR and IconIntent

Icon Classification

Setting	P (%)	R(%)	F (%)
SIFT	43.0	54.5	48.1
Without Mutation	91.2	64.9	75.8
IconIntent	88.2	87.3	87.7

Object-Icon Classification

Setting	P (%)	R(%)	F (%)
Without Mutation	91.7	22.9	36.6
IconIntent	89.8	89.8	89.8

Text-Icon Classification

- IconIntent achieves an average F-score of 87.7% (with distance threshold as 0.3)
- IconIntent greatly improves F-score with image mutation (from 36.6% to 89.8%)

Conclusion

- IconIntent
  - Program analysis techniques to associate icons and UI widgets
  - Computer vision techniques to classify the associated icons into eight sensitive categories
- Evaluation on 150 apps from Google Play
  - Detect 248 sensitive UI widgets in 97 apps, achieving a precision of 82.4%
  - SUPOR +IconIntent can detect 487 sensitive UI widgets (101.2% improvement over SUPOR only)
  - Image mutations improves icon classification

Thank You !

Questions ?