AppContext: Differentiating Malicious and Benign Mobile App Behavior Under Contexts

Tao Xie

Joint Work w/ **David Yang,** Sihan Li (Illinois)

Xusheng Xiao, Benjamin Andow, Rahul Pandita, William Enck (NCSU)

UNIVERSITY OF ILLINOIS

AT URBANA-CHAMPAIGN

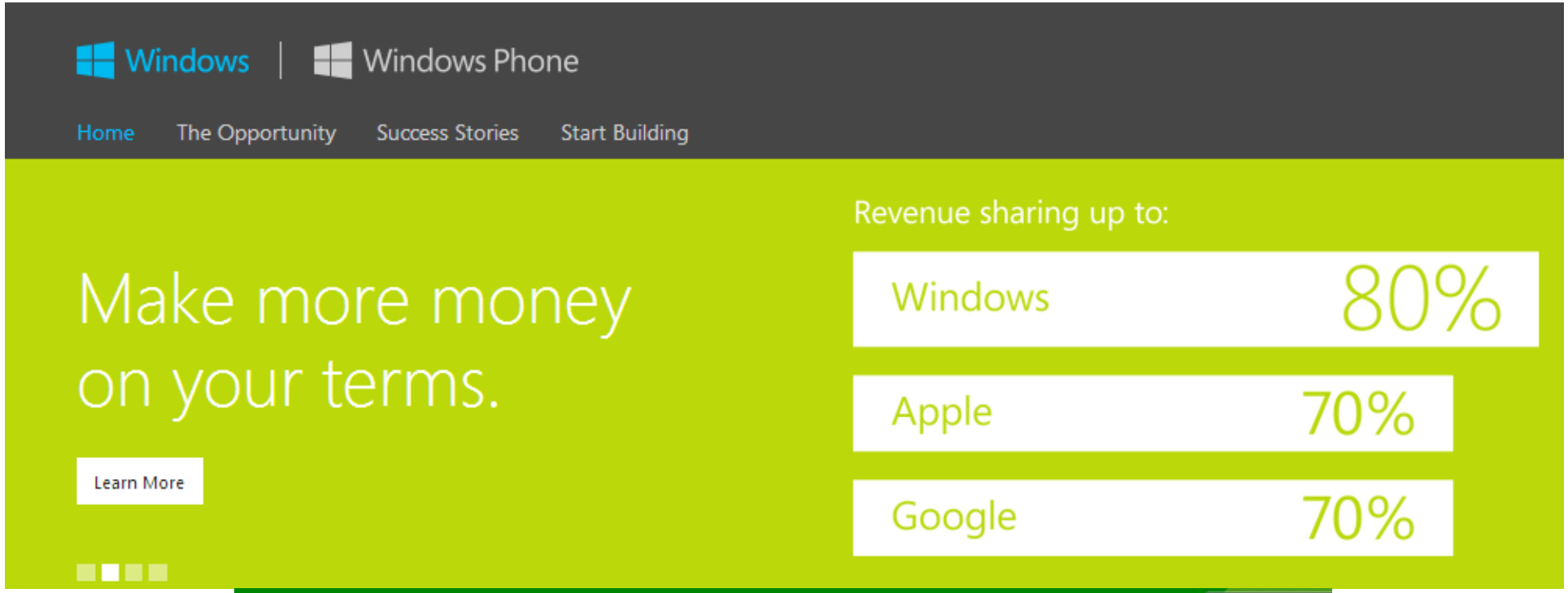
Mobile App Markets

Apple App Store

Google Play

Microsoft Windows Phone

App Store beyond Mobile Apps!



App Store within Mobile App!

招行信用卡

服务号 -- 如果你是持卡人，可快捷查询信用卡账单、额度及积分；快速还款、申请账单分期；微信转接人工服务；信用卡消费，微信免费笔笔提醒。如果不是持卡人，可以微信办卡！

What If Formal Specs Are Written?!

informal: **app description**, etc.

permission list, etc.

APP DEVELOPERS

App Functional Requirements

App **Security** Requirements

APP USERS

User Functional Requirements

User **Security** Requirements

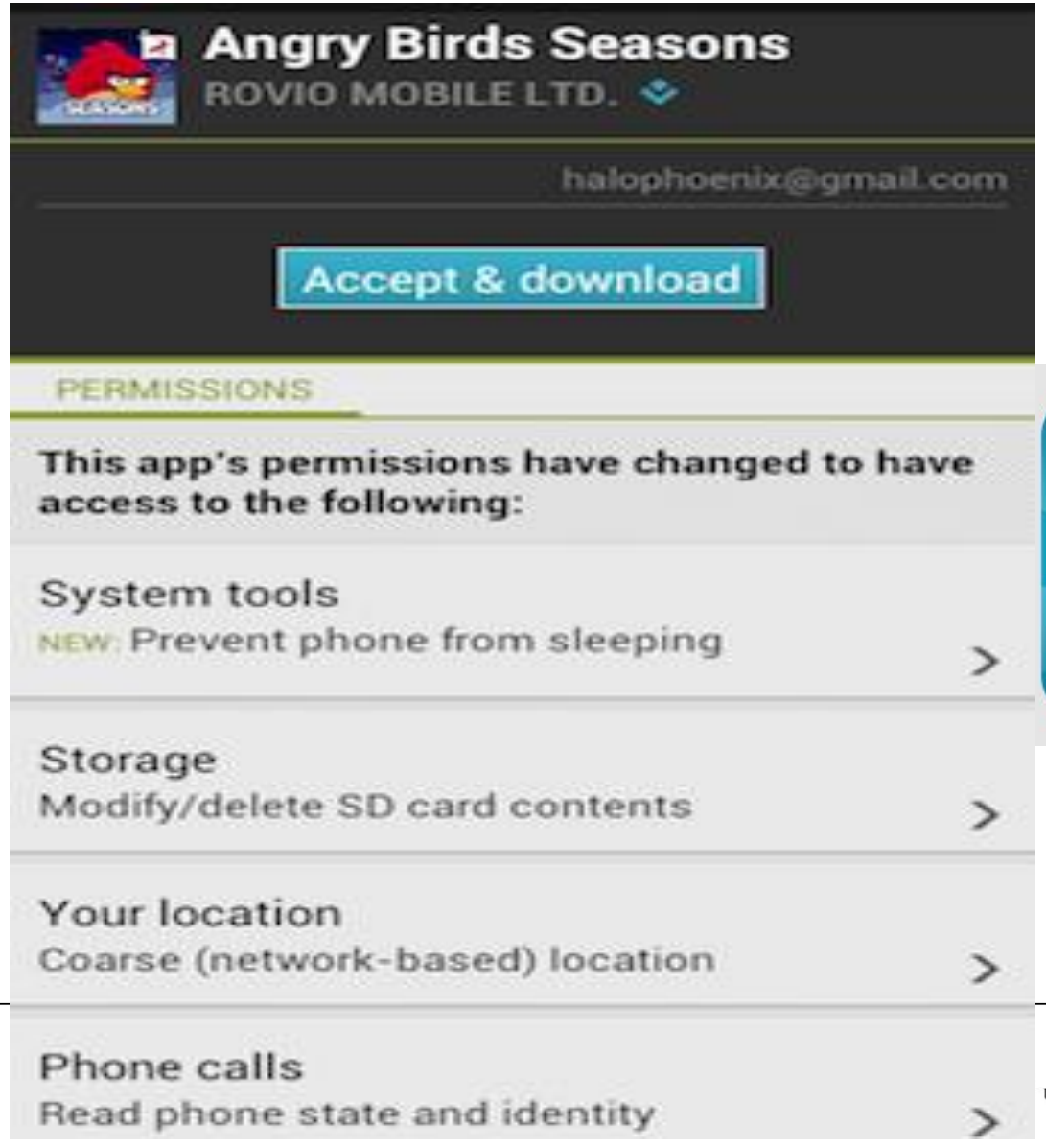
Informal App Functional Requirements: App Description

Description

The survival of the Angry Birds is at stake. Dish out revenge on the greedy pigs who stole their eggs. Use the unique powers of each bird to destroy the pigs' defenses. Angry Birds features challenging physics-based gameplay and hours of replay value. Each level requires logic, skill, and force to solve.
If you get stuck in the game, you can purchase the Mighty Eagle! Mighty Eagle is a one-time in-app purchase Angry Birds that gives unlimited use. This phenomenal creature will soar from the skies to wreak havoc and smash the pesky pigs into oblivion. There's just one catch: you can only use the aid of Mighty Eagle to pass a once per hour. Mighty Eagle also includes all new gameplay goals and achievements!

In addition to the Mighty Eagle, Angry Birds now has power-ups! Boost your birds' abilities and three-star leve

App Security Requirements: Permission List



u

What If Formal Specs Are Written?!

informal: **app description**, etc.

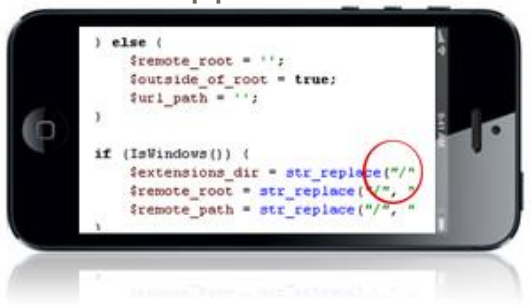
permission list, etc.

APP DEVELOPERS

App Functional Requirements

App **Security** Requirements

App Code



APP USERS

User Functional Requirements

User **Security** Requirements

Example Andriod App: Angry Birds

Strassia
Guest

Why does Angry Birds need to know who I call?

11-19-2012, 06:15 PM

DrDeth
Charter Member

Marketing.

Rovio explains the new permission that Angry Birds Seasons requires

Tweet ShareThis StumbleUpon Pinterest Reddit

Published on Friday, 02 December 2011 10:42

11-19-2012, 05:46 PM

Baron Greenback
Guest

Join

Angry Birds just needs to know the phone state so that it can handle an incoming call when you are playing.

What If Formal Specs Are Written?!

informal: **app description**, etc.

permission list, etc.

APP DEVELOPERS

App Functional Requirements

App **Security** Requirements

App Code

APP USERS

User Functional Requirements

User **Security** Requirements

In reality, few of these requirements are (formally) specified!!

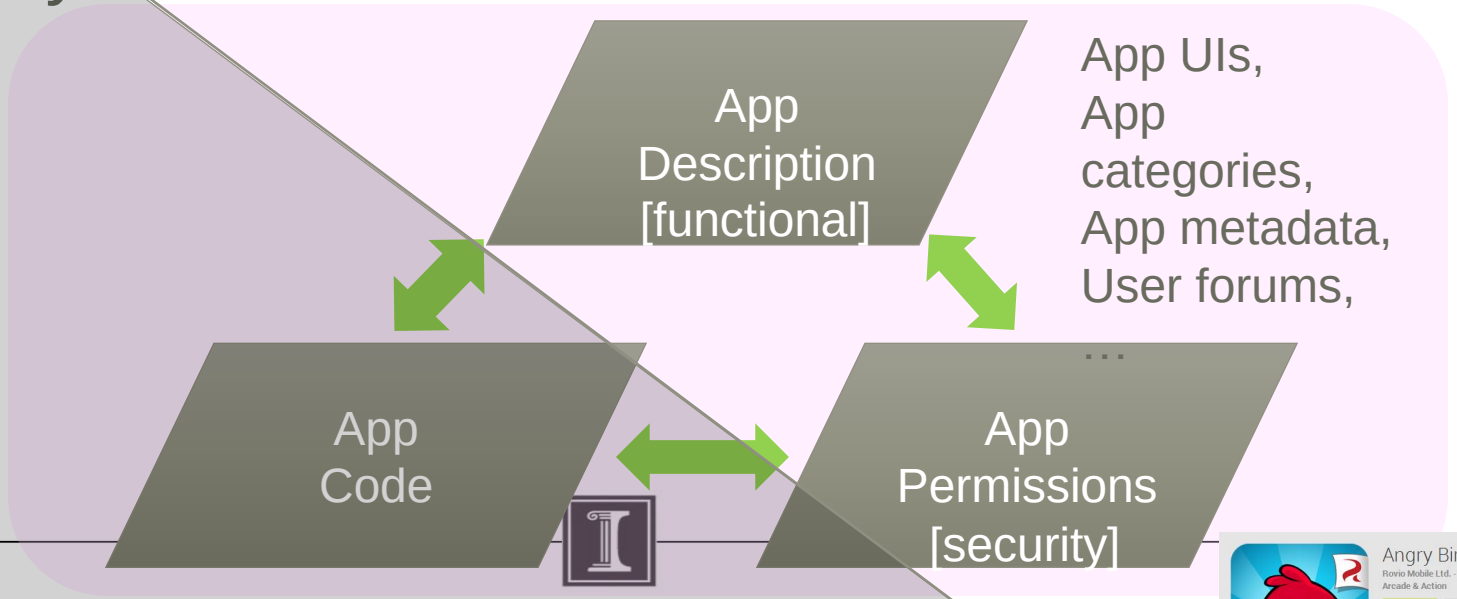
➔ Hope?!: Bring human into the loop: user perception + judgment

Our Yin-Yang View on Mobile App Security

User-Perceived Information

App Security Behavior

- Reason about user-perceived info, e.g., WHYPER (⬆)
- Push app security behavior across the boundary, e.g., AppContext (➔)
- Check consistency across the boundary (🡐➔)
- Reduce user judgment effort (⬇)

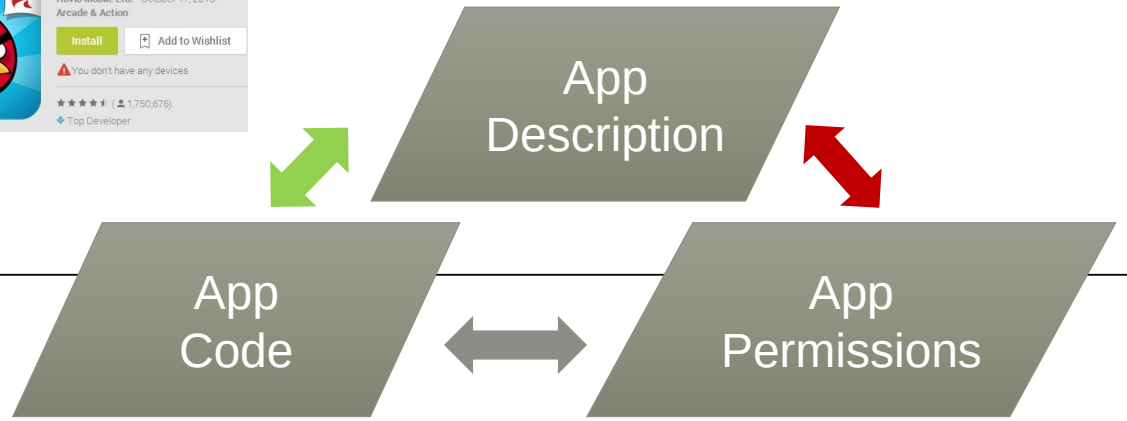


Assuring Market Security/Privacy

- **Apple** (*Market's* Responsibility)
  - Apple performs **manual inspection**
- **Google** (*User's* Responsibility)
  - **Users approve** permissions for security/privacy
  - Bouncer (static/dynamic malware analysis)
- **Windows Phone** (Hybrid)
  - Permissions / **manual inspection**

Program Analysis + Natural Language Processing

- Previous approaches look at
  permissions ←→ code (runtime behaviors)
- ***What does the users expect?***
  - **GPS Tracker:** record and send location
  - **Phone-Call Recorder:** record audio during phone call

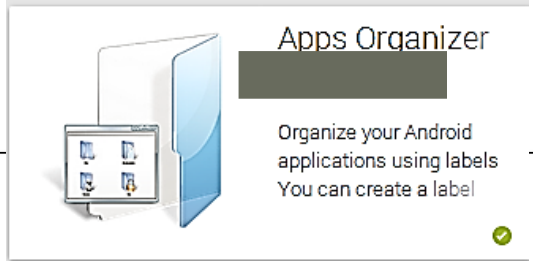


Vision

"Bridging the gap between user expectation ←→ app behaviors"

- User expectations
  - user perception + user judgment
- Focus on permission ←→ app descriptions
  - permissions (protecting user understandable resources) should be discussed

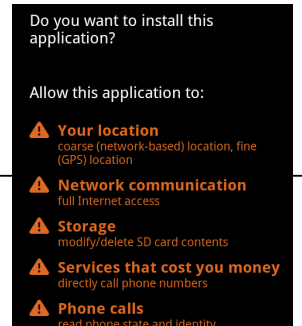
App Description Sentence



Linkage

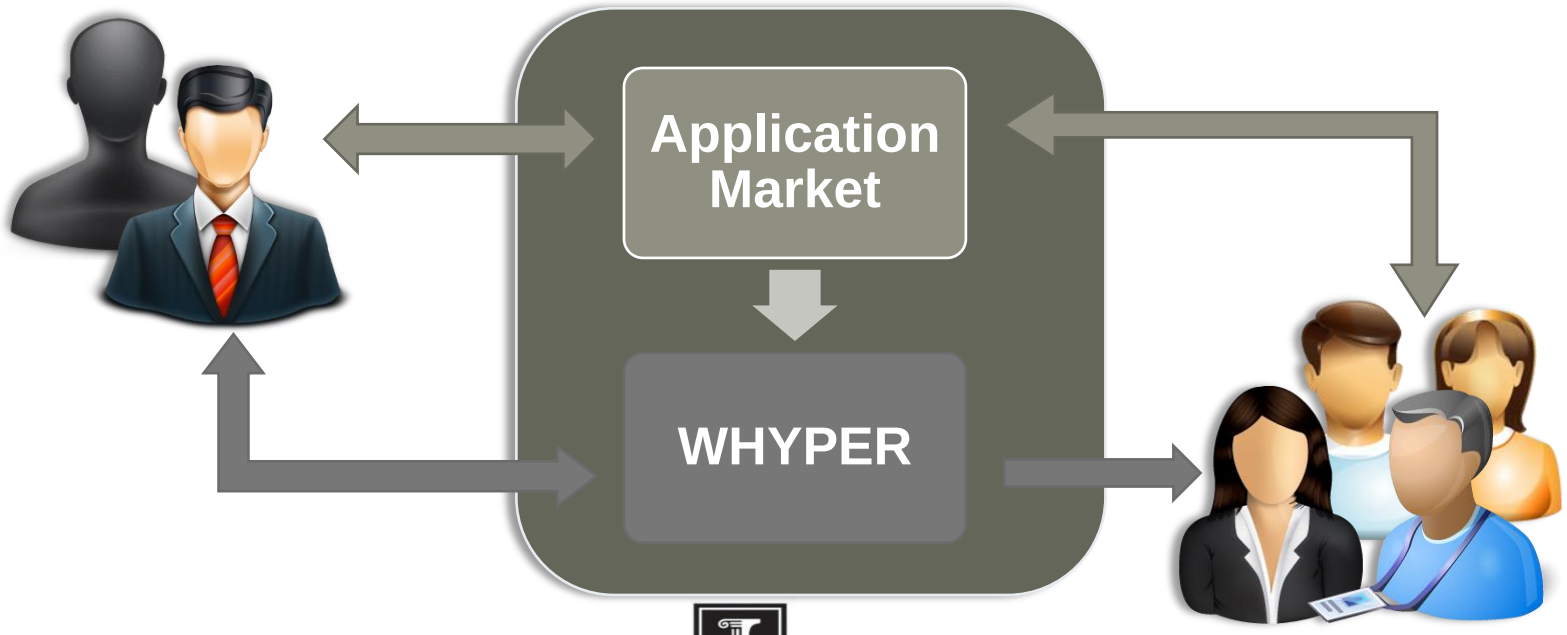
illinois.edu

Permission



WHYPER Overview

- Enhance **user** experience while installing apps
- Enforce functionality disclosure on **developers**
- Complement program analysis to ensure justifications

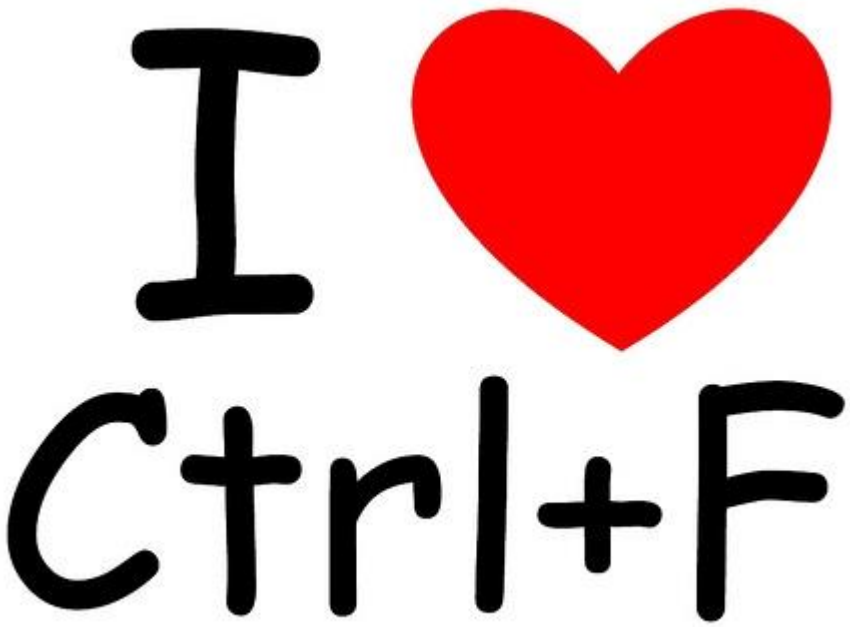


Pandita et al. WHYPER: Towards Automating Risk Assessment of Mobile Applications. *USENIX Security 2013*
http://web.engr.illinois.edu/~taoxie/publications/usenixsec13-whyper.pdf

Example Sentence in App Desc.

E.g., "*Also you can share the yoga exercise to your friends via Email and SMS.*"

- Implication of using the contact permission
- Permission sentences



Keyword-based search on application descriptions

Problems with Ctrl + F

- **Confounding effects:**
  - Certain keywords such as "**contact**" have a confounding meaning
  - E.g., "... displays user contacts, ..." vs "... contact me at abc@xyz.com".

- **Semantic inference:**
  - Sentences often describe a sensitive operation without actually referring to keywords
  - E.g., "share yoga exercises with your friends via Email and SMS"

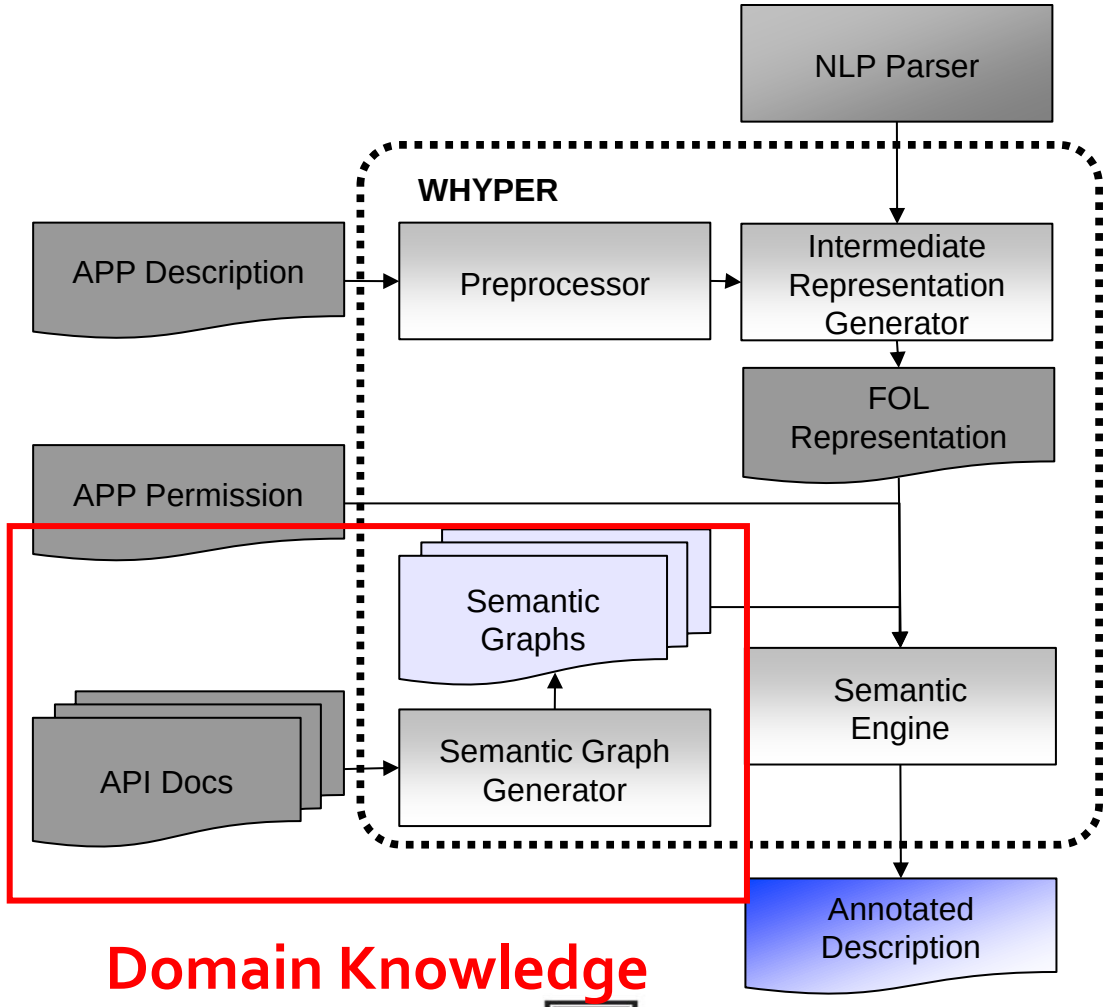
Natural Language Processing

- Natural Language Processing (NLP) techniques help computers understand NL artifacts

- In gene

- NLP styles is feas
  - Te s [FSE 12]
  - AP
  - W ecurity 13]

Overview of ***WHYPER***



Evaluation

- Subjects
  - Permissions:
    - READ_CONTACTS
    - READ_CALENDAR
    - RECORD_AUDIO
  - 581 application descriptions
  - **9,953** sentences
- Evaluation setup
  - Manual annotation of the sentences
  - WHYPER for identifying permission sentences
  - Comparison to keyword-based searching

Evaluation Results

- Precision and recall of ***WHYPER***
  - Average precision (82.8%) and recall (81.5%)

Permission	Keywords
READ_CONTACTS	contact, data, number, name, email
READ_CALENDAR	calendar, event, date, month, day, year
RECORD_AUDIO	record, audio, voice, capture, microphone

- Comparison to keyword-based searching
  - Improving precision (41.6%) and recall (-1.2%)

Result Analysis (False Positives)

- **Incorrect parsing**
  - *"MyLink Advanced provides full synchronization of all Microsoft Outlook emails (inbox, sent, outbox and drafts), contacts, calendar, tasks and notes with all Android phones via USB"*
- **Synonym analysis**
  - *Ex non-permission sentence: "You can now turn recordings into ringtones."*
    - *functionality that allows users to create ringtones from previously recorded sounds but NOT requiring permission to record audio*
    - *false positive due to using synonym: (turn, start)*

Result Analysis (False Negatives)

- **Incorrect parsing**
  - Incorrect identification of sentence boundaries and limitations of underlying NLP infrastructure
- **Limitations of Semantic Graphs**
  - *Ex. permission sentence: "blow into the mic to extinguish the flame like a real candle"*
    - *false negative due to failing to associate "blow into" with "record"*
  - Automatic mining from user comments and forums

Our Yin-Yang View on Mobile App Security

User-Perceived Information

- Reason about user-perceived info, e.g., WHYPER (⬆)
- Push app security behavior across the boundary, e.g., AppContext (➔)
- Check **consistency** across the boundary (🡐➔)
- Reduce user judgment effort (⬇)

App Security Behavior

App Description [functional]

App UIs, App categories, App metadata, User forums, …

App Code

App Permissions [security]

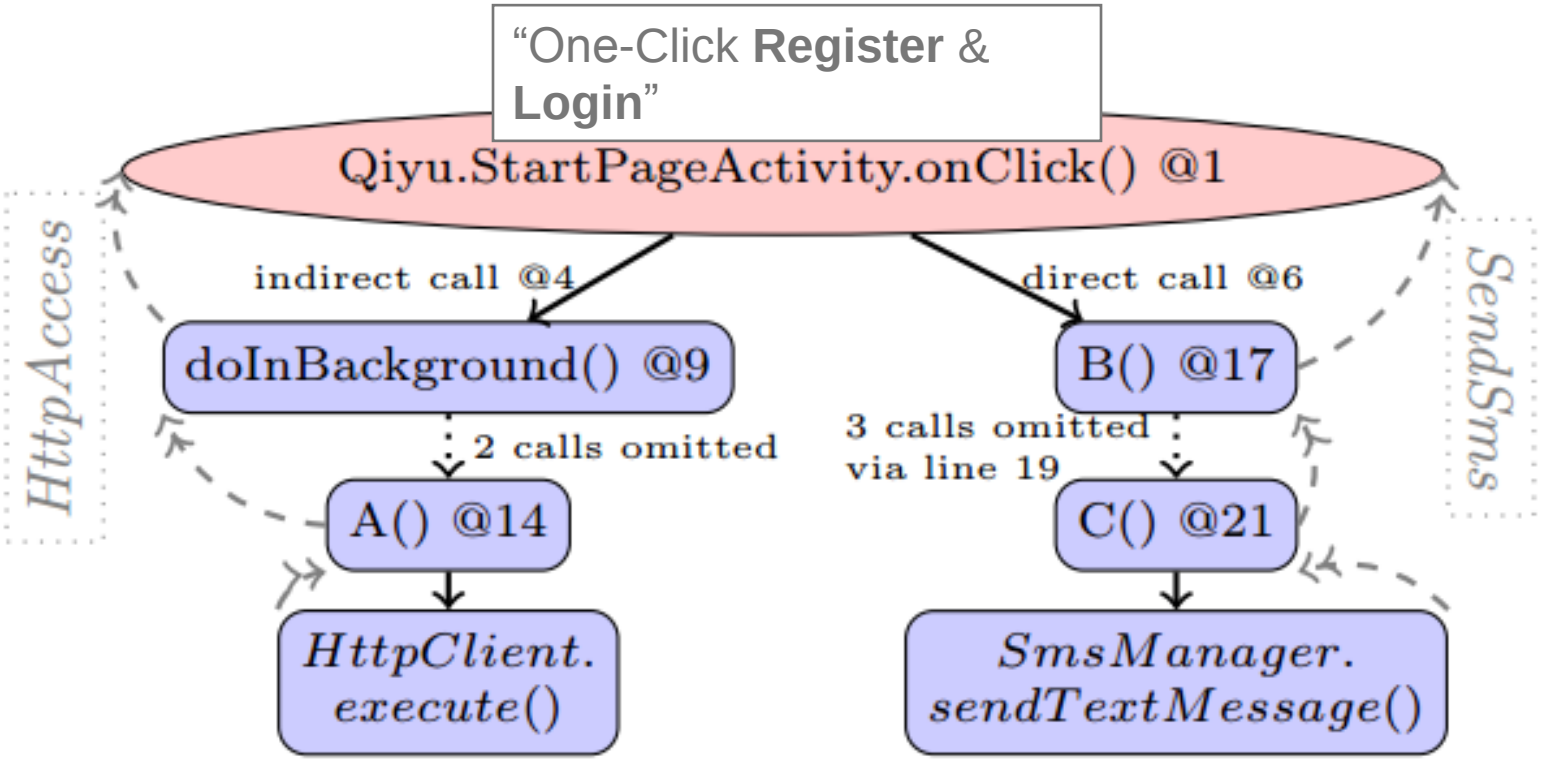
illinois.edu

Related Work

- **AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction**. Jianjun Huang, Xiangyu Zhang, Lin Tan, Peng Wang, and Bin Liang. ICSE 2014.

- **Checking App Behavior Against App Descriptions**. Alessandra Gorla and Ilaria Tavecchia and Florian Gross and Andreas Zeller. ICSE 2014.

AsDroid: Detecting Stealthy Behaviors in Android Applications [ICSE'14 Huang et al.]



(b) Call Graph and Intent Propagation

Figure 1: Motivating Example in app *Qiyu*.

CHABADA: Checking App Behavior Against App Descriptions [ICSE'14 Gorla et al.]

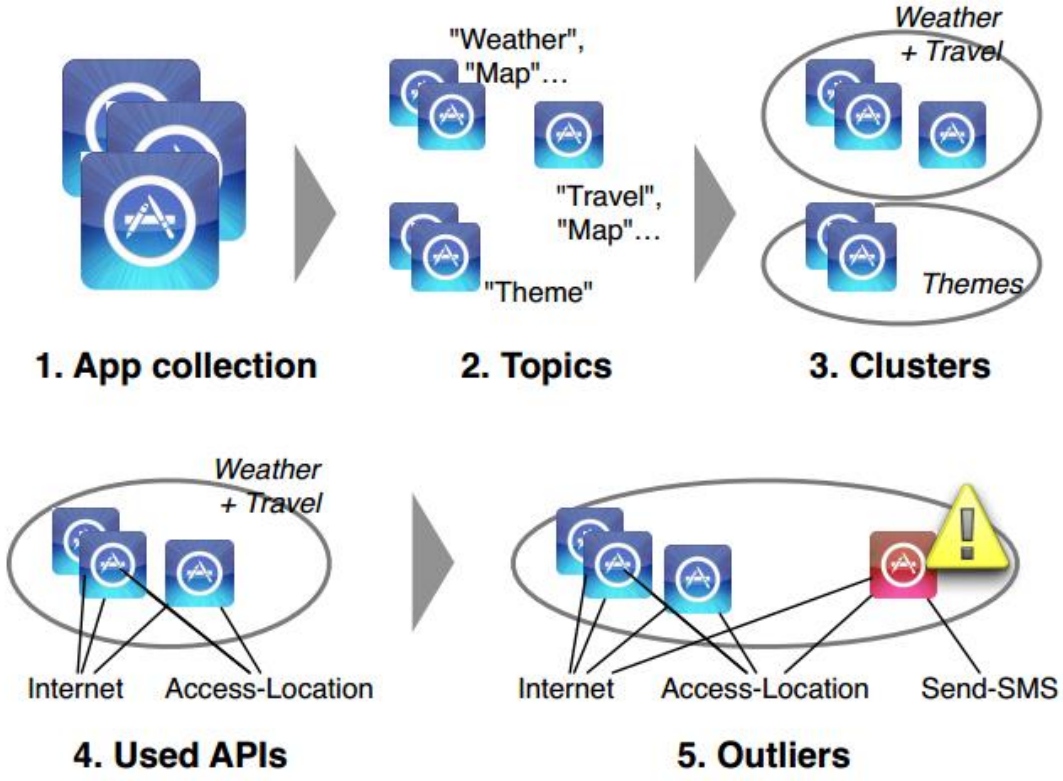


Figure 1: Detecting applications with unadvertised behavior. Starting from a collection of "good" apps (1), we identify their description topics (2) to form clusters of related apps (3). For each cluster, we identify the sentitive APIs used (4), and can then identify outliers that use APIs that are uncommon for that cluster (5).

Our Yin-Yang View on Mobile App Security

- Reason about user-perceived info, e.g., WHYPER (⬆)
- Push app security behavior across the boundary, e.g., AppContext (➔)
- Check **consistency** across the boundary (🡐➔)
- Reduce user judgment effort (⬇)

User-Perceived Information

App Security Behavior

App Description [functional]

App UIs, App categories, App metadata, User forums, …

App Code

App Permissions [security]

illinois.edu

How to Define Malicious Behavior?

– What makes a behavior (app) malicious?

– Existing techniques
  - Permissions
  - API Method Calls
  - Information Flows

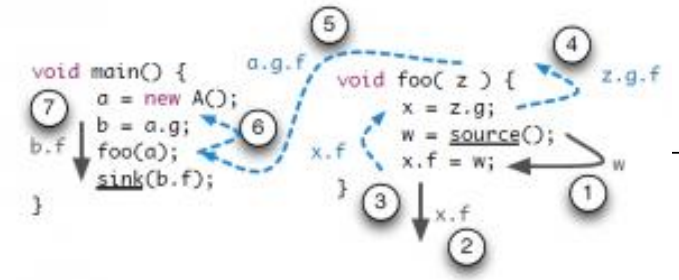
This application has access to the following:

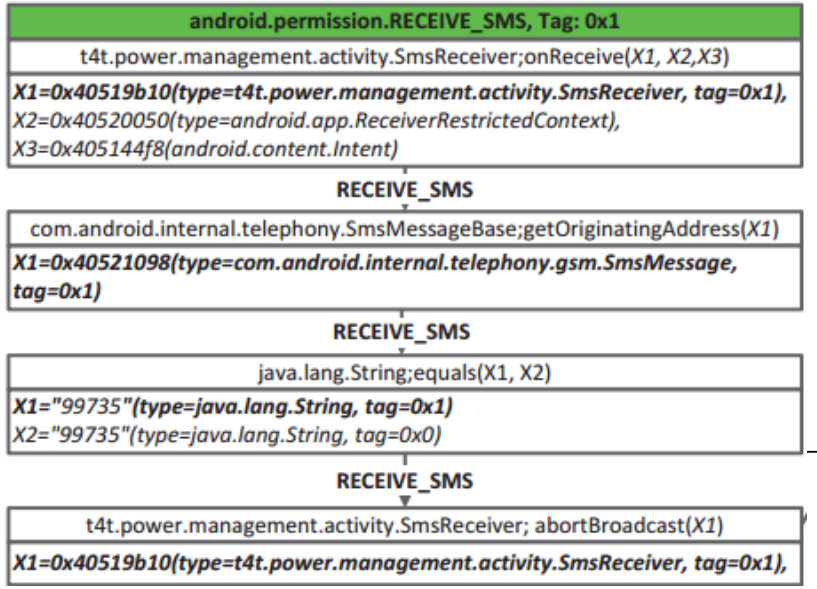
Storage
modify/delete SD card contents

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





How to Define Malicious Behavior?: Examples

– Sending a text msg to a premium number to charge money

Being legitimate payment method for unlocking game features in Andriod. (Same Permission, Same API)

– Taking all of your contacts and sends them to some server

Being done by WhatsApp upon initialization (acquired by Facebook for $19 billion in Feb 14) (Same Permission, Same API, Same Information flow)

– Tracking your current position

Being done by TapSnake (a clone of the Snake game, also a spyware to track a phone's location) (Same Permission, Same API, Same Information flow)

Motivation

Fundamental difference between malware & benign apps:

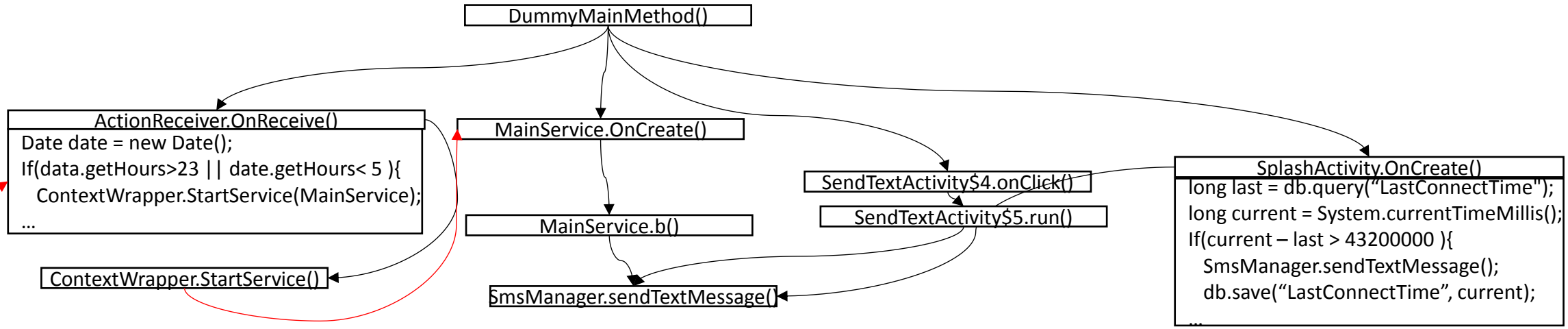
different **design principles**

- Benign apps
  - Meet requirements from users (as delivering utility)
- Malware
  - Meet requirements from users (as covering up)
  - Trigger executing malicious behaviors **frequently** to seek max benefits (as delivering "**utility**")
    - Evade detection to prolong lifetime (as covering up)

Motivation - cont.

- Mobile malware leverage two major mobile-platform features
  - **Frequent** occurrences of **imperceptible** system events
    - E.g., many malware families **trigger** malicious behaviors via background events; in contrast, UI events activate when users using the app ➔ users are **around**!!
  - **Indicative** changes in external environments ➔ users not **around**!!!
    - E.g., DroidDream malware families **suppress**/**trigger** malicious behaviors during **day**/**night** time
- Malware strive to reach a *balance* between **prolonging** life time and **increasing** invocation chance, e.g., malicious behaviors invoked
  - **frequently enough** to meet the need, e.g., a few clicks/day from the device to improve search engine ranking of website X
  - **not too frequently/not wrong timing** for users to notice anomaly

Example

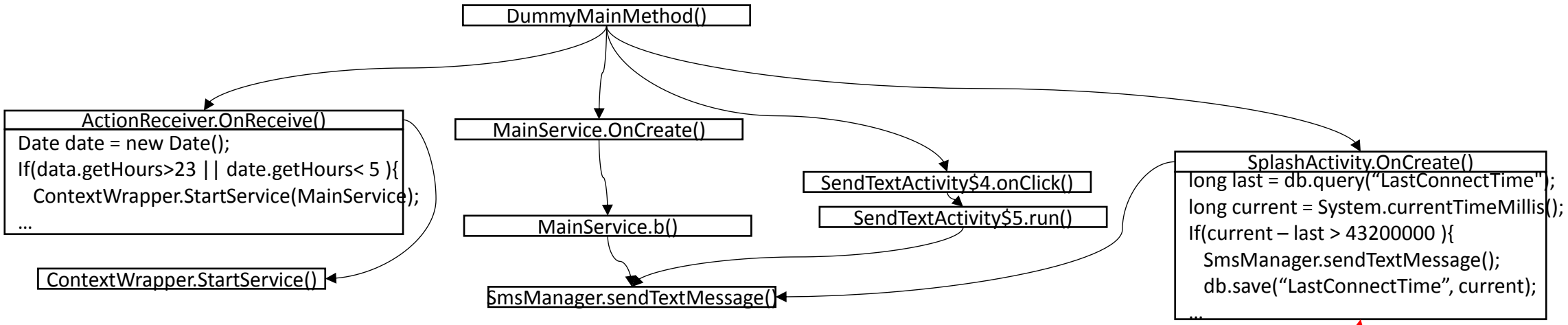



```
<activity android:label="@string/app_name" android:name=".SplashActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<service android:name="com.android.main.MainService" android:process=":main" />
<receiver android:name="com.android.main.ActionReceiver">
    <intent-filter>
        <action android:name="android.intent.action.SIG_STR" />
    </intent-filter>
</receiver>
```

The app will send a SMS when

- phone signal strength changes and
- current time is within 11PM-5 AM

Example

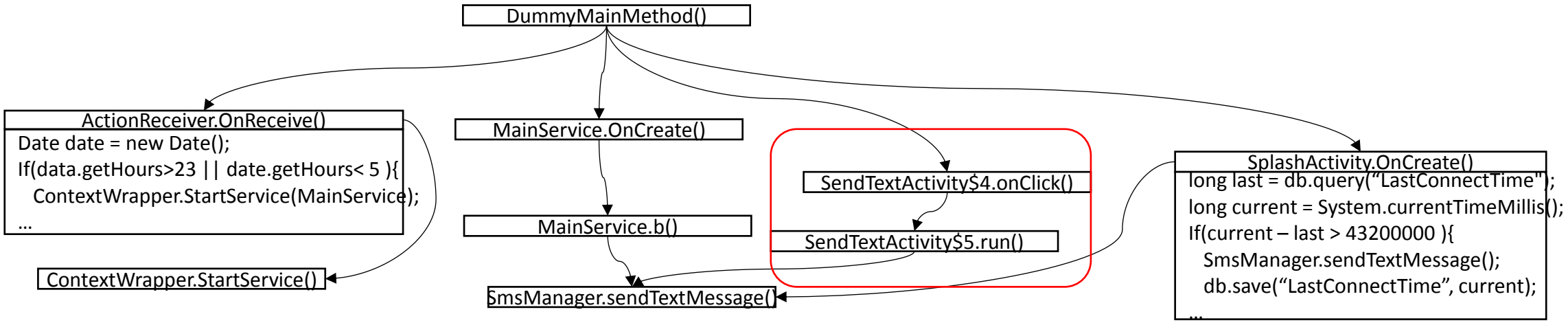



```
<activity android:label="@string/app_name" android:name=".SplashActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<service android:name="com.android.main.MainService" android:process=":main" />
<receiver android:name="com.android.main.ActionReceiver">
    <intent-filter>
        <action android:name="android.intent.action.SIG_STR" />
    </intent-filter>
</receiver>
```

The app will send a SMS when

- user enters the app and
- (current time – time when last msg sent) >12 hours

Example

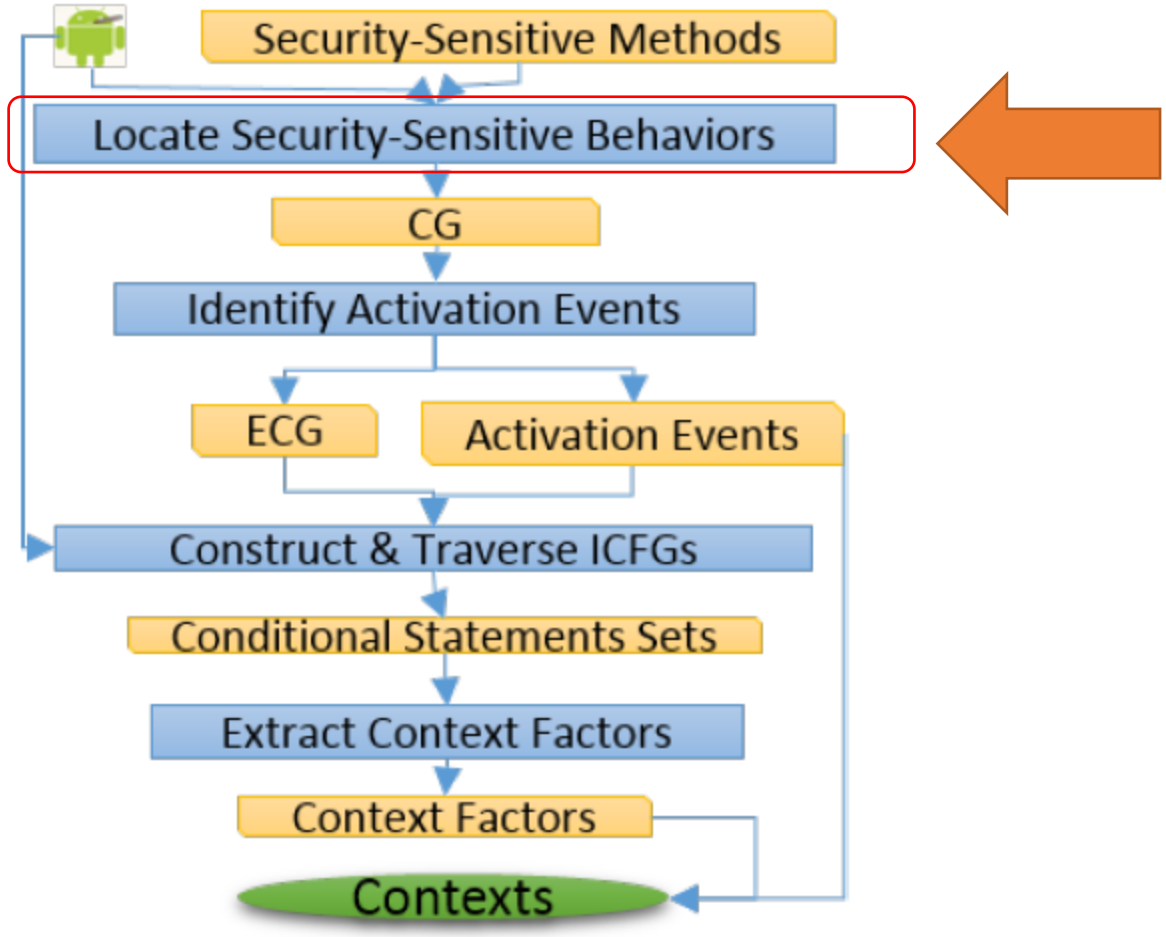



```
<activity android:label="@string/app_name" android:name=".SplashActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<service android:name="com.android.main.MainService" android:process=":main" />
<receiver android:name="com.android.main.ActionReceiver">
    <intent-filter>
        <action android:name="android.intent.action.SIG_STR" />
    </intent-filter>
</receiver>
```

The app will send a SMS when

- user clicks a button in the app

AppContext [Yang et al. ICSE 2015]

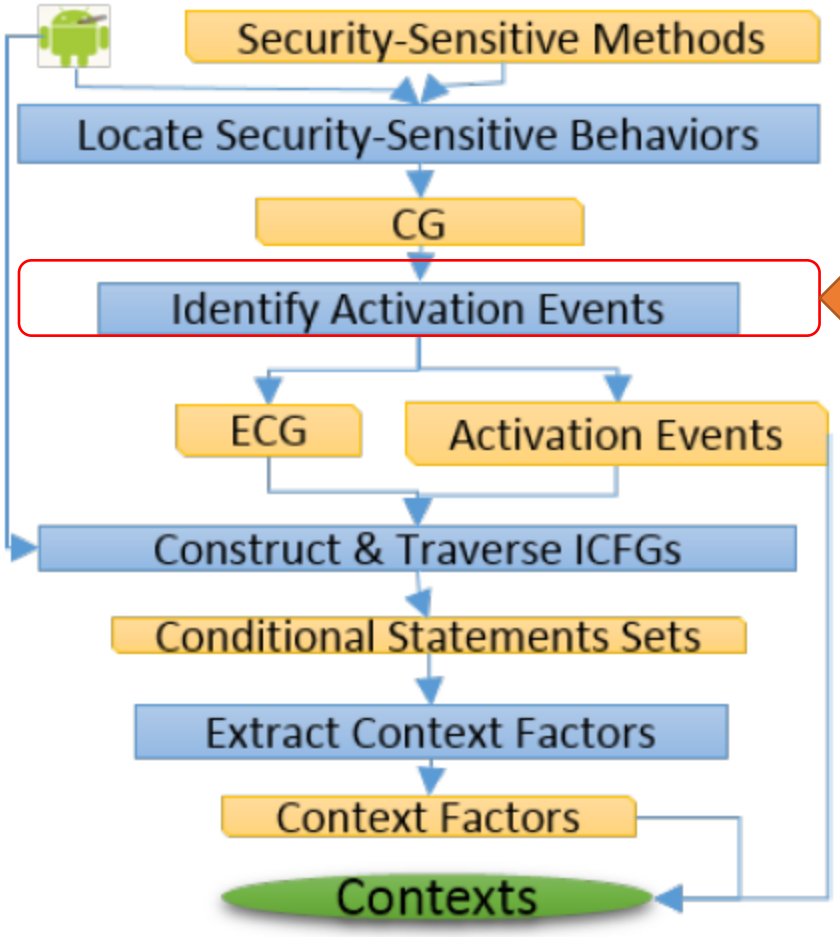


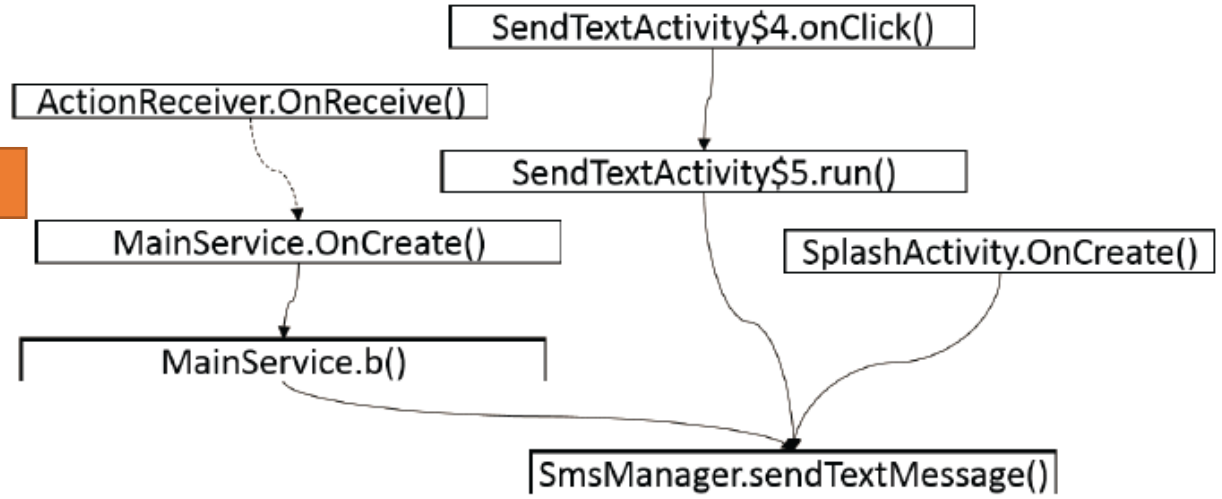
Permission protected API methods

Sources & Sinks of information flows

Reflections and dynamic code loading

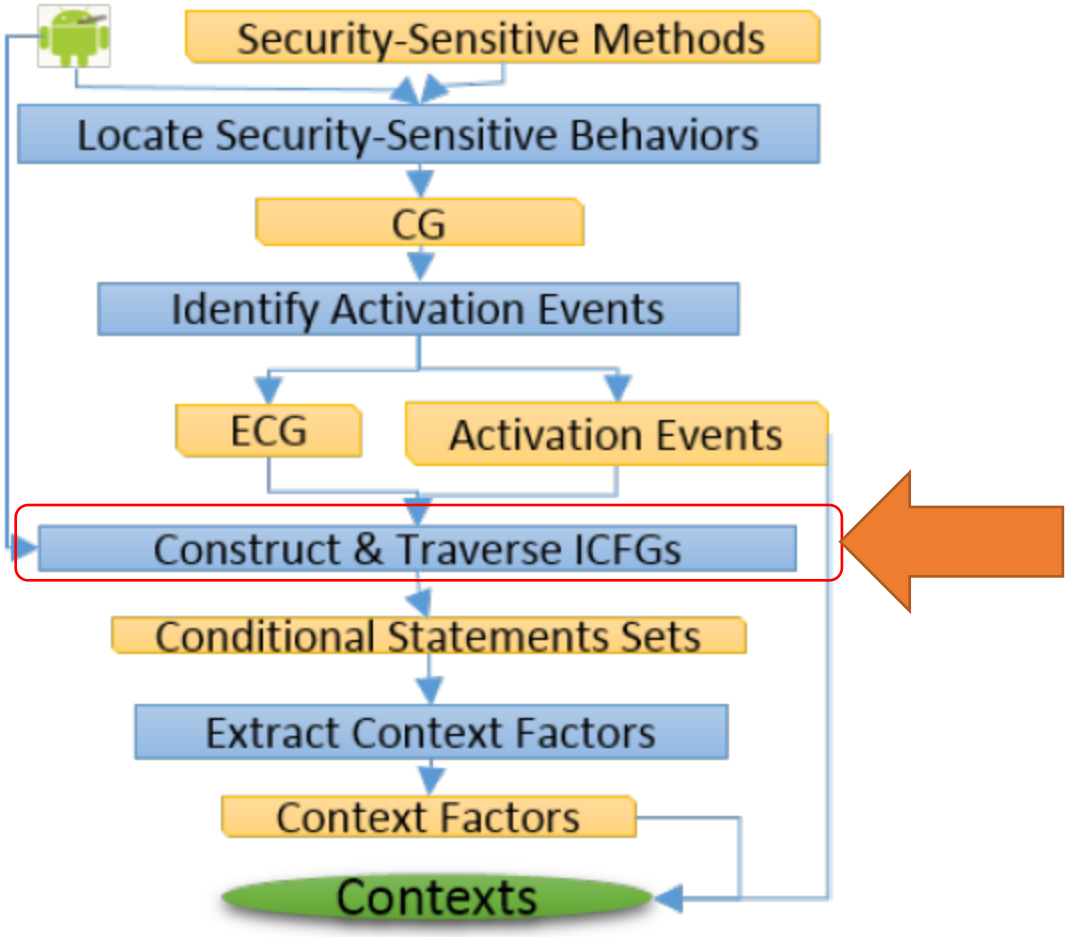
AppContext





Activation event1: Signal strength changes
Activation event2: Entering the app
Activation event1: Clicking a button

AppContext



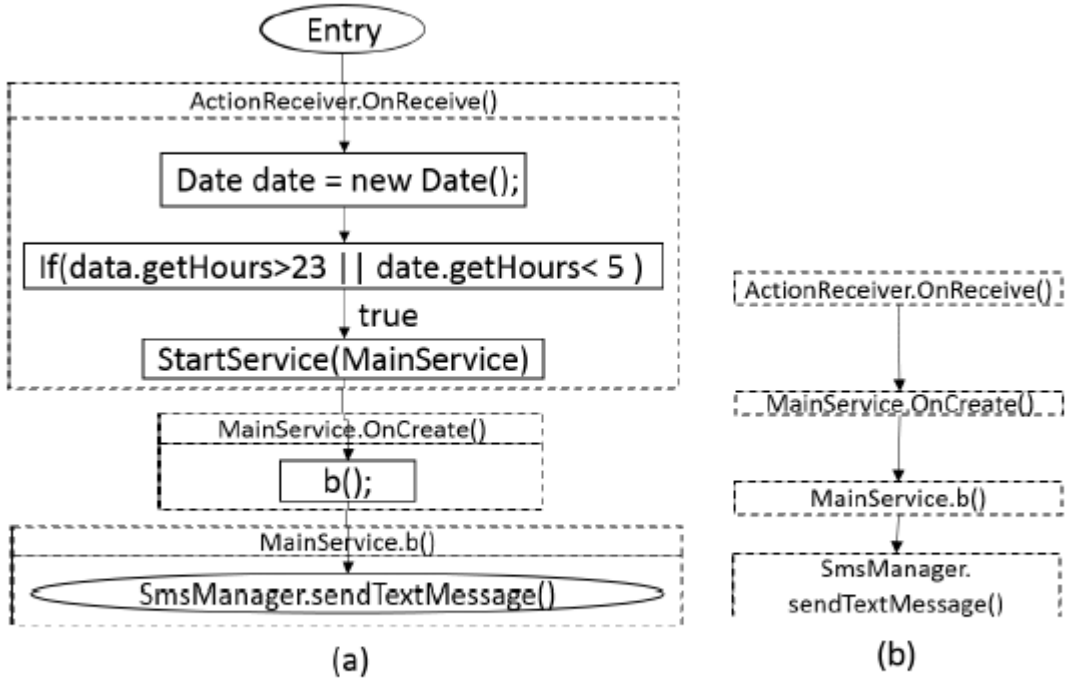
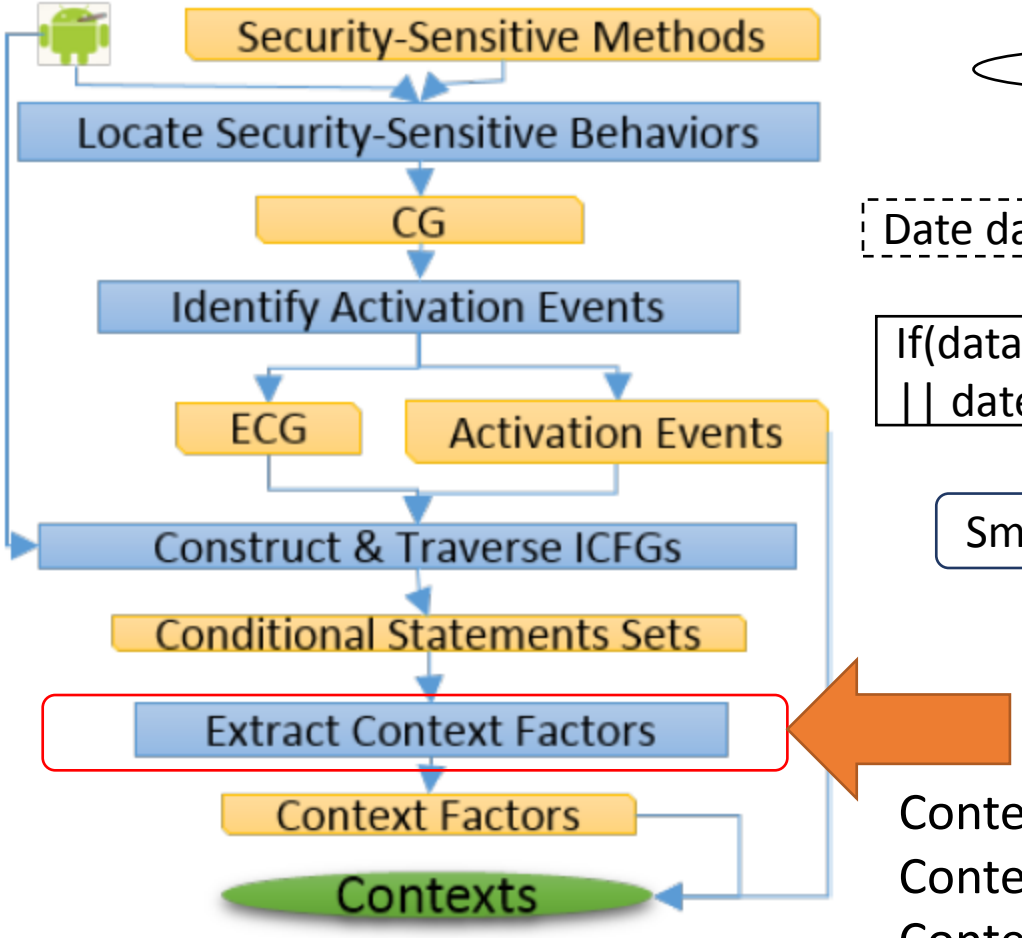
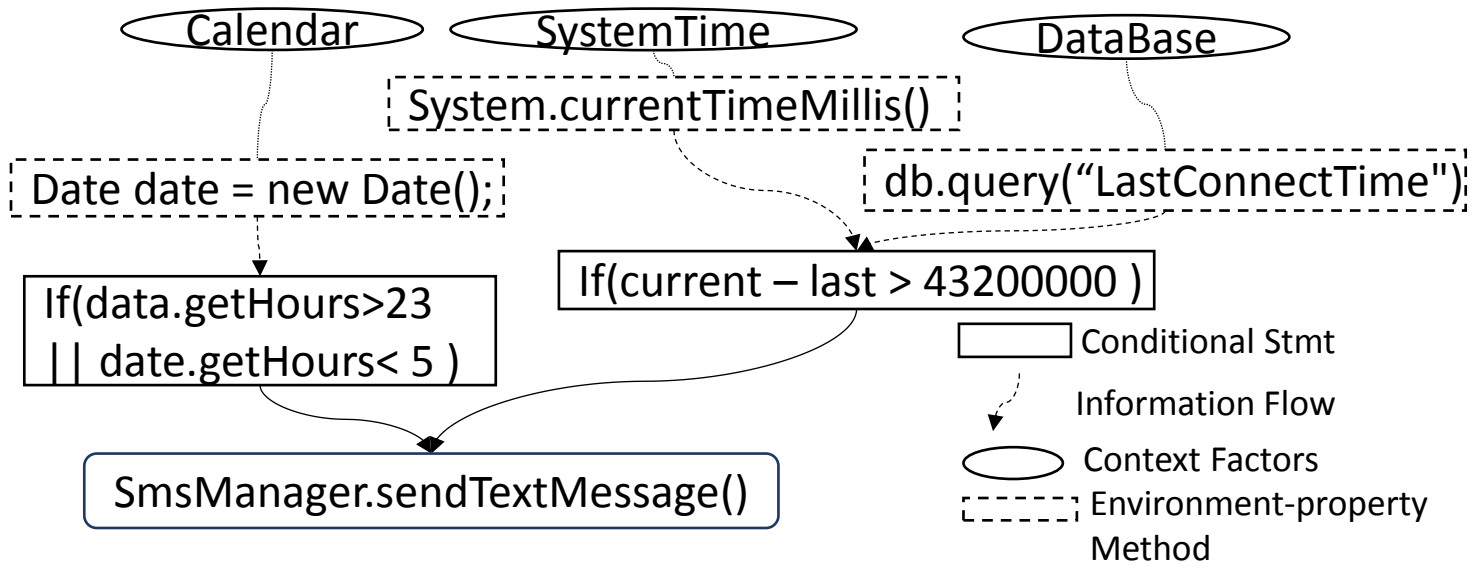


Fig. 4. An ICFG (a) and its corresponding call path in ECG (b)

AppContext





Context1: (Event: Signal strength changes), (Factor: Calendar)
Context2: (Event: Entering app), (Factor: Database, SystemTime)
Context3: (Event: Clicking a button)

Context factors: environmental attributes for affecting security-sensitive behavior's invocation (or not)

Context-based Security-Behavior Classification

TABLE I
LIST OF FEATURES FOR CLASSIFICATION

Features of Behavior Information		
Permission	Security-sensitive method call	
Features of Activation Event		
SystemUI event	System event	UI event
Features of Context Factors		
List of environmental attributes		

Step 1. Transform contexts for each app's security behavior as features

Step 2. Label each behavior in training set as malware or benign

Step 3. Learn a predictive model via ML technique, e.g., support vector machine (SVM)

Step 4. Classify an unlabeled behavior as malware or benign based on the model

Addressing Challenges:

- Vague and subjective by nature when reasoning about maliciousness of a behavior
- Difficult to determine a proper threshold

Permission	Method Call	SystemUI	System	UI	F_1	F_2	F_3*	F_4*	F_5*	F_6	...	F_{142}
SEND_SMS	*sendTextMessage*	N/A	SIG_STR	N/A	0	0	1	0	0	0	...	0
SEND_SMS	*sendTextMessage*	EnterApp	N/A	N/A	0	0	0	1	1	0	...	0
SEND_SMS	*sendTextMessage*	N/A	N/A	Click	0	0	0	0	0	0	...	0

* F_3 = Calendar, F_4 = System Time, F_5 = Database

Evaluation

- Security behaviors as all security-sensitive method calls
- Security behaviors as reflective or dynamic code loading method calls
  - Among all 922 malicious **reflective** method calls, accurately identify 872 malicious method calls and misidentify 180 benign method calls
    82.9% precision, 94.5% recall
  - Of all 787 malicious **dynamic code loading** method calls, accurately identify 710 malicious method calls and misidentify 137 benign method calls
    83.8% precision, 90.2% recall

Main Results

TABLE III
MALICIOUS SECURITY-SENSITIVE BEHAVIORS IDENTIFIED BY APPCONTEXT

Features Used	P(%)	R(%)
Complete Context (C)	94.8	84.8
Behavior Information (B)	79.0	37.3
Activation Events (E)	83.2	49.5
Context Factors (F)	90.6	71.2
B & E	88.0	71.3
B & F	90.2	76.9
E & F	92.5	77.3

TABLE IV
IDENTIFICATION OF MALWARE BY APPCONTEXT

Features Used	TP	FP	FN	P(%)	R(%)
Complete Context (C.)	192	27	10	87.7	95.0
Behavior Information (B.)	169	78	33	68.4	83.6
Activation Events (E.)	163	78	39	67.6	80.6
Context Factors (F.)	150	26	52	85.2	74.2
B. & E.	193	63	9	75.3	95.5
B. & F.	180	46	22	79.6	89.1
E. & F.	187	27	15	87.3	92.5

TP = True Positive, FP = False Positive, FN = False Negative
P = Precision, R = Recall

Summary of Findings

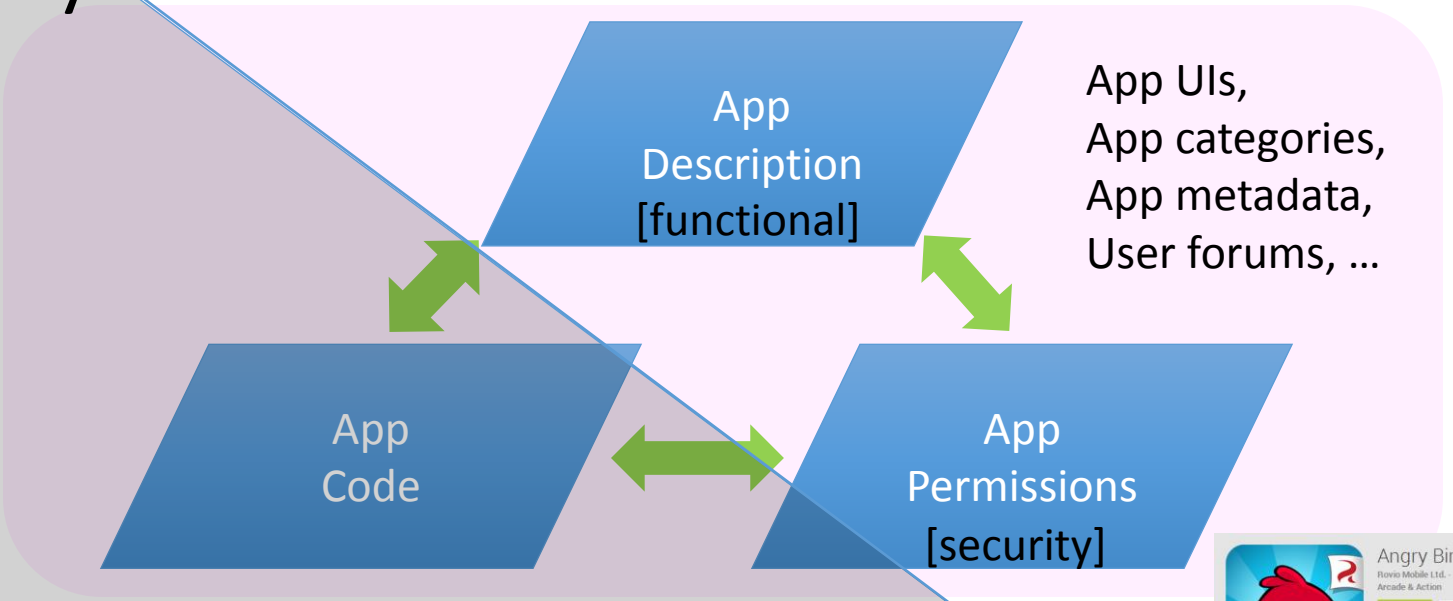
- **Activation events** effectively help identify malicious method calls **without context factors**
- **Context factors** effectively help identify malicious behaviors triggered by **UI events (activation events)**
- **Context factors** also differentiate controls of security-sensitive method calls (e.g., *TelephonyManager.getDeviceId* for reading device info) in benign apps and malware
  - benign apps: read device info only when auto logins are successful
    - Context factors: **info from database or Internet**
  - malware: directly read and send device info to a server
    - Context factors: **NONE**

Road Ahead: Yin-Yang View

User-Perceived Information

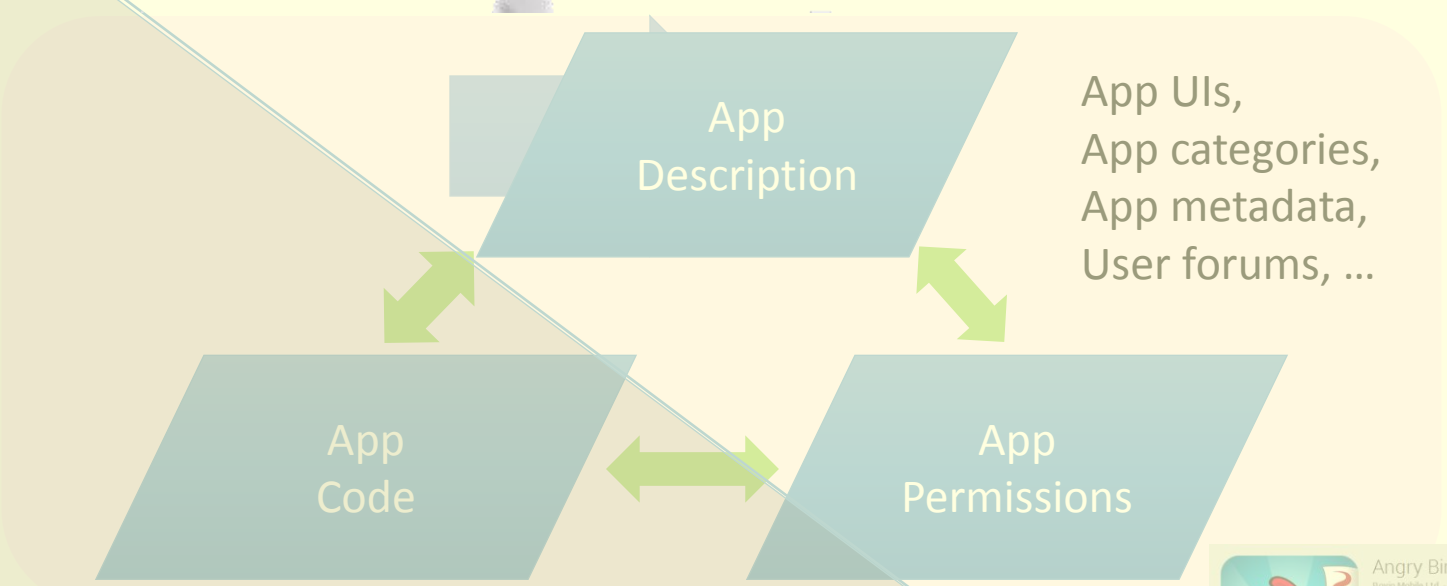
- Reason about user-perceived info, e.g., WHYPER (⬆)
- Push app security behavior across the boundary, e.g., AppContext (➔)
- Check consistency across the boundary (🡐➔)
- Reduce user judgment effort (⬇)

App Security Behavior



App UIs,
App categories,
App metadata,
User forums, …

taoxie@illinois.edu



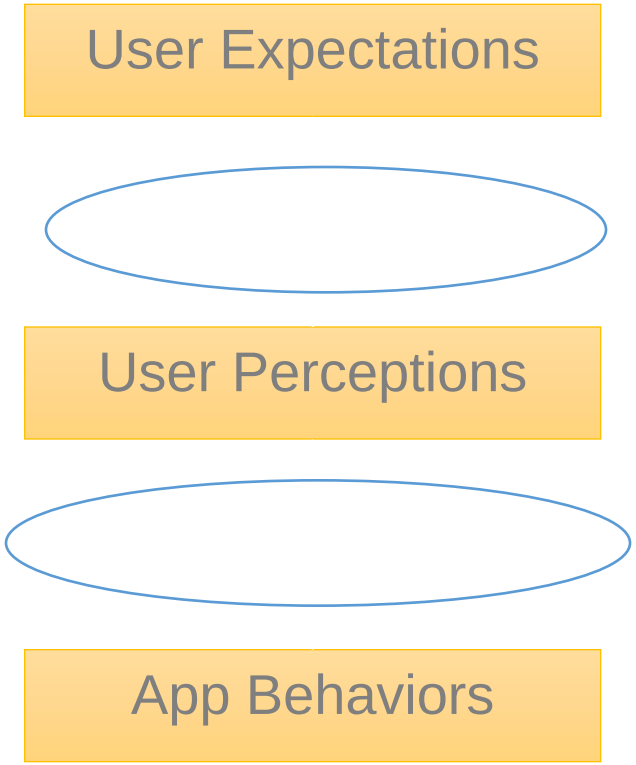
App UIs,
App categories,
App metadata,
User forums, …

Acknowledgments: Supported in part by NSA Science of Security (SoS) Lablet

User Expectations and App Behaviors

- **User expectations** are reflected via **user perceptions** of **app behaviors** (in combination with user judgments)

- There are ***gaps*** btw. user expectations and application behaviors
  - Some application behaviors may be user imperceptible, or contradict w/ user perceptions
  - The user/inspector may not be able to make right judgments based on perceived information



Bridging User Expectations and Application Behaviors

- **User expectations** are reflected via **user perceptions** of **app behaviors** (in combination with user judgments)

- There are ***gaps*** btw. user expectations and application behaviors
  - Some application behaviors may be user imperceptible, or contradict w/ user perceptions
  - The user/inspector may not be able to make right judgments based on perceived information

User-Aware Privacy Control
ASE 2012

AppContext
ICSE 2015

WHYPER
USENIX Security 2013