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)
Mobile App Markets

Apple App Store  Google Play  Microsoft Windows Phone
App Store beyond Mobile Apps!

Make more money on your terms.

Revenue sharing up to:
- Windows 80%
- Apple 70%
- Google 70%

Meet Windows 8.1

[Download] [Learn More]
App Store within Mobile App!
What If Formal Specs Are Written?!

APP DEVELOPERS
- App Functional Requirements
- App Security Requirements
- informal: app description, etc.
- permission list, etc.

APP USERS
- User Functional Requirements
- User Security Requirements
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 level 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 level
App Security Requirements:
Permission List

- System tools
- Storage
- Your location
- Phone calls

This app's permissions have changed to have access to the following:
NEW: Prevent phone from sleeping
Modify/delete SD card contents
Coarse (network-based) location
Read phone state and identity
What If Formal Specs Are Written?!

APP DEVELOPERS

informal: app description, etc.

permission list, etc.

App

Functional
Requirements

App

Security
Requirements

User

Functional
Requirements

User

Security
Requirements

APP USERS

illinois.edu
Example Android App: Angry Birds

Why does Angry Birds need to know who I call?

Rovio explains the new permission that Angry Birds Seasons requires.

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?!

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

- 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 Code
- App Permissions [security]
- App Description [functional]
- App UIs, App categories, App metadata, User forums,

illinois.edu
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**
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
WHYPER Overview

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


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 general, NLP is still difficult

- NLP on domain specific sentences with specific styles is feasible
  - Text2Policy: extraction of security policies from use cases [FSE 12]
  - APIInfer: inferring contracts from API docs [ICSE 12]
  - WHYPER: domain knowledge from API docs [USENIX Security 13]
Overview of WHYPER

APP Description

APP Permission

API Docs

WHYPER

Preprocessor

Intermediate Representation Generator

FOL Representation

Semantic Engine

Semantic Graphs

Semantic Graph Generator

Annotated Description

Domain Knowledge
Evaluation

• Subjects
  – Permissions:
    • READ_CONTACTS
    • READCALENDAR
    • 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%)

<table>
<thead>
<tr>
<th>Permission</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_CONTACTS</td>
<td>contact, data, number, name, email</td>
</tr>
<tr>
<td>READCALENDAR</td>
<td>calendar, event, date, month, day, year</td>
</tr>
<tr>
<td>RECORD_AUDIO</td>
<td>record, audio, voice, capture, microphone</td>
</tr>
</tbody>
</table>

• 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 Permissions [security]

- App UIs, App categories, App metadata, User forums,
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.]

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 sensitive 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

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 Code
- App Permissions [security]
- App Description [functional]
- App UIs, App categories, App metadata, User forums, …
How to Define Malicious Behavior?

– What makes a behavior (app) malicious?

– Existing techniques
  • Permissions
  • API Method Calls
  • Information Flows
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 Android. (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
The app will send a SMS when
- phone signal strength changes and
- current time is within 11PM-5 AM
The app will send a SMS when
• user enters the app and
• (current time – time when last msg sent) >12 hours
Example

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

Security-Sensitive Methods

Locate Security-Sensitive Behaviors

CG

Identify Activation Events

ECG

Activation Events

Construct & Traverse ICFGs

Conditional Statements Sets

Extract Context Factors

Context Factors

Entry

ActionReceiver.onReceive()

Date date = new Date();

if(data.getHours>23 || data.getHours<5) true

StartService(MainService)

MainService.onCreate()

b();

MainService.b()

SmsManager.sendTextMessage()

(a)

SmsManager.sendTextMessage()

(b)

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

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

Context1: (Event: Signal strength changes), (Factor: Calendar)
Context2: (Event: Entering app), (Factor: Database, SystemTime)
Context3: (Event: Clicking a button)
Context-based Security-Behavior Classification

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
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
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 (leftrightarrow)
- Reduce user judgment effort (↓)

App Security Behavior

App Description [functional]

App Code

App Permissions [security]

- 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