DECAF: Detecting and Characterizing Ad Fraud in Mobile Apps

Bin Liu
Suman Nath, Ramesh Govindan, Jie Liu
The Mobile Ad Ecosystem

App Developer

Paid by Impressions

Paid by User Clicks

App User

See/Click Ads

Phone/Tablet App

Ad Network

Ad Plugin

Paid by User Clicks

Paid by Impressions

Introduction

DECAF

Evaluation

Characterization

Conclusion
App developers have incentive to commit fraud by inflating clicks and impressions
Ad Fraud: a Big Business

Very large mobile marketplaces

1 billion dollars lost due to ad fraud in 2013
We explore a sub-class of ad fraud, called placement ad fraud

Developers manipulate visual layouts to trigger invisible impressions or unintentional clicks

Microsoft Advertising Prohibits Placement Ad Fraud

“A developer must not edit, resize, modify, filter, obscure, hide, make transparent, or reorder any advertising“
Intrusive ads
Placement Ad Fraud Examples

Many ads

Introduction  DECAF  Evaluation  Characterization  Conclusion
Placement Ad Fraud Examples

Hidden ads

Scoreboard

scores
Final

Alabama 42 Notre Dame 14

AL 14 14 7 7
ND 0 0 7 7

Introduction  DECAF  Evaluation  Characterization  Conclusion
Current Approach

Manual inspection, which is labor-intensive and error-prone

Several tens of minutes to manually scan one app

Cannot detect some placement ad fraud, like hidden ads
To design an automated system for detecting placement fraud
Challenge 1: Scaling to thousands of visually complex apps

Tradeoff between a more complete search (accuracy) and processing more apps in a given time (scalability)
Challenges

Challenge 2: accurately and quickly identify fraud

“Sliding Screen” Problem (in a Panoramic Page)
Challenges

Challenge 2: accurately and quickly identify fraud

Hidden Ads (Z-index not Available)
Our Approach – Dynamic Analysis

Use UI-automation based dynamic analysis to detect placement fraud

Dealing with Visual Complexity

Develop automated scalable navigation of app pages through dynamic execution

Accurate Fraud Detection

Design several efficient fraud detectors, one for each fraud type
Contributions

Design and implementation of the DECAF system to detect placement fraud

Characterization of placement fraud by analyzing 50,000 Windows Phone apps and 1,150 tablet apps using DECAF

Deployment of DECAF in the ad fraud team at Microsoft, which has helped detect many instances of fraud
DECAF Overview

DECAF
Automated UI Navigation (Monkey)

Movie
ShowTime

Fraud Checkers

Many Ads

Hidden Ads

Introduction
DECAF
Evaluation
Characterization
Conclusion
Automated UI Navigation

Introduction

DECAF

Evaluation

Characterization

Conclusion

The set of previously visited pages

UI Extraction

UI Extraction Channel

UI Action Channel

UI Action

Next App Page

Chest pain

- pain
- cold sweat
- weakness
- difficult breathing
- anxiety
- semi-sitting position
- loosen the tightening
- avoid unnecessary movement
- DO NOT give medication

Click “Continue” Button

Monkey

Page

Text

Image

List

Button

UI Structure

Click

Button

Click

Swipe

List

Scroll

Action Dictionary
Problem: Reducing the Search Space

Avoid clicking UI elements on previously visited pages?

UI page space can be practically infinite

The post list is updated every several minutes

One Reddit App Page
Reducing the Search Space

Key observation

For placement fraud, it is sufficient to visit structurally dissimilar pages.

Two pages can be structurally similar even if their content differs.
Structurally Similar Pages!

Introduction

DECAF

Evaluation

Characterization

Conclusion
Determining Structural Similarity

Two pages are structurally similar if they have “similar” UI hierarchies
Two pages are structurally similar if they have “similar” UI hierarchies.

Structurally Dissimilar
Defining Structural Similarity

Feature vector defined on UI elements

Encodes type of UI element and position in hierarchy

Feature vector defined on UI elements

Use cosine similarity metric

Users specify a similarity threshold

Introduction  DECAF  Evaluation  Characterization  Conclusion
Problem: Avoid previously visited states

Monkey can waste time by visiting previously visited pages

The Monkey needs to anyway go back to page 1 again and click button 2
Avoiding Previously Visited States

To avoid backtracking costs, can we predict if two buttons on a page lead to structurally similar pages?
Avoiding Revisiting Similar Pages

Our method is to use machine learning classifiers

Two buttons that have a similar neighborhoods in UI hierarchy likely to lead to structurally similar pages
DECAF Overview

DECAF

Automated UI Navigation (A Monkey)

Movie ShowTime

DECAF

Many Ads

Fraud Checkers

Hidden Ads

Introduction

DECAF

Evaluation

Characterization

Conclusion
## Fraud Checkers

Input to checkers: structural data of ad and non-ad elements

<table>
<thead>
<tr>
<th>Fraud Type</th>
<th>Checker Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invisible/Hidden Ads</td>
<td>Whether visual elements are overlapped with ads</td>
</tr>
<tr>
<td>Smaller Ads</td>
<td>Compare the actual display size of the ad with the minimal valid size</td>
</tr>
<tr>
<td>Intrusive Ads</td>
<td>Compare the distance between an ad and clickable non-ad elements</td>
</tr>
<tr>
<td>Many Ads</td>
<td>Whether the number of viewable ads is more than the maximum allowed</td>
</tr>
</tbody>
</table>
Efficient Many-Ads Checkers

Many ads in one display screen

Challenge: the “sliding screen” problem

No Ad
Efficient Many-Ads Checkers

Many ads in one display screen

Challenge: the “sliding screen” problem

Introduction

DECAF

Evaluation

Characterization

Conclusion
Efficient Many-Ad Checker

Many ads in one display screen

Challenge: the “sliding screen” problem

We have designed an efficient algorithm to detect many-ad fraud
Other Optimizations

State Importance Assessment to further reduce the number of app pages that the Monkey needs to explore

Rendering Order Inference and Proxy-Assisted Screen Analysis to efficiently detect hidden ads
Evaluation and Characterization

**Basic Setup**
- Run DECAF on each app for 20 minutes
- Collect information from structurally different pages

**Structural Page Coverage**
- Manually find ground truth number of page structure
- Limited to 100 top free apps from the tablet app store

---

Introduction | DECAF | Evaluation | Characterization | Conclusion
Evaluation and Characterization

**Basic Setup**

- Run DECAF on each app for 20 minutes
- Collect information from structurally different pages

**Characterization of Placement Ad Fraud**

- Run DECAF on 50,000 phone apps and 1,150 tablet apps
- Characterize fraud by fraud type, rating and publisher
Introduc+on

DECAF

Evalua)on

Characteriza+on

Conclusion

A Basic Monkey

A Classifier-Enhanced Monkey

71 apps finish with the Classifier-Enhanced Monkey, but only 30 finish with the Basic monkey

29 apps cannot finish in 20 minutes

The Monkey fails to recognize some clickable elements

Some scenarios require app-specific text input that Monkey cannot handle

Some apps simply have a very large set of structural pages
## Characterizing Fraud by Types

<table>
<thead>
<tr>
<th>Fraud Type</th>
<th>Phone Apps (1000+)</th>
<th>Tablet Apps (50+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Many Ads</td>
<td>11%</td>
<td>4%</td>
</tr>
<tr>
<td>Smaller Ads</td>
<td>33%</td>
<td>48%</td>
</tr>
<tr>
<td>Hidden Ads</td>
<td>47%</td>
<td>32%</td>
</tr>
<tr>
<td>Intrusive Ads</td>
<td>9%</td>
<td>16%</td>
</tr>
</tbody>
</table>

1,000+ phone apps (out of 50,000) and 50+ tablet apps (out of 1,150) commit at least one fraud
Characterizing Fraud by Types

<table>
<thead>
<tr>
<th>Fraud Type</th>
<th>Phone Apps (1000+)</th>
<th>Tablet Apps (50+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Many Ads</td>
<td>11%</td>
<td>4%</td>
</tr>
<tr>
<td>Smaller Ads</td>
<td>33%</td>
<td>48%</td>
</tr>
<tr>
<td><strong>Hidden Ads</strong></td>
<td><strong>47%</strong></td>
<td><strong>32%</strong></td>
</tr>
<tr>
<td>Intrusive Ads</td>
<td>9%</td>
<td>16%</td>
</tr>
</tbody>
</table>

“Hidden Ads” violations are more prevalent on the phone, which has a smaller screen for displaying content
Characterizing Fraud by Types

<table>
<thead>
<tr>
<th>Fraud Type</th>
<th>Phone Apps (1000+)</th>
<th>Tablet Apps (50+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Many Ads</td>
<td>11%</td>
<td>4%</td>
</tr>
<tr>
<td>Smaller Ads</td>
<td>33%</td>
<td>48%</td>
</tr>
<tr>
<td>Hidden Ads</td>
<td>47%</td>
<td>32%</td>
</tr>
<tr>
<td>Intrusive Ads</td>
<td>9%</td>
<td>16%</td>
</tr>
</tbody>
</table>

“Intrusive Ads” violations are more prevalent on the tablet, which has richer controls to be used to trigger accidental clicks.
Characterizing Fraud by Rating

Rating values are rounded to a number from 1-5

Fraud level does not seem to depend on rating

- Fraud (phone)
- NoFraud (Phone)

- Fraud (Tablet)
- NoFraud (Tablet)
Introducing DECAF

**Characterization**

The distribution of the number of fraud across publishers who commit fraud exhibits a heavy tail.

Each app is developed by a publisher.

A small number of publishers are responsible for most of the fraud.

- **Publisher ID**
  - Linear scale
  - Range: 1 to 1000

- **# Fraudulent apps**
  - Logarithmic scale
  - Range: 1 to 1000

**Fraudulent App Count per Publisher**
Mobile ad fraud is a 1 billion dollar business, and ad networks need effective tools to detect fraud.
Conclusion

DECAF: a system for detecting placement ad fraud in mobile apps

- Efficiently explore structurally different pages of mobile apps
- Accurately detect placement ad fraud in a fast and scalable way
- Case study of 51,150 apps reveals interesting variability in the prevalence of fraud by type, rating, publisher and etc.