PUMA: Programmable UI Automation for Large-Scale Dynamic Analysis of Mobile Apps

Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, Ramesh Govindan
Mobile App Explosion

Number of Apps

Source: http://en.wikipedia.org/wiki/Google_Play

1.2 million
App Behaviors are Complex

Some iPhone Users Complaining Of iOS 6.1 Battery Drain Problem

Source: forbes.com

Amazon App Store sluggish & killing my battery

Source: droidforums.net

Fake Gmail Android application steals personal data

Summary: Mobile security researchers from NQ Mobile have intercepted a fake Gmail Android application dubbed DDSpy.

Source: zdnet.com
Dynamic Analysis

These app behaviors can be studied by dynamic analysis.

But given the huge number of apps,

We need **scalable dynamic** analysis methods.
Dynamic Analysis with UI Automation

- UI-driven app exploration (or Monkey)

Source: http://mttnow.com/ios-automated-ui-testing
Monkey: High-level Idea

UI events

UI info

Software

Monkey
Monkey in Dynamic Analysis

Example: Check number of Ads on each page
Monkey: Overview

UI Extraction Channel

UI Extraction

UI Action Channel

UI Action

Decide UI Action

Action Dictionary

<table>
<thead>
<tr>
<th>Button</th>
<th>Click</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipage</td>
<td>Scroll</td>
</tr>
<tr>
<td>List</td>
<td>Scroll</td>
</tr>
</tbody>
</table>

The set of previously visited pages
## Monkey-based App Analyses

<table>
<thead>
<tr>
<th>Tool</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMC</td>
<td>Check accessibility of UI elements</td>
</tr>
<tr>
<td>MobiSys’13</td>
<td></td>
</tr>
<tr>
<td>DECAF</td>
<td>Check ad violation on each page</td>
</tr>
<tr>
<td>NSDI’14</td>
<td></td>
</tr>
<tr>
<td>SmartAds</td>
<td>Deliver ad based on page content</td>
</tr>
<tr>
<td>MobiSys’13</td>
<td></td>
</tr>
<tr>
<td>VanarSena</td>
<td>Test whether app crashes</td>
</tr>
<tr>
<td>MobiSys’14</td>
<td></td>
</tr>
<tr>
<td>ContextualFuzzing</td>
<td>Test app crash and performance issue</td>
</tr>
<tr>
<td>MSR-TR’13</td>
<td></td>
</tr>
<tr>
<td>AppsPlayground</td>
<td>Detect privacy leak and malware</td>
</tr>
<tr>
<td>CODASPY’13</td>
<td></td>
</tr>
</tbody>
</table>
Monkey Coverage vs. Speed

**Problem**

Monkey may never finish app exploration

**Solution**

Optimize app exploration in an analysis-specific way

**Example**

AMC: uses a structural similarity measure

DECAF: uses a structural similarity measure and ML-based technique to avoid similar page visit

Updates can also be pushed to device
Flexibility in Access to App Info

Problem

Analysis may require data that is not available in UI structure

Solution

Instrument app or extract info from environment

Example

SmartAds: uses app instrumentation to obtain in-app context info
VanarSena: uses app instrumentation to control memory and isolated storage
Monkey-based App Analyses

Result

Complex and customized code

E.g. DECAF: > 4300 LOC with 70% for monkey

Impact

Tool for analysis X is very hard to be reused for analysis Y.
Our Goal

Provide a **programmable** way for monkey-based app analyses.

- so that both analyses X and Y can be implemented easily by customization
Approach

Event-Driven UI Automation

UI Property
- Monkey
  - AMC

Ad Fraud
- Monkey
  - DECAF

Page Content
- Monkey
  - SmartAds
Approach

Event-Driven UI Automation

UI Property

Ad Fraud

Page Content

AMC

DECAF

SmartAds

Programmable Monkey

PUMA

events
## Framework Requirements

<table>
<thead>
<tr>
<th>System</th>
<th>Exploration Target</th>
<th>Page Transition</th>
<th>Property Checked</th>
<th>Action Taken</th>
<th>Required Instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMC</td>
<td>Distinct</td>
<td>UI</td>
<td>Accessibility</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>DECAF</td>
<td>Distinct</td>
<td>UI</td>
<td>Ad layout</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>SmartAds</td>
<td>All</td>
<td>UI</td>
<td>Page content</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>A³E</td>
<td>Distinct</td>
<td>UI</td>
<td>None</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Apps Playground</td>
<td>Distinct</td>
<td>UI, text</td>
<td>Info flow</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>VanarSena</td>
<td>Distinct</td>
<td>UI, text</td>
<td>Crash</td>
<td>Inject fault</td>
<td>Yes</td>
</tr>
<tr>
<td>Contextual Fuzzing</td>
<td>All</td>
<td>UI</td>
<td>Crash, perf.</td>
<td>Change context</td>
<td>No</td>
</tr>
</tbody>
</table>

[Specify instrumentation for the app]
Key Idea: Event-driven Paradigm

1: while not all apps have been explored do
2:   pick a new app
3:   $S \leftarrow$ empty stack
4:   push initial page to $S$
5:   while $S$ is not empty do
6:     pop an unfinished page $s_i$ from $S$
7:     go to page $s_i$
8:     pick next clickable UI element from
9:     if user input is needed (e.g., login/password) then
10:        provide user input by emulating keyboard clicks
11:        effect environmental changes.
12:        perform the click
13:        wait for next page $s_j$ to load
14:        analyze pag
15:        flag $\leftarrow s_j$ is equivalent to an explored 1
16:        if not flag then
17:           add $s_j$ to $S$
18:        update finished clicks for $s_i$
19:        if all clicks in $s_i$ are explored then
20:           remove $s_i$ from $S$
21:        flag $\leftarrow$ monkey has used too many resou
22:        if flag or $S$ is empty then
23:           terminate this app
Key Idea: Event-driven Paradigm

1. Go to initial page
2. Pick a clickable UI element
3. Effect environment changes, if needed
4. Perform click, user input
5. Wait for page done
6. Check page equivalence
7. Need continue

- App-specific Events
- Next Click
- Modify environment
- Text Input
- In-line Analysis
- State Equivalence
- Terminating App
## Framework Requirements

<table>
<thead>
<tr>
<th>System</th>
<th>Exploration Target</th>
<th>Page Transition</th>
<th>Property Checked</th>
<th>Action Taken</th>
<th>Required Instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMC</td>
<td>Distinct</td>
<td>UI</td>
<td>Accessibility</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>DECAF</td>
<td>Distinct</td>
<td>UI</td>
<td>Accessibility</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>SmartAI</td>
<td>Distinct</td>
<td>UI</td>
<td>None</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>A³E</td>
<td>Distinct</td>
<td>UI</td>
<td>None</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Apps Playground</td>
<td>Distinct</td>
<td>UI, text</td>
<td>Info flow</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>VanarSena</td>
<td>Distinct</td>
<td>UI, text</td>
<td>Crash</td>
<td>Inject fault</td>
<td>Yes</td>
</tr>
<tr>
<td>Contextual Fuzzing</td>
<td>All</td>
<td>UI</td>
<td>Crash, perf.</td>
<td>Change context</td>
<td>No</td>
</tr>
</tbody>
</table>

- **State Equivalence**: Modify environment
- **App-specific Events**: App-specific Events
- **Text Input**: Text Input
- **Next Click**: Next Click
- **In-line Analysis**: In-line Analysis
Example: Network Usage Profiler

Count bytes sent and received in HTTP traffic

```java
import monkey.types.UIService; // should be explicitly imported
import monkey.types.UIState;

class NetworkProfiler extends PUMAScript {
  boolean compareState(UIState s1, UIState s2) {
    return MouseMotionMismatchStructureMatch(s1, s2, 0.95);
  }

  int getNextClick() {
    return MouseMotionMismatch.nextClickSequential(s);
  }

  void specifyInstrumentation() {
    <described later>
  }
}
```

State Equivalence

Next Click

App-specific Events

200 KB

500 KB
Example: Network Usage Profiler

Count bytes sent and received in HTTP traffic

class NetworkProfiler extends PUMAScript {
  boolean compareState(UIState s1, UIState s2) {
    return MonkeyStateStructureMatch(s1, s2, 0.95);
  }
}

int getNextClick(UIState s) {
  return MonkeyInputFactory.nextClickSequential(s);
}

void specifyInstrumentation() {
  <described later>
State Equivalence

Determine whether two pages are equivalent or not

Structural similarity, content similarity, customized
Example: Network Usage Profiler

Count bytes sent and received in HTTP traffic

```java
class NetworkProfiler extends PUMAScript {
    boolean compareState(UIState s1, UIState s2) {
        return MonkeyInputFactory.stateStructureMatch(s1, s2, 0.95);
    }

    int getNextClick(UIState s) {
        return MonkeyInputFactory.nextClickSequential(s);
    }

    void specifyInstrumentation() {
        <described later>
    }
}
```
Next Click

Determine which UI element to click

Sequential order, max-type order, customized
Example: Network Usage Profiler

Count bytes sent and received in HTTP traffic

```java
class NetworkProfiler extends PUMAScript {
    boolean compareState(UIState s1, UIState s2) {
        return MonkeyInputFactory.stateStructureMatch(s1, s2, 0.95);
    }
    int getNextClick(UIState s) {
        return MonkeyInputFactory.nextClickSequential(s);
    }
    void specifyInstrumentation() {
        ...
    }
}
```
App-Specific Events

Instrumentation through SIF

class Logger {
    void countRequest (HttpUriRequest req) {
        Log(req.getRequestLine().getUri().getLength());
    }
    void countResponse (HttpResponse resp) {
        Log(resp.getEntity().get CONTENTLENGTH());
    }
}

Putting It All Together

- PUMAScript
- Original App
- Instrumenter
- Monkey
- Instrumented App
Implementation Challenges

- **Obtaining UI structure**
  - Intercept `uiautomator` events

- **Supporting page scrolling**
  - Check for scrolling direction
  - Scroll in a zig-zag pattern
  - Combine partial pages

- **Detecting page loading completion**
  - `WINDOW_CONTENT_CHANGED` event
  - Use window-based moving average heuristic
Can PUMA support rapid development of large-scale dynamic app analyses?

What insights into the app ecosystem can these analyses provide?
Evaluation

Methodology

✓ Implemented 7 analysis studies
✓ Downloaded 3,600 apps from Google Play
✓ Ran over emulators phones
<table>
<thead>
<tr>
<th>Implemented Analysis Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accessibility</strong></td>
</tr>
<tr>
<td><strong>Violation Detection</strong></td>
</tr>
<tr>
<td>Check accessibility of UI elements</td>
</tr>
<tr>
<td><strong>Content-based App Search</strong></td>
</tr>
<tr>
<td>Crawl in-app data for search engine</td>
</tr>
<tr>
<td><strong>UI Structure Classifier</strong></td>
</tr>
<tr>
<td>Classify apps based on UI structure</td>
</tr>
<tr>
<td><strong>Ad Fraud Detection</strong></td>
</tr>
<tr>
<td>Detect ads placement violation</td>
</tr>
<tr>
<td><strong>Network Usage Profiler</strong></td>
</tr>
<tr>
<td>Profile in-app HTTP traffic use</td>
</tr>
<tr>
<td><strong>Permission Usage Profiler</strong></td>
</tr>
<tr>
<td>Profile in-app permission usage</td>
</tr>
<tr>
<td><strong>Stress Testing</strong></td>
</tr>
<tr>
<td>Inject null HTTP response for test</td>
</tr>
</tbody>
</table>
RQ1: Rapid Development?

Accessibility
Violation Detection

Content-based
App Search

UI Structure
Classifier

Ad Fraud
Detection

Network Usage
Profiler

Permission Usage
Profiler

Stress Testing

Finished development and experiments within 2 weeks
RQ1: Rapid Development?

<table>
<thead>
<tr>
<th>Study Task</th>
<th>PUMAScript (LOC)</th>
<th>User Code (LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility violation detection</td>
<td>11</td>
<td>60</td>
</tr>
<tr>
<td>Content-based app search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UI structure classifier</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Ad fraud detection</td>
<td>11</td>
<td>52</td>
</tr>
<tr>
<td>Network usage profiler</td>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>Permission usage profiler</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>Stress testing</td>
<td>16</td>
<td>5</td>
</tr>
</tbody>
</table>

PUMA only uses < 100 LOC for all studies
Cross-Analysis Scaling Optimization

With PUMA

UI Property
Monkey API

Ad Fraud
Monkey API

Page Content
Monkey API

Computing Resources
Cross-Analysis Scaling Optimization

With PUMA

For 100 apps and 1 phone, it took 8 hours instead of 20 hours to finish.

→ 2.5x speed up
<table>
<thead>
<tr>
<th>RQ2: Insights into App Ecosystem?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accessibility</strong></td>
</tr>
<tr>
<td>Violation Detection</td>
</tr>
<tr>
<td>Check accessibility of UI elements</td>
</tr>
<tr>
<td><strong>Content-based App Search</strong></td>
</tr>
<tr>
<td>Crawl in-app data for search engine</td>
</tr>
<tr>
<td><strong>UI Structure Classifier</strong></td>
</tr>
<tr>
<td>Classify apps based on UI structure</td>
</tr>
<tr>
<td><strong>Ad Fraud Detection</strong></td>
</tr>
<tr>
<td>Detect ads placement violation</td>
</tr>
<tr>
<td><strong>Network Usage Profiler</strong></td>
</tr>
<tr>
<td>Profile in-app HTTP traffic use</td>
</tr>
<tr>
<td><strong>Permission Usage Profiler</strong></td>
</tr>
<tr>
<td>Profile in-app permission usage</td>
</tr>
<tr>
<td><strong>Stress Testing</strong></td>
</tr>
<tr>
<td>Inject null HTTP response for test</td>
</tr>
</tbody>
</table>
RQ2: Insights into App Ecosystem?

Accessibility Violation Detection

RQ2: Insights into App Ecosystem?

Accessibility Violation Detection

scrolling
button distance
button size
word count
user action per task

~55% of our apps violate ≥1 accessibility rules
RQ2: Insights into App Ecosystem?

Content-based App Search

Google Play Store

Apps
- My apps
- Shop

Games
- Editors’ Choice

Best Deals

Description

This app brings you the best daily deals available from the Internet. Features:
* Daily deals from popular deal sites
* Customize the deal list
* Local deals
### RQ2: Insights into App Ecosystem?

#### Content-based App Search

**Search Term:** justin bieber

<table>
<thead>
<tr>
<th>Name</th>
<th>Name + Desc.</th>
<th>Name + Desc. + In-app Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**App Results:**

<table>
<thead>
<tr>
<th>App</th>
<th>Name + Desc. + In-app Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MixerBox</td>
</tr>
<tr>
<td>2</td>
<td>Shane Dawson Episodes</td>
</tr>
<tr>
<td>3</td>
<td>Hits Music Ringback Tone</td>
</tr>
<tr>
<td>4</td>
<td>Mobo Movie News &amp; Stars</td>
</tr>
<tr>
<td>5</td>
<td>Mobo Fashion Trends &amp; Deals</td>
</tr>
<tr>
<td>6</td>
<td>How to Draw Singers</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>19</td>
<td>Rangers</td>
</tr>
</tbody>
</table>
### RQ2: Insights into App Ecosystem?

**Content-based App Search**

<table>
<thead>
<tr>
<th>Search Index</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>8.31</td>
<td>3</td>
</tr>
<tr>
<td>Name + Desc.</td>
<td>199.43</td>
<td>66</td>
</tr>
<tr>
<td>Name + Desc. + In-app Data</td>
<td>300.37</td>
<td>131</td>
</tr>
</tbody>
</table>

In-app data can greatly improve search relevance.
Programmable UI-Automation Framework

- Separates exploration and analysis logic
- Incorporates a generic monkey
- Exposes event driven programming abstractions

https://github.com/USC-NSL/sif
https://github.com/USC-NSL/puma