Improving Android Reliability and Security

Iulian Neamtiu, Assoc. Prof.

CS Chairs Meeting
June 14, 2018
Mobile OSes rapidly expanding their device range and user base.…

… but users/developers/vendors have little insight/assurance of device behavior/reliability/security

Core SW stack (AOSP)

Preinstalled software

Third-party apps

Android: 2 billion monthly active users
The Nature of Mobile Bugs

Study: 22,000 confirmed & fixed mobile bugs (Android, iOS) over 7 years
Focus: High-severity bugs (crashes, data loss/corruption)

<table>
<thead>
<tr>
<th>Android</th>
<th>iOS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concurrency:</strong></td>
<td><strong>Crash (non-concurrency):</strong> 52%</td>
</tr>
<tr>
<td>66%</td>
<td></td>
</tr>
<tr>
<td><strong>GUI:</strong></td>
<td><strong>App. logic:</strong> 32%</td>
</tr>
<tr>
<td>23%</td>
<td></td>
</tr>
<tr>
<td><strong>Security:</strong></td>
<td><strong>Build:</strong> 12%</td>
</tr>
<tr>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>

How to find/reproduce/fix these?
Concurrency Errors in Android
Harmful race in the Android OS!

mr...@gmail.com  <mr...@gmail.com> #23  Jan 11, 2015 08:59AM

I have around 700 crashes DAILY in production! This issue is not fixed and should be reopened. Come on guys! This is a really nasty bug that has been around for 3 months already!

va...@gmail.com  <va...@gmail.com> #25  Jan 11, 2015 01:05PM

Unfortunately no, I cannot provide the code because it's really hard to reproduce this, but it is still happening in com.android.support:recyclerview-v7:21.0.3 with stacktrace

ya...@gmail.com  <ya...@gmail.com> #70  Aug 27, 2015 01:13AM

Hi,

This is very random, cannot reproduce every time.
Device: Nexus 6

https://code.google.com/p/android/issues/detail?id=77846
Our Approach

- Prior approaches: all dynamic
  - Low coverage, high rate of false negatives & false positives

- Our approach
  - **Action** = context-sensitive event handling
    - Novel abstraction, reifies Callbacks, Threads, AsyncTasks, MsgHandler
  - **Happens-before** (HB): A1 ⊂ A2
    - A1 is completed before A2
  - **Bootstrapping** via program synthesis ➔ precise static analysis
  - **Backward symbolic execution** to refute (most) false positives

⇒ Effective and efficient
Program Synthesis to “Boot-strap” Precise Static Analysis

• Create synthetic “main” activity
• Create synthetic method call sites
  1. Add lifecycle/GUI/system actions
  2. Build call graph for found actions
  3. Go back to step 1, iterate until fixpoint

Prior Android static analyses: imprecise

SyntheticMain.java

* onCreate onItemClick
  thread
  onStart onLongPress
  onResume
  onClick
  AsyncTask
  onLow Memory
  onPost Execution
  msg1. handleMessage

Runnable1
Runnable2

onCreate onStart onResume
AsyncTask onItemClick onLow Memory onLow Memory
onCreateOptionsMenu onLow Memory
msg1. handleMessage
Symbolic Execution-based Refutation

```java
void stop() { // action B
    if (mIsRunning) {
        mIsRunning = false; // α_B
        mAccumTime = ... // α_B
    }
}
Timer.Runnable runner = {
    void run() { // action A
        if (mIsRunning) {
            mAccumTime = ... // α_A
            if (*) {
                ...
                postDelayed(runner,...);
            } else
                mIsRunning = false; }
    }
else
    mIsRunning = false; }
}
```

Race Refuted!

Can $\alpha_B$ occur before $\alpha_A$? No!
## Evaluation

### Effectiveness: races found

<table>
<thead>
<tr>
<th>App</th>
<th>Installs (millions)</th>
<th>Candidate racy pairs</th>
<th>After our analysis</th>
<th>True races</th>
<th>False positives</th>
<th>Event Racer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barcode Scanner</td>
<td>&gt; 100</td>
<td>64</td>
<td>15</td>
<td>11</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>VLC</td>
<td>&gt; 100</td>
<td>202</td>
<td>35</td>
<td>32</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>FB Reader</td>
<td>&gt; 10</td>
<td>836</td>
<td>106</td>
<td>93</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>K-9</td>
<td>&gt; 5</td>
<td>1,347</td>
<td>89</td>
<td>72</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>NPR</td>
<td>&gt; 1</td>
<td>607</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Across 20 apps</td>
<td></td>
<td>431</td>
<td>33</td>
<td>29</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Dataset: 194 open-source apps; 20 analyzed manually
Analysis time: about 30 minutes/apps

Best prior work (dynamic) 13x

---

### Additional apps and distributions

<table>
<thead>
<tr>
<th>App</th>
<th>Installs (millions)</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skype</td>
<td>&gt; 1,000</td>
<td>178</td>
</tr>
<tr>
<td>Snapchat</td>
<td>&gt; 500</td>
<td>464</td>
</tr>
<tr>
<td>Netflix</td>
<td>&gt; 100</td>
<td>793</td>
</tr>
<tr>
<td>Acrobat Reader</td>
<td>&gt; 100</td>
<td>3,134</td>
</tr>
<tr>
<td>PayPal</td>
<td>&gt; 50</td>
<td>51</td>
</tr>
<tr>
<td>Walmart</td>
<td>&gt; 10</td>
<td>48</td>
</tr>
</tbody>
</table>

(paper in preparation)
GUI bugs
GUI Bugs: Restart Errors

Earlier version of **Alarm Clock Plus** app, more than 5 million installs

- Change phone orientation, alarm is reset!
- Or send app to background, then bring to foreground, alarm is reset!

**Results:** *49 confirmed bugs in 37 apps*, including in well-known apps:
  - Dr. Web Anti-virus Light, Symantec Norton Snap, Motorola Camera,
  - Alarm Clock Plus, OI File Manager

Restart is good: cleans the state
Restart is bad: cleans the state

Our approach exposes data kept/lost upon restart
Security
A malicious app has deleted a user's messages without their knowledge or consent.

SHB = Behavior meant to hide an app's presence or actions → Fundamentally Deceptive
Our approach detects and characterizes this behavior.

App stores do not disclose such behavior.

‘Self Hiding’ Behavior (SHB)
SHB#6: Delete Message

Malware **DroidKungFu1** deletes SMS messages

```
$r4 = virtualinvoke $r3.<java.lang.StringBuilder: java.lang.String toString()>()
$r5 = staticinvoke <android.net.Uri: android.net.Uri parse(java.lang.String)>($r4)
virtualinvoke $r2.<android.content.ContentResolver: int delete(android.net.Uri.java.lang.String.java.lang.String[])>($r5, null, null)
```

Essentially, the SMS messages self destruct!

NOT user-initiated!
Results: “Good” Apps Behave Badly

Dataset: 3,219 malicious apps; 6,233 benign apps

Our static analysis can separate benign from malicious with 87.19% F-measure
Malware employs self-hiding (1.5 SH/app; unsurprising)
Some good apps employ self-hiding (0.2 SH/app)

HIDE RUNNING APP

ACCESS AND INITIALIZES location without user’s knowledge

BLOCK CALL

Not a False Positive!

MUTE PHONE

Manipulates the ringer mode

HIDE NOTIFICATION / BLOCK MESSAGE

Blocks notifications without user consent

LURK/HOVER for a File Explorer?

Interposes between user and app