AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction

Jianjun Huang  
Department of Computer Science  
Purdue University, USA  
huang427@cs.purdue.edu

Xiangyu Zhang  
Department of Computer Science  
Purdue University, USA  
xyzhang@cs.purdue.edu

Lin Tan  
Electrical and Computer Engineering  
University of Waterloo, Canada  
lintan@uwaterloo.ca

Peng Wang  
School of Information  
Renmin University of China, China  
pengwang@ruc.edu.cn

Bin Liang  
School of Information  
Renmin University of China, China  
liangb@ruc.edu.cn
Outline

• Overview
• Intent Propagation
• UI Compatibility Check
• Evaluation
Idea

Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction

Insight!
Stealthy Behavior Candidate

• Send SMS
  • `sendTextMessage()`, `sendDataMessage()`, `sendMultipartTextMessage()`

• Phone call
  • `startActivity()` with action `android.intent.action.CALL`

• Http Access
  • `URL.openConnection()`, `URL.openStream()`, `AbstractHttpClient.execute()` …

• Install
  • `Runtime.exec()` with “pm install”, `ProcessBuilder.start()` using “pm” and “install”
Detection

```java
// In class Qiyu.StartPageActivity
01:  public void onClick(View v){
02:      if(/*test environment*/){
03:         Woa.F f = new Woa.F(v, this);
04:         f.execute(new String[0]); // trigger line 9
05:     } else {
06:         Woa.AG.B(); // invoke line 17
07:     }
08: }
09:  }
10:  }
11:  }
12: // In class Woa.F
13: private Object doInBackground(Object[] obbaj){
14:    // transitively calls Woa.BA.A() at line 14
15: 
16:  }
17:  }
18: // In class Woa.BA
21: public void A(){
22:    this.h.execute(this.d); // HttpClient.execute(...)
23: }
24:  }
25: // In class Woa.AG
26: public static void B(){
27:    Woa.U u = new Woa.U();
28:    u.execute(...); // transitively calls C() at line 21
29:  }
30:  }
31: // In class Woa.AK
32: public static boolean C(Context c, String s1, String s2){
33:    SmsManager sm = SmsManager.getDefault();
34:    sm.sendTextMessage(s1, null, s2, null, null);
35:  }
```
Special Cases

• SMS notify
• UI operation
Intent Propagation

• How intents are propagated to top level functions.
• How to detect correlation between intents.
Atoms

apiIntent\( (L,T) \): API call at program point \( L \) has intent type \( T \).
def\( (L,X) \): variable \( X \) is defined at program point \( L \).
use\( (L,X) \): variable \( X \) is used at program point \( L \).
actual\( (L,M,X) \): variable \( X \) is the \( M^{th} \) actual argument at call site \( L \).
formal\( (F,M,X) \): variable \( X \) is the \( M^{th} \) formal argument of function \( F() \).
inFunction\( (F,L) \): program point \( L \) is in function \( F() \).
funEntry\( (F,L) \): program point \( L \) is the entry of function \( F() \).
hasDefFreePath\( (L_1,L_2,X) \): there is a path from \( L_1 \) to \( L_2 \) along which \( X \) may not be defined.
componentEntry\( (X,F) \): \( F() \) is the entry of Android component \( X \), e.g. \textit{onCreate()} of an Activity or a Service component.
immediateCD\( (L_1,L_2) \): program point \( L_2 \) is immediately control dependent on \( L_1 \) in the same function.
directInvoke\( (F_1,F_2,L) \): \( F_1 \) invokes \( F_2 \) at program point \( L \).
indirectInvoke\( (F_1,F_2) \): \( F_2 \) is the actual destination of \( F_1() \) in event-driven circumstances, e.g. (1) \textit{Thread.start()} \( \rightarrow \) \textit{Runnable.run()}; (2) \textit{Handler.sendMessage()} \( \rightarrow \) \textit{Handler.handleMessage()}.
iccInvoke\( (F_1,F_2,L) \): \( F_1 \) invokes a function \( F_2 \) for inter-component communication purpose at \( L \). \( F_2 \) should be APIs like \textit{startActivity()} \( \backslash \) \textit{startService()}.
Rules

/\* invoke(F_1, F_2, L) \*: F_1 invokes F_2 at program point L. */
invoke(F_1, F_2, L) :- directInvoke(F_1, F_2, L)
invoke(F_1, F_2, L) :- indInvoke(F_1, F_3, L) & actual(L_1, X) & "L_1: X.setClass(...)" & actual(L_1, 2, Y) & componentEntry(Y, F_2)
invoke(F_1, F_2, L) :- invoke(F_1, F_3, L) & indirectInvoke(F_3, F_2)
invoke(F_1, F_2, L) :- invoke(F_1, F_3, L) & invoke(F_3, F_2, L)

/\* hasIntent(F, T, L) \*: F() has intent type T and the corresponding API call is at L. */
hasIntent(F, T, L) :- invoke(F, A, L) & apiIntent(L, T)
hasIntent(F, T, L_1) :- hasIntent(F, T, L_1) & invoke(F, F_1, L_2)

/\* controlDep(L_1, L_2) \*: program point L_2 is control dependent on L_1. */
controlDep(L_1, L_2) :- immediateCD(L_1, L_2)
controlDep(L_1, L_2) :- inFunction(F_1, L_1) & inFunction(F_2, L_2) & invoke(F_1, F_2, L_3) & controlDep(L_1, L_3)

/\* defUse(L_1, L_2), useUse(L_1, L_2) \*: data at L_1 and L_2 are data correlated. */
defUse(L_1, L_2) :- def(L_1, X) & use(L_2, X) & hasDefFreePath(L_1, L_2, X)
defUse(L_1, L_2) :- invoke(F_1, F_2, L_1) & actual(L_1, M, X) & formal(F_2, M, Y) & funEntry(F_2, L_3) & hasDefFreePath(L_3, L_2, Y) & use(L_2, Y)
useUse(L_1, L_2) :- defUse(L_3, L_1) & defUse(L_3, L_2)
useUse(L_2, L_1) :- defUse(L_3, L_1) & defUse(L_3, L_2)

/\* correlated(L_1, L_2) \*: L_1 and L_2 are data/control correlated. */
correlated(L_1, L_2) :- controlDep(L_1, L_2)
correlated(L_1, L_2) :- defUse(L_1, L_2)
correlated(L_1, L_2) :- useUse(L_1, L_2)
correlated(L_1, L_2) :- correlated(L_1, L_3) & correlated(L_2, L_2)

/\* correlatedIntent(F, T_1, L_1, T_2, L_2) \*: In function F, intent T_1 at L_1 is correlated to T_2 at L_2. */
correlatedIntent(F, T_1, L_1, T_2, L_2) :- hasIntent(F, T_1, L_1) & hasIntent(F, T_2, L_2) & correlated(L_1, L_2)
Corner Case

\[
\text{invoke}(F_1,F_2,L) \\
\text{:- } \text{iccInvoke}(F_1,F_3,L) \land \text{actual}(L,1,X) \land "L_1: X.setClass(...)" \land \text{actual}(L_1,2,Y) \land \text{componentEntry}(Y,F_2)
\]

\[
\text{hasIntent}(F,T,L) \quad \text{:- } \text{invoke}(F,A,L) \land \text{apiIntent}(L,T)
\]
UI Compatibility Check

• Acquiring User Interface Text
• Text Analysis
• Compatibility Check
Acquiring User Interface Text

- Statically through an XML resource file
  - Parent
  - Activity lifecycle methods, eg. onCreat(), onStart()
- Dynamically by constructing the view tree at runtime (not implemented)
Text Analysis

• Machine Learning

Algorithm 1 Generating Keyword Cover Set.

\[
\text{train}(S, F) \\
KWD = \phi \text{ /*the keyword cover set*/} \\
\text{while } F \neq \phi \text{ do} \\
\quad \text{sort } S \text{ by keyword (or keyword pair) frequency} \\
\quad k = \text{the top ranked keyword (or pair) in } S \\
\quad X = \text{the functions in which } k \text{ occurs} \\
\quad KWD = KWD \cup k \\
\quad F = F - X \\
\quad S = S - \{\text{all the keywords (pairs) in } X\} \\
\text{end while}
\]

HttpAccess: Load, Register...
PhoneCall: Call
Compatibility Check

• if S is incompatible with T and all the intents correlated with T, it is considered a mismatch. Note that we consider empty text is incompatible with any intent.

• If T is a SendSms intent and has a correlated SmsNotify intent. It is not a mismatch regardless of the UI text.

• If T is HttpAccess, the technique checks if the corresponding UI text is compatible. If not, it further checks if T is correlated to any UiOperation intent. If not, the intent is considered stealthy.
Evaluation

We transform the DEX file of an app to a JAR file with dex2jar [31] and then use WALA [22] as the analysis engine. Our implementation is mainly on top of WALA.
App Source

- Contagio Mini Dump
- Google Play
- Wandoujia
### Result

<table>
<thead>
<tr>
<th></th>
<th>HTTP</th>
<th>SMS</th>
<th>CALL</th>
<th>INSTALL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#App</td>
<td>#Intent</td>
<td>#Rep</td>
<td>#FP/#FN</td>
</tr>
<tr>
<td>Contagio</td>
<td>96</td>
<td>189(69)</td>
<td>136(64)</td>
<td>28/7(14/2)</td>
</tr>
<tr>
<td>Google Play</td>
<td>12</td>
<td>19(9)</td>
<td>12(7)</td>
<td>3/0(2/0)</td>
</tr>
<tr>
<td>Wandoujia</td>
<td>74</td>
<td>166(39)</td>
<td>70(23)</td>
<td>23/5(10/1)</td>
</tr>
<tr>
<td>Total</td>
<td>182</td>
<td>374(117)</td>
<td>218(94)</td>
<td>54/12(26/3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>#Intent</th>
<th>#Rep</th>
<th>#FP/#FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contagio</td>
<td>287(82)</td>
<td>228(77)</td>
<td>28/14(14/8)</td>
</tr>
<tr>
<td>Google Play</td>
<td>27(10)</td>
<td>18(8)</td>
<td>5/0(3/0)</td>
</tr>
<tr>
<td>Wandoujia</td>
<td>220(47)</td>
<td>83(28)</td>
<td>26/7(11/3)</td>
</tr>
<tr>
<td>Total</td>
<td>534(139)</td>
<td>329(113)</td>
<td>59/21(28/11)</td>
</tr>
</tbody>
</table>
Observation 1 ------ Overview

- Stealthy SMS sends & Phone calls -> unexpected charges.
- Stealthy HTTP accesses -> notify the remote servers the status of device or the app.
- Stealthy HTTP accesses are very common, less harmful.
- SMS sends are another dominant category of stealthy behaviors.
Observation 2 ------ False Positive

- Reason of False Positive
  - AsDroid cannot analyze dynamically generated text associated with a UI component
  - The dictionary we use is incomplete
  - Some reported intents are along infeasible paths but AsDroid does not reason about path feasibility

- Http Accesses
  - Download advertisement materials and store them to external les that are later read and displayed.
Observation 3 ------ False Negative

• Install, Reason
  • Current implementation cannot model some of the implicit call edges.
  • native libraries are used to perform stealthy behavior, which is not handled by AsDroid.

• HttpAccess, Reason
  • Accesses doesn't match the textual semantics even with “download” and “login”
Comparison with FlowDroid

• Flow Droid
  • Run 96 apps, 55 ran out of memory
  • In remaining 41 apps, reports 4 SMS sends in 3 apps and 1 HTTP access in 1 app

• AsDroid
  • Reports 26 stealthy HTTP access in 18 apps including 1 reported by FlowDroid
  • Reports 35 SMS sends in 21 apps, including 2 SMS sends by FlowDroid

2 SMS sends are false positive in FlowDroid
Limitation

• UI analysis is simply based on textual keywords, which may be insufficient. It is possible that apps use images or obfuscated texts.

• May not be insufficient if adversary has the prior knowledge of AsDroid.

• Cannot reason about correlations through external resources, leading to false positives.

• Does not support native code or reflection.

• Misses some Inter-Component Communication correlations.
Conclusion

Key Idea
• Identify contradiction between program behavior and user interface text.

Test AsDroid on 182 apps, reports 113 apps that have stealthy behaviors, with 28 false positives and 11 false negatives.