FlowDroid

Alex Mariakakis

From CSE 501…again

Motivation

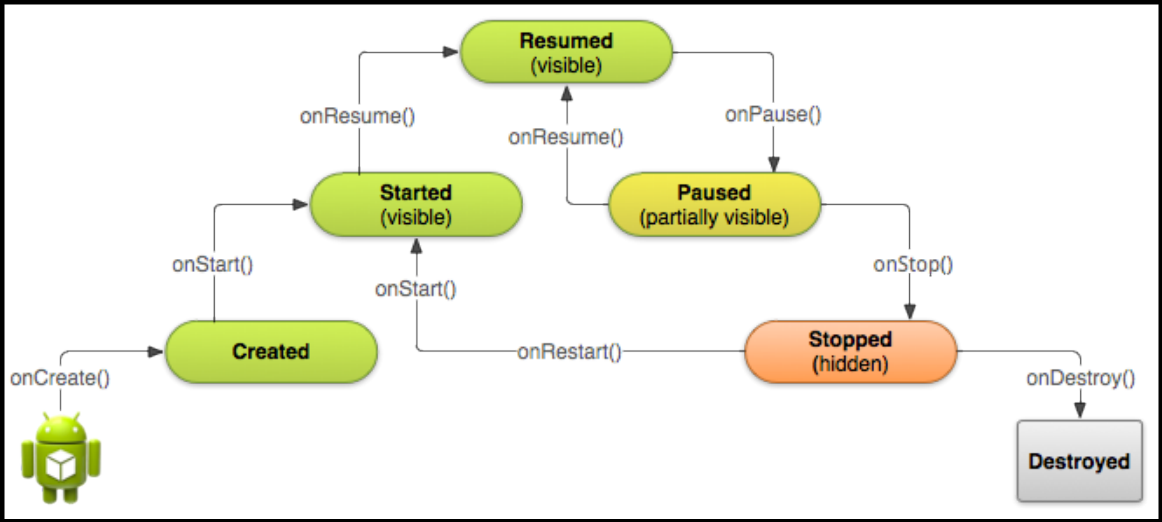
- All sorts of mobile malware exist
  - Selling user information to advertisement/ marketing companies
  - Stealing user credentials
  - Premium rate calls and SMS
  - SMS spam
  - Search engine optimization
  - Ransom

Contributions

- **FlowDroid:** the first fully context, field, object and flow-sensitive taint analysis which considers the Android application lifecycle and UI widgets, and which features a novel, particularly precise variant of an on-demand alias analysis
- **DroidBench:** a novel, open and comprehensive micro benchmark suite for Android flow analyses
- **Experiments:** demonstrate superior precision and recall to commercial tools and manageable runtimes on real-world apps

Challenges

1. Multiple entry points
2. Asynchronously executing components
3. Callbacks



Challenges

1. Multiple entry points

2. Asynchronously executing components

3. Callbacks

Challenges

1. Multiple entry points

2. Asynchronously executing components

3. Callbacks

SensorReaderActivity.java × SensorService.java × activity_main.xml ×

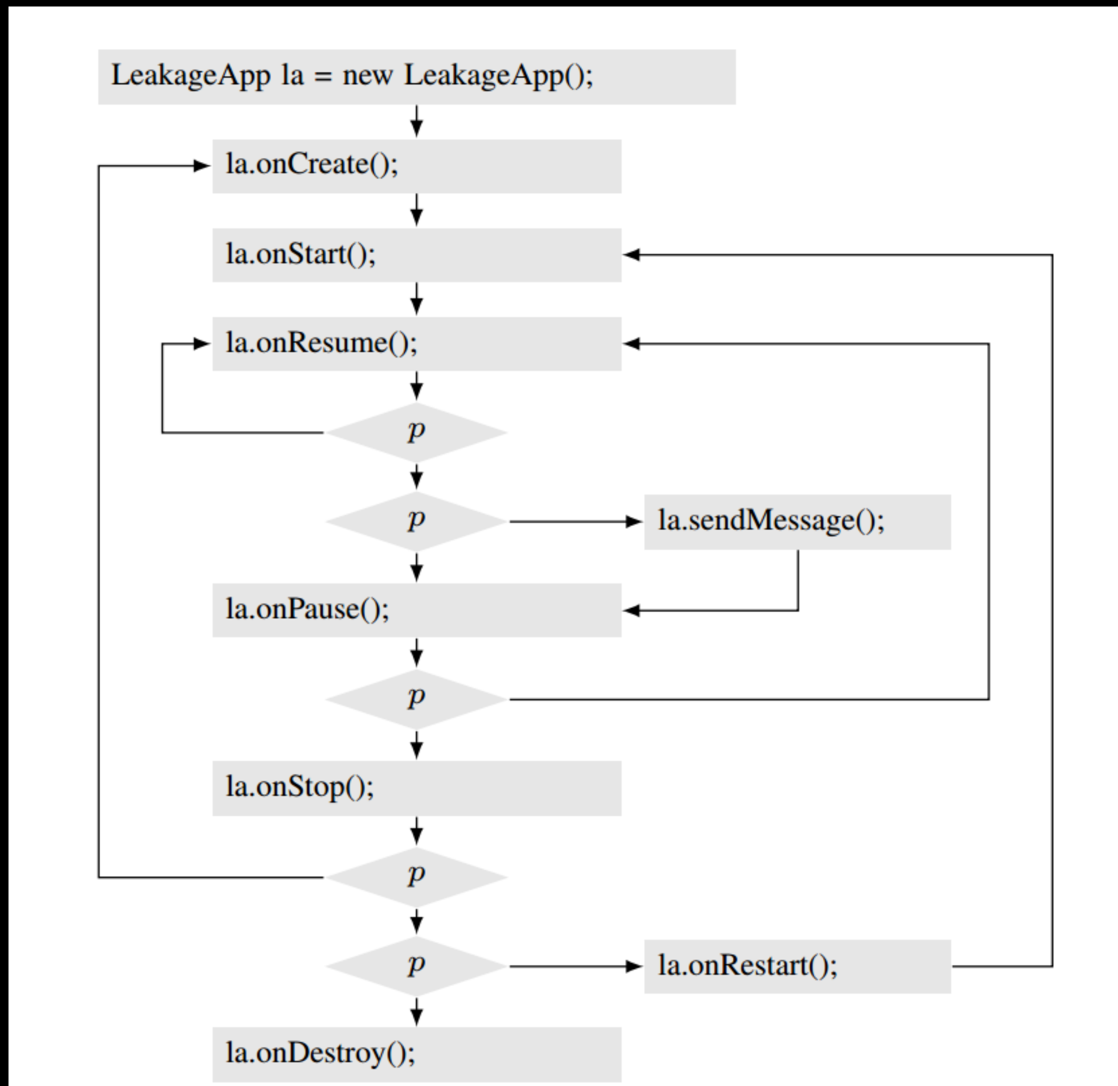
```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/startButton"
        android:layout_width="wrap_content"
        android:layout_height="150dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="Start" />

    <Button
        android:id="@+id/stopButton"
        android:layout_width="wrap_content"
        android:layout_height="150dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:onClick="onClick"
        android:layout_below="@+id/startButton"
        android:text="Stop" />

    <ProgressBar
        android:id="@+id/runProgressBar"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:indeterminate="true"
        android:visibility="gone" />

</RelativeLayout>
```

```java
public class LeakageApp extends Activity {
   private User user = null;
   protected void onRestart() {
      EditText usernameText = (EditText) findViewById(R.id.username);
      EditText passwordText = (EditText) findViewById(R.id.pwdString);
      String uname = usernameText . toString ();
      String pwd = passwordText . toString();
      if (!uname.isEmpty() && !pwd.isEmpty())
         this.user = new User(uname, pwd);
   }
   // Callback method in xml file
   public void sendMessage(View view) {
      if (user == null) return;
      Password pwd = user.getpwd();
      String pwdString = pwd.getPassword();
      String obfPwd = "";
      // must track primitives
      for (char c: pwdString.toCharArray())
         obfPwd += c + "_"; // String concat
      String message = " User : " + user.getName() + " | Pwd: " + obfPwd;
      SmsManager sms = SmsManager.getDefault();
      sms.sendTextMessage(" +44 020 7321 0905 ", null, message, null, null);
   }
}
```

#1

#2 and 3

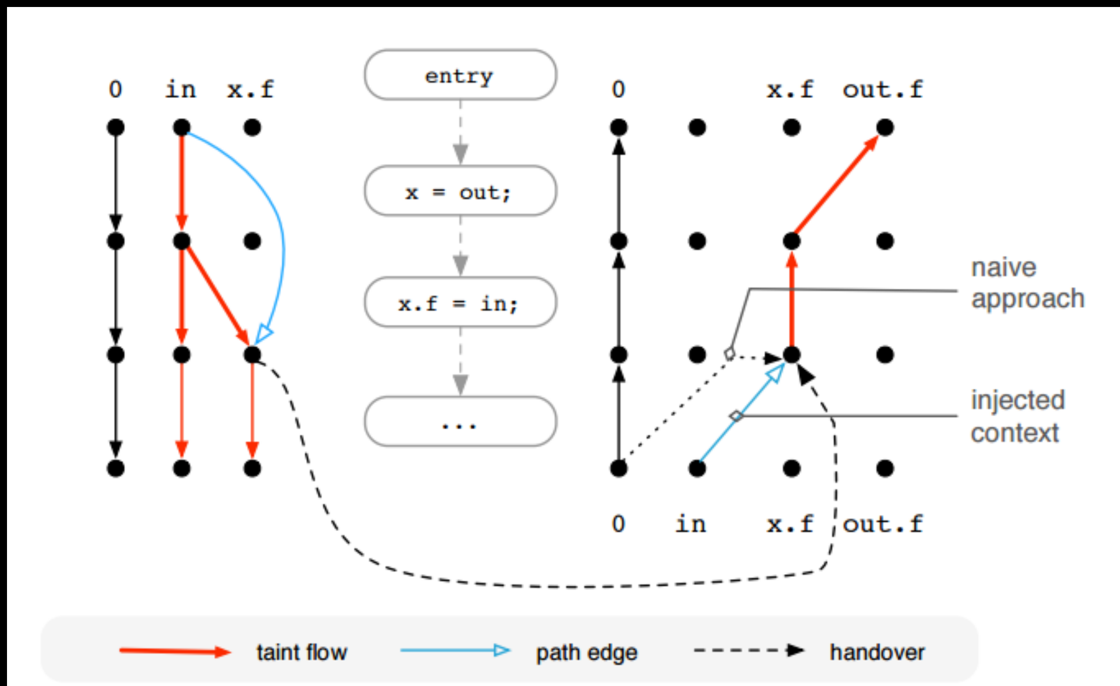
Dummy Main Method



On-Demand Alias Analysis

```
void main() {
    Data p = new …; Data p2 = new …;
    taintIt(source(), p);
    sink(p.f);
}

void taintIt(String in, Data out) {
    x = out;         // x = p  → p.f = source()
    x.f = in;        // x.f = source()
    sink(out.f);     // sink(p.f) → sink(source())
}
```

Context Sensitivity



Visualization from IFDS

- Inject context of forward analysis into backward analysis since not all inputs will lead to taints

  Ex: `taintIt(source(), p1)` vs. `taintIt("public", p2)`
- Whenever an alias is found, work forward from the beginning (rather than backwards) to map taints and avoid unrealizable paths

Flow Sensitivity

```
Data p = new …; Data p2 = p;
sink(p2.f);
p.f = source();
sink(p2.f);
```

not tainted

activation statement

tainted

p's taint not yet activated

Concept from Andromeda

Evaluation

RQ1: How does FlowDroid compare to commercial taint-analysis tools for Android in terms of precision and recall?

precision = 86%

recall = 93%

$\odot$ = correct warning, $\star$ = false warning, $\bigcirc$ = missed leak
multiple circles in one row: multiple leaks expected
all-empty row: no leaks expected, none reported

App Name	AppScan	Fortify	FlowDroid
Arrays and Lists			
ArrayAccess1			$\star$
ArrayAccess2	$\star$	$\star$	$\star$
ListAccess1	$\star$	$\star$	$\star$
Callbacks			
AnonymousClass1	$\bigcirc$	$\odot$	$\odot$
Button1	$\bigcirc$	$\odot$	$\odot$
Button2	$\odot$ $\bigcirc$ $\bigcirc$	$\odot$ $\bigcirc$ $\bigcirc$	$\odot$ $\odot$ $\odot$ $\star$
LocationLeak1	$\bigcirc$ $\bigcirc$	$\bigcirc$ $\bigcirc$	$\odot$ $\odot$
LocationLeak2	$\bigcirc$ $\bigcirc$	$\bigcirc$ $\bigcirc$	$\odot$ $\odot$
MethodOverride1	$\odot$	$\odot$	$\odot$
Field and Object Sensitivity			
FieldSensitivity1			
FieldSensitivity2			
FieldSensitivity3	$\odot$	$\odot$	$\odot$
FieldSensitivity4	$\star$		
InheritedObjects1	$\odot$	$\odot$	$\odot$
ObjectSensitivity1			
ObjectSensitivity2	$\star$		
Inter-App Communication			
IntentSink1	$\odot$	$\odot$	$\bigcirc$
IntentSink2	$\odot$	$\odot$	$\odot$
ActivityCommunication1	$\odot$	$\odot$	$\odot$
Lifecycle			
BroadcastReceiverLifecycle1	$\odot$	$\odot$	$\odot$
ActivityLifecycle1	$\odot$	$\odot$	$\odot$
ActivityLifecycle2	$\bigcirc$	$\odot$	$\odot$
ActivityLifecycle3	$\bigcirc$	$\bigcirc$	$\odot$
ActivityLifecycle4	$\bigcirc$	$\odot$	$\odot$
ServiceLifecycle1	$\bigcirc$	$\bigcirc$	$\odot$
General Java			
Loop1	$\odot$	$\bigcirc$	$\odot$
Loop2	$\odot$	$\bigcirc$	$\odot$
SourceCodeSpecific1	$\odot$	$\odot$	$\odot$
StaticInitialization1	$\bigcirc$	$\odot$	$\bigcirc$
UnreachableCode		$\star$	
Miscellaneous Android-Specific			
PrivateDataLeak1	$\bigcirc$	$\bigcirc$	$\odot$
PrivateDataLeak2	$\odot$	$\odot$	$\odot$
DirectLeak1	$\odot$	$\odot$	$\odot$
InactiveActivity	$\star$	$\star$	
LogNoLeak			
Sum, Precision and Recall			
$\odot$, higher is better	14	17	26
$\star$, lower is better	5	4	4
$\bigcirc$, lower is better	14	11	2
Precision $p = \odot/(\odot + \star)$	74%	81%	$\boxed{86\%}$
Recall $r = \odot/(\odot + \bigcirc)$	50%	61%	$\boxed{93\%}$
F-measure $2pr/(p+r)$	0.60	0.70	0.89

Table 1: DROIDBENCH test results

Evaluation

RQ2: Can FlowDroid find all privacy leaks in InsecureBank, an app specifically designed by others to challenge vulnerability detection tools for Android, and what is its performance?

Finds all seven data leaks in 31 seconds

Evaluation

RQ3: Can FlowDroid find leaks in real-world applications and how fast is it?

App Source	Run Time	Notes
Google Play	Mean < 1 min Max ≈ 4.5 min	Found lots of leaks, claims that most are not malicious
VirusShare Project	Mean = 16 s Min = 5 s Max = 71 s	Samples were smaller than Google Play apps

Evaluation

RQ4: How well does FlowDroid perform when being applied to taint-analysis problems related to Java, not Android, both in terms of precision and recall?

precision = 93%

recall = 97%

Test-case group	TP	FP
Aliasing	11/11	0
Arrays	9/9	6
Basic	58/60	0
Collections	14/14	3
Datastructure	5/5	0
Factory	3/3	0
Inter	14/16	0
Pred	n/a	n/a
Reflection	n/a	n/a
Sanitizer	n/a	n/a
Session	3/3	0
StrongUpdates	0/0	0
Sum	117/121	9

Table 2: SecuriBench Micro test results

Limitations from Implementation

- Rule-based taint propagation for external libraries
  - *E.g.*, adding a tainted element to a set taints the whole set
- Native C calls treated as black box
  - If not predefined rule, assume tainted input leads to tainted output
- Assumes arbitrary, but sequential ordering, so can't handle multi-threading

Interesting Questions

- Why so much focus on Android? Does it generalize?
- Which do you value more: precision or recall?