Information-Flow Analysis of Android Applications in DroidSafe

MICHAEL I. GORDON, DEOKHWAN KIM, JEFF PERKINS , LIMEI GILHAM , NGUYEN NGUYEN , AND MARTIN RINARD

NDSS 2014

PRESENTED BY KE TIAN

Outlines

- Overview
- Motivation
- Approach/methodology
- Experiment
- Discussion

Overview

- Problem:

  Critical source-sink flow detection

- Solution

  Static information flow analysis +accurate analysis

  stubs (a static implementation to mimic android runtime environment with simplified APIs for analysis)

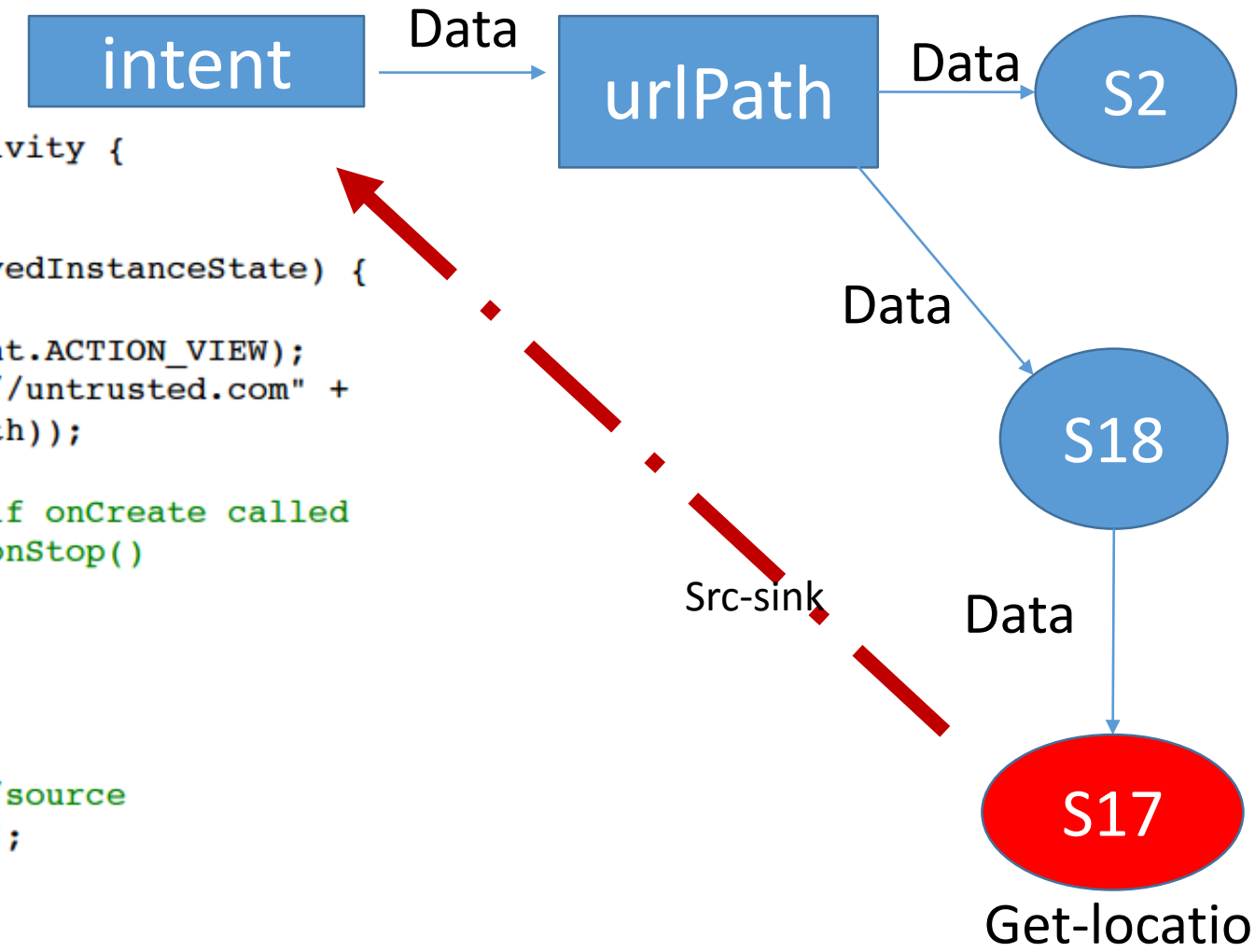
- Experiment

  Higher accuracy than FlowDroid

Motivation

```
1 public class EventOrder extends Activity {
2   String urlPath = "";
3
4   protected void onCreate(Bundle savedInstanceState) {
5     //...
6     Intent intent = new Intent(Intent.ACTION_VIEW);
7     intent.setData(Uri.parse("http://untrusted.com" +
8                              urlPath));
9
10    startActivity(intent); //sink, if onCreate called
11                           //after onStop()
12  }
13
14  //.... Other events
15
16  protected void onStop() {
17    Location loc = <get location> //source
18    urlPath = loc.getLatitude() + "";
19  }
20 }
```

(a) Event ordering example: Green arrow shows end-to-end flow.

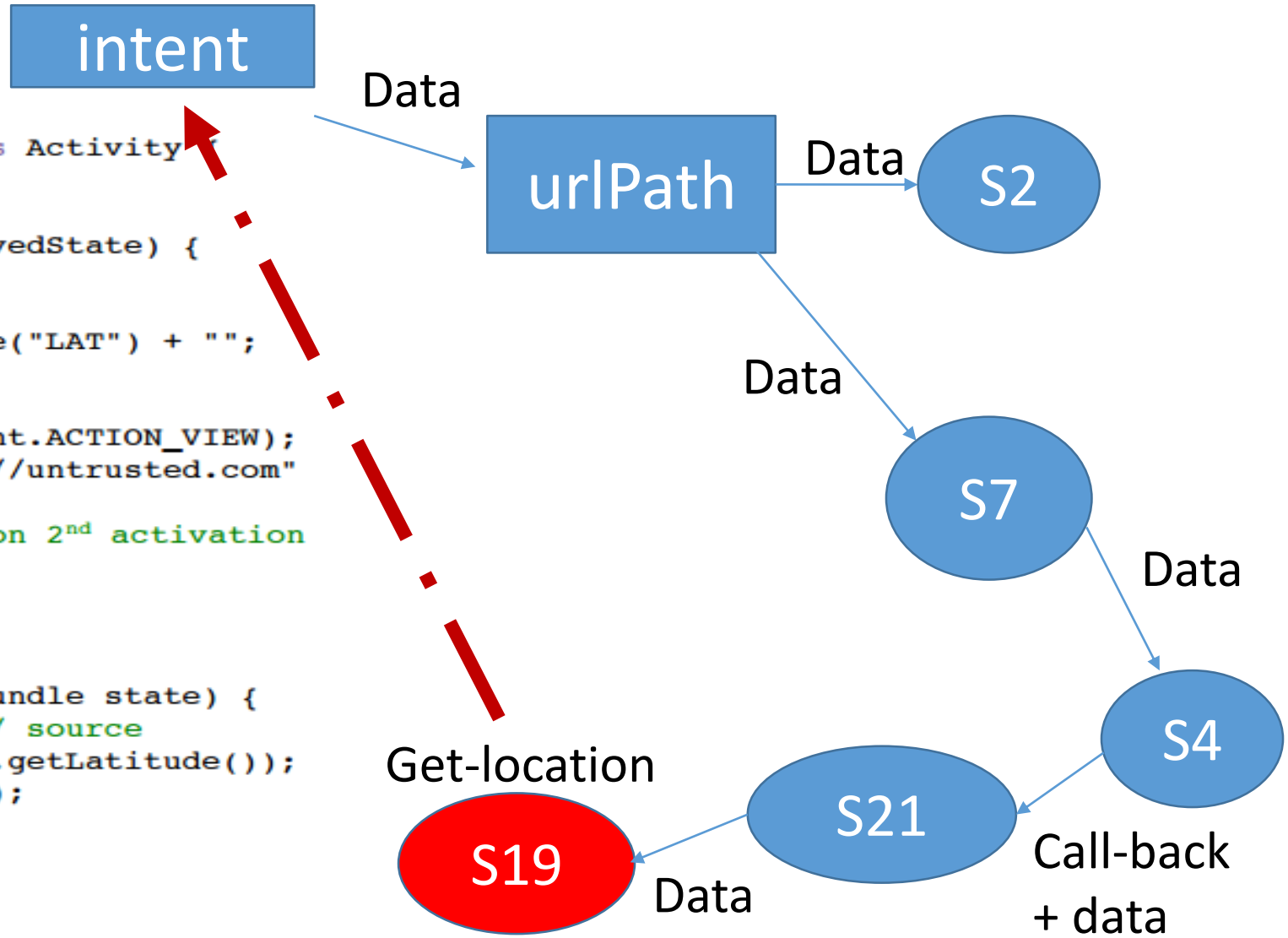


Order matters!

Motivation

```
1 public class CallbackContext extends Activity {
2   String urlPath = "";
3
4   protected void onCreate(Bundle savedState) {
5     //...
6     if (savedState != null) {
7       urlPath = savedState.getDouble("LAT") + "";
8     }
9
10    Intent intent = new Intent(Intent.ACTION_VIEW);
11    intent.setData(Uri.parse("http://untrusted.com"
12                    + urlPath));
13    startActivity(intent); //sink, on 2nd activation
14  }
15
16  //.... Other events
17
18  public void onSaveInstanceState(Bundle state) {
19    Location loc = <get location> // source
20    savedState.putDouble("LAT", loc.getLatitude());
21    super.onSaveInstanceState(state);
22  }
23 }
```

(b) Callback context example: Green arrow shows end-to-end flow.



callbacks matters!

Motivation

```
6                                        IBinder iB) {
7         mService = new Messenger(iB);
8       }
9       public void onServiceDisconnected(ComponentName cN) {
10        mService = null;
11      }
12    };
13
14    protected void onCreate(Bundle savedState) {
15      Intent intent = new Intent("ICCServiceAction");
16      bindService(intent, sc,
17                  Context.BIND_AUTO_CREATE);
18    }
19
20    public void buttonClick(View v) {
21      double lat =<get location>.getLatitude();  //source
22      Message msg = Message.obtain(null, 0, lat, 0);
23      mService.send(msg);
24    }
25  }
```

```
1   // IntentFilter defined in manifest to accept
2   // action = "ICCServiceAction"
3   public class ICCService extends Service {
4
5     final Messenger mMessenger =
6       new Messenger(new IncomingHandler());
7
8     class IncomingHandler extends Handler {
9       public void handleMessage(Message msg) {
10        double data = msg.arg1;     //tainted
11        Intent intent = new Intent(ICCService.this,
12                                   ICCSink.class);
13        intent.putExtra("DATA", data);
14        startActivity(intent);
15      }
16    }
17
18    public IBinder onBind(Intent intent) {
19      return mMessenger.getBinder();
20    }
21  }
```

implicit

explicit

```
1   public class ICCSink extends Activity {
2     double data = 0.0;
3
4     protected void onCreate(Bundle savedState) {
5       Intent intent = getIntent();
6       data = intent.getDoubleExtra("DATA", 0.0);
7     }
8
9     public void buttonClick(View v) {
10      Log.v("ICCSink", data + "");  //sink, leak of location
11    }
12  }
```

ICC matters!

Approaches:

Accurate Analysis Stubs:

1. (simply-implement of Android Device): 1.3M + 70K LOC

Object-Sensitive Points-to Analysis:

1. Add more precision with context sensitivity

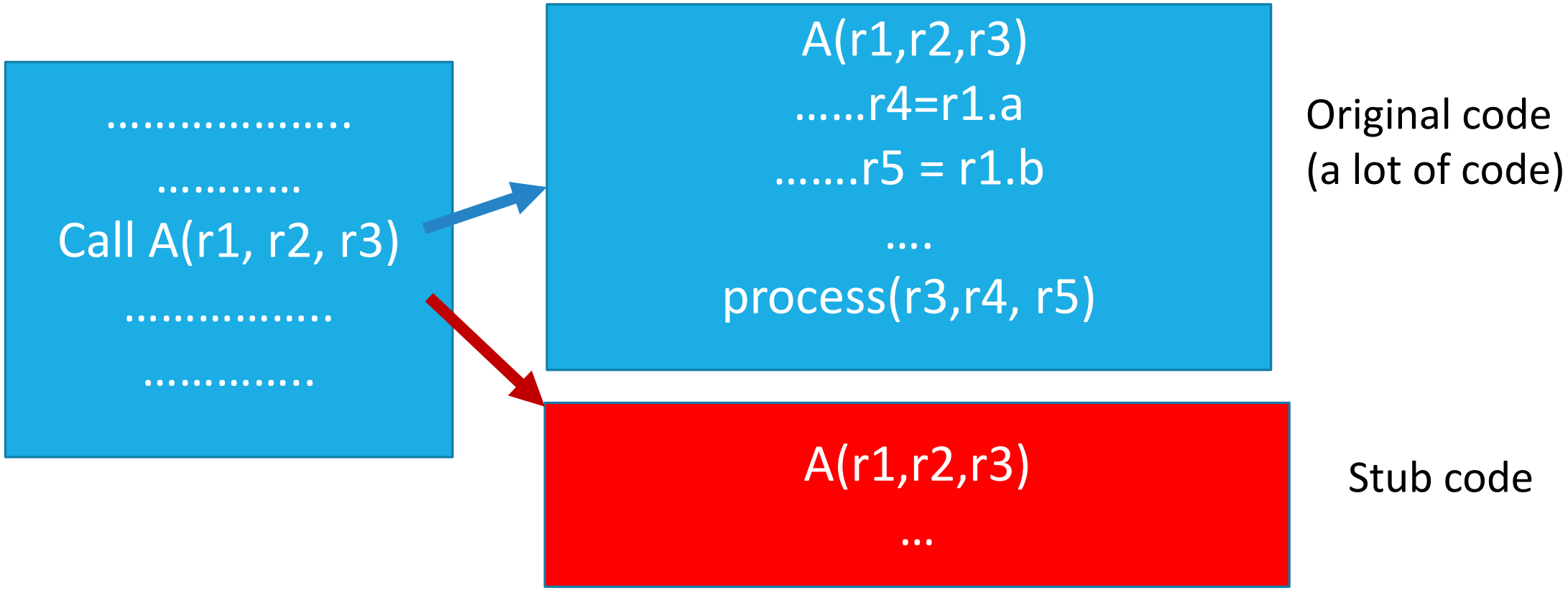
ICC- modeling :

1. JSA framework to resolve string analysis.

Accurate Analysis Stubs:

1. AOSP (Android Open Source Project) cannot model runtime behavior of Android applications.
2. Stubs(written) in Java, incompletely model the runtime behavior of model code, but keep semantics.
3. (Manually) add 3,176 native methods implementation, simplify 117 classes in standard library.

Accurate Analysis Stubs:



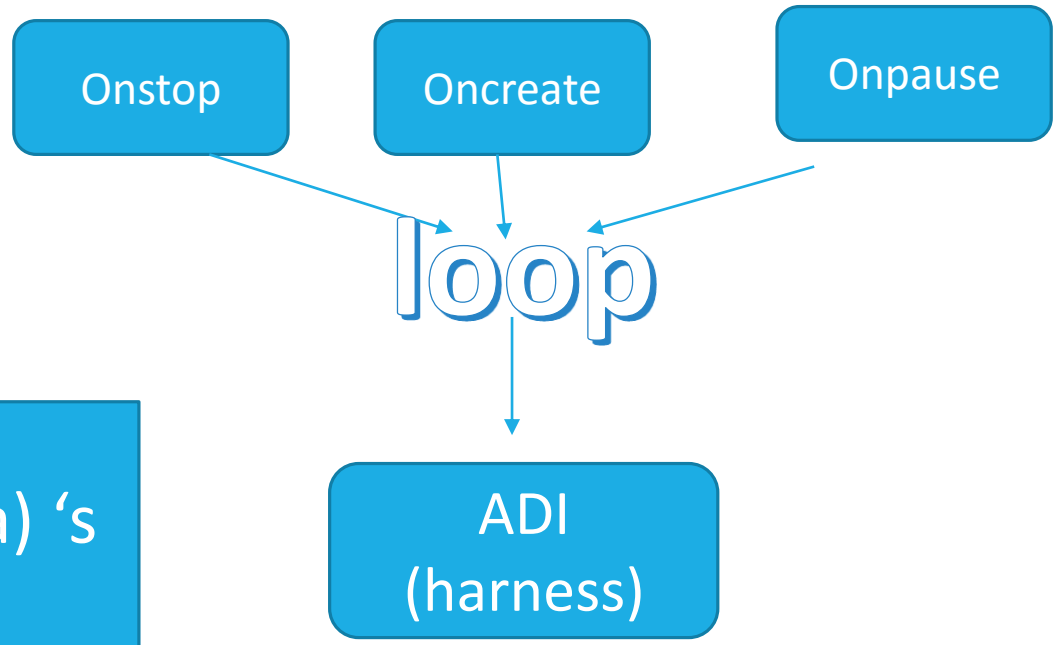
Event and Callback Dispatch

```
1 public class EventOrder extends Activity {
2   String urlPath = "";
3
4   protected void onCreate(Bundle savedInstanceState) {
5     //...
6     Intent intent = new Intent(Intent.ACTION_VIEW);
7     intent.setData(Uri.parse("http://untrusted.com" +
8                              urlPath));
9
10    startActivity(intent); //sink, if onCreate called
11                           //after onStop()
12  }
13
14  //.... Other events
15
16  protected void onStop() {
17    Location loc = <get location> //source
18    urlPath = loc.getLatitude()
19  }
20 }
```

(a) Event ordering example: Green a

Resolve Figure 1(a) 's problem

1. "A runtime implementation that models component event."
2. "Harness".

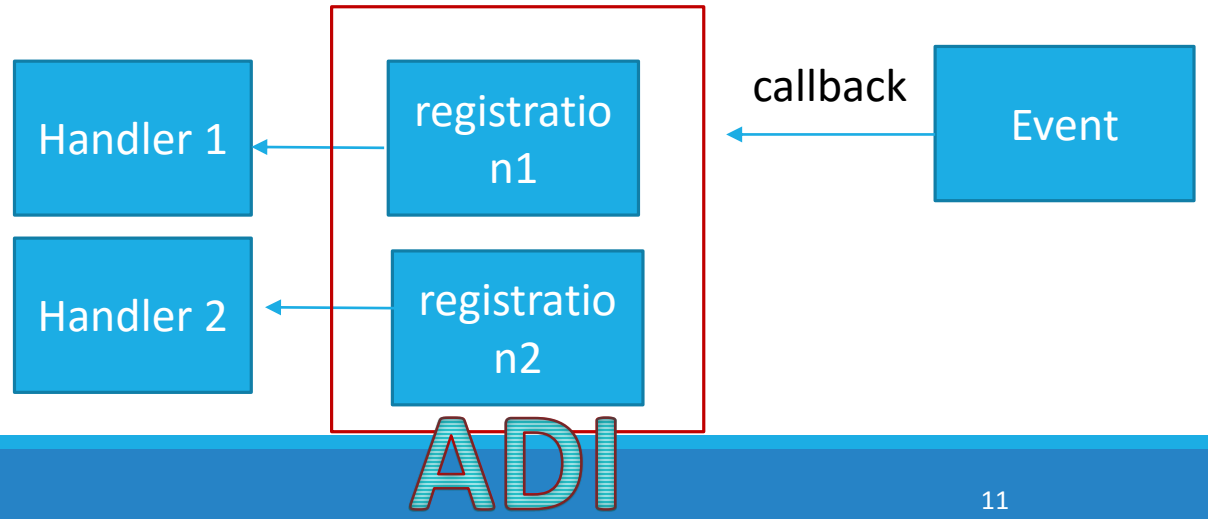


Event and Callback Dispatch

```
1 public class CallbackContext extends Activity {
2   String urlPath = "";
3
4   protected void onCreate(Bundle savedState) {
5     //...
6     if (savedState != null) {
7       urlPath = savedState.getDouble("LAT") + "";
8     }
9
10    Intent intent = new Intent(Intent.ACTION_VIEW);
11    intent.setData(Uri.parse("http://untrusted.com"
12                    + urlPath));
13    startActivity(intent); //sink, on 2nd activation
14  }
15
16  //.... Other events
17
18  public void onSaveInstanceState(Bundle state) {
19    Location loc = <get location> // source
20    savedState.putDouble("LAT", loc.getLatitude());
21    super.onSaveInstanceState(state);
22  }
23 }
```

Resolve Figure 1(b) 's problem

"Implement the callback registration method to invoke the application's callback handler method with the appropriate arguments."



ADI implementation

Android Device Implementation (ADI)

```
1  package android.os;
2  public class Bundle ... {
3    private Map<String,Object> mMap =
4                  new HashMap<String,Object>(); Ⓗ
5
6    public void put(String k, Object v) {
7      mMap.put(k,v);
8    }
9
10   public Object get(String k) {
11     return mMap.get(k);
12   }
13 }
```

stub

```
1  package java.util;
2  public class HashMap<K,V>... {
3    private Entry[] table = new Entry[size]; Ⓣ
4
5    public void put(K key, V value) {
6      ...
7      table[index] = new Entry<K,V>(key, value); Ⓔ
8    }
9
10   public V get(Object key) {
11     ...
12     e = table[indexFor(hash, table.length)];
13     ...
14     return e;
15   }
16 }
```

stub

Simply implementation

Android Application Source Code

```
1  public class Activity1 extends Activity {
2    ...
3    Bundle bundle1 = new Bundle(); Ⓝ
4    bundle1.put("data", <notSensitive>);
5    ...
6    sink(bundle1);  //not a sensitive flow
7  }
```

```
1  public class Activity2 extends Activity {
2    ...
3    double sensitive = location.getLatitude(); //source
4    Bundle bundle2 = new Bundle(); Ⓢ
5    bundle2.put("data", sensitive);
6    ...
7    sink(bundle2);  //flow of sensitive -> sink
8  }
```

Fig. 2. Example source code for our ADI and two `Activity` objects illustrating the challenges of points-to and information flow analysis.

Object-Sensitive Points-to Analysis

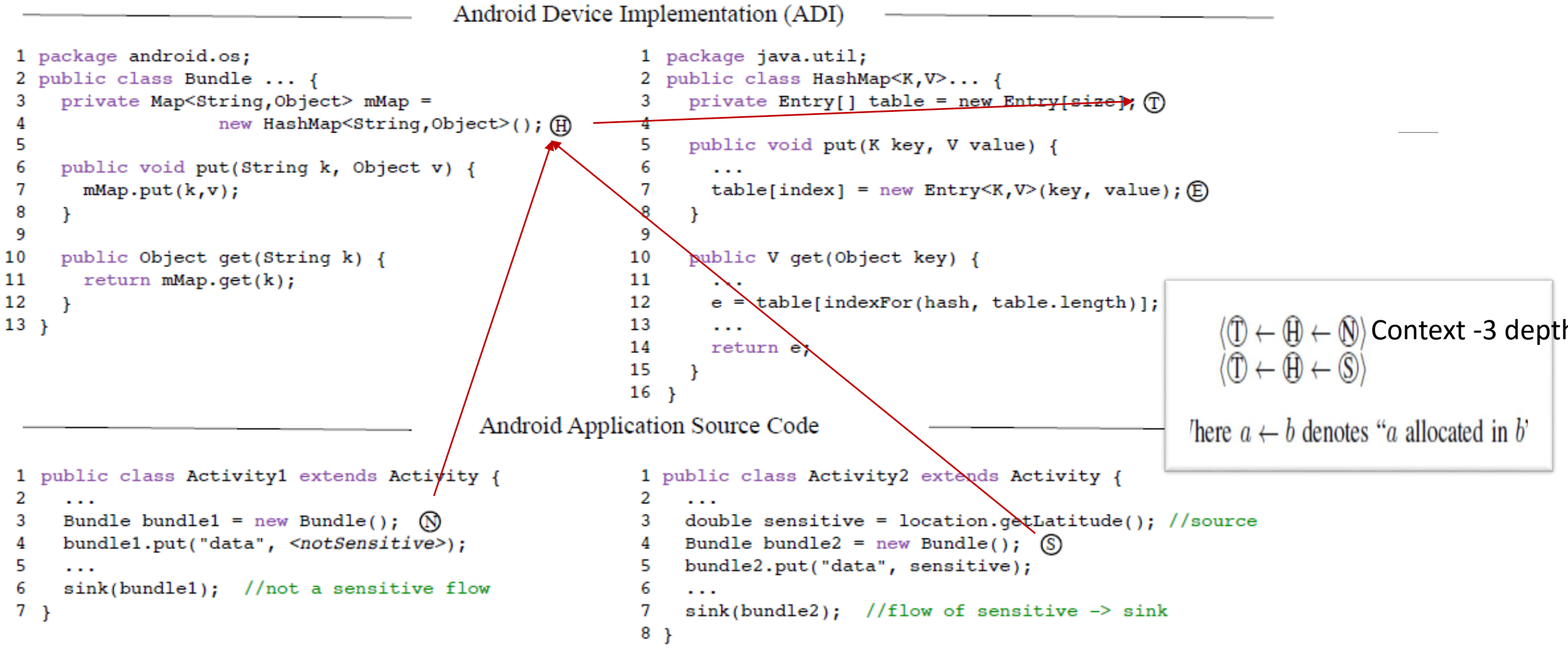


Fig. 2. Example source code for our ADI and two `Activity` objects illustrating the challenges of points-to and information flow analysis.

ICC modeling

1. Re-implement ICC model
2. Resolve explicit calls: JSA (Java string analysis to find the string in an explicit call) – flow sensitive analyzer
3. Resolve implicit calls: Parse Androidmanifest and record implicit intents + intentFilter

Transforming ICC calls (dynamic -> static)

Source Method	Target Method Call Injected
`Context: void send*Broadcast(Intent, ...)` *[6 variants]*	`BroadcastReceiver: void onReceive(Intent)`
`Activity: void startActivit*(Intent, ...)` *[6 variants]*	`Activity: void setIntent(Intent)`
`Context: void bindService(Intent, Connection)`	`Service: void droidSafeOnBind(Intent, Connection)`
`Context: void startService(Intent)`	`Service: void onStartCommant(Intent, ...)`
`ContentResolver: insert, query, delete, update`	`ContentProvider: insert, query, delete, update`

Fig. 3. DroidSafe's ICC source to target methods transformations.

"transform ICC initiation calls into appropriate method calls at the destination(s), linking the data flows between source and destination"

ICC calls

```
6                                        IBinder iB) {
7         mService = new Messenger(iB);
8       }
9       public void onServiceDisconnected(ComponentName cN) {
10        mService = null;
11      }
12    };
13
14    protected void onCreate(Bundle savedState) {
15      Intent intent = new Intent("ICCServiceAction");
16      bindService(intent, sc,
17                  Context.BIND_AUTO_CREATE);
18    }
19
20    public void buttonClick(View v) {
21      double lat =<get location>.getLatitude();  //source
22      Message msg = Message.obtain(null, 0, lat, 0);
23      mService.send(msg);
24    }
25  }
```

```
1   // IntentFilter defined in manifest to accept
2   // action = "ICCServiceAction"
3   public class ICCService extends Service {
4
5     final Messenger mMessenger =
6         new Messenger(new IncomingHandler());
7
8     class IncomingHandler extends Handler {
9       public void handleMessage(Message msg) {
10        double data = msg.arg1;     //tainted
11        Intent intent = new Intent(ICCService.this,
12                                   ICCSink.class);
13        intent.putExtra("DATA", data);
14        startActivity(intent);
15      }
16    }
17
18    public IBinder onBind(Intent intent) {
19      return mMessenger.getBinder();
20    }
21  }
```

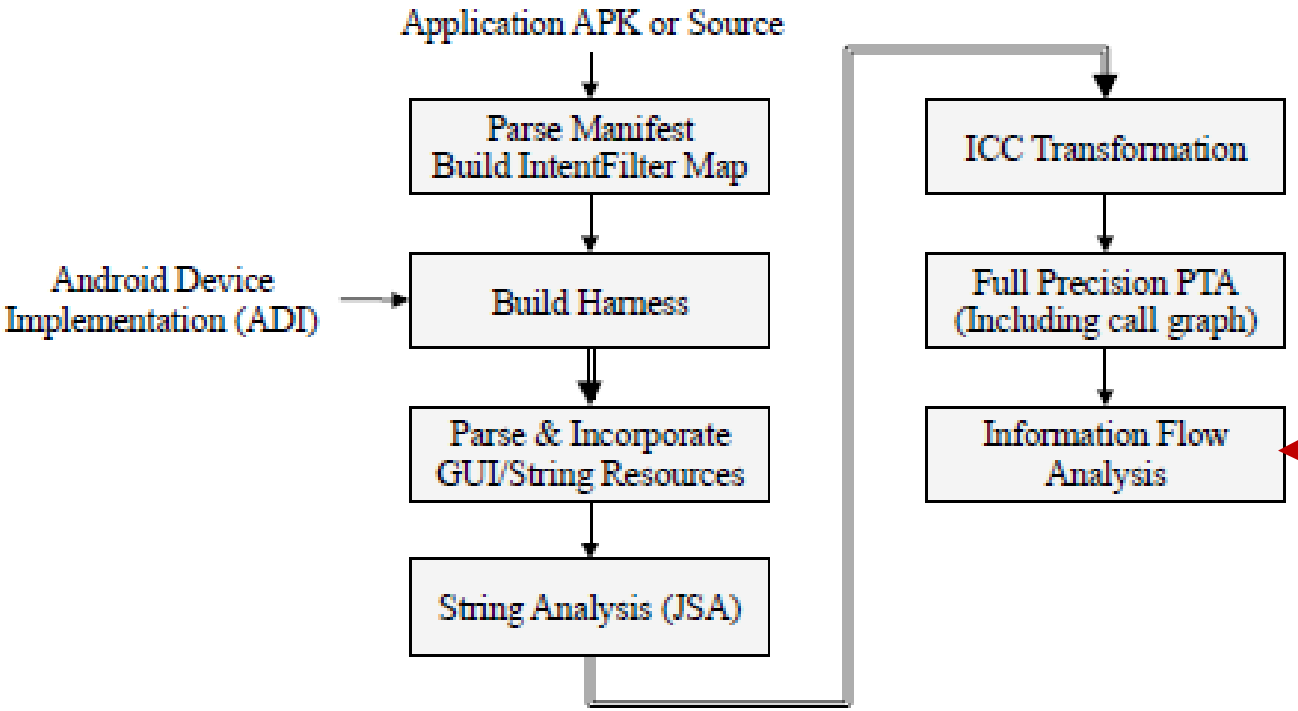
Explicit(JSA)

Implicit(transform)

```
1   public class ICCSink extends Activity {
2     double data = 0.0;
3
4     protected void onCreate(Bundle savedState) {
5       Intent intent = getIntent();
6       data = intent.getDoubleExtra("DATA", 0.0);
7     }
8
9     public void buttonClick(View v) {
10      Log.v("ICCSink", data + "");  //sink, leak of location
11    }
12  }
```

Oncreate -> ICCservice -> ICCSInk

DroidSafe tool



A forward data-flow analysis

From source -> sink

Fig. 4. Phases of the DroidSafe Tool. Double lines denote an update of the PTA result is calculated for the next phase.

DroidBench Results

Tool	Missed Flows Explicit / Implicit	Accuracy	False Positives	Precision
DroidSafe	0/6	93.9%	13	87.6%
FlowDroid	12/7	80.6%	30	72.5%

Fig. 5. DROIDBENCH results for DroidSafe and FlowDroid.

Recall

Precision = find true flows/ total true flows

Recall = find true flows / total find flows

Statistics of APAC apps

Application	Lines of Code	Malicious Flow	
		Source	Sink
AgentSmith	1,481	Clipboard	Network
AndroidGame	63,755	Image Metadata	Network
AndroidMap	8,491	Location	Network
AndroidsFortune	14,621	Device ID	Network
AudioSidekick	2,444	Mic	Network
AWeather	1,837	Network	Network
BatteryIndicator	5,319	Image	Network
Butane	2,506	SMS	Network

Application	Lines of Code	Malicious Flow	
		Source	Sink
CalcF	861	User Input	Network
DeviceAdmin2	2,289	System Info	Network
FillInFun	82,602	Contact	SMS
KitteyKittey	962	Image Metadata	Network
PicViewer	221	Image Metadata	Network
Quickdroid	6,155	Contact, Bookmark	IPC
RunningApp	1,785	User Input	NFC
ShareLoc	372	Location	Network

Application	Lines of Code	Malicious Flow	
		Source	Sink
ShyGuyCRM	3,811	Contact	Email
SmartWebCam	1,176	Camera	AIDL
SMSBackup	387	SMS, Image, Browser	File
SMSBlocker	3,775	SMS	Network
SMSPopup	17,953	SMS	SMS
SnapshotShare	13,461	Screenshot	Network
SourceViewer	208	Device ID	Network
UltraCoolMap	2,658	Location	Network

Fig. 6. APAC Information-Flow Applications: Size and malicious flows details.

APAC results

Application	Malicious Flows	DroidSafe						FlowDroid		
		Reachable Lines (including ADI)	Analysis Time (sec)	Reachable Source Calls	Reachable Sink Calls	Total Flows	Missed Malicious Flows	Analysis Time (sec)	Total Flows	Missed Malicious Flows
AgentSmith	1	123,881	434	53	60	167	0	60	123	1
AndroidGame	1	82,170	499	11	18	37	0	Did not complete		1
AndroidMap	2	102,236	698	78	41	132	0	54	25	2
AndroidsFortune	1	130,003	752	72	183	304	0	159	208	0
AudioSidekick	2	126,223	507	62	50	89	0	41	28	2
AWeather	1	126,218	491	35	30	72	0	116	57	1
BatteryIndicator	1	122,132	846	64	135	113	0	106	176	1
Butane	4	173,934	625	73	102	392	0	68	109	2
CalcF	2	117,414	374	11	21	11	0	33	5	0
DeviceAdmin2	2	137,046	358	17	33	5	0	47	6	2
FillInFun	2	123,016	601	22	64	14	0	75	25	1
KitteyKittey	1	110,584	271	4	1	2	0	47	1	1
PicViewer	3	118,019	360	7	3	8	0	20	0	3
Quickdroid	19	119,427	399	103	65	278	0	64	231	19
RunningApp	1	126,629	579	51	34	59	0	75	94	1
ShareLoc	4	119,771	1,051	6	4	7	0	28	7	4
ShyGuyCRM	1	177,853	1,255	105	99	463	0	78	82	1
SmartWebCam	1	126,029	1,649	101	267	21	0	50	30	1
SMSBackup	10	108,317	269	25	7	26	0	20	0	10
SMSBlocker	1	125,531	419	12	105	23	0	42	12	1
SMSPopup	3	149,824	1,477	180	182	918	0	298	304	3
SnapshotShare	1	130,111	590	89	29	108	0	92	71	1
SourceViewer	1	118,943	384	13	11	8	0	23	4	1
UltraCoolMap	4	121,507	407	14	9	12	0	34	42	4
Total	**69**					**3,269**	**0**		**1,640**	**63**

Fig. 7. APAC Information-Flow Applications: DroidSafe and FlowDroid evaluation results.

Discussions

The paper deploys a static analysis, why it always mention "a runtime implementation" in the paper?

Explicitly resolve dynamic decisions with static analysis

Why do the authors have to implement so many APIs/methods by themselves instead of making some assumptions about these methods?

Precision and scalability

Thanks