DroidSafe

Information-Flow Analysis of Android Applications in DroidSafe

By Angel Castillo and Sam Ingram
What is the problem?

● Sensitive information
  ○ (Device ID, sensor data, file data, etc..) leaks are one of the most prominent security threats to the Android ecosystem
● Applications legitimately need access to sensitive information, but only for a limited specific purpose
● Dynamic analysis frameworks miss information flows, possible DOS attacks
How information leaks happen

Inter-Component Communication

Application A

Component A

....

getDeviceID();

Intent

Application B

Component B

..  

sendEmail(data)
Leaks cont.

Application A

Component A

.....
Data = getLocation();
Intent = new Intent(X.comp.Y)
intent.put("loc",data)
startActivity(intent)

Application B

Component B

Loc = intent.get("loc")
What is DroidSafe?

- Open Source
- 1.2 MOC
- Static Information Flow Analysis Tool
- Android Model
# Previous Solutions

<table>
<thead>
<tr>
<th>Solution</th>
<th>Accuracy</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>DroidBench</td>
<td>34.9%</td>
<td>79%</td>
</tr>
<tr>
<td>FlowDroid</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>DroidSafe</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
APAC

- 3 Red Team Organizations
- 27 Applications
- 69 Malicious Flows
- 8.7% vs 100%
How does it work?

- Android Open Source Project
- Analysis Stubs
- Sources and Sinks/ Intent
- Event and Callback Dispatch
- Object-Sensitive Points-To Analysis
Android Open Source Project

https://source.android.com/

Java implementation of Android environment

Developed analysis and model synergistically
Sources, Sinks and Intents

Sources are flows initiated from calls to Android API methods

Sinks are destinations that attackers route sensitive data to (Network, NFC, file system, email, SMS, ...etc)

Manually identified 4,051 source methods and 2,116 sink methods

Droidsafe takes ICC initiation calls (methods that pass Intent Objects for ICC) and transforms them into method calls to link sources and sinks together
Information Flow Analysis

Memory = Local x Instance x Static x Array

Local = Ctx x Var -> InfoVal

Instance = Loc x Field -> InfoVal

Static = Class x Field -> InfoVal

Array = Loc -> InfoVal

Ctx,Loc = AllocSite
Accurate Analysis Stubs

Written in Java, these analysis stubs are used to analyze code whose full semantics lie outside the scope of AOSP.

Models things like event callback initiation, life-cycle events, hidden state maintained by android runtime (Intent, Parcel, Activity).

Added accurate analysis stubs for 3,176 native methods to model missing native code.
Event And Callback Dispatch

- Some classes are asynchronous
  - Activity, Service, BroadcastReceiver, and ContentProvider
- They perform many actions
  - Component creation, shared and saved state, life-cycle event firing and argument context, and callback event firing and argument context.
- All event orderings must be considered
  - Major reason for static analysis
- Callback handler object intercepts database changes via a stub
public class EventOrder extends Activity {
    String urlPath = "";

    protected void onCreate(Bundle savedInstanceState) {
        //...
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://untrusted.com" +
            urlPath));

        startActivity(intent); //sink, if onCreate called
        //after onStop()
    }

    //.... Other events

    protected void onStop() {
        Location loc = <get location> //source
        urlPath = loc.getLatitude() + "";
    }
}
Object-Sensitive Points-To Analysis

- Android uses complicated features like polymorphism and inheritance
- This technique allows the code to be analysed statically
- Depth varies from 0-4

```java
public class Activity1 extends Activity {
    ... 
    Bundle bundle1 = new Bundle(); // T
    bundle1.put("data", <notSensitive>);
    ... 
    sink(bundle1); // not a sensitive flow
}

public class Activity2 extends Activity {
    ... 
    double sensitive = location.getLatitude(); //source
    Bundle bundle2 = new Bundle(); // S
    bundle2.put("data", sensitive);
    ... 
    sink(bundle2); // flow of sensitive -> sink
}
```
ICC Modeling

- Message passing via Intent objects
  - Many ways to pass messages
  - Too many components to test all possible combinations
- IntentFilter find which components the message could be passed through
- Android Model used to track the way that messages can be passed
Example:

```java
public void buttonClick(View v) {
    double lat = <get location>.getLatitude(); //source
    Message msg = Message.obtain(null, 0, lat, 0);
    mService.send(msg);
}

public void handleMessage(Message msg) {
    double data = msg.arg1;       //tainted
    Intent intent = new Intent(ICCService.this,
                                 ICCSink.class);
    intent.putExtra("DATA", data);
    startActivity(intent);
}

public void buttonClick(View v) {
    Log.v("ICCSink", data + ""); //sink, leak of location
}
```
Advantages

● Much better than the alternatives
  ○ Accurate
  ○ Precise
● Easy to update
  ○ 2 person-weeks to update
● Fast Runtime
Disadvantages

- Some missed flows
- Some false positives
- Device can’t be rooted
- No dynamic code loading
- No side channels or implicit flows
- Inject a sensitive flow in a native method
- Exfiltrate API-injected information
- Android is large, could have missed something
Conclusion

- It is significantly more useful than the competition
- Hard to say how well it performs
  - Glossed over disadvantages
  - Not tested on popular applications
- A program called Merlin is doing similar work that might complement DroidSafe
Questions?