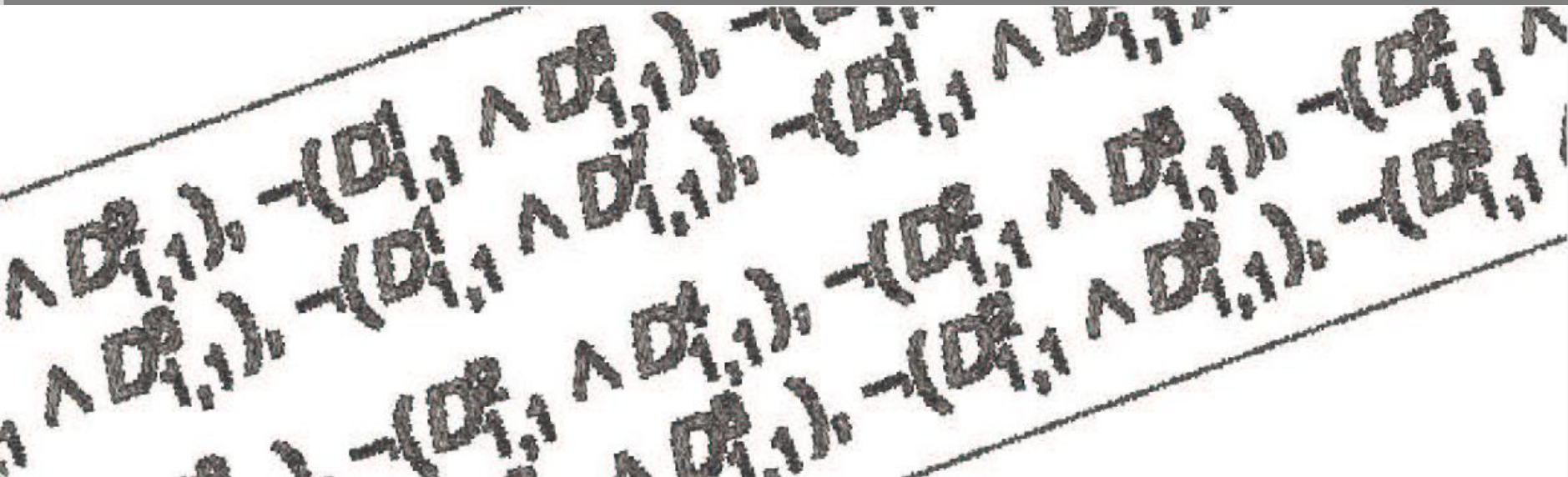


Taint Analyse für Android Apps

Proseminar im WiSe15/16: Disaster in der Software-Sicherheit

INSTITUTE FOR THEORETICAL INFORMATICS, KIT



Motivation

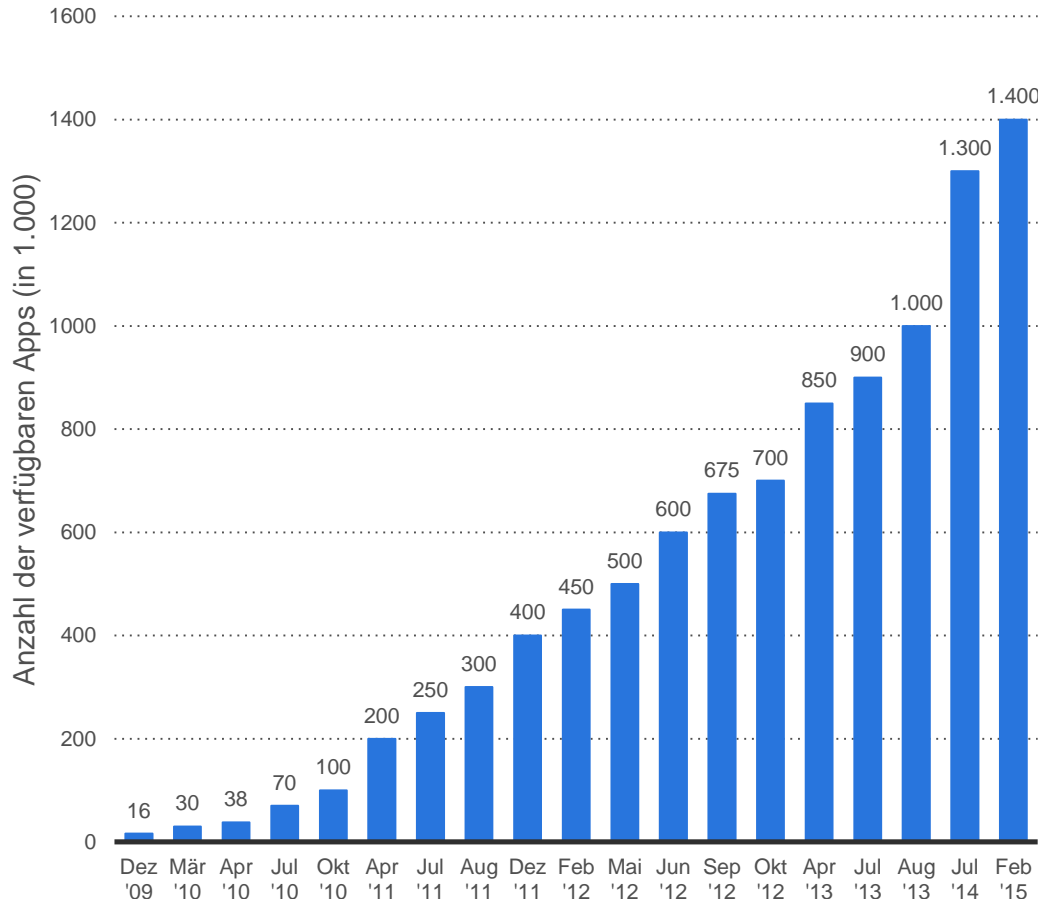
■ Beispiel

```
1 connection = ...
2 stmt = connection.createStatement();
3
4 BufferedReader br = ...
5 String name = br.readLine();
6 String sql = "SELECT * FROM users WHERE name=" + name + ";";
7
8 stmt.executeQuery(sql);
```

■ SQL Injektion möglich

- foo; DROP TABLE users
- Oder z.B. ein beliebiges SELECT Statement

Motivation



© Statista 2016, **Quellen:** Android; Google

- Google Play Store bietet große Angriffsfläche
- Analyse die solche gefährlichen Apps aufspürt wäre sinnvoll

Motivation – Statische Analyse

- Im folgenden statische statt dynamische Analyse
- Vorteil
 - Kompilat muss nicht ausgeführt werden
 - heutige Malware kann erkennen, ob sie überwacht wird und kann ihr Verhalten anpassen

Gliederung

- Motivation
- Taint Analyse
 - Formal
 - Sanitization
 - Präzision
- Android
- FlowDroid
 - Idee
 - Evaluierung
- Ausblick

TAINT ANALYSE

Taint Analyse

- *to taint = beschmutzen*

- Datenfluss Analyse
- Bestandteil einiger Programmiersprachen
 - Pearl, Ruby

- Die Idee:
 - jede von außen veränderbare Variable birgt ein Sicherheitsrisiko

Taint Analyse – Spezifikation

■ Das Ziel:

- absichern der Software gegen externe Angriffe, sowie interne Risiken
- finde Datenflüsse zwischen möglichen tainted Datenquellen (**source**) und kritischen Funktionen (**sinks**)

■ Spezifikation

- **source**: Quelle von sensitiven Daten
- **sink**: Funktion, die Daten möglicherweise an nicht vertrauenswürdige Beobachter weitergibt

Taint Analyse – Vorgehen

- **tainted** Daten kommen durch **sources** rein
 - z.B. Benutzereingabe
- **tainted** Daten werden Variablen zugewiesen
- suche Datenfluss von **sources** zu möglichen **sink**

Taint Analyse – Formal


Source

$$\frac{src(m, l), m \rightarrow o}{tainted(o, l)}$$

- $src(m, l)$
 - Parameter m bekommt tainted Daten aus der Source l
- $m \rightarrow o$
 - m zeigt auf Objektrepräsentant o
- $tainted(o, l)$
 - Objektrepräsentant o durch l tainted


Taint Analyse – Beispiel

```
1 connection = ...  
2 stmt = connection.createStatement();  
3  
4 BufferedReader br = ...  
5 String name = br.readLine();  
6 String sql = "SELECT * FROM users WHERE name=" + name + " ";  
7  
8 stmt.executeQuery(sql);
```



Taint Analyse – Beispiel

```
1 connection = ...
2 stmt = connection.createStatement();
3
4 BufferedReader br = ...
5 String name = br.readLine();
6 String sql = "SELECT * FROM users WHERE name=" + name + ";";
7
8 stmt.executeQuery(sql);
```



Taint Analyse – Beispiel

```
1 connection = ...
2 stmt = connection.createStatement();
3
4 Buffered reader br = ...
5 String name = br.readLine();
6 String sql = "SELECT * FROM users WHERE name=" + name + " ";
7
8 stmt.executeQuery(sql);
```

Taint Analyse – Beispiel

```

1 connection = ...
2 stmt = connection.createStatement();
3
4 Buffered Reader br = ...
5 String name = br.readLine(); //name → o1
6 String sql = "SELECT * FROM users WHERE name=" + name + " ";
7
8 stmt.executeQuery(sql);

```

Source

$$\frac{src(m, l), m \rightarrow o}{tainted(o, l)}$$

$$\frac{src(name, br.readLine), name \rightarrow o1}{tainted(o1, br.readLine)}$$

Taint Analyse – Formal

Transfer

$$\frac{\textit{tainted}(o1, l), m \rightarrow o1, n \rightarrow o2, \textit{transfer}(m, n)}{\textit{tainted}(o2, l)}$$

- $\textit{tainted}(o1, l)$
 - Objektrepräsentanten $o1$ durch l tainted
- $m \rightarrow o1, n \rightarrow o2$
 - m und n zeigen jeweils auf einen Objektrepräsentanten
- $\textit{transfer}(m, n)$
 - Daten von m werden nach n übertragen
- $\textit{tainted}(o2, l)$

Taint Analyse – Beispiel


Transfer

$$\frac{\text{tainted}(o1, l), m \rightarrow o1, n \rightarrow o2, \text{transfer}(m, n)}{\text{tainted}(o2, l)}$$

```

1 connection = ...
2 stmt = connection.createStatement();
3
4 BufferedReader br = ...
5 String name = br.readLine(); //name → o1
6 String sql = "SELECT * FROM users WHERE name=" + name + ";";
7 //sql → o2
8 stmt.executeQuery(sql);

```



$$\frac{\text{tainted}(o1, \text{br.readLine}), \text{name} \rightarrow o1, \text{sql} \rightarrow o2, \text{transfer}(\text{name}, \text{sql})}{\text{tainted}(o2, \text{br.readLine})}$$

Taint Analyse – Formal

Sink

$$\frac{\textit{tainted}(o, so), m \rightarrow o, \textit{sink}(m, si)}{\textit{flow}(so, si)}$$

- $\textit{tainted}(o, so), m \rightarrow o$
 - m zeigt auf Objektrepräsentant o , der durch so tainted ist
- $\textit{sink}(m, si)$
 - Parameter m wird Sink si übergeben
- $\textit{flow}(so, si)$
 - möglicher verschmutzter Datenfluss von so nach si

Taint Analyse – Beispiel

Sink

$$\frac{\textit{tainted}(o,so), m \rightarrow o, \textit{sink}(m,si)}{\textit{flow}(so,si)}$$

```

1 connection = ...
2 stmt = connection.createStatement();
3
4 BufferedReader br = ...
5 String name = br.readLine(); //name → o1
6 String sql = "SELECT * FROM users WHERE name=" + name + " ";
7 //sql → o2
8 stmt.executeQuery(sql);

```

$$\frac{\textit{tainted}(o2,br.readLine), sql \rightarrow o2, \textit{sink}(sql,stmt.executeQuery)}{\textit{flow}(br.readLine,stmt.executeQuery)}$$

Taint Analyse – Sanitization

■ Beispiel

```
1 connection = ...
2 stmt = connection.createStatement();
3
4 BufferedReader br = ...
5 String name = br.readLine();
6 name = onlyLetters(name);
7
8 String sql = "SELECT * FROM users WHERE name=" + name + " ";
9
10 stmt.executeQuery(sql);
```

Taint Analyse – Sanitization

- *to sanitize = säubern*
- eigene oder eingebaute Funktionen können zur sanitization genutzt werden
 - PHP: htmlspecialchars: konvertiert HTML Zeichen in ihre Instanzen (“<” → “<”)

Taint Analyse – Sanitization

- **Vorsicht:** Daten werden nach sanitization als harmlos angenommen
- Probleme
 - Eingabe sanitization hängt vom sink Typ ab
 - schwierig im vornherein Kriterien für sanitization festzulegen

Taint Analyse – Präzision

■ Beispiel

```
1 void flowSensitive() {  
2     String s = "value";  
3     sink(s);  
4     s = source();  
5 }
```

- s wird zuerst übergeben, danach kommen **tainted** Daten
- Beachtung der Reihenfolge sinnvoll
→ **Fluss-Sensitivität**

Taint Analyse – Präzision

■ Beispiel

```
1 void contextSensitive() {  
2     String s1 = id(source());  
3     String s2 = id("123");  
4     sink(s2);  
5 }  
  
6 String id(String s) {  
7     return s;  
8 }
```

■ s zeigt auf s1 und s2 → **tainted** Daten fließen in **sink**

■ beachtet man Kontext zeigt s entweder auf s1 oder s2

→ **Kontext-Sensitivität**

Taint Analyse – Präzision

■ Beispiel

```
1 void objectSensitive() {  
2     Foo o1 = new Foo();  
3     Foo o2 = new Foo();  
4     o1.value = "123";  
5     o2.value = source();  
6  
7     sink(o1.getValue());  
8 }  
  
9 class Foo {  
10     String value;  
11     String getValue() {  
12         return value;  
13     }  
14 }
```

■ *value* zeigt auf *o1* und *o2* → **sink** ist **tainted**

■ Subtyp der Kontext-Sensitivität

→ **Objekt-Sensitivität**

Taint Analyse – Präzision

■ Beispiel

```
1 void fieldSensitive() {  
2     Foo o = new Foo();  
3     o.field1 = "123";  
4     o.field2 = source();  
5     sink(o.field1);  
6 }  
7 class Foo {  
8     String field1;  
9     String field2;  
10 }
```

- Felder werden nicht einzeln betrachten sondern im Basis Objekt zusammengeführt

- *o1* wird als **tainted** markiert, da *o1.field2* **tainted** ist

→ **Feld-Sensitivität**

Taint Analyse – Zusammenfassung

Schlussregeln

$$\textbf{SOURCE: } \frac{\text{src}(m, l), m \rightarrow o}{\text{tainted}(o, l)}$$

$$\textbf{TRANSFER: } \frac{\text{tainted}(o1, l), m \rightarrow o1, n \rightarrow o2, \text{transfer}(m, n)}{\text{tainted}(o2, l)}$$

$$\textbf{SINK: } \frac{\text{tainted}(o, so), m \rightarrow o, \text{sink}(m, si)}{\text{flow}(so, si)}$$

- Sanitization
- Vermeidung von false positives

ANDROID

- Android Apps können aus vier verschiedenen Komponenten bestehen

Activity

- Einzelner Screen, der für den Benutzer sichtbar ist

Service

- Aktion die im Hintergrund abläuft

Content Provider

- Zuständig für das Speichern von Daten

Broadcast Receiver

- Wartet auf globales Ereignis und führt vordefinierte Aktion durch

Android

■ Probleme für Taint Analyse

- mehrere Einstiegspunkte
- Komponenten haben besonderes Verhalten
 - start/stop oder pause/resume
- komplizierter Lebenszyklus
- Wie spezifiziert man **sinks** und **sources** bei Android?

FlowDroid

■ Idee

- Modellierung eines präzisen Android Lebenszyklus
- Analyse besitzt folgende Eigenschaften
 - Fluss-Sensitivität
 - Kontext-Sensitivität
 - Objekt-Sensitivität
 - Feld-Sensitivität
- benutzt das Tool **SuSi** zum identifizieren von **sources** und **sinks**
- <https://github.com/secure-software-engineering/soot-infoflow-android/wiki>

FlowDroid – Lebenszyklus

- dummy-main-Methode für jede App
- enthält nur Teile des Lebenszyklus die auch auftreten können
- Dummy-main ruft nacheinander Teile des Zykluses auf

FlowDroid – Evaluierung

■ DroidBench

- Testumgebung
- enthält 39 Apps mit verschiedenen Analyse Problemen
- <https://github.com/secure-software-engineering/DroidBench/>

- Vergleich mit **App Scan** von *IBM* und **Fortify SCA** von *HP*

FlowDroid – Evaluierung

⊛ korrekter Fehler gefunden

★ false positive

○ Fehler nicht gefunden
kein Fehler und keiner gefunden

■ FlowDroid deutlich besser bei *Callbacks* und *Lifecycle* Tests

App Name	AppScan	Fortify	FlowDroid
Arrays and Lists			
ArrayAccess1			★
ArrayAccess2	★	★	★
ListAccess1	★	★	★
Callbacks			
AnonymousClass1	○	⊛	⊛
Button1	○	⊛	⊛
Button2	⊛ ○ ○	⊛ ○ ○	⊛ ⊛ ⊛ ★
LocationLeak1	○ ○	○ ○	⊛ ⊛
LocationLeak2	○ ○	○ ○	⊛ ⊛
MethodOverride1	⊛	⊛	⊛
Field and Object Sensitivity			
FieldSensitivity1			
FieldSensitivity2			
FieldSensitivity3	⊛	⊛	⊛
FieldSensitivity4	★		
InheritedObjects1	⊛	⊛	⊛
ObjectSensitivity1			
ObjectSensitivity2	★		
Inter-App Communication			
IntentSink1	⊛	⊛	○
IntentSink2	⊛	⊛	⊛
ActivityCommunication1	⊛	⊛	⊛
Lifecycle			
BroadcastReceiverLifecycle1	⊛	⊛	⊛
ActivityLifecycle1	⊛	⊛	⊛
ActivityLifecycle2	○	⊛	⊛
ActivityLifecycle3	○	○	⊛
ActivityLifecycle4	○	⊛	⊛
ServiceLifecycle1	○	○	⊛

FlowDroid – Evaluierung

- ⊛ korrekter Fehler gefunden
- ★ false positive
- Fehler nicht gefunden
- kein Fehler und keiner gefunden

- Präzision etwas besser bei FlowDroid (p)
- deutlich besser beim finden aller Datenlecks (r)

App Name	AppScan	Fortify	FlowDroid
General Java			
Loop1	⊛	○	⊛
Loop2	⊛	○	⊛
SourceCodeSpecific1	⊛	⊛	⊛
StaticInitialization1	○	⊛	○
UnreachableCode		★	
Miscellaneous Android-Specific			
PrivateDataLeak1	○	○	⊛
PrivateDataLeak2	⊛	⊛	⊛
DirectLeak1	⊛	⊛	⊛
InactiveActivity	★	★	
LogNoLeak			
Sum, Precision and Recall			
⊛, higher is better	14	17	26
★, lower is better	5	4	4
○, lower is better	14	11	2
Precision $p = \frac{\text{⊛}}{(\text{⊛} + \text{★})}$	74%	81%	86%
Recall $r = \frac{\text{⊛}}{(\text{⊛} + \text{○})}$	50%	61%	93%
F-measure $2pr/(p+r)$	0.60	0.70	0.89

FlowDroid – Evaluierung

- Top 500 Apps des Google PlayStores analysiert
 - ca. eine Minute pro App
 - Datenlecks bei den meisten Apps

- 1000 Malware Apps des VirusShare Projects analysiert
 - ca. 16 Sekunden pro App
 - zwei Datenlecks pro App

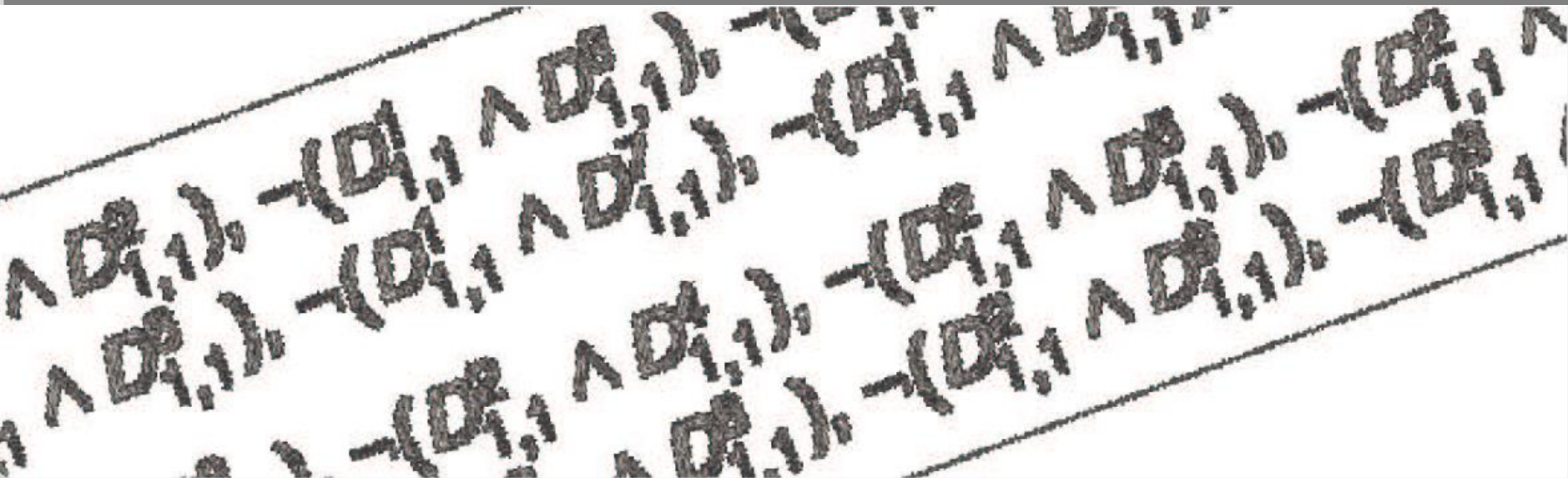
Ausblick

- Analyse von Inter-App-Kommunikation verbessern
- Sanitization einbauen
- Unterscheidung von notwendigen und bösartigen Datenlecks
- **sources** und **sinks** vor Analyse einschränken

Vielen Dank für Ihre Aufmerksamkeit

Fragen?

INSTITUTE FOR THEORETICAL INFORMATICS, KIT



- (1) Christian Fritz, Master thesis: *FlowDroid: A Precise and Scalable Data Flow Analysis for Android*, TU Darmstadt, July 2013. URL: https://ec-spride.sit.fraunhofer.de/fileadmin/user_upload/Group_EC_Spride/FinishedTheses/Masterthesis20130702.pdf
- (2) Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves le Traon, Damien Ochteau and Patrick McDaniel, *FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps*, PLDI 2014. URL: <http://www.bodden.de/pubs/far+14flowdroid.pdf>
- (3) Christian Fritz, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves le Traon, Damien Ochteau and Patrick McDaniel, *Highly Precise Taint Analysis for Android Applications*, EC SPRIDE Technical Report TUD-CS-2013-0113, May 2013. URL: https://www.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_CASED/Publikationen/TUD-CS-2013-0113.pdf
- (4) Yu Feng, Saswat Anand, Alex Aiken, Isil Dillig, *Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis*, FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Pages 576-587
- (5) Davide Balzarotti, Marco Cova, Vika Felmetzger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, Giovanni Vigna, *Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications*, 2008 IEEE Symposium on Security and Privacy, Pages 387 – 401, 18-22 May 2008

Taint Analyse – Sanitization

■ $sani(m, s)$

■ $(m: \text{Parameter}, s: \text{Sanitization Funktion})$

Sanitization

$$SOURCE: \frac{src(m, l), m \rightarrow o, \neg sani(m, s)}{tainted(o, l)}$$

$$TRANSFER: \frac{tainted(o1, l), m \rightarrow o1, n \rightarrow o2, transfer(m, n), \neg sani(m, s)}{tainted(o2, l)}$$

$$SINK: \frac{tainted(o, so), m \rightarrow o, sink(m, si), \neg sani(m, s)}{flow(so, si)}$$

Android

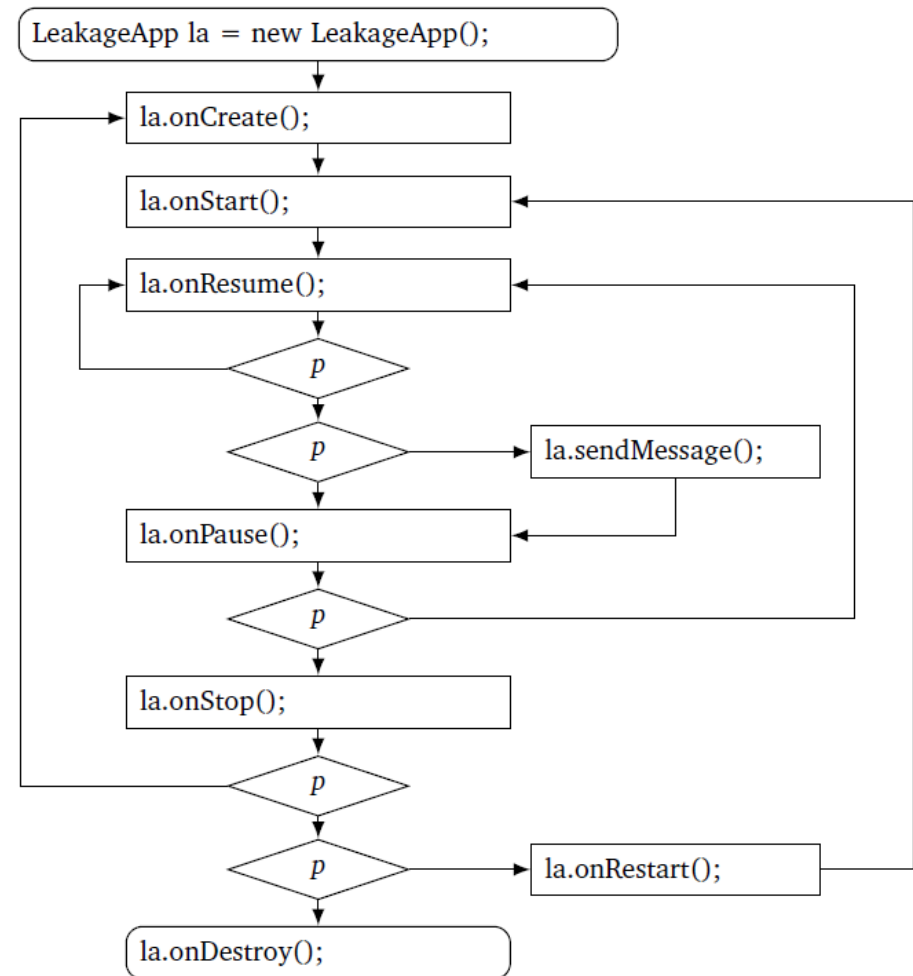
- Zugriff auf Systemfunktionalität muss vom Entwickler bekanntgegeben werden
 - z.B. Nutzung des Standorts
- Benutzer muss zustimmen, wenn er die App installieren möchte
- Oft zu viele Berechtigungen gefordert
- Berechtigungen teilweise zu mächtig
 - Können ausgenutzt werden

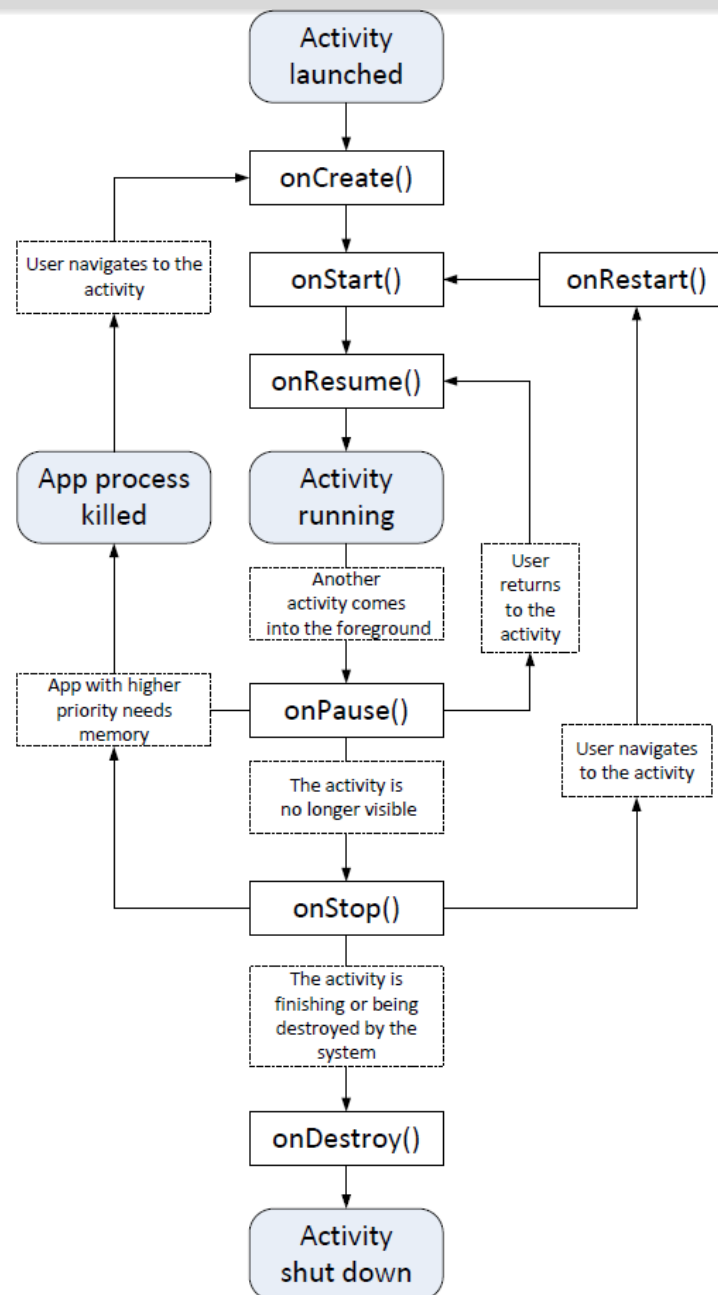
Android – Lebenszyklus

```

1 public class LeakageApp extends Activity {
2     protected void onRestart() {
3         EditText userText = ...
4         EditText pwdTest = ...
5         ...
6     }
7     //Callback Methode. In Layout-XML definiert
8     public void sendMessage(View v) {
9         Password pwdObject = ...
10        String pwd = pwdObject.getPassword();
11        smsManager sms = ...
12        sms.sendTextMessage("0123456789",
13        null, pwd, null, null);
14    }
15 }

```





FlowDroid – SuSi

- vollautomatisches Tool zum Identifizieren von **sinks** und **sources** in Android
- analysiert Android Source Code und generiert Liste von Sources und Sinks
- klassifiziert und kategorisiert

