



# **Semi-automated Discovery of Server- Based Information Oversharing Vulnerabilities in Android Applications**



ISSTA 17

Titik



**1**

---

**Introduction**



# Introduction



## SIFON

Server-based InFormation  
Oversharing

```
tv0.setText("First Name:");  
tv1.setText(p.firstName);  
tv2.setText("Last Name:");  
tv3.setText(p.lastName);
```



## JSON representation

```
{"firstName": "Donald",  
 "lastName": "Knuth",  
 "email": "donald.k@spambox.us"}
```

Listing 1: JSON representation of a user profile.

## Java Model

```
public class Profile {  
    public String firstName;  
    public String lastName;  
    public String email; }
```

## Deserializati on

```
InputStream is = getProfileInputStream();  
Reader r = new InputStreamReader(is);  
Profile p = new Gson().fromJson(r, Profile.class);
```



**2**

---

**System Overview**



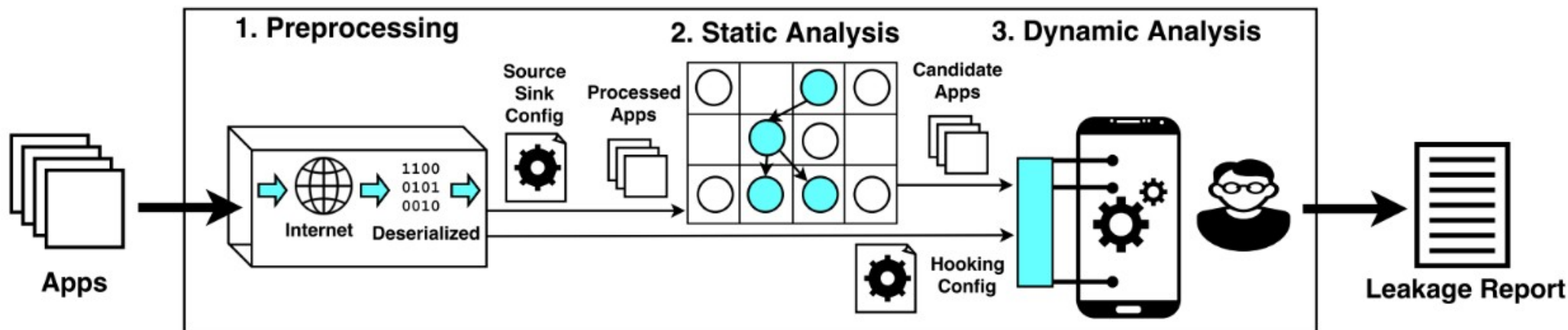
# System Overview

Preprocessing

01

Static Analysis

02



Dynamic Analysis



**3**

**Methodology**

# Preprocessing

**INTERNET permission**

**Serialization library**

Gson

**Obfuscation**

Signature matching





# Static Analysis

## sCFG

flowdroid

## Model Deserialization

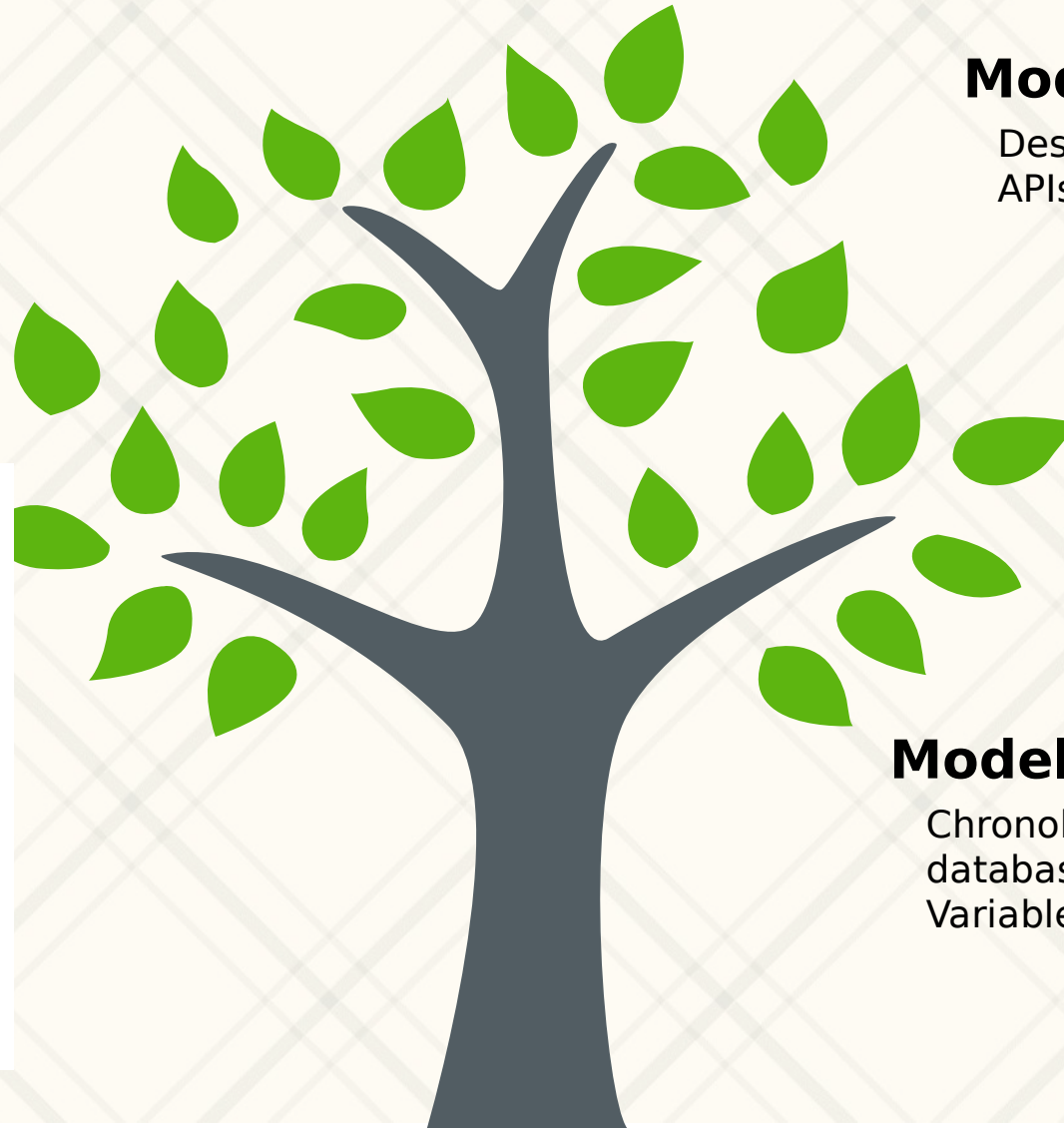
Network API(sources)  
Call sites of deserialization  
libraries(sinks)

## Model Field Identification

Deserialization points(sources)  
APIs that render text in UI (sinks)

## Model Field Classification

Chronology of Data Breaches  
database  
Variable name



```
1 public class A {  
2   public void a() {  
3     is = conn.getInputStream();  
4     new B().b(is);  
5   }  
6 }  
7 public class B {  
8   public void b(InputStream is) {  
9     Profile p = deserialize(is, Profile.class);  
10    new C().c(p);  
11  }  
12  public void b2(Cursor c) {  
13    String s = c.getString(0);  
14    Profile p = deserialize(s, Profile.class);  
15    new C().c(p);  
16  }  
17 }  
18 public class C {  
19   public void c(Profile p) {  
20     tv0.setText("First Name");  
21     tv1.setText(p.firstName);  
22     tv2.setText("Last Name");  
23     tv3.setText(p.lastName);  
24   }  
25 }
```

Diagram illustrating the flow of data and control flow (sCFG) through the code. The flow starts at  $S_0$  (line 3) and proceeds to  $t_0$  (line 9). From  $t_0$ , the flow splits into two paths: one leading to  $t_1$  (line 21) and another leading to  $t_0$  (line 23). The flow then proceeds to  $t_0$  (line 23) and finally to  $t_0$  (line 23). The flow is labeled  $f_0$  and  $f_1$ .



# Dynamic Analysis

## Hooking

Xposed

## Application seeding

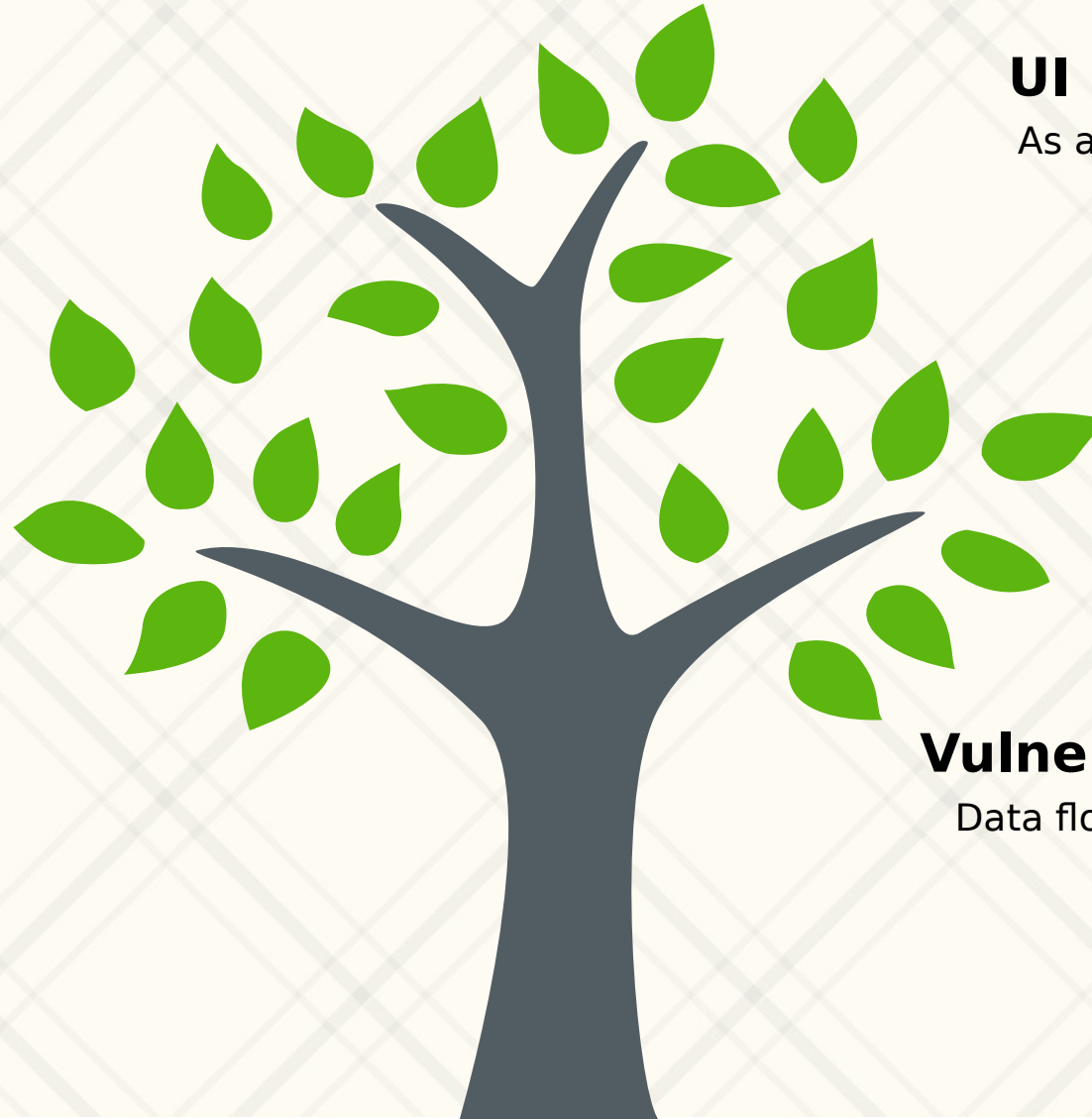
Manually(user account creation)

## UI Exploration

As a real user

## Vulnerability Confirmation

Data flows exist at run time





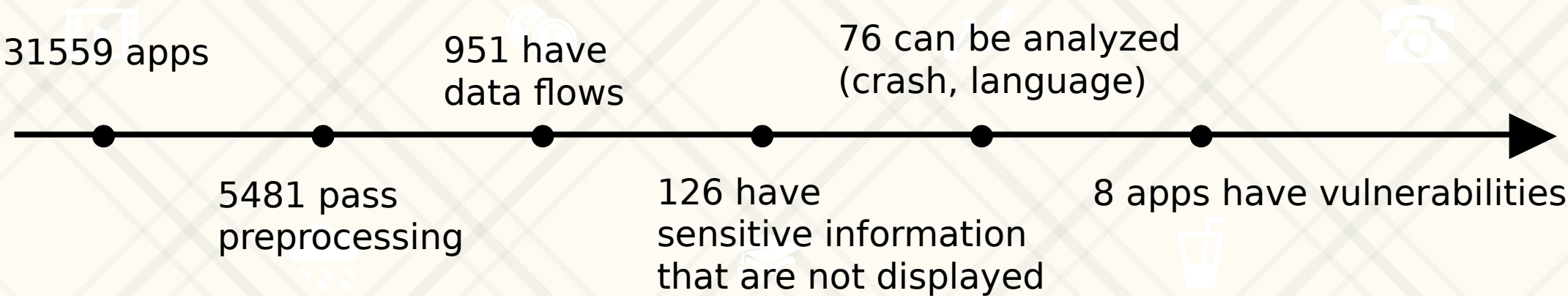
**4**

---

**Evaluation**



单击此处添加标题



Apps	Leaked Data	Number Installs
App A	first and last name, DOB, last action*, ZIP, gender*, user ID*, email, profile status	10,000 - 50,000
App B	email, home page, street, ZIP code, phone number	10,000 - 50,000
App C	Client OS*, email*, friend list*, user ID*, latitude, longitude	10,000 - 50,000
App D	latitude, longitude, last action	5,000 - 10,000
App E	Phone number	1000 - 5000
App F	userRelationID, latitude*, longitude*	500 - 1,000
App G	address, DOB, phone number, email*, deviceOS, Facebook ID, encrypted latitude, longitude, and password	100-500
App H	DOB, hashed password, last login, user type	100 - 500



**5**

---

**Conclusion**