Mobile Security: Vulnerability & Privacy

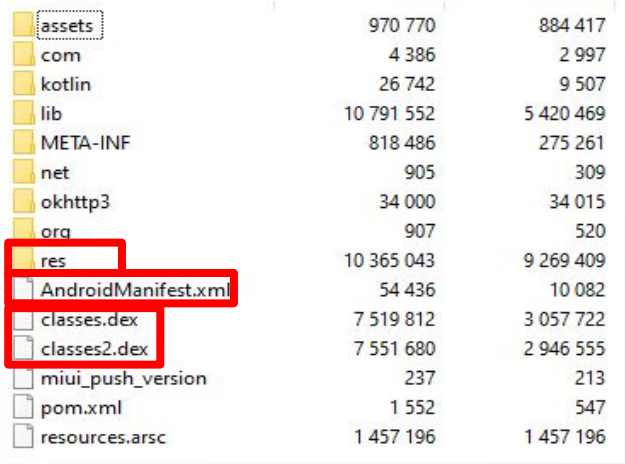
Yue Duan

Outline

- Android Application basics
- Research paper:
  - Android Permissions Demystified
  - Chex: statically vetting android apps for component hijacking vulnerabilities
  - Towards automatic generation of security-centric descriptions for android apps

Android Application Basics

- Android app
  - an APK file (zip file)
  - resource files
  - dex files
    - similar to java class file
  - Android manifest file



assets	970 770	884 417
com	4 386	2 997
kotlin	26 742	9 507
lib	10 791 552	5 420 469
META-INF	818 486	275 261
net	905	309
okhttp3	34 000	34 015
org	907	520
res	10 365 043	9 269 409
AndroidManifest.xml	54 436	10 082
classes.dex	7 519 812	3 057 722
classes2.dex	7 551 680	2 946 555
miui_push_version	237	213
pom.xml	1 552	547
resources.arsc	1 457 196	1 457 196

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.homeandlearn.ken.twoactivities">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="TwoActivities"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"></activity>
    </application>

</manifest>
```

Android Application Basics

- Android app
  - most likely written in Java
  - Java compiled into dex bytecode

```
public boolean offer(E e) {
        checkNotNull(e);
        final ReentrantLock lock = this.lock;
        lock.lock();
        try {
            if (count == items.length)
                return false;
            else {
                enqueue(e);
                return true;
            }
        } finally {
            lock.unlock();
        }
    }
```

```
.METHOD offer : boolean
        .PARAM java.lang.Object
.MODIFIERS public
.REGISTERS 5
.ANNOTATION dalvik.annotation.Signature
        value=["(TE;)Z"]
.CODE
            1190288  invoke-static {v4}, meth@12229
            1190294  iget-object v0, v3 field@4169
            1190298  invoke-virtual {v0}, meth@14543
                      .TRY #0
            1190304  iget v1, v3 field@4166
            1190308  iget-object v2, v3 field@4167
            1190312  array-length v2, v2
                      .CATCH
                              ALL address:1190344
            1190314  if-ne v1, v2, 7
            1190318  const/4 v1, #0
            1190320  invoke-virtual {v0}, meth@14549
            1190326  return v1
                      .TRY #1
            1190328  invoke-direct {v3, v4}, meth@12236
                      .CATCH
                              ALL address:1190344
            1190334  const/4 v1, #1
            1190336  invoke-virtual {v0}, meth@14549
            1190342  goto -8
            1190344  move-exception v1
            1190346  invoke-virtual {v0}, meth@14549
            1190352  throw v1
```

Reverse Engineering

- apktool: reverse engineering tool for Android apps
  - disassemble dex file to smali
  - redirect dex disassembly to Java class files
    - a dex file could contain multiple classes
  - decompile Manifest file and resource files

```
yue@yue-home-ubuntu:~/yueduan/github_projects/AndroidAppNativeLibScanner$ apktool/apktool d apks/DirectLeak1.apk
I: Using Apktool 2.2.0 on DirectLeak1.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/yue/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Reverse Engineering

- Disassembled dex bytecode

```
yue@yue-home-ubuntu:~/yueduan/github_projects/AndroidAppNativeLibScanner/DirectLeak1/smali$
ls de/ecspride/
 BuildConfig.smali   'R$attr.smali'      'R$id.smali'      'R$menu.smali'   'R$string.smali'
 MainActivity.smali  'R$drawable.smali'  'R$layout.smali'   R.smali         'R$style.smali'
```

```
.class public Lde/ecspride/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 8
    invoke-direct {p0}, Landroid/app/Activity;-><init>()V

    return-void
.end method
```

Deeper Analysis

- Soot: A Java analysis framework
  - lift dex/java bytecode into one of its IRs (intermediate representation)
  - generate call graph and control-flow graph
  - analyze def-use chain (for data-flow analysis)
  - perform points-to analysis
  - etc.

Jimple IR

```
public abstract class com.dropbox.core.DbxUploader extends java.lang.Object implements java.io.Closeable
{
    private final com.dropbox.core.http.HttpRequestor$Uploader httpUploader;
    private final com.dropbox.core.stone.StoneSerializer responseSerializer;
    private final com.dropbox.core.stone.StoneSerializer errorSerializer;
    private boolean closed;
    private boolean finished;

    protected void <init>(com.dropbox.core.http.HttpRequestor$Uploa                                    erializer, com.dropbox.core.stone.StoneSerializer)
    {
        com.dropbox.core.DbxUploader r0;
        com.dropbox.core.http.HttpRequestor$U
        com.dropbox.core.stone.StoneSerializ

        r0 := @this: com.dropbox.core.

        r1 := @parameter0: com.dropbox.c

        r2 := @parameter1: com.dropbox.co

        r3 := @parameter2: com.dropbox.core.stone.Stone

        specialinvoke r0.<java.lang.Object: void <init>()>();

        r0.<com.dropbox.core.DbxUploader: com.dropbox.core.http.HttpRequestor$Uploader httpUploader> = r1;

        r0.<com.dropbox.core.DbxUploader: com.dropbox.core.stone.StoneSerializer responseSerializer> = r2;

        r0.<com.dropbox.core.DbxUploader: com.dropbox.core.stone.StoneSerializer errorSerializer> = r3;

        r0.<com.dropbox.core.DbxUploader: boolean closed> = 0;

        r0.<com.dropbox.core.DbxUploader: boolean finished> = 0;

        return;
    }
```

Not exactly SSA format

Permission system

- Each app explicitly declares a set of permissions in the Manifest file

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

- Before installation
  - Android system asks users to approve
  - one-time effort to grant permissions with some exceptions
    - location, camera, ...

Android Permissions Demystified

Andrienne Felt, Erika Chin, Steve Hanna, Dawn Song, David Wagner

UC Berkeley

CCS 2011

Overview

- More of an empirical study paper for:
  - Android permission system
  - What permissions do Andriod’s API methods require?
  - Do Android apps request more permissions than necessary

Permission Testing

- Goal
  - Correlate each API method with permissions
  - For example,
    - getLastKnownLocation() API ⇐⇒ Manifest.permission.ACCESS_COARSE_LOCATION or Manifest.permission.ACCESS_FINE_LOCATION

- Can’t you find the info in Android documentation?
  - Sort of, but incomplete

Permission Testing

- Methodology
  - dynamic analysis
  - modify Android 2.2 system to log permission checks
  - invoke each API to observe
- Challenges
  - MANY Android's API methods
  - some maybe private and hidden
  - permissions may depend on argument and ordering of methods calls

Permission Testing

- Permission map results
  - Android API 2.2 consists of
    - 1,665 classes with a total of 16.732 methods
    - 85% coverage for the testing
      - uncovered ones: native calls, classes do not require permissions
  - Discovered 1,259 methods with permission checks
    - Android 2.2 documentation: only 78 methods
    - discovered 6 inconsistencies

Application Permission Analysis

- Goal: discover the potential discrepancy between
  - permissions that android apps actually need
  - permissions declared

Application Permission Analysis

- Methodology
  - static analysis
  - analyze Java bytecode to discover what API methods an app invokes
  - together with previous findings, infer the permissions needed
  - check against permissions declared

Application Permission Analysis

- Analysis results
  - 900 Android apps for automated analysis
  - identified 323 of 900 (35.8%) as having over-privilege issue

Permission	Usage
ACCESS_NETWORK_STATE	16%
READ_PHONE_STATE	13%
ACCESS_WIFI_STATE	8%
WRITE_EXTERNAL_STORAGE	7%
CALL_PHONE	6%
ACCESS_COARSE_LOCATION	6%
CAMERA	6%
WRITE_SETTINGS	5%
ACCESS_MOCK_LOCATION	5%
GET_TASKS	5%

Table 2: The 10 most common unnecessary permissions and the percentage of overprivileged applications that request them.

Application Permission Analysis

- Reasons for over-privilege issues
  - confusing permission names
  - documentation errors
  - etc

Chex: statically vetting android apps for component hijacking vulnerabilities

Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, Guofei Jiang

CCS 2012

Chex

- Goal: vetting vulnerable apps in large scale

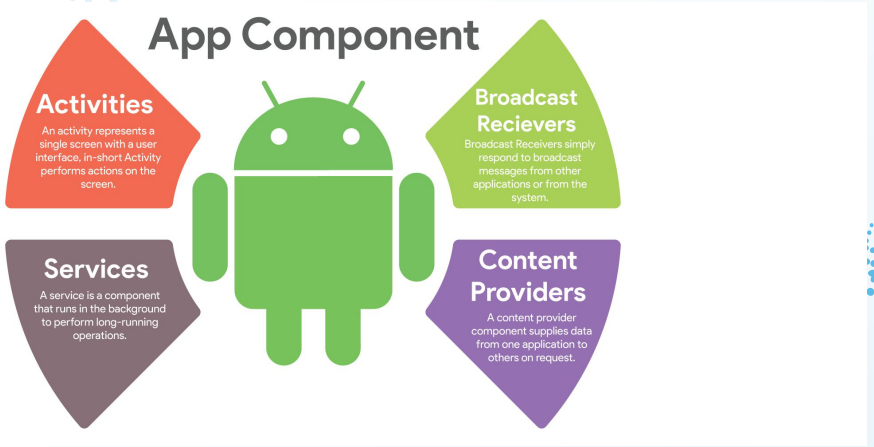
Accurate
- Deep inspection
- Generic coverage

Fast
- Static analysis
- No de-compilation

App market model
- No source code required
- No human assistance

Component Hijacking Recap

- Android application components
  - activity
  - service
  - content provider
  - broadcast receiver

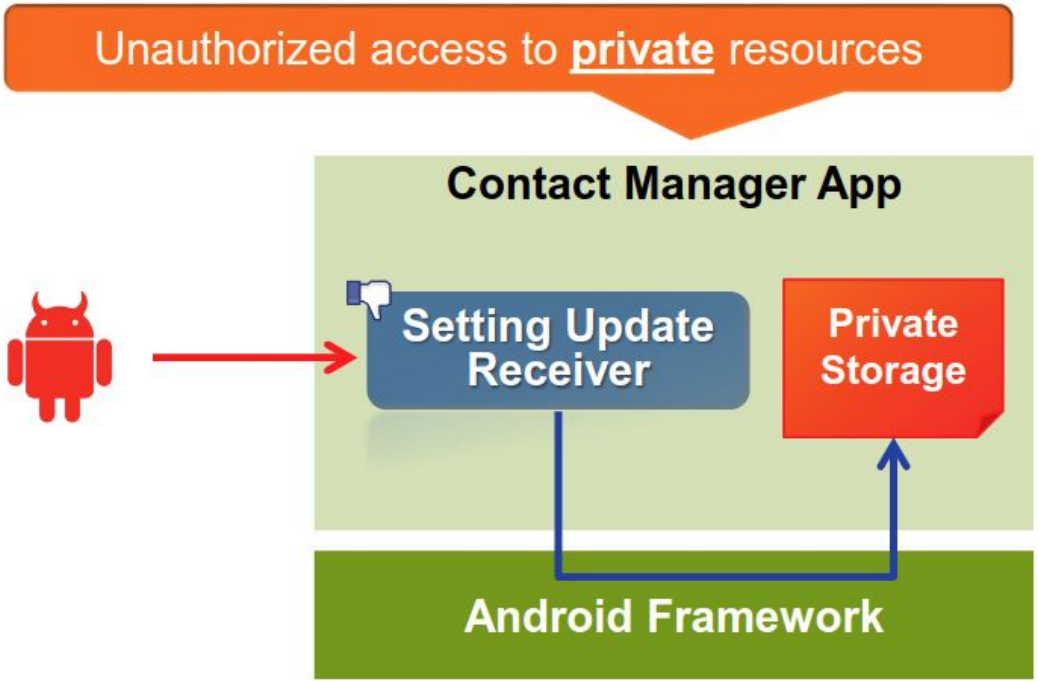


Component Hijacking Recap

- export components
  - publicly available
  - can be launched by other components from a different app
  - lead to accidentally shared permissions

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools">

    <application tools:ignore="GoogleAppIndexingWarning">

        <activity
                android:name="com.samples.medium.SecondaryActivity"
                android:exported="true"
                android:screenOrientation="sensorLandscape"/>

    </application>

</manifest>
```

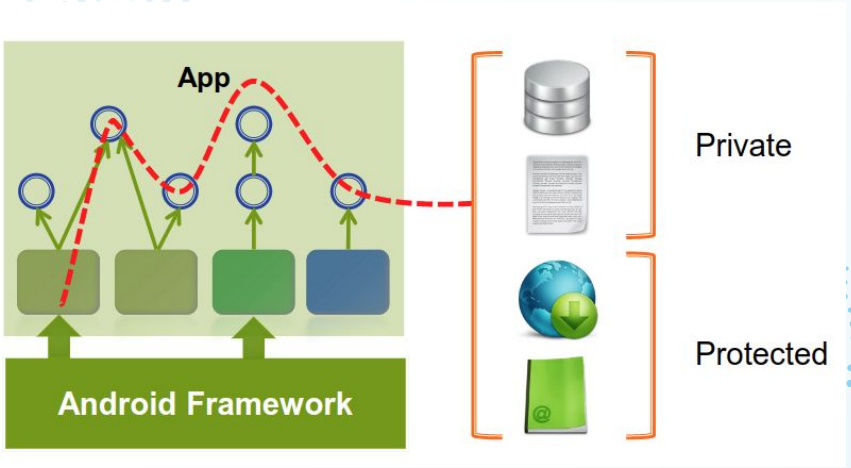
Component Hijacking Recap



Setting Update Receiver	Overwrites sensitive data upon update
	Accepts external updates

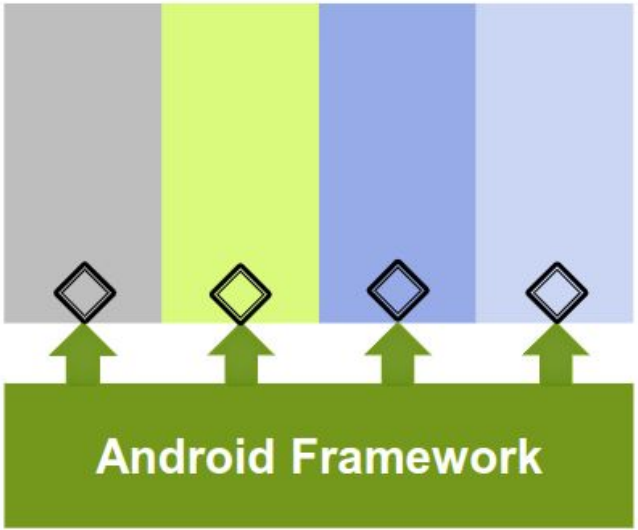
CHEX Analysis Approach

- A data-flow perspective
  - Component hijacking ⇒
    read/write protected or private data via exported components
  - detecting component hijacking ⇒ finding “hijack-enabling flows”



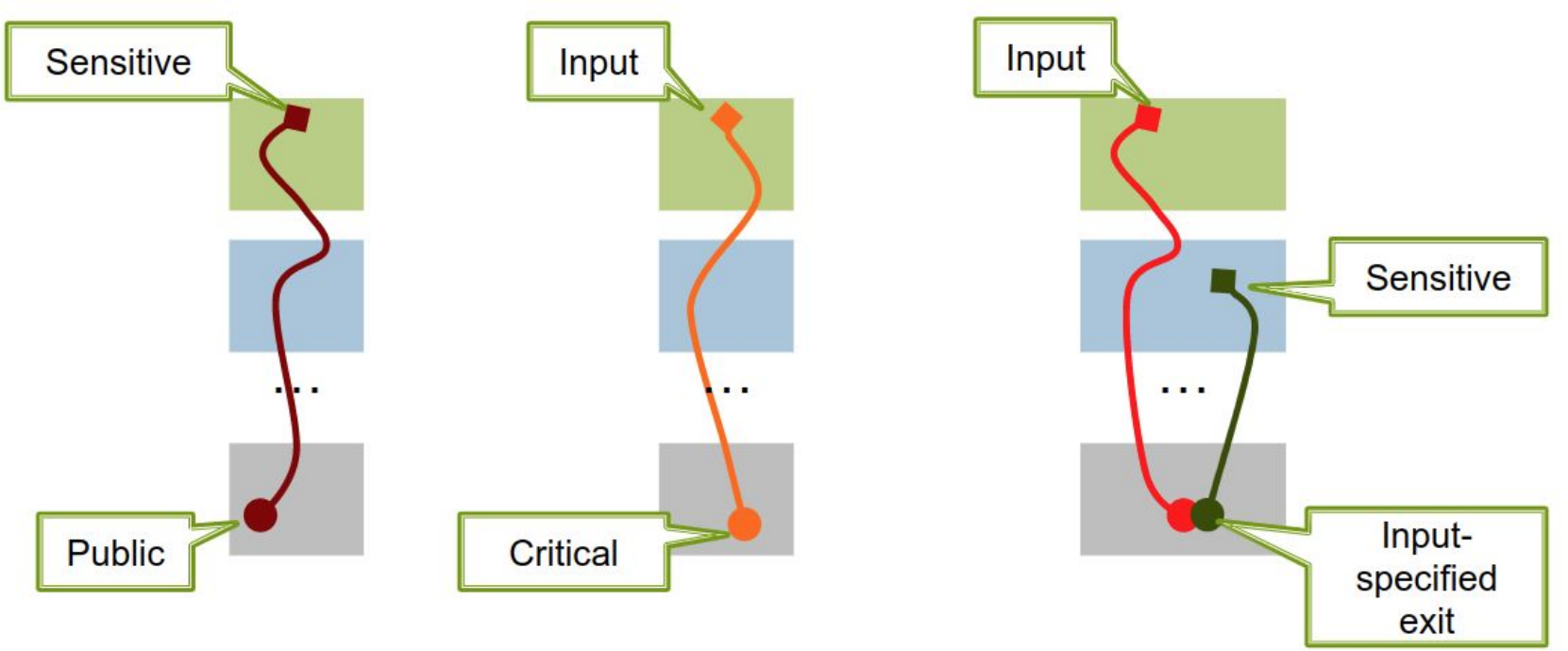
App splitting

- An Android app has multiple entry points
  - defined by the app
  - intended to be called only by the framework
- A split is a subset of the app code that is reachable from an entry point.



Identifying Potential Issues

- link splits together to find all possible data-flows



Evaluation

- 5,486 apps are examined
  - performance
    - median processing time: 37s
    - 22% apps took > 5 min
  - accuracy
    - 254 apps flagged as vulnerability
    - TPR: 81%
  - Insights
    - 50 entry points of 44 types per app
    - 99.7 apps contain inter-split data-flows

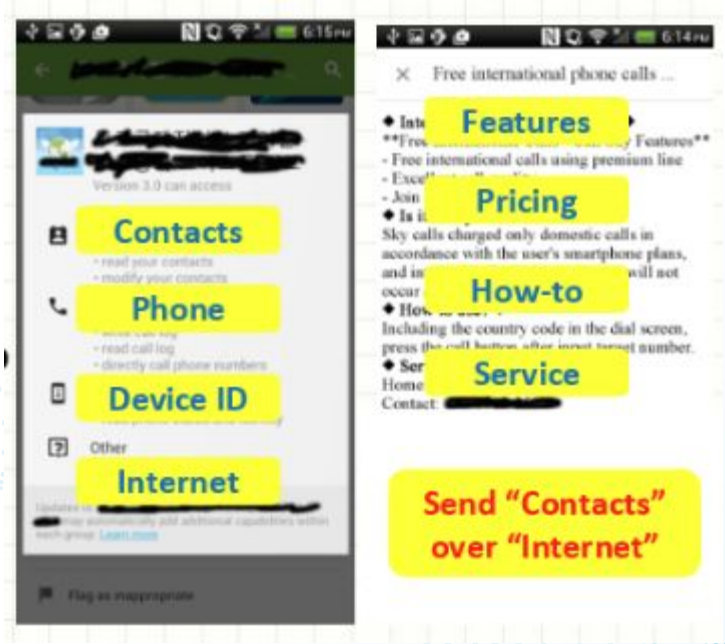
Towards automatic generation of security-centric descriptions for android apps

Mu Zhang, Yue Duan, Qian Feng, Heng Yin

CCS 2015

Motivation

Permissions:
hard to read
insufficient
to tell ‘how’

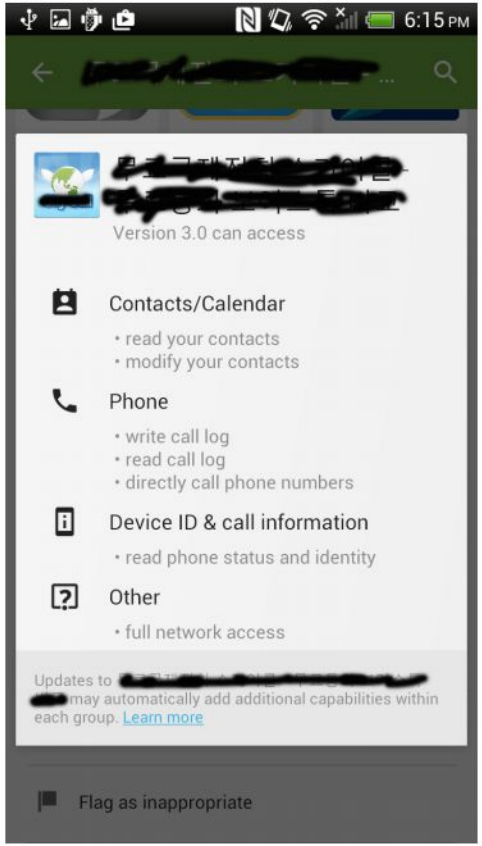


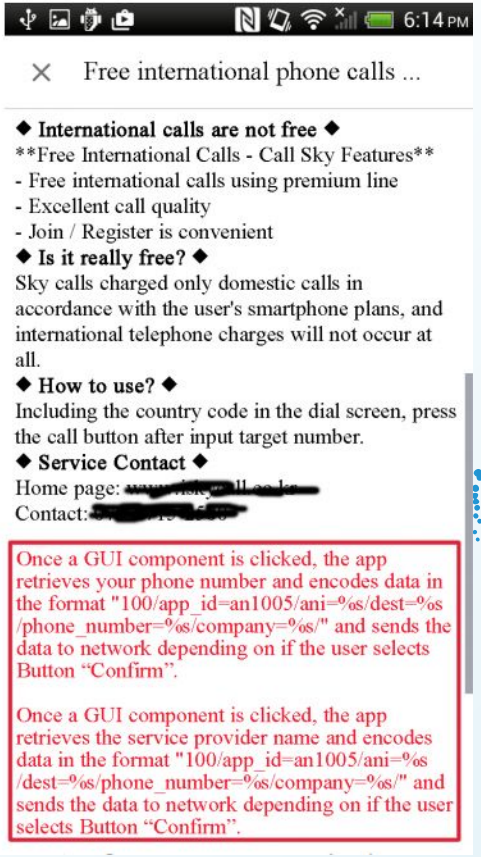
Textual description:
not really about security

What an app claims to do **VS.** what the app actually does

Approach

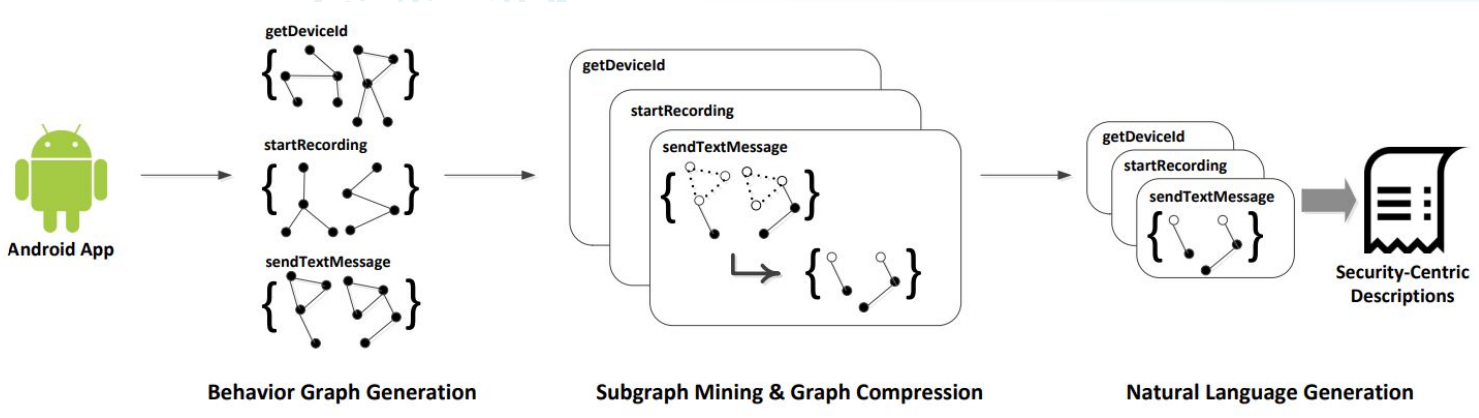
Automatically generate **security-centric** descriptions for apps



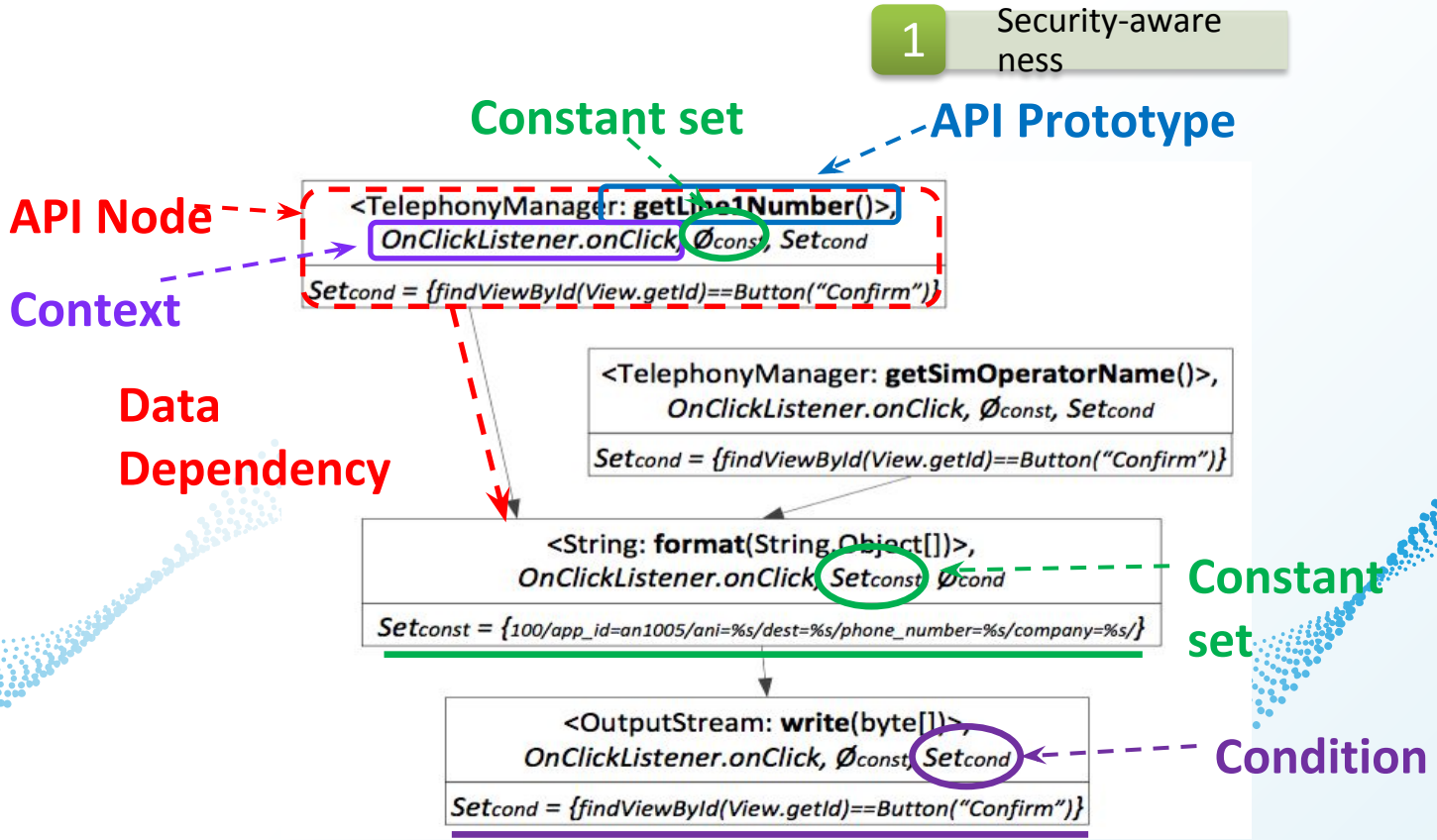


Approach

- Methodology
  - static analysis for
    - data-flow analysis
    - condition analysis
  - data mining for graph compression
  - natural language generation

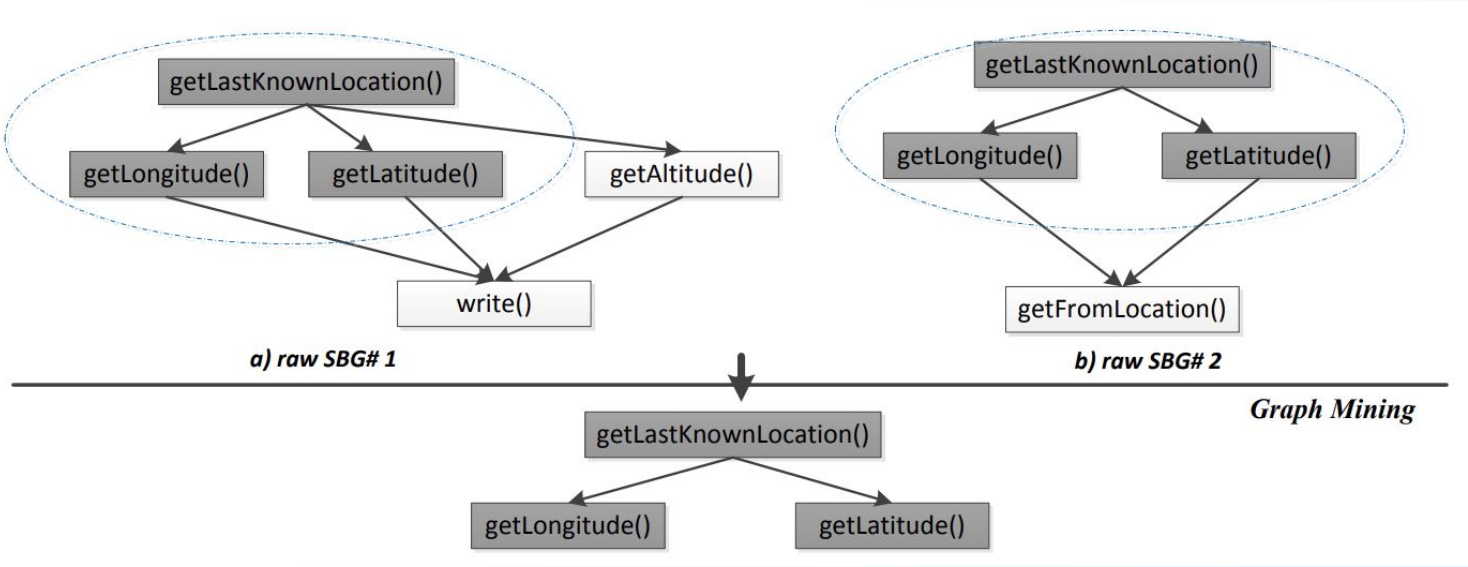


Behavior Graph

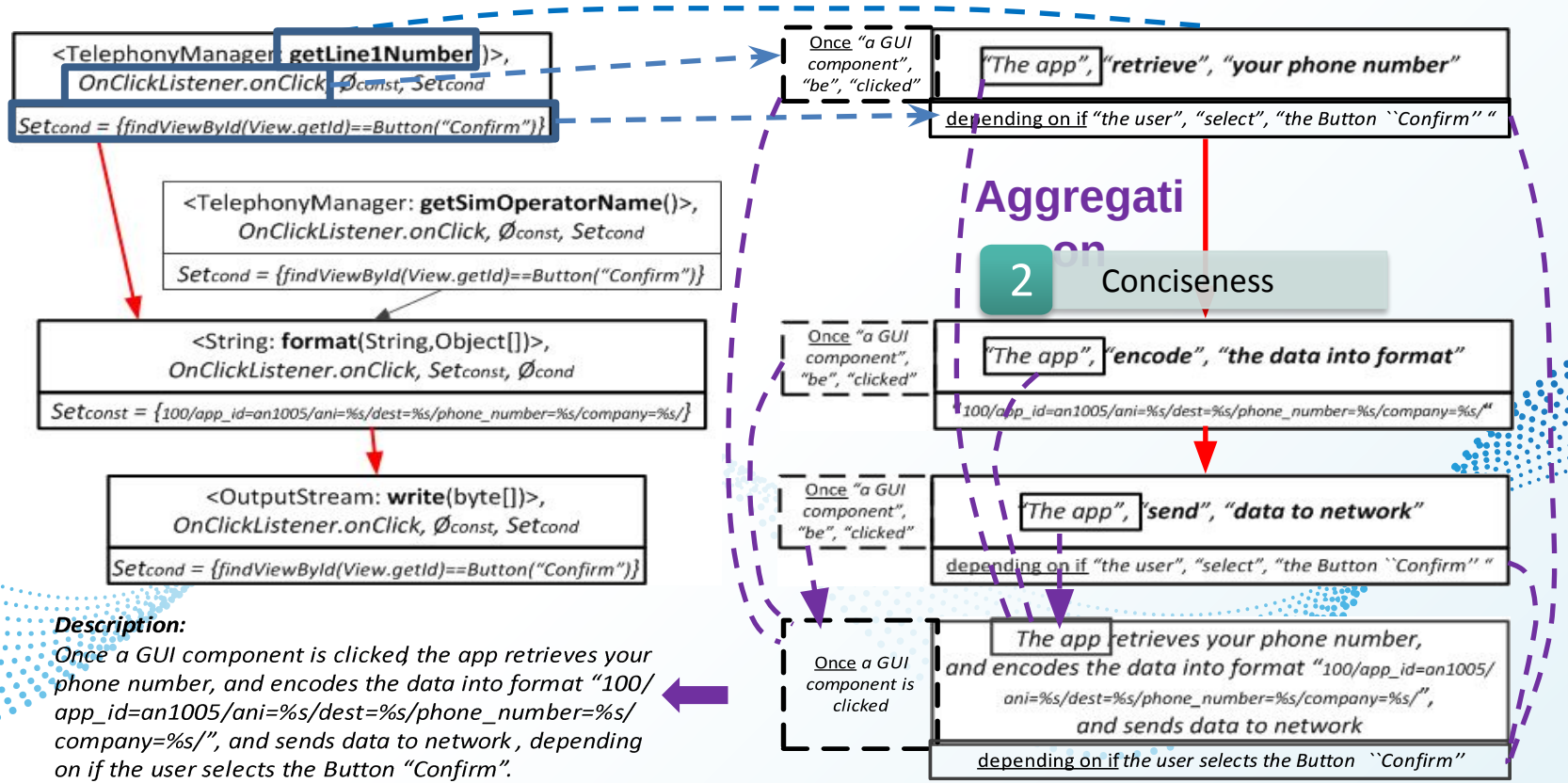


Subgraph Mining

- Discovered 109 patterns
- Graph Compression: replace the subgraphs with single nodes



Natural Language Generation

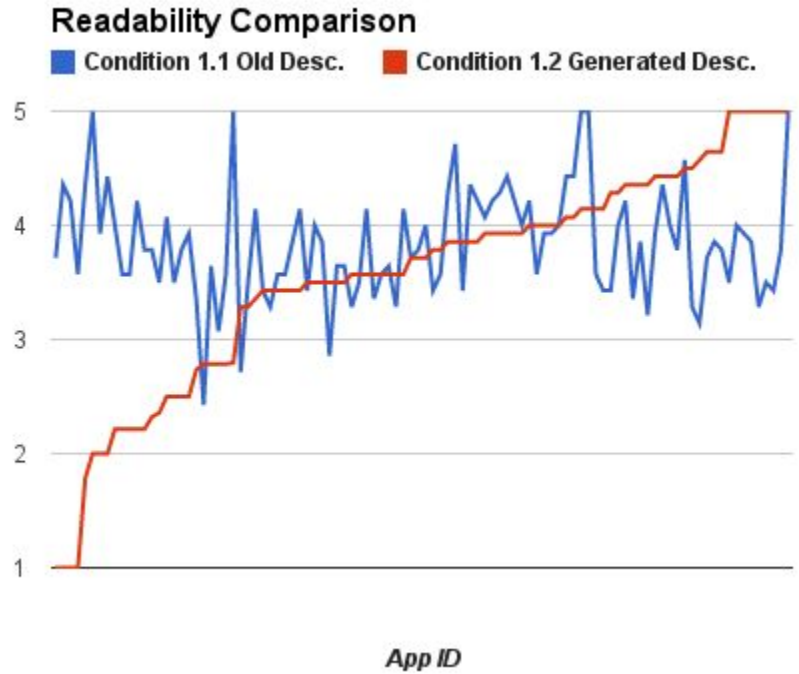


Description:

Once a GUI component is clicked, the app retrieves your phone number, and encodes the data into format "100/app_id=an1005/ani=%s/dest=%s/phone_number=%s/company=%s/", and sends data to network, depending on if the user selects the Button "Confirm".

Evaluation

- Can average users read the machine generated descriptions?



Evaluation

- Can our descriptions help users avoid risks?

#	Condition	ADR
2.1	Malware w/ old desc.	63.4%
2.2	Leakage w/ old desc.	80.0%
2.3	Clean w/ old desc.	71.1%
2.4	Malware w/ new desc.	24.7%
2.5	Leakage w/ new desc.	28.2%
2.6	Clean w/ new desc.	59.3%

Summary

- Android Application basics
- Research paper:
  - Android Permissions Demystified
    - over-privilege issues
  - Chex: statically vetting android apps for component hijacking vulnerabilities
    - component hijacking vulnerability
  - Towards automatic generation of security-centric descriptions for android apps
    - help users better understand the security of an app

Thank you!

Questions?