Privacy

Project 4

Static Program Analysis to identify privacy leaks in Android applications using FlowDroid

Submitted by : Prashanthi Kanniappan Murthy

Dataset used :

playdrone-apk-c7

Question 1 : High Level Statistics

The dataset was chosen at random from the Playdrone dataset available. A Python script was written which chose the .apk files based on 2 criteria.

1) The time taken for running FlowDroid against these .apk files was not more than 15 mins (to decrease the runtime of the script)
2) There was some leakage in the application (to have more data for analysis, here leaks>0)

Experiences on running FlowDroid:

The following python script had methods to run FlowDroid, calculate the number of leaks, conditions applied as mentioned above. It kills the process thats running FlowDroid if the time taken was more than 15 mins and deletes the corresponding output file that is being saved for further analysis. The script also deletes the output file of apks that has no leaks.

Python Script for App selection and running FlowDroid :

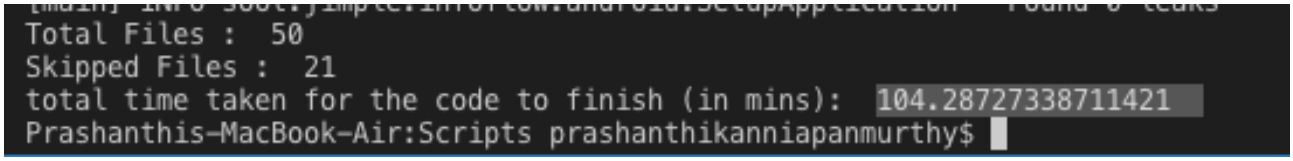
```
from subprocess import Popen, PIPE, check_output
import os, fnmatch
import time
import xml.etree.ElementTree as ET

def runFlowDroid():
     """
        Method runs FlowDroid as per given instructions. Commands are
wired inside the method. The apk files (entire set of 3k apps) are
downloaded already and are available in the folder as mentioned in the
commands. An improvement and future work could be making REST calls to
get the .apk s dynamically from the dataset site.
     Code has security issues while using inbuilt ElementTree and
subprocess (shell = True)[sending a malicious modified xml in
ElementTree, shell injection attacks]
        !!! DO NOT USE THIS CODE OUTSIDE YOUR LAPTOP !!! or if you dont
have control over your xml files.
           input: Takes no argument
           returns a list containing 2 values - [ files processed ,
number of files skipped ]
    """
```

```
    count = 0
    skipped = 0
    listOfFiles = os.listdir('/Users/prashanthikanniapanmurthy/Desktop/
Fall18/CNS/proj4/testAPIS/playdrone-apk-c7/')  #playdrone-apk-c7/
    pattern = "*.apk"
    cmd1 = "java -Xms2048m -jar soot-infoflow-cmd/target/soot-infoflow-
cmd-jar-with-dependencies.jar -a /Users/prashanthikanniapanmurthy/
Desktop/Fall18/CNS/proj4/testAPIS/playdrone-apk-c7/" #playdrone-apk-c7/
    cmd2 = " -p /Users/prashanthikanniapanmurthy/Library/Android/sdk/
platforms  -s ./soot-infoflow-android/SourcesAndSinks.txt -out "
    for entry in listOfFiles:
            if fnmatch.fnmatch(entry, pattern) and count < 50:
                print("Running .....", end=" ")
                print (entry)
                os.chdir("/Users/prashanthikanniapanmurthy/Desktop/
Fall18/CNS/proj4/FlowDroid")
                output_file = "/Users/prashanthikanniapanmurthy/Desktop/
Fall18/CNS/proj4/OutFiles/"+entry+"_out.xml"
                extensions = "" # for adding any other options for
performance
                cmd = cmd1+entry+cmd2+output_file+extensions
                rm_cmd = "rm "+output_file
                print (cmd)
                try :
                    output = check_output(cmd, timeout=(60*15),
shell=True) #Timing out after 15 mintutes
                except :
                    print ("Timed out ... Moving on to next file ")
                    check_output(rm_cmd, shell=True)
                    skipped+=1
                    continue
                if ( leaksinApps(output_file) == 0 ):
                    check_output(rm_cmd, shell=True)
                    skipped +=1
                count +=1
    return [count,skipped]

def leaksinApps(xml):
    """
    Method returns the leakages in the apk file. Counts the number of
sink tags in the parsed xml

    input: takes the xml file as an argument
    returns the number of leakages
    """
    tree = ET.parse(xml)
    root = tree.getroot()
    leaks = 0
    for Sink in root.iter('Sink'):
        leaks+=1
    return leaks

start_time = time.time()
output_list = runFlowDroid()
print ("Total Files : ",output_list[0])
```

```
print ("Skipped Files : ",output_list[1])
print ("total time taken for the code to finish (in mins): ",
(time.time() - start_time)/60)
```

List of first 50 apks that ran against the conditions and were included in the analysis are:
Out of the 50 apks that ran against the code :
21 had no leaks and it took approximately 1 hour 45 mins for this. So I stopped in here, as 50 applications, all with leaks > 0 (as I had my conditions in the code) could take up a very long time.

Screenshot of the code results:



List of .apk Files Chosen	Leaks in each
com.linsar.smartcenter-405000088.apk	7
com.robtheis.android.phrasebook.pt.md-7.apk	0
com.appbuilder.u684733p1138879-8.apk	15
com.vikingpanzer.horsesbesthdlivewallpaper-11.apk	0
com.LovePicturesNewPhotoFrames-3.apk	6
com.risingstorm.jack32xfanclub-41.apk	0
com.jb.gokeyboard.theme.kustomix.newyork-22.apk	4
com.jothisofttech.southgate-1.apk	2
com.blackboard.mosaic.wpi-2.apk	5
com.gtu_tw-2.apk	0
com.etgames.SubmarineBobAdventure-1000001.apk	4
com.SecretDoc-1.apk	0
kr.co.wbg.clubone-5.apk	0
com.whitespace.standews-5.apk	0
com.moedev.DodgeAR-5.apk	7
com.threepoundhealth.euco-11.apk	0

List of .apk Files Chosen	Leaks in each
com.kdfod159.kwhap240-1.apk	16
com.geniusfootballplayer.android.danielederossiskills-1.apk	0
com.kedros.basicfluidmech-6.apk	0
com.FPSyndicate.TapDot-8.apk	5
air.com.GoldCupGames.KitchenCatBeanCornSideDishMexican-1352000.apk	0
org.cardinal.rstart-5.apk	3
com.spanishgamechangerlitenew-8.apk	0
com.appfondue.dailycleaningtips-1.apk	2
pl.przepisywanie.android-41.apk	8
com.projecthelp.wwwwrappercc-1.apk	0
edu.neu.madcourse.juhiparalkar-11.apk	2
com.Company.TrainSpace_RFV95-3.apk	0
com.lightsleepers.bamboo-5.apk	6
number.puzzle.number.puzzle-13.apk	12
com.hitgpx.memelulz-1.apk	13
com.jaludo.wyspagier-18.apk	5
com.Black_Side.Slender_Man_Dark_Town-1.apk	5
dk.aau.cs.giraf.tortoise-1.apk	4
bong.android.hanmain-7.apk	0
com.dlaor48345.freewifi-1.apk	13
com.sappsuma.salonapps.beautyatthemanor-302.apk	15
blak3n.sltheme.metal-1.apk	0
com.newappempire.evilmonkeyempire-1.apk	0
com.ZaiqStudio.AirBattle-1009002.apk	6
com.app_bipc.layout-399.apk	2
fat.burning.exercises-7.apk	3
com.offline.ocr.english.image.to.text-8.apk	1
com.conduit.app_8d435ca0cfe342d6b4b12d7920db3edb.app-4.apk	2
ie.kalibra.apps.android.supermacs-2.apk	0

List of .apk Files Chosen	Leaks in each
it.pgmobapp.app_1405978421277_0uJ6w7-2.apk	1
com.myexample.kcse_kenya_math-4.apk	0
sandhills.hosteddealerapp.carringtonequipment-3.apk	2
com.monigot.lantern_ag-30.apk	0
com.mar3h.lao.laokeyboard-4.apk	0

After getting the output XML from FlowDroid, another Python Script parses it and creates 2 log files - Sources , Sinks with the application name from which the paths were identified. These log files are then onboarded to splunk for further analysis and visualisations. Pipelines are used as delimiters in the script to make it simpler for Splunk analysis.

Python script for parsing the XML from FlowDroid output:

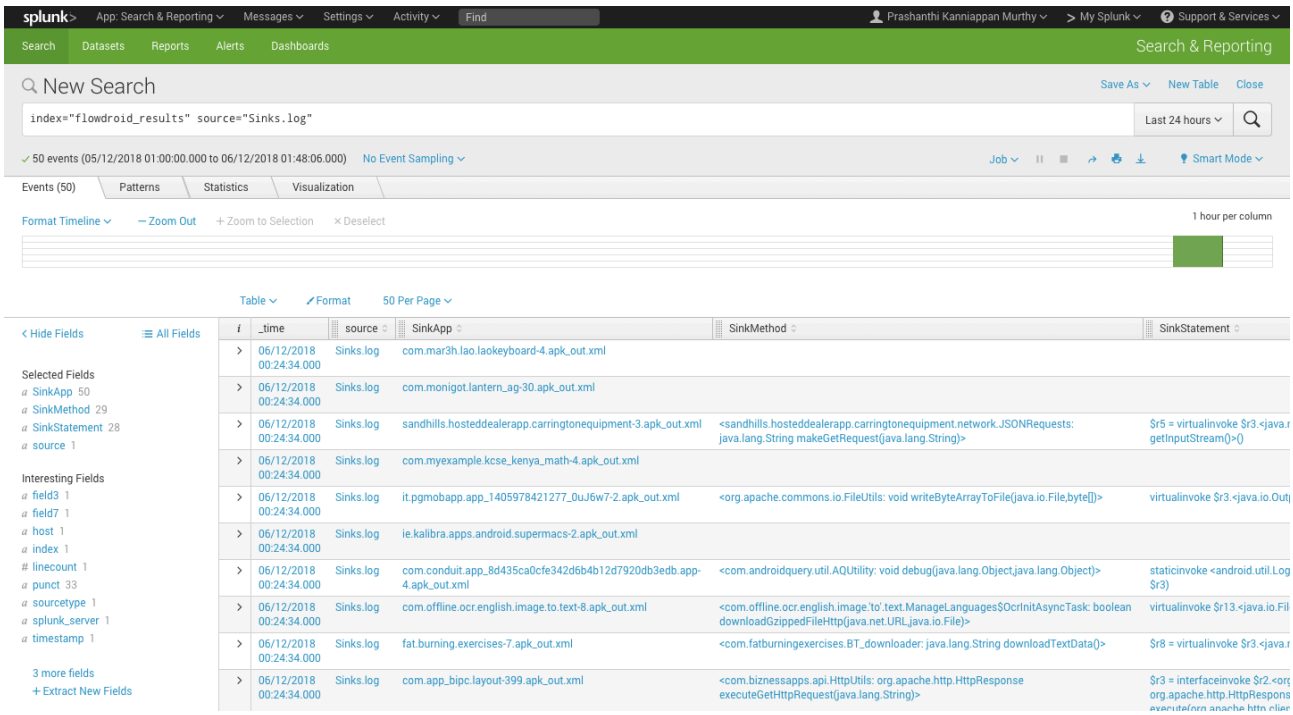
```
import xml.etree.ElementTree as ET
from subprocess import Popen, PIPE, check_output
import os, fnmatch

def parseXML(xmlfile,App):
    tree = ET.parse(xmlfile)
    root = tree.getroot()
    leaks = 0
    sources = App
    sinks = App
    sink = ''
    source = ''

    for ele in root.iter('Sink'):
        leaks+=1
        for key,val in ele.attrib.items():
            if key == 'Statement':
                sink = sink + '||Sink Statement||' + val
            if key == 'Method':
                sink = sink + '||Sink Method:||'+val

    for ele in root.iter('Source'):
        for key,val in ele.attrib.items():
            if key == 'Statement':
                source = source + '||Source Statement:||'+val
            if key == 'Method':
                source = source + '||Source method:||'+ val

    sinks = sinks + sink + "\n"
    sources = sources + source + "\n"

    with open("Sinks.log", "a") as sinkfile:
        sinkfile.write(sinks)
```

```
    with open("Sources.log", "a") as sourcefile:
        sourcefile.write(sources)

count = 0
listOfFiles = os.listdir('/Users/prashanthikanniapanmurthy/Desktop/
Fall18/CNS/proj4/Outfiles/')
pattern = "*.xml"
for entry in listOfFiles:
    if fnmatch.fnmatch(entry, pattern) :
        count+=1
        print ("Printing ..... ",entry)
        xmlfile = "/Users/prashanthikanniapanmurthy/Desktop/Fall18/CNS/
proj4/Outfiles/" +entry
        parseXML(xmlfile,str(entry))
        print ("done parsing : ",entry)
```

After Onboarding to Splunk, following analysis are done to get a high level statistics.

Following is the screenshot of how data looks like on Splunk.
Contains the follwing fields:

1) Application Name
2) Sink Method
3) Sink Statement
4) Source Method
5) Source Statement



Semantic grouping of taint sources:

Taint Source Group	Splunk Query Used	Applications
Location	index="FlowDroid_results" source="Sources.log" location	com.Black_Side.Slender_Man_Dark_Town-1.apk
		com.FPSyndicate.TapDot-8.apk
		com.app_bipc.layout-399.apk
		com.appbuilder.u684733p1138879-8.apk
		com.blackboard.mosaic.wpi-2.apk
		com.dlaor48345.freewifi-1.apk
		com.kdfod159.kwhap240-1.apk
		com.sappsuma.salonapps.beautyatthemanor-302.apk
		fat.burning.exercises-7.apk
		org.cardinal.rstart-5.apk
DeviceId	index="FlowDroid_results" source="Sources.log" *device*	com.appfondue.dailycleaningtips-1.apk
		com.blackboard.mosaic.wpi-2.apk
		com.jb.gokeyboard.theme.kustomix.newyork-22.apk
		fat.burning.exercises-7.apk
		org.cardinal.rstart-5.apk
Country	index="FlowDroid_results" source="Sources.log" *country*	com.ZaiqStudio.AirBattle-1009002.apk
		com.appbuilder.u684733p1138879-8.apk
		com.etgames.SubmarineBobAdventure-1000001.apk
		number.puzzle.number.puzzle-13.apk
Telephony Services	index="FlowDroid_results" sourcetype=sources_FlowDroid telephony	com.appfondue.dailycleaningtips-1.apk
		com.blackboard.mosaic.wpi-2.apk

Taint Source Group	Splunk Query Used	Applications
		com.jb.gokeyboard.theme.kustomix.newyork-22.apk
		edu.neu.madcourse.juhiparalkar-11.apk
		fat.burning.exercises-7.apk
		org.cardinal.rstart-5.apk
Storage Access		com.app_bipc.layout-399.apk
		com.blackboard.mosaic.wpi-2.apk

Grouping of Taint Sinks:

Taint Sink	Splunk Query	Applications
Logging (using android.util.log)	index="FlowDroid_results" source="Sinks.log" log	com.Black_Side.Slender_Man_Dark_Town-1.apk
		com.FPSyndicate.TapDot-8.apk
		com.LovePicturesNewPhotoFrames-3.apk
		com.ZaiqStudio.AirBattle-1009002.apk
		com.appbuilder.u684733p1138879-8.apk
		com.appfondue.dailycleaningtips-1.apk
		com.blackboard.mosaic.wpi-2.apk
		com.conduit.app_8d435ca0cfe342d6b4b12d7920db3edb.app-4.apk
		com.dlaor48345.freewifi-1.apk
		com.etgames.SubmarineBobAdventure-1000001.apk
Writing onto some File (using File Util)	index="FlowDroid_results" source="Sinks.log" file	com.LovePicturesNewPhotoFrames-3.apk
		com.offline.ocr.english.image.to.text-8.apk
		it.pgmobapp.app_1405978421277_0uJ6w7-2.apk

Taint Sink	Splunk Query	Applications
Network Sinks	index="FlowDroid_results" source="Sinks.log" SinkStatement="*" *HTTP* OR *net*	com.app_bipc.layout-399.apk
		com.appbuilder.u684733p1138879-8.apk
		com.appfondue.dailycleaningtips-1.apk
		com.dlaor48345.freewifi-1.apk
		com.hitgpx.memelulz-1.apk
		com.jaludo.wyspagier-18.apk
		com.kdfod159.kwhap240-1.apk
		com.lightsleepers.bamboo-5.apk
		com.linsar.smartcenter-405000088.apk
		com.moedev.DodgeAR-5.apk

Using Splunk made it easier to go over these parsed FlowDroid outputs. Overall observation is that Device ID and getting location access are considered as most serious privacy concerns. It even gets bad when these private data is passed over the network or written into logs unknowingly or without prior permission from the user.We could also see some rare cases of private sources like mobile number of the device registered, storage access etc. A deeper dive into the apk files can give a better solution to these leakages.

Question 2 : Determining Privacy Violations

For analysis of section the following 5 Android applications which had network sinks are used:

1) fat.burning.exercises-7.apk with 3 leaks from FlowDroid
2) com.appfondue.dailycleaningtips-1.apk with 2 leaks from FlowDroid
3) com.app_bipc.layout-399.apk with 2 leaks from FlowDroid
4) com.blackboard.mosaic.wpi-2.apk with 5 leaks from FlowDroid
5) number.puzzle.number.puzzle-13.apk with 12 leaks from FlowDroid

1) fat.burning.exercises-7.apk

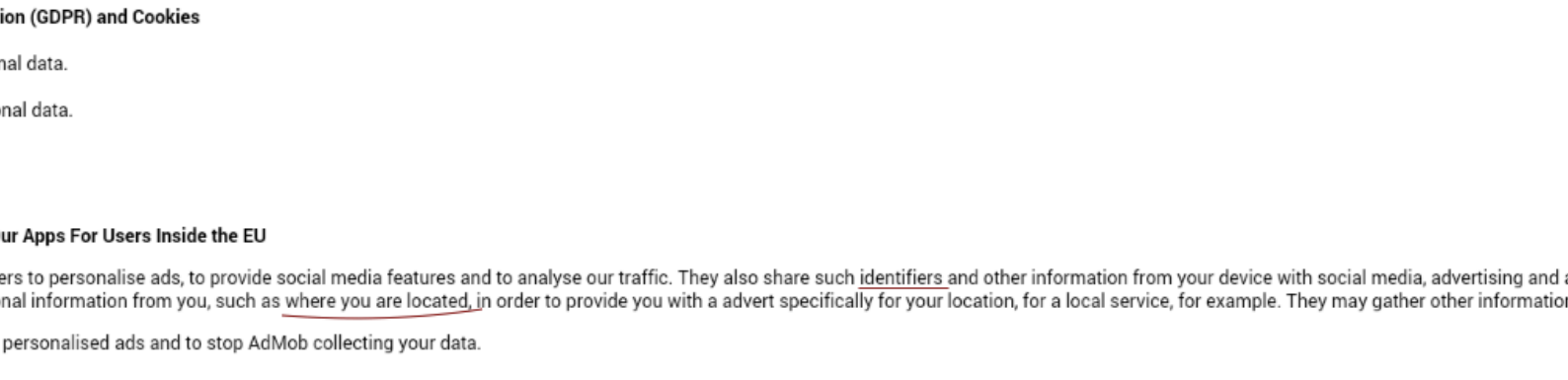
There were 3 leaks for 'fat.burning.exercises-7.apk' from FlowDroid.
Splunk entry :

i	Time	Event
>	06/12/2018 04:14:20.000	fat.burning.exercises-7.apk_out.xml||Source Statement:||$d0 = virtualinvoke $r1.<android.location.Location: double getLatitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void onLocationChanged(android.location.Location)>\|\|Source Statement:\|\|$r5 = virtualinvoke $r14.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.fatburningexercises.BT_device: void <init>()>\|\|Source Statement:\|\|$d0 = virtualinvoke $r4.<android.location.Location: double getLatitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void getLastLocation()>\|\|Source Statement:\|\|$d0 = virtualinvoke $r1.<android.location.Location: double getLatitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void onLocationChanged(android.location.Location)>\|\|Source Statement:\|\|$d0 = virtualinvoke $r1.<android.location.Location: double getLongitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void onLocationChanged(android.location.Location)>\|\|Source Statement:\|\|$d0 = virtualinvoke $r4.<android.location.Location: double getLongitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void getLastLocation()>\|\|Source Statement:\|\|$r5 = virtualinvoke $r14.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.fatburningexercises.BT_device: void <init>()>\|\|Source Statement:\|\|$d0 = virtualinvoke $r4.<android.location.Location: double getLatitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void getLastLocation()>\|\|Source Statement:\|\|$d0 = virtualinvoke $r1.<android.location.Location: double getLatitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void onLocationChanged(android.location.Location)>\|\|Source Statement:\|\|$d0 = virtualinvoke $r4.<android.location.Location: double getLatitude()>()\|\|Source method:\|\|<com.fatburningexercises.BT_activity_base: void getLastLocation()>\|\|Source Statement:\|\|$r5 = virtualinvoke $r14.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()||Source method:||<com.fatburningexercises.BT_device: void <init>()> Application = fat.burning.exercises-7.apk_out.xml host = PrashMac source = Sources.log
>	06/12/2018 04:13:34.000	fat.burning.exercises-7.apk_out.xml||Sink Statement||$r13 = interfaceinvoke $r3.<org.apache.http.client.HttpClient: org.apache.http.HttpResponse execute(org.apache.http.client.methods.HttpUriRequest)>($r4)||Sink Method:||<com.fatburningexercises.BT_gcmServerUtils: void gcmServerPOST(java.lang.String,java.lang.String,java.util.Map)>||Sink Statement||staticinvoke <android.util.Log: int w(java.lang.String,java.lang.String)>("ZZ", $r0)\|\|Sink Method:\|\|<com.fatburningexercises.BT_debugger: void showIt(java.lang.String)>\|\|Sink Statement\|\|$r8 = virtualinvoke $r3.<java.net.URL: java.io.InputStream openStream()>()||Sink Method:||<com.fatburningexercises.BT_downloader: java.lang.String downloadTextData()> Application = fat.burning.exercises-7.apk_out.xml host = PrashMac source = Sinks.log

On diving deep into the leaks - the app was accessing getLongitude(), getLatitude(), getDeviceId() from different methods in the application. The sinks were Http POST requests (can be seen in Splunk) and Loggers.
It was collecting Device ID and Location which has triggered the FlowDroid's leaks.

On pulling up the app on the emulator, the following can be seen in EULA .

"***Our Advertisers***
AdMob display adverts in our apps.
How Admob Uses Data When You Use Our Apps For Users Inside the EU
Our partner, AdMob, uses ***device identifiers*** *to personalise ads, to provide sociall media features and to analyse our traffic. They also share such identifiers and other information.....in order to provide you with a advert specifically for your* ***location****...*"



The same is given for users outside EU too. *Thus, these leakages are not privacy violations as they have clearly mentioned about AdMob collecting these data in their EULA.*

2) com.appfondue.dailycleaningtips-1.apk:

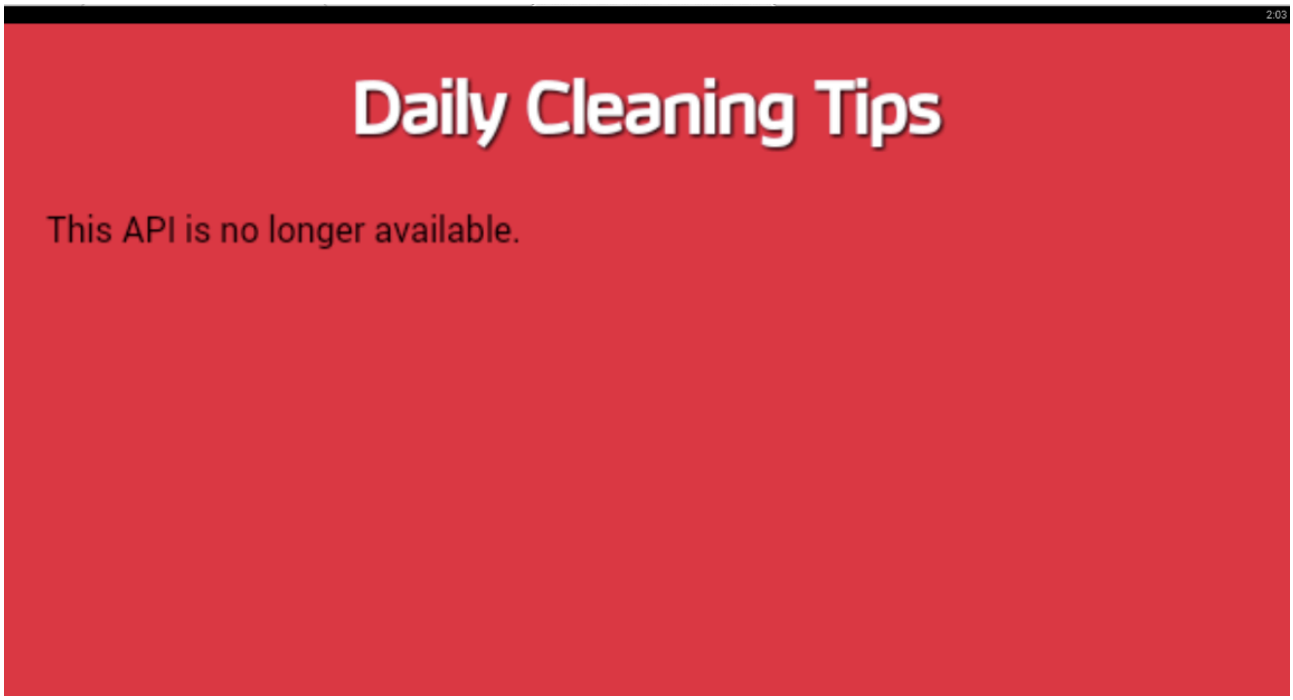
This application gave 2 leaks from FlowDroid output.

Splunk entry :

i	Time	Event
>	06/12/2018 04:14:20.000	com.appfondue.dailycleaningtips-1.apk_out.xml||Source Statement:||$r4 = virtualinvoke $r3.<android.telephony.TelephonyManager: java.lang.String getLine1Number()>()\|\|Source method:\|\|<com.appenda.Appenda: java.lang.String getPhoneNumber()>\|\|Source Statement:\|\|$r4 = virtualinvoke $r3.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.appenda.Appenda: java.lang.String getDeviceId()>\|\|Source Statement:\|\|$r4 = virtualinvoke $r3.<android.telephony.TelephonyManager: java.lang.String getLine1Number()>()\|\|Source method:\|\|<com.appenda.Appenda: java.lang.String getPhoneNumber()>\|\|Source Statement:\|\|$r4 = virtualinvoke $r3.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()||Source method:||<com.appenda.Appenda: java.lang.String getDeviceId()> Application = com.appfondue.dailycleaningtips-1.apk_out.xml host = PrashMac source = Sources.log
>	06/12/2018 04:13:34.000	com.appfondue.dailycleaningtips-1.apk_out.xml||Sink Statement||$r7 = interfaceinvoke $r3.<org.apache.http.client.HttpClient: org.apache.http.HttpResponse execute(org.apache.http.client.methods.HttpUriRequest)>($r4)||Sink Method:||<com.appenda.Utils: java.lang.String curl(java.lang.String)>||Sink Statement||staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>("Curl", $r0)||Sink Method:||<com.appenda.Utils: java.lang.String curl(java.lang.String)> Application = com.appfondue.dailycleaningtips-1.apk_out.xml host = PrashMac source = Sinks.log

The leak is triggered when getDeviceId() , getPhoneNumber() are called, basically DeviceID and Phone Number of the android device are accessed. The sink here goes to Logs as well over internet.

When tried through an emulator :



Although the App says the API is no longer available, we arent sure if its still collecting out data (device ID and Phone number) and passing it over network and storing it in Logs. Also, as an intuition, I think a "Daily Cleaning Tips" App doesnt need someone's Phone number. This gets me to an inclination that *this is leakage is a privacy violation.*

3) com.app_bipc.layout-399.apk:

This application gave 2 leaks from FlowDroid.
Splunk entry is as follows:

i	Time	Event
›	06/12/2018 04:14:20.000	com.app_bipc.layout-399.apk_out.xml||Source Statement:||$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(1)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List requestAllNotes()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(0)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List requestAllNotes()>\|\|Source Statement:\|\|$r1 = virtualinvoke $r2.<android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)>("gps")\|\|Source method:\|\|<com.biznessapps.components.LocationFinder: android.location.Location getCurrentLocation()>\|\|Source Statement:\|\|$r3 = virtualinvoke $r2.<android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)>("network")\|\|Source method:\|\|<com.biznessapps.components.LocationFinder: android.location.Location getCurrentLocation()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(2)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List requestAllNotes()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(0)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List getPhotos()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(1)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List requestAllNotes()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(0)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List requestAllNotes()>\|\|Source Statement:\|\|$r1 = virtualinvoke $r2.<android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)>("gps")\|\|Source method:\|\|<com.biznessapps.components.LocationFinder: android.location.Location getCurrentLocation()>\|\|Source Statement:\|\|$r3 = virtualinvoke $r2.<android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)>("network")\|\|Source method:\|\|<com.biznessapps.components.LocationFinder: android.location.Location getCurrentLocation()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(2)\|\|Source method:\|\|<com.biznessapps.storage.StorageAccessor: java.util.List requestAllNotes()>\|\|Source Statement:\|\|$r3 = interfaceinvoke $r4.<android.database.Cursor: java.lang.String getString(int)>(0)||Source method:||<com.biznessapps.storage.StorageAccessor: java.util.List getPhotos()> Application = com.app_bipc.layout-399.apk_out.xml host = PrashMac source = Sources.log
›	06/12/2018 04:13:34.000	com.app_bipc.layout-399.apk_out.xml||Sink Statement||$r8 = interfaceinvoke $r6.<org.apache.http.client.HttpClient: org.apache.http.HttpResponse execute(org.apache.http.client.methods.HttpUriRequest)>($r5)||Sink Method:||<com.biznessapps.api.HttpUtils: org.apache.http.HttpResponse executePostHttpRequest(java.lang.String,java.lang.String[],java.lang.String[])>||Sink Statement||$r3 = interfaceinvoke $r2.<org.apache.http.client.HttpClient: org.apache.http.HttpResponse execute(org.apache.http.client.methods.HttpUriRequest)>($r1)||Sink Method:||<com.biznessapps.api.HttpUtils: org.apache.http.HttpResponse executeGetHttpRequest(java.lang.String)> Application = com.app_bipc.layout-399.apk_out.xml host = PrashMac source = Sinks.log

The leak is triggered by Storage Access and Location access in the sinks and these details are going over internet on http.
When opening it up on an emulator, the screen as shown at the end of section comes up and crashes.

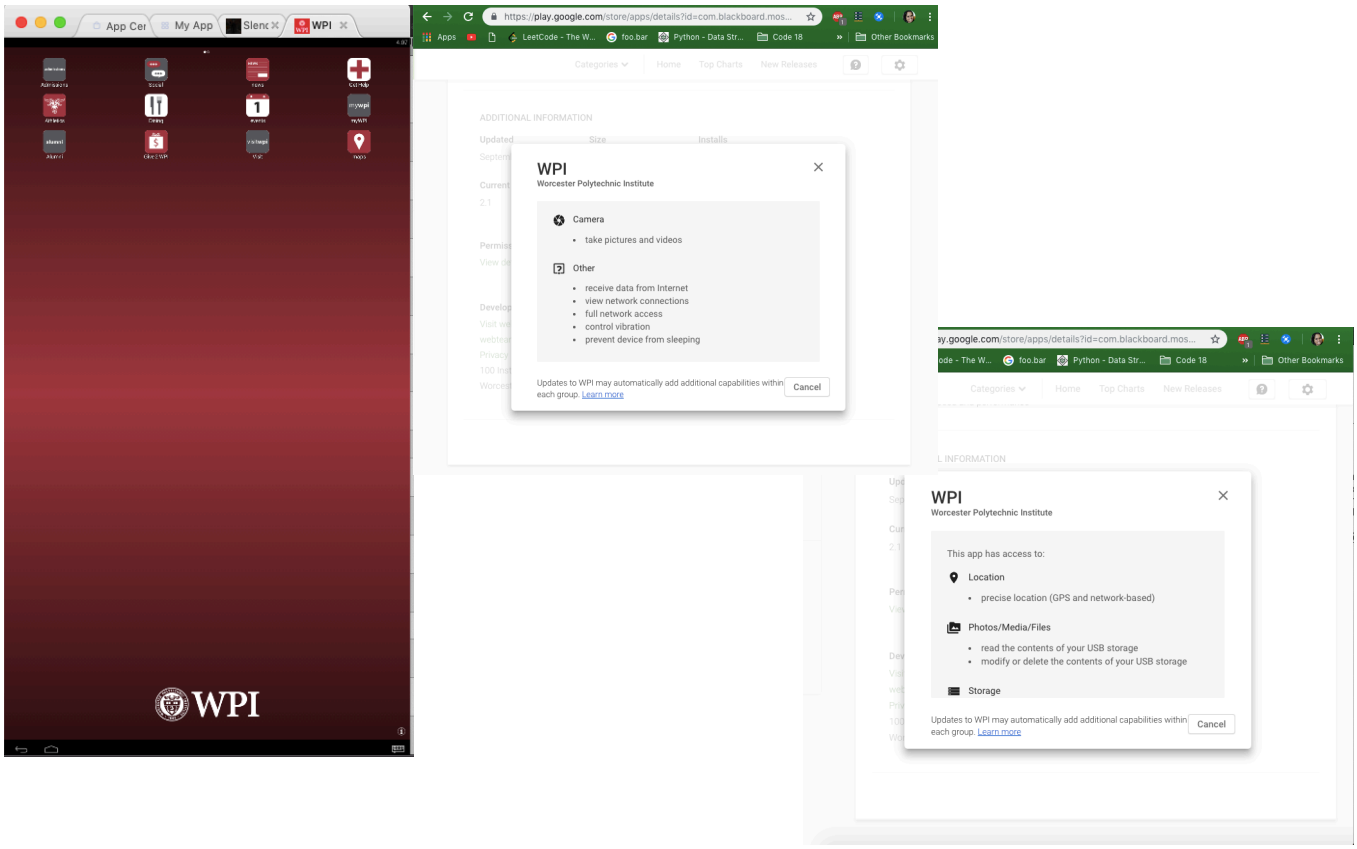
Since the app is unstable and it has access to storage and location details and sending them over internet, it is unclear as to if it has permissions to do that, considering the app crashes. It maybe that the app collects data and then crashes.A rough online search also doesnt give the privacy permissions required. Considering all the above factors, *the leakage is a privacy violation*.

4)com.blackboard.mosaic.wpi-2.apk:

There were 5 leaks for this particular app in FlowDroid .
Splunk entry :

i	Time	Event
>	06/12/2018 04:14:20.000	com.blackboard.mosaic.wpi-2.apk_out.xml||Source Statement:||$r3 = virtualinvoke $r2.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.blackboard.android.core.j.g: java.lang.String a(android.content.Context)>\|\|Source Statement:\|\|$r3 = virtualinvoke $r2.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.blackboard.android.core.j.g: java.lang.String a(android.content.Context)>\|\|Source Statement:\|\|$r6 = virtualinvoke $r3.<android.content.ContentResolver: android.database.Cursor query(android.net.Uri,java.lang.String[],java.lang.String,java.lang.String[],java.lang.String)>($r4, $r2, "title = ? AND dtstart = ?", $r5, null)\|\|Source method:\|\|<com.blackboard.android.mosaic_shared.util.CalendarUtil: android.database.Cursor getCursorAfterQueryingEvent(android.content.Context,java.lang.String,long)>\|\|Source Statement:\|\|$d0 = virtualinvoke $r2.<android.location.Location: double getLongitude()>()\|\|Source method:\|\|<com.blackboard.android.mosaic_shared.analytics.MosaicAnalyticsUtil: com.blackboard.android.mosaic_shared.data.LatLon getLocation()>\|\|Source Statement:\|\|$r3 = virtualinvoke $r2.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.blackboard.android.core.j.g: java.lang.String a(android.content.Context)>\|\|Source Statement:\|\|$r1 = interfaceinvoke $r0.<org.apache.http.HttpResponse: org.apache.http.HttpEntity getEntity()>()\|\|Source method:\|\|<com.blackboard.android.core.g.b: java.io.InputStream getStreamFromResponse(org.apache.http.HttpResponse)>\|\|Source Statement:\|\|$r3 = virtualinvoke $r2.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()\|\|Source method:\|\|<com.blackboard.android.core.j.g: java.lang.String a(android.content.Context)>\|\|Source Statement:\|\|$r5 = virtualinvoke $r3.<java.net.HttpURLConnection: java.io.InputStream getInputStream()>()||Source method:||<com.b.a.b.d.a: java.io.InputStream b(java.lang.String,java.lang.Object)> Application = com.blackboard.mosaic.wpi-2.apk_out.xml | host = PrashMac | source = Sources.log
>	06/12/2018 04:13:34.000	com.blackboard.mosaic.wpi-2.apk_out.xml||Sink Statement||staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>($r1, $r2)\|\|Sink Method:\|\|<com.blackboard.android.core.f.d: void a(java.lang.String,java.lang.String)>\|\|Sink Statement\|\|$r3 = interfaceinvoke $r3.<android.content.SharedPreferences$Editor: android.content.SharedPreferences$Editor putString(java.lang.String,java.lang.String)>("analytics_session_id", $r1)\|\|Sink Method:\|\|<com.blackboard.android.mosaic_shared.util.MosaicAndroidPrefs: void setAnalyticsSessionId(java.lang.String)>\|\|Sink Statement\|\|staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>($r2, $r1)\|\|Sink Method:\|\|<com.blackboard.android.core.f.d: void a(java.lang.String)>\|\|Sink Statement\|\|staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>($r2, $r1)||Sink Method:||<com.blackboard.android.core.f.d: void b(java.lang.String)>||Sink Statement||virtualinvoke $r1.<java.io.OutputStream: void write(byte[],int,int)>($r2, 0, $i0)||Sink Method:||<com.b.a.c.b: void a(java.io.InputStream,java.io.OutputStream)> Application = com.blackboard.mosaic.wpi-2.apk_out.xml | host = PrashMac | source = Sinks.log

The logs reported that leakages were in getting deviceID and location and passing it over network and log files.
Following are the screenshots from an emulator and permissions of app from Playstore



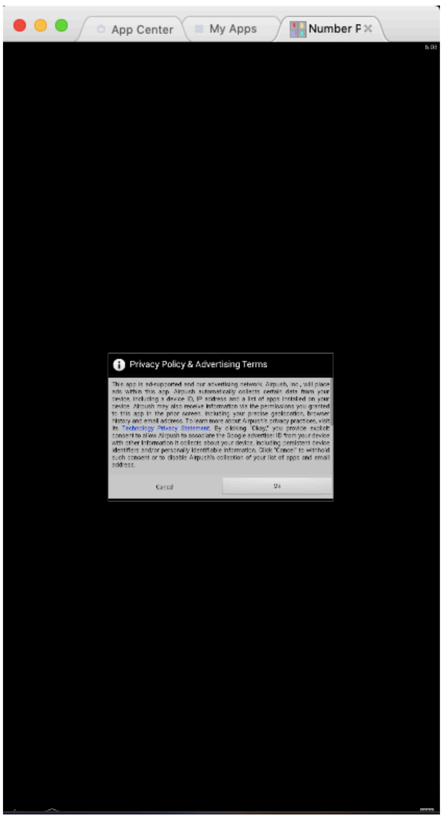
As it can be seen, there is a maps icon on the screen, which indicates that location is indeed needed for navigation.Also the permissions are correctly mentioned on Playstore for location and device details access.Considering all these facts, *the given leakages are not privacy violations.*
Ref : https://play.google.com/store/apps/details?id=com.blackboard.mosaic.wpi&hl=en

5) number.puzzle.number.puzzle-13.apk

There were 12 leaks that were reported by FlowDroid for this particular application.
Splunk entry:

<table>
<tr><th>i</th><th>Time</th><th>Event</th></tr>
<tr><td>></td><td>06/12/2018 04:14:20.000</td><td>number.puzzle.number.puzzle-13.apk_out.xml||Source Statement:||$r4 = staticinvoke <org.apache.http.util.EntityUtils: java.lang.String toString(org.apache.http.HttpEntity)>($r16)||Source method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r17 = interfaceinvoke $r15.<org.apache.http.HttpResponse: org.apache.http.StatusLine getStatusLine()>()||Source method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r17 = interfaceinvoke $r15.<org.apache.http.HttpResponse: org.apache.http.StatusLine getStatusLine()>()||Source method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Source Statement:||$r17 = interfaceinvoke $r15.<org.apache.http.HttpResponse: org.apache.http.StatusLine getStatusLine()>()||Source method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r5 = virtualinvoke $r13.<java.util.Locale: java.lang.String getCountry()>()||Source method:||<com.heyzap.house.request.FetchRequest: com.heyzap.http.RequestParams getParams(android.content.Context)>||Source Statement:||$r2 = virtualinvoke r5.<java.net.URL: java.io.InputStream openStream()>()||Source method:||<com.heyzap.house.model.AdModel$HtmlAssetFetcher: java.lang.String getBase64FromUrl(java.lang.String)>
Application = number.puzzle.number.puzzle-13.apk_out.xml host = PrashMac source = Sources.log</td></tr>
<tr><td>></td><td>06/12/2018 04:13:34.000</td><td>number.puzzle.number.puzzle-13.apk_out.xml||Sink Statement||staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("BunSDK", $r4)||Sink Method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Sink Statement||virtualinvoke $r5.<java.io.ByteArrayOutputStream: void write(byte[],int,int)>($r6, 0, $i0)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.io.InputStream,java.lang.String)>||Sink Statement||staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("BunSDK", $r4)||Sink Method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Sink Statement||staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>($r1, $r0)||Sink Method:||<com.heyzap.internal.Logger: void debug(java.lang.String)>||Sink Statement||virtualinvoke $r4.<java.io.ByteArrayOutputStream: void write(byte[])>($r5)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.lang.String)>||Sink Statement||virtualinvoke $r4.<java.io.ByteArrayOutputStream: void write(byte[])>($r5)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.lang.String)>||Sink Statement||staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("BunSDK", $r4)||Sink Method:||<com.rpkzmdqwdm.mkcfhxhzfr197254.p$1: void run()>||Sink Statement||virtualinvoke $r4.<java.io.ByteArrayOutputStream: void write(byte[])>($r5)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.lang.String)>||Sink Statement||virtualinvoke $r5.<java.io.ByteArrayOutputStream: void write(byte[])>($r6)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.io.InputStream,java.lang.String)>||Sink Statement||virtualinvoke $r5.<java.io.ByteArrayOutputStream: void write(byte[])>($r6)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.io.InputStream,java.lang.String)>||Sink Statement||virtualinvoke $r5.<java.io.ByteArrayOutputStream: void write(byte[])>($r6)||Sink Method:||<com.heyzap.http.SimpleMultipartEntity: void addPart(java.lang.String,java.lang.String,java.io.InputStream,java.lang.String)>||Sink Statement||virtualinvoke $r1.<java.io.ByteArrayOutputStream: void write(byte[],int,int)>($r3, 0, $i0)||Sink Method:||<com.heyzap.house.model.AdModel$HtmlAssetFetcher: java.lang.String getBase64FromUrl(java.lang.String)>
Application = number.puzzle.number.puzzle-13.apk_out.xml host = PrashMac source = Sinks.log</td></tr>
</table>

FlowDroid leakages were triggered because of access to location and sending them over http to the internet.
However, when pulling up the application from an emulator,



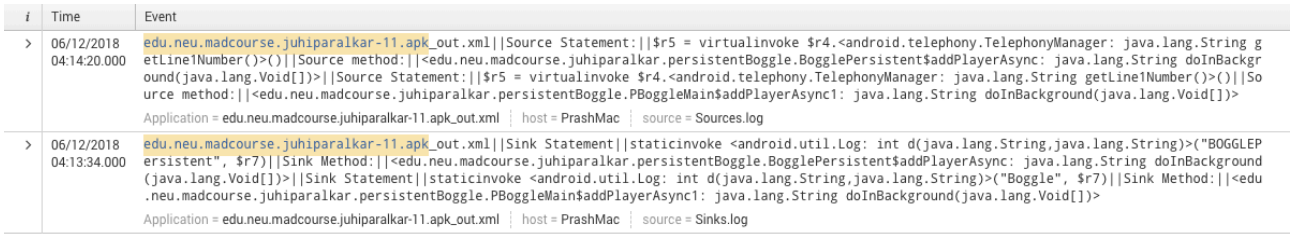
The screen opened a pop up message that read the Privacy Policy and Advertising Terms. It was clearly mentioned that for advertising reasons, location of the user, device ID etc were collected and sent to the Advertising agent.

Considering all these, *the leakages by FlowDroid are not privacy violations.*

One leak that wasnt flowing to network, but was a good one to cover was this,

6) edu.neu.madcourse.juhiparalkar-11.apk

There was 2 leaks for edu.neu.madcourse.juhiparalkar-11.apk from FlowDroid .
Splunk entry:



i	Time	Event
>	06/12/2018 04:14:20.000	edu.neu.madcourse.juhiparalkar-11.apk_out.xml||Source Statement:||$r5 = virtualinvoke $r4.<android.telephony.TelephonyManager: java.lang.String getLine1Number()>()\|\|Source method:\|\|<edu.neu.madcourse.juhiparalkar.persistentBoggle.BogglePersistent$addPlayerAsync: java.lang.String doInBackground(java.lang.Void[])>||Source Statement:||$r5 = virtualinvoke $r4.<android.telephony.TelephonyManager: java.lang.String getLine1Number()>()\|\|Source method:\|\|<edu.neu.madcourse.juhiparalkar.persistentBoggle.PBoggleMain$addPlayerAsync1: java.lang.String doInBackground(java.lang.Void[])> Application = edu.neu.madcourse.juhiparalkar-11.apk_out.xml | host = PrashMac | source = Sources.log
>	06/12/2018 04:13:34.000	edu.neu.madcourse.juhiparalkar-11.apk_out.xml||Sink Statement||staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>("BOGGLEPersistent", $r7)\|\|Sink Method:\|\|<edu.neu.madcourse.juhiparalkar.persistentBoggle.BogglePersistent$addPlayerAsync: java.lang.String doInBackground(java.lang.Void[])>||Sink Statement||staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>("Boggle", $r7)\|\|Sink Method:\|\|<edu.neu.madcourse.juhiparalkar.persistentBoggle.PBoggleMain$addPlayerAsync1: java.lang.String doInBackground(java.lang.Void[])> Application = edu.neu.madcourse.juhiparalkar-11.apk_out.xml | host = PrashMac | source = Sinks.log

The leak is triggered when the " getLine1Number() " is called to get the Phone Number of the user and it is stored in a Log file.

When Trying to access the UI, It quickly throws an error saying "Phone number not authorised ...." that disappears quickly making it difficult to even screenshot.
Since every android device trying to hit the application is checked for its phone number and its stored in Log files without the consent of the user (For eg, when I tried it in the emulator it threw me an error, but I never gave it permission to store my phone number nor I could read the EULA).

So this leakage is a privacy violation.