1 Introduction

Taint-analysis of software is the process potential user inputs are evaluated to see if they can manipulate the programs execution in a malicious intent. With the rise of mobile applications the past decade, many third-party apps can be approved for distribution on Google's Play store or Apple's App store despite there being fundamental security flaws in the software. In this project, there will be taint-analysis done on roughly 25 Android applications using Argus-SAF (previously known as and referred hereon as Amandroid) with further investigation into why and how these flaws occur.

The appset is downloaded from a local repository on Google Drive, using `mp4-appset-A`, since the first hexadecimal character of the SHA256 hash of my last name (lowercase) is `a`.

```
>>echo "voros" | openssl sha256
(stdin)= aac25310f5066dd495e440543ded228ffdd8cec5e632d04b55a6af78e366524d
```

2 High Level Statistics

The applications within the appset are listed in the following table. Running a simple bash script calling Amandroid on each `.apk` file results in 15 apps producing outputs. The remaining 10 are excluded from further analysis, due to failure of analysis or output production.

Name (full)	Taint-Analysis Successful
com.atpc-347	Yes
com.auction.resi.buyer-54	-
com.brainpop.brainpopjuniorandroid-30	Yes
com.castify-364	-
com.cootek.smartinputv5.skin.keyboard_theme_water-588	-
com.freevpnintouch-40801	Yes
com.ilikeyou-787	Yes
com.joom-3153925	Yes
com.lbrc.PeriodCalendar-61100	Yes
com.northpark.beautycamera-77	Yes
com.nosixfive.verto-1050019	Yes
com.peoplemedia.blackpeoplemeet-311	Yes
com.scannerradio-6749	-
com.thetransitapp.droid-3013721	-
com.tql.carrierdashboard-135	Yes
com.transloc.android.rider-44	Yes
com.wildec.meet24-165	-
com.yazio.android-41104110	-
gov.irs-63	Yes
navigation.location.maps.finder.directions.gps.gpsroutefinder-19	-
pedometer.steptracker.calorieburner.stepcounter-48	Yes
photocollage.photoeditor.photocollageeditor-11	-
pl.trpaslik.babynoise-172	Yes
tv.telepathic.hooked-105	-
twitch.angelandroidapps.tracerlightbox-27001	Yes

Table 1: Which apps within the appset produced output from Amandroid

The machine I am currently on is a laptop with limited RAM and computing power, therefore the taint-analysis would have taken many hours and potentially days. Therefore, Amandroid taint-analysis was run externally on NCSUs remote EOS servers to decrease run-time. The binaries for the latest snapshot of Amandroid alongside the `.apk` files within appset-A were uploaded to a Github repository and cloned onto my local directory in AFS. A bash script entered the appset folder and looped through all the `.apk` files and ran the following command for each file

```
java -Xmx4g -jar argus-saf-3.2.1-SNAPSHOT-assembly.jar t -mo DATA_LEAKAGE -a
    COMPONENT_BASED -o app_results $app_folder/$file
```

where the `Xmx4g` refers to allocating 4GB of RAM toward this process (8GB would result in EOS killing the command. Locally, only 1GB was possible), `.jar` file is the Amandroid binary, `t` refers to taint-analysis, `DATA_LEAKAGE` further specifies type of taint-analysis, `COMPONENT_BASED` reduces RAM requirements and run-time due to how it handles inter-component communication (ICC), `app_results` refers to name of the output folder, and `$app_folder/$file` refer to the bash scripts environment variables to reference the source `.apk` file.

The resulting `AppData.txt` output files (for all Android applications that successfully performed taint-analysis) were compiled into a folder for further parsing and analysis. The points of interest for compiling statistics regarding security flaws come from taint paths, where transfer nodes of potentially tainted data (source and sink) are determined by Amandroid. The tainted data can be malicious in numerous ways. Since analysis was done with the `DATA_LEAKAGE` run configuration, all the taint paths within our output show potential sources of data leakage and thus information theft. Other possible run configurations for Amandroid include: `INTENT_INJECTION`, `PASSWORD_TRACKING`, `OAUTH_TOKEN_TRACKING`, and `COMMUNICATION_LEAKAGE`.

Without any prior knowledge of Amandroid, it was assumed that some of the `DATA_LEAKAGE` taint paths could potentially be used maliciously to fit other categories, depending on the sink of the tainted data. For example, if the sink is `Landroid/util/Log` with datatype `Ljava/lang/String`, then it is evident this can only contribute to data leakage. However, if the sink is `Landroid/os/Handler` with the same `String` datatype, then this could contribute to `INTENT_INJECTION` as well. Fortunately, it was discovered that Amandroid's discovered taint paths are mutually exclusive with respect to the analysis configuration, so all taint paths output by `DATA_LEAKAGE` are for data leakage only.

Sink Routine of Interest	**Description**
`Landroid/util/Log`	Log, mainly used for debugging purposes
`Landroid/os/Handler`	OS handler, typically error messages
`Landroid/app/Activity`	Manages activity, input of `Intent`
`Landroid/content/SharedPreferences$Editor?`	Locally saved preferences w.r.t. the app
`Landroid/content/Context`	Interface to global environment
`Landroid/content/ContextWrapper`	Interface to global environment
`Ljava/io/Writer`	Write to file
`Ljava/io/FileOutputStream`	Write to file
`Ljava/net/URL`	Net request
`Ljava/net/URLConnection`	Net connection
`Ljava/net/HttpURLConnection?`	Net connection

Table 2: Tainted data sink routines: descriptions

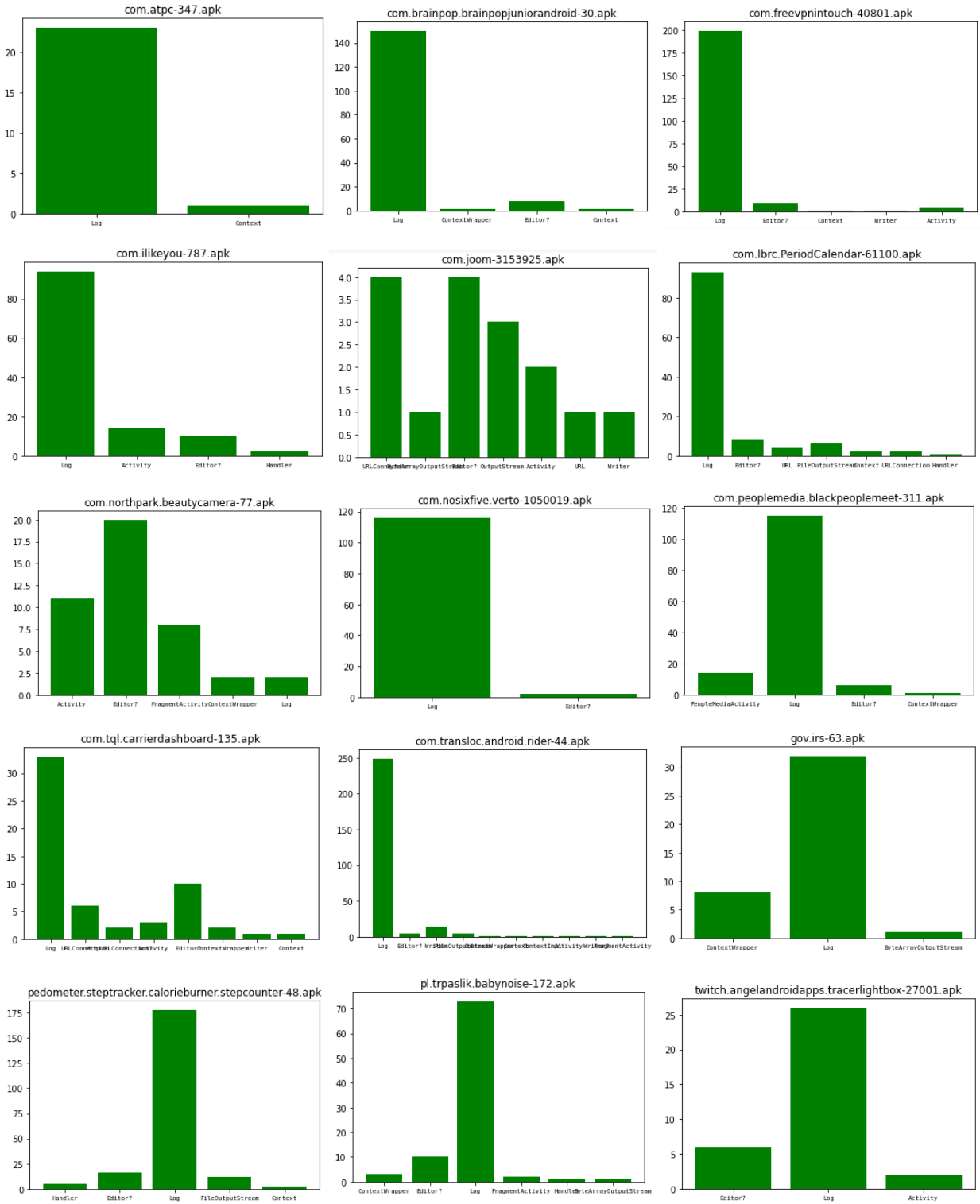

Figure 1: Tainted data sink routines: count per application

As seen in Figure 1, the 15 Android apps which were able to be analyzed gave information regarding routines that are able to accept potentially tainted data. The full data lists can be seen in the Appendix, since the figures might be difficult to read. The most common routine was `Landroid/util/Log`. Above in Table 2 is a list of other common/notable routines with a brief description of each.

Below in Table 3, a summary of sink routines can be seen. The higher the number in the two leftmost columns, the more potential data leakage and security flaws. The total number of sinks column is simply a sum of all instances from the Amandroid analysis of all sink routines (also seen as a sum of all counts seen in the histograms of Figure 1). The leftmost column indicates the number of identified taint paths. (*Note: I am unsure if some of the Amandroid outputs were not complete, but some of the applications showed 0 taint paths. Which is unusual, since how can there be a source and a sink of tainted data but no path connecting them? However, I could simply be misinterpreting what this means.*) Lastly, network sinks are ones of high interest so they have their own column.

Name (short)	**# of Net Sinks**	**Total # of Sinks**	**Identified # of Paths**
atpc	0	24	0
brainpop	0	160	0
freevpnintouch	0	213	8
ilikeyou	0	120	2
joom	5	16	2
PeriodCalendar	6	116	3
beautycamera	0	43	8
verto	0	118	54
blackpeoplemeet	0	136	120
carrierdashboard	8	58	0
transloc rider	0	277	0
irs	0	41	10
stepcounter	0	213	6
babynoise	0	90	17
tracerlightbox	0	34	0

Table 3: Sink routine counts per Android application

3 Determining Privacy Violations

Privacy violations can be determined by further interpreting Table 2 and assuming the worst-case scenario.

1. `Landroid/util/Log`: The log is usually only used by developers during development for debugging purposes. This data flow can potentially display vital information with respect to program variables for an adversary to gain information about the program.

   For example, a developer could have accidentally kept displaying private information from the network, local storage, etc., in the logger which can now be intercepted by the adversary.

2. `Landroid/os/Handler`: All data paths and instances of `DATA_LEAKAGE` taint-analysis, this OS sink had an OS message as the source. Meaning, this was displaying errors when

encountered. Errors can hold vital information with respect to how a program functions, which can be beneficial to an adversary.

For example, an adversary can find weak points in the software by taking advantage what they have learned from these OS handler messages.

3. `Landroid/app/*`: Tainted data serving as input parameters to `Landroid/app/*` can be dangerous because it can affect the overarching run-time of the application. I.e., activities and fragments can be created, destroyed, started, stopped, resumed, or paused in an unorthodox manner.

   For example, instead of calling a 'microphone' activity to be destroyed, it can remain functioning even when the application does not intend it to.

4. `Landroid/content/*`: Somewhat similar to the OS handler above, `Landroid/content/*` handles data flow between components (ICC) of global environment variables (content).

   For example, after importing data (taking a picture, recording audio, inputting private information, etc.) this can be stored in a global environment to be potentially used again by another component. An adversary can now intercept or manipulate the data along these data paths.

5. `Ljava/io/*`: Tainted data potentially affecting IO data flow is dangerous. Manipulated data directly changes input or output file-streams which an adversary can take advantage of in many ways.

   For example, an adversary can utilize this point in the data flow to intercept private stored information, such as API keys, passwords, etc.

6. `Ljava/net/*`: Tainted data potentially affecting network sinks is especially dangerous, since an adversary can pull private information from responses or affect the outgoing request in a way to act maliciously by posing as the client.

   For example, after incoming data has already been decrypted OR before outgoing data has been encrypted, an adversary can utilize this point in the data flow to intercept raw messages.

There were only 3 applications of the 15 with affected network sinks (as seen from Table 3: joom, PeriodCalendar, and carrierdashboard. A process that can be used to determine whether any of these paths in taint-analysis w.r.t. network (and other) sinks are false positive would be decompilation. The `.apk` files can be decompiled using different available Android decompiler tools and the source code can be examined, so see exactly what is a true positive by Amandroid and what is actually safe but reported as data leakage.

4 Appendix

Simple script used for parsing

```
import os
import re
import json

from collections import Counter
import matplotlib.pyplot as plt

flatten =lambda x: [i for row in x for i in row]
```

```
# loop through files in directory
foldername ="outputs"
fnames =os.listdir(foldername)
for fname in fnames:
    # read in file as string
    fstr =open(foldername +"/" +fname).read()

    # app name
    appname =re.findall('(?<=Application Name: ).+?(?=\n)', fstr)[0]
    print(appname)

    # all source/sink
    unparsedss =re.findall('(?:  <Descriptors: )(.+?)_(.+): (.+?);(?:.+):\((.*?)\)(?:.+)(?=\n)',
                                                    fstr)

    # parse sources
    sources =[[ss[0], ss[2], ss[3]] for ss in unparsedss if ss[1] =="source"]
    sidx =0;
    for src in sources:
        if ";" in src[2]:
            datatypes =re.findall('.*;', src[2])[0].split(";")[:-1]
            didx =0;
            for dt in datatypes:
                datatypes[didx] =re.findall('(?:)L.*', dt)[0]
                didx +=1
            sources[sidx][2] =datatypes
        else:
            sources[sidx][2] =["any"]
        sidx +=1

    # parse sinks
    sinks =[[ss[0], ss[2], ss[3]] for ss in unparsedss if ss[1] =="sink"]
    sidx =0;
    for snk in sinks:
        if ";" in snk[2]:
            datatypes =re.findall('.*;', snk[2])[0].split(";")[:-1]
            didx =0;
            for dt in datatypes:
                datatypes[didx] =re.findall('(?:)L.*', dt)[0]
                didx +=1
            sinks[sidx][2] =datatypes
        else:
            sinks[sidx][2] =["any"]
        sidx +=1

    # get source counts + uniques
    src_type_counts =dict(Counter([src[0] for src in sources]))
    src_type_unique =[x for x in src_type_counts]
    src_func_counts =dict(Counter([src[1] for src in sources]))
    src_func_unique =[x for x in src_func_counts]
    src_dttp_counts =dict(Counter(flatten([src[2] for src in sources])))
    src_dttp_unique =[x for x in src_dttp_counts]

    # get sink counts + uniques
    snk_type_counts =dict(Counter([snk[0] for snk in sinks]))
    snk_type_unique =[x for x in snk_type_counts]
    snk_func_counts =dict(Counter([snk[1] for snk in sinks]))
    snk_func_unique =[x for x in snk_func_counts]
    snk_dttp_counts =dict(Counter(flatten([snk[2] for snk in sinks])))
    snk_dttp_unique =[x for x in snk_dttp_counts]
```

```
    print(src_type_counts)
    print(src_func_counts)
    print(src_dttp_counts)
    print()
    print(snk_type_counts)
    print(snk_func_counts)
    print(snk_dttp_counts)

    print()
    print()
```

Taint-analysis data. Used in Figure 1 and all tables

```
com.atpc-347.apk
Source routines:
{'Landroid/content/Intent': 2, 'Landroid/content/pm/PackageManager': 1, 'Landroid/app/
    PendingIntent': 1}
Sink routines:
{'Landroid/util/Log': 23, 'Landroid/content/Context': 1}

com.brainpop.brainpopjuniorandroid-30.apk
Source routines:
{'Landroid/content/pm/PackageManager': 1}
Sink routines:
{'Landroid/util/Log': 150, 'Landroid/content/ContextWrapper': 1, 'Landroid/content/
    SharedPreferences$Editor?': 8, 'Landroid/content/Context': 1}

com.freevpnintouch-40801.apk
Source routines:
{'Landroid/content/Intent': 11, 'Landroid/content/pm/PackageManager': 3, 'Landroid/app/
    PendingIntent': 1, 'Lcom/zendesk/sdk/feedback/ui/ContactZendeskFragment': 1, 'Lcom
    /zendesk/sdk/requests/ViewRequestFragment': 1}
Sink routines:
{'Landroid/util/Log': 199, 'Landroid/content/SharedPreferences$Editor?': 8, 'Landroid/
    content/Context': 1, 'Ljava/io/Writer': 1, 'Landroid/app/Activity': 4}

com.ilikeyou-787.apk
Source routines:
{'Landroid/content/Intent': 62, 'Landroid/os/Handler': 2, 'Lcz/ackee/androidskeleton/ui
    /activity/base/BaseFragmentActivity': 3, 'Landroid/app/PendingIntent': 1, 'Lcz/
    ackee/androidskeleton/ui/activity/MainDrawerActivity': 4}
Sink routines:
{'Landroid/util/Log': 94, 'Landroid/app/Activity': 14, 'Landroid/content/
    SharedPreferences$Editor?': 10, 'Landroid/os/Handler': 2}

com.joom-3153925.apk
Source routines:
{'Ljava/net/URLConnection': 2, 'Landroid/content/Intent': 6, 'Landroid/content/pm/
    PackageManager': 1}
Sink routines:
{'Ljava/net/URLConnection': 4, 'Ljava/io/ByteArrayOutputStream': 1, 'Landroid/content/
    SharedPreferences$Editor?': 4, 'Ljava/io/OutputStream': 3, 'Landroid/app/Activity
    ': 2, 'Ljava/net/URL': 1, 'Ljava/io/Writer': 1}

com.lbrc.PeriodCalendar-61100.apk
Source routines:
```

```
{'Landroid/content/Intent': 22, 'Landroid/os/Handler': 2, 'Landroid/app/PendingIntent':
     5, 'Ljava/net/URLConnection': 2}
Sink routines:
{'Landroid/util/Log': 93, 'Landroid/content/SharedPreferences$Editor?': 8, 'Ljava/net/
    URL': 4, 'Ljava/io/FileOutputStream': 6, 'Landroid/content/Context': 2, 'Ljava/net
    /URLConnection': 2, 'Landroid/os/Handler': 1}

com.northpark.beautycamera-77.apk
Source routines:
{'Landroid/content/Intent': 52, 'Lcom/northpark/beautycamera/AdActivity': 4}
Sink routines:
{'Landroid/app/Activity': 11, 'Landroid/content/SharedPreferences$Editor?': 20, '
    Landroid/support/v4/app/FragmentActivity': 8, 'Landroid/content/ContextWrapper':
    2, 'Landroid/util/Log': 2}

com.nosixfive.verto-1050019.apk
Source routines:
{'Landroid/content/Intent': 8, 'Lcom/google/example/games/basegameutils/
    BaseGameActivity': 2, 'Landroid/app/PendingIntent': 1, 'Lcom/nosixfive/anative/
    aNativeActivity': 2}
Sink routines:
{'Landroid/util/Log': 116, 'Landroid/content/SharedPreferences$Editor?': 2}

com.peoplemedia.blackpeoplemeet-311.apk
Source routines:
{'Landroid/content/Intent': 26, 'Lcom/pm/android/todays_matches/TodaysMatchesFragment':
     1, 'Lcom/pm/android/PeopleMediaActivity': 5}
Sink routines:
{'Lcom/pm/android/PeopleMediaActivity': 14, 'Landroid/util/Log': 115, 'Landroid/content
    /SharedPreferences$Editor?': 6, 'Landroid/content/ContextWrapper': 1}

com.tql.carrierdashboard-135.apk
Source routines:
{'Landroid/app/PendingIntent': 5, 'Landroid/content/Intent': 15, 'Landroid/content/pm/
    PackageManager': 1, 'Ljava/net/URLConnection': 3}
Sink routines:
{'Landroid/util/Log': 33, 'Ljava/net/URLConnection': 6, 'Ljava/net/HttpURLConnection?':
     2, 'Landroid/app/Activity': 3, 'Landroid/content/SharedPreferences$Editor?': 10,
    'Landroid/content/ContextWrapper': 2, 'Ljava/io/Writer': 1, 'Landroid/content/
    Context': 1}

com.transloc.android.rider-44.apk
Source routines:
{'Landroid/content/Intent': 3, 'Landroid/app/PendingIntent': 3, 'Landroid/content/pm/
    PackageManager': 1}
Sink routines:
{'Landroid/util/Log': 249, 'Landroid/content/SharedPreferences$Editor?': 4, 'Ljava/io/
    Writer': 14, 'Ljava/io/FileOutputStream': 4, 'Landroid/content/ContextWrapper': 1,
     'Landroid/content/Context': 1, 'Landroid/app/ContextImpl': 1, 'Landroid/app/
    Activity': 1, 'Ljava/io/Writer?': 1, 'Landroid/support/v4/app/FragmentActivity':
    1}

gov.irs-63.apk
```

```
Source routines:
{'Landroid/content/pm/PackageManager': 3, 'Landroid/content/Intent': 3, 'Lorg/apache/
    http/HttpResponse': 1}
Sink routines:
{'Landroid/content/ContextWrapper': 8, 'Landroid/util/Log': 32, 'Ljava/io/
    ByteArrayOutputStream': 1}

pedometer.steptracker.calorieburner.stepcounter-48.apk
Source routines:
{'Landroid/content/Intent': 13, 'Landroid/location/LocationManager': 2, 'Landroid/os/
    Handler': 5, 'Landroid/app/PendingIntent': 1, 'Lsteptracker/stepcounter/pedometer/
    service/WorkOutService': 1}
Sink routines:
{'Landroid/os/Handler': 5, 'Landroid/content/SharedPreferences$Editor?': 16, 'Landroid/
    util/Log': 178, 'Ljava/io/FileOutputStream': 12, 'Landroid/content/Context': 2}

pl.trpaslik.babynoise-172.apk
Source routines:
{'Landroid/app/PendingIntent': 4, 'Landroid/content/Intent': 5, 'Landroid/os/Handler':
    1}
Sink routines:
{'Landroid/content/ContextWrapper': 3, 'Landroid/content/SharedPreferences$Editor?':
    10, 'Landroid/util/Log': 73, 'Landroidx/fragment/app/FragmentActivity': 2, '
    Landroid/os/Handler': 1, 'Ljava/io/ByteArrayOutputStream': 1}

twitch.angelandroidapps.tracerlightbox-27001.apk
Source routines:
{'Landroid/content/Intent': 4}
Sink routines:
{'Landroid/content/SharedPreferences$Editor?': 6, 'Landroid/util/Log': 26, 'Landroid/
    app/Activity': 2}
```