Detecting and Measuring Aggressive Location Harvesting in Mobile Apps via Data-flow Path Embedding

HAORAN LU*, Indiana University Bloomington, USA
QINGCHUAN ZHAO*, City University of Hong Kong, Hong Kong
YONGLIANG CHEN, City University of Hong Kong, Hong Kong
XIAOJING LIAO, Indiana University Bloomington, USA
ZHIQIANG LIN, Ohio State University, USA

Today, location-based services have become prevalent in the mobile platform, where mobile apps provide specific services to a user based on his or her location. Unfortunately, mobile apps can aggressively harvest location data with much higher accuracy and frequency than they need because the coarse-grained access control mechanism currently implemented in mobile operating systems (*e.g.*, Android) cannot regulate such behavior. This unnecessary data collection violates the data minimization policy, yet no previous studies have investigated privacy violations from this perspective, and existing techniques are insufficient to address this violation. To fill this knowledge gap, we take the first step toward detecting and measuring this privacy risk in mobile apps at scale. Particularly, we annotate and release the *first* dataset to characterize those aggressive location harvesting apps and understand the challenges of automatic detection and classification. Next, we present a novel system, LocationScope, to address these challenges by *(i)* uncovering how an app collects locations and how to use such data through a fine-tuned value set analysis technique, *(ii)* recognizing the fine-grained location-based services an app provides via embedding data-flow paths, which is a combination of program analysis and machine learning techniques, extracted from its location data usages, and *(iii)* identifying aggressive apps with an outlier detection technique achieving a precision of 97% in aggressive app detection. Our technique has further been applied to millions of free Android apps from Google Play as of 2019 and 2021. Highlights of our measurements on detected aggressive apps include their growing trend from 2019 to 2021 and the app generators' significant contribution of aggressive location harvesting apps.

1 INTRODUCTION

Nowadays, location-based services (*LBSes*) have become extremely popular in the mobile platform, and many mobile applications (*apps*) deliver various LBSes to their users based on their locations. For example, Yahoo Weather acquires locations to provide local weather forecasts, and Google Map collects locations in real-time to navigate users to their destinations. Considering the location data is greatly sensitive to user privacy, the access control of users' location data is enforced by mobile operating systems (*e.g.*, Android) with a permission-based security mechanism, which typically provides three permissions, *i.e.*, the one-time permission, the foreground permission, and the background permission.

However, current location-related permissions only regulate *when* an app could access the location data but put no restrictions on *how* an app access the location, such as the frequency to retrieve the data and the granularity in terms of location accuracy, after being granted permissions. Consequently, apps can collect location data beyond their need to deliver their LBSes. For example, we found 41 Google Play apps providing the zipcode auto-fill service that collects the *precise location* less than every 0.3 seconds on average, while one-time access of the *coarse location* appears to be sufficient. In this paper, an app performing such unnecessary data collection is an *aggressive location harvesting app (ALHA)*.

ALHA raises significant privacy concerns and brings an alarming situation which has been described as "*every moment of every day, mobile apps collect detailed location data*" [93]. In addition

*Both authors contributed equally to this research.

to our newly discovered 41 aggressive zipcode auto-fill apps, the media has reported a study of 20 weather apps that showed a similar situation of aggressive location data harvesting [88, 93]. In particular, most of these weather apps aggressively collect GPS coordinates of users beyond the necessity to deliver the functionality as a weather app, and the harvested data had been confirmed to be able to perform precise in-door tracking (within a building) and out-door tracking (driving from work to home) [88, 93]. Even worse, the location data harvested from several apps had been sold to third parties, including advertisers, retail outlets, hedge funds, and political campaigns [93].

Even if this privacy-concerning behavior in *ALHA* could be hopefully mitigated by newly launched regulations (*e.g.*, GDPR [44]), existing techniques and solutions are yet sufficient to detect, measure, and suppress such aggressive behaviors in mobile apps. In fact, *ALHA* violates a basic principle, *i.e.*, data minimization, in these regulations. For example, the GDPR [44] defines this principle as "personal data shall be adequate, relevant and limited to what is necessary for relation to the purposes for which they are processed (data minimization)". Though using different expressions, other similar policy regulations and advocacy, *e.g.*, CCPA [28], also contain this minimization principle. Unfortunately, most previous studies, which investigated location privacy issues from the perspective of understanding privacy policy violations (*e.g.*, [29, 39, 40, 61, 62, 80, 81, 86, 94, 102]), could fail to identify the data minimization violations due to the absent knowledge of the location collection precision and frequency requirement and the lack of ability to recognize fine-grained LBSes.

Specifically, while some other works build tools to detect location data leakages [31, 81, 82, 85] and measure the location tracking in certain mobile apps [82, 84, 102], they primarily focus on coarse-grained LBSes instead of the fine-grained LBSes. On the other hand, even if it appears feasible to use a methodology from works that detect violations in mobile apps without well-acknowledged standards by comparing its consumption of privacy-related resources with its peer apps providing similar functionality (*e.g.*, AAPL [64]), it poses another challenge on recognizing similar functionality on the fine-grained level. Unfortunately, existing solutions, such as depending on recommendation systems (*e.g.*, Google Play) [64] or a workaround to extract functionalities for later similarity comparison from both the text (*e.g.*, app description [76]) and the code [71, 72, 75] of an app, cannot differentiate the fine-grained LBSes. Therefore, there is an urgent need to detect *ALHA* and measure its severity at scale.

Yet, it is non-trivial to address the challenges in *ALHA* detection. At a high level, its major challenges include *(i)* how to identify the fine-grained LBSes an app provides; *(ii)* how to associate each identified fine-grained LBS to its location data retrieving behavior within an app; and, most importantly, *(iii)* how to determine the minimum location requirement for each fine-grained LBS without a consensus on data minimization standards. The last one is extremely challenging because different LBSes can have completely different demands of the location data accuracy and frequency. For example, it may only need a one-time and district-level location to geo-tag photos while navigation services have to retrieve real-time and the most accurate locations periodically to navigate users. Moreover, while a large body of work is devoted to inspecting the location collections in mobile apps, none of them investigates location privacy from the perspective of data minimization violation. Their proposed methodology, algorithms, and tools primarily aim to detect whether location data has been collected and rarely consider their justification for collecting location data in terms of the precision of location and the frequency of data collection.

In this paper, we propose LocationScope to address these challenges and detect *ALHA* at scale. Specifically, LocationScope contains three phases. The first phase is to take an app as input and use static program analysis to uncover its periodical location usages, including (*i*) how it accesses the location data and (*ii*) how it processes the retrieved location to implement an LBS. Specifically, location data access is analyzed in a multi-dimension manner by capturing the accuracy of location

data, and the *spatial and temporal* frequency of location updates it desires, and the location data processing is understood from the location data-flow paths that send location to the outside (*e.g.*, servers). Next, the second phase is to recognize fine-grained LBSes by clustering the location data-flow paths. In particular, LBSes are abstracted from the corresponding location data-flow path as numerical embeddings based on code embedding techniques. The embeddings are then clustered based on their semantic similarity such that each cluster holds LBSes of similar functionalities. Last, a semi-supervised approach is used to identify *ALHA* by looking for outliers in each specific LBS cluster, which avoids extracting specific domain knowledge for each LBS in terms of the minimum data usage requirement. Our corresponding evaluation on a ground-truth dataset, which consists of 200 manually annotated *ALHA* (§3.1), shows that this algorithm can achieve a precision of 97%.

We have built a prototype of LocationScope and evaluated it on two datasets: one consists of 2.16 million free apps, and the other one contains 2.05 million free apps, which were all crawled from Google Play as of 2021 and 2019, respectively. We discovered 14,091 aggressive apps from 2021 and 38,688 from 2019 that provide LBSes, including current address auto-filling, nearby service searching, weather forecasting, *etc.* Among the 14,091 aggressive apps from 2021, 10,819 apps (76%) even retain effective location collection behavior against the background location protection introduced in Android 10 (see §6). In addition, our study revealed a significant source of aggressive apps when investigating the root causes. Specifically, we find that 58.84% and 65.43% of aggressive apps in 2021 and 2019 are built from app generators, such as Goodbarber, Biznessapps, and Appsgeyser. Interestingly, we have identified that apps generated by the latter two generators would request fine-grained location permissions and send the locations back to the remote servers of the app generators, even if there are no LBS visible to users.

Contribution. In short, we make the following contributions:

- **Novel Approach.** We present a novel approach of using code analysis and representation learning to detect aggressive location harvesting in mobile apps.
- **New Dataset.** We annotate and release the first aggressive location harvesting dataset of mobile apps[1], in support of open science and any follow-up research.
- **Empirical Findings.** With the built prototype, LocationScope, we discovered a growing trend of aggressive location harvesting in Android apps from 2019 to 2021 and revealed that app generators contributed to over half of these apps.

2 BACKGROUND

2.1 Mobile Apps Location Collection

Mobile apps can retrieve the location data from almost all smartphones by invoking relevant system APIs. In Android, apps can use system APIs to access the location data from the following two sources with different granularities of accuracy.

- **GPS signals**: The Global Positioning System (GPS) is a satellite-based navigation system that provides time and location information to any receiver with the line of sight to at least four of the thirty-one GPS satellites that can give fine-grained locations. In addition, most modern smartphones contain Assisted GPS (A-GPS) receivers which provide faster location readings by caching and obtaining satellite data.
- **Geo-location of networks**: The local networks can also be used to obtain location information, which is coarser than GPS signals. In particular, the device can retrieve location from remote location databases by providing nearby cell towers or WiFi routers. This approach allows for fast and reasonably accurate location reports in areas with numerous wireless access points,

[1]The Dataset is available at https://sites.google.com/view/aggressive-app/aggressive-app-dataset (see Table 3 in Appendix)

particularly in the in-door environment [10]. In addition, the device can also obtain the location via IP-based geolocation [52] when the accuracy is less critical [79].

How to access location data. Android apps can read locations from GPS receivers or the network by claiming the associated `provider`: (*i*) the `gps` provider, requiring an app to hold `ACCESS_FINE_LOCATION` permission, that can read the current location from the GPS sensor, (*ii*) the `network` provider, requiring the `ACCESS_COARSE_LOCATION` permission that can obtain a less accurate location from the network with less battery consumption, and (*iii*) the `passive` provider that simply collates the most up-to-date information from the other two providers depending on the held permissions without draining the battery. In addition, apps can request to retrieve the updated location data periodically by setting both time and distance intervals between two consecutive retrieval attempts. For example, the Android API `requesttLocationUpdates` can be used to periodically retrieve updated locations by configuring two parameters, *i.e.*, the `minTime` that determines the minimum time interval (*e.g.*, 1,000 ms) between two consecutive location data access (frequency) and the `minDistance` which depicts the interval of data access in terms of distance (*e.g.*, 100 m). Moreover, apps can use Fused Location Provider APIs [35], which is part of the Google Play Services APK working with Google Mobile Service (*GMS*), to configure its time interval in receiving periodical location updates, such as using `locationRequest.setInterval` and `locationRequest.setFastestInterval` to obtain location updates every 5 seconds and 1 second, respectively.

2.2 Data Minimization in Location Privacy

Being effect in 2018 in the EU, the General Data Protection Regulation (*GDPR*) expresses the data minimization principle in Article 5(1)(c) that personal data should be "*adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed*" [44], which also complies with Article 4(1)(c) of Regulation (EU) 2018/1725 and the upcoming Proposition 24 of California Consumer Privacy Act (*CCPA*). Specific to location data, which is widely recognized as a major piece of personal data, American Civil Liberties Union (*ACLU*) has specified that "*only necessary information is collected*" [47] (*a.k.a.*, the minimum usage policy), especially when apps are capable of accessing locations, such as COVID-19 contact tracing. While mobile operating systems (*e.g.*, Android) keep evolving and proposing mechanisms (some of which are optional) to address this privacy violation, their effectiveness in practice still remain largely unknown.

3 OVERVIEW

3.1 Annotating the *ALHA*

We have to build a ground truth dataset for the *ALHA* to characterize apps performing such an aggressive behavior. However, to the best of our knowledge, there are no existing public datasets of the *ALHA*, nor approaches to identify such an app. Therefore, we use the qualitative open-coding techniques [92] to determine whether an app sample is an *ALHA* or not. More specifically, four mobile security researchers were asked to annotate all aggressive apps independently following the *ALHA* annotation guide[2] as elaborated below.

Annotating LBS(es) of an app. The whole annotation procedure starts with annotating LBS(es) of an app. The annotators are asked to install the apps on an instrumented Android device, grant all permission, and manually explore and trigger functionalities in the apps with their best efforts. If noticed invocations of `Location.getLatitude` and `getLongigude`, they will record and describe the found LBS and the associated `minTime` and `minDistance`. In cases that the annotator cannot identify any service being provided after the location invocations (*e.g.*, the invocation took place

[2]The detail of the annotation guide is available at https://sites.google.com/view/aggressive-app/app_annotation_guide

immediately after location permission was granted at app launch), the LBS is recorded as "no LBS provided". For all LBSes, the annotators independently code all the LBS descriptions using the Grounded Theory [92] and compare their final code books using inter-coder agreement (following the method used in [70]), record each LBS's corresponding LBS category. For example, an LBS of a weather app could be recorded as "Uses location to decide which city's weather information should be displayed", and the LBS category assigned to it could be "City Weather". Next, the annotators annotate each single LBS as aggressive or not using two rules: *(i)* if the LBS does not need constant location collection (*e.g.*, LBS that "Uses location to auto-fill zipcode") and *(ii)* if the LBS needs constant location collection, but the `minTime` and `minDistance` parameters are not adequate, relevant, and limited to what is necessary in relation to the LBS. For example, if the LBS "Uses location to decide which city's weather information should be displayed" uses 0 for both `minTime` and `minDistance` and the minimal value used by the other apps in the "City Weather" LBS category is larger than 0, this LBS is annotated as aggressive.

Annotating *ALHA*. After each LBS is annotated, an app could be annotated based on its LBS annotation. For apps only having a single LBS, the app will be annotated as aggressive if its LBS is aggressive. When an app has more than one LBS, different LBSes within the same app may share the `minTime` and `minDistance` values, due to the common programming pattern of setting a global shared Location variable. As a result, the parameter value fulfilling the most demanding LBS in an app is used for the other less demanding LBSes as well, causing them to be annotated as aggressive during the LBS annotation step. To avoid overestimating the aggressiveness of an app with multiple different LBSes, the annotator only labels an app as aggressive if its most demanding LBS (*i.e.*, LBS having the smallest `minTime` and `minDistance` values) is labeled as aggressive. Hence, we consider the result as the "lower-bound". We detail the statistics of the dataset in §5.1.

3.2 Challenges and Insights

Based on our observations in annotating the *ALHA* uncovered during the initial manual aggressive app investigation, the automatic detection approach should address several challenges. In the following, we list these main challenges and present our corresponding insights to address them.

How to uncover location access and usages. It is challenging to automate the process of uncovering the behavior of location access (*i.e.*, the accuracy of the location and the frequency of retrieving up-to-date location) because of the diversified implementations in practice, such as using system APIs or customized methods that are specific to developers. Specifically, if an app uses system APIs, we need to resolve the input values that may not always be visible at the invocation of these APIs at the code level. As such, we have to trace these inputs across the whole app to obtain their values, and it could be challenging to precisely trace them across different components of an Android app. On the other hand, with respect to customized methods, it could be difficult to analyze them in a generic way when coming to large-scale analysis.

Fortunately, we find that it is popular (around 60% apps requesting location permission) among apps to use the system-provided APIs to periodically retrieve location data. Although inputs to these APIs may not always be directly visible, these inputs (e.g., `minDistance` and `minTime`) are in the form of `string`, `float`, and `long` whose value could be resolved by value set analysis (*VSA*) [23]. Therefore, without loss of generality, we choose to use VSA [23] to resolve the inputs to the dedicated system APIs allowing apps to receive location updates periodically. We acknowledge that some apps may use non-system APIs to retrieve Location data that are not covered by our method, we discussed this limitation in (§6).

How to recognize fine-grained LBSes. It is non-trivial to automate the process of recognizing the LBSes an app provides. LBSes in real-world apps are heterogeneous (*e.g.*, address auto-fill, weather

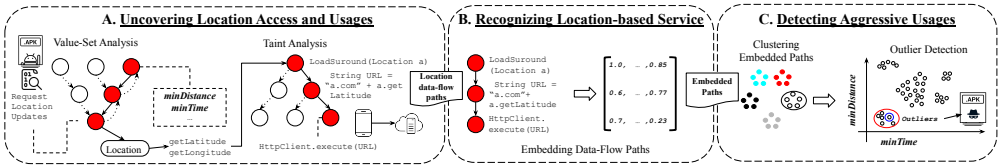


Fig. 1. The workflow of LocationScope.

forecasts, and local news), and they are often integrated deeply inside the primary functionalities of an app (*e.g.*, a car insurance app will geo-tag the photo a user takes to report an accident), or even serve as a hidden component (*e.g.*, user profiling or account login tracing). Thus, LBSes are often absent in the documentation of an app (*e.g.*, app store description, privacy policy, and UI text). Even in cases when an LBS is documented, its description could be too vague to distinguish different LBSes (*e.g.*, "*the app provides services based on user's location*" instead of explicit descriptions such as "*the app will use your location to find nearby churches and non-profit organizations*").

Therefore, we propose a new approach combining techniques from both program analysis and representation learning to recognize LBSes by determining the semantics of fine-grained location data usage at the code level without relying on the unreliable app documentation that is required in previous works [76]. The key observation is that different apps providing the same type of LBSes often have semantically similar location data-flow paths. For example, in the LBS using the user's current location to auto-fill the address and zip code, the corresponding data-flow path starts from fetching the user's latitude and longitude and then using these two pieces of data to invoke the Android API `Geocoder.getFromLocation` or `Geocoder.getFromLocationName` to obtain the user's address. In addition, this usage similarity enables the representation learning technique to capture the semantics of LBS-related APIs among different apps even with different functionalities and further ensures that the data-flow paths with similar semantics will have similar vectors for quantitative analysis. These unique features make it possible to detect LBSes through a combination of code analysis and representation learning.

How to identify aggressive collection. Similarly, it is also challenging to automatically identify whether an app is an aggressive location collecting one. It is hard to obtain all the domain knowledge required for judging whether a collection behavior is aggressive given the corresponding LBS.

We address this challenge by applying the outlier detection approach that looks for intrusive outlier location collection behavior among apps providing the same LBS. Its advantage is that we can avoid summarizing the specific requirements for an LBS regarding location collection as a standard for identifying aggressive behaviors, given the inconsistent implementations across apps for the same LBS and the variety of LBSes.

3.3 Scope and Assumption

In this study, we focus on the location data-flow paths starting from retrieving location data to sending them to the outside (*e.g.*, servers) that are implemented at the Java bytecode level in Android apps at scale. Other aberrant location access channels such as using `WebView` or native libraries are out of scope. In addition, our tool is resilient to many code obfuscation techniques (*e.g.*, renaming) but may fail in cases when system APIs in Java are obfuscated (*e.g.*, using reflection). Though we wish to extend support on the above cases, they are out of the scope of the current version of our framework. In addition, since this study would like to comprehensively analyze periodical location access in both spatial and temporal frequency dimensions, APIs (*e.g.*, Fused Location Provider APIs) that cannot provide both information is out of the scope of this study.

4 DETAILED DESIGN

This section presents the detailed design of LocationScope, which consists of three key phases as shown in Figure 1. Specifically, (*i*) in the first phase, it takes an app as input to uncover its associated

data-flow paths that indicate how a piece of location data is retrieved and processed (§4.1); (*ii*) in the second phase, it recognizes the location-based service by embedding its associated location data-flow path; and *(iii)* in the last phase, it identifies the *ALHA* by first clustering location-based service groups, where each group consists of apps providing such a service, based on the semantic similarity of the embedded location data-flow paths from its contained apps, and then applying an outlier detection algorithm over each location-based service group (§4.3).

4.1 Uncovering Location Data Usages

The objective of this phase is to uncover how an app uses its periodically retrieved location data. To this end, it has to understand how an app *(i)* retrieves location data periodically from the system and *(ii)* processes its retrieved data.

Understanding how apps retrieve location data. An app can configure how fast (*frequency*) to periodically retrieve the location data at which granularity (*accuracy*) based on system APIs provided by the Android. Since both accuracy and frequency are specified by configuring the input values to specific parameters (*i.e.*, `provider`, `minTime`, and `minDistance`) of relevant system APIs, the way an app retrieves location data can be understood by resolving those input values. Considering these values may not always be resolved at the place where the APIs are invoked in the code (they could be declared somewhere else), it is necessary to trace them across the whole body of the code to understand how they are generated, and then resolve their values accordingly.

While the technique of the value set analysis (VSA) [23] appears to be suitable for tracing a given variable in the code to calculate its possible values, it cannot directly apply in our work because it targets a different platform. As such, we build our own technique to accomplish this task. Specifically, the first step is to build an inter-procedural control-flow graph (ICFG) for all methods within an app where the nodes are the instructions and the edges indicate the corresponding control-flow transfers. Next, a target parameter (*e.g.*, `V0`) will be traced backward along this ICFG to its initialization place. Meanwhile, an inter-procedural data-dependency graph (IDDG) will be maintained to record all computations related to this parameter in the reverse sequence. When the tracing stops, associated computations that have been recorded in IDDG will be repeated in the correct order to resolve the value of the target parameter.

Uncovering how apps process location data. Having understood how an app retrieves the location data periodically from the system by resolving relevant parameter values, the next task is to uncover how such an app processes its retrieved location data within the app before being sent to the outside (*e.g.*, servers). Since this process can be represented as a path that starts from reading the up-to-date location data periodically from the operating system and ends at system APIs sending them out, this task could be formalized as identifying these location data-flow paths which are often treated as location leaking paths in the literature [19].

In particular, when using the Android APIs (§2.1) to retrieve location updates, the Android will associate a `Listener` to a `Location` object allowing such an object to retrieve periodical location updates as configured, or sending an `Intent` carrying the location data to a specific component for further processing. Since the `Listener` or the `Intent` is a parameter of the related system APIs, we would use the similar technique above to trace the parameter in the backward direction to identify the object, and then analyze the internal callback functions (*e.g.*, `onLocationChanged`) of that object to pinpoint the starting point where the location data will be processed (*e.g.*, `Location.getLatitude`). Finally, we apply static taint analysis by setting such points as taint sources and defining the taint sinks which indicates the collected data will be sent to the externals (*e.g.*, remote hosts), to capture location data-flow paths that are of our interest.

Example. Figure 2 shows a running example that illustrates the core workflow of this component. First, in step ❶, the app sets the minimum time interval as 0 second and the minimum distance interval as 0 meter for its request to the system to read the location data from the GPS provider without interruption. Next, in step ❷, according to the listener assigned to react to location updates, this component is able to identify an implicit location data-flow from the `gpsListener` to the callback of `onLocationChanged` method. In this callback, the location data is further processed by the method `loadSurround` in step ❸, and this method is invoked immediately in step ❹ where the location is used to generate a `string` value. Then, this generated `string` value goes into the method of `urlRequester.requestUrl` in step ❺ which is the end of this location data-flow. Finally, this component outputs this location leaking path in step ❻ to complete its task.

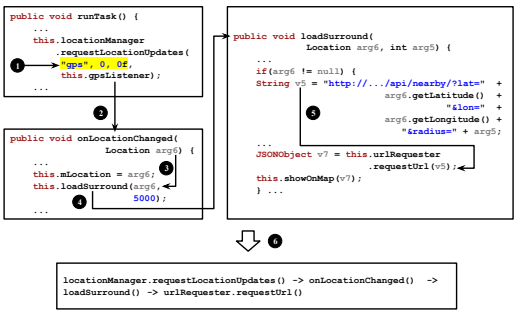


Fig. 2. Example of uncovering location data usages.

4.2 Recognizing Location-based Services

Having uncovered location data usages of apps, the second phase investigates why an app uses its retrieved location data in such a way from the perspective of its provided LBSes. To this end, in the absence of prior knowledge and the vague expressions of fine-grained LBSes in app documentations, it aims to recognize the associated LBSes from data-flow paths leaking location data that are uncovered in the previous phase. In particular, it adopts a novel approach that leverages techniques from representation learning which embeds each location data-flow path as a numeric embedding to abstract its LBS semantics.

Formalizing location data-flow paths. Considering each data-flow path that is uncovered in the first phase is outputted as a vector of functions, it is necessary to build a signature for each function to uniquely represent such a function to further ensure each data-flow path is distinguishable. To this end, the signature for each function is generated as a combination of its package name, class name, method name as well as its all parameter types, and the return value type, all of which are concatenated in this order with the symbol "`.`" for separation as `Package_name.Class_Name.Method_Name (Parameter_Type, ...).Return_Type`. Accordingly, a path c inside an app F_{app} can be denoted as $c = [f_1, f_2, ...]$, where f_i is the i_{th} function signature in the path c.

Standardizing functions in data-flow paths. At a high level, each function in a location data-flow path c could be either a custom function belonging to specific developers, $F_{Developer}$, or a system API, F_{API}, which could be standard Java or an Android API. While standard Java and Android APIs are rarely obfuscated because the relevant ramification might hinder normal functionality, custom functions are highly likely to be obfuscated (*e.g.*, renaming) which could affect the accuracy and reliability of its functional semantic extraction. To avoid this uncertainty, all functions in a location data-flow path will be standardized as standard Java or Android APIs following procedures shown in Algorithm 1. Specifically, it first extracts the call graph of a given app (line 2) where each node represents a function signature f and each directional edge from f_i to f_j represents that f_j is invoked inside f_i. When encountering any function signature f_i that is not in F_{API} (line 9), f_i will be substituted with the sequence consisting of all of the functions invoked inside the body of function f_i (line 17-20). Since a single round of substitution may also introduce functions belonging to $F_{DEVELOPER}$, an iterative substitution procedure is enforced until all the function signatures in

Algorithm 1 Developer-defined function translation

```
1: procedure TRANSLATE(c_app_i)
2:     G ← callgraph(app_i)
3:     L ← c_app_i                          ▷ Initial Location usage embedding
4:     F_translated ← [ ]                   ▷ For function detection
5:     while ∃f in L ∈ F_Developer do
6:         f_developer ← the first developer-defined function in L
7:         F_substitute ← [ ]
8:         for f_i in functions invoked by f_developer in G do
9:             if f_i ∈ F_Developer then
10:                if f_i ∈ F_translated then   ▷ Recursive detected
11:                    F_translated ← [ ]       ▷ Reset
12:                else
13:                    F_substitute ← f_i
14:                    F_translated ← append(f_i)
15:                end if
16:            else
17:                F_substitute ← append(f_i)
18:            end if
19:        end for
20:        subsitute f_i with F_substitute
21:    end while
22: end procedure
```

Algorithm 2 Location Usage Analysis Process

```
1: L_All ← {}                                ▷ The set of all call paths
2: for app_i in Apps in the same category do
3:     for c_i in callpath(app_i) do         ▷ Data flow call paths
4:         L_i ← Translate(c_i)
5:         L_temp ← [ ]                      ▷ Temporary variable to store new L_i
6:         for f in L_i do
7:             V_fi ← API_Embedding(f_i)
8:             L_temp ← append(V_fi)
9:         end for
10:        L_i ← GAP(L_temp)                 ▷ dimension reduction
11:        L_All ← add(L_i)
12:    end for
13: end for
14: Mapping(L_i, L_id_i) ← DBScan(L_All)     ▷ L_i belongs to L_id_i
15: for app_i in Apps in the same category do
16:    LBS_app_i ← {}                        ▷ Initial empty set
17:    for c_i in callpath(app_i) do
18:        L_i ← find c_i's corresponding L_i in L_All
19:        L_id_i ← Mapping(L_i, L_id_i)
20:        LBS_app_i ← add(L_id_i)
21:    end for
22: end for
```

this path c are functions from F_{API} (line 7-20). Note that recursive functions in $F_{DEVELOPER}$ are opted out of the substitution process to avoid infinite loop (line 10-11).

Embedding location data-flow paths. As mentioned in §3, it has been observed that similar sets of system APIs (F_{API}) including standard Java and Android APIs are often present in the location-data flow paths that implement the same LBS across different mobile apps, which indicates that a specific LBS could be recognized by the set of system APIs appeared in its associated data-flow path. Inspired by this observation, the semantic of an LBS can be represented by the combination of semantics of a set of system APIs.

Algorithm 2 shows the main procedures of the location data-flow path embedding. First, given an app, its location data-flow $c = [f_1, f_2, ...f_m]$ is extracted and further translated into a standardized function list (line 2-4). Since the standardized functions can be treated as word tokens, a representation learning algorithm (*i.e.*, word embedding) can be resorted to generate length-unified $1 \times N$ numerical vectors, which are then concatenated to construct an $M \times N$ matrix where M is the number of APIs in the standardized data-flow path (line 6-9). Considering the different number of system APIs in different data-flow paths, which makes the embedding matrix of each path in different shapes hindering similarity comparisons in the following steps, an $M \times N$ matrix will be finally squeezed to a $1 \times N$ numerical vector using the GAP technique [53] (line 10) by calculating the column-wise mean.

Example. The main procedures of this component are illustrated in Figure 3. First, a developer-defined function (*i.e.*, `loadSurround`) is standardized as a bunch of standard APIs presented in the left rounded rectangle in step ❶. Next, in step ❷, each standard system or Java API (*e.g.*, `Landroid/location/Location;->getLatitude()D`) is embedded following a trained model as a vector of N elements. Since this path contains M standard APIs, in step ❸, it first constitutes an $M \times N$ matrix, and then reduces this matrix to a $1 \times N$ vector in step ❹.

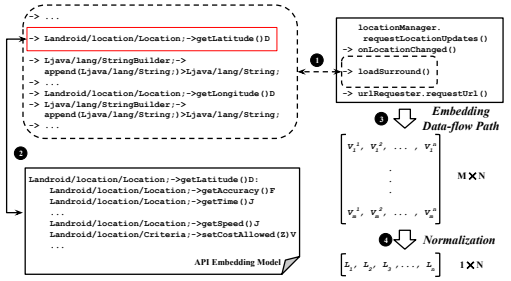


Fig. 3. Embedding location data-flow paths.

4.3 Detecting Aggressive Usages

Having uncovered location data usages and recognized LBSes via embedding their data-flow paths, the last component aims to detect mobile apps that aggressively retrieve location data in the wild. To this end, this component follows the line of privacy violation research that detects a violation in a mobile app, in the absence of outstanding and well-acknowledged standards, by comparing its consumption of privacy-related resources with its peer apps that provide similar functionalities (*e.g.*, AAPL [64]). Specifically, it proposes a novel method to identify peer apps of a target app by comparing their provided LBSes based on the associated semantic similarity. Next, within each group of apps, an aggressive app is identified by applying a semi-supervised classification method on the `minTime` and `minDistance` parameters configuring how an app retrieves location updates.

Clustering similar location-based services. Similar LBSes are identified by clustering their associated embedding data-flow paths based on their semantic similarities. Considering the total number of LBSes is unknown in advance, an unsupervised clustering method is preferred because it can automatically identify clusters of similar inputs without specifying the number of clusters; therefore, this component uses DBScan [38] as the clustering method. Specifically, the cosine similarity, which is defined as $S(L_i, L_j) = L_i \cdot L_j / (\|L_i\| \|L_j\|)$, is used as the distance metric between the two embeddings L_j and L_j. After the clustering, each L_i will be assigned a cluster id, ID_{L_i}. Since it is possible for an app to provide multiple LBSes at the same time, for example, an car insurance app can retrieve the location to auto-fill zipcode, and also geo-tag watermarks on photos of the vehicle to show its location, each app could be labeled with multiple LBS cluster ids. Accordingly, the LBSes in an app can be expressed as $LBS(app) = \{ID_{L1}, ID_{L2}, ...\}$.

Training location usage classification model. Having clustered similar LBSes, the next task is to detect the aggressive LBSes in each LBS cluster where we need to train a model. Specifically, it consists of the following steps to train a supervised classification model based on a training data set containing 32 aggressive apps and 32 non-aggressive apps and applies the trained model to detect aggressive LBSes within each cluster.

Pre-processing. The literature (*e.g.*, [43, 57]) has well acknowledged two special LBSes in apps: *advertisement* that are primarily used for increasing revenue by presenting advertisements, and *analytics* services that often help developers measure user activities and app performance. In particular, they are special because they work independently to other services that an app provides (*i.e.*, they are distributed via 3rd-party libraries, and the same library often behaves the same in different apps), and the majority of them have incentives to harvest locations [57] aggressively. Therefore, we profile their location retrieving behaviors separately and exclude them from the following classification steps since they cannot represent the purposed functionalities of an app. To recognize these services, it adopts a similar approach presented in [43] that first uses LibRadar [3] to identify 3rd-party libraries from an app and then compares these libraries with the list of known advertisement and analytics libraries that are released from reliable sources (*e.g.*, [11, 57]).

Data selection. The second step is to select training data apps to constitute two groups: a positive group and a negative group, to train the classification model for each LBS group. Specifically, the positive group is built with 32 aggressive apps we found during an initial manual investigation (Table 1), annotated following the aggressive app annotation guide (§3.1). In particular, annotators achieve a high agreement on these 32 aggressive apps, the Cohen's Kappa agreement measurement is 1.0. We chose 32 apps since 32 is the total number of aggressive apps we found during the initial manual exploration of the app samples. One could choose a larger number as more and more aggressive apps are discovered following our method, increasing the diversity and coverage of the training data. 32 is used in our case due to the limited labor capacity we had for analyzing

App Name	Provider			Freq.			Manual Inspection		
	GPS	Network	Passive	Time	Dist.	Has LBS	LBS Description	Need Loc.	Need Per.
56.FBS	✔	✔	✔	0	0	✔	Nearby POI	✔	✗
Affirmations	✔	✔	✗	0	0	✔	Nearby safety issues	✔	✗
Asian Turbo	✔	✗	✗	0	0	✔	Auto-fill address	✔	✗
Corq	✔	✔	✗	0	0	✔	Nearby events	✔	✗
Door Ding Defender	✔	✔	✔	0	0	✔	Wind speed	✔	✗
Easy Markets	✔	✔	✗	0	0	✗	-	✗	-
Emarits Halall	✔	✔	✗	10	0	✔	Search users in a city	✔	✗
Feira	✔	✔	✔	0	0	✗	-	✗	-
Goodwill Rewards	✔	✗	✗	0	0	✔	Restrict services	✔	✗
Great Date Ideas	✔	✔	✔	0	0	✗	-	✗	-
IFTA 2013	✔	✔	✗	0	0	✔	Location-related ad	✔	✗
Investing	✔	✔	✗	0	0	✗	-	✗	-
Mosquito Alert	✔	✔	✗	0	0	✔	Nearby mosquitos	✔	✗
NAO 1999	✔	✔	✗	0	0	✔	Location-related ad	✔	✗
Nature et aromes	✔	✔	✗	0	0	✔	Nearby branch	✔	✗
Ongo e-Money	✔	✔	✔	0	0	✔	Nearby stores	✔	✗
Playz	✔	✔	✗	0	0	✗	-	✗	-
Quadrant2	✔	✔	✔	0	0	✔	Help in emergency	✔	✗
Quaran For All	✔	✔	✗	0	0	✔	Direction to quaran	✗	-
Quatroamigos	✔	✔	✔	0	0	✔	Distance to others	✔	✗
RTP Noticias	✔	✔	✔	0	0	✔	Show on map	✗	-
Sabbar	✔	✔	✔	0	0	✔	Distance to others	✔	✗
Saloner	✗	✗	✔	0.1	0	✔	Nearby salons	✔	✗
SchoolCNXT	✔	✔	✗	0	0	✗	-	✗	-
Sikh World	✔	✔	✔	0	0	✔	Find gurudwara	✔	✗
Speaking Robots	✔	✔	✗	0	0	✔	-	✗	-
Spice Camera CS	✔	✔	✗	1	0	✔	Geo-tagging photos	✔	✗
Urban Usher Transportation	✔	✔	✗	0	0	✔	-	✗	-
Vivance	✔	✔	✔	0	0	✔	Show store location	✗	-
Wefun Reader	✔	✔	✗	1	0	✗	-	✗	-
Wooask	✔	✔	✔	0	0	✔	Distance to others	✔	✗
Yumi	✔	✔	✗	0	0	✔	Nearby users	✔	✗

Table 1. Details of the 32 ground truth aggressive apps. Freq: Frequency; Loc: Location; Dist: Distance; Per: Periodical Location Acquisition; ✔: presence; ✗: absence; -: N/A.

app samples. In respect of the negative group, it consists of 32 non-aggressive apps that are auto-selected from the target LBS cluster where there are 16 apps that specify the top 16 largest `minTime` values and the rest 16 apps that are configured with the top 16 largest `minDistance` values. The intuition behind this selection rule is that apps with the largest parameter values represent the most conservative location-collecting behavior, thus, most likely to be non-aggressive apps.

Feature engineering. The classifier uses two fundamental features (*i.e.*, `minTime` and `minDistance`) that describes the periodic location-retrieving behavior of an app. However, the preliminary study identifies that these two features vary in a scale ranging from extremely large ones (*e.g.*, 3,600,000 seconds) to small values (*i.e.*, 0 second) that request as fast as possible, which can affect the performance of the numeric sensitive classifier. To mitigate this, the following two normalization techniques are applied:

$$X_i = \Phi^{-1}(F(x_i)) \tag{1}$$

$$X_i = softmax(Max(X) - X_i) \tag{2}$$

The first equation is to re-scale values in the feature vector X to a uniform distribution in the range between 0 and 1 using the quantile normalization method [12]. Specifically, F and Φ are the cumulative distribution function of the feature vector itself and uniform distribution, respectively. The second equation is to apply the $softmax$ normalization technique [14] on the output of the first equation, and the outcome will be fed into the classifier. Since the second equation reversed the input by subtracting $Max(X_i)$ with X_i, comparing to large values, smaller `minTime` and `minDistance` values will be distributed in a more crowded feature range, such that limited effects, are induced in cases such as the time intervals are different in 1 or 2 seconds.

Post-processing. After processing data through the above procedures, a k-nearest neighbors classifier [9] is trained to identify aggressive behaviors, and we choose this algorithm based on its performance compared to other popular classification algorithms (details of this comparison are pushed to §5.2). While its direct classification results achieve high accuracy, an additional post-processing step is still taken to reduce false positives. These false positives result from the nature of both positive and negative example apps in the training phase where they may share the same value of `minTime` and `minDistance`, making the classifier ambiguous in the borderline examples. Therefore, it may mistakenly recognize a normal behavior as an aggressive one. Accordingly, this post-processing reduces such false positives by unflagging those recognized aggressive ones that contain more conservative behaviors (*i.e.*, larger feature values) than the negative ones.

Detecting aggressiveness. After applying the above method to each LBS group, apps' aggressive behaviors in each of its LBSes are identified. Since an app could contain multiple LBSes, it is possible that some of its LBSes are labeled as non-aggressive, and some labeled as aggressive in other groups. We could simply label an app as aggressive if any of its LBS is aggressive, but doing

so would introduce false positives. The reason is that multiple LBSes in the same app may rely on the locations obtained from the same location request (e.g., setting a global location request is a common programming pattern, we found 2924 aggressive apps providing more than one LBS using such a pattern), causing some LBSes' corresponding values of `minTime` and `minDistance` being affected by other more demanding LBSes in the same app. When the less demanding LBS is reusing the location request intended to serve the more demanding LBS, it will be detected as an aggressive LBS due to the higher precision and frequency required by the more demanding LBS; however, the app should not be considered as aggressive. For instance, the same location data request in a restaurant app is used to provide two LBSes: i) navigation to the nearest branch and ii) auto-filling zip-code based on the user's coarse location, and the second LBS should not be considered as aggressive even if it's using the more precise location information intended for navigation. As a result, to precisely identify aggressive apps, a strict and conservative policy is adopted. That is, an app will not be labeled as aggressive if any of its most precise tracking behaviors is not aggressive in the LBS cluster it belongs to.

5 EVALUATION

This section presents the evaluation and analysis of LocationScope. It first describes the setup, including experiment datasets, implementations, and environment (§5.1), next evaluates and analyzes the effectiveness of our proposed system (§5.2), and finally shows its novel findings (§5.3) from the detected aggressive location tracking apps with three case studies (§5.4).

5.1 Experiment Setup

Datasets. This work constitutes five different datasets in total, and the primary dataset is the *Google Play Free Apps Dataset (D_F)* which consists of 2.05 million free apps that are crawled from Google Play by the end of 2019 and 2.16 million free apps also crawled from the Google Play during the year of 2021. Derived from D_F, there are four datasets that are used for training location the usage classification model, evaluating the effectiveness of LBS recognition in two different levels, and evaluating the end-to-end aggressive app detection effectiveness. Specifically:

- *Ground-truth dataset of Aggressive Apps (D_A)*: D_A is constituted to understand the true positive and false positive (non-aggressive but falsely detected as aggressive) during the end-to-end evaluation of our aggressive app detection pipeline. It consists of 200 apps that are randomly selected from the aggressive apps detected in D_F, which took each annotator around 240 hours for labeling (following the aggressive app annotation guide 3.1), yielding the Krippendorff's alpha coefficient of 0.94.
- *Ground-truth dataset of non-aggressive Apps (D_N)*: D_N is constituted to assist in evaluating the false negatives (indeed aggressive apps but detected as non-aggressive) and true negatives in the end-to-end evaluation of the aggressive app detection pipeline. It included 100 apps randomly selected from apps not detected as aggressive by our pipeline. It took each annotator around 120 hours to label them following the annotation guide 3.1, yielding the Krippendorff's alpha coefficient of 1.0.
- *Ground-truth dataset for LBS Recognition at LBS-level (D_{LL})*: D_{LL} is constituted to evaluate the effectiveness of recognizing LBSes from their associated location data-flow paths. It consists of 266 location data-flow paths manually labeled by two annotators from 34 apps randomly selected in D_F, covering 83 LBS clusters. The annotators are asked to check whether two location data-flow inside the same LBS cluster are indeed used for the same LBS. Each annotator took around 60 hours to label apps, yielding the Krippendorff's alpha coefficient of 0.99, only samples labeled as correct by all annotators are considered correct.
- *Ground-truth dataset for LBS Recognition at App-level (D_{LA})*: D_{LA} is constituted to evaluate the effectiveness of recognizing the same LBS among different apps. It consists of 295 apps manually

labeled by two annotators randomly picked from D_F, which contains 157 LBSes. Annotators are asked to label whether two apps sharing the same LBS clusters apps provide the same set of LBSes. It took each annotator 70 hours for labeling, yielding the Krippendorff's alpha coefficient of 0.99, only samples labeled as correct by both annotators are considered correct.

Implementation. We have implemented a prototype of our proposed solution atop several tools instead of developing from scratch. Specifically, we optimized and tuned FlowDroid [19], Soot [15], and AndroGuard [5] to perform value set analysis and identify data-flows of location data leakages. In addition, we build a tool atop API2VEC [73] to generate the embeddings of Java and Android APIs to recognize location-based services. Specifically, we train the API embedding model over the decompiled code from the top 500 Android apps based on the number of their installs. This trained model embeds each system API as a 1×300 vector (the same as the default setting in the API2VEC [73] which has been proved as a great balance between efficiency and effectiveness). In addition, we also leverage scikit-learn [13] to cluster location-based services and identify outliers during aggressive app identification. In particular, in respect of parameters used in DBScan, we choose 0.9 for `eps` and 5 for `min_samples`.

Experiment environment. We conducted our experiment on several groups of machines with different computation power in parallel. Specifically, all experiments analyzing Android apps from 2019 were conducted on our workstations, which are equipped with AMD EPYC 7251 CPU with 256GB memory, and the rest of the experiments relating to code embedding processing were conducted on six desktops running with an Intel i7-7700 CPU and 32GB memory. In respect of apps from 2021, all experiments were conducted on two servers, each of which is equipped with two Intel E5-2695 v2 CPUs alongside 192GB memory.

5.2 Effectiveness

Comparison of different classification algorithms. As mentioned in §4.3, we selected the k-nearest neighbors as the algorithm to classify location retrieving behaviors because of its better performance compared to other popular classification algorithms. In this evaluation, we present the details of such a comparison to support our decision. Specifically, as shown in Figure 4, at first, 5 different algorithms (*i.e.*, Support Vector Machine(SVM) [50], DT [63], Quadratic Discriminant Analysis (QDA) [49], Naive Bayes(NB) [68], and K-Nearest Neighbor (KNN) [77] have been chosen to compare their performance over the training app dataset used in (§4.3). The performance metrics used are the F1-score and Accuracy. Figure 4 shows that the KNN methods have relatively high accuracy compared to other algorithms. As such, we choose KNN over other algorithms. In addition, we also compare the F1-score by assigning different k values (*i.e.*, 2, 3, and 4), and Figure 4 also shows that K=3 achieves the best performance. Therefore, we finally select k-nearest neighbors (k=3) as the classification algorithm in this work.

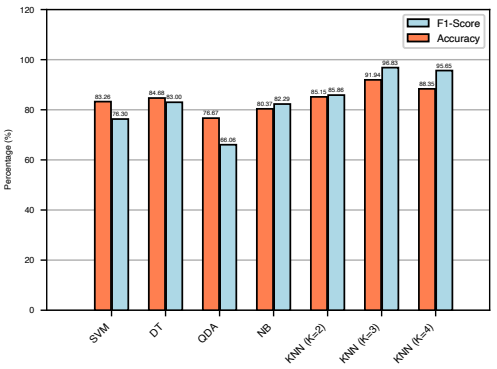


Fig. 4. Comparison of classification algorithms.

Effectiveness of LBS identification. The effectiveness of the main objective of this work, *i.e.*, identifying aggressive apps, is based on the accuracy of LBS recognition via embedding associated location data-flow paths. In order to evaluate the effectiveness of LBS recognition, we design a two-level (*i.e.*, LBS-level and app-level) evaluation method. In particular, the LBS-level evaluation

is to justify whether two location data-flows that are clustered into the same LBS group based on their embedding similarity actually represent the same LBS, and the app-level evaluation is to examine whether apps that are identified as sharing the same set of LBS clusters indeed provide the same LBSes. This two-level evaluation shows that the proposed algorithm is effective:

- *LBS-level evaluation*: Among the 266 location data flows 262 (98.4%) are correctly clustered, and the 4 wrongly clustered data flows that are clustered into the same cluster belong to 4 distinctive apps. Upon investigation, while these 4 data flows indeed share similar LBS, which draws markers on a map, they contain subtle differences. Specifically, two of them use the marker to indicate the user locations for real-time navigation, and the other two use the marker to indicate nearby point-of-interest (*POI*) locations without navigation.
- *App-level evaluation*: If two apps are labeled as providing the same set of LBS (*e.g.*, both apps use locations to provide directions to the store and auto-fill zip code), both apps are true positive apps (TP); otherwise, they are both false positives (FP). The results show a precision of 90.5% where there are 28 FPs and 267 TPs. Further analysis shows that 22 out of the 28 FPs are apps never trigger the paths identified by our tool (*i.e.*, dead codes), and the rest 6 FPs are from groups that are loosely clustered.

Effectiveness of aggressive app identification. Since we depend on an automatic pipeline to identify aggressive apps, its effectiveness needs to be evaluated. It turns out that in 200 apps that are detected as aggressive in D_A, 194 apps are true positives, showing a 97% precision ($\frac{TP}{TP+FP}$). In respect of the 6 false positives, all of them are due to the limitation in the FlowDroid where paths are not captured completely, aligned with the observation made in [27], resulting in the LBS not being captured by our pipeline. Among 100 apps that are detected as non-aggressive in D_N, 14 apps are actually aggressive (14 false negatives and 86 true negatives), showing a false negative rate ($\frac{FN}{FN+TP}$) of 6.7%. All 14 false negatives do not rely on location to work but still collect locations (*e.g.*, a puzzle game that does not rely on the geo-location of the user to work properly) without interruption (`minTime` = 0 and `minDistance` = 0). Our pipeline did not capture them since 0 is already the maximum `minTime` and `minDistance` value used by apps in their LBS cluster.

Run-time performance. Among three components in our proposed workflow, the most time-consuming part is uncovering location access and usages, which include running value set analysis to resolve location collection parameters and taint analysis using FlowDroid [19] to detect location leakage data-flows where its default settings and a 10-minute timeout strategy are used. Particularly, it took around 1 week and 3 weeks to conduct value set analysis and taint analysis on apps collected in 2019 and almost 4 days and 7 days on apps in 2021 due to the differences in computation power of the testing environments. In respect of the other two components, it took around 3 days to train the embedding model, and then used this model to embed location leakage data-flow paths in 3 days and 1 day for apps from 2019 and 2021 respectively, and an additional roughly 20 hours to cluster embedded paths and detect outliers in apps from both datasets.

5.3 Findings

5.3.1 ***Overall Statistics.*** As shown in Table 2, we have identified 590,599 apps that request permissions for fine-grained location data access from totally 2,164,859 free apps collected as of 2021. Compared to apps in 2019, the number of apps requesting fine-grained location permissions has increased by 15.03%, alongside the 5.04% growth of free apps. In addition, within the apps from 2021 with fine-grained location access permissions, we have identified 337,790 (57.20%) apps that register system APIs to periodically retrieve location updates, a 12.43% growth compared to apps from 2019. Moreover, among these periodical location collecting apps from 2021, we found 50,178 apps that can potentially leak users' location (*e.g.*, sending them to an app's back-end) where 14,091 of them have been recognized as aggressive apps. Interestingly, there is a decrease in

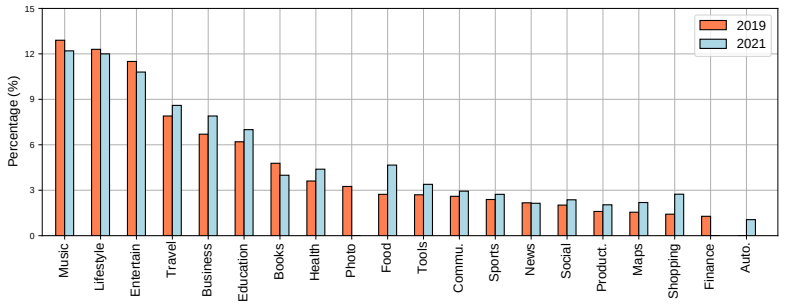


Fig. 5. The distribution of the aggressive apps.

the number of apps associated with location leakage and recognized aggressive apps, 48.93% and 63.58%, respectively. The main reason for this decrease is the increased failure rate of FlowDroid in location leakage detection, *i.e.*, the failure rate increased from 42.8% to 73.5% in a 10-minute timeout. FlowDroid cannot finish finding paths in more than 90% of failed cases, and the rest are due to the insufficient memory space assigned to run each app (*i.e.*, 20 GB). These observations comply with related works [20, 97], which pointed out the increase in time consumption and memory space requirement of FlowDroid when analyzing new and larger Android apps. Our study reveals the growing trend of aggressive location harvesting in Android apps through a longitude study of the aggressive apps presented later in this section.

Item	2019	2021
# Apps Collected	2,055,822	2,164,859
# Apps W/ Fine Loc. Permission	513,428	590,599
# Apps Register Loc. Updates	301,341	337,790
# Apps W/ Loc. Leakage	97,870	50,178
Avg # LBSes/App	2.11	1.39
# Aggr. Apps	38,688	14,091
Avg # LBSes/App	1.06	1.57

Table 2. Overall Statistics.

5.3.2 *Aggressive Location Collection Apps.*

Distribution and popularity. Aggressive apps are not distributed uniformly across different categories, and there is a slight shift from 2019 to 2021. In 2019, 34 categories contain aggressive apps, the number dropped to 33 in 2021. As Figure 5 depicted, in both 2019 and 2021, the same 6 categories together account for more than 50% of aggressive apps and there are 12 categories each contributing between 1% to 5% of aggressive apps. While the number of categories containing less than 1% apps dropped from 17 to 15 from 2019 to 2021.

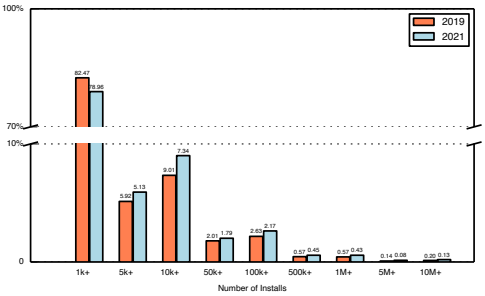


Fig. 6. Popularity comparison of aggressive apps.

With respect to the popularity (Figure 6), we have identified aggressive apps with up to 5 million installs, and all these apps together can impact up to 2.5 billion users globally based on their install numbers in the Google Play. Moreover, it shows that the installs of aggressive apps shift forward from 2019 to 2020, indicating that aggressive apps are becoming more popular than before. Surprisingly, we have identified a popular app (5 million installs) lasting from 2019 to 2021, `com.androidlost`, which appears to report the device location to the device owner when its battery level is low or received commands sent from Google Cloud Messaging Service or SMS messages, actually has a background location collection service that can be triggered by a hidden command "startrack". This hidden command is only available to the app developer, making its location collection behavior not only aggressive but also suspicious.

Contributor of aggressive apps. In this study, we have found three primary contributors involved in developing and deploying aggressive location harvesting apps including independent app developers, third-party LBS providers, and app generators. First, we have identified a surprising number of aggressive apps that are generated by app generators. In particular, these generators are identified by extending the relevant list that is collected in 2018 [74] and following its method to search for up-to-date generators that are available in 2020 with the help of a variety of search engines (*e.g.*, Google). In total, we have identified ten app generators contributing 7,728 (58.84%) aggressive apps in 2021, and there are nine app generators leading to 25,315 (65.4%) aggressive apps in 2019. We elaborate on several case studies in Section 5.4. Second, we have also found cases where apps rely on third-party LBS providers to provide LBSes, and the aggressive location harvesting in these providers is integrated into many similar apps. For example, there are five aggressive apps in the `Weather` category that use `Baron Service` [8], a weather service provider, to provide their weather-relevant services. Specifically, the `Baron Service` provides weather solutions to developers by giving out ready-to-use code packages, acting as a third-party library provider, but its main business focuses are on other areas instead of software services.

Associated LBSes. In this study, we have identified 14,091 in 2021 and 38,688 in 2019 mobile apps providing app-dependent LBSes that violate the minimum usage policy of location usage. To better understand the LBS that are related to aggressive tracking, we manually analyzed the randomly selected 100 aggressive apps following the distribution of apps in Google Play in both years (detailed results are listed in Table 3 in Appendix). At a high level, based on their provided LBSes, these apps contain 6 popular LBSes: presenting the weather, filtering nearby users (*e.g.*, distance-based), searching local events and spots, geo-tagging, detecting speed, and pushing location-based notifications. In regards to the distribution, as presented in Figure 7, the LBSes providing services of local searching, in which only one request for location data is adequate when users search relevant information, are shown to be more prone to contain aggressive data-accessing behaviors. Other LBSes that require periodical access to location data, such as speed monitoring and location-based notification pushing, are classified as aggressive since they are showing higher aggressiveness on accessing the data, compared with their corresponding peer apps (*e.g.*, updating speed every few seconds is enough in apps not targeted for driving navigation and location-based notification can be pushed when the distance interval is 1,000 m). Interestingly, we even found a set of apps providing no LBSes that users can notice. Basically, these inspected apps usually have no requirement for high-accurate data, and it does not affect the search results if changing the geo-location of the tested Android device to another place within one mile.

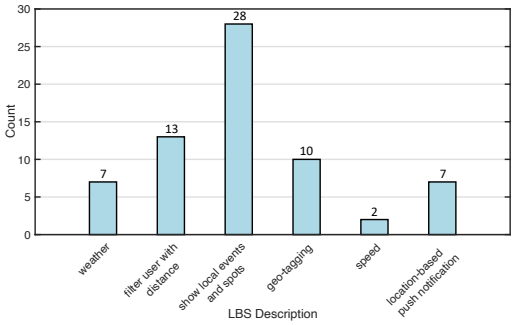
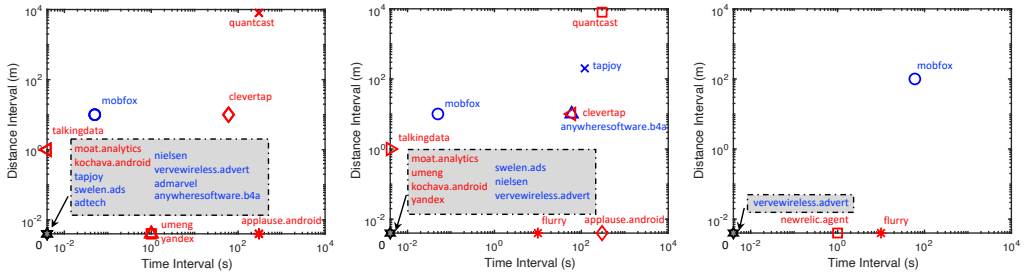


Fig. 7. Distribution of the verified aggressive LBSes.

Shift of aggressive apps from 2019 to 2021. In the longitudinal study, we studied 7,650 apps being flagged as aggressive either in 2019 or 2021, and check how these apps evolve from aggressive to non-aggressive or vice versa. Specifically, 4,982 (65.12%) apps remain aggressive from 2019 to 2021, 691 apps were aggressive only in 2019 (referred to as *AN* (aggressive to non-aggressive)), and 1,977 apps were non-aggressive in 2019 but become aggressive in 2021 (referred to as *NA*). This indicates that the number of apps that aggressively collect locations in the last 3 years is 2 times



(a) GPS **(b)** Network **(c)** Passive

Fig. 8. The visualization of the top 10 A&A libraries w.r.t time and distance intervals using different providers. Blue and red markers denote advertisement and analytics libraries, respectively, and the dashed boxes contain libraries setting both intervals to zero.

larger than the number of apps eliminating that behavior. In respect of app categories, we found that 31 out of 37 categories have a growing number of aggressive apps from 2019 to 2021.

Note that, regarding apps in *AN*, we found 593 of the 691 apps actually have not changed their parameter values of `minTime` and `minDistance`. They are not flagged as aggressive in 2021 because apps providing the same LBS are trending towards collecting more accurate location data, causing the detection model to shift the classification boundary. According to our measurement result, there are 2,139 apps in 2019 that shared the same LBS, and only 904 apps used 0 as parameters value, but there are 1,783 apps sharing the same LBS in 2021 and 1,235 apps use 0 as their location retrieving parameters.

Privacy policies. Recent works on privacy policy compliance have developed methods to automatically check policy compliance of a given apk [18]. However, to the best of our knowledge, no tools can check whether the policy quantitatively clarifies the precision and frequency of location data they collect. To better understand the policy compliance of identified aggressive apps, we studied the privacy policy of all 194 verified aggressive apps identified in our study, and check whether they mentioned the frequency and accuracy of location collection. We collected the privacy policy of identified aggressive apps following the privacy policy links displayed in the Google Play store and provided in the app. We manually checked all the collected policies, and only 53 apps from 2019 and 55 apps from 2021 have a privacy policy that acknowledged the user the existence of location collection behavior in the app. However, we found that *none* of these policies explicitly documented the frequency of location data collection in the corresponding apps. This raises concerns about refining current privacy policy enforcement. We believe that a privacy policy that quantitatively clarifies the precision and frequency of collected location data is beneficial to enforcing the data minimization principle clarified in privacy policies, such as GDPR [44] and CCPA [28].

A&A Libraries. As mentioned earlier (§4.3), A&A libraries create an independent code space apart from the other program code in an app. While we have excluded them from our results because of their homogeneous behavior across different apps, considering they are also aggressive participants in terms of location collection, we also present a corresponding analysis. In particular, we have identified 77 advertising libraries and 12 analytics libraries that collect user location periodically, and the majority of these libraries request location data at the fastest rate. We visualize the location collection behaviors of the top 10 A&A libraries in Figure 8 and push the details in Table 4 in the Appendix. It can be observed from Figure 8 that popular advertising libraries such as "tapjoy", "admarvel", and "adtech" and analytics libraries, *e.g.*, "moat.analytics", "kochava", and "umeng" intend to obtain location without interruption. In addition, some libraries use the same parameter settings in location services provided by different providers, *e.g.*, "clevertap" set the parameters to $(60, 10)$

in both GPS and Network services. Moreover, the advertising library "vervewireless" seems to be the most aggressive since it continuously requests location data from the three sources all the time.

5.4 Case Studies on App Generators

Figure 9 shows how apps generated by app generators distribute across different mainstreaming parties. Alongside observations from a prior study [74], we also find that app generators usually generate apps using the same boilerplate code. For a better understanding, we reverse-engineered several aggressive apps created by three generators which appeared in both 2019 and 2021 aggressive apps with our best understanding.

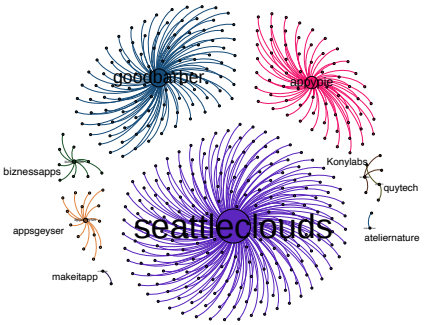


Fig. 9. App generators of aggressive tracking apps.

***(I)* Biznessapps.** Apps generated by this generator ask users for location data access with a user consent of *"to receive location-specific push notifications and other features"* in a pop-up window right after the first start. However, the claimed push notification service is implemented using the obsolete Google Cloud Messaging service which has been deprecated since April 10, 2018, and stopped after May 29, 2019. We found that the collected locations are actually used for data analytics. Specifically, whenever the user takes an action in the app, the location data will be automatically sent to the remote host owned by Biznessapps.

***(II)* Goodbarber.** Similarly, apps generated by Goodbarber also ask for location permission right after the first start. Unlike Biznessapps, Goodbarber does not send location upon every action but collects and shares multiple location data periodically with its host. Upon investigation, we found the location data were encoded in the body of a `POST` request and used for user geo-distribution profiling, i.e., counting the number of users in each country. Since such geo-distribution profiling works only at the country level, the collection of accurate geo-location coordinates using the highest precision without interruption is obviously unnecessary.

***(III)* Appsgeyser.** Unlike the above two generators, this app generator uses different boilerplate codes for apps delivering different functions. Interestingly, we discovered that its generated apps always ask for location permissions right after the first start regardless of whether an app provides LBSes. This is because Appsgeyser has packed advertisement libraries that need to access location data into its generated apps. In addition, advertisement is the major profit revenue of Appsgeyser [4].

6 DISCUSSION AND FUTURE WORK

Limitations. Our proposed framework is built atop FlowDroid [19] and AndroGuard [5], and inevitably inherits limitations from these two tools that could result in missing data flows, such as insufficient modeling on reflective calls, Android runtime framework and asynchronous callbacks, and non-Java level operations (*e.g.*, `WebView` and native code). Although recent works [42], [87], [58], and [55] enhanced static analysis regarding the native code, reflection call, and hybrid-apps in Android, their techniques are not straightforward to be combined together as a systematic analysis tool and generate sound call graphs. Also, false negatives can come from our conservative parameters chosen in selecting negative model training samples, which is based on the assumption that the majority of apps are benign. While it holds true in many categories, it may fail in certain cases. Although their consequent false negatives are even difficult for experts to be noticed due to the lack of consensus on the understanding of data minimization in location privacy, we still treat it as one of our limitations. In addition, our solution to recognize LBSes based on its data flows within apps, missing flows on the inaccessible server-side may lead to false positives, although it achieves 90% recognition precision. Moreover, as mentioned in §4.3, our strict policy could result in

false negatives when a shared global location object is used for multiple LBS with different location needs. Since we aim at reducing false positives, this compromised design is acceptable.

Latest location privacy countermeasures in Android. Android 10 and above support a new permission `ACCESS_BACKGROUND_LOCATION` [46]) which is to forbid apps, if not being granted this permission, from accessing location in the background (running invisibly to the users). However, it requires the developers to compile their app targeting Android 10 (API level 29) or higher to enforce the permission. Apps compiled targeting Android 9 or lower holding `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION` will be automatically granted the new permission on devices running Android 9 (or lower) and devices who are upgraded to Android 10. Among the 14,091 detected aggressive apps, there are 10,349 apps (73%) that are built for Android 9 (or lower), which remain unaffected by this new background location limitation mechanism. Besides, even the rest 3,742 apps which are built for Android 10 (or higher) can still aggressively collect location data through either: *(i)* requesting the new background permission or *(ii)* creating a Android foreground service by setting `android:foregroundServiceType=''location''` [6]. In particular, among these 3,742 apps, 353 apps have requested the new permission and another 117 apps have created such a foreground service. Therefore, there are 10,819 aggressive apps (76%) that retain effective location collection behavior against this new restriction mechanism on background location access.

Apart from increasing permission granularity, Android 11 and 12 introduced new system dialogues [7] during the location permission granting process that give the users extra options to grant "only this time" and coarse location access, ignoring the permissions an app has requested. Particularly, if chosen, these extra options grant apps temporary location access that is only valid until the app exits, and apps need to ask for user authorization again when they were started the next time. Studying how effective such extra options are in preventing aggressive location collection is out of the scope of this paper and left as future work since it will require collecting users' behaviors when using the Android system.

Root causes and mitigation. LBS in the identified aggressive apps either comes from third parties (e.g., app generators and service provider SDK) or from app developers' own implementation. In particular, while third-party tools and libraries can significantly assist developers in building mobile apps or increase revenue, default values used for location collection by those tools could be beyond necessity. As such, app developers should carefully configure the location access behaviors in these tools. Additionally, although many third parties claim only to collect locations in anonymity, it is still possible for them to track users from the anonymous data [93, 99]. Therefore, restrictions on these types of collections should be suggested. Moreover, we would also like to suggest mobile app markets (*e.g.*, Google Play) to adopt more robust app vetting techniques to identify these aggressive apps and protect their users' privacy. This is because, first, apps may have illicit purposes, even reliable apps may accidentally acquire the accurate locations without awareness, and second, mobile operating systems may not be willing to provide a restriction mechanism allowing users to configure the location access frequency due to user experience or usability reasons.

The principle of data minimization. The principle of data minimization has been clearly stated in recently released privacy regulations, including GDPR [44], CCPA [28], and ACLU [47]. The proper enforcement of this principle is believed to be able to suppress the aggressive location data collections in the mobile platform; however, our study on the apps from 2019 to 2021 has not noticed its effectiveness yet. A potential reason is that there lacks proper techniques for detecting such violations. To facilitate, our work provides the *first* approach to detect the violation of this usage principle in mobile apps. Moreover, our solution can recognize which party, the app developer or third-party libraries providers, actually violates this principle, which is believed to be beneficial for relevant authorities to enforce the privacy regulation on the corresponding party. We would like to

use our proposed solution to conduct a further longitudinal study to track the effectiveness of this principle over the foreseen years.

Responsible disclosure. We have reported to the Google Play store about all the aggressive apps we found during our study. As of December 2022, we are still awaiting a response. Considering the number of aggressive apps found, reporting to app developers takes time and is our ongoing work.

7 RELATED WORK

Detection and defense for location privacy in mobile apps. There is a large body of works that are able to detect location data leaks in Android apps. For example, Flowdroid [19], IccTA [56], and Amandroid [95] track data-flows in general, while Extractocol [30] and WARDroid [66] specifically focus on network data-flows. Additionally, there are works [26, 33, 54, 67, 78, 81, 86, 96, 99–102] that focus on location leaks from other perspectives. On the other hand, there are also many works that propose defense mechanisms. For instance, MockDroid [25] and LP-Guardian [39] restrict apps access to location data, PlaceMask [1] fakes users' locations, and Caché [17] enforces apps to access coarsened locations. Inspired by these works, our work focuses on the location-related data-flows to identify aggressive fine-grained location collections in Android apps at scale.

A&A library analysis. There are numerous works that studied security and privacy issues in A&A libraries, such as Stevens *et al.* [91]. In addition, Gibler *et al.* [45] studied security and privacy in advertisement libraries, Ashford analyzed privacy issues related to A&A libraries in free apps [2], and Crussell *et al.* [32] investigated advertisement frauds. Several other works [22, 57, 59, 65] proposed effective algorithms to detect A&A libraries, and mitigation to such privacy issues has been studied in [51, 60]. Complementary to these works, our study is beyond the location tracking behaviors in the A&A libraries and focuses on other app-specific location services.

Code embeddings. Code embeddings have been explored in the field of software engineering and program language in a variety of granularity, such as binary code [24, 83], code token [16, 21], and single method [34, 69]. In addition, this technique has also been applied in binary code representation (*e.g.*, DeepBinDiff [37], Genius [41], Gemini [98], Asm2Vec [36]), code similarity comparison [103], and cross-platform code comparison [98]. Moreover, there are also works that took advantage of embedding generation techniques in forensic analysis. For instance, Attack2Vec [89] attempts to understand cyber-attacks, DeepMem [90] generates abstract representations for kernel objects from memory dumps, and Log2Vec [48]) represents user actions from logs. These works are orthogonal to our work but use embedding techniques to generate the representation of program semantics.

8 CONCLUSION

We have presented a system, LocationScope, to automatically identify mobile apps that aggressively harvest users' locations on a large scale. In particular, we annotate and release the first aggressive app dataset, and present a suit of novel techniques to detect those apps automatically. We have tested LocationScope with millions of free apps from Google Play and conducted a longitudinal study on 38,688 and 14,091 detected apps with aggressive location harvesting, as of 2019 and 2021, respectively. Additionally, we identify the growing trend of aggressive apps from 2019 to 2021 and find the app generators as major contributors to such apps. Our findings made the first step towards better understanding and detection of this new privacy risk.

ACKNOWLEDGMENTS

We thank our shepherd Dr. Luoyi Fu and anonymous reviewers for their insightful and constructive comments. This work was partly supported by NSF awards 2112471, 1850725, CityU APRC grant 9610563, and CityU SRG-Fd grant 7005853. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of the supported organizations.

REFERENCES

[1] 2008. PlaceMask - Protecting Location Privacy. http://www.placemask.com/

[2] 2012. Free mobile apps a threat to privacy, study finds. https://www.computerweekly.com/news/2240169770/Free-mobile-apps-a-threat-to-privacy-study-finds

[3] 2018. pkumza/LibRadar: LibRadar - A detecting tool for 3rd-party libraries in Android apps. https://github.com/pkumza/LibRadar

[4] 2020. App Monetization Lifehacks: Everything you wanted to know about the revenue from free apps. https://appsgeyser.com/blog/monetization-lifehacks-everything-you-wanted-to-know-about-the-revenue-from-apps-with-appsgeyser/

[5] 2022. androguard/androguard: Reverse engineering, Malware and goodware analysis of Android applications ... and more (ninja !). https://github.com/androguard/androguard

[6] 2022. Android Foreground Service Location Label. https://developer.android.com/training/location/permissions

[7] 2022. Android Location Permission. https://developer.android.com/training/location/permissions#request-location-access-runtime

[8] 2022. Baron Service. https://www.baronweather.com.

[9] 2022. k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[10] 2022. Location Strategies | Android Developers. http://developer.android.com/guide/topics/location/strategies.html.

[11] 2022. Monetize, advertise and analyze Android apps | AppBrain.com. https://www.appbrain.com/

[12] 2022. Quantile normalization. https://en.wikipedia.org/wiki/Quantile_normalization.

[13] 2022. scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation. https://scikit-learn.org/stable/

[14] 2022. Softmax function. https://en.wikipedia.org/wiki/Softmax_function.

[15] 2022. soot-oss/soot: Soot - A Java optimization framework. https://github.com/soot-oss/soot

[16] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29.

[17] Shahriyar Amini, Janne Lindqvist, Jason Hong, Jialiu Lin, Eran Toch, and Norman Sadeh. 2011. Caché: caching location-enhanced content to improve user privacy. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*. 197–210.

[18] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. 2020. Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck. In *29th USENIX Security Symposium (USENIX Security 20)*. 985–1002.

[19] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.

[20] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining apps for abnormal usage of sensitive data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 426–436.

[21] David Azcona, Piyush Arora, I-Han Hsiao, and Alan Smeaton. 2019. user2code2vec: Embeddings for profiling students based on distributional representations of source code. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*. 86–95.

[22] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 356–367.

[23] Gogul Balakrishnan and Thomas Reps. 2004. Analyzing memory accesses in x86 executables. In *International conference on compiler construction*. Springer, 5–23.

[24] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. 2018. Neural code comprehension: A learnable representation of code semantics. In *Advances in Neural Information Processing Systems*. 3585–3597.

[25] Alastair R Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. 2011. Mockdroid: trading privacy for application functionality on smartphones. In *Proceedings of the 12th workshop on mobile computing systems and applications*. 49–54.

[26] Claudio Bettini, X Sean Wang, and Sushil Jajodia. 2005. Protecting privacy against location-based personal identification. In *Workshop on Secure Data Management*. Springer, 185–199.

[27] Richard Bonett, Kaushal Kafle, Kevin Moran, Adwait Nadkarni, and Denys Poshyvanyk. 2018. Discovering flaws in security-focused static analysis tools for android using systematic mutation. In *27th USENIX Security Symposium (USENIX Security 18)*. 1263–1280.

[28] CCPA. 2019. California Consumer Privacy Act. https://reciprocity.com/california-consumer-privacy-act-ccpa/.

[29] Anne SY Cheung. 2014. Location privacy: The challenges of mobile service devices. *Computer Law & Security Review* 30, 1 (2014), 41–54.

[30] Hyunwoo Choi, Jeongmin Kim, Hyunwook Hong, Yongdae Kim, Jonghyup Lee, and Dongsu Han. 2015. Extractocol: Automatic extraction of application-level protocol behaviors for android applications. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 593–594.

[31] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. 2017. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis.. In *NDSS*.

[32] Jonathan Crussell, Ryan Stevens, and Hao Chen. 2014. Madfraud: Investigating ad fraud in android applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. 123–134.

[33] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. 2013. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports* 3 (2013), 1376.

[34] Daniel DeFreez, Aditya V Thakur, and Cindy Rubio-González. 2018. Path-based function embedding and its application to specification mining. *arXiv preprint arXiv:1802.07779* (2018).

[35] Google Developers. 2022. Fused Location Provider API. https://developers.google.com/location-context/fused-location-provider.

[36] Steven HH Ding, Benjamin CM Fung, and Philippe Charland. 2019. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 472–489.

[37] Yue Duan, Xuezixiang Li, Jinghan Wang, and Heng Yin. 2020. DEEPBINDIFF: Learning Program-Wide Code Representations for Binary Diffing. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS'20)*.

[38] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.

[39] Kassem Fawaz, Huan Feng, and Kang G Shin. 2015. Anatomization and Protection of Mobile {Apps'} Location Privacy Threats. In *24th USENIX Security Symposium (USENIX Security 15)*. 753–768.

[40] Kassem Fawaz and Kang G Shin. 2014. Location privacy protection for smartphone users. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 239–250.

[41] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. 2016. Scalable graph-based bug search for firmware images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 480–491.

[42] George Fourtounis, Leonidas Triantafyllou, and Yannis Smaragdakis. 2020. Identifying java calls in native code via binary scanning. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 388–400.

[43] Julien Gamba, Mohammed Rashed, Abbas Razaghpanah, Juan Tapiador, and Narseo Vallina-Rodriguez. 2020. An analysis of pre-installed android software. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1039–1055.

[44] GDPR. 2022. Art.5: Principles relating to processing of personal data. https://gdpr-info.eu/art-5-gdpr/.

[45] Clint Gibler, Ryan Stevens, Jonathan Crussell, Hao Chen, Hui Zang, and Heesook Choi. 2013. Adrob: Examining the landscape and impact of android application plagiarism. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. 431–444.

[46] Inc Google. 2022. Android location in the background. https://developer.android.com/training/location/background

[47] Jennifer Stisa Granick. 2020. Apple and Google Announced a Coronavirus Tracking System. How Worried Should We Be? https://www.aclu.org/news/privacy-technology/apple-and-google-announced-a-coronavirus-tracking-system-how-worried-should-we-be/.

[48] Wenbo Guo, Dongliang Mu, Xinyu Xing, Min Du, and Dawn Song. 2019. {DEEPVSA}: Facilitating Value-set Analysis with Deep Learning for Postmortem Program Analysis. In *28th USENIX Security Symposium (USENIX Security 19)*. 1787–1804.

[49] Peter E Hart, David G Stork, and Richard O Duda. 2000. *Pattern classification*. Wiley Hoboken.

[50] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and their applications* 13, 4 (1998), 18–28.

[51] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. 2011. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*. 639–652.

[52] Ethan Katz-Bassett, John P. John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. 2006. Towards IP Geolocation Using Delay and Topology Measurements. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC '06)*. ACM, New York, NY, USA, 71–84. https://doi.org/10.1145/1177080.1177090

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and

K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[54] John Krumm. 2007. Inference attacks on location tracks. In *International Conference on Pervasive Computing*. Springer, 127–143.

[55] Sungho Lee, Julian Dolby, and Sukyoung Ryu. 2016. HybriDroid: static analysis framework for Android hybrid applications. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 250–261.

[56] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. 2015. Iccta: Detecting inter-component privacy leaks in android apps. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 280–291.

[57] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. An investigation into the use of common libraries in android apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 403–414.

[58] Li Li, Tegawendé F Bissyandé, Damien Octeau, and Jacques Klein. 2016. Droidra: Taming reflection to support whole-program analysis of android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 318–329.

[59] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. 2017. LibD: scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 335–346.

[60] Bin Liu, Bin Liu, Hongxia Jin, and Ramesh Govindan. 2015. Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the 13th annual international conference on mobile systems, applications, and services*. 89–103.

[61] Bo Liu, Wanlei Zhou, Tianqing Zhu, Longxiang Gao, and Yong Xiang. 2018. Location privacy and its applications: A systematic study. *IEEE access* 6 (2018), 17606–17624.

[62] Dachuan Liu, Xing Gao, and Haining Wang. 2017. Location privacy breach: Apps are watching you in background. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2423–2429.

[63] Wei-Yin Loh. 2014. Fifty years of classification and regression trees. *International Statistical Review* 82, 3 (2014), 329–348.

[64] Kangjie Lu, Zhichun Li, Vasileios P Kemerlis, Zhenyu Wu, Long Lu, Cong Zheng, Zhiyun Qian, Wenke Lee, and Guofei Jiang. 2015. Checking more and alerting less: detecting privacy leakages via enhanced data-flow analysis and peer voting.. In *NDSS*.

[65] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. LibRadar: fast and accurate detection of third-party libraries in Android apps. In *Proceedings of the 38th international conference on software engineering companion*. 653–656.

[66] Abner Mendoza and Guofei Gu. 2018. Mobile application web api reconnaissance: Web-to-mobile inconsistencies & vulnerabilities. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 756–769.

[67] Joseph Meyerowitz and Romit Roy Choudhury. 2009. Hiding stars with fireworks: location privacy through camouflage. In *Proceedings of the 15th annual international conference on Mobile computing and networking*. 345–356.

[68] Marvin Minsky. 1961. Steps toward artificial intelligence. *Proceedings of the IRE* 49, 1 (1961), 8–30.

[69] Vijayaraghavan Murali, Letao Qi, Swarat Chaudhuri, and Chris Jermaine. 2017. Neural sketch learning for conditional program generation. *arXiv preprint arXiv:1703.05698* (2017).

[70] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why do developers get password storage wrong? A qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 311–328.

[71] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. 2015. Uipicker: User-input privacy identification in mobile applications. In *24th USENIX Security Symposium (USENIX Security 15)*. 993–1008.

[72] Yuhong Nan, Zhemin Yang, Min Yang, Shunfan Zhou, Yuan Zhang, Guofei Gu, Xiaofeng Wang, and Limin Sun. 2016. Identifying user-input privacy in mobile applications at a large scale. *IEEE Transactions on Information Forensics and Security* 12, 3 (2016), 647–661.

[73] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N Nguyen. 2017. Exploring API embedding for API usages and applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 438–449.

[74] Marten Oltrogge, Erik Derr, Christian Stransky, Yasemin Acar, Sascha Fahl, Christian Rossow, Giancarlo Pellegrino, Sven Bugiel, and Michael Backes. 2018. The rise of the citizen developer: Assessing the security impact of online app generators. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 634–647.

[75] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, Rui Shao, and Yan Chen. 2018. Flowcog: context-aware semantics extraction and analysis of information flow leaks in android apps. In *27th USENIX Security Symposium (USENIX Security 18)*. 1669–1685.

[76] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. 2013. {WHYPER}: Towards automating risk assessment of mobile applications. In *22nd USENIX Security Symposium (USENIX Security 13)*. 527–542.
[77] Leif E Peterson. 2009. K-nearest neighbor. *Scholarpedia* 4, 2 (2009), 1883.
[78] Aniket Pingley, Nan Zhang, Xinwen Fu, Hyeong-Ah Choi, Suresh Subramaniam, and Wei Zhao. 2011. Protection of query privacy for continuous location based services. In *2011 Proceedings IEEE INFOCOM*. IEEE, 1710–1718.
[79] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. 2011. IP Geolocation Databases: Unreliable? *SIGCOMM Comput. Commun. Rev.* 41, 2 (April 2011), 53–56. https://doi.org/10.1145/1971162.1971171
[80] Vincent Primault, Antoine Boutet, Sonia Ben Mokhtar, and Lionel Brunie. 2018. The long road to computational location privacy: A survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2018), 2772–2793.
[81] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. 2018. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. (2018).
[82] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium (USENIX Security 19)*. 603–620.
[83] Kimberly Redmond, Lannan Luo, and Qiang Zeng. 2018. A cross-architecture instruction embedding model for natural language processing-inspired binary code analysis. *arXiv preprint arXiv:1812.09652* (2018).
[84] Jingjing Ren, Martina Lindorfer, Daniel J Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. 2018. A longitudinal study of pii leaks across android app versions. In *Network and Distributed System Security Symposium (NDSS)*.
[85] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. 2016. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 361–374.
[86] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman. 2018. "Won't somebody think of the children?" examining COPPA compliance at scale. (2018).
[87] Jordan Samhi, Jun Gao, Nadia Daoudi, Pierre Graux, Henri Hoyez, Xiaoyu Sun, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2021. JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis. *arXiv preprint arXiv:2112.10469* (2021).
[88] Sam Schechner, Emily Glazer, and Patience Haggin. 2019. Political Campaigns Know Where You've Been. They're Tracking Your Phone. https://www.wsj.com/articles/political-campaigns-track-cellphones-to-identify-and-target-individual-voters-11570718889
[89] Yun Shen and Gianluca Stringhini. 2019. Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *28th USENIX Security Symposium (USENIX Security 19)*. 905–921.
[90] Wei Song, Heng Yin, Chang Liu, and Dawn Song. 2018. Deepmem: Learning graph neural network models for fast and robust memory forensic analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 606–618.
[91] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. 2012. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, Vol. 10. Citeseer.
[92] Anselm Strauss and Juliet Corbin. 1990. *Basics of qualitative research*. Sage publications.
[93] Jennifer Valentino-DeVries, Natasha Singer, Michael H Keller, and Aaron Krolik. 2018. Your apps know where you were last night, and they're not keeping it secret. *New York Times* 10 (2018).
[94] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, et al. 2021. Understanding malicious cross-library data harvesting on android. In *30th USENIX Security Symposium (USENIX Security 21)*. 4133–4150.
[95] Fengguo Wei, Sankardas Roy, and Xinming Ou. 2014. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 1329–1341.
[96] Haohuang Wen, Qingchuan Zhao, Zhiqiang Lin, Dong Xuan, and Ness Shroff. 2020. A Study of the Privacy of COVID-19 Contact Tracing Apps. In *International Conference on Security and Privacy in Communication Networks*.
[97] Daoyuan Wu, Debin Gao, Robert H Deng, and Chang Rocky KC. 2021. When Program Analysis Meets Bytecode Search: Targeted and Efficient Inter-procedural Analysis of Modern Android Apps in BackDroid. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 543–554.
[98] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 363–376.
[99] Hui Zang and Jean Bolot. 2011. Anonymization of location data does not work: A large-scale measurement study. In *Proceedings of the 17th annual international conference on Mobile computing and networking*. 145–156.

[100] Qingchuan Zhao, Haohuang Wen, Zhiqiang Lin, Dong Xuan, and Ness Shroff. 2020. On the Accuracy of Measured Proximity of Bluetooth-based Contact Tracing Apps. In *International Conference on Security and Privacy in Communication Networks*.

[101] Qingchuan Zhao, Chaoshun Zuo, Giancarlo Pellegrino, and Zhiqiang Lin. 2019. Geo-locating Drivers: A Study of Sensitive Data Leakage in Ride-Hailing Services. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS'19)*. San Diego, CA.

[102] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R Reidenberg, N Cameron Russell, and Norman Sadeh. 2019. Maps: Scaling privacy compliance analysis to a million apps. *Proc. Priv. Enhancing Tech.* 2019 (2019), 66.

[103] Fei Zuo, Xiaopeng Li, Patrick Young, Lannan Luo, Qiang Zeng, and Zhexin Zhang. 2018. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv preprint arXiv:1808.04706* (2018).

APPENDIX

App Name	Provider: GPS	Provider: Network	Provider: Passive	Freq.: Time Inter	Freq.: Dist. Inter	Manual Inspection: Has LBS	Manual Inspection: LBS Description	Privacy P.: Need Loc.	Privacy P.: Need Pvc	Privacy P.: Has Policy	Privacy P.: Loc. Usage	Privacy P.: Clarify	Privacy P.: Correct
Accelerate 1.0	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
AEAF 2018	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
AMANTES DEL TERROR(contiene chat)	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Arab Man Suit Photo Maker	✔	✔	✔	0	0	✗	-	✗	-	✔	✔	✗	✔
Assam Engineering College	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Azalea - Compras Online	✔	✗	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Azan and prayer times	✔	✗	✗	0	0	✔	Show local time	✔	✗	✗	-	-	✔
Babysitters America Training	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
Bajar de peso	✔	✔	✔	0	0	✗	-	✗	-	-	-	-	✔
BeeControls Automation	✔	✗	✗	1	0	✔	Weather	✔	✗	✗	-	-	✔
Best Hairstyle Step by Step	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
BHB School 2	✔	✔	✗	0	0	✔	Show location of the school	✗	✗	-	✔	-	✔
BourseTrading	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
CAMT/ ACM	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
CFP	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Chat with Juan Luis Guerra	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Check phone number location	✔	✔	✔	1	0	✗	-	✗	-	✔	✔	✗	✔
Competitors View	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Conciertos USA	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✔	✔	✗	✔
Consolidated411.com Directory	✔	✔	✗	1	0	✔	Weather and nearby places	✔	✗	✗	-	-	✔
Cool Pixel Launcher - pixel style launcher	✔	✗	✗	1	0	✔	Analytics and weather	✔	✗	✔	✔	✗	✔
Creative Activity Team	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	-	-	-	✔
Curso Palestra Gratuita	✔	✔	✗	0	0	✔	Location based push notification	✔	✗	-	-	-	✔
DancingNext	✔	✔	✗	3	2	✔	Search Dance Competition	✔	✗	-	-	-	✔
DSLR Professional Camera Effect 2019	✔	✔	✗	0	0	✔	Geo-tagging photos	✔	✗	✔	✗	-	✔
EL SHADDAI WORDS FM	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	-	-	-	✔
Emita lamasbonita	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Feira do Empreendedor Alagoas	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Fires Girona 2018	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
FrontLine Playbook	✗	✗	✔	0	0	✗	-	✗	-	✔	✔	✗	✔
Führerschein Persisch	✔	✔	✔	0	0	✔	Location based push notification	✔	✗	-	-	-	✔
Funciones y Responsabilidades de Recursos Humanos	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
Funciones y Responsabilidades del Contador	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
Geo-photo editor	✔	✗	✗	0	0	✔	Geo-tagging photos	✔	✗	✔	✔	✗	✔
GHANIAN SINGLES CHAT & DATE	✔	✔	✗	15	3	✔	Find nearby user to chat	✔	✗	-	-	-	✔
GLAM	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Gomez 360	✔	✔	✗	1	1	✗	-	✗	-	-	-	-	✔
Hair Claudio	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✔	✔	✗	✔
House of Chic	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✗	✔	-	✔
IHerb Arabic guide	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
IKiS	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✔	✔	✗	✔
Inkollo	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✔	✔	✗	✔
INSIDE-Photo Editing	✔	✔	✗	0	0	✔	Geo-tagging photos	✔	✗	✔	✔	✗	✔
Italian Women Dating & Hookups	✔	✔	✗	15	3	✔	Find nearby user to chat	✔	✗	✔	✔	✗	✔
K7	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Key Account Management	✗	✗	✔	0	0	✔	Low accuracy requirement	✔	✗	✔	✔	✗	✔
Kobufest	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Le Trou	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
"MakeUp Tutorial, Eyes, Lips, Eyeliner, Tips, 2019!"	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
MakeUrSport	✔	✗	✗	1	0	✔	Show current speed and ave. speed	✔	✔	✔	✗	-	✗
Meet Big Beautiful Women Near You	✔	✔	✗	15	3	✔	Find nearby user to chat	✔	✗	✔	✔	✗	✔
Mobile Number Location : Live Phone Locator	✔	✗	✗	1	0	✔	Nearby places	✔	✗	✔	✗	-	✔
Momming University	✔	✔	✗	0	0	✔	Location based push notification	✔	✗	-	-	-	✔
Monumental Sorteo HECA	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Moto Sport Race Championship	✗	✗	✔	0	0	✗	-	✗	-	-	-	-	✔
My Insta photo editor:Online photo editor app	✔	✗	✗	2	0	✔	Geo-tagging photos	✔	✗	✔	✔	✗	✔
Nails decoration	✔	✔	✗	2	1	✗	-	✗	-	✔	✔	✗	✔
Nashwa Beauty	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
New York Travel Guide -Tourias	✔	✗	✗	0	0	✔	Distance to attaractions	✔	✗	✔	✗	-	✔
NST Blue Event	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	✗	✔
parquenaturalxview	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Peinados Fáciles Y Bonitos Paso A Paso	✔	✔	✔	0	0	✗	-	✗	-	-	-	-	✔
Perfect Weather	✔	✗	✗	0	0	✔	Get curret location	✔	✗	✗	-	-	✔
Photo Editor 2019	✔	✗	✗	1	0	✔	Geo-tagging photos	✔	✗	✔	✔	✗	✔
Physics 1	✔	✔	✗	0	0	✔	Location based push notification	✔	✗	-	-	-	✔
Piano Video Lessons	✔	✔	✗	0	0	✔	Location based push notification	✔	✗	-	-	-	✔
Portage church of Christ	✔	✔	✗	0	0	✔	Location based push notification	✔	✗	✔	✗	-	✔
Prestige B	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Quero Festas	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✔	✔	✗	✔
quick edit	✔	✗	✗	1	0	✔	Geo-tagging photos	✔	✗	✔	✔	✗	✔
RADIO ASTROS - Serrano	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
RADIO BOCA DE CORDOBA	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Radio de Jesucristo Proyeccion y Poder	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Radio Locutor Free	✔	✔	✗	0	0	✔	Find Radio station	✔	✗	-	-	-	✔
Radio Salvación Estéreo	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Romantic Love Stories	✔	✔	✗	15	3	✔	Find nearby user to chat	✔	✗	-	-	-	✔
Room Temperature Info	✔	✗	✗	0	0	✔	Weather but fake data	✔	✗	✔	✔	✗	✔
RRB Group D/RPF/RPSF/SSC Model Paper Free	✔	✔	✗	0	0	✔	Location based push notification	✔	✗	-	-	-	✔
SingleSpot	✗	✗	✔	0	0	✔	Filter user with distance	✔	✗	✗	-	-	✔
Smart Kit 360	✔	✔	✗	0	0	✔	Speedometer; altimeter	✔	✗	✗	✗	-	✔
Smart Society Pioneer Park	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Southmead Primary School	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	✗	-	-	✔
Springfield Academy	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	✗	-	-	✔
STC's (Siam Toastmasters Club)	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	-	-	-	✔
SWAGTRON Snowtide	✔	✔	✗	0	0	✔	Weather and speedometer	✔	✔	✗	-	-	✔
Teeth Whitening Naturally	✔	✔	✔	0	0	✗	-	✗	-	✔	✗	-	✔
Televisión Bolivia (Tv Bolivia)	✔	✔	✗	0	0	✗	-	✗	-	✗	-	-	✔
TopKanale - Shiko Tv Shqip	✔	✔	✗	0	0	✗	-	✗	-	✗	-	-	✔
Total Sports League	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Tourist guide of Jiaojiaodayuan	✔	✗	✗	5	0	✔	Current location	✔	✔	✔	✗	-	✗
Tourist guide of Xitangguzhen	✔	✗	✗	1	0	✔	Current location	✔	✔	✔	✗	-	✗
Tribal / EPA Conference	✔	✔	✗	0	0	✔	Search Dance Competition	✔	✗	✔	✔	✗	✔
TRIHABBO	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
TV Indonesia - Live Streaming	✔	✔	✗	1	0	✗	-	✗	-	-	-	-	✔
United Body of CHRIST	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	✔	✗	-	✔
V Tech Club	✔	✔	✗	0	0	✔	Filter user with distance	✔	✗	-	-	-	✔
Video Llamadas con Chicas en El Celular Guide	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	✗	✔
Wellnajare	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
WMBB Weather - MyPanhandle.com	✔	✗	✗	0.01	0	✔	Weather; geo-tagging photo	✔	✗	✔	✔	✗	✔
Zaglu Event Finder	✗	✗	✔	5	0	✔	Display near by POI on map	✔	✗	-	-	-	✔

App Name	Provider: GPS	Provider: Network	Provider: Passive	Freq.: Time Inter	Freq.: Dist. Inter	Manual Inspection: Has LBS	Manual Inspection: LBS Description	Privacy P.: Need Loc.	Privacy P.: Need Pvc	Privacy P.: Has Policy	Privacy P.: Loc. Usage	Privacy P.: Clarify	Privacy P.: Correct
AL KAHRABA	✔	✔	✗	0	0	✔	Show PoI on the map	✔	✗	✔	✗	-	✔
AM FM Radio Tuner Online	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Apyar Myanmar	✔	✔	✗	0	0	✔	Show user position to others	✔	✗	✔	✗	-	✔
Athan Times	✔	✗	✗	0	0	✔	Prayer times calculation	✔	✗	✔	✔	✗	✔
Bajar Música - Descargar Videos	✔	✔	✗	0	0	✔	Show user position on map	✔	✔	✔	✔	-	✔
Bajar Musica Facil y Rapido MP3	✔	✔	✗	0	0	✗	-	✗	-	✔	✔	-	✔
Bajar musica gratis mp3	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Bajar Musica TUTORIAL Fast	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Bajar Música Videos Gratis	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Bajar Música Y Videos	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Bajar Música Y Videos	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
BGS - Brasil Game Show	✔	✔	✗	0	0	✔	Show route to PoI without navigation	✔	✗	-	-	-	✔
Browser for Android	✔	✔	✗	1	0	✔	Local weather information	✔	✗	-	-	-	✔
Cara Membaca Pikiran Orang	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Change beep	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Christian Worship Songs Mp3	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Colmédica	✔	✔	✗	0	0	✔	Show the nearby hospitals	✔	✗	✔	✗	-	✔
Como Hacer Crecer Las Uñas	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Como Saber Con Quien Habla	✔	✔	✗	0	0	✔	Show user position on map	✔	✗	-	-	-	✔
Como Saber Quien Te Llama	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Con Quien Chatea Tu Pareja	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Con Quien Esta En Linea	✔	✗	✔	0	0	✗	-	✗	-	-	-	-	✔
Como ver la	✔	✗	✔	0	0	✗	-	✗	-	✗	-	-	✔
Cycling - Bike Tracker	✔	✗	✗	0	0	✔	Record the trail of your riding	✔	✔	-	-	-	✔
Daff Moon Phase	✔	✔	✗	0	0	✔	Show local sunrise and sunset times	✔	✗	✔	✔	✗	✔
Dangdut KOPLO	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
E-GetS	✔	✔	✗	1	0	✔	Obtain takeaway delivery address	✔	✗	✔	✔	✗	✔
Eatthis	✔	✔	✗	0	0	✔	Show the nearby events	✔	✗	✔	✔	✗	✔
EMI Calculator	✔	✔	✗	0.03	0	✔	Show the nearby banks and ATMs	✔	✗	✔	✔	-	✔
Emisoras Colombianas en Vivo	✔	✗	✗	0	0	✗	-	✗	-	✔	✔	-	✔
Esan Vakti Pro	✔	✔	✗	0	0	✔	Prayer times calculation	✔	✗	✔	✔	✗	✔
Fly Delta	✔	✔	✗	0	0	✔	Show the nearby airports	✔	✗	✔	✔	✗	✔
Free Golf GPS APP - FreeCaddie	✔	✔	✗	0	0	✔	Show the nearby golf courses	✔	✗	✔	✔	✗	✔
Free Romantic Music Ballads of Love Online	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
freshwater fishing in Korea	✔	✗	✗	0	0	✔	Nearby fishing boats and grounds	✔	✗	✔	✔	-	✔
G-r-a-t-i-s Lagu Lama	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
Garmin Smartphone Link	✔	✔	✗	0	0	✔	Show the nearby attractions and traffic	✔	✔	✔	✔	✗	✔
Get Audio Tour Guide and Travel Guide	✔	✔	✗	0	0	✔	Show the nearby attractions or stories	✔	✗	✔	✔	-	✔
GlobaLive - video chat	✔	✗	✗	1	0	✗	-	✗	-	-	-	-	✔
GOLF NETWORK PLUS	✔	✔	✗	0	0	✔	Show the nearby golf courses	✔	✗	✗	-	-	✔
GPS Live Navigation	✔	✔	✗	1	1	✔	Use location for navigation	✔	✔	✔	✔	✗	✔
GPS Maps Navigation	✔	✔	✗	1	0	✔	Show the nearby hotels	✔	✗	✔	✔	✗	✔
HungryPanda	✔	✔	✗	0	0	✔	Obtain the takeaway delivery address	✔	✗	✔	✔	-	✔
iQiyi	✔	✔	✗	1	0	✗	-	✗	-	-	-	-	✔
iSearching	✔	✔	✗	1	0	✔	Record the loc. of last connection	✔	✗	✔	✔	-	✔
Islam.ms Prayer Times	✔	✗	✗	1	0	✔	Prayer times and nearby mosques	✔	✗	✔	✗	-	✔
JUKUSUI	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
KazaanBfio	✔	✔	✗	0	0	✔	Show the nearby gifts	✔	✗	✔	✗	✗	✔
KIKUU: Online Shopping Mall	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Live Football Tv Stream HD	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Lost Android	✔	✔	✗	1	0	✔	Show position of lost device	✔	✗	✔	✔	✗	✔
markt.de Kleinanzeigen	✔	✔	✗	0	0	✔	Show the nearby goods	✔	✗	✔	✔	✗	✔
MARTI Scooter	✔	✔	✗	0	0	✔	Show the nearby shared balance car	✔	✔	✔	✗	-	✔
Mesaenger.Go for Social Media	✗	✗	✗	0	0	✔	Browser; location based requirements	✔	✗	✔	✔	-	✔
Metro and Bus Tickets	✔	✗	✗	1	0	✔	Show the nearby stops and traffics	✔	✔	✔	✔	✗	✔
Microsoft Events	✔	✔	✗	0	0	✔	Show the nearby events	✔	✗	✔	✔	✗	✔
Mobile agent	✔	✔	✗	0	0	✔	Show PoI on the map	✔	✗	✔	✔	✗	✔
Mobile Observatory Free - Astronomy	✔	✗	✗	0	0	✔	Show the local constellation orientation	✔	✗	✔	✔	✗	✔
MotorData OBD Car Diagnostics	✔	✗	✔	0	0	✗	-	✗	-	✔	✔	-	✔
MTR Mobile	✔	✔	✗	0	0	✔	Provide drop-off tips	✔	✗	✔	✔	-	✔
Música Cumbia Radios	✔	✗	✗	0	0	✗	-	✗	-	✗	-	-	✔
Musik Dj Terbaru	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Noor : Quran and Hadith	✔	✔	✗	1	0	✔	Prayer times and nearby mosques	✔	✗	-	-	-	✔
Oldies Radio Station For Free	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
One Security	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
part-time job search	✔	✔	✗	1	1	✔	Show the nearby jobs	✔	✗	✔	✗	✗	✔
Pedometer - Step Counter and walking tracker	✔	✗	✗	0	0	✔	Record the trail of your running	✔	✔	✗	✗	✗	✔
Philips Kitchen	✔	✔	✗	2	0	✗	-	✗	-	-	-	-	✔
Prayer times and Azan without Net	✔	✔	✗	0	0	✔	Prayer times and nearby mosques	✔	✗	✔	✔	✗	✔
Puhu – eBooks and Videos Anytime	✔	✔	✗	0	0	✗	-	✗	-	✔	✗	-	✔
QR Droid	✔	✔	✗	0	0	✔	Produce QR codes	✔	✗	✔	✔	✗	✔
Questionnaire points-Powl	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Quran Majeed: Prayer Times and Athan	✔	✔	✗	0	0	✔	Prayer times calculation	✔	✗	✔	✔	✗	✔
RadioApp – FM, AM, DAB+	✔	✔	✗	10	0	✔	Show the local radio stations first	✔	✗	✔	✗	✗	✔
rap phrases	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
RTI Mobile	✔	✗	✗	0	0	✗	-	✗	-	✔	✗	-	✔
Salaat First (Prayer Times)	✔	✔	✗	1.5	0	✔	Prayer times and nearby mosques	✔	✗	✔	✔	✗	✔
Sarkari Naukri - Govt Job alerts	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
Seminit Resepi	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
Shell	✔	✗	✔	0	0	✔	Show the nearby gas stations	✔	✗	✔	✔	✗	✔
Smart Bijlee	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
Smart Wristband	✔	✔	✗	0	0	✔	Record the trail of your exercises	✔	✔	✗	✔	-	✔
Snap Camera HDR - Trial	✔	✗	✗	1	0	✗	-	✗	-	✔	✔	✗	✔
Swahili Worship and Gospel Songs	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔
T-Mobile FamilyWhere	✔	✔	✗	0	0	✔	Family members locations sharing	✔	✔	✗	✔	✗	✔
Tener Internet Gratis	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
TV México Señal Abierta	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
TweetCaster for Twitter	✔	✔	✗	0	0	✔	Show the nearby tweets	✔	✗	✔	✗	✗	✔
Uplifting Psalms Daily	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
Vivino: Buy the Right Wine	✔	✗	✗	1	0	✗	-	✗	-	-	-	-	✔
Voice Answer Free	✔	✗	✗	5	0	✗	-	✗	-	-	-	-	✔
Voxer Walkie Talkie Messenger	✔	✔	✗	0	0	✔	Geo-tagging	✔	✗	✔	✔	✗	✔
Wheres My Droid	✔	✔	✗	0	0	✔	Show the position of lost device	✔	✗	✔	✔	✗	✔
Wise Proverbs Daily	✔	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
Words of Jesus Each Day	✗	✗	✗	0	0	✗	-	✗	-	-	-	-	✔
WorldCard Mobile Lite	✔	✔	✔	0	0	✗	-	✗	-	-	-	-	✔
Yahoo! MAP	✔	✔	✗	5	0	✔	Use location for navigation	✔	✔	✔	✔	✗	✔
YY Circle	✔	✔	✗	1	0	✔	Show the nearby people and jobs	✔	✗	✔	✔	✗	✔
ZenUI Dialer and Contacts	✔	✔	✗	0.5	0	✗	-	✗	-	-	-	-	✔
Zombie Coloring	✔	✔	✗	0	0	✗	-	✗	-	-	-	-	✔

Table 3. Details of Manual Verification of Aggressive Apps (left: 2019, right: 2021): Privacy P: Privacy Policy; Freq: Frequency; Loc: Location; Time Inter: Time Interval; Dist Inter: Distance Interval; Per: Periodical Location Acquisition; ✔: presence; ✗: absence; -: N/A.

Lib Identifier	Provider					
	GPS		Network		Passive	
	Time	Dist.	Time	Dist.	Time	Dist.
admob	-	-	-	-	-	-
mobfox	0.05	10	0.05	10	60	100
tapjoy	0	0	120	200	-	-
smaato.soma	-	-	-	-	-	-
swelen.ads	0	0	0	0	-	-
adtech	0	0	-	-	-	-
nielsen	0	0	0	0	-	-
vervewireless.advert	0	0	0	0	0	0
admarvel	0	0	-	-	-	-
anywheresoftware.b4a	0	0	60	10	-	-

(a) Advertisement libraries

Lib Identifier	Provider					
	GPS		Network		Passive	
	Time	Dist.	Time	Dist.	Time	Dist.
flurry	-	-	10	0	10	0
moat.analytics	0	0	0	0	-	-
newrelic.agent	-	-	-	-	1	0
quantcast	300	8000	300	8000	-	-
umeng	1	0	0	0	-	-
kochava.android	0	0	0	0	-	-
applause.android	300	0	300	0	-	-
yandex	1	0	0	0	-	-
clevertap	60	10	60	10	-	-
talkingdata	0	1	0	1	-	-

(b) Analytics libraries

Table 4. Location Collection in Top 10 A&A Libraries: Time: Time Interval; Dist: Distance Interval; -: N/A.