"Why Are They Collecting My Data?": Inferring the Purposes of Network Traffic in Mobile Apps

HAOJIAN JIN, Carnegie Mellon University, USA
MINYI LIU, Tsinghua University, China
KEVAN DODHIA, Carnegie Mellon University, USA
YUANCHUN LI, Peking University, China
GAURAV SRIVASTAVA, Carnegie Mellon University, USA
MATTHEW FREDRIKSON, Carnegie Mellon University, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA
JASON I. HONG, Carnegie Mellon University, USA

Many smartphone apps collect potentially sensitive personal data and send it to cloud servers. However, most mobile users have a poor understanding of why their data is being collected. We present MobiPurpose, a novel technique that can take a network request made by an Android app and then classify the data collection purposes, as one step towards making it possible to explain to non-experts the data disclosure contexts. Our purpose inference works by leveraging two observations: 1) developer naming conventions (e.g., URL paths) often offer hints as to data collection purposes, and 2) external knowledge, such as app metadata and information about the domain name, are meaningful cues that can be used to infer the behavior of different traffic requests. MobiPurpose parses each traffic request body into key-value pairs, and infers the data type and data collection purpose of each key-value pair using a combination of supervised learning and text pattern bootstrapping. We evaluated MobiPurpose's effectiveness using a dataset cross-labeled by ten human experts. Our results show that MobiPurpose can predict the data collection purpose with an average precision of 84% (among 19 unique categories).

CCS Concepts: • **Security and privacy → Privacy protections**;

Additional Key Words and Phrases: Mobile Privacy, Purposes of Data Collection, Contextual Integrity, Privacy in Context

ACM Reference Format:
Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. 2018. "Why Are They Collecting My Data?": Inferring the Purposes of Network Traffic in Mobile Apps . *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4, Article 173 (December 2018), 27 pages. https://doi.org/10.1145/3287051

Authors' addresses: Haojian Jin, haojian@cs.cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA; Minyi Liu, Tsinghua University, 30 Shuangqing Rd, Haidian Qu, Beijing, 100084, China; Kevan Dodhia, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA; Yuanchun Li, Peking University, 5 Yiheyuan Rd, Haidian Qu, Beijing, 100871, China; Gaurav Srivastava, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA; Matthew Fredrikson, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA; Yuvraj Agarwal, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA; Jason I. Hong, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA.

2474-9567/2018/12-ART173 $15.00
https://doi.org/10.1145/3287051

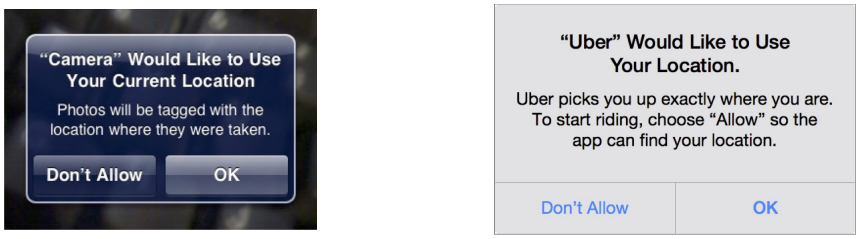

Fig. 1. Android and iOS ask developers to offer explanations about why an app is accessing sensitive user data by way of a purpose string or a "usage description". However, these purpose strings are only shown at the user interface layer, and can be arbitrary text. There is no easy way to verify if the purpose strings are accurate. Furthermore, these purpose strings are a relatively recent addition to Android and iOS, and are not yet widely adopted.

1 INTRODUCTION

A major privacy concern of mobile apps is that they can potentially access a great deal of sensitive personal information [2, 30]. Here, we focus on one dimension of mobile privacy, namely the point when sensitive data leaves the device and is sent to remote servers over the network. Many privacy researchers have adopted this network perspective, studying the raw attributes of privacy-sensitive mobile data sharing [34, 61, 65, 66, 77, 78], namely which app is sharing data, what data is being shared, and where that data is going.

Currently, though, there is little research looking into *why* an app is requesting access to sensitive data (we also refer to this as the *purpose* of data collection). "Why" is a fundamental component of contextual definitions of privacy [48, 54]. For example, users might be more willing to provide their locations to make search results more relevant, but less for targeted advertising. Past work has also found that surfacing what data an app is using without explaining why can raise privacy concerns [43, 48, 70]. For example, Lin et al. [43] found that when end-users were told that the Dictionary app accessed their location, they were very concerned about privacy. However, when told that location data was only used to search for trending words that people nearby are looking up, they felt much less concerned. As another example, Brush et al. [7] found that people were very willing to share their location data to help cities plan bus routes or to get traffic information, but reluctant to share the same data for ads or for maps showing one's travel patterns.

A major challenge here is that end-users currently have little support for understanding the purpose of data use in smartphone apps [26, 43]. Android and iOS now offer the capability to explain why an app is accessing sensitive user data by way of a purpose string or a "usage description" [18]. Figure 1 illustrates two example app permission modal dialog boxes, which depicts that Camera/Uber (who) is requesting location (what) to tag photos or locate passengers (why). However, these purpose strings are only shown at the user interface layer, and can be arbitrary text. There is no easy way to verify if the purpose strings are accurate. Furthermore, these purpose strings are a relatively recent addition to Android and iOS, and are not yet widely adopted [68].

There has been some past work looking at how to infer purposes, for example based on static text analysis of executables [71] or by considering the libraries used by the app [11, 36]. Instead, we look at a new approach for inferring purpose, namely based on network traffic, since looking at data egress provides a better vantage point in understanding the specific data that is flowing out of the phone.

In this paper, we present *MobiPurpose*, a novel technique to automate the inference of mobile traffic purposes. MobiPurpose takes a network request made by a smartphone app and classifies the purposes of each of the key-value pairs, to help explain to non-experts why their data is being collected.

MobiPurpose is designed to run on in-lab devices instead of end-users' smartphones. We built tools to automate downloading and installing apps onto a smartphone, interacting with those apps using an Android Monkey

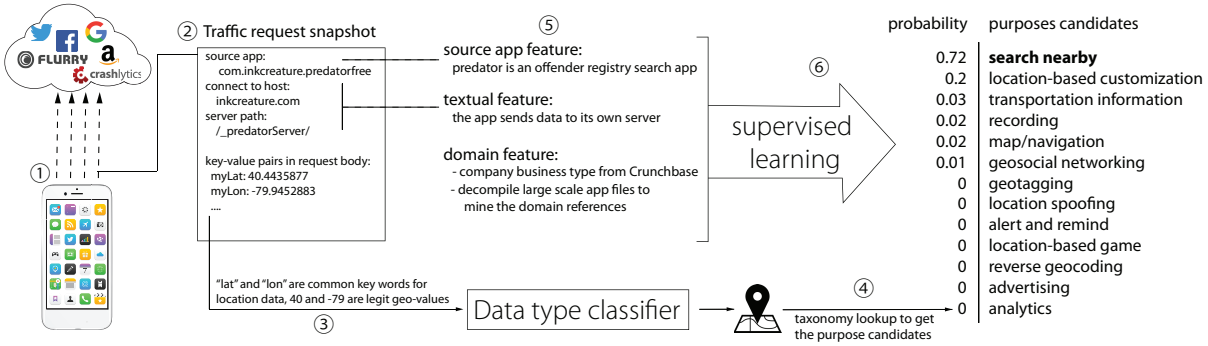

Fig. 2. Overview of MobiPurpose's workflow (running on in-lab devices). Numbers indicate the order of the methodological steps. (1) MobiPurpose uses a UI automation tool to simulate user interaction with an app, allowing us to navigate an app and generate network traffic. (2) MobiPurpose uses a VPN that acts as a Man-in-the-middle (MITM), intercepting all outgoing traffic requests from a smartphone and parsing the body of requests into key-value pairs. (3) A bootstrapping algorithm automatically classifies each key-value pair data type based on the textual patterns. (4) Based on the identified data type, we find associated purpose candidates through a taxonomy lookup. (5) Based on internal and external features extracted from the traffic snapshot, (6) we use a supervised machine learning model to predict the data collection purpose.

Script [67], and capturing synthetic network data from in-lab devices. We assume that the network service API collects nearly the same types of data from different devices. While we are intercepting data using in-lab devices, the learned knowledge can be transferred to real users as well. MobiPurpose is a key step towards our broader goal of offering a public, large-scale database (similar to PrivacyGrade [36]) that can show not only what data is being collected by different services around the world via smartphone apps, but also help explain why. This will allow us to analyze the privacy practice in mobile apps as well as API service providers in an unprecedented way.

1.1 MobiPurpose: Inferring the Purposes of Network Traffic in Mobile Apps

MobiPurpose uses a VPN approach to intercept all outgoing traffic requests and parses the HTTP(S) request body into KV pairs (see Fig. 2 ②). HTTP(S) is considered the most common protocol in smartphone-server communication [25] and recent studies show that more than 80% of apps have structured responses [66]. Using a combination of supervised machine learning and natural language processing (NLP) techniques, *MobiPurpose* inspects the data disclosure context of each KV pair. Figure 2 illustrates an example of MobiPurpose inferring the data type of *"myLat:40.44; myLon:-79.94"* as *LOCATION* and the corresponding purpose as "search nearby".

At the core of *MobiPurpose* is our purpose taxonomy (§2), in which we enumerate the potential data collection purposes for each data type explicitly (See the complete taxonomy in Appendix §A). That is, rather than having to generate text describing the purpose, our goal is to instead select the appropriate purpose from this taxonomy. Given any traffic request, we first infer what data types are involved using a bootstrapping NLP approach (§3), and then find the associated purpose candidates through a taxonomy lookup. Then, we use a supervised machine learning approach to predict the data collection purposes (§4).

Our machine learning approach emulates how reverse engineering experts [16, 56] use various cues in network traffic (e.g., variable names, the URL path, etc.) to infer API details (e.g., data types, data collection purposes). We first observed a group of experts examine a large set of network traffic requests and label data types and purposes (§6), and then asked the participants to describe their reasoning processes (§4). Based on these observations, we propose a set of computational features (§4) to model these data patterns by leveraging two intuitive facts:

- Developer naming conventions often make identifiers self-explainable, in both KV pairs and URLs. For example, a KV pair *"userAdvertisingId : 901e3310-3a26-487e-83c7-2fa26ac2786c"* is likely a unique ID, because the keyname contains keywords such as "advertising" and "id", and the value is a machine generated UUID. As another example, a network request connecting to reports.crashlytics.com is likely for crash reports generation, since the domain is a neologism of "crash" and "analytics", and the subdomain is "reports". We also discuss potential obfuscations in §9.3.
- External knowledge, such as the app description and the domain owner, can be meaningful cues to infer the data collection purpose [43]. For example, suppose we have a game app that sends location data to admob.com. Since AdMob is a mobile advertising company, we can infer that the location data is probably used to tailor what advertisements to show based on the user's location.

We used automated tests to scale the traffic data collection. We developed a test harness[1] that can install apps on an Android device and then explore those apps using an interface automation monkey [67] that randomly clicks on UI elements. Over 50 days, we used 8 Android devices to collect data on 15k apps, each running the monkey for 3 minutes, and intercepted 2 million unique traffic requests (§5). We then sampled 1059 traffic requests and had 10 human experts manually label the behavior of each traffic request using our What&Why Taxonomy. Since data collection purposes can be subjective and ambiguous, we collected 3 independent labels for each traffic request.

We used the labeled data to evaluate our data type inference algorithm and train a purpose classifier. In our experiments, we found that MobiPurpose achieved an average precision of 95% in predicting "what" (among 8 unique data type categories) and 84% in predicting "why" (among 19 unique purpose categories).

As noted earlier, our approach is intended to replicate what developers already do when attempting to reverse engineer APIs, so our method does not account for things like deliberate obfuscation of variable names or server names (See §9.3), or account for additional hidden purposes of how the data is used once it is on a remote server. However, we believe that MobiPurpose can be an effective starting point in improving the transparency of the smartphone app ecosystem, by mapping out what data is being shared with why it is being collected.

1.2 Contributions

Our contributions lie in the technical implementation of purpose inference. MobiPurpose is the first work to address automatic mobile network purpose inference. To achieve that, we first make the machine inference feasible by turning the purpose description generation task into a classification task. We then propose practical computational methods to emulate how reverse engineering experts use various cues in network traffic (e.g., variable names, the URL path, etc.) to infer data collection purposes. Our specific contributions are as follows:

- We present the design and implementation of the first system that can automatically categorize data collection purposes of sensitive data in mobile network traffic. To achieve this, we developed an extensible taxonomy of purposes, and turned the purpose description generation task into a classification task.
- We propose a set of practical computational features and patterns for inferring purposes. We investigate the effectiveness of different kinds of features, showing that text-based features and domain features offer high gains, while source app features offer marginal improvements.
- We evaluate our framework using a human-labeled data set of 1059 labeled instances of network data from 815 different apps, which contact 636 distinct domains. We found that MobiPurpose can achieve an average accuracy of 95% for data type inference and 84% for data purpose inference.

The remainder of this paper is organized as follows. The system design of MobiPurpose is discussed in §2-4. We present our experiment and evaluation in §5-7, followed by a discussion of the related work in §8. We conclude with the limitations and design decisions of our system in §9-11.

[1] Open sourced at https://github.com/CMUChimpsLab/MobiPurpose

2 BUILDING A PURPOSE TAXONOMY

This section describes the design and development of our purpose taxonomy, which turns the purpose description generation task into a classification task. We present the full taxonomy in the Appendix (§A). The goal of this taxonomy is to offer a collection of purposes that is comprehensive (covers the vast majority of use cases of sensitive data), have a meaningful granularity (a given purpose should not be too narrow in only characterizing very few apps, nor too broad), and understandable (developers and end-users can understand each purpose with minimal explanation). Below, we describe our process for developing this taxonomy of purposes.

2.1 Existing Purpose Taxonomies

We first conducted a broad survey of existing purpose taxonomies (Table 1 enumerates a partial list). We found that two distinct types of purposes were used mixedly in the past work. Some of the purposes are proposed to describe the specific context explicitly (e.g., nearby search, map navigation), while the rest are more general (e.g., legitimate, primary).

Our goal is not to argue if the privacy-sensitive data disclosure is legitimate. Instead, we want to communicate to non-experts the data collection contexts and help them understand what they should expect. So we opted to focus on purposes that better characterize the functionality, e.g. advertising, SNS, etc, but exclude purposes such as "primary" [29], "internal use" [44], "legitimate" [24], or "core functionality" [70]. We discuss the detailed rationale behind that choice in §10.2.

Table 1. A partial list of employed purposes in past work and MobiPurpose. MobiPurpose taxonomy does not capture if a data disclosure is legitimate or malicious. Rather, the goal is to characterize the traffic requests and explain why and how they can cause trouble potentially.

	Data collection purposes (Why)
Lin et al. [44]	utility, advertising, UI customization, content host, game, SNS, analytics, payment, internal use
Wang et al. [71]	10 fine-grained purposes (e.g. search nearby places, map/navigation) for two permissions (location, contact list)
Han et al. [29]	primary, advertising, content server, analytics, API, don't know
TaintDroid [24]	legitimate or non-legitimate
Kleek et al. [70]	core/non-core functionality, marketing
Martin et al. [48]	game, weather, social networking, navigation, music, finance, shopping, productivity
MobiPurpose	76 fine-grained purposes (e.g., ad, analytic) for 16 data types (e.g., device info, tracking ID)

2.2 Methodology

We iteratively developed our taxonomy by examining a large number of smartphone apps, in terms of the sensitive data they used (Android permissions) and network traffic behaviors.

Permission access. Android classifies all permissions into three protection levels: normal, signature, dangerous. Requesting the 9 "dangerous" permissions require users' explicit consent [20]. We indexed Android apps (see dataset description in §5) based on their required permissions, and sampled 100 apps for each "dangerous" permission, resulting in a total of 900 apps. We then printed the app name, app description, and permission on 900 index cards, which would be used for the later card sorting session.

Network traffic. We installed 20 most popular free apps across all categories on two smartphones and then actively used each app for 3 minutes, with the goal of invoking major pieces of functionality in these apps. We used a MITM (man-in-the-middle) VPN app [8] to intercept 5504 unique HTTP(S) traffic requests connecting to 321 unique domains. We then sampled 400 of these traffic requests, each contacting a unique URL (domain + path). We printed the destination host name, app name, app description, and data body on 400 index cards.

Fig. 3. Taxonomy card sorting sessions. Left: participants first sort the raw cards (white cards) and proposed their categorizations (post stickers in assorted colors) independently, in which each cluster (single color) is the individual result. Right: after the individual sorting session, we have group discussions to reach an agreement and use an affinity diagram technique to form a hierarchical taxonomy.

We conducted four card sorting [75] sessions to create and refine the taxonomy. In each iteration, participants first independently created their own taxonomy (Fig. 3 left) and then discussed with other participants to reach an agreement (Fig. 3 right). The converged taxonomy would then be the starting point of the next iteration. Overall, we had 12 participants across these four sessions, involving a mix of mobile developers and UX designers.

In our earlier iterations of the taxonomy, one main question that arose was how to organize the purposes into meaningful categories.

- We started with ***independent purpose categories*** similar to [29, 44], such as Social/Communication, Advertising, Game, Multimedia consumption, Content, Analytic, Personalization, and Map/Navigation. This structure is intuitive for both developers and users since apps are organized through these categories in the App Stores. However, this purpose granularity mainly stays at the app-category level, only providing little privacy insights.
- We then switched to a ***hierarchical structure*** similar to [48, 70] where the end leafs were purposes, and the parent nodes were app categories. However, participants in later iterations found this hard to use, in terms of being able to look up purposes quickly. Apps in any category can make traffic requests for any purpose. Besides, this structure can help users understand how privacy friendly the app is, but not the disclosure context of the specific network request.
- Our final taxonomy is inspired by the app permission modal dialog boxes (Fig. 1). We still use a hierarchy in our final taxonomy (see Appendix Fig. 9, Table 10,11,12,13), but the parent nodes are now data types, with the end leafs being purposes (same as before). The purpose candidates listed in Figure 2 are the leaf nodes of "*LOCATION*" node. We also organize the data types into four groups (see Appendix Fig. 9): *PHONE ID*, *PHONE STATUS*, *PERSONAL DATA*, and *SENSOR*.

3 DATA TYPE (WHAT) INFERENCE

The hierarchical structure of our taxonomy allows us to narrow the purpose candidates down significantly by inferring the data type first. This section describes how MobiPurpose can predict the data type of *"myLat:40.44; myLon:-79.94"* as *LOCATION* (Figure 2③).

Data type inference in the network traffic has been studied extensively in the past. Past projects use either hard-coded regular expressions [37, 65] or supervised machine learning approaches [61] to classify data types.

However, the regular expressions cannot be generalized to unseen patterns, and labeling training data for our taxonomy would be a daunting task. Instead, we adapt a traditional generative NLP technique to avoid data labeling and scale the automation to fine-grained data types. More specifically, we first use a bootstrapping method to build a key-value text pattern corpus, and then derive a Bayesian probabilistic model from the corpus to classify the data type.

3.1 Using Pattern Bootstrapping to Build a Corpus

Our bootstrapping approach is inspired by prior work in information extraction tasks, which requires *minimal human participation* [3, 32, 33]. The key idea is leveraging the key-value text pattern redundancy. We noticed that many key-value pairs in our data set were combinations of constrained text patterns, owing to two reasons. First, developers share similar naming conventions, which limits the key name variations. For example, typical patterns for latitude key names are: "lat," "***_lat," "lati," "latitude," etc. Second, our devices also share similar configurations (e.g., physical location, device models), which limits the value variations as well. If a developer tries to collect some GPS data, the devices will send nearly the same values to the server.

More concretely, if we manually start with a handful of patterns for LOCATION key names, we can then find more key-value pairs containing unknown value patterns. These new value patterns can then be used to find more key-value pairs again, this time with new key name patterns. In each iteration, the algorithm will only keep the most reliable ones for the next iteration. This iterative process can collect a large text pattern corpus with minimal human efforts. We describe the approach in more detail below.

Table 2. Extracted text patterns for key-value examples. We extract two types of text patterns for any key/value string: bag-of-words & special string, and then use these text patterns for the later bootstrapping process.

Key: Value	bag-of-words	special string
devid: 99349319-a6c7-4657-a3bc-6929c52090e1	dev, id	UUID
ip_addr: 128.237.175.242	ip, addr	IP
device_model: Nexus_6P	device, model, nexus, 6p	
pickup_lat: 40.4431531	pickup, lat	LATITUDE_NUM

Extracting text patterns. A strawman solution is to use the exact string match as the text pattern. However, it doesn't fit well with the developer variable contexts. For example, common latitude key names include "dev_lat," "device_lat," "my_lat," "mylat," etc. Values of UUIDs are universally unique but share a common string pattern. We need a more generalizable representation to capture these repetitive text patterns. Through experimentation, we extract two types of text patterns for any key/value string:

- *Special-string check*: we first use regex expressions and bi-gram models to recognize common special strings, such as MAC address, IP, Email, URL address, UUID [21], Advertising-ID [19], MD5 Hash, package name, timestamp, isnumber, latitude/longitude, developer version number, randomly generated strings [51], etc.
- *Bag-of-words*: If the key/value string is not a special string, we parse the string into a bag-of-words representation using an open source English WordSegmentation model [55]. To better handle technical jargons (e.g., lte, gsm) and casual abbreviations (e.g., ad for advertise), we manually add these common terms into the open source model.

Bootstrapping the text patterns. Algorithm 1 shows pseudo-code for our bootstrapping process. The method is initialized with some manual seed rules to provide initial rules for learning. For example, if the key contains "lat" or "lon" and the value is a legitimate latitude/longitude string, the data type would be *LOCATION*.

We (1) use the initial rules to find an initial *LOCATION* key-value set (denoted as L-KV), (2) extract all the text patterns that appear in the L-KV set, (3) compute the frequency of different text patterns and identify the text patterns that commonly occur (if their frequencies exceed a threshold), (4) use these text patterns to search the

whole data set to recognize more *LOCATION* key-value pairs, and (5) update the L-KV set and then go back to (2). This process is repeatedly performed, each time with higher frequency thresholds to ensure high reliability, until the *LOCATION* KV set converges.

Identifying the initial seed set is the necessary first step for snowball bootstrapping algorithms [3]. Choosing different seeds should only impact the results slightly if the frequency thresholds are carefully tuned. In our implementation, the initial rule set contains an average of 7.5 seed rules per data type. The data type of Device has most seed rules (14) as the text patterns are diverse, while data type of IP has the least seed rules (4).

ALGORITHM 1: MobiPurpose's bootstrapping algorithm for data type inference

Input: A large collection of key-value pairs R and a list of data types $D = \{d_1, d_2, ...\}$
Define: For any key/value string, we extract text patterns $T = \{t_1, t_2, ...\}$.
Initialization: For each data type d_i in D, we initialize a set of seed rules to extract a initial key-value set E_i.
Bootstrapping Algorithm:
Step 1: Traverse all the keys and values in E_i, and extract the patterns as T_k and T_v.
Count the frequency of unique patterns in T_{key} and T_{value}, and remove less common patterns.
Annotate the results as T'_k and T'_v.
Step 2: Search key-value pairs in R using T'_k and T'_v.
If the key matches T'_k **or** the value matches T'_v, we add that pair to E_i.
Go back to Step 1 to find new patterns, and repeat the iteration until the size of E_i does not grow.
Output: A text pattern corpus where key-value pairs are labeled with different data types.
Bayesian classifier:
Given any key-value pair, we estimate the likelihood of different data types using a bayesian method:
$P(c|k,v) = \frac{P(k,v|c)*P(c)}{P(k,v)}$
Determining the data type for that key-value pair is equivalent to finding c^* that maximizes $P(c|k,v)$:
$c^* = \arg\max_c P(c|k,v)$

3.2 Bayesian Classification

It is possible that the bootstrapping approach might miss some key-value pairs for each data type. Furthermore, running the bootstrapping iteration can be slow. To address these problems, we used the bootstrapped results as a corpus to quantify the weights of different text patterns, and built a probabilistic Bayesian classifier to infer the data type. Let $C = \{c_1, c_2, ..., c_n\}$ be the n data types (n = 16 in our taxonomy), $E = \{E_1^*, E_2^*, ..., E_n^*\}$ be the converged KV set for each data type. Given any key-value pair (k, v), we can extract the text patterns $X = x_1, x_2, x_3, ...$ to represent that pair. The conditional probability that the data type of (k, v) is c is $P(c|k, v)$. Determining the data type is equivalent to finding c^* that maximizes $P(c|k, v)$. From Bayes'rule, $P(c|k, v)$ can be formulated as:

$$c^* = \arg\max_c P(c|k,v), \qquad where \qquad P(c|k,v) = \frac{P(k,v|c) * P(c)}{P(k,v)} \tag{1}$$

in which $P(c)$ represents the prior probability of selecting c without the observation of the (k, v), $P(k, v|c)$ is the likelihood function, which expresses how probable the text patterns can be seen in data type c, and $P(k, v)$ is the normalization constant.

We model $P(c_i)$ as the percentage of each data type in all key-value pairs. For example, if MobiPurpose identifies 1,000 privacy-sensitive pairs and 30 of them are *LOCATION* pairs, $P(LOCATION)$ is 0.03. We use text patterns to represent each key-value pair; therefore the likelihood function is interpreted as the product of the possibility $P(x_i|c)$ each text pattern happening in class c:

$$P(k,v|c) \sim P(X|c) = \prod_1^n P(x_i|c) \tag{2}$$

In our traffic data set, not all data fields are necessarily privacy-sensitive (e.g., timestamp, version number). We further empirically determine a threshold. If all the likelihood predictions are below that threshold, we classify the key-value pair into the *NON-PRIVACY* category.

4 DATA COLLECTION PURPOSE (WHY) INFERENCE

Based on the data type prediction, we can find a small list of associated purpose candidates through a taxonomy lookup (Fig. 2④), and then use machine learning techniques to infer the likely purpose. To do so, we leverage our ten participants from the data labeling step (see details in §6). Specifically, after the labeling, we discussed the data patterns (DP) they observed and asked them how they determined the purpose that they labeled. Based on this discussion, we derive a number of computational features from three separate data sources: traffic requests (G.1,2,3), app descriptions (G.4) and the host domain information (G.5,6) (Table 3).

4.1 Data Patterns Observed by Labeling Participants & Features Extraction

Here we report the data patterns which our participants mentioned for inferring purposes, and how they informed our proposed features.

Table 3. The features used in classification model can be classified into three groups: embedded textual features, source app features and domain features.

Group	Feature	Details
Embedded textual features	G1: *url path bag-of-words*	A bag-of-words representation of the URL path and sent data (160~350 dimensions)
	G2: *package-endpoint similarity*	A 3 dimension vector, each value represents if the URL components share a common string with the package name.
	G3: *co-sent data types*	The length of k-v pairs and involved data types.
Source app features	G4: *app category*	The app category can be queried from Google Play using the package name (15 binary dimensions).
Domain features	G5: *owner business type*	The business categories from Crunchbase.
	G6: *domain occurrences in app corpus*	We first decompile the app file, then find the apps that contains the domain inside their source code, and count the app distribution across different app categories. We use the top app categories to represent the business type.

DP.1: *The keywords in the host path (e.g., searchnearby, fetchads) and key-value pairs (e.g., accesstoken) often describe the API behavior directly. Nearly all the participants actively looked for special keywords, such as "ad" and "analytic", during the labeling process.* Given any endpoint address (host+path), we parse the address into components by the URI protocol (i.e., *scheme :// subdomain.domain / path / document . extension ? query # fragment*), segment each component (except scheme) into words using the probabilistic model described in Section 3, and encode the results into a "bag-of-words" representation (i.e., number of word occurrences in the URL). We maintain a stopword list, such as "com", "www", "android", "google", to filter out some unnecessary features. We also apply the same approach to the key-value pairs and merge the results into one "bag-of-words" representation.

DP.2: *The string similarity between the app package name and the domain name can indicate if the app is connecting to third party services. For example, the app "net.passone.gwabangeng" contacts "api.passone.net/ggwabang/questionApi.php", suggesting that the endpoint is not a third party service.* We segment the endpoint address into three parts (subdomain, domain, path), and compute the longest common substrings between the package name and the three components separately. We then use a vector to describe this data

pattern, with each item in the vector representing if the corresponding component contains a non-stop-word common string longer than four characters.

DP.3: *The co-sent data can help determine the purpose. For example, advertising services often collect ID as well as screen size, since they need to determine the visual appearance of the advertisement. Analytic services often collect key-value pairs (10+) more aggressively than the rest.* We use a numerical vector to represent the number of total key-value pairs as well as the number of different data types of key-value pairs.

DP.4: *Participants often read the app description in Google Play to understand the network request context.* In this feature group, we only use the app category in Google Play. To reduce unnecessary features, we merge all the game categories to reduce the feature dimensions [1].

DP.5: *Understanding the service provider (domain owner) business can help the purpose inference. For example, ID collected by "doubleclick.com" are most likely for advertising purpose, since 'DoubleClick' is a subsidiary of Google which provides ad services. Participants often check Google Search, WHOIS to infer the business type.* We developed a set of scripts to automate the domain owner lookup using WHOIS API[2] and Crunchbase API[3]. We use the corporate categories in Crunchbase to represent the organization business type.

However, most international companies do not have a profile on Crunchbase and many domain owners choose "private registration" to hide their WHOIS registration information. We were only able to identify 22% domains using that approach. To address this issue, we leverage the app source code to represent the business types. The intuition is that if an app contacts a domain, the app category can indicate the business type of the domain owner. For example, *uber.com* is contacted by five apps in our network tracing dataset, three in the "Maps & Navigation" category and two in the "Travel & Local" category. We can use these two categories to describe Uber's business. This approach can also identify the domains providing third party services since they are contacted by apps across all the categories.

We extract this domain feature as follows. We first decompile all the 185k apps using Androguard [17] and index the domains in the decompiled source code. For each domain, we find the apps that contain the domain inside their source code and count the app distribution across different app categories. We use two heuristic rules to extract features representations: 1) if the domain is contacted by 10+ apps across 5+ categories, we set the business type to "third party library"; 2) otherwise, we use top 3 app categories to represent the business type.

4.2 Feature Selection

A simple bag-of-words model produces too many features which may lead to overfitting. We assume the low-frequency features are less generalizable, so we filter out all the features with a document (i.e. traffic request) frequency below 3%, resulting in feature vectors with a dimension between 160 and 350 for different data types.

4.3 Supervised Machine Learning

We use supervised machine learning to train a purpose classifier based on the proposed features. We assume the heterogeneous data sources contribute similarly to the classification process, so we set all the weight components to 1.0. We maintain an independent classifier for each data type and experimented three different classification algorithms: Support Vector Machine using a linear kernel (SVM), Maximum Entropy (ME), and C4.5 Decision Tree (C4.5). In §7.2, we compare the performance of different algorithms and different feature combinations.

5 DATA COLLECTION

In this section, we describe the design and implementation of our network tracing system (Fig. 4). The goal of this network tracing system is to make it fast and easy for us to collect diverse network data from a large set of apps in the lab.

[2] https://www.whoisxmlapi.com/ [3] https://data.crunchbase.com/docs

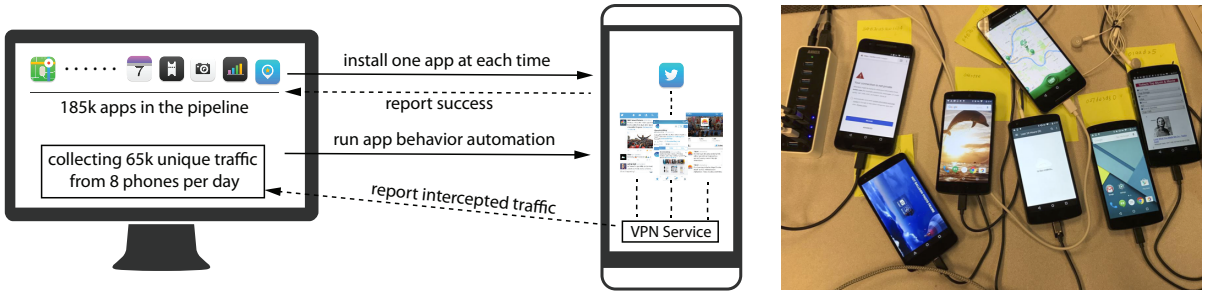

Fig. 4. Scalable Network Tracing. Left: the pipeline of the network tracing. Right: the hardware configuration. We install the app one by one on eight Android devices, connecting to a PC running app interface real-time analysis. For each app, we will explore the app behavior for 3 minutes and uninstall the app afterward. This configuration can intercept around 65,000 unique traffic requests every day.

Our tracing system is comprised of two main components: a UI automation test tool and a network sniffing app. We implemented the UI automation tool using DroidBot [41], a lightweight UI-guided test input generator, which analyzes the user interface on-the-fly and randomly traverse the UI elements on the screen. We then built our network sniffing app by using the internal Virtual Private Networking (VPN) service provided by Android OS. We manually added a trusted root certificate to our test devices and performed a Man-in-the-Middle (MITM) SSL injection to decrypt SSL/TLS traffic. Our architecture made it easier for us to capture which app initiated the network request, which would have been more difficult if we had used a server proxy approach (e.g. [70]). For each traffic request, we recorded the destination domain, path, source app, and the network request body.

We tested a large collection of Android apps using this tracing system. We crawled the Google Play web pages in November 2016 to create an index of all the apps that were visible to US users, among which 1.5 million of them were free apps. We decided to only focus on free apps updated after 2015, resulting in 185,173 apps.

For each app, our automation tool first explored the app behavior for 3 minutes and then uninstalled the app afterward. The 3 minutes duration is a tradeoff between our computation resources and our goal to collect as many unique traffic requests as possible. Our hardware configuration (Fig 4 right) consisted of 1 PC and 8 Android phones. Roughly, using this set up, we could intercept around 65k network requests and 3k apps every day.

Descriptive statistics. Our data collection process took 50 days to traverse the 185k apps. Due to OS compatibility (our devices were running on Android 7), we were only able to install 30,075 apps. In total, we intercepted 2,008,912 traffic requests from 14,910 apps, contacting 12,046 unique domains (302,893 unique end-point URLs), sending 6,376,833 key-value pair data to a remote server. The top 3 active domains were *doubleclick.net* (261048 requests), *googlesyndication.com* (158672 requests), and *startappservice.com* (124289 requests). The number of requests across domains followed a long tail distribution: 9320 (77.3%) domains were only contacted by one app, and 5653 (46.9%) domains were contacted less than 10 times. Figure 5 illustrates a more comprehensive view of the traffic distribution.

Preprocessing the dataset. If we study the individual request directly, the final system would be biased to top domains like *doubleclick.net* and less adaptive for the wild traffic requests. So we first filtered out traffic requests that sent an empty data body and then merged repetitive traffic requests. We grouped requests based on the end-point address (i.e. host + path) using manually crafted regex expressions, resulting in 60,753 unique network API traces. For example, we use `graph.facebook.com\/v\d.\d\/\d{15}` to group similar traffic requests sent to facebook graph API, and merge their parameters (i.e. key-value pairs).

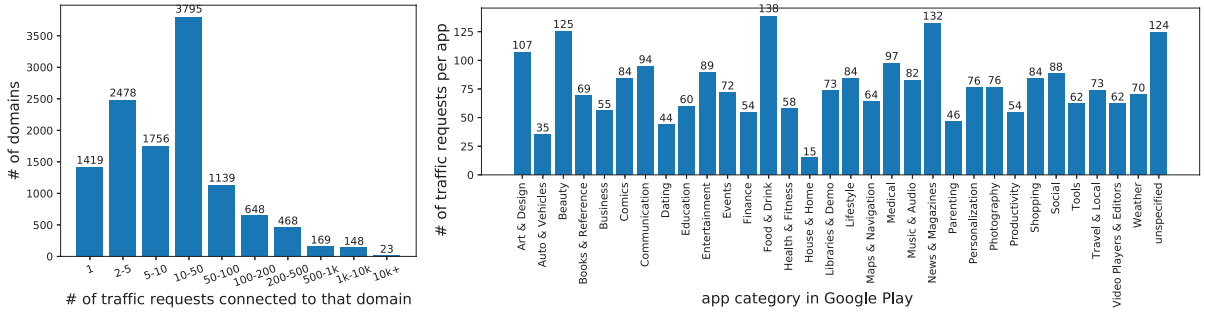

Fig. 5. Data stats of the 2 million outing traffic requests. The left figure illustrates the long tailed distribution of domains in the data set. The right figure shows that the apps in different categories generate similar numbers of traffic requests.

6 LABELING THE GROUND TRUTH BEHAVIOR OF TRAFFIC REQUESTS

Our goal of data labeling was threefold: 1) to evaluate the accuracy of data type classification, 2) to test the taxonomy completeness, and 3) to prepare a data set for purpose inference. We designed each labeling task as a short set of questions about a specific traffic request, with a focus on one key-value pair (Fig. 6). We presented participants raw traffic requests as well as the data type classification results. We then ask them to judge if the classification is correct, and label the data collection purpose.

In the pilot test, we found judging the data type was straightforward for engineers, while labeling the data collection purpose required more contextual knowledge [16, 56]. To accommodate that, we incorporated multiple shortcuts (e.g., Google Play, WhoIS) to help participants access relevant external information quickly.

In the example illustrated in Figure 6, a participant can find that the app is a shopping coupon app named “The UOL Club” from the Google Play link, infer that the traffic is sent to a UOL server. Combining with the host path "/coupon/category/nearby" and the data body, she can then infer that the traffic request is to "Search Nearby" using longitude and latitude information.

Procedure. To avoid test data leakage [35], we randomly sampled 5k API traces from the preprocessed dataset for evaluation. We developed our bootstrapping algorithm based on the remaining 56k API traces, and run the final data type inference algorithm on the 5k reserved API traces. The labeling instances were from the 5k reserved API traces. Our labeling strategy was similar to Wang et al. [71]. We tried to label at least 30 instances for each purpose manually. However, some purpose instances are scarce (<5%) in our traffic data set. For example, we only observed 5 instances of "reverse geocoding" among 250 location instances. We stopped once we got 30 instances for common purposes (>=5%).

Participants. We recruited ten engineering graduate students with prior Android app development experience. All participants had over three years of experience in Android/iOS development, six were Ph.D. students in computer science, and four are familiar with reverse engineering and mobile privacy research. Participants are also aware of the GPS location of the lab, so they can recognize if the set of coordination is the nearby location.

We introduced our taxonomy in a five-minute labeling tutorial. Since the purpose interpretation can be subjective and in some cases ambiguous (e.g. due to insufficient or incomplete information), we collect three independent labels for each task and allow participants to label multiple purposes for each entity if needed. The purpose labeling is quite time-consuming. Each labeling task takes between 30 seconds to 2 minutes since the participant often needs to generate search queries and check the facts on external websites. We organized 2 group data labeling hackathons with free food, each lasting around 5 hours. No monetary remuneration was paid.

Labeled data stats: We collected labels from 3,177 (1,059 x 3) labeling tasks, where each entity was labeled by three independent human experts, covering 7 data types and 34 data purposes (Table 4). The data type annotations

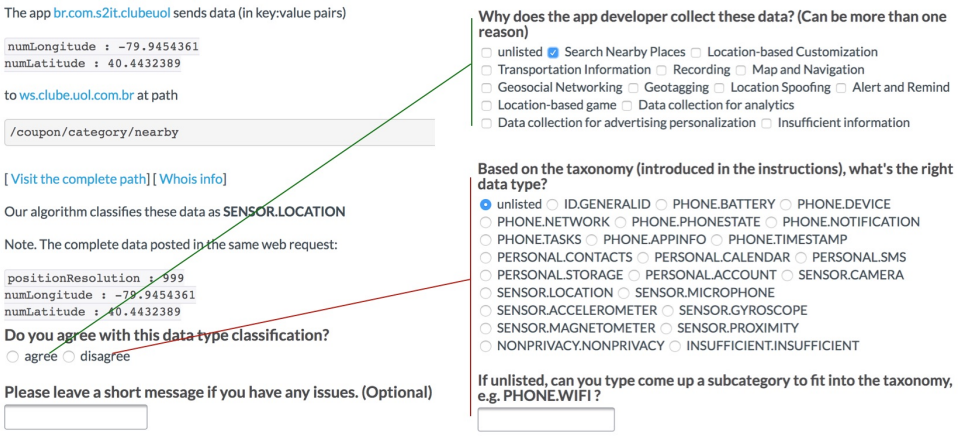

Fig. 6. An example labeling task. We classified the "numLongitude" and "numLatitude" key-value pairs as *LOCATION* data. Participants determined if our automated classification is correct. If correct, participants continue by labeling the data collection purpose. Otherwise, participants are asked to select the correct data type. In this example, "The UOL Club", a shopping coupon app, sends *LOCATION* data to the host path "/coupon/category/nearby" on a UOL server. So the purpose is "Search Nearby Places."

had a very high inter-annotator agreement: only 23 traffic requests (23/1059 <2%) received inconsistent annotations, giving us confidence that the quality of labels is high and mostly consistent.

Table 4. The examples of different data type and the # of labeled instances.

data type	#labels	examples	data type	#labels	examples
ID	400	MAC address, uuid	Battery	7	battery level, if charging
Device	150	phone model, screen size	Network	100	IP address, wifi, lte, RSSI
Running state	2	foreground tasks	Account	150	email, age, gender, zip
Location	250	GPS coord, place name			

Total: 1059

The purpose labeling results further illustrate the feasibility of inferring data collection purpose from the embedded attributes of the app and external factors such as where it contacts. In the labeling tasks, the participants can choose to annotate it with "Insufficient Information" (II) if they find there is not enough information to support their judgment. 123 (123/3177 < 4%) labeling tasks were marked as such, among which only 3 entities were marked as II by all three participants, and 20 entities were marked by two participants.

We merged the purpose labels from different participants using majority voting. If at least two independent participants believe the key-value pair is associated with a specific purpose, we accept that label. Based on this standard, there were 118 traffic requests (118/1059 < 12%) that are too ambiguous to reach an agreement. We excluded these traffic requests in the later purpose inference evaluation.

Taxonomy in Practice: The labeling interface also allows participants to add a new purpose label if they find the behavior is not covered by the current taxonomy. Our participants nominated new purposes in 30 labeling tasks (1.2%), resulting in 4 new categories after re-coding: "Network Info - Advertising", "Location - Reverse Geocoding", "Location - Malicious", "ID - Malicious". We incorporate the first two purposes in our final taxonomy and have a separate discussion regarding malicious purposes (Sec. 10.2).

Based on the labeling tasks, we believe the current taxonomy has a good coverage for most regular network traffic. The described process can be repeated from time to time if we need to add/update the taxonomy.

7 EVALUATION

MobiPurpose uses a two-step procedure to infer purposes. MobiPurpose first infers the data types, using it to find the associated purpose candidates through a taxonomy lookup. Then, MobiPurpose uses a supervised machine learning approach to predict the data collection purpose. Here, we evaluate both the data type inference and purpose inference, and found that MobiPurpose can achieve an average accuracy of 95% for data type inference and 84% for data purpose inference.

7.1 Data Type Inference Performance

7.1.1 Methodology. To evaluate the performance of data type inference, we measure the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) for each data type, and present our results regarding precision, recall, and f-score.

We define precision as the fraction of true predictions (TP) among all the predictions (TP+FP) regarding one class. In the labeling tasks, participants determined if our data type inference was correct and provided the correct label if the inference was wrong. We define recall as the fraction of true predictions (TP) among all the true instances (TP+FN) regarding one class. We manually inspect all the fields classified as *NON-PRIVACY* to determine if any privacy-sensitive key-value pairs are missed. We label obfuscated key-value pairs as *NON-PRIVACY*. F-score is the harmonic mean of precision and recall.

To measure the overall correctness of our approach, we use two standard multi-class accuracy metrics: *micro-averaged* (equal weight for each instance) and *macro-averaged* (equal weight for each class) [46, 71]. For micro-averaged metrics, we first sum up the TP, FP, FN for all the classes, and then calculate precision and recall using these sums. So classes that have many instances are given more importance. In contrast, macro-averaging is the mean of the metrics for all the individual classes. Since we do not have enough data samples in *Battery* and *Running State*, we exclude them in computing the macro-average.

7.1.2 Results. Our results for classifying data types are shown in Table 5. Our approach has precisions above 93% for all the classes and an overall precision (micro-averaged) of 95.9%. For recall, our approach identifies over 86% of the privacy-sensitive entities for all the categories with an overall recall (micro-averaged) of 89.9%.

Most FP instances are due to the similar text patterns shared between different classes. For example, *gps_adid* is classified as a location data type but is actually an ID. Reasons for TN are diverse: some personal data are nested in a serialized dictionary; several unusual variable names (e.g., using a token to name ID data) have not been captured by the bootstrapping algorithms, etc.

Table 5. Data type inference accuracy

	ID	Battery	Device	Network	State	Account	Location	**Macro-avg**	**Micro-avg**
Precision	97%	100%	93%	94%	100%	96%	96%	95.6%	95.9%
Recall	86.8%	100%	87.5%	92.1%	100%	92.3%	95.2%	90.8%	89.9%
F-score	0.925	1.00	0.902	0.930	1.00	0.941	0.956	0.931	0.928

7.2 Purpose Inference Performance

7.2.1 Evaluation Metrics & Methodology. We use 10-fold cross validation to evaluate the performance of purpose inference. For the purpose predictions of each key-value pair, we measure the accuracy, precision, recall and f-score as we did in the data type inference evaluation.

To measure overall performance, we aggregate the individual metrics using micro-average and macro-average for both precision and recall. Since we do not have enough data samples for the Battery and Running State data categories, we only cover the other five data categories (i.e., ID, Device, Network, Account/Profile, and Location) in the purpose evaluation. Due to the imbalance of purpose distributions, some purposes are not common in our dataset. For example, we only have 5 SENSOR.LOCATION.ReverseGeoCoding instances in the labeled data set. We opt to run the classification only among the purpose categories with more than 20 instances.

7.2.2 Results. Our results in classifying the purpose of different data types are shown in Figure 7. The Maximum Entropy algorithm performs the best, with an overall accuracy (macro-average) of 84% for all the categories, while SVM outperforms ME in Account/Profile and Device categories. Overall, all the algorithms perform reasonably well across all the data types. MobiPurpose has a comparable accuracy as [71] (85%), which infers the Android permission purposes by analyzing the source code, though MobiPurpose treats the app as a black box.

	SVM	**ME**	C4.5
Precision	0.85	**0.88**	0.83
Recall	0.84	**0.85**	0.82
F-1 Score	0.84	**0.86**	0.82

(a) ID.

	SVM	ME	C4.5
Precision	**0.82**	0.82	0.81
Recall	**0.81**	0.79	0.80
F-1 Score	**0.81**	0.80	0.80

(b) Device.

	SVM	**ME**	C4.5
Precision	0.82	**0.86**	0.84
Recall	0.81	**0.86**	0.84
F-1 Score	0.81	**0.86**	0.84

(c) Network.

	SVM	ME	C4.5
Precision	**0.85**	0.84	0.84
Recall	**0.85**	0.83	0.83
F-1 Score	**0.85**	0.84	0.84

(d) Account/Profile.

	SVM	**ME**	C4.5
Precision	0.79	**0.82**	0.80
Recall	0.79	**0.82**	0.80
F-1 Score	0.79	**0.82**	0.80

(e) Location.

	SVM	ME	C4.5
Precision	0.83	**0.84**	0.82
Recall	0.82	**0.83**	0.82
F-1 Score	0.82	**0.84**	0.82

(f) Overall (Macro-avg).

Fig. 7. Performance of different machine learning models (using all the features) for classifying purposes

Figure 8 shows more details about misclassifications across all the data types. The micro-average accuracy of 82.3% (Fig. 8f) is similar to the macro-average accuracy. However, the specific classification errors across different categories vary greatly.

For ID data (Fig. 8a), the category "Ad" achieves the best result, with precision and recall both higher than 90%, which is due to the domain name as a feature in recognizing the business types of the domain owner. The category of "Authentication" purposes have 94% precision but recall under 60%. The main misclassifications are in classifying "Sign-out personalization" and "Authentication" into "Analytics," which may because these three purposes collect similar types of data.

For Device data (Fig. 8b), the main classification errors come from the confusion between "Ad" and "Analytics": 11 "ad" (35%) instances are classified as "analytics". Similar situations also happen to the Account/profile data (Fig. 8d): 8 "ad" (23%) instances are classified as "analytics". In the location matrix (Fig. 8e), the major confusion happens between "Location-based personalization" and "Search nearby," which may be due to the semantic similarity between these two purposes.

Through inspecting these individual misclassifications, we categorize errors into two types. First, developers use diverse vocabularies for purpose instances such as “sign-out personalization” and our model didn’t capture the statistical patterns of these less common keywords. For example, one term in a “sign-out personalization” instance is “visualdna.” Continuing to increase size and diversity of the labeled data set will help here. Second, the semantic correlation between different purposes, such as “Ad” & “Analytic,” or “Location-based personalization &

		P1	P2	P3	P4	P5	Total
Anti-fraud	P1	26	-	-	1	4	31
Authentication	P2	-	16	1	3	7	27
Personalization	P3	3	1	8	1	11	24
Ad	P4	-	-	-	162	15	177
Analytics	P5	-	-	1	11	114	126

(a) Confusion Matrix for ID

		P1	P2	P3	Total
Ad	P1	19	11	1	31
Analytics	P2	3	62	2	67
Interface	P3	3	6	19	28

(b) Confusion Matrix for Device Information

		P1	P2	P3	Total
Ad	P1	20	3	3	26
Analytics	P2	4	29	4	37
NetOptimization	P3	3	1	24	28

(c) Confusion Matrix for Network

		P1	P2	P3	Total
Ad	P1	26	8	1	35
Analytics	P2	5	59	3	67
Login	P3	2	6	17	25

(d) Confusion Matrix for Account/Profile

		P1	P2	P3	P4	Total
Ad	P1	102	4	3	-	109
Analytics	P2	4	26	4	3	37
Personalization	P3	2	3	16	7	28
Search Nearby	P4	-	4	4	25	33

(e) Location

		Prediction	
		Pos	Neg
Truth	Pos	770	166
	Neg	166	2683

(f) Overall (Micro)

Fig. 8. Detailed confusion matrices using all the features with Maximum Entropy algorithm. Each column represents the instances in a predicted class, while each row represents the instances in an actual class. Note, the P1-Pn shown in each table denote the different purposes, and they vary for each feature (a)-(e). For example, P1 is “Anti-Fraud” for the ID feature, but is “Ad” for the Network feature.

“Search nearby,” also imposes extra challenges in classifying the purposes. This semantic noise is not introduced only from the data labeling bias, but also the developer’s naming decisions.

The classification results in Figure 8 is biased to "Analytics", since there are many "Analytics" instances. We also run an evaluation based on a sampled dataset, where each purpose has the same number of instances. This experiment shows similar performance results with an average accuracy of 81%. We report the original data as it fits better the real-world situation.

7.2.3 Feature Comparison. We perform component evaluations to find the efficacy of different feature groups (Table 3). We use the bag-of-words features (G.1) as the baseline and experiment with different feature combinations. All the models are trained with Maximum Entropy algorithm. The overall experimental results are summarized in Table 6. We can see the bag-of-words features (G.1) alone can achieve an accuracy above 74% and the rest of the features play supporting roles.

Different supporting features illustrate different performance gains in different data categories. The domain features (G.6) improve the performance by 12% in the "ID" category but has little impact on the performance in "Device" and "Account" categories. The co-sent features (G.3) do not perform well in general but is the most valuable feature in “Device” category.

Inspecting the individual results, we find that different categories suffer from different ambiguities. For example, it is hard to differentiate “ad” instances from “analytics” instances in the “ID” category using just the URL. Domain features would be the primary differentiator in this case. In contrast, “ad” services may also collect “Device” information for “analytics” purpose as well. In this case, domain features do not help much in resolving that

confusion. Instead, "analytic" services usually collect more key-value pairs than "ad" service, so G.3 (co-sent data types) is the most helpful feature in "Device" category.

Table 6. F1-score performance across different feature combinations (Maximum entropy algorithm). Bag-of-words features (G.1) achieve good baseline accuracy alone, with domain features (G.6) offering significant improvements. Using only G1,6 can have a similar performance as using all the features.

	G.1	G.1,2	G.1,2,3	G.1,2,3,4	G.1,2,3,4,5	G.1,2,3,4,5,6	G.1,6
ID	0.74	0.74	0.74	0.75	0.78	**0.86**	**0.86**
Device	0.76	0.77	**0.81**	**0.81**	**0.81**	**0.81**	0.80
Network	0.81	0.81	0.77	0.74	0.76	0.80	**0.86**
Account	0.83	**0.84**	0.81	0.81	0.81	**0.84**	0.82
Location	0.77	0.79	0.78	0.76	0.78	0.82	**0.83**

8 RELATED WORK

MobiPurpose is mainly related to three major approaches to characterize mobile data access and disclosure: static analysis, dynamic analysis, and network analysis. We organize the relevant work in Table 7 & 8, and discuss these areas in detail below.

8.1 Network Analysis

Table 7. Network analysis research at different granularities using different techniques

		Traffic request	Key-value Pair
Network Analysis	Differential test	Zhang et al. [78], AntMonitor [37]	PrivacyOracle [34], PrivacyProxy [66]
	Text-based	PrivacyGuard [65], Zang et al. [77]	Recon [61], **MobiPurpose**

Many commercial tools [14, 27, 64, 76] have been developed to support analysis of network flows and identification of leaks of potentially sensitive data. For example, VIP Defense [14] and Forcepoint [27] can detect and block leaks of personal information (e.g., credit card number and social security number) automatically.

Past projects have mainly looked at two granularities of network flows: individual traffic requests [60, 78] or specific data fields inside a request. Zhang et al. [78] is an example of the first type, which labels the entire network request as legitimate if the request is associated with a UI event. In contrast, projects like Recon [61] parse the request body into key-value pairs and check if the request contains some specific types of privacy-sensitive data.

While looking at data egress is a good vantage point, searching for privacy-sensitive data in large quantities of network data is still challenging. Black box differential testing is commonly used to tackle this issue [12, 34, 66]. It first establishes a network behavior baseline of an application. It then modifies some specific data on the device and detects leaks by observing deviations in the resulting network traffic. However, this approach is mostly limited to recognizing personally identifiable information and does not generalize to other kinds of sensitive data (e.g., device model, screen resolution, etc.). An alternative choice is leveraging text patterns in traffic requests, which use either hard-coded regular expressions [37, 65] or supervised machine learning approaches [61] to classify network flows.

In contrast, MobiPurpose analyzes the key-value pairs inside a traffic request by leveraging text patterns [61] as well as external knowledge (e.g., app description, domain owner business, etc.). More fundamentally, we not only categorize the data types in a fine-grained manner, but more importantly, classify the purpose of each key-value pair.

8.2 Dynamic and Static Analysis of Mobile Apps

Table 8. Using dynamic and static analysis to analyze mobile privacy at different granularities.

	App	Library	Permission	Data flow
Dynamic Analysis		Chitkara et al.[11] Han et al. [29]		PmP [2], TaintDroid [24] AppsPlayground [59]
Static Analysis	PrivacyGrade [36] CHABADA [28]	Amandroid [73]	Pscout [5], WHYPER [57] AutoCog [58], ASPG [72] Wang et al. [71]	PiOS [23] DroidJust [10]

Dynamic analysis tracks the flow of privacy sensitive data at runtime. By modifying the device OS, solutions like TaintDroid [24] can label (taint) data from privacy-sensitive sources and transitively apply these labels as the data propagates through program variables, files, and interprocess messages. In this way, the system can track the data flow, namely which app transmits the data and where the data is being sent to. However, this approach suffers from large false positives (due to coarse tainting granularity) and false negatives (e.g., the data does not leave the device) detection issues. The significant runtime overheads and the compatibility of modified OS are also barriers to widespread use.

Static analysis detects *potential* privacy leaks through analyzing the source code (e.g., permission file analysis [5, 36, 43], data flow analysis [10, 23], or text pattern analysis [71]). For example, PrivacyGrade [36] inspects the permission files and app descriptions to identify unnecessary permissions, since these permissions may lead to privacy leaks eventually. This approach is designed to be more scalable than dynamic analysis since it can avoid runtime overhead without code execution [40]. However, it is still hard to infer the privacy context using static analysis, e.g., where the data is sent to. Besides, running symbolic execution is very time-intensive, and loading code dynamically is increasingly common [45].

Past research has used these two approaches across a range of different granularities (see Table 8). For example, PrivacyGrade [36] grades the privacy behavior of each app by measuring the gap between people's expectations and the app's actual behavior. Chitkara et al. [11] use code injection to infer whether the data access is by a third-party library or by the app itself for its functionality. To justify if the data access is legitimate, DroidJust [10] uses static taint analyses to link each data flow with certain application function.

Both approaches can be complementary to the network analysis approach [37, 61]. For example, AntMonitor [37] uses the dynamic analysis to cross-check with the result of network analysis. MobiPurpose leverages the idea of static analysis in feature extractions (Sec. 4). We decompile the app file into source code, and count the number of occurrences of domain names across different apps to infer the information of the domain owners.

9 LIMITATIONS

9.1 Network Tracing Coverage

Gathering large-scale comprehensive data from mobile apps is a challenging task [39, 60]. The UI monkeys approach used in our paper can collect a large dataset with relatively small computing resources [9, 42]; however, it also suffers from the issue of lacking comprehensiveness. For example, text entry box validation (e.g., log in screens) can impede the monkey's progress through an app [60, 62]. In our experiment, we also find monkeys cannot parse the Unity[4] interface in some game apps because it's hard to parse the UI element tree.

We have implemented several heuristics[5] to mitigate these issues. First, we manually login into the popular apps (e.g., Yelp, Twitter, Google) to avoid the login screens in these apps. Second, we detect third-party login options (e.g., log in with your Google account) and select these options if available. A recent study shows that

[4] https://unity3d.com/ [5] Open sourced at https://github.com/CMUChimpsLab/MobiPurpose

more than 40% mobile login screens support third-party login functionalities [47]. Third, our monkey clicks different coordinates randomly if the app interface is written in Unity.

While developing a better monkey alternative is beyond the scope of this paper, we conducted a preliminary experiment to understand the network tracing coverage. We selected the top 20 popular free apps and manually logged into the apps before the study. We then tested the apps under 3 conditions: manual interaction (3 minutes), monkey interaction (3 minutes), and monkey interaction (15 minutes). We tested the monkey interaction at a longer duration since the monkey duration can be scaled up with little extra resources in practice.

Table 9. The Mean (SD) of detected unique network API traces per app under different conditions.

	Manual interaction (3 min)	Bot interaction (3 min)	Bot interaction (15 min)
# of unique requests	45.2 (20.3)	33.1 (16.7)	40.8 (23.5)
# of unique purposes	5.4 (2.2)	3.1 (1.8)	3.9 (1.9)

Table 9 illustrates the statistics of unique network API traces detected across different conditions. In general, manual interaction has the best coverage regarding unique requests and unique purposes. The coverage of monkey interaction will increase if we extend the test duration (15 min vs. 3 min). The network traces observed for the same app in 3 conditions are not fully overlapped, so we cannot compare the recall directly. We empirically noticed that monkey interaction generates less diverse traffic than manual interaction, and we observed more ad/analytic traffic requests in the monkey interaction dataset. Future works may consider incorporating app interface semantic knowledge [15, 40] to improve the bot implementation.

9.2 Apps Requiring Login

To further understand the login limitation, we studied how much the login blocking will impact the bot interaction. We manually installed the top 80 free apps in US Android stores, found 41 of them support login and 22 of them support Google/Facebook/Twitter login. Among the remaining 19 apps, 15 apps (e.g., Chase Mobile, Robinhood) requires a critical login to interact with core functionalities. Most critical login apps appear in the top 40 apps (13/15), while only 2 apps between 40-80 require a critical login. Modern mobile app design guidelines often suggest registration is a road-block to adoption [6]. Forcing registration too early can cause more than 85% of users to abandon the product [63]. We expect to see fewer apps require a critical login in "lower-ranked" apps.

Beyond the technical perspective, automated logins with fake profiles can be a legal grey zone, which may raise two legal concerns: 1) packet sniffing while accepting terms and conditions and 2) login with fake profiles. First, logging into a service generally requires accepting terms and conditions, which may ban the packet sniffing analysis. The banning of packet sniffing appeared in the early End-User License Agreement (EULA) and received strong opposition [52]. Recent EULA templates [69] and Terms of Service employed by major apps (e.g., Google, Facebook, Yelp) only prohibit source code reverse engineering. Second, login with fake profiles has been heavily used in web/mobile data scraping [13]. Most recently, a US court ruled that creating fake profiles can be protected by The First Amendment [49]. Future work should pay attention to these potential legal implications as well.

9.3 Traffic Obfuscation

Our premise is that mobile network traffic textual data, the app source, and the domain information reflect underlying developer purposes. However, developers might deliberately or unintentionally have obfuscated names. For example, we noticed some apps have unclear key names like "v2", "c12", and these key names might not be consistent across different traffic requests. We found 218 out of 12,046 (1.8%) domains and 405 out of 14,910 (2.7%) apps have some form of obfuscation.

9.4 Certificate Pinning

Our current implementation of MITM SSL injection cannot intercept certificate pinned traffic. Using modified OS that disables SSL certificate checking at the system level can potentially mitigate that issue [74]. In our early experiments, we found that this approach only works for the apps developed with Android SSL libraries. So we decided to use the original OS for better compatibility. To quantify the problem of certificate pinning, we selected the top 500 free apps from Google Play retrieved on March 13th, 2018 and found only 22 apps (<5%) used certificate-pinning on all network requests [66].

10 DISCUSSION

10.1 Theoretical Framework: Contextual Integrity

Our work can be thought of helping contribute to the growing body on privacy as contextual integrity [53], which argues that a data access or disclosure is legitimate based on the specific context and norms in which the information flow happens. In 2012, the White House further espoused this framing in the Privacy Bill of Rights [31]: *"consumers have a right to expect that companies will collect, use, and disclose personal data in ways that are consistent with the context in which consumers provide the data."*

Some of the ideas behind contextual integrity have started to diffuse into research in mobile computing (either deliberately or through convergent evolution). One example is the app permission system (Fig.1). Researchers have also started to look at ways to incorporate users' privacy expectations [4, 38, 43, 57, 58, 71]. For example, WHYPER [57] and AutoCog [58] build a machine learning model to determine if the purposes of permissions are consistent with the app description. ASPG [72] and CHABADA [28] identify outlier permissions by clustering similar apps based on app descriptions. Beyond binary anomalous detection, Lin et al. [43] and Wang et al. [71] framed mobile privacy in the form of people's expectation about the data collection purpose.

There has been a rich literature in studying the benefits of understanding "why" [48, 54, 70] in network traffic. For example, Van Kleek et al. [70] manually labeled the network traffic purposes and found purposes can help users make confident and consistent choices. In contrast to prior work, MobiPurpose is the first solution that can automate the mobile traffic purpose inference.

10.2 Taxonomy Motivation & Completeness

Our taxonomy does not capture if a data disclosure is legitimate or malicious. The question of whether a traffic request is legitimate or not can only be answered in each specific context in which the question arises. But attempts to answer this question are challenging because of confusion about defining the troublesome activities that fall under the rubric of privacy. Our taxonomy will aid us in analyzing various privacy problems so we can better address them and balance them with opposing interests.

We have created a taxonomy of 16 privacy-sensitive data types and 76 purposes, and 10 participants used this taxonomy to label 1059 instances. While our taxonomy is good enough for our purpose, it is possible that there are other purposes that we did not find. Further, depending on how purposes are used, our taxonomy might be too fine-grained or too coarse-grained. For example, we can further branch the data analytics purpose into developer analytics (e.g., crash reports) and marketing analytics (e.g., marketing attribution analysis). However, we believe that the proposed approach should generalize for new purposes and data types.

10.3 Apps from Non-English Speaking Developers

Though we crawled apps from US Android app market, we noticed that many apps are from non-English speaking developers. During data labeling, participants used Google Translate to understand the app functions and the developer's business type. One challenge of the inference pipeline is that developers with different language backgrounds may have different naming conventions. The bootstrapping algorithm works across different

languages since it only identifies repetitive patterns [50]. For example, different traffic requests will send the same GPS decimal degree values (e.g., 41.40338, 2.17403) with some common key names. These key names will become parts of the bootstrapping extraction. In fact, “weidu”, the Chinese pinyin romanization of “latitude,” was captured in our experiment.

11 CONCLUSION & FUTURE WORK

In this paper, we present the design and implementation of the first system that can automatically categorize data collection purposes of sensitive data in mobile network traffic. The core of our solution is our data-type dependent purpose taxonomy, in which we enumerate the potential purposes associated with each data type. Given any key-value pair in the traffic request, we first infer the data type using a bootstrapping NLP approach and then use a supervised machine learning approach to predict the data collection purpose. We evaluated our approach using a dataset cross-labeled by ten human experts. Our experiments show that our approach can predict "what" with an average precision of 95% (among 8 unique categories) and "why" with an average precision of 84% (among 19 unique categories).

Past research shows that the privacy indicators that surface the purpose of data collection can help users make privacy decisions [36, 43, 70]. As such, our next step is to scale up our analysis even further and to build a public resource that can help developers, end-users, journalists, and policy makers better understand which app is collecting data about us, what data, where that data is going, and why it is being collected.

A APPENDIX: COMPLETE MOBIPURPOSE PURPOSE TAXONOMY

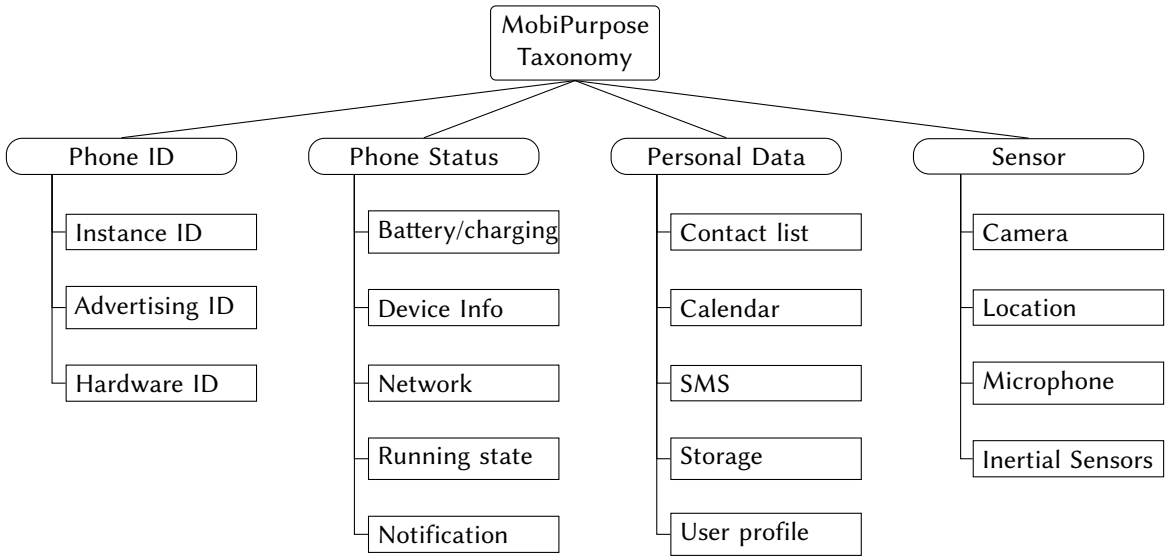

Fig. 9. Taxonomy overview for Data Type Groups

Table 10. The purpose taxonomy for *Phone ID*.

Data types (what)	Purposes (why)	Example usages
Instance/hardware/ advertising ID[6]	Tracking for advertising	Support ad targeting/evaluation
	Tracking for data analytics	Avoid redundant device counting in marketing.
	Signed-out personalization	Personalize news for sign-out users
	Anti-fraud	Enforce free content /advertisement limits
	Authentication	Relogin a user with a cookie

Table 11. The purpose taxonomy for *Phone Status*.

Data types (what)	Purposes (why)	Example usages
Battery/charging	Battery-based event trigger	Show charging/low battery notifications
	Power management	Adapt the phone settings to save battery
	Data collection for analytics	Analyze the battery assumption of apps
Device Info	Interface customization	Customize the interface based on the resolution
	Data collection for ad	Collect data for ad personalization
	Data collection for analytics	Collect data for marketing analysis
Network	Network switch notification	Show wifi/lte switch notification
	Network optimization	Download low resolution images when on LTE
	Geo localization	Use IP to infer the geo location
	Data collection for ad	Collect data for ad personalization
	Data collection for analytics	Collect data for marketing analysis
Running state	Cross-app communication	Support cross-app communication
	Task management	Detect foreground tasks
Notification	Interface customization	Customize lock screen notifications
	Interruption management	Delay notification to manage interruption

[6] MobiPurpose runs apps on in-lab devices so that MobiPurpose can identify the unique identifier (ID) types (e.g., Ad ID, MAC address, Instance ID). Official development guidelines [22] often recommend developers to use different types of ID in different contexts; however, our dataset shows that developers use different types of ID mixedly. If future developers better adopt these best practice guidelines, the taxonomy of ID purposes would be listed separately.

Table 12. The purpose taxonomy for *Personal Data*.

Data types (what)	Purposes (why)	Example usages
Contact list	Backup and Synchronization	Backup contacts to the server
	Contact management	Merge duplicate contacts
	Blacklist	Block unwanted calls
	Call and SMS	Make VoIP/Wifi calls using Internet
	Contact-based customization	Add contacts to the personalized dictionary of input typing apps
	Email	Send Email to contacts
	Find friends	Find common friends who use the same service
	Record	Display call history
	Fake calls and SMS	Select a contact to fake calls
Calendar	Context prediction	Predict if the user is commute to workplace
	Schedule	Manage schedule conflicts
	Alarm	Notify calendar events ahead
SMS	Send messages	Send messages through apps
	Organize messages	Cluster/delete/re-rank SMS messages
	Extract message content	Extract the verification code in SMS
	Block messages	Block unwanted SMS
	Schedule messages	Delay the messaging sending
	Backup/sync messages	Backup messages to the server
Storage	Access photo album	Modify or upload photos
	Manage downloaded files	Save photos to local storage
	App data persistent storage	Save configuration files
Account/user profile	Third-party login	Login through Facebook/Google accounts
	Data collection for analytics	Collect data for marketing analysis
	Data collection for ad	Collect data for ad personalization

Table 13. The purpose taxonomy for *Sensor*.

Data types (what)	Purposes (why)	Example usages
Camera	Flashlight	Turn on/off flashlight
	Video streaming	Steam the video capture
	Code scanning	Scan QR code
	Document scanning	Scan document
	Augment reality	Capture videos
	Text recognition	Recognize the text in the live video capture
	Photo taking	Take photos
Location[7]	Nearby Search	Search nearby POIs/real estates
	Location-based Customization	Fetch local weather/radio information
	Query Transportation Information	Estimate the trip time through Uber API
	Recording	Track the running velocity
	Map and Navigation	Find the user location in Map apps
	Geosocial Networking	Find nearby users in the social network
	Geotagging	Tag photos with locations
	Location Spoofing	Set up fake GPS locations
	Alert and Remind	Remind location-based tasks
	Location-based game	Play games require users' physical location
	Reverse geocoding	Use the GPS coords to find the real world address.
	Data collection for analytics	Collect data for marketing analysis
	Data collection for ad	Collect data for ad personalization
Microphone	Voice Authentication	Authenticate users using voices
	Audio streaming	Make VOIP phone calls
	Voice control	Use voice to send the command
	Speech recognition	Turn the speech audio into text
	Audio recording	Record voice messages
	Acoustic event detection	Sense users' health using microphone
	Acoustic communication	Decode audios to receive messages
	Music	Record songs
Accelerometer/ gyroscope/ magnetometer	Step-counter	Count users' steps
	Game input controller	Detect device movements to control game input
	Map/Navigation/Compass	Use dead reckoning to improve localization
Proximity sensor	Speaker/display activation	Turn off screen if the phone is near the users' ear

REFERENCES

[1] 42matters. 2018. Google Play Categories. https://42matters.com/docs/app-market-data/android/apps/google-play-categories. (2018). (Accessed on 02/11/2018).

[7] SENSOR.LOCATION in MobiPurpose only refers to the location information of the device, but not the location data in the search query.

[2] Yuvraj Agarwal and Malcolm Hall. 2013. ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 97–110.

[3] Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*. ACM, 85–94.

[4] Noah Apthorpe, Yan Shvartzshnaider, Arunesh Mathur, Dillon Reisman, and Nick Feamster. 2018. Discovering Smart Home Internet of Things Privacy Norms Using Contextual Integrity. *arXiv preprint arXiv:1805.06031* (2018).

[5] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 217–228.

[6] Nick Babich. 2016. Mobile App UX Design: Making a Great First Impression. https://uxplanet.org/mobile-app-ux-design-making-a-great-first-impression-bed2805b967d. (2016). (Accessed on 10/23/2018).

[7] A.J. Bernheim Brush, John Krumm, and James Scott. 2010. Exploring End User Preferences for Location Obfuscation, Location-based Services, and the Value of Location. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing (UbiComp '10)*. ACM, New York, NY, USA, 95–104. DOI: http://dx.doi.org/10.1145/1864349.1864381

[8] Packet Capture. 2017. Packet Capture - Android Apps on Google Play. https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture&hl=en. (2017). (Accessed on 11/10/2017).

[9] Patrick Carter, Collin Mulliner, Martina Lindorfer, William Robertson, and Engin Kirda. 2016. CuriousDroid: automated user interface interaction for android application analysis sandboxes. In *International Conference on Financial Cryptography and Data Security*. Springer, 231–249.

[10] Xin Chen and Sencun Zhu. 2015. DroidJust: Automated functionality-aware privacy leakage analysis for Android applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM.

[11] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I Hong, and Yuvraj Agarwal. 2017. Does this App Really Need My Location?: Context-Aware Privacy Management for Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 42.

[12] 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 *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*.

[13] Amit Datta, Michael Carl Tschantz, and Anupam Datta. 2015. Automated experiments on ad privacy settings. *Proceedings on Privacy Enhancing Technologies* 2015, 1 (2015), 92–112.

[14] VIP Defense. 2017. VIP Defense - a privacy and anonymity keeping company. http://www.vipdefense.com/. (2017). (Accessed on 11/08/2017).

[15] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 845–854.

[16] Nikolay Derkach. 2017. Tutorial: Reverse Engineering a Private API. https://www.toptal.com/back-end/reverse-engineering-the-private-api-hacking-your-couch. (2017). (Accessed on 11/04/2017).

[17] Anthony Desnos and others. 2011. Androguard. *URL: https://github.com/androguard/androguard* (2011).

[18] Apple Developer. 2018. Expected App Behaviors. https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ExpectedAppBehaviors/ExpectedAppBehaviors.html. (2018). (Accessed on 01/31/2018).

[19] Android Developers. 2017. Advertising ID. http://www.androiddocs.com/google/play-services/id.html. (2017). (Accessed on 12/30/2017).

[20] Android Developers. 2017. Requesting Permissions. https://developer.android.com/guide/topics/permissions/requesting.html. (2017). (Accessed on 11/10/2017).

[21] Android Developers. 2017. UUID. https://developer.android.com/reference/java/util/UUID.html. (2017). (Accessed on 12/30/2017).

[22] Android Developers. 2018. Best Practices for Unique Identifiers. https://developer.android.com/training/articles/user-data-ids.html. (2018). (Accessed on 02/14/2018).

[23] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. 2011. PiOS : Detecting privacy leaks in iOS applications. In *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, 6-9 February 2011, San Diego, CA, USA*. San Diego, UNITED STATES. http://www.eurecom.fr/publication/3282

[24] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.

[25] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. 2010. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 281–287.

[26] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*. ACM.

[27] Forcepoint. 2017. Forcepoint | Human-centric Cybersecurity. https://www.forcepoint.com/. (2017). (Accessed on 11/08/2017).

[28] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 1025–1035.
[29] Seungyeop Han, Jaeyeon Jung, and David Wetherall. 2012. A study of third-party tracking by mobile apps in the wild. (2012).
[30] Jason I Hong and James A Landay. 2004. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. ACM, 177–189.
[31] White House. 2012. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *White House, Washington, DC* (2012), 1–62.
[32] Jeff Huang, Oren Etzioni, Luke Zettlemoyer, Kevin Clark, and Christian Lee. 2012. Revminer: An extractive interface for navigating reviews on a smartphone. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 3–12.
[33] Haojian Jin, Tetsuya Sakai, and Koji Yatani. 2014. ReviewCollage: a mobile interface for direct comparison using online reviews. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*. ACM, 349–358.
[34] Jaeyeon Jung, Anmol Sheth, Ben Greenstein, David Wetherall, Gabriel Maganis, and Tadayoshi Kohno. 2008. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 279–288.
[35] Kaggle. 2018. Test Leakage | Wiki. https://www.kaggle.com/wiki/Leakage. (2018). (Accessed on 02/15/2018).
[36] CMU CHIMPS Lab. 2017. Privacygrade: Grading the privacy of smartphone apps. http://privacygrade.org/. (2017).
[37] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. 2015. AntMonitor: A system for monitoring from mobile devices. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*. ACM, 15–20.
[38] Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. 2019. Coconut: An IDE plugin for developing privacy-friendly apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* (2019).
[39] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. SUGILITE: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 6038–6049.
[40] Toby Jia-Jun Li and Oriana Riva. 2018. KITE: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 96–109.
[41] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: a lightweight UI-guided test input generator for Android. In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 23–26.
[42] Chieh-Jan Mike Liang, Nicholas D. Lane, Niels Brouwers, Li Zhang, Börje F. Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, Ranveer Chandra, and Feng Zhao. 2014. Caiipa: Automated Large-scale Mobile App Testing Through Contextual Fuzzing. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom '14)*. ACM, New York, NY, USA, 519–530. DOI: http://dx.doi.org/10.1145/2639108.2639131
[43] Jialiu Lin, Shahriyar Amini, Jason I Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. 2012. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM, 501–510.
[44] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. 2014. Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. (2014), 199–212.
[45] Yabing Liu, Han Hee Song, Ignacio Bermudez, Alan Mislove, Mario Baldi, and Alok Tongaonkar. 2015. Identifying personal information in internet traffic. In *Proceedings of the 2015 ACM on Conference on Online Social Networks*. ACM, 59–70.
[46] Sina Madani. 2013. Development and Evaluation of an Ontology-Based Quality Metrics Extraction System. (2013).
[47] Luka Malisa, Kari Kostiainen, and Srdjan Capkun. 2017. Detecting Mobile Application Spoofing Attacks by Leveraging User Visual Similarity Perception. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY '17)*. ACM, New York, NY, USA, 289–300. DOI: http://dx.doi.org/10.1145/3029806.3029819
[48] Kirsten Martin and Katie Shilton. 2016. Putting mobile application privacy in context: An empirical study of user privacy expectations for mobile devices. *The Information Society* 32, 3 (2016), 200–216.
[49] Mike Masnick. 2018. Court Says Scraping Websites And Creating Fake Profiles Can Be Protected By The First Amendment | Techdirt. https://www.techdirt.com/articles/20180401/22565539541/court-says-scraping-websites-creating-fake-profiles-can-be-protected-first-amendment.shtml. (2018). (Accessed on 10/24/2018).
[50] Hajime Morita, Tetsuya Sakai, and Manabu Okumura. 2012. Query snowball: a co-occurrence-based approach to multi-document summarization for question answering. *Information and Media Technologies* 7, 3 (2012), 1124–1129.
[51] Rob Neuhaus. 2018. rrenaud/Gibberish-Detector: A small program to detect gibberish using a Markov Chain. https://github.com/rrenaud/Gibberish-Detector. (2018). (Accessed on 01/15/2018).
[52] Annalee Newitz. 2005. Dangerous Terms: A User's Guide to EULAs | Electronic Frontier Foundation. https://www.eff.org/wp/dangerous-terms-users-guide-eulas. (2005). (Accessed on 10/24/2018).
[53] Helen Nissenbaum. 2004. Privacy as contextual integrity. *Wash. L. Rev.* 79 (2004), 119.
[54] Helen Nissenbaum. 2009. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press.

[55] Peter Norvig. 2009. Natural language corpus data. (2009).
[56] Jeff Pan. 2017. Reverse Engineering the AirBNB API. https://medium.com/switchcm/reverse-engineering-the-airbnb-api-c0f5cb609b. (2017). (Accessed on 11/04/2017).
[57] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications.
[58] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. 2014. Autocog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1354–1365.
[59] Vaibhav Rastogi, Yan Chen, and William Enck. 2013. AppsPlayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 209–220.
[60] 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).
[61] 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*. ACM, 361–374.
[62] Irwin Reyes, Primal Wiesekera, Abbas Razaghpanah, Joel Reardon, Narseo Vallina-Rodriguez, Serge Egelman, and Christian Kreibich. 2017. " Is Our Children's Apps Learning?" Automatically Detecting COPPA Violations. (2017).
[63] Luke Rolka. 2012. Infographic: How to Solve the Online Registration Challenge | Janrain. https://www.janrain.com/blog/infographic-how-solve-online-registration-challenge. (2012). (Accessed on 10/23/2018).
[64] Objective Development Software. 2017. Little Snitch 4: Makes the invisible visible! https://www.obdev.at/products/littlesnitch/index.html. (2017). (Accessed on 07/28/2017).
[65] Yihang Song and Urs Hengartner. 2015. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 15–26.
[66] Gaurav Srivastava, Saksham Chitkara, Kevin Ku, Swarup Kumar Sahoo, Matt Fredrikson, Jason Hong, and Yuvraj Agarwal. 2017. PrivacyProxy: Leveraging Crowdsourcing and In Situ Traffic Analysis to Detect and Mitigate Information Leakage. *arXiv preprint arXiv:1708.06384* (2017).
[67] Android Studio. 2018. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html. (2018). (Accessed on 02/02/2018).
[68] Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. 2014. The Effect of Developer-specified Explanations for Permission Requests on Smartphone User Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 91–100. DOI: http://dx.doi.org/10.1145/2556288.2557400
[69] template.com. 2018. EULA Template and Generator - Free and for 2018. https://eulatemplate.com/. (2018). (Accessed on 10/24/2018).
[70] Max Van Kleek, Ilaria Liccardi, Reuben Binns, Jun Zhao, Daniel J Weitzner, and Nigel Shadbolt. 2017. Better the devil you know: Exposing the data sharing practices of smartphone apps. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 5208–5220.
[71] Haoyu Wang, Jason Hong, and Yao Guo. 2015. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 1107–1118.
[72] Jiayu Wang and Qigeng Chen. 2014. ASPG: Generating android semantic permissions. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*. IEEE, 591–598.
[73] Fengguo Wei, Sankardas Roy, Xinming Ou, and others. 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*. ACM, 1329–1341.
[74] Ryan Welton. 2018. Fuzion24/JustTrustMe: An xposed module that disables SSL certificate checking for the purposes of auditing an app with cert pinning. https://github.com/Fuzion24/JustTrustMe. (2018). (Accessed on 02/11/2018).
[75] Wikipedia. 2018. Card sorting - Wikipedia. https://en.wikipedia.org/wiki/Card_sorting. (2018). (Accessed on 08/15/2018).
[76] Wireshark. 2017. Wireshark · Go Deep. https://www.wireshark.org/. (2017). (Accessed on 08/28/2017).
[77] Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. 2015. Who knows what about me? A survey of behind the scenes personal data sharing to third parties by mobile apps. *Technology Science* 30 (2015).
[78] Hao Zhang, William Banick, Danfeng Yao, and Naren Ramakrishnan. 2012. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*. IEEE, 104–112.

Received February 2018; revised May 2018; revised August 2018; accepted October 2018