# Are You Spying on Me?
# Large-Scale Analysis on IoT Data Exposure through Companion Apps

Yuhong Nan[1*], Xueqiang Wang[2*], Luyi Xing[3], Xiaojing Liao[3],
Ruoyu Wu[4], Jianliang Wu[4], Yifan Zhang[3], and XiaoFeng Wang[3]

[1]Sun Yat-sen University, *nanyh@mail.sysu.edu.cn*
[2]University of Central Florida, *xueqiang.wang.01@gmail.com*
[3]Indiana University Bloomington, *{luyixing, xliao, yz113, xw7}@indiana.edu*
[4]Purdue University, *{wu1377, wu1220}@purdue.edu*

## Abstract

Recent research has highlighted privacy as a primary concern for IoT device users. However, due to the challenges in conducting a large-scale study to analyze thousands of devices, there has been less study on how pervasive unauthorized data exposure has actually become on today's IoT devices and the privacy implications of such exposure. To fill this gap, we leverage the observation that most IoT devices on the market today use their companion mobile apps as an intermediary to process, label and transmit the data they collect. As a result, the semantic information carried by these apps can be recovered and analyzed automatically to track the collection and sharing of IoT data.

In this paper, we report the first of such a study, based upon a new framework *IoTProfiler*, which statically analyzes a large number of companion apps to infer and track the data collected by their IoT devices. Our approach utilizes machine learning to detect the code snippet in a companion app that handles IoT data and further recovers the semantics of the data from the snippet to evaluate whether their exposure has been properly communicated to the user. By running *IoTProfiler* on 6,208 companion apps, our research has led to the discovery of 1,973 apps that expose user data without proper disclosure, covering IoT devices from at least 1,559 unique vendors. Our findings include highly sensitive information, such as health status and home address, and the pervasiveness of unauthorized sharing of the data to third parties, including those in different countries. Our findings highlight the urgent need to regulate today's IoT industry to protect user privacy.

## 1 Introduction

The pervasiveness of Internet of Thing (IoT) devices has brought in concerns about their privacy implications when operating in scenarios such as smart homes, health care, etc. A recent study has uncovered unauthorized data collection and exposure in these devices, which could lead to significant privacy breaches [63]. For example, it is reported that vendors of smart sex toys are sued for their "spying vibrators" stealthily gathering private user data such as one's email, device usage patterns (vibration duration, start and stop time, etc.) [45]. With the recent discussion on IoT data transparency supported by privacy policies or data labels [25, 31], little has been done, however, from the side of device vendors to clarify what data are collected and how the data are used and shared. As a result, today's IoT users are still under the persistent risk of uninformed data exposure, which not only endangers their privacy but also weakens public trust in modern IoT ecosystems. Therefore, there is an urgent need to understand the problem to help IoT users better assess privacy risks they are facing and policy makers better regulate the IoT industry.

**Challenges in IoT data exposure analysis**. A large-scale study of IoT data exposure, however, turns out to be hard, being impeded by the difficulty in accessing a large number of IoT devices, which are expensive to get and challenging to inspect. Hence, prior research focuses on a small set of IoT devices deployed in the lab environment [51, 63, 79] and typically utilizes their network traffic to identify the types of data they transfer over the Internet, and the parties receiving the data [13, 63, 66, 67]. This requires collecting the IoT communication traffic, and therefore cannot easily scale. For example, one of the largest studies on IoT privacy just involves 81 devices [63]. To address the scalability problem, IoT Inspector [39] made attempts to crowd-source the traffic collection through an application installed willingly by IoT users in home networks to capture the communication between IoT devices and the Internet. For privacy concerns of the research process, IoT Inspector removes sensitive data and collects only aggregated statistics or traffic/device metadata (e.g., remote IP, device labels). Also in the presence of encrypted IoT traffic, such traffic-based approaches tend to be less effective. In addition, several prior works proposed assessing IoT data leaks by inspecting available cloud-side IoT apps. Examples are IoTWatcH [21] and SAINT [23] that analyzed source code of hundreds of SmartThings SmartApps [70]. These methods, often limited to specific IoT platforms, unfor-

*The two lead authors contributed equally to this work.

tunately, cannot be applied to the vast majority of devices in the wild, whose back-end services are essentially black boxes that are difficult to analyze.

To the best of our knowledge, prior techniques could not fully enable a large-scale, fine-grained discovery and analysis of the information gathered and disseminated by IoT devices and IoT vendors (e.g., analyzing thousands of unique IoT devices and vendors with their actual data content).

**Our work**. To address the imperative problem, we present a novel approach that achieves a large-scale, fine-grained inspection of IoT data exposure by analyzing IoT mobile companion apps (mobile apps), which complements prior studies relying on access to real devices, traffic, and server-side components and contributes to gaining a more complete view of the threats of IoT data exposure in the wild. Our solution is an automatic, generic framework, called *IoTProfiler*, that inspects IoT *data exposure* utilizing mobile *companion apps* of those devices, and further checks whether such exposure has been done in a responsible way. *IoTProfiler* leverages two key observations. **O1:** First, although IoT devices can directly transmit user data to their cloud back-ends, many of them (more than 70% of devices [60, 77]) need to locally connect to their mobile companion apps (e.g., through Bluetooth, local area networks) for the purposes of data processing (parsing, organizing, labeling, etc.) and preliminary analysis (e.g., associating unique identifiers with the user) before forwarding them to the cloud. **O2:** Second, these companion apps are usually semantic-rich, carrying a variety of textual descriptions in natural languages (e.g., string constants), which indicate not only the presence of IoT data but also their semantics (e.g., heart rates and blood pressure detected by the IoT devices). To this end, we show that it is feasible to *statically* analyze IoT companion apps to automatically assess IoT data collected by IoT devices/vendors and further evaluate the potential exposure *on a large scale*.

Also importantly, sensitive IoT data are highly diverse thanks to a large variety of IoT devices related to healthcare, automotive, security cameras, and home automation and entertainment, etc. Unlike prior studies [21, 39, 46, 51, 63] that can only focus on a limited set of data (e.g., persistent device identifiers and device status), *IoTProfiler* is capable of analyzing a wide spectrum of IoT data by leveraging the known most comprehensive, fine-grained IoT data taxonomy we constructed, which contains 550 privacy-sensitive IoT data generalized from IoT-industry standards/protocols, research papers, and news reports (see Section 3). Using the taxonomy, our approach recovers diverse semantic-rich IoT data from mobile companion apps (e.g., sensitive health data such as lung capacity, heart rates, smoke habits, see Section 6).

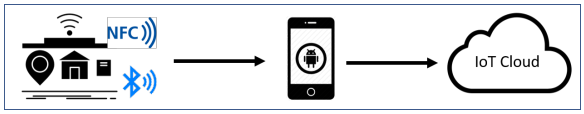
Such data-exposure analysis is nontrivial, due to the challenges in (1) automatically locating the IoT data handled in companion apps, which are often kept together with the apps' local data, and (2) understanding the semantics of the data so as to determine the legitimacy of their collection and sharing. Our solution leverages the observation that IoT data are usually handled by certain dedicated code snippets, and the code snippets contain semantics about the data (e.g., data labels in the form of constant strings). More specifically, *IoTProfiler* first locates IoT-related code snippets by running a text classification model based on fastText [44], and then identifies sensitive IoT data from the discovered code snippets using our IoT data taxonomy. Further, we perform a data-flow analysis to determine what data are exposed to the Internet, and whether they are properly disclosed to IoT users.

**Evaluation and discoveries**. We implemented *IoTProfiler* and evaluated its effectiveness, generality, and performance. This evaluation shows that our approach effectively detects sensitive IoT data with a 93.8% precision and 83.5% recall on our validation dataset. Running *IoTProfiler* on 6,208 IoT companion apps (on Android) from both official and third-party app markets, we were able to conduct a large-scale IoT data exposure analysis, which sheds light on the *pervasiveness of the privacy risks* today's IoT devices pose to general users. These apps cover at least 4,493 unique device vendors, allowing us to gain a more complete view of IoT data exposure on today's market. More specifically, 1,973 companion apps (31.8%), corresponding to at least 1,559 unique IoT vendors, expose diverse user data from IoT devices to the Internet, with each app disclosing 5.6 data items on average (e.g., accurate home address, weight, BMI, apnea count, breath rate, blood sugar level, etc.). However, we found no evidence that the collection of these data has been properly disclosed, e.g., through a privacy policy provided by device vendors. For example, a cigarette holder [17] supposed to keep track of its user's smoking frequency is found to stealthily measure the user's lung capacity.

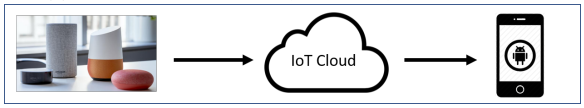
Also, our research shows that IoT vendors disseminate user data broadly without proper disclosure, not just to their own back-ends, but also to a large number of third parties: 557 (9.0%) companion apps (from 465 IoT vendors) transfer IoT data to more than 695 unauthorized parties (Section 6.1). These parties include not only traditional data brokers but also upstream IoT vendors and health-related platforms. For example, we have initial evidence that health analytics platforms (e.g., *healthlink.cn*) connected to health insurance companies may have gathered users' health-related data, which can have serious privacy impacts. Further, our study shows that many devices are involved in cross-region data sharing, which could violate privacy regulations like GDPR [43]. For example, smanos [8], a smart alarm system whose vendor is in Amsterdam, periodically sends data to servers in China.

**Contributions**. We summarize our contributions as follows:

• *New techniques*. We developed a suite of new techniques for the large-scale, fine-grained, semantic-aware analysis of IoT data exposure without relying on the access to real IoT devices, their traffic, or IoT back-ends which are typically not available for third-party assessment.




(a) Device-to-mobile data management (using NFC, Bluetooth, BLE, etc.)




(b) Cloud-based data management (Google Home, Amazon Echo, etc.)

Figure 1: Two typical data management modes in IoT.

• *Large-scale study of IoT data exposure*. We reported a large scale study of exposed IoT data on 6,208 IoT companion apps from 4,493 IoT vendors. Our new findings help to understand the gravity of the IoT data exposure risk in the wild and highlight the importance of regulating the data practices of IoT industry to protect user privacy.

• *New taxonomy of sensitive IoT data*. *IoTProfiler* is built on top of our new taxonomy of sensitive IoT data, which covers more IoT data types with finer granularity than those studied in prior research (e.g., [39, 63]). Such a taxonomy contributes to future IoT research by generalizing the level of privacy and also better informs policy/law makers and IoT vendors of the data that need to be protected.

## 2 Background

**Two modes of IoT data management.** IoT devices on today's market typically share the data they collect with their back-ends or third parties in two ways. First, IoT vendors often utilize companion apps as an intermediary to process the data from their devices, before sending them to the clouds (Figure 1(a)). This data management mode is particularly popular among the devices with short-range communication (e.g., BLE, NFC) only, which tend to leverage the computing power of mobile phones to manage and transmit data. Note that even Internet-connected devices often support a peer-to-peer connection (or a relayed connection) with their companion apps, for transmission of IoT data to their clouds through the apps. Examples include P2P cameras and devices featured with local controls (e.g., Philips Hue lights, August locks, Belkin Wemo, and TP-Link). A recent study [77] shows that even among the IoT devices with Internet access, 76.3% of them let data collected at the devices to go through companion apps.

In the meantime, some IoT devices directly transfer their data to their cloud back-ends. Examples include Amazon Echo, Google Home, Samsung SmartThings (Figure 1(b)). Although this communication path does not go through companion apps, such devices tend to forward some data to the apps for the purposes of device management. For example, an air conditioner may display room temperature measured by its sensor in its companion app. Interestingly, such information could be shared with third parties (see Section 6.1), which once happens often goes through the companion apps, due to the convenience in integrating the SDKs from those parties into the apps. We found in our research that analysis of such integration can also help understand IoT data exposure.

**Scope of our study.** In this research, we focus on the IoT devices that share user data they collect with cloud back-ends or third parties *through their companion apps*. We do not consider the devices uploading *all* their data through direct connections with the cloud back-ends, without transmitting any data to mobile companion apps. On the other hand, we acknowledge that we may miss information exposures through the direct device-cloud channels and data sharing on the cloud side: for example, unauthorized transfer of user data by a device's cloud to another party. Notably, such data exposure is difficult to detect and measure from the user side due to the invisibility of an IoT cloud back-end's internal operations.

**Threat model.** Our study primarily focuses on legitimate IoT vendors and companion apps whose data collection and sharing practices can lead to unexpected privacy exposure and, thus, legal non-compliance. Although real-world IoT companion apps may involve code obfuscation (e.g., variable-renaming), in practice, common obfuscation techniques/tools, such as ProGurad [55], have little impacts on our approach *IoTProfiler*. This is because *IoTProfiler* only leverages constant strings in the code to analyze semantics, whose obfuscation is complicated and costly [33], easily introducing prominent performance overhead [83], bugs and functionality-disruptions [47]. Thus, constant string obfuscation is rarely adopted by legitimate IoT vendors in the wild based on our study and prior work [29]. We do not consider malicious and evasive operations, such as customized logic/-time bomb and code/data encryption, that are meant to evade analysis, which pose fundamental challenges for an analysis at our scale.

## 3 A Taxonomy of Privacy-Sensitive IoT Data

Given the diversity of IoT devices, our work proposes an IoT privacy-data taxonomy that covers different categories of IoT data, with a variety of sensitive data items being used in the wild. Such a taxonomy is essential towards a fine-grained IoT-data exposure analysis (see Section 6). It is constructed by gathering IoT data items that are considered privacy-sensitive in prior research papers and media reports, and further enriched by IoT industry standards/protocols that define IoT data models. As a distinction from user data concerned about in prior mobile privacy research [16, 53, 54, 68, 76, 81, 84], our taxonomy describes IoT data that are obtained from device sensors (e.g., blood pressure), IoT device operation/usage (turning on/off a blood pressure monitor), and IoT device metadata.

**Taxonomy construction.** Our taxonomy (Table 1) is built

Table 1: Summary of the taxonomy with examples of privacy-sensitive IoT data items

| Category | Subcategory | Identified data items from news reports and research papers |
|---|---|---|
| Device Tracking Data | Device Identifier | device id, device fingerprint id, hardware id, identity id |
| | Network Identifier | IP address, mac address, bssid, ssid |
| Sensor Data | Biometric Data | blood oxygen level, user voice, body weight, blood pressure, stroke volume, facial image, hand size, palm humidity, kids voice, walking length, medical search history |
| | Location Data | vehicle location, lock location, camera location, drive route, drone address |
| | Environmental Data | device/home temperature, street lights, air quality, road surface quality |
| Device-attached Data | Device Metadata | device name, device model, screen resolution, device DPI, paired device, temperature setting, intensive setting, vibration mode, bluetooth info |
| | Device Usage | camera image, camera video, device usage status, driving speed, watched/listened channels, watched videos, voice clip, favorite show, light on/light off, media information, audio recording, smoke puffs, maps of room, door status |
| | Timing Info | smoke time, device usage frequency, watch time, bed time, driving time, turn on time, at home time, having sex time, light on time, travel time |

* We provide the references (research papers and news reports) of each category and data item in our project repository [2].

by inspecting 29 research papers and 54 news reports that cover disparate privacy threats associated with IoT sensitive data, ranging from device identifiers to IoT sensor data and usage data attached to the IoT devices, etc. To collect such data, we manually inspected research papers published in the past five years from major security and privacy related venues (see the full list online [2]), and identified those focusing on IoT privacy research and their related works. We also used keywords, such as "IoT privacy", "smart home privacy", "IoT data exposure" and "smart home data exposure", to query Google Scholar and Google News to gather relevant papers and news reports. After that, we asked three domain experts to review those papers and news reports, annotate privacy-sensitive IoT data items that users are indeed concerned about, and determine the categories of those data items. Overall, the task took them around 60 hours (20 hours for each expert) to annotate data, with a Cohen's Kappa coefficient [26] (which measures both inter- and intra-rater reliability) of 0.92.

**Taxonomy enrichment.** We leverage data models defined in IoT industry standards and development documentation to enrich the above taxonomy with the best possible coverage. Our corpus includes the IoT interoperability standards such as *Matter* [12] and *HomeKit Accessory Protocol* [42], and developer documentation crawled from mainstream IoT platforms [3, 4, 7, 69], including *Tuya*, *Philips Hue*, *MiHome*, and *openHAB*. Notably, this corpus, especially those IoT industry standards, has defined the functionalities and involved data of a number of mainstream smart home devices. To help analyze the developer documents, we adopted a security-domain name entity recognition (NER) model reported in XFinder [75] to identify potential IoT data items (i.e., entities) from the documents. In this step, we extracted 3,510 data items in forms of short terms and keywords from 3,102 document sentences. After that, we manually reviewed those data items and merged them into corresponding categories in the taxonomy. In this way, we extended the taxonomy to include 550 sensitive IoT data items, with each subcategory covering an average of 68 sensitive IoT data items. Note that manual examination is necessary to maintain the high-quality of our taxonomy. While the taxonomy is still not comprehensive, it covers much more sensitive IoT data with finer granularity than prior works [13, 21, 39, 46, 51, 63]. For example, IoT Inspector [39] mainly focuses on device metadata (e.g., device name, device model, mac address), a sub-category in our taxonomy, and traffic statistics (e.g., bytes sent/received). Ren et. al [63] discussed device identifiers and a few device activities (video, power, movement) inferred from network traffic, while our taxonomy covers much more fine-grained data items such as driving speed, door status, etc.

**Discussion.** According to the GDPR Article 4 - Definition [6], at least the IoT data items under the *Device Tracking Data* and *Biometric Data* (sub)categories of our taxonomy (see Table 1) are considered to be personal data. Particularly, items under *Device Tracking Data* are "general personal data" as they can be used to uniquely identify a natural person; items under *Biometric Data* are considered as "special personal data" as they represent the physical, physiological or behavioural characteristics of a natural person. Even though the other data items might not have been explicitly mentioned by GDPR or be used to directly identify a person, they may be used by vendors to better profile a user once linked with a user ID (e.g., for understand the user's preference and promote targeted advertising) and thus considered privacy-sensitive in our research and prior papers and news articles.

## 4 Design of *IoTProfiler*

**Motivation, challenges and idea.** *IoTProfiler* aims for a large-scale understanding of IoT data exposure in the wild, through automated static analysis of IoT companion apps. A key challenge, as mentioned earlier, is how to effectively locate the IoT device data "in-transit" within the app, which is a necessary step before one can track their exposure. A naive idea is to rely on a pre-defined list of keywords (e.g., "device id", "username" and "password"), which has been adopted in prior works to find sensitive data from general

mobiles apps or IoT-device traffic [40, 51, 52, 53]. However, prior lists of keywords — usually small — do not suffice, not only for the high diversity of IoT data (e.g., heterogeneous health data), but also thanks to their contextual dependency. For example, "start_time" in a companion app could indicate either the information received from an IoT device (e.g., start time when a person sleeps) or start time of the app – a non-IoT data.

```
1  public class StdiDeviceStatus extends BluetoothEvent {
2      private int deviceId;
3      private boolean a;
4      private int b, c, d;
5      private string pkgName;
6
7      public void getDeviceStatus(byte[] bleData){
8          this.a = (bleData[0] == 1);
9          this.b = getVoltage(bleData[4]);
10         this.c = bleData[5];
11         this.d = bleData[6] & 15;
12     }
13
14     public String updateDevStatus() {
15         String devStatus =
16         "DeviceInfo␣[deviceId␣" + this.deviceId +
17         "␣,isRunning=" + this.a +
18         ",␣vibrationMode=" + this.b +
19         ",␣batteryLevel=" + this.c +
20         ",␣temperature=" + this.d +
21         ",␣eventTime=" + Utils.getCurrentTime() +
22         ",␣packageName=" + this.pkgName +
23         "]";
24         HTTPRequest.send(devStatus);
25     }
26 }
```

Figure 2: A code snippet from We-Vibe companion app.

Our idea to address this challenge is based upon the observation that in a typical IoT companion app, there are certain methods that handle the data transmission from the IoT device and parse the data before they can be used (e.g., displaying to the user or transmitting to the cloud). Figure 2 shows a real example — a code snippet from the companion app of We-Vibe [45] (a sex toy). Here, the app first gets the raw data from the BLE-connected device in a byte-array `bleData` (Line 7). Then, it parses each data item and saves them to a set of class fields `a, b, c`, and `d` (Line 8-11). Further, when the companion app transmits the data to the server (Line 14-25), it first aggregates the data items and labels each of them with a set of textual descriptions, for example, "`vibrationMode`" (Line 18, vibration mode of the toy), and "`temperature`" (Line 20, temperature of the toy). Such structured and labeled data will be easy for the server to use.

Such an observation allows us to identify the classes or methods that handle IoT data received from the IoT device, by leveraging the "collective semantics" of all descriptive texts in a method, in particular constant strings, to determine semantics of the data and their correlation with IoT. More specifically, in this motivating example, the textual descriptions in the code can adequately indicate semantics of the data, although the names of class fields often can be

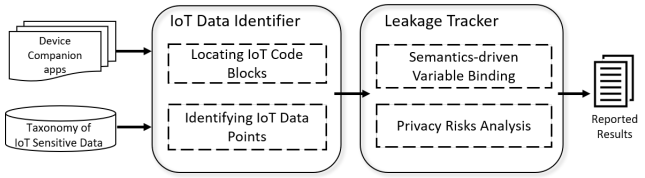



Figure 3: Overview of *IoTProfiler*

obfuscated, e.g., `a`, `b`, `c`, and `d` in Line 8-11 (see discussion of obfuscation in Section 7). Also interestingly, by looking at a single text label (e.g., `isRunning`), it is difficult to tell with high confidence whether the data in the class are related to the IoT device. However, when combining all text labels (e.g., `batteryLevel`, `vibrationMode`, and `temperature`), we get a strong indicator that the method (`updateDevStatus`) and the corresponding variables (`a`, `b`, `c`, and `d`) are handling IoT data.

To understand the generality of the observation ("IoT data are often clustered for processing"), we randomly sampled 500 IoT companion apps (out of a total of 6,208 IoT companion apps found in the wild, see Section 6) and confirmed that each app included at least one method that processes a cluster of IoT data indicated by their textual descriptions (see Section 4.2). In our research, we designed a semantic-based learning model to discover such methods from the apps, which are further analyzed to pinpoint the actual IoT data items and determine whether the data practices incur privacy risks.

**Terminologies.** Following are the terminologies used throughout the paper:

• **IoT code block**. For an IoT companion app, we consider an *IoT code block* as a method which includes *a cluster of* semantic-meaningful text labels describing information related to the IoT device, for example, the method `updateDevStatus()` in Figure 2.

• **IoT data point**. Given an IoT code block, we define each of the text labels (i.e., constant strings) it includes as an IoT data point if the text label clearly indicates information related to the IoT device. In Figure 2, for example, the IoT data points in the IoT code block `updateDevStatus()` include text labels `isRunning`, `batteryLevel`, etc. Note that not all text labels in an IoT code block are necessarily referring to IoT data. For example, the label "`packageName`" (Line 24) only represents the app's package name.

## 4.1 Approach Overview

As outlined in Figure 3, *IoTProfiler* takes IoT companion apps (on Android) as input and reports whether an app incurs privacy risks by exposing IoT data to the Internet without disclosure. *IoTProfiler* uses two major components:

The first component is an *IoT Data Identifier* (IDI). *IDI* first obtains decompiled app code using Soot [74], and extracts *code blocks* (i.e., methods containing text labels) from

the decompiled app code. Then, *IDI* determines which data blocks are semantically related to IoT data through a text classification model (Section 4.2). Afterwards, all text labels in an identified IoT code block are analyzed, based on their semantic similarities to our IoT data taxonomy, to find out those that indeed describe the data of an IoT device (Section 4.3). Taking the code block Line 16-22 in Figure 2 as an example, *IDI* will report that all text labels except the one in Line 22 (i.e., `packageName`) are IoT data points.

The second component is a *Leakage Tracker* (LT). With the identified IoT data points, *LT* checks if the corresponding IoT data points are exposed to the Internet through a data flow analysis. Here a key challenge is how to effectively find all program variables related to a particular IoT data point. This is beneficial since multiple variables across nearby IoT code blocks may represent the same IoT data point, and finding these variables would improve the coverage of the data flow analysis. To overcome the challenge, *LT* leverages a semantic-driven approach to relate IoT data points to potentially multiple program variables that can be found in the code (Section 4.4). *LT* then checks if any of those variables is exposed through a privacy risk analysis (Section 4.5), i.e., determining whether data collection and sharing are disclosed in IoT-related privacy policies.

## 4.2 Locating IoT Code Blocks

As mentioned earlier, the first step to identify IoT data points is locating the IoT code blocks from the decompiled app code. To this end, *IDI* extracts text labels for all code blocks, and uses a binary text classifier to tell whether a code block is IoT related.

**Collecting (non-)IoT code blocks for training.** We randomly sampled 500 IoT companion apps from our app set (see Section 6), and extracted a set of candidate IoT code blocks based on IoT related keywords such as "bluetooth", "battery". After manual inspection of these candidates, we finally labeled 962 IoT code blocks for model training. In the meantime, we labeled an equivalent number (962) of non-IoT code blocks from these apps, which are further used as negative samples for model training. The non-IoT code blocks only perform app functions irrelevant to IoT devices, for example, loading and parsing the user profile (as a JSON object) from the remote server. The text labels in these two sets are used as a balanced training dataset to build the IoT code block classifier.

**Text classification.** For each code block in the app, *IDI* extracts constant strings appearing in the code block as semantic text labels. These text labels will first go through the standard text pre-processing (e.g., word splitting and non-character symbol removal) for classification.

In our research, *IDI* employs fastText [44], a state-of-the-art word embedding model based on neural network to learn and identify IoT code blocks through the text label semantics. More specifically, we first send the code blocks extracted from the aforementioned 500 IoT apps to a domain-specific word-embedding model based upon fastText, following the standard procedure in its tutorial [9]. Then, with this embedding model, *IoTProfiler* trains an SVM classifier and performs supervised learning over the labelled corpus (i.e., the 1,924 IoT and non-IoT code blocks) for IoT code block identification. Particularly, given a future unknown code block, text labels (words) it contains are transformed into numerical embeddings by fastText, which are further fed to the SVM classifier to determine whether it is an IoT/non-IoT code block.

Unlike other embedding models such as word2vec [50] which use the word-level n-gram tokenization, fastText uses the character-level n-gram tokenization instead to generate text embeddings. In this way, it can provide better representations for misspelled words, rare words and words that do not exist in the training corpus [22]. This is particularly helpful in our task, as the namings of same subjects in program code may change from app to app by different developers. For example, the two strings "`body_temperature`" and "`body_temp`" in different apps have a similar meaning, but the latter might even not appear in any vocabulary set. Our further evaluation by comparing different word embedding models (word2vec and BERT [28]) showed that fastText indeed performs better (see Section 5.2).

**Enriching semantics of code blocks.** Although most code blocks contain a sufficient amount of text labels reflecting their semantics, in some cases, the number of valid text labels in the code block could be limited. This often happens when a code block does not have constant strings, but only includes heavily obfuscated class member variables (e.g., a variable named `aa`). As a result, the coverage of *IoTProfiler* could be reduced due to the lack of text labels in such code blocks. To address the problem, we come up with a set of heuristics to enrich the semantics by correlating scattered text labels to these code blocks. Taking Figure 4 as an example, the member variable, `this.a`, is included in a code block, but is obfuscated and do not convey any meaning. To enrich the semantics, we analyze all methods in the class and find the text labels that relate to the member variable. In this case, the text label "`day_step:`" is correlated to the code block since it describes the `this.a` field (Line 4). Through this process, the code blocks with obfuscated fields can be better correlated with semantic descriptions.

```
1 package com.huawei.health.d;
2 public class f {
3     public boolean b() {
4         Log.e("Step_Report", "day step:" + this.a);
5     }
6 }
```

Figure 4: A code block in `com.huawei.health` app.

## 4.3 Identifying IoT Data Points

The identified IoT code blocks may include not only text labels for IoT data points, but also those describing an app's internal states (e.g., "`error_msg`", "`response`"). To find those indeed related to IoT devices, we designed a taxonomy guided approach in which *IDI* compares the similarity between a given text label discovered from an IoT code block and each data item in our taxonomy (Section 3) as follows.

**Semantic similarity.** We use a state-of-the-art and best available sentence embedding model, Sentence-BERT [62], to measure the semantic textual similarity. The model takes sentences (short terms in our case) as input and outputs its vector representations so the sentences with similar meanings are close in the vector space. Specifically, we select a pre-trained sentence embedding model with the highest score on STS-benchmark (i.e., stsb-roberta-large) in our task. This model was trained on large open-domain corpora (i.e., the Stanford Natural Language Inference (SNLI) corpus) and optimized for computing semantic textual similarity.

To determine whether a data point indeed relates to a taxonomy data item, we need a similarity threshold. Such a threshold is found using a ground truth dataset extracted from the 962 training IoT code blocks in Section 4.2. From these IoT code blocks, we collected 9,977 text labels, and 1,861 of them are unique IoT labels. In our research, we searched between 0 and 1 using a step 0.01 on the dataset for the cut-off that determines the relation between a text label and data items, and further evaluated the precision and recall under each cut-off. Our experiments show that a cut-off of 0.81 yields an optimal result, with a precision of 93.6% and recall of 81.5% (more details in Figure 9), which was therefore chosen as the similarity threshold. Note that with this threshold, a given data point could match multiple taxonomy entries. In this case, we use the entry with the highest similar score.

**Handling local app data.** Some data items in our taxonomy describe both sensitive data related to IoT devices and those solely used by mobile apps (called *local app data*). A prominent example is location-related data items, such as `latitude` and `longitude`. Therefore, looking up the taxonomy for the text labels in a companion app may introduce false positives, i.e., reporting the local data as IoT data points.

*IoTProfiler* takes a few measures to eliminate such false positives. First, when searching for code blocks from an IoT companion app, our approach focuses on the app developer's code, which is supposed to be mostly related to device management, as identified from the app's package names. This helps remove a significant amount of data points in third-party libraries that are irrelevant to IoT devices, such as location data collected by mobile advertising/analytics libraries. Second, *IoTProfiler* leverages the fastText model to detect IoT code blocks with clustered semantics. This ensures that a code block will not be labeled IoT-relevant just due to the presence of an individual data point of interest, so the data like `latitude` need to coexist with other identified data points (according to our taxonomy) to be considered as IoT data. Notably, based on a thorough evaluation (Section 5.2) on 2,659 IoT data points labeled by *IoTProfiler*, 2,495 of them (93.8%) are indeed IoT data (from IoT-devices or their usage statistics) and local app data, including mobile sensor data returned by Android APIs, are typically not included in the results.

## 4.4 Finding IoT-Related Program Variables

With the identified IoT data points, the *Leakage Tracker (LT)* then finds out the corresponding variables in the code that store the values of the data points. Section 5.3 will show that these variables are used as taint sources to check whether the IoT data are exposed to the Internet.

**Identifying the variables inside IoT code blocks.** In general cases, an IoT data point comes from a textual string (i.e., constant string, see Figure 2) that describes a variable within the app, e.g., the key value in the form of a constant string describing the value variable in a key-value store such as Map and JSON object. To find the variable that stores the value of such an IoT data point, we inspect the data operation statements involving the IoT data point. For example, in the statement that operates a JSON object `jsonObject.put("walk_distance", a)`, "`walk_distance`" is the IoT data point we identified and `a` is the variable we found that stores its value.

**Identifying the variables across the app.** Outside IoT code blocks, a particular app could have multiple copies or references of an IoT data across the app, for which we also aim to track exposure. Compared to an IoT code block which may parse/process a cluster of IoT data and put them in a dataset, another method/class in the app may reference and use just one (or few) of the IoT data (e.g., `jsonObject.get("walk_distance")`). Note that *IoTProfiler* identifies an IoT block based on its clustered IoT semantics (see Section 4.2), so an individual reference of an IoT data in a method may not be identified and thus tracked for privacy exposure. Even more challenging is, those individual references of the IoT data may use text labels (e.g., variable names) different from what we identified in the IoT code blocks, making it ineffective to simply search the IoT data point across the app. For example, for the same IoT data, the label "battery_level" is used to present the data to the user interface, while the label "voltage_value" is used for device monitoring.

To address the problems, we developed a simple yet effective approach. In particular, given an IoT data point we have identified from IoT code blocks, *IoTProfiler* searches and maintains a set of its alias labels used in the particular app, based on which we search those variables related to the aliases. For instance, given an identified IoT data point "`battery_level`" with its associated variable `dev.battery_level`,

the statement `intent.putExtra("voltage_val", dev.battery_level)` allows us to link "`battery_level`" to its alias "`voltage_val`", which could then be used to further relate to more variables (based on data operation statements, same as how IoT data points are related to variables discussed above) across the particular app. Note that the aliases are only used within the same app. Therefore, this approach is not affected by choices of different developers or apps.

### 4.5 Detecting IoT Data Exposure

Leakage Tracker relies on static taint analysis to identify IoT data exposures. Specifically, we use the IoT variables identified in Section 4.4 as taint sources and use networking APIs as sinks. Then we leverage FlowDroid [20] to report source-sink connections that expose IoT data points to the Internet. For any exposed data point, we check if its exposure is clearly stated in the device vendors' privacy policies — the current *de facto* and *de jure* standard for the service provider to communicate to users its applicable privacy practices [15, 16, 48, 85].

To assess whether device vendors collect IoT data without disclosure, we collect their privacy policies from multiple sources, i.e., on the app store pages, on the device vendor websites, and in the app resources (under file path *res/* and *assets/*). In total, we gathered 4,182 valid privacy policy pages for 3,686 companion apps from the above sources in August 2020 (each app and its privacy policies were collected at the same time). We then preprocess the pages with an HTML parser [14] to convert them to plaintext format. After that, we detect the language of privacy policy paragraphs with langdetect [27]. Those non-English paragraphs are then translated into English with Google Translate [35] before being fed into PolicyLint. According to previous studies, Google Translate achieves an overall accuracy of 82.5% for different languages including Chinese, Korean, etc [71]. In our research, we further verified that the translation was accurate by manually sampling and checking 200 translated policy paragraphs. Last, we adopt PolicyLint [15] (a state-of-the-art tool for privacy policy analysis) to extract data collecting and sharing related statements. These statements are modeled as `(actor, action, data point, entity)` tuples, which represent the data practices of "We [`actor`] collect [`action`] your email address [`data point`] with advertisers [`entity`]."

We compare the output of FlowDroid (data exposure from the app) with the tuples collected from privacy policies (i.e., disclosed data collection), and report a privacy risk if any of the following conditions is met: 1) an exposed data point does not show up in its device's privacy policy, 2) the privacy policy explicitly describes that an exposed data point will not be collected, or 3) an exposed data point is shared to a third party by an app while the privacy policies indicate that the data point (or any data) will not be shared. A key challenge here is that automatically checking their flow-to-policy consistency is non-trivial, due to the inconsistent descriptions between the IoT data points found in the apps and those mentioned in privacy policies. Specifically, a substantial amount of IoT data points found in the apps are related to low-level device operations unique to the IoT context (e.g., `door_opened`), while such data could be described with more general terms in privacy policies (e.g., "we may collect certain `device usage data` from your device" [11]). To address the challenge, we first map each exposed data $d$ in the app to a data item $d\_t$ in our taxonomy (see the semantic-based approach in Section 4.3). If the collection of either $d\_t$ or its hypernym (i.e., the category/subcategory name recorded in the taxonomy, see Table 1) is disclosed in the privacy policies, we consider it as a proper, accountable disclosure. Otherwise, *IoTProfiler* reports a privacy risk with $d$.

**Discussion.** As required by privacy regulations [32] and public platforms (e.g., Google Play [1] and Apple App Store [41]), privacy policies are usually posted publicly on the Web, and used as a standard method to disclose the collection/usage of private user data. In addition to that, the apps and device vendors we studied may use a different disclosure method, i.e., in-app disclosure. However, this method is just a complementary way to seek user consent based upon online privacy policies, rather than a substitute for privacy policies [1]. Therefore, we consider using privacy policies as a comprehensive source for validating proper disclosure of IoT data.

## 5 Evaluation

In this section, we report our evaluation on *IoTProfiler*, including the effectiveness of its end-to-end execution and individual components.

### 5.1 Experiment Datasets

• **IoT companion apps collected in the wild (denoted as $D_{cp}$).** In our research, we collected a set of IoT mobile companion apps from public app markets. Specifically, we crawled both Google Play [34], third-party websites (i.e., APKPure [18] and 360 Store [10]) to download their most up-to-date apps in August 2020. From each source, we searched with a list of IoT-related keywords, such as smart home, thermostat, and smart plug, and crawled the corresponding apps from their websites. We used app package name (e.g., *com.xiaomi.smarthome*) as the unique identifier and removed redundant ones to make sure that the dataset does not include overlapped apps from different sources. To further ensure that the collected apps are indeed relevant to IoT devices, we asked two researchers to manually inspect their names, descriptions, and drop the app if anyone does not think that it is IoT-related.

In total, 6,208 IoT companion apps were collected from a total of 9,995 candidates (with 3,331 non-IoT apps, and 456 packed apps [30]). Among the IoT companion apps, 74.6% of them are from Google Play, 11.4% of them are from APKPure and 14.0% of them are from the 360 Store. While our crawler targeted the apps available in U.S. and China, we noticed that the collected IoT companion apps are actually available in most other regions, with each app covering an average of 63 locales (or regions). The average app size is 15.7MB (ranging from 16KB to 133MB). By checking the developer name and supporting websites of the apps, we found that our dataset covers at least 4,493 unique IoT vendors. These apps are associated with various types of IoT devices such as smart lights and controls, home security systems, and health-related devices.

• **Ground-truth dataset (denoted as $D_{gt}$).** We randomly selected 60 apps from $D_{cp}$ as a ground truth set: we had two researchers manually identify the IoT code blocks and data points within these apps independently. The Cohen's Kappa [26] value for the labeled IoT code blocks and data points are 0.95 and 0.87, respectively. The two researchers resolved their disagreement by onsite discussion, which leads to the identification of 852 IoT code blocks and 10,409 data points (2,987 IoT data points and 7,422 non-IoT data points) from these apps.

## 5.2 Effectiveness of IoT Data Identifier (*IDI*)

**Effectiveness of IoT code block identification.** *IDI* relies on a fastText embedding model and an SVM classifier to identify IoT code blocks. Overall, *IDI* achieves 94.5% precision and 94.5% recall over the training dataset. Using our selected classifier, *IDI* effectively identified 775 IoT code blocks (91.0% recall), with only 41 false positives (95.0% precision), on the $D_{gt}$ dataset. More specifically, we first trained a fastText model to generate embeddings for the words in our labelled code blocks[1]. Then we trained an SVM classifier using the manually labelled code blocks – the 1,924 equal number of IoT and non-IoT code blocks (elaborated in Section 4.2). To get the optimum classification results, we carefully tuned the parameters of the two models using grid search. Particularly, the fastText model was trained in "skipgram" mode, with a learning rate of 0.5 and epochs = 5. The vocabulary size is 12,059 and the number of dimensions is 100. The SVM model is with an rbf kernel, and gamma = 0.01.

In addition to fastText, we also tried two other embedding models, i.e., word2vec and BERT, both of which are widely adopted word embedding models. We trained the models on the same corpus as the one used for the fastText model, and fine-tuned their corresponding SVM classifiers to get the optimum results in each setting. Lastly, we followed the 5-fold cross-validation procedure over the labeled dataset set to evaluate the classification results for each model. We leave more details about such embedding models in Appendix A.

[1]We trained our own embedding model instead of using pre-trained models because they often fail to cover words in programs.

Table 2: Using different classifiers for IoT code block identification.

| Classifier | Precision | Recall | F-1 Score | Training Time |
|---|---|---|---|---|
| word2vec + SVM | 91.8% | 87% | 89.3% | 7s |
| BERT + SVM | 94.1% | 82.2% | 87.7% | 230s |
| fastText + SVM | 94.5% | 94.5% | 94.5% | 3s |

Table 2 shows the average results for classifiers with different embedding models. As can be seen, the SVM classifier with fastText embeddings performs better than other alternatives in terms of the overall F1-score.

**Effectiveness of IoT data point identification.** We compare the ground truth in $D_{gt}$ with the IoT data points identified by *IDI*, and report the number of true positives (TP, 2,495), false positives (FP, 164), true negatives (TN, 7,258) and false negatives (FN, 492). Therefore, *IDI* achieves 93.8% precision and 83.5% recall for apps in $D_{gt}$. Our further inspection shows that almost all false positives are caused by the inaccurate mappings between text labels and the IoT taxonomy. For example, the "`initCalled`" label shows whether an app class has been initialized. However, when it appears in an IoT code block, *IDI* incorrectly identified the label as an IoT data point since it shares a high semantic similarity (87.2%) with "opened at" in the taxonomy. Such false positives can be eliminated by using more specific data items in the taxonomy or introducing a more effective mapping method. We also observe that 68 false positive data points fall in the "location data" subcategory in the taxonomy. But in fact, the data points are from GPS sensor of the mobile device, which are returned by Android APIs, rather than an IoT device (we did not notice any other mobile sensor data in the identified IoT data points). Besides, although we have gathered IoT data points from a large corpus technical reports and IoT documentation, our IoT taxonomy is by no means exhaustive. Thus, *IDI* missed 492 IoT-related data points in $D_{gt}$, e.g., "`drink cycle`" that represents a previous unseen usage data of a smart water bottle.

To better understand the distribution of false positives and false negatives among each app of the $D_{gt}$ dataset, we also report the false positives and false negatives within each app. This evaluation metric answers the important question of how effective can *IDI* identify the IoT data for a given IoT companion app. Specifically, we calculate the precision and recall of each app by comparing the ground truth with the identified IoT data points. The results is shown in Figure 8 of Appendix. Given each app, *IDI* achieves an average precision of 94.5% and a recall of 85.1%.

In addition, we evaluated *IDI* without using alias labels to identify IoT variables. Compared to full *IDI*, this baseline approach reported 46.0% less variables that contain IoT data points, confirming that the apps are indeed using different variables for the same IoT data. This resulted in 54.7% less

data exposure reports since the IoT variables are dispensable inputs to the IoT data exposure analysis.

## 5.3 Overall Effectiveness

**Evaluation of false alarms on $D_{gt}$.** *IoTProfiler* reported that 447 IoT data points are transmitted over the Internet by apps in $D_{gt}$. After checking these data points against privacy policies, *IoTProfiler* identified that 56 data points are disclosed in the privacy policies, and raised alarms for 391 data points in 23 apps. Among the 391 alarms, we confirmed 26 false positives (93.4% precision). Most of the false positives (24) are caused by IDI that identifies some non-IoT data points as IoT data points (see Section 5.2). In the remaining alarms, *IoTProfiler* failed to match the data points to the related terms in privacy policies using semantic analysis. An example is the `bike.cobi.app` app: the app exposes an IoT data point "distance" (i.e., scheduled cycling distance of the user) to the Internet, while discloses the collection of a seemingly different term "planned km" in its privacy policy. Note that in this experiment, we were able to find privacy policies for 39 of the apps in $D_{gt}$. Among these apps, only seven apps disclosed the collection of some IoT data, while 14 apps did not disclose the collection of any specific data point and 18 apps only focused on the mobile/website data.

We were not able to discuss the recall of *IoTProfiler* due to the lack of ground truth, i.e., how many IoT data points are leaked without being detected by LT. However, the tool we adopted in LT for data flow analysis, i.e., FlowDroid with IccTA, is still the state-of-the-art solution that achieves a better precision than other tools such as Amandroid [78] and DroidSafe [38], and a comparable recall at 76% to 90% on different benchmarks [19, 58, 78]. As we will show in the following section, FlowDroid indeed allows us to identify many data exposure risks in real devices.

**Dynamic validation with real IoT devices.** We performed dynamic validation to inspect whether the data items labeled by *IoTProfiler* are indeed exposed to the Internet. For this purpose, we purchased 10 IoT devices (actively being used and available in the market) that are reported to expose at least three IoT data items from $D_{cp}$. We report our dynamic analysis results in Table 3. Specifically, we performed as many device operations as possible in the companion apps, and inspected the apps' runtime traffic[2]. Among the reported 97 data items by *IoTProfiler*, 77 (72.2%) of them are confirmed to be exposed. Note that this does not mean that the rest 20 data items are false negatives of *IoTProfiler*. For example, we observed that IoT devices often implement custom encryption algorithms, whose data exposure can not be confirmed due to the encrypted network traffic. Besides, the exposure of some IoT data items can be triggered only under specific conditions such as IoT device crashes, which may prevent us from observing data leaks in a dynamic experiment.

[2]We decrypted HTTPS traffic by installing a root certificate to the test device.

Table 3: Dynamic analysis over 10 real IoT devices.

| Device Name | App Package Name | # Confirmed/ Reported |
|---|---|---|
| RENPHO body scale | com.renpho | 3/5 (60.0%) |
| Kankun smart plug | com.kankunit.smartplugcronus | 7/10 (70.0%) |
| Yi home camera | com.ants360.yicamera | 12/17 (70.6%) |
| LIVALL smart helmet | com.livallsports | 2/3 (66.7%) |
| PETKIT pet feeder | com.petkit.android | 12/17 (70.5%) |
| Lexin Smart Band | com.lifesense.LSWearable.intl | 10/11 (90.9%) |
| Yeelight smart bulb | com.yeelight.cherry | 6/8 (75.0%) |
| YUNMAI smart scale | com.yunmai.scaleen | 7/11 (63.6%) |
| ASUS router | com.asus.aihome.profile | 6/8 (75.0%) |
| | Total | 70/97 (72.2%) |

## 5.4 Performance Overhead

We use the dataset $D_{cp}$ to evaluate the performance of *IoTProfiler* on one workstation with Red Hat Linux with an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz and 16 GB memory. The overhead mainly focuses on two parts: (1) pre-processing the app and using semantic analysis to locate interesting IoT data, (2) data-flow analysis to detect IoT data exposure. Our evaluation shows that, for each app, *IoTProfiler* takes only 22.2 seconds on average for identifying IoT data, with an average of 169.6 seconds for detecting and reporting IoT data exposures. Overall, *IoTProfiler* spent a total of 330.7 hours, with an average of 3.2 minutes for processing each app. Among all apps in $D_{cp}$, 168 (2.7%) could not be successfully analyzed, either due to a time-out or memory running out issue in FlowDroid [20]. The result shows *IoTProfiler* can effectively process a large number of apps.

# 6 Measurement

## 6.1 IoT Data Exposure in the Wild

**Landscape.** Our study shows that IoT data exposure is pervasive in the wild (see Table 4). In total, *IoTProfiler* identified 50,667 IoT code blocks and 174,943 IoT data points from 5,795 apps (93.3% out of 6,208 apps in $D_{cp}$). Among these apps, 1,973 apps (31.8%) from at least 1,559 unique device vendors are found to collect sensitive IoT data without proper disclosure, and each app exposes 5.6 data items on average. This result indicates that a significant portion of IoT devices on the market endangers user privacy for the lack of data transparency. Also seriously, we observed that 557 (9.0%) apps share IoT data with third parties, and 425 (6.8%) apps use clear-text transmission (although this bad practice has been extensively discussed [40, 73]).

A comparison of app stores shows that 47.5% apps from the Chinese store (i.e., 360 Store) inappropriately exposed IoT data, while the ratio for US store is only 29.2%. We believe the main cause of the difference is that the Chinese store do

not implement similar levels of privacy protections (e.g., not requiring privacy policies on the app store [56]) as the US store. Another notable finding is that there are significantly larger portion of apps in the Chinese store than in the US store that deliver IoT data via insecure transmission (15.6% v.s. 5.4%) and send the data to third parties (16.2% v.s. 7.8%). This finding is aligned with other data privacy studies [57, 80], and confirmed the serious privacy situation of IoT users in the Chinese market.

**Exposed contents.** As shown in Table 4, there are 17.4% apps that expose device tracking data, with each app having 1.9 unique data items (see row-3). In particular, 4.7% of the apps the device tracking data to third parties. Note that while the scale of data exposure is less prevalent than the leaks of personal data (e.g., 12.9% apps leak gender info as shown in previous research [64], such practice can have serious privacy implications [6, 59] as it allows third-party trackers to profile a device owner's activities by associating their IoT data to the device tracking data.

We also found that 19.7% of apps expose the timing data of IoT devices. Specifically, as shown in Table 5, 348 (5.6%) apps collect the `start/end time` and 191 (3.1%) apps collect the `duration` data, which indicates the time schedule of specific activities of the user, such as smoking, sleep, and sexual activities. Further, there are 11.5% apps that expose IoT sensor data. Prominent examples of such data are `body weight` which is exposed by 177 apps, and `temperature` value exposed by 92 apps.

In addition, we noticed that the US and Chinese stores expose a very similar set of IoT data items (see the top identified data items in Table 5), meaning the stores are not targeting specific IoT data items. However, the apps from the Chinese store are indeed more likely to expose the data items, e.g., the data items such as *device id*, *wifi password*, and *start/end time* are over twice as likely to be exposed by Chinese apps.

**Data exposure in different categories of IoT devices.** We studied the IoT devices in 25 most popular categories as reported by *iotlineup.com* [5] and analyzed data exposure across these categories. Specifically, we collected keywords for each device category (e.g., "smart bulb" for *Indoor Lighting*) and then classified the apps into the categories by matching keywords in app descriptions. This allows us to report privacy risk for different device categories. As can be seen in Figure 5, the apps related to *Smoke/CO Sensors*, *Remote Controls* and *Sleep Trackers* are most likely to expose IoT data (with a possibility ranging from 61.6% to 81.6%), while only 3.0% of *Toys* apps and 20.4% of the *Automotive* apps disclose sensitive IoT data. Taking a closer look at those apps and related IoT devices, we found that unlike other companion apps, the *Toys* and *Automotive* apps are mostly controllers for smart toys and vehicles that are expected to work in short range, e.g., via Bluetooth, without Internet access. We also noticed that *Smart TVs* and *Residential HVAC Systems* expose the most number of IoT data items. An example is the *Midea Air Conditioner* app that collects over 20 distinctive data items, such as appliance information and user activities, to **.appsmb.com*. It is worth mentioning that we did not notice a major difference between the devices in the Chinese and US markets, in terms of which device type exposes the most/least amount of IoT data.

**Data exposure based on device popularity.** We grouped IoT devices based on their companion apps' number of installs (an indicator for device popularity), and evaluated how likely IoT devices of different popularity would expose IoT data. Our findings are surprising: there is an apparent positive correlation between device popularity and likelihood of data exposure, as illustrated by the dash line in Figure 6. Particularly, 41.8% of the devices in the most popular group (i.e., >500K installs) improperly expose IoT data through the companion apps, while the percentage for the least popular group (i.e., <1K) is only 27.8%). We randomly sampled ten devices from each of the two groups for manual inspection. Not surprisingly, the devices in the most popular group are mainly from large vendors, such as Samsung and ASUS, and *they all have a privacy policy that notes the collection of some IoT data items*. However, there are still three apps that missed certain data items (e.g., timing data and device status) in their privacy policies. In the least popular group, while these devices often handle no less data than more popular devices, most of them (8/10) are not found to share the data to any cloud back-ends, probably for the lack of resources to support cloud services. This experiment likely suggests that compared to the most popular devices, IoT devices from smaller vendors are more likely to process user data locally, posing a lower privacy risk.

We observed that the US and Chinese stores share a similar correlation between the device popularity and the likelihood of improper data exposure. However, there is a significant difference in the distribution of apps in the popularity groups: only 28.6% apps in the Chinese store are installed >1K times, while the percentage for the US store is 62%. Therefore, although only a smaller percentage of apps in the US store will expose IoT data, the apps still have a high potential to affect a large number of users.

**Data sharing with third parties.** For the reported data exposure, we first gather URL domains that receive IoT data, and identify which ones are third-party domains by comparing them to the app package names. Our study shows that 16.2% apps from the Chinese store expose IoT data items to third party domains, over twice as high as the apps from the US stores (7.6%). We report where the third parties are located. In particular, we check the DNS information of the third-party domains found in the code, and leverage `ipvigilante.com` to identify the related countries. In total, we were able to find country information for 588 out of 695 third-party domains. We use the same method to identify the countries of app developers, and plot the cross-country data flows from app developers to third parties in Figure 7. We observed that

Table 4: Overall leakage statistics in $D_{cp}$.

| App Store | Data Type | | Exposure w/o Disclosure | | Insecure Transmission | | Share to Third Party | | Total | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | # Items per App | # Apps | # Items per App | # Apps | # Items per App | # Apps | # Items per App | # Apps |
| Any Store | Device Tracking Data | Device Identifier | 1.2 | 535 (8.6%) | 1.2 | 96 (1.5%) | 1.3 | 137 (2.2%) | 1.2 | 568 (9.1%) |
| | | Network Identifier | 1.8 | 823 (13.3%) | 1.6 | 130 (2.1%) | 1.6 | 208 (3.4%) | 1.8 | 833 (13.4%) |
| | | Subtotal | 1.9 | 1,078 (17.4%) | 1.6 | 197 (3.2%) | 1.7 | 292 (4.7%) | 2 | 1,102 (17.8%) |
| Any Store | Sensor Data | Biometric Data | 2.1 | 278 (4.5%) | 1.8 | 76 (1.2%) | 1.8 | 82 (1.3%) | 2.1 | 285 (4.6%) |
| | | Location Data | 1.9 | 290 (4.7%) | 1.9 | 73 (1.2%) | 1.9 | 83 (1.3%) | 1.9 | 318 (5.1%) |
| | | Environmental Data | 1.6 | 287 (4.6%) | 1.6 | 60 (1.0%) | 1.5 | 78 (1.3%) | 1.6 | 287 (4.6%) |
| | | Subtotal | 2.3 | 711 (11.5%) | 2.2 | 172 (2.8%) | 2.1 | 199 (3.2%) | 2.3 | 735 (11.8%) |
| Any Store | Device Attached Data | Device Metadata | 1.9 | 841 (13.5%) | 1.7 | 142 (2.3%) | 1.8 | 223 (3.6%) | 1.9 | 860 (13.9%) |
| | | Device Usage and Status | 2.4 | 1,128 (18.2%) | 2.1 | 218 (3.5%) | 2.1 | 288 (4.6%) | 2.4 | 1,177 (19.0%) |
| | | Timing Data | 2.6 | 1,225 (19.7%) | 2.3 | 243 (3.9%) | 2.2 | 311 (5.0%) | 2.6 | 1,238 (19.9%) |
| | | Subtotal | 4.3 | 1,722 (27.7%) | 3.6 | 350 (5.6%) | 3.5 | 476 (7.7%) | 4.4 | 1,742 (28.1%) |
| US Store | Any Data Type | | 5.5 | 1,560 (29.2%) | 4.6 | 289 (5.4%) | 4.6 | 416 (7.8%) | 5.7 | 1,579 (29.6%) |
| Chinese Store | | | 6.2 | 413 (47.5%) | 4.5 | 136 (15.6%) | 4.8 | 141 (16.2%) | 6.3 | 413 (47.5%) |
| Any Store | | | 5.6 | 1,973 (31.8%) | 4.6 | 425 (6.8%) | 4.7 | 557 (9.0%) | 5.8 | 1,992 (32.1%) |

Table 5: Top identified data objects with privacy risks.

| Data Type | Data Item | # Apps | |
|---|---|---|---|
| | | US Store | Chinese Store |
| Device Tracking Data | device id | 318 (6.0%) | 113 (13.0%) |
| | wifi password | 247 (4.6%) | 110 (12.6%) |
| | mac address | 154 (2.9%) | 36 (4.1%) |
| | ssid | 154 (2.9%) | 32 (3.7%) |
| Sensor Data | body weight | 135 (2.5%) | 42 (4.8%) |
| | temperature | 69 (1.3%) | 23 (2.6%) |
| | altitude | 39 (0.7%) | 21 (2.4%) |
| | humidity | 37 (0.7%) | 9 (1.0%) |
| Device Attached Data | start/end time | 251 (4.7%) | 97 (11.1%) |
| | model name | 244 (4.6%) | 62 (7.1%) |
| | device name | 210 (3.9%) | 71 (8.2%) |
| | duration | 162 (3.0%) | 29 (3.3%) |

Table 6: Top 10 third-party domains that collect IoT data.

| 3P Domain | # Apps | Country | Domain type |
|---|---|---|---|
| *.tuya.com | 69 | China | IoT B2B Supplier |
| *.ys7.com | 14 | China | IoT B2B Supplier |
| *.keeprapid.com | 14 | China | IoT Solution Provider |
| *.dropcam.com | 11 | United States | IoT Manufacturer |
| *.mykronoz.com | 9 | Switzerland | IoT Manufacturer |
| *.fitbit.com | 7 | United States | Connected Health Platform |
| *.facebook.com | 5 | United States | Social Networking Service |
| *.strava.com | 5 | United States | IoT Data Integration Platform |
| *.sleepace.net | 3 | China | IoT Solution Provider |
| *.anlian.co | 3 | China | IoT Manufacturer |

cross-country data transfer is very common, with the United States and China being two major data hubs that receive most IoT data from other regions, followed by Canada, France, Germany, and South Korea.

Further, Table 6 shows the top 10 third parties that receive data from the most number of companion apps. On top the list are IoT B2B suppliers and solution providers. Tuya [69], as an example, is such an IoT B2B supplier that produces physical IoT devices and companion apps on behalf of IoT device vendors according to their hardware/software requirements. Although accelerated device creation, this workflow indeed poses new privacy risks: IoT device users often have an illusion that only the device vendors have access to their data, but in fact any upstream vendors may collect their data. As another example, a smart mattress sold by an Australian company (*AH Beard Pty Ltd*) to local customers is found to send various sleep-related data to a company in China (i.e., **.sleepace.net*). For these cases, the users should be alerted for the risks of cross-region data sharing, considering privacy regulations such as GDPR that restricts such kind of data practices [43].

**Privacy policies of IoT vendors.** We found that about a third (678) of the 1,973 apps that expose sensitive IoT data fail to provide a valid privacy policy. Specifically, up to 79.9% of the apps from Chinese market (i.e., 360 Store) do not have privacy policies, compared to only 17.5% on Google Play. This is likely because Chinese market put less restrictions on app/device privacy compliance. Even in the privacy policies we found, there are a few alarming problems that may prevent them from providing proper data transparency to device users.

First, IoT vendors may provide different types of devices that collect distinctive IoT data points. An example is `D-Link` that provides WiFi routers, smart cameras, and smoke sensors. For better privacy disclosure, the vendors are expected to have dedicated privacy policies for each type of devices. However, our study on the 347 vendors that own at least two types of devices shows that almost all vendors use generic policies to cover different devices rather than providing device specific policies. This finding reveals that the privacy policies in IoT context are likely too coarse-grained to provide reliable disclosure to the users.

Also surprisingly, we noticed that privacy policies are often shared by different IoT vendors. Specifically, we clustered 4,182 privacy policies based on their document similarity (with the documents' TD-IDF features [65]), and identified 122 clusters of similar privacy policies. We found that some clusters indeed correspond to different device vendors. An example is a P2P camera (`x.p2p.cam`), which although is not built on `Tuya` [69] — an IoT development platform through which many IoT device manufacturers easily deploy OEM devices and companion apps, uses `Tuya` privacy policy without any modification. Also concerning is that, some IoT vendors that leverage Tuya, such as `Enerwave WiFi Switch`, also share the generic privacy policy of `Tuya`. Such findings highlight the need for an effective approach to inspect the accuracy and completeness of IoT related privacy policies.

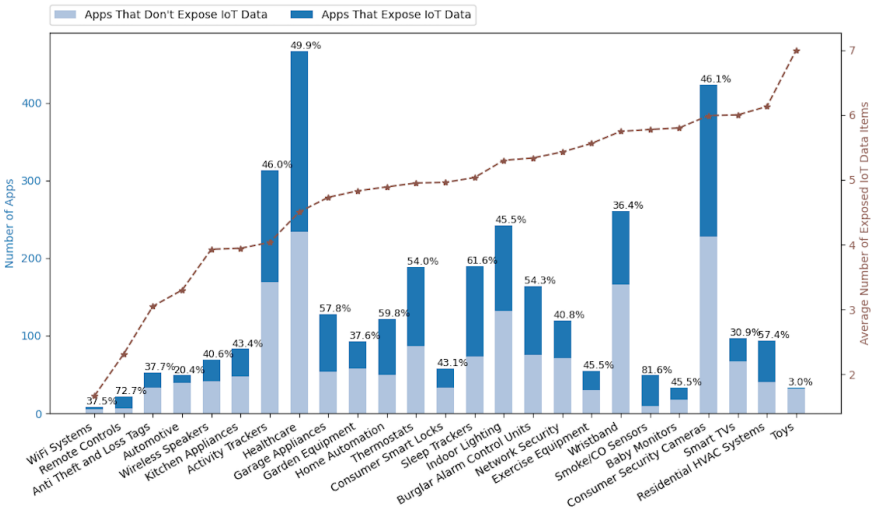



Figure 5: IoT data exposure for different device types.

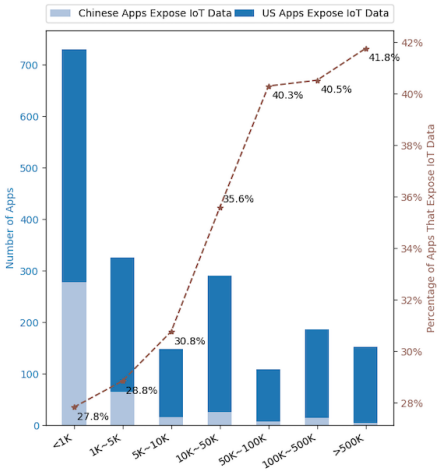



Figure 6: IoT data exposure for different device popularity.

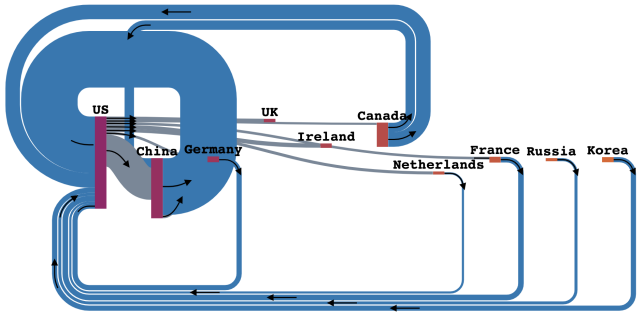



Figure 7: Cross-region IoT data flows.

## 6.2 Case Studies

Below we elaborate two noteworthy cases discovered in the research.

**Health monitoring devices.** Health monitoring devices, such as smart bracelets and sleep trackers, have gained great popularity in recent years. These devices are capable of collecting highly-sensitive health data as medical devices, and therefore may lead to serious privacy risks. To give a few examples, a smart bracelet called 37°, whose companion app has over 140K downloads. On the one hand, the companion app would upload any possible data, such as user heart rate, blood pressure and sleep start time, to its own server *https://d37service.37bit.net:8443*. But on the other hand, the app has a partnership with a health analytics platform *healthlink.cn* that helps customers to understand their health status. The way to achieve this is interesting: the data is not shared directly via the app for analysis. Instead, the app sends *userid* (bound to the app's own server) to *healthlink.cn*. This caught our attention as *37°* may allow third parties to retrieve customer's health data using a *userid*. Moreover, *healthlink.cn* has a partnership with more than 20 health insurance and health tech companies [49].

**Cigarette holder.** The Angmi cigarette holder [17] allows smokers to monitor how much harmful substances (e.g., nicotine) they have inhaled via a companion app. We found that, the app not only collects data that are shown in the app, such as the number of cigarettes one has smoked, but also exposes extra data without the user's awareness, e.g., gathering data points that may reveal a user's smoking habits (such as when and how many times a user smokes), health conditions (e.g., breathing capacity), and even where they smoke, etc. Even worse, we noticed that such extra data points are sent over HTTP to the cloud. This case confirmed a worsening situation that the device users are not disclosed of IoT data collection, and meantime the device vendors are not capable of safeguarding the data.

## 6.3 Responsible Disclosure

We have made responsible disclosures to both the app developers and app stores. For most of the apps (e.g., apps available on Google Play), we can get their developer emails from the app stores. Therefore, we have sent a total of 1,381 emails for the apps that fail to disclose some IoT data items, and suggested the developers update their privacy policies. We used a Google Sheets extension "Yet Another Mail Merge" (YAMM) [72] to track the status of the emails. One month after our report, the emails were opened by 381 developers, remain unopened for 850 developers, and bounced from 150 invalid email addresses. Note that in this process, we only collected aggregated results of the emails provided by YAMM. We never inspected the tracking information of individual email addresses. The IRB from our institutions also confirmed

that this process was not subject to IRB review of human subject research since we did not obtain information about/from individuals. Among the 381 opened emails, 21 developers have acknowledged our findings and made changes to their privacy policies to better disclose IoT data collection. Two developers asked us to not include their names in our report. For the rest of the emails, we did not receive any response yet, or only received automatic responses.

Due to the low response rate of from app developers, we also reported the list of apps to Google Play, 360 Store, and APKPure, and asked their help to contract the developers. Google Play responded to our request quickly and were investigating the privacy issues of the apps, while both 360 Store and APKPure have not responded to our requests yet.

## 7 Discussion

Besides enabling the large-scale study, *IoTProfiler* can contribute to other important scenarios. In particular, *IoTProfiler* allows benign IoT device vendors to automatically check privacy compliance of their IoT products. This is especially important since the developers often fail to know what data are collected by third-party libraries, which leads to serious privacy risks as reported in the prior study [75]. Further, complementary to the line of works that define privacy labels [25, 31, 82], *IoTProfiler* can be easily tuned to automatically generate fine-grained privacy labels, and provide better data transparency to IoT users and potential customers.

**Limitations.** *IoTProfiler* relies on the assumption that IoT companion apps carry text labels that indicate the presence of IoT data. Our results show that the assumption holds for the vast majority of apps, which allows us to locate IoT data in 93.3% of the apps (i.e., 5,795 out of 6,208 apps in $D_{cp}$, see Section 6.1). However, *IoTProfiler* is still limited in cases where app developers deliberately avoid using any meaningful labels to describe their IoT data. Further, we build an IoT taxonomy by collecting IoT data points from a known set of technical reports and documentations, which may inevitably introduce biases to our study due to the potential presence of other uncovered IoT devices and their data. Therefore, our approach will benefit from more effort to add diversified IoT data to the taxonomy. *IoTProfiler* also inherits a few weaknesses from its underlying techniques. For instance, we perform static app analysis and thus can not handle any dynamic features of the apps, e.g., such as side-loaded code. There are also false positive/negative reports due to the inaccuracies in data flow and privacy policy analysis. Such limitations can be alleviated by incorporating more advanced app and privacy policy analysis tools. Besides, similar to previous works [53, 63], we identify third-party domains by comparing them to package names. This method is not reliable when package names fail to reveal the identity of IoT device vendors.

## 8 Related Work

**Analysis of IoT mobile companion apps.** A few prior works focused on analyzing the mobile apps of IoT devices for security vulnerability discovery, e.g., prior studies [24, 77] use in-app firmware fingerprints to identify IoT vulnerabilities. In contrast, *IoTProfiler* leverages the observation that companion apps process and label IoT data to identify exposed IoT data, which has not been done before.

**IoT privacy studies.** Recent research extensively discussed privacy implications of IoT devices. Specifically, Naeini et al. [31] conduct interviews and surveys with privacy experts to identify "privacy labels" that are required to inform IoT device users. IoTWatch [21] and SAINT [23] perform code instrumentation and taint analysis to improve privacy awareness for SmartThings apps and device users. In comparison, *IoTProfiler* does not analyze the cloud-side SmartThings apps (simple and open-source scripts), but instead focuses on the more complicated mobile-side IoT companion apps. *IoTProfiler* entails new techniques (Section 4) and enables us to study data exposure of a much broader range of IoT devices (compared to devices only under SmartThings).

Other works detect privacy issues based on network traffic generated by real IoT devices. Ren et al. [63] evaluate and report data exposure of 81 real IoT devices. Kumar et al. [46] and IoTInspector [39] utilize crowdsourcing and network-level scanning to collect labeled IoT traffics from smart homes. The network traffic based approaches suffered from key limitations such as the lack of concrete data items that can be collected for analysis (Section 3). In sharp contrast, for example, they could only analyze aggregated staticstics while we can scrutinize fine-grained date items. Further, we developed novel techniques (Section 4) such as a learning-based semantic-locator to identify IoT data in apps, and new data taxonomy to facilitate the semantic-analysis. These techniques enabled us to perform a large-scale, fine-grained analysis of IoT data exposure different from prior work (e.g., thousands of devices compared to tens and low-hundreds, fine compared to coarse grained analysis).

**Privacy analysis of mobile apps.** Recent works have discussed mobile apps and their privacy policies [16, 68, 76, 81, 84]. These works first analyze mobile apps to identify what data are collected and shared. Afterwards, they check whether the data practices are disclosed in the app privacy policies. We note that these works cannot be used to identify IoT data exposure. Specifically, they rely on a pre-defined list of information-accessing APIs to locate mobile sensitive data. However, the sensitive IoT data in our study are not returned by such APIs, and thus identifying them in mobile companion apps is more difficult. To address the challenge, we build an IoT data taxonomy and new semantic-aware approach to identify IoT data blocks and data points (Section 4.3).

## 9 Conclusion

In this paper, we performed a comprehensive measurement study on unauthorized data collection and exposure of the IoT devices through their mobile companion apps. We propose *IoTProfiler*, a novel framework that combines machine learning and program analysis, to address the challenge of identifying IoT device data and tracking their disclosure from mobile apps. We performed an analysis of over 6,208 IoT companion apps to understand the data exposure risks at the market level. Our research highlighted a series of findings about IoT data exposure, which reveal the urgent need for technologies that provide transparency of IoT data practices for device consumers.

## Acknowledgments

We thank our shepherd and the anonymous reviewers for their valuable comments and suggestions. Yuhong Nan was supported in part by the National Natural Science Foundation of China (62202510), the Fundamental Research Funds for the Central Universities, Sun Yat-sen University (22lgqb26), the Alibaba Innovative Research Program (AIR) of Alibaba Group. Xiaojing Liao was partially supported by the funding from the Grant Thornton Institute and Indiana University Institute for Advanced Study (IAS). Luyi Xing was supported in part by Indiana University's IAS Collaborative Research Award, FRSP-SF, and REF. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of our sponsors.

## References

[1] Best practices for prominent disclosure and consent. https://support.google.com/googleplay/android-developer/answer/11150561. Accessed:2022-03-06.

[2] Iotprofiler data release. https://sites.google.com/view/iotprofiler. Accessed: 2020-09-04.

[3] Openhab - the open home automation bus. https://www.openhab.org/docs/. Accessed: 2020-06-14.

[4] Philips hue developer program. https://developers.meethue.com/develop/hue-api/. Accessed: 2020-06-14.

[5] Iot devices list. http://iotlineup.com, Aug 2016. Accessed: 2022-03-12.

[6] General data protection regulation (gdpr). https://gdpr-info.eu, May 2018. Accessed: 2021-01-19.

[7] Mi home - documents and resource center. https://iot.mi.com/new/doc/cloud-to-cloud/app/api.html, 2020. Accessed: 2020-06-14.

[8] Smanos wifi alarm system. https://www.smanos.com/w120i, 2020. Accessed: 2020-06-16.

[9] Text classification - fasttext. https://fasttext.cc/docs/en/supervised-tutorial.html, Oct 2020. Accessed: 2022-03-12.

[10] Qihoo 360. 360 mobile assistant - app installation. http://zhushou.360.cn/soft/. Accessed: 2020-11-01.

[11] Adafruit. Adafruit privacy policy. https://www.adafruit.com/privacy. Accessed: 2020-09-04.

[12] Connectivity Standards Alliance. Matter. https://buildwithmatter.com. Accessed: 2021-10-10.

[13] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *Proc. SP*. IEEE, 2019.

[14] Ben Andow. Html privacy policy to plaintext converter. https://github.com/benandow/HtmlToPlaintext, May 2020. Accessed: 2022-03-12.

[15] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. Policylint: investigating internal privacy policy contradictions on google play. In *Proc. USENIX Security*, 2019.

[16] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with policheck. In *Proc. USENIX Security*, 2020.

[17] Angmi. Angmi cigarette holder. http://www.angmi.com.cn/index.html. Accessed: 2018-08-14.

[18] APKPure. Apkpure, download apk free online. https://apkpure.com/. Accessed: 2020-06-16.

[19] Steven Arzt. Static data flow analysis for android applications. 2017.

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

[21] Leonardo Babun, Z Berkay Celik, Patrick McDaniel, and A Selcuk Uluagac. Real-time analysis of privacy-(un) aware iot applications. *Proceedings on Privacy Enhancing Technologies*, 2021.

[22] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5, 2017.

[23] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. Sensitive information tracking in commodity iot. In *Proc. USENIX Security*, 2018.

[24] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In *Proc. NDSS*, 2018.

[25] Richard Chow. The last mile for iot privacy. *IEEE Security & Privacy*, 15(6), 2017.

[26] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1), 1960.

[27] Michal Danilk. langdetect. https://github.com/Mimino666/langdetect. Accessed: 2022-04-13.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[29] Shuaike Dong, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, Xiaofeng Wang, and Kehuan Zhang. Understanding android obfuscation techniques: A large-scale investigation in the wild. In *Proc. Security and Privacy in Communication Systems*. Springer, 2018.

[30] Yue Duan, Mu Zhang, Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li, Xueqiang Wang, and XiaoFeng Wang. Things you may not know about android (un) packers: A systematic study based on whole-system emulation. In *Proc. NDSS*, 2018.

[31] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an iot privacy and security label? *arXiv preprint arXiv:2002.04631*, 2020.

[32] Patrick X. Fowler. Why you need a privacy policy. https://www.swlaw.com/blog/data-security/2015/03/12/why-you-need-a-privacy-policy-part-2-avoiding-three-common-fumbles/, Mar 2015. Accessed: 2022-04-13.

[33] GenericException. Obfuscation tools. https://github.com/GenericException/SkidSuite/blob/master/obfuscation.md. Accessed: 2021-10-05.

[34] Google. Android apps on google play. https://play.google.com/store/apps. Accessed: 2020-06-14.

[35] Google. Google translate. https://translate.google.com. Accessed: 2022-03-12.

[36] Google. bert. https://github.com/google-research/bert, Oct 2020. Accessed: 2022-03-12.

[37] Google. word2vec. https://code.google.com/archive/p/word2vec/, Oct 2020. Accessed: 2022-03-12.

[38] Michael I Gordon, Deokhwan Kim, Jeff H Perkins, Limei Gilham, Nguyen Nguyen, and Martin C Rinard. Information flow analysis of android applications in droidsafe. In *Proc. NDSS*, 2015.

[39] Danny Yuxing Huang, Noah Apthorpe, Gunes Acar, Frank Li, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *arXiv preprint arXiv:1909.09848*, 2019.

[40] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. Supor: Precise and scalable sensitive user input detection for android apps. In *Proc. USENIX Security*, 2015.

[41] Apple Inc. App privacy details on the app store. https://developer.apple.com/app-store/app-privacy-details/. Accessed: 2022-04-13.

[42] Apple Inc. Homekit accessory protocol. https://developer.apple.com/support/homekit-accessory-protocol. Accessed: 2020-09-03.

[43] intersoft consulting services AG. Gdpr: Transfers of personal data to third countries or international organisations. https://gdpr-info.eu/chapter-5/. Accessed: 2020-06-14.

[44] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[45] Nicole Kobie. A sex toy lawsuit highlights privacy concerns around 'smart' dildos. https://www.vice.com/en_us/article/ae3y8e/yes-your-smart-dildo-can-be-hacked, Feb 2015. Accessed: 2020-06-16.

[46] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. All things considered: an analysis of iot devices on home networks. In *Proc. USENIX Security*, 2019.

[47] Eric Lafortune. Proguard faq. http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/external/proguard/docs/FAQ.html#encrypt. Accessed: 2021-10-05.

[48] Thomas Linden, Rishabh Khandelwal, Hamza Harkous, and Kassem Fawaz. The privacy policy landscape after the gdpr. In *Proc. PETS*.

[49] HealthLink Services Co. Ltd. Healthlink partners. https://www.healthlink.cn/website/article/partners. Accessed: 2020-06-14.

[50] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proc. the north american chapter of the association for computational linguistics: Human language technologies*, 2013.

[51] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W Felten, Prateek Mittal, and Arvind Narayanan. Watching you watch: The tracking ecosystem of over-the-top tv streaming devices. In *Proc. CCS*,

2019.

[52] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. Uipicker: User-input privacy identification in mobile applications. In *Proc. USENIX Security*, 2015.

[53] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. Finding clues for your secrets: semantics-driven, learning-based privacy discovery in mobile apps. In *Proc. NDSS*, 2018.

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

[55] Guardsquare nv. Java obfuscator and android app optimizer | proguard. https://www.guardsquare.com/en/products/proguard. Accessed: 2020-06-14.

[56] 360 Mobile Open Platform. Software submission. https://dev.360.cn/wiki/index/id/21. Accessed: 2022-04-13.

[57] Danish Khan Priyanka Sangani. Chinese apps seek excessive information from users: Survey. https://economictimes.indiatimes.com/tech/internet/chinese-apps-seeking-way-more-information-than-needed-survey/articleshow/67633562.cms, Jan. 2019. Accessed: 2022-02-20.

[58] Lina Qiu, Yingying Wang, and Julia Rubin. Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe. In *Proc. ISSTA*, 2018.

[59] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. Apps, trackers, privacy, and regulators a global study of the mobile tracking ecosystem. In *Proc. NDSS*, 2018.

[60] Nilo Redini, Andrea Continella, Dipanjan Das, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Diane: identifying fuzzing triggers in apps to generate under-constrained inputs for iot devices. In *Proc. SP*. IEEE, 2021.

[61] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[62] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[63] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Proc. IMC*, 2019.

[64] Jingjing Ren, Martina Lindorfer, Daniel J Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. Bug fixes, improvements, ... and privacy leaks: A longitudinal study of pii leaks across android app versions. 2018.

[65] scikit-learn developers. Tfidfvectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. Accessed: 2021-10-12.

[66] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8), 2018.

[67] Vijay Sivaraman, Hassan Habibi Gharakheili, Clinton Fernandes, Narelle Clark, and Tanya Karliychuk. Smart iot devices in the home: Security and privacy implications. *IEEE Technology and Society Magazine*, 37(2), 2018.

[68] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. Toward a framework for detecting privacy policy violations in android application code. In *Proc. ICSE*, 2016.

[69] Tuya Smart. Tuya api overview. https://docs.tuya.com/docDetail. Accessed: 2020-06-19.

[70] Inc. SmartThings. Smartthings api. https://smartthings.developer.samsung.com/docs/index.html, 2020. Accessed: 2020-06-16.

[71] Breena R Taira, Vanessa Kreger, Aristides Orue, and Lisa C Diamond. A pragmatic assessment of google translate for emergency department instructions. *Journal of General Internal Medicine*, 36(11), 2021.

[72] Talaria. Yet another mail merge: Mail merge for gmail. https://workspace.google.com/marketplace/app/yet_another_mail_merge_mail_merge_for_gm/52669349336. Accessed: 2022-04-13.

[73] Vincent F Taylor and Ivan Martinovic. To update or not to update: Insights from a two-year study of android app evolution. In *Proc. AsiaCCS*, 2017.

[74] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot: A java bytecode optimization framework. In *CASCON First Decade High Impact Papers*. 2010.

[75] Jice Wang, Yue Xiao, Xueqiang Wang, Yuhong Nan, Luyi Xing, Xiaojing Liao, JinWei Dong, Nicolas Serrano, Haoran Lu, XiaoFeng Wang, et al. Understanding malicious cross-library data harvesting on android. In *Proc. USENIX Security*, 2021.

[76] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. Guileak:

Tracing privacy policy claims on user input data for android applications. In *Proc. ICSE*, 2018.

[77] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. Looking from the mirror: evaluating iot device security through mobile companion apps. In *Proc. USENIX Security*, 2019.

[78] Fengguo Wei, Sankardas Roy, and Xinming Ou. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. *ACM Transactions on Privacy and Security (TOPS)*, 21(3), 2018.

[79] Daniel Wood, Noah Apthorpe, and Nick Feamster. Cleartext data transmissions in consumer iot medical devices. In *Proc. COMM*, 2017.

[80] Adam Xu. Cybersecurity experts worried by chinese firm's control of smart devices. https://www.voanews.com/a/east-asia-pacific_voa-news-china_cybersecurity-experts-worried-chinese-firms-control-smart-devices/6209815.html, Aug 2021. Accessed: 2022-02-20.

[81] Le Yu, Xiapu Luo, Xule Liu, and Tao Zhang. Can we trust the privacy policies of android apps? In *Proc. DSN*. IEEE, 2016.

[82] Pierre-Antoine Vervier Yun Shen. Why we need a security and privacy "nutrition label" for iot devices. https://symantec-enterprise-blogs.security.com/blogs/expert-perspectives/why-we-need-security-and-privacy-nutrition-label-iot-devices, Feb 2019. Accessed: 2020-06-17.

[83] Yan Zhuang. The performance cost of software obfuscation for android applications. *Computers & Security*, 73, 2018.

[84] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven Bellovin, and Joel Reidenberg. Automated analysis of privacy requirements for mobile apps. In *Proc. AAAI*, 2016.

[85] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shormir Wilson, Norman Sadeh, Steven M. Bellovin, and Joel Reidenberg. Automated analysis of privacy requirements for mobile apps. In *Proc. NDSS*, 2017.

## Appendix A Comparing different models for IoT Code Block Identification

To justify our selected fastText embedding model, we also tried other alternative text embeddings. Specifically, we implemented word2vec and BERT, both of which are widely used word-embeddings in NLP. For word2vec, we first performed word-piece tokenization for each IoT code block. Then, we trained a word2vec skip-gram model using Gensim [61]. We set the min_count as 5 and set the vector dimension as 100. Note that we did not take those public-available word2vec model (e.g., the one trained on the Google News dataset [37]) for our classification task, as the semantics of program code are very different from natural language, which causes many out-of-vocabulary words. For the BERT embedding, we used the BERT-base uncased model [36], and fine-tuned it over our training corpus. In our research, all text labels in each code block were treated as a single sentence. The BERT model outputs the sentence-level embeddings of each code block. The model has 12 transformer encoding layers, with the embedding dimension of 768. The number of word-pieces is around 30K.

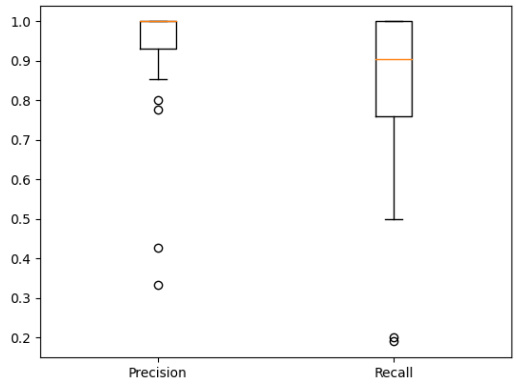



Figure 8: App-level precision and recall distribution of IoT Data Identifier.

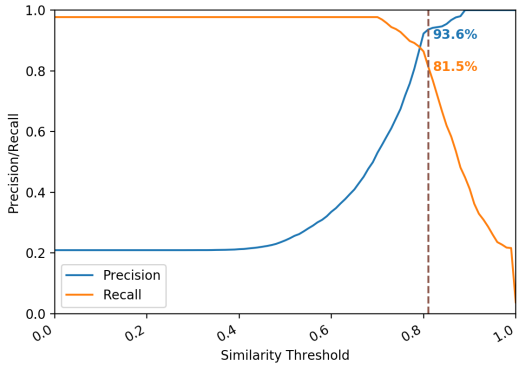



Figure 9: Precision and recall of identifying IoT data items with different similarity thresholds.