

Research Article

Statistical Feature-Based Personal Information Detection in Mobile Network Traffic

Shuang Zhao , Shuhui Chen , and Ziling Wei 

School of Computer, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Shuhui Chen; shchen@nudt.edu.cn

Received 24 December 2021; Revised 29 April 2022; Accepted 23 June 2022; Published 6 July 2022

Academic Editor: Ding Wang

Copyright © 2022 Shuang Zhao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the popularity of smartphones, mobile applications (apps) have penetrated the daily life of people. Although apps provide rich functionalities, they also access a large amount of personal information simultaneously. As a result, privacy concerns are raised. To understand what personal information the apps collect, many solutions are presented to detect privacy leaks in apps. Recently, the traffic monitoring-based privacy leak detection method has shown promising performance and strong scalability. However, it still has some shortcomings. Firstly, it suffers from detecting the leakage of personal information with obfuscation. Secondly, it cannot discover the privacy leaks of undefined type. Aiming at solving the above problems, a new personal information detection method based on traffic monitoring is proposed in this paper. In this paper, statistical features are designed to depict the occurrence patterns of personal information in the traffic, including local patterns and global patterns. Then a detector is trained based on machine-learning algorithms to discover potential personal information with similar patterns. Since the statistical features are independent of the value and the type of personal information, the trained detector is capable of identifying various types of privacy leaks and obfuscated privacy leaks. As far as we know, this is the first work that detects personal information based on statistical features. Finally, the experimental results show that the proposed method could achieve better performance than the state-of-the-art.

1. Introduction

With the popularity of mobile devices, millions of apps have been developed to facilitate daily living activities. According to FinancesOnline [1], there are 5.22 billion unique mobile phone users worldwide, and a total of 218 billion mobile apps are downloaded in 2020. Apparently, apps have been indispensable in the daily life of people.

When providing service, apps would request access to personal information (PI), such as device identifiers and location [2, 3]. Some requested PI is necessary to support the functionalities of apps, while some PI is deliberately collected by apps with unilateral intent. Since the majority of users lack professional knowledge, they will unconsciously assent to all requests of the app, even if some requests are unreasonable. In this case, users would be victims of unnecessary privacy leaks. For example, PI of up to 87 million Facebook users had been collected by a third-party Cambridge Analytica without user awareness [4]. Cambridge

Analytica developed a quiz app. Once a user logs into the quiz app with his Facebook account, his data could be collected by the quiz app, such as profile details, user histories, and friends lists. Then the data can be used to automatically predict a range of highly sensitive personal attributes, such as the sexual orientation and personal traits. Similarly, Rela, a Chinese social app, was found to have leaked 5.3 million user profiles [5]. With the disclosure of such privacy leakage events, more and more users are aware of privacy concerns and are likely to appreciate transparency when regarding how the app collects personal data. Furthermore, Ren et al. [6] point out that the data leakage of apps has gotten worse after investigating how privacy leaks change over time for 512 apps. Therefore, there is an urgent need to detect privacy leaks in apps.

Existing privacy leak detection methods can be divided into three categories, including the static analysis-based detection [7, 8], the dynamic analysis-based detection [9, 10], and the network traffic monitoring-based detection

[11–13]. With the data flow analysis technology, the static analysis-based method finds the privacy leakage path from the source code of an app. Such a method is scalable, while its accuracy is heavily affected by false positives and false negatives. Besides, the effectiveness of this method will further decline with the popularization of the source code protection technology. The dynamic analysis-based method usually utilizes the customized OS combined with taint analysis to discover the privacy leakage. It could achieve better accuracy than the static analysis-based method. However, it is difficult to deploy this method on a large scale due to the high overhead. Compared with the above two methods, the network traffic monitoring-based method provides a user-friendly and scalable detection scheme. This method first monitors the traffic of apps passively, then analyzes the content of the traffic and discovers the PI that is transmitted through the traffic. For instance, Recon [12] regards the traffic as a short text and applies the bag-of-words model to extract features. Then a Random Forest classifier is trained to detect the personally identifiable information (PII) in the traffic. The network traffic monitoring-based method can be easily deployed on personal mobile devices. Besides, its detection performance is comparable to that of the other two methods. Hence, this method has aroused considerable interest among related parties [14].

However, the existing traffic monitoring-based detection methods identify PI by the semantics and the formats of PI, which leads to some shortcomings. Firstly, those methods have poor performance in detecting the PI with obfuscation, such as the PI that is hashed or encrypted. Secondly, they cannot identify the PI of undefined types (e.g., the PI with unknown formats). In this paper, a novel traffic monitoring-based detection method is proposed. It could solve the above two problems effectively. Instead of distinguishing PI and non-PI by their semantics and formats, our method focuses on the differences in their occurrence patterns. Specifically, our method is inspired by the observation that PI and non-PI transmitted in the traffic could show different occurrence patterns. Take IMEI (a device identifier) and the timestamp (a non-PI) as examples. An app would obtain IMEI from multiple users multiple times, while the IMEI obtained from the same user is unique. Meanwhile, the IMEI of a user would be collected by multiple apps; hence, the IMEI could occur in the traffic of multiple apps. As for the timestamp, the timestamp collected from one user would change frequently, and a timestamp rarely appears in multiple apps at the same time. For PI and non-PI, their occurrence patterns are independent of their semantics and formats. Therefore, the undefined PI can be discovered as long as it has similar occurrence patterns. In addition, the obfuscated PI can also be detected by our method when the obfuscation does not affect the patterns of how PI occurs in the traffic.

The occurrence patterns are captured from local and global perspectives in our method. Specifically, local and global statistical features are designed to describe how PI occurs in the traffic of one app and all apps. Then a group of string-form rules and manual work are employed to build a labeled dataset. Finally, a detector is trained to learn the

occurrence patterns of PI and non-PI based on machine-learning algorithms. With the detector, the PI transmitted through the traffic can be discovered automatically. The contributions of this paper are summarized as follows (the preprint version of our paper can be found in [15]):

- (1) A set of local and global statistical features is designed to describe the occurrence patterns of PI in the traffic. To our best knowledge, this is the first work to detect PI in the traffic from the perspective of statistical features
- (2) A PI detector based on machine-learning is presented in this paper. The detector could identify various types of PI without knowing the formats and the semantics of PI. Besides, the detector is effective for detecting a portion of obfuscated PI
- (3) Comprehensive experiments are conducted on a real-world dataset to validate the effectiveness of the proposed detector. The experimental results show that the proposed detector could achieve better performance than the state-of-the-art

The rest of the paper is organized as follows. Section 2 provides the necessary background. Section 3 introduces the related work. Then, the designed statistical features are given in Section 4. Section 5 presents the detection method, and Section 6 shows the experimental results. Finally, Section 7 concludes the paper.

2. Background

In this section, background information is provided to help understand this paper.

2.1. Personal Information. PI refers to the information that can identify a specific individual or reflect the activities of an individual [16]. Under different standards and application scenarios, the specific information belonging to PI is different. PII refers to the PI that can be used to identify or track an individual, which is the most sensitive information about one person. So far, there are several types of well-defined PII, including device identifiers, user identifiers, contact information, credentials, and location. Current research focuses on identifying those PII. However, in addition to those PII, there are other undefined PII and PI that can reveal sensitive information about users. It is far from enough to identify only the well-defined PII.

Our work is aimed at identifying as much PI as possible. A critical question is how to define those PI, such as what they look like and what they mean. Fortunately, our work could neatly bypass this question. During the detection, our method analyzes the data in the traffic based on the statistical features. Other knowledge, such as the format of the PI, is not required. In this way, our method could discover the PI without a precise definition of it.

2.2. Privacy Leak. The term privacy leak has been widely used to represent the behaviors of apps collecting and transmitting any PI [8, 12]. Recently, a few studies further assess

the rationality of such behaviors. Those studies try to figure out whether the behavior is necessary for the app to provide its service [17, 18]. In those studies, privacy leak indicates the functionality-irrelevant private data transmission.

Since the rationality of transmitting PI is beyond the scope of this paper, privacy leak represents any PI transmission behavior in this paper.

2.3. HTTP Request. For an app, to obtain data from the server based on HTTP, an HTTP request would be sent from the app to the server. An HTTP request contains an HTTP request line, as shown in Figure 1. The HTTP request line describes the request type (the Request Method in Figure 1), the resource to be accessed (the Request URI in Figure 1), and the HTTP version (the Request Version in Figure 1). For the Request URI, it consists of two parts: URI path and URI Query. Moreover, the URI Query could include multiple parameters, and the user data will be transferred to the server through them. Each parameter contains a key and its corresponding value, which are connected with “=.” For the unencrypted mobile traffic, most PI is transmitted through these parameters in the HTTP request [10, 19]. In this paper, we are concerned about whether PI is transmitted through the URI Query Parameter under the request type of “Get” and “Post.” As long as PI is found in the parameters, it is considered that a privacy leak occurs in this HTTP request.

2.4. Motivation. This section explains the idea behind our method in detail with an example. Figure 2 provides examples of eight HTTP requests from three users in two apps (Wechat and Tencent News). There are five keys in those requests, including *imei*, *startDate*, *endDate*, *devid*, and *appver*. The first three keys are from Wechat, and the other two belong to Tencent News. Pair $\langle app, key \rangle$ is used to represent the dependency relationship between a key and an app. Therefore, several $\langle app, key \rangle$ pairs can be obtained from these requests, such as $\langle Wechat, imei \rangle$. Besides, IMEI is transferred through $\langle Wechat, imei \rangle$ and $\langle TencentNews, devid \rangle$. The remaining three pairs are used to transfer non-PI, i.e., the date ($\langle Wechat, startDate \rangle$ and $\langle Wechat, endDate \rangle$) and the app version ($\langle TencentNews, appver \rangle$). Since IMEI is personal information, a privacy leak occurs once an HTTP request from Wechat contains key *imei* or an HTTP request from Tencent News contains key *devid*.

As shown in Figure 2, the values of $\langle app, key \rangle$ pairs that transfer PI and non-PI exhibit different occurrence patterns.

1. PI-related $\langle app, key \rangle$ pairs

Pattern 1: values are unique for different users in the same app

Pattern 2: the value of one user is unique in one app. Meanwhile, the value may also appear in other apps

2. Non-PI-related $\langle app, key \rangle$ pairs

Pattern 1: different users may have the same values in an app, and one user could take multiple values in an app

Pattern 2: the values may not appear in other apps

It should be noted that the above example only shows several typical patterns that can exist for certain PI and

non-PI. Not all PI and non-PI would present such value patterns, while they may show other patterns. It is such differences that motivate us to detect whether an $\langle app, key \rangle$ pair is transmitting PI based on the occurrence patterns of its values. In the following sections, a set of features is proposed to describe the value characteristics of $\langle app, key \rangle$ pairs, and the potential occurrence patterns are automatically learned based on machine-learning algorithms.

3. Related Work

3.1. Static Analysis-Based Detection. The static analysis-based detection method discovers the possible privacy leak in an app by analyzing its source code. The first step of this method is to find sources and sinks in the source code. For example, the APIs provided by the device OS for accessing user data can be regarded as sources. The interfaces that transmit information are defined as sinks, such as the network interface and file writing functions. Then the control flow graph of the source code is generated. Finally, data flow analysis technology is utilized to find paths between sources and sinks. Each path indicates a potential privacy leak.

Since the static analysis is carried out without running apps, this method has good scalability. Thus, it can be applied to implement the preliminary test of massive apps in the application market. However, this method is prone to generate false negatives. On the one hand, it is difficult to analyze the native code and dynamically loaded code. On the other hand, the privacy leaks generated from non-standard sources (e.g., the PI input by users) would be omitted. Besides, since this method finds the leakage paths statically, false positives would be introduced if the paths are never visited.

Based on the static analysis, FlowDroid [20] finds the leakage paths between the predefined sources and sinks. Apart from that, it proposes a scheme to find the privacy leak caused by the UI widgets within apps. Nattanon et al. [21] extend FlowDroid by adding additional PII-related sources into its source-sink file. Besides, a PII list assembled from previous studies is provided in their work. AndroidLeaks [8] provides a static analysis framework for discovering PII leaks in Android apps. PiOS [7] analyzes the binaries compiled from the Objective-C code and detects privacy leaks for iOS apps. ClueFinder [22] also performs its detection based on the source code of apps. Instead of using the data flow analysis, it detects the PI leakage paths by checking the semantics of program elements.

3.2. Dynamic Analysis-Based Detection. The dynamic analysis-based detection performs its data flow analysis with running apps. It could achieve better performance compared with the static analysis since the discovered privacy leaks happen in real data streams. To track how personal data flows in the app, some detection methods apply taint analysis on customized OSs. TaintDroid [23] modifies the Android OS to support attaching taint tags to sources. Four granularities of taint propagation are implemented in TaintDroid, including

```

GET /i?tn=baiduiimage&ps=1&ct=201326592&lm=-1&cl=2&nc=1&ie=utf-8&dyTabStr=MCwzLDIsNCw2LDEsNSw4LDcs0Q%3D%3D&word=fa HTTP/1.1\r\n
> [Expert Info (Chat/Sequence): GET /i?tn=baiduiimage&ps=1&ct=201326592&lm=-1&cl=2&nc=1&ie=utf-8&dyTabStr=MCwzLDIsNCw2LDEsNSw4LDcs0Q%3D%3D&word=fa HTTP/1.1\r\n]
Request Method: GET
Request URI: /i?tn=baiduiimage&ps=1&ct=201326592&lm=-1&cl=2&nc=1&ie=utf-8&dyTabStr=MCwzLDIsNCw2LDEsNSw4LDcs0Q%3D%3D&word=fa
Request URI Path: /i
Request URI Query: tn=baiduiimage&ps=1&ct=201326592&lm=-1&cl=2&nc=1&ie=utf-8&dyTabStr=MCwzLDIsNCw2LDEsNSw4LDcs0Q%3D%3D&word=fa
Request URI Query Parameter: tn=baiduiimage
Request URI Query Parameter: ps=1
Request URI Query Parameter: ct=201326592
Request URI Query Parameter: lm=-1
Request URI Query Parameter: cl=2
Request URI Query Parameter: nc=1
Request URI Query Parameter: ie=utf-8
Request URI Query Parameter: dyTabStr=MCwzLDIsNCw2LDEsNSw4LDcs0Q%3D%3D
Request URI Query Parameter: word=fa
Request Version: HTTP/1.1

```

FIGURE 1: An example of an HTTP request line.



	 Wechat	 Tencent news
User 1	Get /netlog_se/Select?imei = HJS5T19626000575&startDate = 20200526&endDate = 20200527 HTTP	Get /reportInterest?devId = HJS5T19626000575&appver = 29_android_6.1.21 HTTP
User 2	Get /netlog_se/Select?imei = HJS5T19626000575&startDate = 20200528&endDate = 20200528 HTTP	Get /reportInterest?devId = HJS5T19626000575&appver = 29_android_6.1.21 HTTP
User 2	Get /netlog_se/Select?imei = 074e7fa44aife97a&startDate = 20200526&endDate = 20200528 HTTP	Get /reportInterest?devId = 074e7fa44aife97a&appver = 28_android_6.1.50 HTTP
User 3	Get /netlog_se/Select?imei = eeee88f0f289cb97&startDate = 20200621&endDate = 20200621 HTTP	Get /reportInterest?devId = eeee88f0f289cb97&appver = 28_android_6.1.20 HTTP

FIGURE 2: Examples of HTTP requests from three users in Wechat and Tencent News.

variable-level, method-level, message-level, and file-level. Achara et al. [9] present MobileappScrutinator, a dynamic analysis platform for Android and iOS. MobileappScrutinator rewrites the source code of Android OS to track personal data. Then similar functions are implemented in iOS with jailbreaking. PrivacyCapsules [24] implements a customized OS. It requires the apps running above it to comply with its PI access rules. At the same time, the PI access rules prevent the PI from leaving the personal device. iABC [25] evaluates the risks of privacy leaks of iOS apps by combining dynamic analysis with static analysis. With a customized OS, He et al. [26] hook the privacy-related APIs to identify privacy leaks of the third-party libraries inside apps. However, these methods usually run the apps with automatic tools, which leads to incomplete coverage of app execution paths [6]. Therefore, false negatives would be introduced. Meanwhile, those methods could generate false positives because of coarse-grain taint and tainted information explosion [23].

Another method is proposed based on differential analysis. AGRIGENTO [10] establishes the network behavior baseline of an app by running the app multiple times. Then the input value of the PI to the app is changed. Finally, privacy leaks are found by observing deviations in the resulting network traffic. In this way, AGRIGENTO also could detect obfuscated PI in the traffic. Although this method could achieve high precision, its practicability is relatively poor as a large number of manual operations are involved.

3.3. *Traffic Monitoring-Based Detection.* The traffic monitoring-based detection passively monitors all the traffic generated by apps; then it detects the PI transmitted through the traffic. The traffic is generally monitored by self-developed tools [11–13, 27, 28]. These tools utilize the VPNService provided in Android OS to collect the traffic without requiring root permission. Besides, since these tools are deployed on mobile devices, the source app of the traffic could be obtained simultaneously. Compared with the other two types of detection methods, the traffic monitoring-based detection method is lighter and easier to deploy. In addition, it could continuously monitor the traffic generated by apps; hence, full coverage of detection can be achieved.

With the self-developed tool Antmonitor [11], Anastasia et al. find the PI that is readily available to apps on the phone using simple string matching, such as IMEI, the email account, and the phone number. PrivacyGuard [27] adopts regexes to detect the PI with specific formats in the traffic. Similarly, Liu et al. [19] design regexes to find five kinds of PI in the traffic. Then the values of the discovered PI are added to the regex rules to mine more privacy leaks. Recon [12] firstly proposes a detection scheme based on machine-learning. It regards the traffic flows as short texts. Then the bag-of-words model is applied to extract the features of the traffic flows. Finally, a decision tree classifier is trained for each domain name to identify five types of PII. Anastasia et al. [13] present Antshield, a similar on-device privacy leak detection method. Besides finding the known PII based on

string matching, Antshield detects the unknown PII by machine-learning classifiers, which are similar to Recon. Compared with Recon, Antshield trains a classifier for each app instead of for each domain name.

A weakness of the traffic monitoring-based detection methods is that they cannot deal with encrypted traffic unless extra technology is adopted, such as the man-in-the-middle (MITM) proxy. However, the MITM proxy is not always allowed to be deployed or run successfully. In this case, such a method will no longer be the first choice under the analysis demand of encrypted traffic. On the contrary, it has advantages in scenarios where the traffic is not encrypted or the encrypted traffic can be decrypted. Additionally, the detection methods based on simple string/regex matching are not resilient to obfuscation technology. Lastly, the existing methods can only detect the PII with predefined types, while our method takes a step toward the detection of undefined PI.

4. Personal Information Features

In this section, the $\langle app, key \rangle$ pair, which is the basic detection object of our method, is defined in Section 4.1. Then local features and global features for a $\langle app, key \rangle$ pair are given in Section 4.2. Those features are used to depict the value characteristics of a $\langle app, key \rangle$ pair.

4.1. $\langle app, key \rangle$ Pair. As described in Section 2.4, we find that the value occurrence patterns of $\langle app, key \rangle$ pairs that transmit PI and non-PI are different. Besides, it has been observed that HTTP requests generated by the same action in an app are likely to have a similar structure [29]. More precisely, those HTTP requests have the same keys. Moreover, considering that the developer of one app usually adopts certain naming conventions to name keys, we further assume that a key in an app constantly transmits the same type of information. Therefore, the detection task is transformed into identifying whether each $\langle app, key \rangle$ pair is PI-related, key is a key used in the HTTP requests of the app . Once a pair $\langle app_i, key_j \rangle$ is PI-related, then a privacy leak occurs when key_j appears in an HTTP request of app_i , and the value of key_j is not empty or a default value.

4.2. Statistical Features of the $\langle app, key \rangle$ Pair. For clarity, the symbols used to define the proposed features are first given as follows. Suppose that there are n apps: app_1, \dots, app_n . For pair $\langle app_i, key_j \rangle$, $V_{ij} = \{v_{ij}^1, v_{ij}^2, \dots, v_{ij}^m\}$ is the values of key_j , and $\#(v_{ij}^t)$ calculates how many times v_{ij}^t appears. H_{ij} denotes the set of domain names that visited by the app_i 's HTTP requests containing key_j . V_{ij} and H_{ij} are regarded as the attributes of key_j . In following sections, 11 local features and 6 global features are designed to describe the value characteristics of $\langle app, key \rangle$ pairs. For pair $\langle app_i, key_j \rangle$, local features are extracted from the HTTP requests of app_i , while global features are extracted from the HTTP requests of all apps.

4.2.1. Local Features

(1) *Key-Value Statistics.* For some PI, such as device identifiers, their values for a user would be unique or remain unchanged for a period of time. In contrast, the values of non-PI may change frequently, such as the visited resource and timestamps. We take two indicators to describe this difference: the number of different values per user and the value entropy per user. For pair $\langle app_i, key_j \rangle$, these two indicators are first computed for each user. Since different $\langle app, key \rangle$ pairs may have a different number of users, eight statistical features are further extracted from the indicators as key-value statistics, including the max/min/ava/var values of the two indicators of all users. The key-value statistics indicate how the values of one user on key_j change in the HTTP requests of app_i .

(2) *Local-Value Reuse Degree (L-VRD).* L-VRD is the ratio of the values in V_{ij} used by at least two users. This feature measures the possibility of different users taking the same value.

(3) *Key Frequency.* Key frequency refers to the proportion of HTTP requests that contain key_j to all HTTP requests from the app_i , i.e., the frequency of app_i uses key_j to transfer the information. This feature describes the difference in the frequency at which PI and non-PI are collected by app_i .

(4) *Number of Users.* Lastly, to reduce the impact of the number of users on the above features, the number of users that contribute to the HTTP requests of app_i is added as one of the local features.

On the whole, local features focus on how the value of one key of an app changes among different users. It implies that these features would be affected by the number of users. In addition, the occurrence frequency of the key in the app is also taken into consideration.

4.2.2. Global Features. It is prone to introduce false predictions when the occurrence patterns are inferred only by local features. Take $\langle TencentNews, appver \rangle$ as an example. The $appver$ is the version of the app Tencent News installed on the device, which is not PI. However, $\langle TencentNews, appver \rangle$ could show similar local features as the PI-related pairs when the HTTP requests of Tencent News are collected from one user. Therefore, global features are further designed to characterize how key_j and its attributes distribute in the HTTP requests of all apps.

(1) *Key Reuse Degree (KRD).* KRD refers to the number of other apps that take key_j as one of their keys. Its calculation is defined in Equations (1) and (2). KRD indicates whether key_j is app-specific or is commonly employed among apps.

$$KRD = \sum_{k=1, k \neq i}^n f(key_j, app_k), \quad (1)$$

$$f(\text{key}_j, \text{app}_k) = \begin{cases} 1, & \text{if } \text{key}_j \text{ is one of the keys of } \text{app}_k. \\ 0, & \text{else.} \end{cases} \quad (2)$$

(2) *Domain Reuse Degree (DRD)*. DRD refers to the number of other apps that visit at least one domain name in H_{ij} , as shown in Equations (3) and (4). If H_{ij} is visited by multiple apps, it is possible that key_j is generated by a third-party library rather than by the function of app_i itself. DRD is designed to distinguish the patterns of how the third-party library collects information from those of the app itself.

$$\text{DRD} = \sum_{k=1, k \neq i}^n g(H_{ij}, \text{app}_k), \quad (3)$$

$$g(H_{ij}, \text{app}_k) = \begin{cases} 1, & \exists h \in H_{ij}, \text{app}_k \text{ visits } h. \\ 0, & \text{else.} \end{cases} \quad (4)$$

(3) *Weighted Value Distribution Features*. The weighted value distribution features capture the distribution of V_{ij} in the HTTP requests of other apps. To provide formal descriptions of these features, Value Distribution Matrix (VDM) is defined in our work. Figure 3 illustrates the VDM of pair $\langle \text{app}_i, \text{key}_j \rangle$, a matrix with a size of $m \times (n-1)$. Element C_{tk} ($1 \leq t \leq m, 1 \leq k \leq n, k \neq i$) is one kind of statistical data that relates to the occurrence of v_{ij}^t in the HTTP requests of app_k . Besides, $W_{ij} = \{w_{ij}^1, w_{ij}^2, \dots, w_{ij}^m\}$ is the weight vector of V_{ij} , and w_{ij}^t is the weight of v_{ij}^t , as computed in Equation (5).

$$w_{ij}^t = \frac{\#(v_{ij}^t)}{\sum_{t=1}^m \#(v_{ij}^t)}. \quad (5)$$

Based on four different C_{tk} , i.e., four different statistical values, four global features are designed as follows:

(i) *Weighted Global-Value Reuse Degree (Weighted GVRD)*. The weighted G-VRD is the weighted sum of the number of times each value in V_{ij} appears in other apps' HTTP requests. Its calculation is given in Equation (6), where C_{tk} is the number of times v_{ij}^t appears as a value in the HTTP requests of app_k .

$$\text{weighted G-VRD} = \sum_{t=1}^m \left(w_{ij}^t \times \sum_{k=1, k \neq i}^n C_{tk} \right) \quad (6)$$

(ii) *Weighted App Reuse Degree (Weighted ARD)*. The weighted ARD is similar to the weighted G-VRD, except for C_{tk} which indicates whether v_{ij}^t appears in the HTTP requests of app_k . Specifically, C_{tk} is 1

	app_1	app_2	\dots	app_k	\dots	app_{i-1}	app_{i+1}	\dots	app_n
V_{ij}^1	C_{11}	C_{12}	\dots	C_{1k}	\dots	C_{1i-1}	C_{1i+1}	\dots	C_{1n}
V_{ij}^2	C_{21}	C_{22}	\dots	C_{2k}	\dots	C_{2i-1}	C_{2i+1}	\dots	C_{2n}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
V_{ij}^t	C_{t1}	C_{t2}	\dots	C_{tk}	\dots	C_{ti-1}	C_{ti+1}	\dots	C_{tn}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
V_{ij}^m	C_{m1}	C_{m2}	\dots	C_{mk}	\dots	C_{mi-1}	C_{mi+1}	\dots	C_{mn}

FIGURE 3: The Value Distribution Matrix of $\langle \text{app}_i, \text{key}_j \rangle$.

when v_{ij}^t appears in the HTTP requests of app_k . Otherwise, C_{tk} is 0

(iii) *Weighted User Reuse Degree (Weighted URD)*. The weighted URD is the weighted sum of the number of users using v_{ij}^t in the HTTP requests of other apps. As Equation (7) shows, C_{tk} is the user set using value v_{ij}^t in the HTTP requests of app_k . More specific, $C_{tk} = [\text{user}_1, \text{user}_2, \dots]$ and each user in C_{tk} have used v_{ij}^t in his HTTP requests of app_k . When calculating the weighted URD, the union of each row of VDM is computed first. Then the length of a union can be obtained, which is computed as $|\bigcup_{k=1, k \neq i}^n C_{tk}|$ in Equation (7). Finally, the weighted URD is the weighted sum of the lengths of the unions

$$\text{weighted URD} = \sum_{t=1}^m \left(w_{ij}^t \times \left| \bigcup_{k=1, k \neq i}^n C_{tk} \right| \right) \quad (7)$$

(iv) *Weighted New User Reuse Degree (Weighted NURD)*. Compared with the weighted URD, the weighted NURD concerns new users. Suppose the users involved in pair $\langle \text{app}_i, \text{key}_j \rangle$ is denoted as U , the weighted NURD of pair $\langle \text{app}_i, \text{key}_j \rangle$ is calculated as Equation (8). The calculation of this feature is similar to the weighted URD. The difference is that after obtaining the union of each row of VDM, the users who have used key_j in app_i would be removed from the union. The weighted NURD indicates how many new users use the values in V_{ij} in the HTTP requests of other apps

$$\text{weighted NURD} = \sum_{t=1}^m \left(w_{ij}^t \times \left| \bigcup_{k=1, k \neq i}^n C_{tk} - U \right| \right) \quad (8)$$

It is worth mentioning that only KR D is related to the name string of the key itself among the six global features. Taking $\langle \text{Wechat}, \text{imei} \rangle$ in Figure 2 as an example, KR D calculates whether "imei" is also a key for other apps. The

rest five features describe the statistics of the values and the visited domain names of the key. For instance, for $\langle Wechat, imei \rangle$, weighted value distribution features capture the value characteristics of “HJS5T19626000575” (the value of “imei”) in other apps. Therefore, even if an app has unique naming rules or the application scenario changes (e.g., under different cultures), only KRD would be affected.

5. Detector Construction

With the designed statistical features, the PI detector is implemented as Figure 4 illustrates. Suppose that a group of traffic that is labeled with source apps has been collected. At the preprocessing stage, $\langle app, key \rangle$ pairs and their features are extracted from the traffic. To construct a labeled dataset, a set of string-form rules is applied to discover positive samples, i.e., the $\langle app, key \rangle$ pairs that transmit PI. Meanwhile, negative samples are labeled manually. Finally, a binary-classification detector is trained based on the machine-learning algorithm. The detector can be applied to identify whether the unlabeled samples are PI-related.

5.1. Preprocessing. In this stage, $\langle app, key \rangle$ pairs are extracted from the HTTP requests. Then the $\langle app, key \rangle$ pair is removed if all values of the key are default or empty. The default values are collected by examining a part of sampled traffic, including “none,” “unknown,” “-,” “[IMEI],” and “[MAC].”, etc. Besides, the $\langle app, key \rangle$ pair whose key appears only once in the HTTP requests is also removed since its data volume is too small to extract meaningful occurrence patterns. At last, the default values are deleted from the value set of $\langle app, key \rangle$ pairs; then 11 statistical features are computed for each valid $\langle app, key \rangle$ pair.

5.2. Dataset Preparation. For supervised machine-learning algorithms, a labeled dataset is required to train the detector. Since we have no prior knowledge of PI, the samples cannot be labeled based on the values of PI. In our work, string-form rules and manual labeling are employed to find a handful of positive samples and negative samples, respectively.

The string-form rules include a group of keywords and regexes for identifying six kinds of PI, as listed in Table 1. The keywords come from our previous work [30], which summarizes the commonly used keywords that transmit PI. In addition, regex rules are applied to find MAC addresses, email accounts, and phone numbers since they have special formats. For pair $\langle app_i, key_j \rangle$, if key_j is one of the keywords or at least one value of key_j satisfies the regexes, it is labeled as positive. Then, the values of the found positive samples can be exploited to discover more positive samples. Finally, the found positive samples are manually checked, and the unqualified samples are deleted. For different scenarios, the rules used for finding positive samples can be flexibly modified and expanded.

As for the negative samples, they are labeled manually. On the one hand, the negative samples lack prominent rules to be discovered automatically. On the other hand, the introduced labor work is acceptable since only a small number of negative samples need to be labeled. Furthermore, as the

majority of $\langle app, key \rangle$ pairs are non-PI-related in practice, negative samples can be labeled easily. Although the final labeled dataset has a small size, the experimental results in Section 6 show that it is enough to train an effective detector.

5.3. Detector. Finally, a detector is trained based on the machine-learning algorithm to predict whether a $\langle app, key \rangle$ pair transmits PI. As previously stated, the proposed detector can identify unknown types of privacy leaks. In other words, although our labeled dataset contains only six types of PI-related samples, the detector could identify the $\langle app, key \rangle$ pairs that transmit PI beyond those six types.

From the implementation of the detector, it can be seen that the above detection can be performed after a certain amount of traffic has been collected. However, the advantage of this detection model is that it could predict a large number of $\langle app, key \rangle$ pairs within one prediction. Then a blacklist can be built from the $\langle app, key \rangle$ pairs that are predicted as positive. After deploying the blacklist in the traffic monitoring apps similar to Recon [12], only simple string matching is required to find the privacy leaks in real-time traffic. While processing the real-time traffic, new $\langle app, key \rangle$ pairs and their data can be sent to a centralized server so that the detector and its prediction results can be updated periodically.

5.4. Deployment. Figure 5 details the deployment and the execution process of the proposed detector. There are two main subjects in the deployment environment. One is the personal device, on which the user installs various apps (e.g., app1 and app2 in Figure 5). The other is the detection tool, which detects whether the apps installed by the user are transmitting PI. The detection tool consists of a client and a server. The server deploys the proposed detector. The client is an app (the APP in Figure 5) that monitors the HTTP requests generated by the personal device. Meanwhile, it maintains a blacklist that contains the $\langle app, key \rangle$ pairs found by the detector for transmitting PI. After launching the monitoring app, it continuously monitors the HTTP requests of the personal device and checks whether a key in the blacklist appears in these HTTP requests based on string matching. If a match occurs, a privacy leak will be reported to the user. Meanwhile, the app continuously collects the necessary HTTP request information and sends it to the server for retraining the detector. In this way, the detector could be updated periodically; then the blacklist can be updated simultaneously.

6. Experiments

6.1. Dataset. In our work, a self-developed app Netlog [31] is utilized to collect the traffic of mobile devices. The working principle of Netlog is similar to the tool used in Recon [12]. Then 224 volunteers are recruited to participate in the traffic collection. At last, the collected traffic dataset covers 350 apps with a total of 5,374,633 HTTP requests. This dataset is planned to be made public in our future work.

After preprocessing, 16,968 valid $\langle app, key \rangle$ pairs are extracted from 335 apps. The cumulative distribution

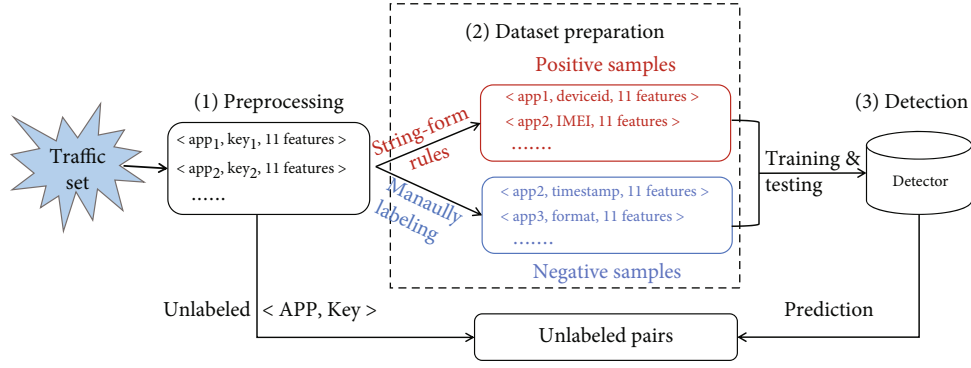


FIGURE 4: The implementation of the PI detector.

TABLE 1: The rules for positive samples discovering.

PI type	String-form rules
User identifier	user, userid, user_cid, user_id, and user-id
Device identifier	imei, meid, imsi, misi, deviceid, device_id, and serialnumber
MAC address	mac, mac_address, ((([a-f0-9A-F]{2}:){5}z) ((([a-f0-9A-F]{2}-){5})))[a-f0-9A-F]{2}
Location	location, gps, latlng, longitude, ltt, lat, latitude, lgt, lng, lon, and address
Email	[a-zA-Z0-9_+~] + @[a-zA-Z0-9-] + [a-zA-Z0-9-] +
Phone number	^(86)?(13[0-9] 14[5 7] 15[0-9] 166 17[3 6 7 8] 18[0-9] 19[1 8 9]) \ d{8}\$

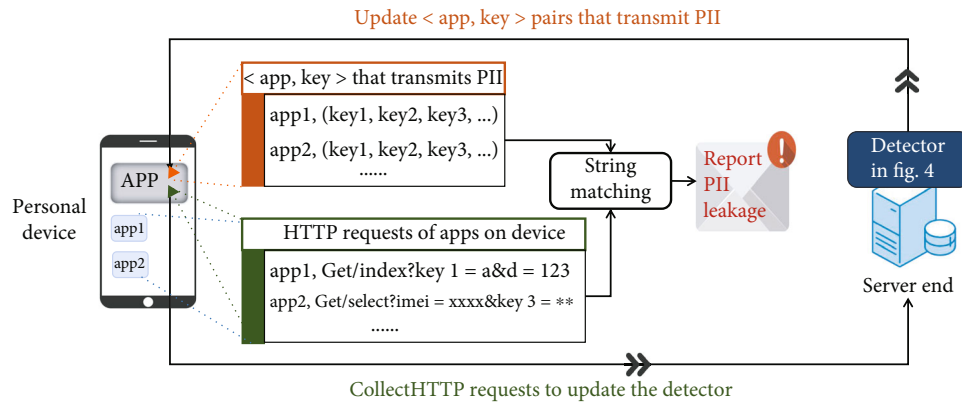


FIGURE 5: The deployment and the execution process of the detector.

function (CDF) of the number of users for $\langle app, key \rangle$ pairs is depicted in Figure 6. It can be seen from Figure 6 that more than 90% of the $\langle app, key \rangle$ pairs have less than five users. Meanwhile, 58.56% of the $\langle app, key \rangle$ pairs come from the traffic of only one user. It indicates that more than half of the $\langle app, key \rangle$ pairs do not have values from different users, and hence, they cannot obtain reliable local features. Consequently, the difficulty of correctly identifying such samples would increase. Subsequent experimental results further prove this viewpoint.

Using the labeling method introduced in Section 5.2, 404 positive samples and 404 negative samples are obtained from 16,968 samples. Those 808 samples come from 229 different apps. Similar to the distribution of the whole dataset, 59.6% of the 808 labeled samples have only one user, and 94.5% of

the samples have no more than four users. Table 2 provides the composition of the positive samples.

Several conclusions can be made from Table 2. First of all, the leakage of IMEI is the worst. About 20% of apps transmit IMEI in 10.35% of their HTTP requests. This should be taken seriously since IMEI can be directly used to identify an individual. Secondly, although 47 $\langle app, key \rangle$ pairs transmit UserID, which is less than those of the other deviceID and MAC, the leakage frequency of UserID exceeds that of the other deviceID and MAC. Therefore, it can be inferred that IMEI and UserID are more commonly utilized to identify the users of the app compared with other device identifiers. Besides, the leakage of the phone number shows a little difference. In our dataset, nine $\langle app, key \rangle$ pairs transmit phone numbers, while 49,453 leaks happen.

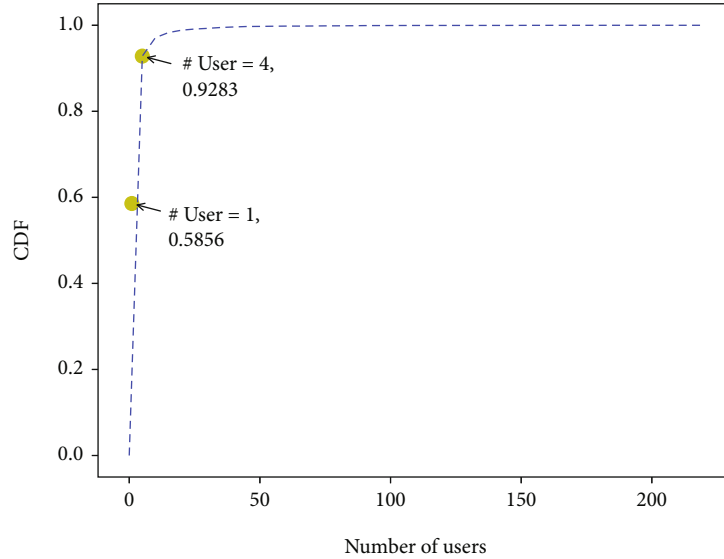
FIGURE 6: The CDF of the number of users for $\langle app, key \rangle$ pairs.

TABLE 2: The composition of positive samples.

Category	PI	No. of $\langle app, key \rangle$	No. of leaks	No. of apps
UserID-related	Email	2	438	2
	UserID	47	134,626	41
DeviceID-related	Phone number	9	49,453	7
	IMEI	124	556,389	67
	Other DeviceID	92	18,340	76
MAC	MAC	66	21,901	57
Location	Location	64	22,516	34
	Total	404	803,663	162

After analyzing these nine samples, we found that 44,561 privacy leaks occurred in the same app. What is more, this app is the official application of a telecom service provider in China. Lastly, only two $\langle app, key \rangle$ pairs are found to transmit email accounts. It seems that the email account has been well protected during its collection.

The positive samples in Table 2 are further divided into four categories: UserID-related, DeviceID-related, MAC, and Location. Then five datasets are obtained by combining different categories of samples, as listed in Table 3. $T_{allType}$ means that all positive samples are randomly split into training samples and test samples. The other four datasets use three categories of positive samples as the training samples and the remaining one as the test samples. For different datasets, negative samples with the same number of positive training samples are randomly selected to build the training set. The remaining negative samples are added to the positive test samples to form the test set.

6.2. Comparison of Machine-Learning Classifiers. To verify the impact of machine-learning algorithms on the performance of the detector, four machine-learning algorithms

are applied to train the detector, including Random Forest (RF), SVM with Gaussian kernel, KNN, and Gaussian Naive Bayes (NB). Three metrics in machine-learning are used to evaluate the performance of the detectors, including precision, recall, and accuracy. Dataset $T_{allType}$ is used in this experiment, and it is randomly split into a training set and a test set with a ratio of 8:2 for each experiment. The average performance of four detectors on 10 experiments is provided in Figure 7.

As illustrated in Figure 7, the detector based on Random Forest achieves the highest accuracy and precision, which are 87% and 84%, respectively. Besides, the recall of the Random Forest detector reaches 91%. The performance of KNN is slightly inferior to Random Forest. Although Naive Bayes realizes the highest recall, its precision and accuracy are far lower than those of the other three detectors. SVM shows similar performance as Naive Bayes.

Then, the detectors are evaluated on the other four datasets. The accuracy of the detectors on different datasets is shown in Figure 8. Overall, our detectors show effectiveness on these four datasets, which proves that it is capable of detecting the PI of unknown types. In terms of accuracy,

TABLE 3: The datasets obtained by different combinations of positive samples.

Datasets	Training	Test
T_allType	All categories	
T_DeviceID	UserID-related, MAC, Location	DeviceID-related
T_UserID	DeviceID-related, MAC, Location	UserID-related
T_MAC	DeviceID-related, UserID-related, Location	MAC
T_Location	DeviceID-related, UserID-related, MAC	Location

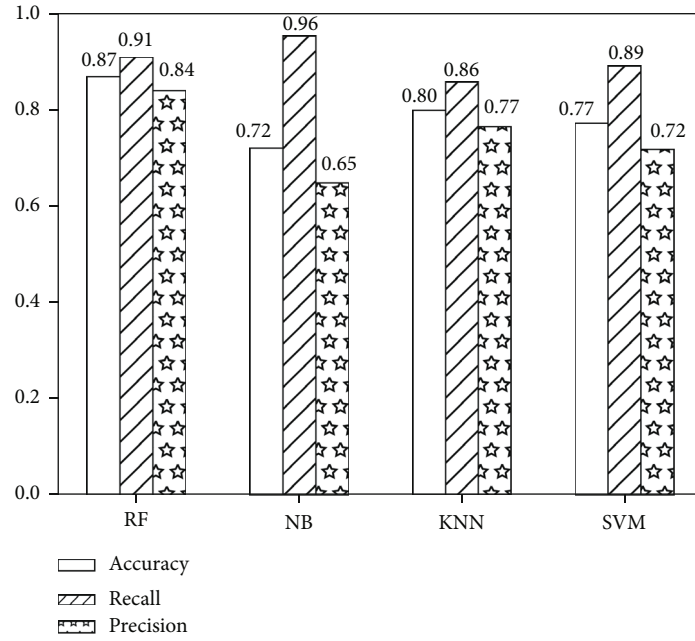


FIGURE 7: The performance of different detectors on T_allType.

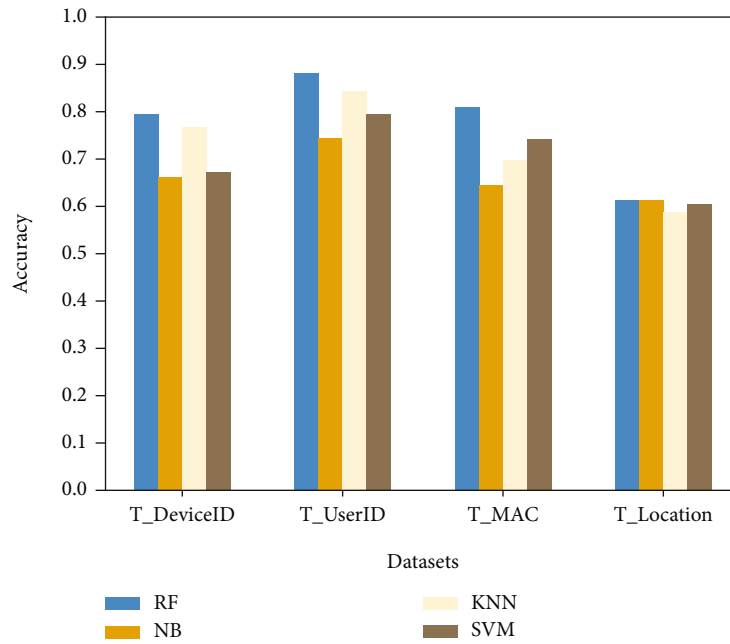


FIGURE 8: The accuracy of detectors on different datasets.

the RF detector also achieves the best performance on these four datasets. In addition, the performance of these detectors on different datasets presents differences. Among these four datasets, all detectors achieve the highest accuracy on the dataset T_UserID, with the accuracy of the RF detector as 88.23%. The best accuracy on T_DeviceID and T_MAC is slightly lower than that of T_UserID, which is around 80%.

In contrast, the accuracy on T_Location declines significantly, which is 61.29% at best. For T_Location, the detectors are trained by the samples from the categories UserID-related, DeviceID-related, and MAC. However, the features of these samples are quite different from that of Location. Therefore, it is difficult to recognize the location information with high accuracy. On the contrary, the performance on the other three datasets is satisfying because the PI of UserID-related, DeviceID-related, and MAC has similar occurrence patterns.

In view of the above results, the RF detector and the dataset T_allType are used in the subsequent experiments.

6.3. Effectiveness of the Proposed Features. The effectiveness of the local features and global features is analyzed in this section. The performance of the detector trained on different feature sets is shown in Table 4.

It can be seen from Table 4 that the detector based on global features shows slightly lower performance than the detector based on whole features, while the performance of the detector is the worst when only local features are used. A possible reason is that more than half of the samples involve one user only; thus, the local features themselves cannot distinguish PI from non-PI effectively. For instance, the keys that transmit IMEI and the package name are likely to have similar local features when the app traffic comes from one user. On the contrary, those two keys would have different L-VRD when there are multiple users.

To verify the above conjecture and illustrate how the number of users affects the performance of the detector, the distribution of the number of users for the false predictions in 10 experiments is depicted in Figure 9. The category “others” in Figure 9 includes 11 types of number of users, i.e., 5, 7, 8, 10, 11, 13, 14, 17, 39, 41, and 43.

As Figure 9 shows, most false predictions result from the samples with a small number of users. The false prediction rate reaches 16.95% and 13.53% when there are one or two users for a sample. With the increase in the number of users, the false predictions decrease significantly. Noted that there are no false predictions for the 52 samples within the category “others.” Besides, only 18 false predictions are generated out of 279 predictions when the predicted samples have more than two users. In this case, the false prediction rate is reduced to 6.45%. Therefore, it can be inferred that the more users exist for a sample, the more reliable the prediction result is.

6.4. Impact of the Number of Apps. For an $\langle \text{app}, \text{key} \rangle$ pair, its local features are computed from the data of the app itself, while its global features are influenced by the data of all apps in the app set. The impact of the number of apps in the app set on the detector is analyzed in this section.

TABLE 4: The performance of the detector trained on different feature sets.

Feature set	Precision	Recall	Accuracy
Local features	0.6718	0.7179	0.6870
Global features	0.8265	0.8817	0.8475
All features	0.8322	0.9177	0.8703

The 808 labeled samples are randomly divided into a fixed training set and a fixed test set firstly. Since these samples are selected from 229 apps, each of our app sets contains those 229 apps to ensure that the local features of each sample can be extracted. Then, the app set starts from a size of 230 and 10 apps are added to the app set at random each time until the app set size reaches 330. For each app set, the features of the labeled samples are extracted (note that only the global features change), and the RF detector is retrained. Figure 10 shows the performance of the RF detector on the test set for different app sets.

As shown in Figure 10, there is no significant change in the performance of the RF detector as the size of the app set increases. It indicates that our initial app set, which contains 230 apps, is sufficient to extract effective global features. Adding more apps is of little use to improve the effectiveness of global features. Therefore, it is feasible to choose a fixed number of apps for feature extraction when applying our detector in practice.

6.5. Analysis on False Prediction. This section further investigates the source of the false predictions of the detector. Then, prediction confidence is utilized to improve the prediction accuracy. In 10 experiments, the true labels of all predicted samples and false predicted samples are collected. The label distribution of false predictions is shown in Figure 11.

Several observations can be obtained from Figure 11. Firstly, our detector cannot identify the “email” samples effectively. One possible reason is that there are only two email samples in our dataset. Even worse, one sample involves one user, and another sample only has four valid values. Another observation is that “location” samples are more prone to be falsely predicted than the samples with other PI types. As mentioned earlier, location has different traits compared with other kinds of PI. For example, its value could change frequently when the user moves. Therefore, more features can be added to identify location information separately in future work, such as the data format. Finally, the false positive is the main source of our false prediction. In the prediction of 807 negative samples, 145 samples are classified as positive incorrectly. In contrast, 88 false negatives are generated in the prediction of 808 positive samples. However, the detector is expected to generate as few false positives as possible in many application scenarios. Therefore, a measure is taken in our work to improve the performance of the detector.

During the prediction, the RF detector can not only give the label but also provide the probability of its prediction.

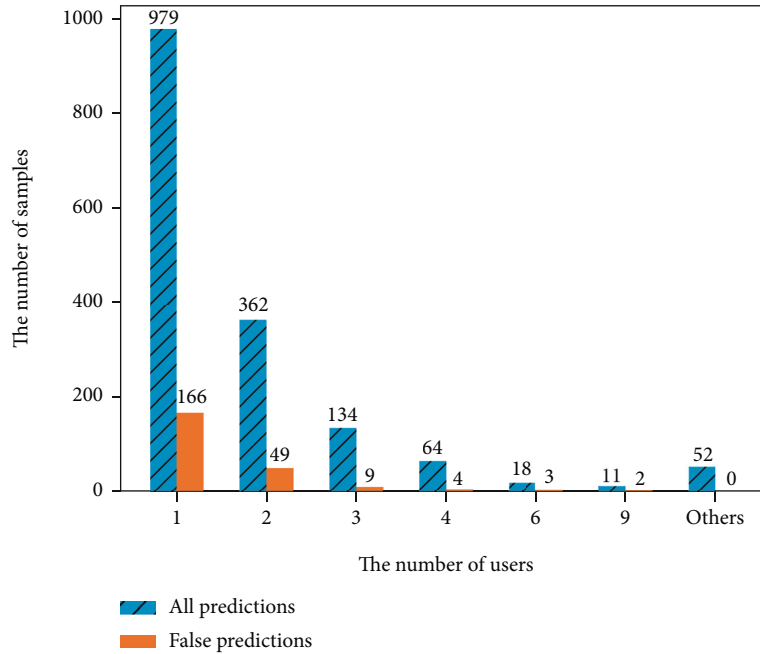


FIGURE 9: The distribution of the number of users for the false predictions.

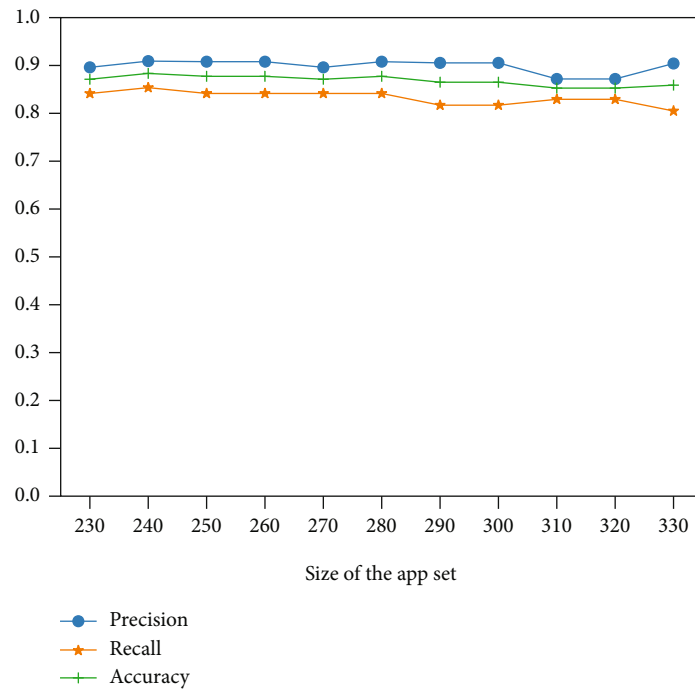


FIGURE 10: The performance of the detector on the app sets of different sizes.

Figure 12 illustrates the prediction probability of one random test.

As Figure 12 shows, the correct prediction usually has a high prediction probability. By contrast, the probability of a false prediction is likely to be below 0.75. Therefore, prediction confidence can be utilized to reduce false predictions. When the prediction probability is lower than the confidence, the detector would reject to

give a prediction. Although this measure would decrease the recall of the detector, the precision can be improved significantly in practice. For example, when the confidence is set as 0.75, 7 of the original 10 false positives are rejected, and the precision of positive samples would be improved to 95.58%. In the meantime, the recall of positive samples is 74.71%. In total, 79.01% of samples can be predicted.

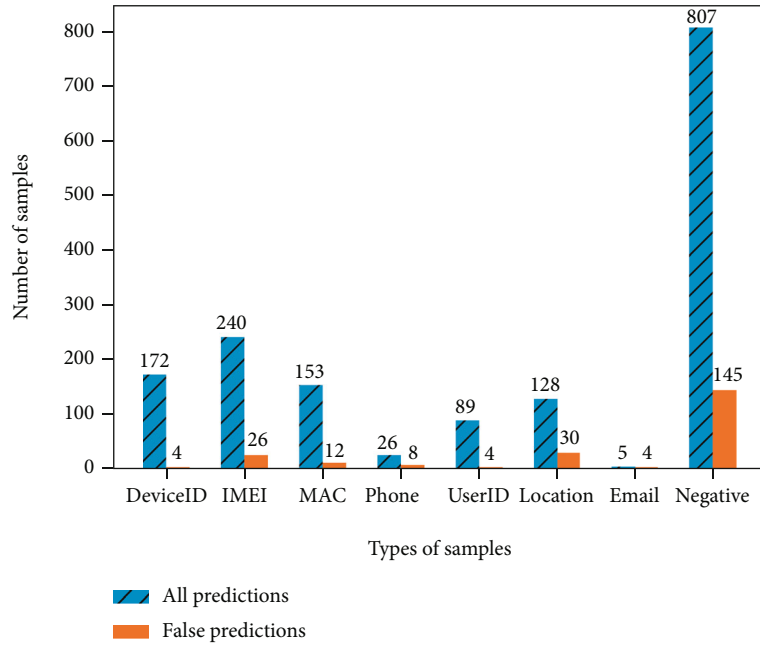


FIGURE 11: The label distribution of false predictions.

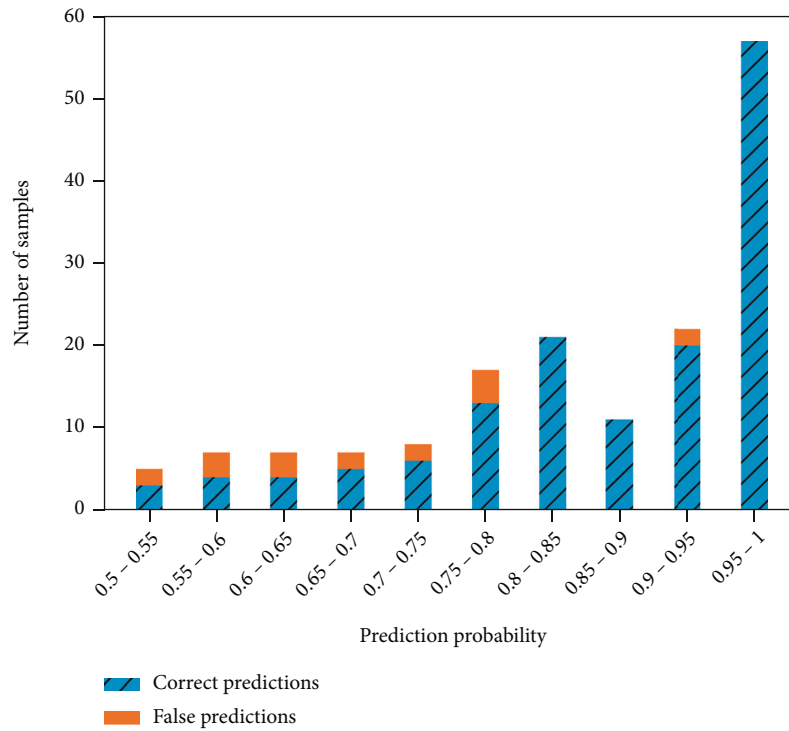


FIGURE 12: The distribution of prediction probability.

6.6. Prediction on Unlabeled Samples. Based on the RF detector and the confidence of 0.75, the remaining 16,160 unlabeled samples in our dataset are classified. Finally, 909 samples are identified as PI-related. We manually check those samples to validate their predictions.

Since the value of PI is unknown and even has been obfuscated, the predictions cannot be verified objectively.

Under this circumstance, conservative evaluation strategies are applied. The strategies are set as follows: (1) When the information transmitted by the $\langle app, key \rangle$ pair is evidently not PI, such as URLs or domain names, then a false positive is generated. Contrarily, the prediction is correct for the $\langle app, key \rangle$ pair with PI-related values or its key has definite PI-related semantics. (2) If semantic information cannot be

inferred from the $\langle app, key \rangle$ pair, the prediction is regarded as a false positive when the key carries different values for one user. Otherwise, the prediction of this $\langle app, key \rangle$ pair is uncertain. Strategy (2) is conservative since the PI that is similar to the location would be misjudged. Another example is the case where a mobile device has two identifiers, such as two IMEIs. In this case, strategy (2) would also misjudge the prediction. Therefore, the following analysis would give a lower bound of the performance of our detector. According to the above strategies, the predictions of the 909 samples are listed in Table 5.

As shown in Table 5, the samples identified are far beyond the six types of PII in the labeled dataset. Among these privacy leaks, device information and AndroidID are the most frequently leaked information. The device information includes the phone model, the phone brand, and CPU information. In addition to these common PI, it can be seen that some user behavior-related PI is also collected by apps, such as the app install time and the market which the app is downloaded from. Note that those collections could happen without user consent and awareness, and our detector could assist in detecting such privacy leaks.

Besides, several interesting points are also found during our manually checking process. Firstly, for several $\langle app, key \rangle$ pairs, their keys contain the string “imei_md5,” and their values are unique to each user. Therefore, it can be inferred reasonably that they are used to transmit the obfuscated IMEI. As for the 443 uncertain samples, all of their values appear to be obfuscated strings, and most $\langle app, key \rangle$ pairs have the string “**id” in the name string of their keys. Hence, we speculate that these keys are likely to transmit user-related identifiers.

Lastly, 270 samples are judged as false positives. Among those samples, 142 samples have a unique value for each user, and 134 samples out of those 142 samples have only one user. We believe some of those false positives can be eliminated after the traffic of more users is added. The rest 128 false positives are judged by the strategy (2). As stated earlier, those judgments are conservative. Moreover, we do find that some $\langle app, key \rangle$ pairs have at most two values for each user.

In conclusion, the above analysis demonstrates that the proposed detector can automatically identify privacy leaks in the traffic, including the unknown types of privacy leaks and the privacy leaks with obfuscation.

6.7. Comparison with Other Works. In this section, the proposed detector is compared with three existing traffic monitoring-based detection methods based on our labeled dataset. The difference between those three works and ours is summarized in Table 6. Compared with the existing three methods, the proposed detector can detect the PI with obfuscation and unknown types. In terms of the number of classifiers, our method builds one general classifier, while Recon and Antshield train a classifier for each domain name or each app. Besides, with one prediction, our method can identify the privacy leaks in the HTTP requests through simple string matching. However, Recon and Antshield have to classify each HTTP request by the classifiers. Therefore, our

TABLE 5: The details of the 909 samples with the positive prediction.

Type	No. of $\langle app, key \rangle$	No. of apps
AndroidID	30	27
Device information	104	70
IMEI	13	13
IP	16	16
Username	1	1
App install time	1	1
Location	3	3
ISP	2	2
Screen size	22	20
Serial number	2	2
App download market	2	2
Uncertain	443	/
Potential false positives	142	/
	128	/

method would introduce less time overhead compared with them, which has been verified in the following experiment.

Table 7 gives a detailed comparison of those methods in terms of detection capability and time overhead. It can be seen from Table 7 that the simplest method, i.e., the Seeded approach, has the worst detection ability. The Seeded approach only identifies 3 positive samples correctly. Meanwhile, it generates 26 false positives. Recon achieves the best detection capability among the existing three methods. Compared with Recon, our detector has the same recall with it. However, our detector achieves higher precision than Recon. The false positives generated by our detector are 20 less than that generated by Recon. As for the time overhead, the training time and prediction time consumed by our model are far less than those of Recon and Antshield. For the Seeded approach, the time overhead of it is close to ours since its detection model is also implemented based on string matching. Overall, the performance of the proposed detector in this paper is better than the existing methods.

6.8. Discussion

6.8.1. Limitation. The proposed detector may show limitations when it is directly applied to the following scenarios:

- (1) Where app developers adopt special naming rules to name their keys

When the naming of the key shows new characteristics (e.g., the key is unique), the corresponding $\langle app, key \rangle$ pairs would present new distributions in their KRD. Note that other features are not affected as their calculations are irrelevant to the naming of the key. If the training samples of the detector do not have $\langle app, key \rangle$ pairs with similar distributions, the trained detector would not identify such samples effectively.

TABLE 6: The difference between the existing methods and ours.

Work	Features	Model	SSL	Obfuscation	Unknown types of PI	No. of classifiers
Seeded approach [19]	String pattern	String matching	N	N	N	/
Recon [12]	Bag-of-words	Machine-learning	Y	N	N	10 ³
Antshield [13]	Bag-of-words	Machine-learning	Y	N	N	10 ²
Our method	Statistical features	Machine-learning	N*	Y	Y	1

*As Recon and Antshield did, our method could integrate MITM to deal with encrypted traffic in future work.

TABLE 7: The performance comparison with other works.

Work	False positive	False negative	Precision	Recall	F1 -measure	Training time (s)	Prediction time per 1000 samples (s)
Recon	28	12	71.13%	85.18%	77.52%	878.85	0.4567
Antshield	30	19	67.39%	76.54%	71.67%	788.62	0.4488
Seeded approach	26	78	/	/	/	/	0.079
Our method	8	12	89.61%	85.18%	87.34%	0.034	0.0129

Take the password as an example. Although the password is highly sensitive personal information, occasional disclosure of it is still be observed [32, 33]. We use keywords such as “password” and “pwd” to match the $\langle app, key \rangle$ pairs contained in our dataset, and one matching $\langle QuanminTVLive, password \rangle$ is found. It means that the key “password” is only used by QuanminTVLive in our dataset; hence, it can be regarded as a unique key. We use the detector trained in Section 6.2 to classify this sample, and the detector cannot recognize it transmits PII. This result is confusing since the password should have similar occurrence patterns with PII such as user ID intuitively. For such PII, the values of a user are unchanged for a period of time, and different users own different values. Therefore, we checked the difference between the features of the UserID-related samples in our training set and those of $\langle QuanminTVLive, password \rangle$. We found that the main difference is in KRD. Since “password” is only used in QuanminTVLive, its KRD is 0. However, none of the KRD of UserID-related samples in our training set is 0. Consequently, the detector fails to learn such patterns of KRD. After modifying the KRD of $\langle QuanminTVLive, password \rangle$ to other values that are not 0, our detector can correctly identify it. In addition, $\langle QuanminTVLive, password \rangle$ can be identified correctly at the cost of reducing the average accuracy of the detector by 2.53% when we remove KRD from the feature set and retrain the detector.

Therefore, it can be inferred that the effectiveness of the proposed detector is closely related to the training samples, which is explicable since the machine-learning algorithm can only learn the experience from the training samples. The performance of the detector would decline when the samples to be identified do not conform to the distribution of training samples. To alleviate this problem, we recommend that the detector should be periodically updated with the latest sample data and recollect samples and train the detector when the apps in the detection scene change significantly.

(2) Where there are severe imbalance problems in the traffic data

(i) Unbalanced PI and non-PI

In practical mobile network scenarios containing a large number of apps, there are usually far more keys transmitting non-PI in the traffic than those transmitting PI. It may result in nonnegligible false positives since our experiments show that the main source of misclassification of our detector is the false positive. However, a reasonable confidence threshold could alleviate this problem as shown in Section 6.5.

(ii) Unbalanced users

The local features proposed in our method are closely related to the number of users. The experiments in Section 6.3 show that the false prediction rate of our detector is higher when there are less than three users for an $\langle app, key \rangle$ pair. Therefore, we suggest that the prediction result of the detector should be treated with caution when few users are discovered for an $\langle app, key \rangle$ pair. A conservative strategy is to make the prediction for an $\langle app, key \rangle$ pair until it has accumulated enough different users.

(iii) Unbalanced values

Some personal data has obvious occurrence patterns of PI, while they may have a particular frequency of occurrence. For example, some PI is obtained by apps with a low frequency (many apps support remembering the accounts and passwords of users for a long time). Hence, the observation of such information in the traffic would be extremely rare. For our detector, it is unreliable to infer whether the information is PI or not when only a few observed values are used to extract features. However, long-term data statistics not only reduce the timeliness of

the prediction but also introduce additional storage overheads. Therefore, the identification of such PI in practical scenarios still needs further research.

6.8.2. Malicious Usage. In this paper, the detector we propose is intended to help users to understand which app transmits their personal information at what time. However, there are two sides of a medal, and the effectiveness of our detector also indicates that users' personal information can be stolen by monitoring the traffic of personal devices. Furthermore, the implementation of our detector is simple as it only requires the traffic data of users. This makes it possible to be maliciously used. For example, for an attacker, as long as he has the ability to eavesdropping the traffic of users' devices, the proposed detector can be employed to discover the potential personal information. Then that personal information can be exploited to analyze and profile users, so as to implement malicious activities such as fraud. To prevent such malicious usage, a powerful strategy is to prevent attackers from eavesdropping on the traffic. Some countermeasures can be taken, such as strengthening the authentication mechanism [34, 35].

7. Conclusion

To discover privacy leaks in mobile traffic, a novel detection method based on traffic monitoring is presented in this paper. The proposed method utilizes statistical features to capture the occurrence patterns of personal information in the traffic. Based on machine-learning, the detector could identify more personal information in the traffic. Compared with the existing methods, the proposed method can identify unknown types of personal information. Besides, the proposed method is resistant to obfuscation technology. Finally, the experimental results show that the proposed method could achieve better performance than the existing methods.

Data Availability

The personal information data used in this paper is not made public due to privacy issues, while the original traffic dataset used in this paper is being considered for publication after necessary anonymization processing.

Disclosure

An earlier version of this paper has been presented as a preprint (<https://arxiv.org/abs/2112.12346>).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61972410) and the Science and Technology Innovation Program of Hunan Province (2020RC2047).

References

- [1] Number of Mobile app Downloads in 2021/2022, "Statistics, current trends, and predictions," <https://financesonline.com/number-of-mobile-app-downloads/>.
- [2] T. Chadza, F. Aparicio-Navarro, K. Kyriakopoulou, and J. Chambers, "A look into the information your smartphone leaks," in *2017 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, Marrakech, Morocco, 2017.
- [3] Y. He, X. Zhao, and C. Wang, "Privacy mining of large-scale mobile usage data," in *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, pp. 81–86, Shenyang, China, 2019.
- [4] D. Patterson, "Facebook data privacy scandal: a cheat sheet," 2020, <https://www.techrepublic.com/article/facebook-data-privacy-scandal-a-cheat-sheet/>.
- [5] "Rela, a Chinese lesbian dating app, exposed 5 million user profiles," <https://techcrunch.com/2019/03/27/rela-data-exposed/>.
- [6] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez, "Bug fixes, improvements, ...and privacy leak, a longitudinal study of PII leaks across Android app version," in *The 25th Annual Network and Distributed System Security Symposium (NDSS 2018)*, IMDEA Networks, 2018.
- [7] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, *PiOS: detecting privacy leaks in iOS applications*, NDSS, 2011.
- [8] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale," in *International Conference on Trust and Trustworthy Computing*, S. Katzenbeisser, E. Weippl, L. J. Camp, M. Volkamer, M. Reiter, and X. Zhang, Eds., vol. 7344 of Lecture Notes in Computer Science, pp. 291–307, Springer, Berlin, Heidelberg, 2012.
- [9] J. P. Acharya, V. Roca, C. Castelluccia, and A. Francillon, "MobileappScrutinator: a simple yet efficient dynamic analysis approach for detecting privacy leaks across mobile OSs," <https://arxiv.org/abs/1605.08357>.
- [10] A. Continella, Y. Fratantonio, M. Lindorfer et al., "Obfuscation-resilient privacy leak detection for mobile apps through differential analysis," *NDSS*, vol. 17, 2017.
- [11] A. Shuba, A. Le, M. Gjoka, J. Varmarken, S. Langhoff, and A. Markopoulou, "AntMonitor: network traffic monitoring and real-time prevention of privacy leaks in mobile devices," in *Proceedings of the 2015 Workshop on Wireless of the Students, by the Students, & for the Students*, pp. 25–27, Paris, France, 2015.
- [12] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Chones, "ReCon: revealing and controlling privacy leaks in mobile network traffic," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 361–374, Singapore, Singapore, 2016.
- [13] A. Shuba, E. Bakopoulou, M. A. Mehrabadi, H. Le, D. Choffnes, and A. Markopoulou, "AntShield: on-device detection of personal information exposure," <https://arxiv.org/abs/1803.01261>.
- [14] T. Zhang and Y. Shunzheng, "Research prospects of user information detection from encrypted traffic of mobile devices," *Journal on Communications*, vol. 42, no. 2, pp. 154–167, 2021.
- [15] S. Zhao, S. Chen, and Z. Wei, "Statistical feature-based personal information detection in mobile network traffic," <https://arxiv.org/abs/2112.12346>.

- [16] The Standardization Administration of China, *Information Security Technology - Personal Information Security Specification*, The Chinese National Standard GB/T, 2020.
- [17] X. Wang, A. Continella, Y. Yang, Y. He, and S. Zhu, "LeakDoctor: toward automatically diagnosing privacy leaks in mobile applications," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 1, pp. 1–25, 2019.
- [18] Y. Zhao, L. He, Z. Li et al., "Large-scale detection of privacy leaks for BAT browsers extensions in China," in *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 57–64, Guilin, China, 2019.
- [19] Y. Liu, H. H. Song, I. Bermudez, A. Mislove, M. Baldi, and A. Tongaonkar, "Identifying personal information in internet traffic," in *Proceedings of the 2015 ACM on Conference on Online Social Networks*, pp. 59–70, Palo Alto, California, USA, 2015.
- [20] S. Arzt, S. Rasthofer, C. Fritz et al., "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *Acm Sigplan Notices*, vol. 49, pp. 259–269, 2014.
- [21] N. Wongwiwatchai, P. Pongkham, and K. Sripanidkulchai, "Comprehensive detection of vulnerable personal information leaks in android applications," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 121–126, Toronto, ON, Canada, 2020.
- [22] Y. Nan, Z. Yang, X. Wang, Y. Zhang, D. Zhu, and M. Yang, "Finding clues for your secrets: semantics-driven, learning-based privacy discovery in mobile apps," in *Network and Distributed Systems Security (NDSS)*, pp. 1–15, San Diego, CA, USA, 2018.
- [23] W. Enck, P. Gilbert, B.-G. Chun et al., "TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones," *Communications of the ACM*, vol. 57, no. 3, pp. 99–106, 2014.
- [24] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee, "Privacy capsules: preventing information leaks by mobile apps," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 399–411, Singapore, Singapore, 2016.
- [25] A. J. Bhatt, C. Gupta, and S. Mittal, "iABC: towards a hybrid framework for analyzing and classifying behaviour of iOS applications using static and dynamic analysis," *Journal of Information Security and Applications*, vol. 41, pp. 144–158, 2018.
- [26] Y. He, H. Binghui, and Z. Han, "Dynamic privacy leakage analysis of android third-party libraries," in *International Conference on Data Intelligence and Security*, pp. 275–280, South Padre Island, TX, USA, 2018.
- [27] Y. Song and U. Hengartner, "PrivacyGuard: a VPN-based platform to detect information leakage on android devices," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp. 15–26, Denver, Colorado, USA, 2015.
- [28] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan et al., "Haystack: a multi-purpose mobile vantage point in user space," <https://arxiv.org/abs/1510.01419>.
- [29] L. Shen, X. Wang, and S. Chen, "Structural signature extraction method for mobile application recognition," *Journal of Computer Applications*, vol. 40, no. 4, pp. 1109–1114, 2020.
- [30] S. Chen, S. Zhao, B. Han, and X. Wang, "Investigating and revealing privacy leaks in mobile application traffic," in *2019 Wireless Days (WD)*, pp. 1–4, Manchester, UK, 2019.
- [31] X. Wang, S. Chen, and S. Jinshu, "Automatic mobile app identification from encrypted traffic with hybrid neural networks," *IEEE Access*, vol. 8, pp. 182065–182077, 2020.
- [32] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: an underestimated threat," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1242–1254, Vienna, Austria, 2016.
- [33] S. Hui, Z. Wang, X. Hou et al., "Systematically quantifying IoT privacy leakage in mobile networks," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7115–7125, 2021.
- [34] Q. Wang, D. Wang, C. Cheng, and D. Hex, "Quantum2FA: efficient quantum-resistant two-factor authentication scheme for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, 2021.
- [35] L. Yanrong, D. Wang, M. S. Obaidat, and P. Vijayakumar, "Edge-assisted intelligent device authentication in cyber-physical systems," *IEEE Internet of Things Journal*, 2022.