SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2020, Vol. 63 132104:1–132104:17
https://doi.org/10.1007/s11432-018-9839-6

Static tainting extraction approach based on information flow graph for personally identifiable information

Yi LIU[1,2], Lejian LIAO[1] & Tian SONG[1*]

[1]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing* 100081*, China;*
[2]*College of Mathematics and Computer Science, Yan'an University, Yan'an* 716000*, China*

Received 26 September 2018/Revised 19 January 2019/Accepted 22 March 2019/Published online 10 February 2020

Abstract Personally identifiable information (PII) is widely used for many aspects such as network privacy leak detection, network forensics, and user portraits. Internet service providers (ISPs) and administrators are usually concerned with whether PII has been extracted during the network transmission process. However, most studies have focused on the extractions occurring on the client side and server side. This study proposes a static tainting extraction approach that automatically extracts PII from large-scale network traffic without requiring any manual work and feedback on the ISP-level network traffic. The proposed approach does not deploy any additional applications on the client side. The information flow graph is drawn via a tainting process that involves two steps: inter-domain routing and intra-domain infection that contains a constraint function (CF) to limit the "over-tainting". Compared with the existing semantic-based approach that uses network traffic from the ISP, the proposed approach performs better, with 92.37% precision and 94.04% recall. Furthermore, three methods that reduce the computing time and the memory overhead are presented herein. The number of rounds is reduced to 0.0883%, and the execution time overhead is reduced to 0.0153% of the original approach.

Keywords personally identifiable information, network privacy leak detection, static tainting, network traffic analysis, information flow graph, inter-domain routing, intra-domain infection, constraint function

Citation Liu Y, Liao L J, Song T. Static tainting extraction approach based on information flow graph for personally identifiable information. Sci China Inf Sci, 2020, 63(3): 132104, https://doi.org/10.1007/s11432-018-9839-6

1 Introduction

Over the last decade, various network applications have been used to improve the quality of life. To provide high-quality services, software applications usually collect personally identifiable information (PII) from their users. In particular, mobile applications collect PII using various sensors built into mobile devices. PII can be used to identify, distinguish, and track individuals. PII may comprise a single piece of information (direct information) or a group of information (indirect information) [1, 2], which may be derived from the device-identifiable information, user input information, local sensor data, or APP logs, such as the international mobile equipment identity (IMEI), telephone number, location, user name (or ID), password. This PII is used to either create a portrait of the network users by the application service provider (ASP) or to trace the footprints of a criminal suspect in cyberspace by the administrator or the security department.

Users' privacy information is directly accessed or indirectly inferred from this PII. ASPs already have the ability to access such information; however, application services do not always appropriately access

* Corresponding author (email: songtian@bit.edu.cn)

and use users' personal information. Certain applications sometimes excessively collect PII; ASPs usually provide users with free or inexpensive services in exchange for access to their information. A more serious offense is the trading of personal information to third parties by a few malicious applications without the users' permission. However, users are usually not mostly notified that their PII has been surreptitiously collected by malicious applications [3, 4]. Therefore, users cannot inspect the information collected by the application and prevent privacy leaks.

Previous studies have proposed numerous approaches to detect whether a given APP leaks private information to the network. Both static taint analysis and dynamic taint analysis [5] using an APP's bytecode rely on symbolic execution [6] and/or control flow graphs [7–9]. While these approaches provide useful insights, they suffer from several limitations when APPs try to wrongfully hide the fact that they are leaking private information. For instance, some code constructs can be deliberately added to break the control flow graphs and to lose the tainted input values [10, 11]. These approaches do not effectively prevent PII transmission to the network because static analysis does not perform real-time detection of privacy leaks and dynamic analysis requires heavy instrumentation. Recently, an alternative has been proposed in which PII is identified on the network layer through traffic interception [3, 12–15].

Network traffic analysis approach focuses on identifying PII on the IP traffic [12–15]. This approach can be easily deployed to clients for real-time analysis and detection because it works across platforms without requiring custom mobile OSes or access to the APP source code [16]. Meanwhile, some data that are encrypted and obfuscated in the massive network traffic also interfere with the identification reliability. However, previous studies have used simple string matching methods with regular expressions, lexicon lists, and keyword semantics to identify the possible PII values [3]. In these approaches, PII requires a stable format that can be distinguished from other values, which can either be expressed by regular expressions (e.g., email) [17] or be summarized in a limited lexicon (e.g., city). Alternatively, keyword semantics that are suggestive of PII (e.g., phone) can be used to identify the values as a PII. Many other PII categories cannot be expressed using regular expressions or listed using dictionaries; keyword semantics is sometimes confused by the obfuscation data in real-time traffic [16]. The "over-tainting" results in additional false positives (FPs) and/or false negatives (FNs). Furthermore, recent studies [13] use a machine-learning approach to infer the probability of PII that includes the user input. However, the training machine learning classifier is limited by the training data, which is usually labeled via manual analysis and using feedback obtained from crowdsourcing [18].

The proposed approach has been inspired by the taint detection method [4]. To trace PII according to the given taint type, the taint detection method mainly obtains information flow by relying on the function call relations between the components and processes in the APP code. First, the proposed approach filters the network traffic into three dimensions: services, keys, and values. Second, an information flow graph is constructed based on the user's access behavior and a graph is drawn using a tainting process comprising two steps: intra-domain infection and inter-domain routing. Then, a constraint function (CF) is used in the intra-domain infection against obfuscation data in real-time traffic. This can help reduce the FP caused by "over-tainting" in the intra-domain infection process and can speed up the algorithm convergence rate. Finally, a tainting-value is tracked as the input in the information flow graph to automatically and accurately find PII and the location of its leak.

This study offers three main contributions. (1) Based on the user's network access behavior pattern, the proposed approach filters the network traffic into three dimensions: service, key, and value. This filtering process greatly reduces the scale of the sample spaces. (2) According to the tainting-value, the proposed approach automatically and accurately extracts PII in the same category as the tainting-value; this reduces the FNs caused by "over-tainting". The proposed approach can also label the training data for machine learning without any manual labeling and feedback from crowdsourcing in large-scale network traffic. (3) Three methods are presented to improve the performance of the proposed approach. These methods reduce the memory overhead and the time overhead in a step-by-step manner. Results indicate that categories of some obfuscation information and encrypted information can be inferred from the plain text related to the taint relation.

The remainder of this paper is organized as follows. Related studies are divided into three categories

in Section 2. Section 3 describes three processes: the information flow graph, inter-domain routing, and intra-domain infection. Section 4 presents experimental and evaluation results as well as discussion. Section 5 optimizes the proposed approach in three steps, and Section 6 concludes the study.

2 Related work

Many related studies usually focus on identifying PII using one or more of the following complementary approaches. These studies can be classified into three categories: static analysis, dynamic analysis, and network traffic analysis. Static and dynamic analyses investigate PII from the APPs generated on the mobile clients, and network traffic analysis investigates PII at the network level.

Static analysis uses symbolic execution [5] and/or control flow graphs [6–8] to analyze the source code of an APP. This analysis has been leveraged to audit third-party libraries [19, 20], to inspect the APP permissions and their associated system calls [7, 21], and to analyze APP usage [22–24]. Static analysis is applicable to large-scale APP analysis without overheads such as running overheads or interactions with APPs. However, static analysis is rarely or never executed because approximately 30% of the dynamically loaded code in the benign APPs [25] cannot be found via static analysis.

Dynamic analysis identifies PII in the information flow created by system calls and accesses them during runtime. PII is flagged from the memory using the tainting-value as input by modifying the device OS (TaintDroid [4]), platform libraries (Phosphor [26, 27]), or APP under analysis (Uranine [27]) [16]. TaintDroid [4] detects privacy leakage from Android applications and uses variable-level tracking within the VM interpreter. TaintDroid does not track taints for native codes and applies a heuristic that taints from the input information to the return value of functions. AppFence [28] has improved TaintDroid in such a way that it detects sensitive information from the encrypted data and formats the data to make it unreadable by humans. TaintEraser [29] identifies whether an application leaks sensitive data, such as passwords and credit card numbers in Windows, but this tool requires users to manually specify the password or the credit card number. PIITracker [30] is another novel tool for tracking PII in Windows by leveraging the synergy of the whole-system taint analysis and monitoring specific function and system calls. In addition, the tool "UI monkeys" [31] is typically used for automated random exploration and synthetic user actions. This tool has a structured approach [32, 33] and does not require manual interactions [13].

Network traffic analysis explores PII at the network level by relying on network protocol analysis [34]. This approach can be easily deployed to clients for real-time analysis and detection because it works across platforms without requiring custom mobile OSes or access to APP source codes [16]. However, the encrypted and/or obfuscated data in the network traffic also interfere with identification reliability in network traffic analysis. To support the transport layer security (TLS) inspection, network traffic analysis usually captures the traffic from the router interface using a virtual private network (VPN) tunnel, which intercepts the TLS protocol and converts the network traffic analysis into plaintext traffic analysis. ReCon [13] is a typical VPN-based approach that uses a machine-learning approach to infer a broader range of PII that includes user inputs. However, ReCon has two problems: it requires the deployment of additional applications on the client and needs a training dataset that is labeled using manual tests and feedback. The proposed approach overcomes these problems by automatically and accurately finding and extracting PII against the manual label with large-scale data and by obtaining the ground truth of the dataset for the training data for machine learning. Moreover, the proposed approach can capture network traffic and analyze it in the ISP-level network, which is a complex environment that cannot sequentially deploy additional applications on the large-scale clients.

3 Static tainting extraction approach

Static tainting extraction approach comprises three processes: network traffic collection, network traffic transformation, and static tainting extraction algorithm, as shown in Figure 1. The network traffic

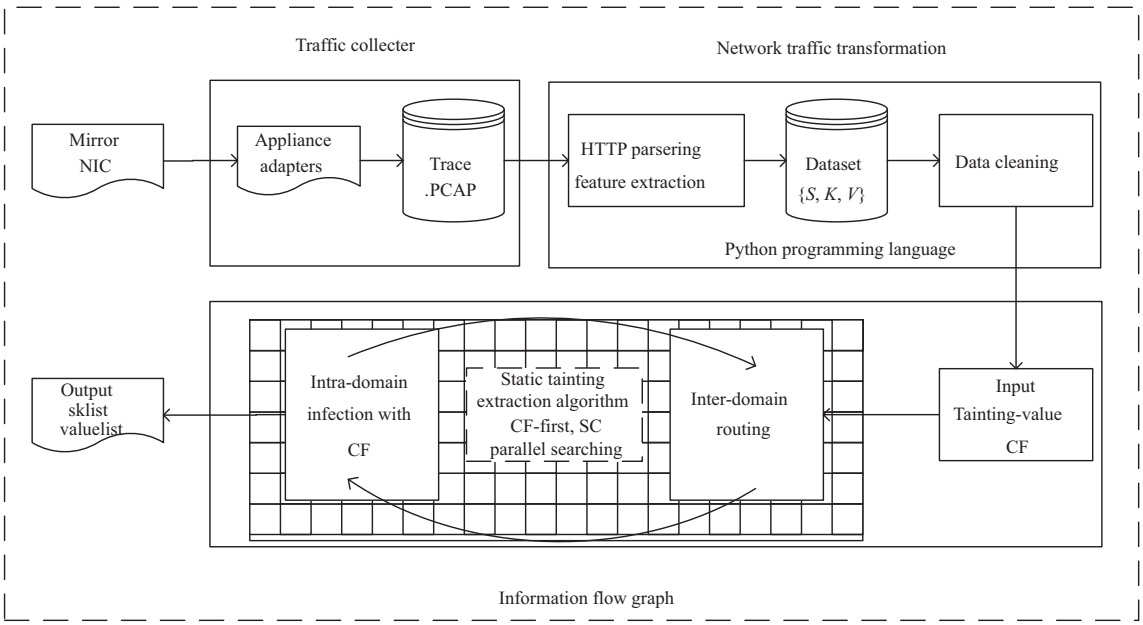

Figure 1 Data flow diagrams of the static tainting extraction approach.

collecting process captures the network traffic from the gigabit interface of an edge routing and stores it in the packet capture format (.PCAP). Network traffic transformation process performs data preprocessing and extracts the features from the hypertext transfer protocol (HTTP) header of the packets. Thus, the network traffic would be transformed from traces with the PCAP format to a dataset with three dimensions: service, key, and value. The methodologies for network traffic collection and network traffic transformation are introduced in Subsection 4.1.

Static tainting extraction algorithm identifies a typical PII type as the tainting-value type for the input, which automatically and accurately tracks PII types and their locations along with the information flow graph. Network traffic transformation process first transform the network traffic into a dataset using the three dimensions of service, key, and value. Then, an information flow graph is produced via a tainting process comprising two steps: intra-domain infection and inter-domain routing. Finally, a CF is used in the intra-domain infection against the obfuscation data in real-time traffic; this helps to execute the proposed approach in the right direction and to speed up the algorithm convergence. Three methods are also proposed to optimize the computational performance of our approach; these methods are introduced in Section 5.

3.1 Network traffic transformation problem

When users access various network services, a large amount of data on user behaviors is generated in the network traffic. In fact, the user behaviors can be summarized as answers to questions starting with four interrogative words: who, when, where, and what (4Ws): (1) Who accessed the network application services? (2) When was the application service accessed by the users? (3) Where is the users' traffic directed? (4) What type of information is leaked in the network traffic? The answers to these questions can be found via protocol analysis of the network traffic.

Consider the service accessed by the same user as S and the protocol used for the traffic transfer as P. If the location of the trace that may contain the personal identification information (or that needs to be detected) is K and the value transmitted at these locations is V, then the network traffic transmitted by the protocol P can be represented as the sample space as follows:

$$F_p = (S, K, V). \tag{1}$$

The network traffic with different network protocols is represented by S, K, and V; therefore, a large number of network traffic data can be reduced to a text format of the dataset. This is convenient for

Table 1 Domain-value data table

Domain	Value														
SK_1	v_1	v_2	–	–	–	–	–	–	–	–	–	–	–	–	–
SK_2	v_1	–	v_3	v_4	–	–	–	–	–	–	–	–	–	–	–
SK_3	–	–	v_3	–	–	v_7	–	–	–	–	–	–	–	–	–
SK_4	–	–	–	–	–	–	v_7	v_8	v_9	v_{10}	v_{11}	–	–	–	–
SK_5	–	–	–	–	–	–	–	–	–	v_{10}	–	v_{12}	v_{13}	–	–
SK_6	–	v_2	–	–	–	–	–	–	–	–	–	v_{12}	–	v_{14}	–
SK_7	–	–	–	–	v_5	–	–	–	v_9	–	–	–	–	v_{14}	v_{15}
SK_8	–	–	–	–	–	–	–	v_8	–	–	v_{11}	–	v_{13}	–	–
SK_9	–	–	–	–	v_5	v_6	–	v_8	–	–	–	–	–	–	–
⋮								⋮							
SK_n								...							

computer storage and calculation. Thus, the 4Ws can be transformed into questions that extract and identify whether values are verified as PII.

3.2 Information flow graph

PII is a generic term referring to "information" that can be used to distinguish or trace an individual's identity. It is unique and stable for a certain period of time. We assume that some values are recognized as PII when the users access the application service. Two user behavior relations are generated. First, the same PII will be shared in different service keys (SKs) when different application services are accessed by each user. Second, the SK will only contain the same category of PII within the same application service.

SKs accessed by the user are defined as the domain in which the passed values of each category represent the different user attributes. In the network traffic generated by the users, we assume that the application services can be represented as Domain $= \{\mathrm{SK}_1, \mathrm{SK}_2, \ldots, \mathrm{SK}_9, \ldots, \mathrm{SK}_n\}$; n is the number of SKs accessed by the user. The values contained in these SKs are represented as Value $= \{v_1, v_2, \ldots, v_{15}, \ldots, v_m\}$, and m is the number of values in the SK. Finally, a domain-value data table is generated using these two sets, as shown in Table 1, and the sample space size is the sum of values contained within each domain. Therefore, the information flow graph is constructed via intra-domain infection and inter-domain routing using this domain-value data table.

3.2.1 *Intra-domain infection*

Intra-domain infection is a method that selects values similar to PII within each SK. Suppose the SK accessed by the user i is SK_i and its value is V_i, whereas the SK accessed by the user j is SK_j and its value is V_j. If $\mathrm{SK}_i = \mathrm{SK}_j$, $V_i \neq V_j$, $V_i \in$ PII, $i, j \in k$, then any other value would also have $V_j \in$ PII. Thus, if any value in an SK is PII, then all the values in this SK are PII; this is similar to an infection process that infects the attributes of the values from one SK to another. This infection process is referred to as intra-domain infection. In other words, SK and the value that it contains can be selected via intra-domain infection to form the set $\mathrm{IDI}_{\mathrm{sk}} = \{\mathrm{SK} \in \mathrm{PII} \,|\, V \in \mathrm{PII}, V \in \mathrm{SK}\}$. The primary role of the intra-domain infection is to extract the same category of value in each SK.

However, network traffic will be mixed with many noises in real complex networks. Usually, the following situations occur: (1) The loss of packets during the transmission process results in incomplete values or missing values that are sometimes replaced by default values. (2) Owing to different versions of the same APP, the length and/or format of the values are different and the plain text values are mixed with the encrypted values. (3) There is confusion regarding similar categories of values in a few SK. If intra-domain infection is simply performed in the above situations, "over-tainting" is likely to result in many FPs. Therefore, we need to make a few appropriate interventions, such as using a CF for the

Table 2 Shared-value adjacency matrix

Domain	SK_1	SK_2	SK_3	SK_4	SK_5	SK_6	SK_7	SK_8	SK_9	...	SK_n
SK_1	–	v_1	–	–	–	v_2	–	–	–	–	–
SK_2	v_1	–	v_2	–	–	–	–	–	v_4	–	–
SK_3	–	v_3	–	v_7	–	–	–	–	v_6	–	–
SK_4	–	–	v_7	–	v_{10}	–	v_9	v_{11}	v_8	–	–
SK_5	–	–	–	v_{10}	–	v_{12}	–	v_{13}	–	–	–
SK_6	v_2	–	–	–	v_{12}	–	v_{14}	–	–	–	–
SK_7	–	v_5	–	v_9	–	v_{14}	–	–	v_5	–	–
SK_8	–	–	–	v_{11}	v_{13}	–	–	–	–	–	–
SK_9	–	v_4	v_6	v_8	–	–	v_5	–	–	–	–
⋮						⋮					
SK_n						...					–

intra-domain infection, which controls the direction of the intra-domain infection so that the same PII category can be accurately extracted.

A CF is used to control the selected content through the whole intra-domain infection. Depending on the prior PII knowledge, different rules can be used as the CF in the intra-domain infection. The CF rules include the string length constants or ranges, regular expressions, dictionaries, semantics, similarities, and so on. In fact, PII having a strict format can usually be directly and accurately identified via string matching using regular expressions or a dictionary, similar to our approach. Most other irregular PII can be identified using our approach that uses only the CF with a constant string length constant or range. Simultaneously, we extracted PII from values in each SK using a simple rule. Then, using these shared values from intra-domain infection, we build relations between any two SKs in the inter-domain routing.

3.2.2 *Inter-domain routing*

When users access different services, they leave the same value in different SKs; these are called shared values. The relation between two SKs linked by two or more shared values is known as inter-domain routing, and this relation is used to establish the adjacency matrix A (Table 2).

Then, SK as the node is represented by the set $N = \{\mathrm{SK}_1, \mathrm{SK}_2, \mathrm{SK}_9, \mathrm{SK}_n\}$ and the relation between these nodes can be represented by the set as follows:

$$\begin{aligned} E = \{&(\mathrm{SK}_1, \mathrm{SK}_2, v_1), (\mathrm{SK}_1, \mathrm{SK}_6, v_2), (\mathrm{SK}_2, \mathrm{SK}_3, v_3), (\mathrm{SK}_2, \mathrm{SK}_9, v_4),\\ &(\mathrm{SK}_7, \mathrm{SK}_9, v_5), (\mathrm{SK}_3, \mathrm{SK}_9, v_6), (\mathrm{SK}_3, \mathrm{SK}_4, v_7), (\mathrm{SK}_4, \mathrm{SK}_9, v_8),\\ &(\mathrm{SK}_4, \mathrm{SK}_7, v_9), (\mathrm{SK}_4, \mathrm{SK}_5, v_{10}), (\mathrm{SK}_4, \mathrm{SK}_8, v_{11}), (\mathrm{SK}_5, \mathrm{SK}_6, v_{12}),\\ &(\mathrm{SK}_5, \mathrm{SK}_8, v_{13}), (\mathrm{SK}_6, \mathrm{SK}_7, v_{14}), \ldots, (\mathrm{SK}_{n-1}, \mathrm{SK}_n, v_m)\}. \end{aligned} \tag{2}$$

The above conditions are used to draw the information flow graph IFG $= (N, E)$, as shown in Figure 2.

For concise representation and easy understanding of the information flow diagram, we assume that only one shared-value relation would exist between the domains; the weight of each edge of the information flow graph is 1. In real-time traffic, there may be more than one shared value having a strong relation between the two SK nodes.

3.2.3 *Static tainting extraction algorithm*

We propose a static tainting extraction algorithm (Algorithm 1) that is used to automatically draw the information flow graph. The datasets generated by filtering the network traffic (as shown in Subsection 3.1) are first input into the algorithm together with the tainting-value. Then, the static tainting extraction algorithm draws an information flow diagram using the shared value of the same category as tainting-value. The diagram is a cyclical pattern of alternate intra-domain infection and inter-domain routing. Finally, the algorithm outputs the two lists, valuelist and SKlist, which respectively represent the set of values and services belonging to PII.

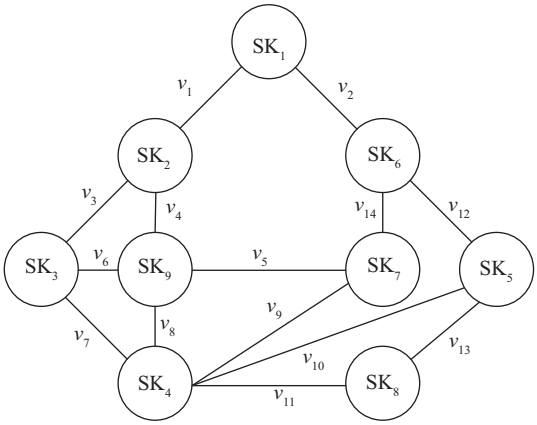

Figure 2 Information flow graph.

Algorithm 1 Static tainting extraction algorithm

Require: Tainting-value, DataSet, CONDITION_RULES(tainting-value);
Ensure: ValueList = ∅, SKList = ∅;

```
ValueList ⇐ tainting-value;
// Select a tainting-value as input and put it in the ValueList.
Constraint function ⇐ CONDITION_RULES(tainting-value);
// You can choose an appropriate constraint function for the different types of tainting-values.
for Value ← each value of ValueList do
    // Beginning of inter-domain routing.
    TempSKList = ∅;
    // Searching each line of dataset.
    for Line ← each line of DataSet do
        if (Value ∈ Line[value]) and (Line[SK] ∉ TempSKList) then
            // Put new SK in the TempSKList if the new SK include a shared value.
            TempSKList ⇐ Line[SK];
        end if
    end for
end for
for  SK ← each value of TempSKList do
    // Beginning of intra-domain infection.
    if SK ∉ SKList then
        SKList ⇐ SK;
        TempValueList = ∅;
        for Line ← each line of DataSet do
            // Put all values belonging to an SK in the TempValueList if the values conform to the rule of the conditional function.
            if SK == Line[SK] and CONDITION(Line[Value]) == constraint function then
                TempValueList ⇐ Line[Value];
            end if
            for Value ← each value of TempValueList do
                if Value ∉ ValueList then
                    ValueList ⇐ Value;
                end if
            end for
        end for
    end if
end for
```

Static tainting extraction algorithm is based on breadth-first search algorithm, as shown in Figure 3. Suppose V_1 is a tainting-value used as the tainting-value input. Initially, the value list contained only one value V_1 and the SK list was empty. Using the value V_1 as a shared value, SK_1 and SK_2 are first determined via inter-domain routing. Then, the other three values, V_2, V_3, and V_4, are determined via intra-domain infection in the SK_1 and SK_2. In the first round, each cyclical pattern of alternating intra-domain infection and inter-domain routing is defined. In the second round, V_2 is used as a shared value and

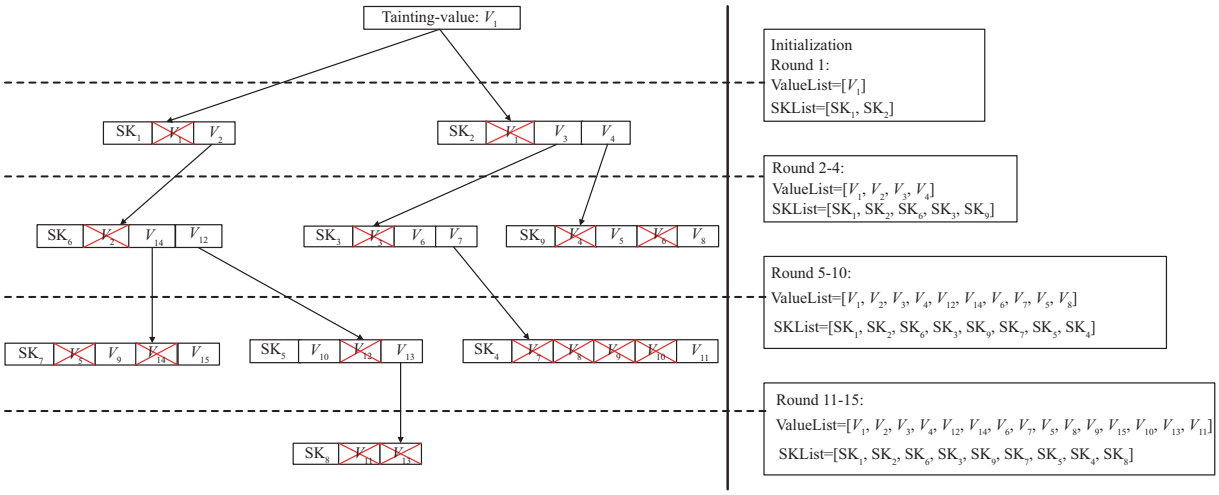

Figure 3 (Color online) Breadth-first search algorithm of the information flow graph.

SK_6 is determined via inter-domain routing; then, V_{12} and V_{14} are determined via intra-domain infection in the SK_6. In the third round, V_3 is used as a shared value and SK_3 is determined via inter-domain routing. Then, V_6 and V_7 are determined via intra-domain infection in the SK_3. This cyclical pattern is executed round by round until no new shared value or new SK is determined. Eventually, the algorithm is converged to the 15th round in the example and the algorithm ends the traversal of the information flow graph. As shown in Figure 3, ValueList $= [V_1, V_2, V_3, V_4, V_{12}, V_{14}, V_6, V_7, V_5, V_8, V_9, V_{15}, V_{10}, V_{13}, V_{11}]$ and SKList $= [\mathrm{SK}_1, \mathrm{SK}_2, \mathrm{SK}_6, \mathrm{SK}_3, \mathrm{SK}_9, \mathrm{SK}_7, \mathrm{SK}_5, \mathrm{SK}_4, \mathrm{SK}_8]$ are the outputs.

4 Evaluation

For our evaluation, we first captured the HTTP trace from a real-time campus network. Then, we transformed the trace into a dataset as a matrix. Next, we applied our proposed approach on the dataset and evaluated our approach in terms of precision and recall. Then, we compared the performance of our approach with the existing method. Finally, we discussed some concerns that are either related to or beyond the scope of our approach.

4.1 Experiment setup and dataset

We conducted our experiments using a real-time capture of high-speed traffic on a multi-core platform [35, 36] and captured the trace from the gigabit interface of an edge router of a campus wireless network from November 7 to November 13, 2016. During this week, we captured a total of 389222281 HTTP packets. The collected raw traffic was stored as raw data using the PCAP format. We used HTTP as an example, but our approach can be extended to other protocols.

The two fields obtained after the process of capturing packets are shown in Figure 4. The POST and GET methods were first extracted from the HTTP header of the packets. Then, the key-value pairs [37] were extracted from the GET method field. The extraction process splits the GET method field using the symbols "?", "&", and "=" by turns. Regular expressions written in Python could implement this extraction process. Next, the network traffic was changed from the PCAP format into the TXT format and constructed into a dataset having three dimensions: service, key, and value. Then, the data filtering processes removed the irregular service, key, and value. If two items were the same, they would be aggregated in the dataset. Finally, the dataset could be represented as a sample space F_{HTTP} (see Eq. (1) in Subsection 3.1) with three dimensions: service, key, and value. For example, 13 items of data were extracted from the HTTP packet, as shown in Figure 4 and Table 3.

We defined that PII comprises four categories: (1) device identifiers, which collected data from a device or an OS installation (IMEI, MAC address, and IDFA); (2) user identifiers, which identified the user (user

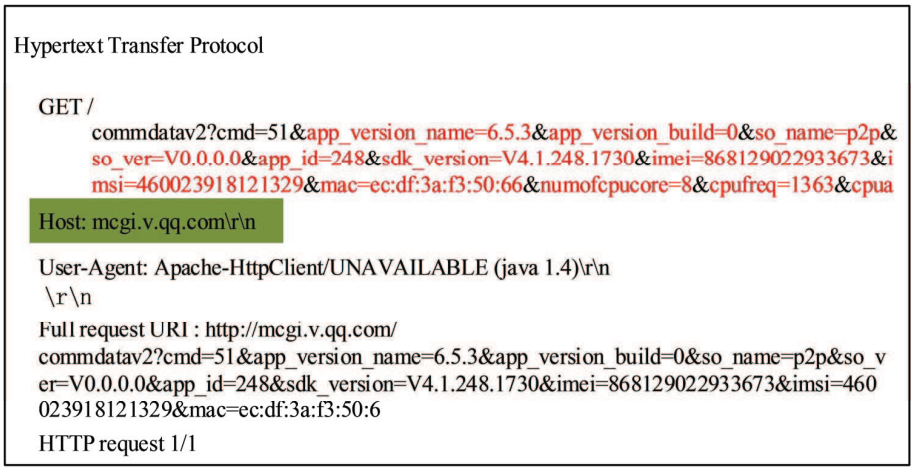

Figure 4 (Color online) Location of service and key-value pairs in the HTTP header field.

Table 3 Feature extraction

Service	Key	Value
mcgi.v.qq.com	cmd	51
mcgi.v.qq.com	app_version_name	6.5.3
mcgi.v.qq.com	app_version_build	0
mcgi.v.qq.com	so_name	p2p
mcgi.v.qq.com	so_ver	V0.0.0.0
mcgi.v.qq.com	app_id	248
mcgi.v.qq.com	sdk_version	V4.1.248.1730
mcgi.v.qq.com	imei	868129022933673
mcgi.v.qq.com	imsi	460023918121329
mcgi.v.qq.com	mac	ec:df:3a:f3:50:66
mcgi.v.qq.com	numofcpucore	8
mcgi.v.qq.com	cpufreq	1363
mcgi.v.qq.com	null	cpua

name/user id, password, name/nick name, and email); (3) contact information (phone numbers); and (4) location (GPS latitude and longitude). This category list of PII was prepared based on the observations made in this study. This list was not exhaustive; however, it covers most PII that concerns users. The list of tracked PII will be updated when we obtain more information about the additional PII types.

The ground truth of PII of these users is not determined; therefore, we establish the ground truth by relying on 10 human raters who label PII. If more than five human raters label the value as PII, then we consider that value is labeled as PII in the dataset. We checked 254398 PII lines from a total of 33703512 lines in this dataset. PII accounted only for 0.7548%. This result means that we accurately determined PII in large network traffic. We selected only the key terms for each PII for the eight types of four categories, along with their evaluation and comparison in the dataset.

4.2 Precision and recall

This subsection evaluates the effectiveness of our approach in terms of precision and recall. We first applied our approach to the dataset and analyzed the results with precision and recall to evaluate our approach for the ground truth of the dataset. We then compared our approach with the latest approach and discussed the results of the evaluation and comparison.

Precision and recall metrics were used to evaluate our approach. These two statistical measures are defined as follows:

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{3}$$

Table 4 Rules of the baseline method

Category	Type	Rules(k-s:key-semantics, reg:regular expression)
User identifiers	User name/id, nick name	k-s: substr. of user name/id, nick, login, or equal to "id" or "name"
	Password	k-s: substr. of password, or equal to "pwd"
	Email	reg: ^ [- _ \w\.]{0,64}@{1}([-\w]{1,63}\.)*[-\w]{1,63}$
Device identifiers	IMEI	reg: value.length =15 and value.isdigit()
	MAC address	reg: ^ ([0-9a-fA-F]{2})?[-:]([0-9a-fA-F]{2}){5}$'
	IDFA	reg: ^ ([0-9a-fA-F]{8}((-[0-9a-fA-F]{4}){3})-[0-9a-fA-F]{12}$
Contact information	Phone number	reg: ^ 1[3458]\d{9}$)
Location	GPS,	reg: ^ -?((0—1?[0-7]?[0-9]?)(([.][0-9]1,6)?)—180(([.][0]1,6)?))$
	Latitude and longitude	and k-s: substr. of lng, loc, long, loc, or equal to "x" or "y"

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{4}$$

$$F_1 = 2 \times \frac{P \cdot R}{P + R} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{5}$$

For a binary classification, the elements will be classified as positive or negative and four cases including true positive (TP), FP, FN, and true negative (TN). TP represents the number of samples of PII that were accurately detected; FP represents the number of normal samples that were detected as PII; FN represents the number of PII that were detected as normal samples; and TN represents the number of normal samples that were correctly classified. From (3) and (4), we can see that precision represents the frequency of actual PII samples within the samples that were classified as PII; recall represents the frequency of samples that were correctly detected as PII within all PII samples that were to be tested. Finally, we used the F1 score to evaluate the performance of the experimental results of our method; this score is the harmonic mean of precision and recall, as shown in (5). The F_1 score combines the results of precision and recall, and a high F_1 score shows a better approach.

4.3 Comparison

Herein, we apply our proposed approach to the dataset and evaluate its performance by comparing it with the baseline approach [3]. We then present interesting findings on the users' PII via an in-depth analysis of the discovered SKs and values. The baseline approach combines the key-semantics method and the regular expression method. The key-semantics method analyzes the semantics of the key names. We leverage the common intuition that keys that are suggestive of PII (e.g., keys named "email") would have PII as their values. The regular expression method matches PII pattern with a uniform format (e.g., IMEI, MAC address, and IDFA) in the three categories of device identifiers, contact information, and location. The key-semantics method is used to search for other PII that do not match the regular expression (except email) with clear semantics (e.g., username, password, user id, and name/nick name) in the user identifiers' category. Occasionally, these unstable PII types take uniform format PII as their value; for example, both phone number and email were often used as user names or IDs. The baseline method rules are given in Table 4.

We then compared our approach to this baseline approach based on our ground truth of the dataset, as shown in Table 5 and Figure 5. The experimental results proved that our approach was superior to the baseline method obtained from eight types of four PII categories. The proposed method overcomes the problems faced by the baseline method with high FPs. The proposed approach is more precise although it shows a few FPs because some values of the timestamp are mixed with positives, particularly the location that has more FPs.

Our approach also missed a few positive samples resulting in some FNs because of the following two reasons: (1) The sample space remains a complete space until it is filtered by the CFs. This space lost some positive samples, which usually exist in the types that use key semantics as the CF. For example, a key transmitting a password sometimes hides its semantics and the key can be named freely using any character. Therefore, some positive samples filtered by the CF were lost. (2) Our approach could

Table 5 Comparison between the proposed approach and the baseline method

Type	Baseline	TP	FP	FN	P	R	F1	Taint	TP	FP	FN	P	R	F1
IMEI value	16559	6519	10040	0	0.3937	1.0000	0.5650	6798	6219	579	300	0.9148	0.9540	0.9340
IMEI SK	4650	3025	1625	0	0.6505	1.0000	0.7883	3045	3009	36	16	0.9882	0.9947	0.9914
Mac value	95703	5822	89881	0	0.0608	1.0000	0.1147	4925	4799	126	1023	0.9744	0.8243	0.8931
Mac SK	5329	1024	4305	0	0.1922	1.0000	0.3224	904	834	70	190	0.9226	0.8145	0.8651
IDFA value	115892	15432	100460	0	0.1332	1.0000	0.2350	13044	13044	0	2388	1.0000	0.8453	0.9161
IDFA SK	3708	1876	1832	0	0.5059	1.0000	0.6719	1517	1517	0	359	1.0000	0.8086	0.8942
Phone value	36680	849	35831	0	0.0231	1.0000	0.0452	541	539	2	310	0.9963	0.6349	0.7755
Phone SK	1434	411	1023	0	0.2866	1.0000	0.4455	185	183	2	228	0.9892	0.4453	0.6141
Email value	25208	443	24765	0	0.0176	1.0000	0.0345	191	191	0	252	1.0000	0.4312	0.6025
Email SK	1850	223	1627	0	0.1205	1.0000	0.2151	141	141	0	82	1.0000	0.6323	0.7747
Location value	15917	9761	6156	224	0.6132	0.9776	0.7537	12788	9719	3069	266	0.7600	0.9734	0.8536
Location SK	770	642	128	170	0.8338	0.7906	0.8116	678	610	68	202	0.8997	0.7512	0.8188
Name value	315206	214580	100626	0	0.6808	1.0000	0.8101	220385	204792	15593	9788	0.9292	0.9544	0.9416
Name SK	8046	4190	3856	0	0.5208	1.0000	0.6849	4495	3932	563	258	0.8747	0.9384	0.9055
Password value	904	631	273	137	0.6980	0.8216	0.7548	1104	575	529	193	0.5208	0.7487	0.6143
Password SK	225	208	17	30	0.9244	0.8739	0.8985	254	223	31	15	0.8780	0.9370	0.9065
Total	648081	265636	382445	561	0.4099	0.9979	0.5811	270995	250327	20668	15870	0.9237	0.9404	0.9320

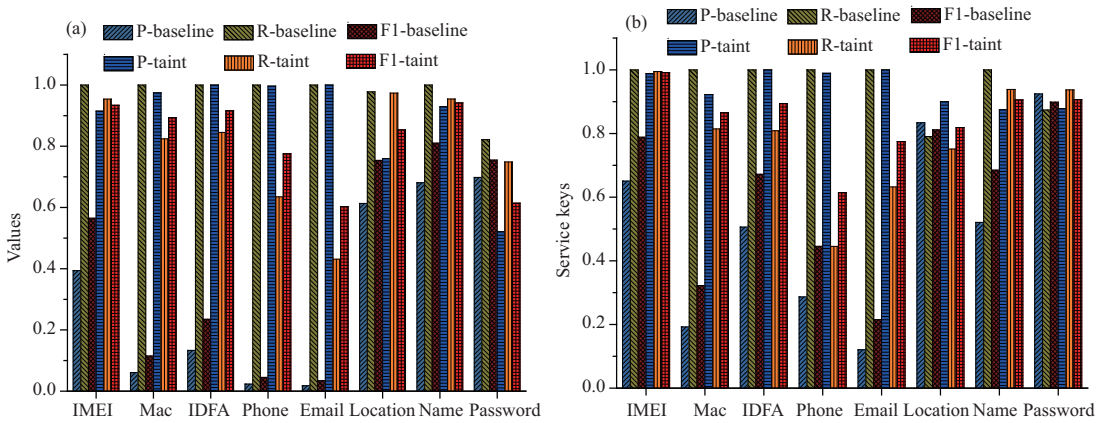

Figure 5 (Color online) Comparison of precision, recall, and F1 scores of eight PII types between the baseline and taint with values (a) and service keys (b).

not always completely route the entire sample space because the shared values between the SKs are insufficient. For example, if shared values in the email, phone, and password types are insufficient, then our approach can only find less than half the shared values. This resulted in more FNs and less recall.

In Table 5, the precision of the password value does not increase with the increase in the precision of the password SK. Thus, some key semantics did not relate to the password meaning; therefore, determining PII using the key semantic analysis of the baseline method is difficult.

Depending on our prior knowledge, we can use a variety of rules as CFs, such as the string length constants, character ranges, regular expressions, dictionaries, and keyword semantics. Herein, the CFs in our approach were taken from the rules of the baseline method. We usually represent regular PII using regular expressions; they can also be represented by the string length constants and/or character ranges. For example, regular expressions that represent device identifiers, phone numbers, e-mails, latitudes, and longitudes can be used as CFs in our approach; in particular, IMEI uses just the string length and character ranges to represent CF. For irregular PII, key semantics or dictionaries can be used as CFs. For example, the key-semantics method used the word "userid" as the CF, and users' names may be listed in the name dictionary as the CF of the user name. If the transmitted information hides the word semantics, the proposed approach with the key semantics of CF will lose the hidden word of the key and

will cause some recall losses. For instance, certain keys that contain the user id values sometimes are named as "number". In this case, the key semantics of CFs cannot use the key semantic of "number" to identify the value of the user id. The hidden semantics also simultaneously reduces the possibility of revealing PII.

In the worst-case scenario, CF might not be used because a suitable CF might not be formed, e.g., in a situation in which the password did not use a CF. However, these results may sometimes have more recall because a hidden SK might be found in the tainting process of the information flow graph.

In summary, our approach has achieved better performance, particularly in the device identifier category and the user name type because more share value is present in PII values. Our approach could recall 94.04% of PII, and the precision reached 92.37%. Correspondingly, the baseline approach could recall 99.79% of PII, and the precision reached 40.99%, which was caused by a large number of FPs generating "over-tainting". Overall, our approach has achieved high precision at the expense of a few recalls. Furthermore, our approach has performed well in determining PII; however, our approach is not limited to identifying only PII. By providing an appropriate input (i.e., a tainting-value) and a conditional function, other target information can be accurately determined, such as the time stamp, via our approach.

5 Optimization

Herein, we propose three methods to optimize our approach: CF-first, parallel searching (PS), and tainting-value sequence choice (SC). These optimization methods were applied step by step to improve our approach. CF-first reduced the sample space to improve the computing space overhead. PS matched the multiple values in one round and the computing time overhead. The tainting-value SC ensured the best results with more number of PII values and less number of searching rounds.

The original approach uses only a single-threaded search to execute the algorithm against the large-scale sample space. In the sample space, each row contains three dimensions ($\{S, K, V\}$) in the matrix dataset; each dimension is called a term. Each value searches the sample space once, and the number of values determines the number of rounds repeated for the search. For instance, as shown in Table 1, the sample space size (SSS) is the sum of the values contained within each domain when the number of values is m. The SSS can be expressed as

$$\begin{aligned}\mathrm{SSS} &= \sum_{n=1}^{n} \mathrm{COUNT}(\mathrm{Value}(\mathrm{SK}_n)) \\ &= \mathrm{COUNT}(\{V_1, V_2 \in \mathrm{SK}_1\}, \{V_1, V_3, V_4 \in \mathrm{SK}_2\}, \ldots, \{V_4, V_6, V_8 \in \mathrm{SK}_9\}, \ldots, \{V_m \in \mathrm{SK}_n\}). \end{aligned} \tag{6}$$

Then, the overhead is

$$O = m \times \mathrm{SSS}. \tag{7}$$

If we assume that in the sample space shown in Figure 3, the number of SK is 9 ($n = 9$) and the number of values is 15 ($m = 15$), then our original approach overhead can be calculated as follows:

$$\begin{aligned} O = m \times \mathrm{SSS} &= m \sum_{n}^{1} \mathrm{COUNT}(\mathrm{Value}(\mathrm{SK}_n)) \\ &= m \times \mathrm{COUNT}(\{V_1, V_2 \in \mathrm{SK}_1\}, \{V_1, V_3, V_4 \in \mathrm{SK}_2\}, \ldots, \{V_4, V_6, V_8 \in \mathrm{SK}_9\}) \\ &= 15 \times (2 + 3 + 3 + 5 + 3 + 3 + 3 + 2 + 4) = 15 \times 28 = 420. \end{aligned} \tag{8}$$

In this sample space, we have 15 values (which are required to search the sample space for 15 rounds) and 28 terms. Therefore, the overhead is 420, which represents the number of times the terms have been searched in the sample space. In our approach, the space complexity depends on the size of the sample space, i.e., the number of terms in the dataset. The time complexity is not only related to the space complexity but also related to the number of rounds that we introduced in Algorithm 1. To reduce the

Table 6 Sample sizes of the value types using CF-first

Category	Types	Sample size (terms)	Proportion (%)
User identifiers	User name/id, nick name	322754	0.9576
	Password	1118	0.0033
	Email	30944	0.0918
Device identifiers	IMEI	107077	0.3177
	MAC address	145472	0.4316
	IDFA	145788	0.4326
Contact information	Phone number	40060	0.1189
Location	GPS, latitude and longitude	24992	0.0742
Total		818295	2.4279

algorithm overhead caused by a large-scale sample space and/or a single-threaded search, we suggest three methods: CF-first, PS, and SC.

In our experiment, all simulation results were produced and compared on the Linux servers with the x86-64 architecture. The server was configured with two CPUs having 32 cores and 64 GB memory, and the CPU model was Inter (R) Xeon (R) E7-4820 2.0 GH, which was used to perform all the calculations on the dataset.

5.1 Constraint function first

Our approach involved a large-scale sample space with massive data; therefore, we used high overheads that consumed a large amount of computing resources and time. The CF-first method first performed the CF, which filtered and cleaned the data. Then, other algorithmic processes were performed to reduce the SSS. The sample space having 33703512 terms was reduced to 818295 terms using the CF-first method.

When the number of values remains constant, the SSS obtained using the CF-first method was only 2.4279% of the previous size, as shown in Table 6. For the IMEI type, the CF-first method reduced the execution time overhead from 3 days 6:33:52.740911 to 0:15:20.720231, which is only 0.3253% of the execution time overhead of our original approach.

5.2 Parallel searching

PS is used to improve single-thread searching. In Algorithm 1, PS uses a single-threaded search to match just one value through one search on the sample space and the number of rounds is determined by the number of values. For example, as shown in Figure 3, PS needs to search the sample space using 15 rounds based on the breadth-first search algorithm. However, many values are obtained during the one round that executed the intra-domain infection and the inter-domain routing once. For example, as shown on Figure 3, PS is also an extraction algorithm based on the breadth-first search algorithm and V_1 is used as a tainting-value for the input of the PS algorithm depending on the data given in Table 1. Initially, the value list contains only one value V_1 and the SK list is empty. In the first round, similar to Algorithm 1, PS uses the value V_1 as a shared value. SK_1 and SK_2 are found during the inter-domain routing. Then, V_2, V_3, and V_4 are found in SK_1 and SK_2 during the intra-domain infection. This process is different from the second round. V_2, V_3, and V_4 are included in the value list, and these values are simultaneously considered as shared values. If any SK contains these shared values in each round, then these values can be placed in an SK list. Thus, SK_6, SK_3, and SK_9 are determined during inter-domain routing and V_{12}, V_{14}, V_6, V_7, V_5, and V_8 are determined in SK_6, SK_3, and SK_9 during intra-domain infection. PS is executed round by round until no new shared value or no new SK is determined in the sample space. For convergence, a parallel search needs only four rounds (from round 1 to 4), as shown in Figure 3, which is better than Algorithm 1, which needs 15 rounds.

Furthermore, PS is applied to reduce the IMEI overhead. PS reduces the number of rounds from 6798 to 9, as shown in Figures 6 and 7. It reduces the execution time overhead from 0:15:20.720231

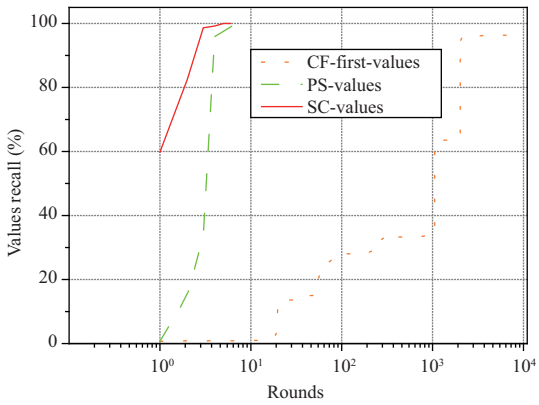

Figure 6 (Color online) Comparison of optimized performance for values recall.

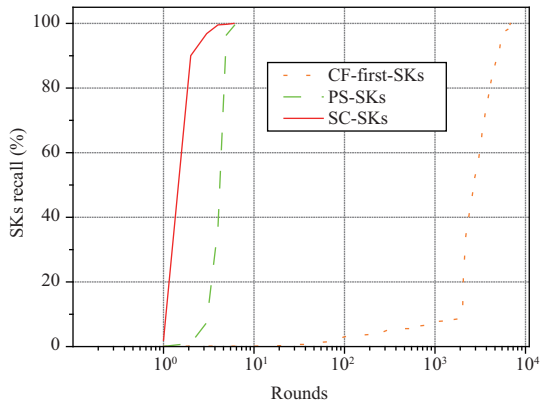

Figure 7 (Color online) Comparison of optimized performance for SKs recall.

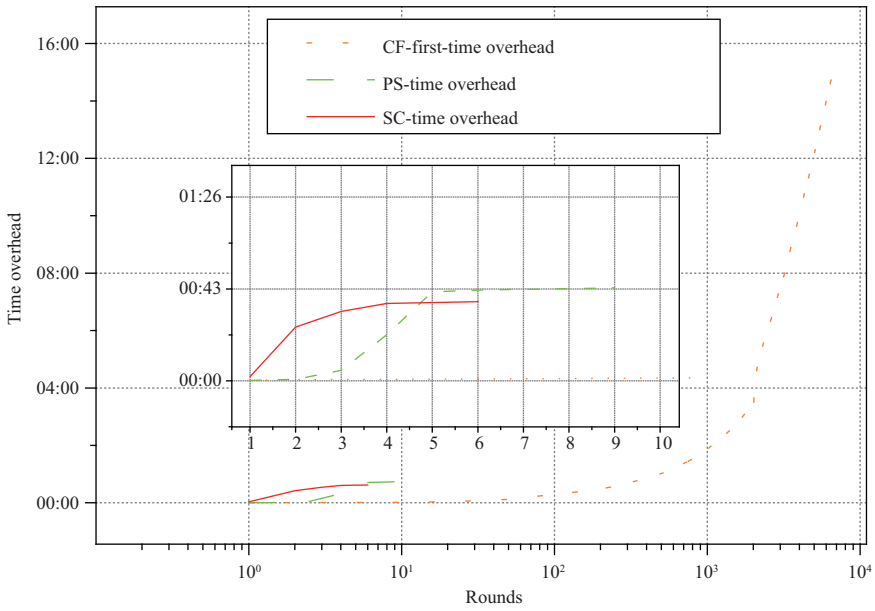

Figure 8 (Color online) Comparison of optimized performance for time overhead.

to 0:00:43.747925, as shown in Figure 8. The number of rounds and execution time are 0.1324% and 4.6739%, respectively, of the overhead of the original approach.

5.3 Tainting-value sequence choice

Our approach faces an additional problem with regard to selecting an appropriate tainting-value for the input. When the breadth-first search algorithm is used to traverse the graph, the number of rounds is not unique at different beginning vertices. In PS, if we chose different tainting-values for the input, then the number of rounds would also change. Therefore, when a random selection is used to choose the tainting-values within PS, it is not stable because sometimes the best results are not obtained.

We propose the tainting-value SC to obtain the best tainting-value for the input. SC counts the two factors for all the values in the sample space after the CF; it computes the number of SKs that contain the same shared value and the values in the SK. Assuming that a value is contained by k SK, its first-taint-factor (FTF) can be calculated as follows:

$$\mathrm{FTF}_{v_m} = k \sum_{1}^{k} \mathrm{COUNT}(\mathrm{value}|\mathrm{value} \in \mathrm{SK}_k). \tag{9}$$

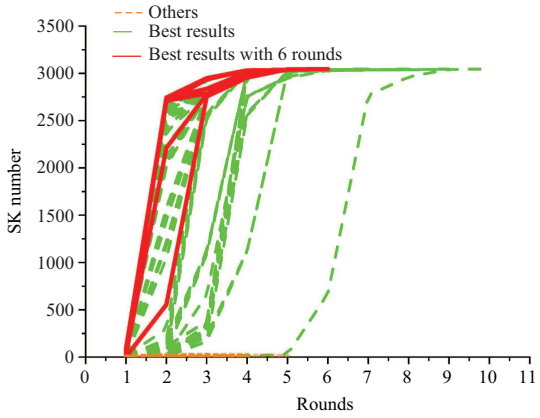

Figure 9 (Color online) Convergence of tainting-value sequence choice with SKs.

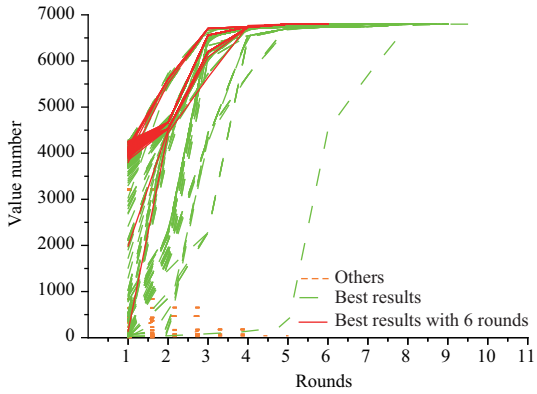

Figure 10 (Color online) Convergence of tainting-value sequence choice with values.

For example, in Tables 1 and 2, the FTF of v_1 can be calculated by $\mathrm{FTF}_{v_1} = 2 \times (2 + 4) = 12$. When the FTFS of all the values have been calculated, we obtain a sequence according to the order of the value of FTF. We choose the value with the maximum FTF as the tainting-value for the input. The sequence $[v_8, v_9, v_5, v_7, v_{10}, v_4, v_6, v_{11}, v_{14}, v_1, v_3, v_{12}, v_2, v_{13}, v_{15}]$ can be obtained for the above example. In this sequence, either v_8 or v_9 can be chosen for the input of SC because sometimes a few of values have same maximum FTF and both v_8 and v_9 have the same FTF = 18. We use v_8 or v_9 as inputs to verify that SC is effective. SC reduces the number of rounds from 4 to 3, and SC is better than PS with the random input. Furthermore, the tainting-value SC method is applied to reduce the IMEI overhead. SC reduces the number of rounds from 9 to 6, which proves that the tainting-value SC method is better than other methods. In fact, tainting-value SC can produce double the results with half the effort and maximum SKs and values can be obtained in the first round for the taint process. Figures 9 and 10 show the convergence rate of the algorithm for all IMEI-type values as the tainting-value of the input. It shows that approximately 62.63% of the inputs (refer to green dashed line in Figures 9 and 10) can converge to the best result. Only 4.44% of the inputs (refer to red solid line in Figures 9 and 10) can converge with six rounds. In other words, more than half of the tainting-value with random choice for input can achieve the best result. However, the tainting-value SC is more stable and always ensures the best results which is only 4.44% possibility.

In summary, we have applied all three optimization methods one by one; these methods reduce the execution time overhead from 3 days 6:33:52.740911 to 0:00:43.747925, and the number of rounds from 6798 to 6; the execution time and number of rounds have reduced by 0.0152% and 0.0883%, respectively, of our original approach. CF-first is used to reduce the sample space. Then, SC is used to evaluate the best input. Finally, the PS algorithm is executed to acquire PII. CF-first and SC are actually executed before the algorithm. In fact, CF-first and SC are usually applied to PII having widely distributed types, such as device identifiers and user names; these method help reduce the sample space and computing overhead. In contrast, CF-first and SC are not significantly applied to PII with less distribution types in the dataset, such as passwords, because 238 passwords have a very small sample space and less shared values to cause tainting. To use this type of PII as passwords, we suggest that CF-first and SC be ignored in the optimization process; this produces results that are approximate to the result before optimization.

6 Conclusion

Our approach automatically extracted PII from large-scale network datasets without using any manual analysis and feedback. We did not deploy any additional applications on the client's devices. Our approach exhibited superior performance as well as low overheads and could be applied for network privacy leak detection, network forensics, and user portraits. Moreover, our approach focused only on the

traffic of the network layer in the ISP level; this could be one way of breaking the user data monopoly of ASPs. Furthermore, we proposed three methods to improve the performance of our approach.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61672101, U1636119, 61866038, 61962059).

References

1 Krishnamurthy B, Wills C E. On the leakage of personally identifiable information via online social networks. In: Proceedings of the 2nd ACM Workshop on Online Social Networks, 2009. 7–12
2 Mccallister E, Grance T, Scarfone K A. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). Special Publication (NIST SP)-800-122. 2010
3 Liu Y, Song H H, Bermudez I, et al. Identifying personal information in internet traffic. In: Proceedings of ACM on Conference on Online Social Networks, 2015. 59–70
4 Enck W, Gilbert P, Chun B G, et al. TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones. Commun ACM, 2014, 57: 99–106
5 Ball J, Schneier B, Greenwald G. NSA and GCHQ target Tor network that protects anonymity of web users. Guardian Web, 2013. https://www.schneier.com/essays/archives/2013/10/nsa_and_gchq_target.html
6 Yang Z, Yang M, Zhang Y, et al. Appintent: analyzing sensitive data transmission in android for privacy leakage detection. In: Proceedings of ACM Sigsac Conference on Computer & Communications Security, 2013. 1043–1054
7 Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In: Proceedings of ACM Sigplan Conference on Programming Language Design and Implementation, 2014. 259–269
8 Au K W Y, Zhou Y F, Huang Z, et al. Pscout: analyzing the android permission specification. In: Proceedings of ACM Conference on Computer and Communications Security, 2012. 217–228
9 Egele M, Kruegel C, Kirda E, et al. PiOS: detecting privacy leaks in IOS applications. In: Proceedings of NDSS, 2011. 177–183
10 Cao Y, Fratantonio Y, Bianchi A, et al. Edgeminer: automatically detecting implicit control flow transitions through the android framework. In: Proceedings of Network and Distributed System Security Symposium, 2015
11 Babil G S, Mehani O, Boreli R, et al. On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices. In: Proceedings of International Conference on Security and Cryptography (SECRYPT), 2013
12 Song Y, Hengartner U. Privacyguard: a VPN-based platform to detect information leakage on android devices. In: Proceedings of ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, 2015
13 Ren J, Rao A, Lindorfer M, et al. Recon: revealing and controlling PII leaks in mobile network traffic. In: Proceedings of International Conference on Mobile Systems, Applications, and Services, 2016. 361–374
14 Razaghpanah A, Vallina-Rodriguez N, Sundaresan S, et al. Haystack: in Situ mobile traffic analysis in user space. 2015. ArXiv:1510.01419
15 Le A, Varmarken J, Langhoff S, et al. Antmonitor: a system for monitoring from mobile devices. In: Proceedings of ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big, 2015. 15–20
16 Continella A, Fratantonio Y, Lindorfer M, et al. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In: Proceedings of Network and Distributed System Security Symposium, 2017
17 Englehardt S, Han J, Narayanan A. I never signed up for this! Privacy implications of email tracking. In: Proceedings of Privacy Enhancing Technologies, 2018
18 Srivastava G, Bhuwalka K, Sahoo S K, et al. Privacyproxy: leveraging crowdsourcing and in situ traffic analysis to detect and mitigate information leakage. 2017. ArXiv:1708.06384
19 Seneviratne S, Kolamunna H, Seneviratne A. A measurement study of tracking in paid mobile applications. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, 2015
20 Chen T, Ullah I, Kaafar M A, et al. Information leakage through mobile analytics services. In: Proceedings of Workshop on Mobile Computing Systems & Applications, 2014
21 Leontiadis I, Efstratiou C, Picone M, et al. Don't kill my ads! Balancing privacy in an ad-supported mobile application market. In: Proceedings of the 12th Workshop on Mobile Computing Systems & Applications, 2012
22 Georgiev M, Iyengar S, Jana S, et al. The most dangerous code in the world: validating ssl certificates in non-browser software. In: Proceedings of ACM Conference on Computer and Communications Security, 2012. 38–49
23 Fahl S, Harbach M, Muders T, et al. Why Eve and Mallory love Android: an analysis of Android SSL (in) security. In: Proceedings of ACM Conference on Computer and Communications Security, 2012. 50–61
24 Ren J J, Lindorfer M, Dubois D J, et al. Bug fixes, improvements, ... and privacy leaks – a longitudinal study of PII leaks across Android app versions. In: Proceedings of Network and Distributed System Security Symposium (NDSS), 2018
25 Lindorfer M, Neugschwandtner M, Weichselbaum L, et al. Andrubis – 1,000,000 apps later: a view on current android malware behaviors. In: Proceedings of International Workshop on Building Analysis Datasets & Gathering Experience Returns for Security, 2016. 3–17
26 Bell J, Kaiser G. Phosphor: illuminating dynamic data flow in commodity JVMs. ACM Sigplan Notice, 2014, 10: 83–101

27 Rastogi V, Qu Z Y, Mcclurg J, et al. Uranine: real-time privacy leakage monitoring without system modification for Android. In: Proceedings of International Conference on Security and Privacy in Communication Systems, 2015. 256–276
28 Hornyack P, Han S, Jung J, et al. "These aren't the droids you're looking for": retrofitting Android to protect data from imperious applications. In: Proceedings of ACM Conference on Computer and Communications Security (CCS), 2011
29 Zhu D Y, Jung J, Song D, et al. TaintEraser: protecting sensitive data leaks using application-level taint tracking. SIGOPS Oper Syst Rev, 2011, 45: 142
30 Arefi Meisam N, Alexander G, Crandall J R. PIITracker: automatic tracking of personally identifiable information in windows. In: Proceedings of the 11th European Workshop on Systems Security (EuroSec'18), 2018
31 Machiry A, Tahiliani R, Naik M. Dynodroid: an input generation system for Android apps. In: Proceedings of Joint Meeting on Foundations of Software Engineering, 2013. 224–234
32 Carter P, Mulliner C, Lindorfer M, et al. Curiousdroid: automated user interface interaction for android application analysis sandboxes. In: Proceedings of International Conference on Financial Cryptography and Data Security, 2016. 231–249
33 Hao S, Liu B, Nath S, et al. Puma: programmable Ui-automation for large-scale dynamic analysis of mobile apps. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, 2014. 204–217
34 Starov O, Nikiforakis N. Extended tracking powers: measuring the privacy diffusion enabled by browser extensions. In: Proceedings of International Conference on World Wide Web, 2017. 1481–1490
35 Liu Y. Design and implementation of high performance IP network traffic capture system. J Yanan Univ (Natl Sci Edit), 2017, 36: 22–24
36 Liu Y, Zhan Y H. Research on mobile terminal equipment recognition method based on HTTP traffic. Modern Electron Tech, 2018, 41: 93–95
37 Dai S F, Tongaonkar A, Wang X Y, et al. NetworkProfiler: towards automatic fingerprinting of Android apps. In: Proceedings of IEEE INFOCOM, 2013. 809–817