Enhanced Identification of Sensitive User Inputs in Mobile Applications

Mashael Aldayel and Mohammad Alhussain
Information System Department, College of Computer & Information Sciences, King Saud University, Riyadh, Saudi Arabia
{maldayel, alhussein}@ksu.edu.sa

Keywords: User Input, Sensitive, Privacy Leak, Disclosure, Data-flaw Analysis.

Abstract: While smartphones and its apps have a fundamental role in our lives, privacy is a critical issue. With the constantly growth of mobile applications, smartphones are now capable of satisfying all kinds of users' needs, dealing with more private and restricted tasks by the users and gain more access to sensitive and private data. This issue is even worse with the current absence of methods that can notify users of possibly dangerous privacy leaks in mobile apps without disturbing users with apps' legitimate privacy exposes. Previous mobile privacy disclosure approaches are mostly concentrated on well-defined sources controlled by smartphones. They do not cover all sensitive data associated with users' privacy. Also, they cannot filter out legitimate privacy disclosures that are commonly found in detection results and consecutively conceal true threats. Sensitive user inputs through UI (User Interface), are the dominant type of sensitive data that has been almost ignored. Defending this kind of information cannot be accomplished automatically using existing techniques because it necessitates understanding of user inputs' semantics in apps, before identifying its positions. Moreover, eliminating legitimate privacy disclosures necessaries tracking of the related app data flows form these users' inputs to various sinks. Such tracking will help to determine if this privacy disclosure is valid or suspicious. To address all these important issues, we propose an enhanced approach for detecting users' inputs privacy disclosures that are truly suspicious.

1 INTRODUCTION

Smartphones are the main type of end-user devices as reflected in the increasing sold units over traditional computers. In the last few years, the use of smartphones for both corporate and personal goals has increased significantly. There are 7,377 billion mobile users around the world, which is equivalent to 99.7% of the world's population, as estimated by the International Telecommunication Union in 2016 (ITU 2016). Smartphone security is an emerging area of rising importance and cumulative requirements, nevertheless its relative weaknesses regard protecting a user's data privacy (Khan et al. 2015). Even though the mobile devices' corporations have considered security issue in designing their smartphones, using mobile applications from the internet creates complicated difficulties in handling threats and vulnerabilities that endangers a user's data privacy.

Many diverse applications with widespread range of purposes can be accessed online from application stores for each mobile device. With the constantly growth of these applications, smartphones are now capable of satisfying all kinds of users' needs, dealing with more private and restricted tasks by the users and gain more access to sensitive and private data. This issue is even worse with the current absence of methods that can notify users of possibly dangerous privacy leaks in mobile apps without disturbing users with apps' legitimate privacy exposes. Consequently, this rise many concerns regard the costs of failure in securing user's privacy in smartphones (i.e., transmitting it to remote individuals or broadcasting it). Although their wide abilities in satisfying users' needs, protecting the privacy of user data in smartphones becomes an essential issue and get more attention in mobile security researches. Many approaches have been developed to detect potential information leaks and privacy disclosures using (1) the phone OS and framework APIs, such as location, contact, calendar…etc. (2) data-flows analysis mechanisms either dynamically or statically (Huang et al. 2015)(Lu et al. 2015).

The former approach is based on access control (permission) techniques that implements fine-grained security strategies to deal with private user data on a

Aldayel, M. and Alhussain, M.
Enhanced Identification of Sensitive User Inputs in Mobile Applications.
DOI: 10.5220/0006238405060515
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 506-515
ISBN: 978-989-758-209-7

mobile system. These data are provided by OS and can be secured by utilizing related data-access APIs to mark the security tags for the data. These permission techniques relay on users' agreement to access their private resources without offering enough information that can help them in their decision. Moreover, success of such techniques mainly assumes a users' perception and understanding on the app privacy effects. In fact, this hypothesis is not accurate (Huang et al. 2015)(Lu et al. 2015).

The latter approach is based on evaluating data-flows in mobile apps in order to automatically reveal privacy leaks (Nan et al. 2015). It tracks the source of the privacy disclosures of user data to various sinks which help users to make further informed judgments. Furthermore, it used for identification of apps vulnerabilities that may accidentally expose such sensitive user data to public or to the attacker (Huang et al. 2015).

Despite the importance of the previous approaches in protecting sensitive data, they do not cover all sensitive data associated with users' privacy. Sensitive user inputs are the dominant type of sensitive data that has been almost ignored. It refers to the sensitive content entered by users in a mobile app through the User Interface (UI) such as financials, credentials and medical information. Recent research (Nan et al. 2015) have found that among 17,425 top Google-Play apps, 35.46% require users to enter their confidential information. Defending this kind of information cannot be accomplished automatically using existing techniques because it necessitates understanding of user inputs' semantics in apps, before identifying its positions. This is challenging because of the content's lankness of fixed constructions that obstruct recovering them without semantics exploration (Huang et al. 2015) (Nan et al. 2015).

After detecting the sensitivity of the user inputs data, it is necessary to track the related app data-flows of privacy disclosures for that user inputs to various sinks. Such tracking will help to determine if this privacy disclosure is valid or suspicious. Another study (Lu et al. 2015) proves that more than 67% of app privacy exposes discovered using traditional approaches are actually valid (i.e., required in the basic functions of the apps). For example, GPS apps need to send user's current location to remote servers to show it on the map. Also, such tracking can help to recognize the vulnerabilities in the apps that may accidentally expose sensitive user inputs to public, to the attacker or to third-party ad reference library. Nevertheless, previous data-flow analysis approaches are incapable of determining the validity of discovered data-flow privacy leak. This weakness regularly leads to overestimated privacy exposures covering numerous false warnings (e.g., benign or functional privacy exposures). Since these warnings are not true user privacy violations, they cause distractions and interruptions for users. Alerting all privacy exposures of sensitive user inputs to the user will seriously annoy him and reduce the usability (Lu et al. 2015).

To sum up, guarding users' privacy along with both deliberate and unintended disclosures requires: (1) automatic recognition of the private sensitive content the user enters into smartphone apps and based on it (2) distinguish only suspicious data flows of privacy disclosures for that user inputs to various sinks. This paper proposes such approach which can ensure appropriate protection that matches with users' confidence and anticipations (Huang et al. 2015) (Nan et al. 2015).

In this paper, we present some application-based threads and propose an enhanced framework architecture to detect truly suspicious user-inputs privacy leaks in Android apps that harmfully impact end-users. Malware detection is described in existing works (Gorla et al. 2014), (Pandita et al. 2013) and is out of scope of this paper. Recent research (Khan et al. 2015) indicates wide ranges of threats and vulnerabilities on smartphones that cyber criminals are now concentrating more and more on them. Attackers utilize the huge diversity of smartphone applications on the internet in order to terminate safety mechanisms, increase threats and expand vulnerabilities. This trend emphasizes the need for further security enhancements on application level of smartphones, as well as more flexible, better security solutions and policies for mobile applications. Some important mobile applications-based threats to user privacy are presented as follows.

2 APPLICATION-BASED THREATS

Adversaries can take advantage of the weaknesses inside current protection techniques in order to steal sensitive user inputs. For instance, fake banking apps can be designed with very similarity UIs for stealing user's financial credentials. Moreover, some developers unintentionally reveal sensitive user data. For example, eavesdropping attack is exposed to the apps that transmits content as plaintext across public networks. (Nan et al. 2015). Therefore, privacy

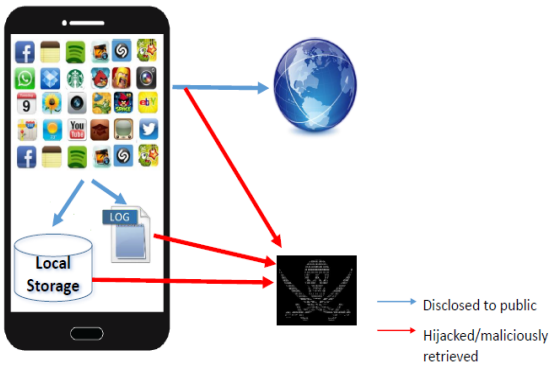

Figure 1: Application-Based Threads.

violations can have resulted from mobile apps that store or transmit sensitive user input without any protection (figure 1).

Mobile applications, either benign or malicious, can be downloaded online which introduces numerous security threats on both types; applications explicitly designed to be malicious as well as useful applications that can be misused for malicious drives. The greatest concern rises when the application is malicious, fake or bogus (Khan et al. 2015)(Sujithra 2012). Accordingly, applications-based threats can be classified into benign app threads, malicious app threads or even combination of them.

2.1 Benign App Threads

Although, mobile application can be found benign, useful and valuable, it can disclose the user privacy accidentally or be exploited for malicious purposes. Applications that reveal privacy threats are not essentially malicious, but collects or uses more sensitive information that users are unwilling to expose to the public. For example, the global positioning system (GPS) can provide information about any location a user goes to. Any leak of this information can lead to serious problems. Additionally, benign applications may contain software faults or vulnerabilities which can be utilized for malicious intent. These permit attackers to access sensitive information, accomplish unwanted actions, halt a service from running correctly, and consequently download further applications without user agreement. Mobile app vulnerability is a safety disclosure that outcomes from an app weakness that the application developer did not anticipate to present and will fix after it is revealed. Vulnerability in mobile apps contains three elements; app flaw or weakness, accessibility to it, and malicious mechanism able of exploits that the flaw and attack it (Khan et al. 2015) (Sujithra 2012).

2.2 Malicious App Threads

Malware, short of malicious applications is any type of aggressive, intrusive, or irritating software (e.g. Trojan, rootkit, virus, worm…etc.) aimed to achieve malicious actions after installation in a user's smartphone without the user's consent. It can be used to gather private and personal information from user smartphones that could result in theft or financial scam. Spyware is another kind of malware that aims to accumulate personal and private information without a user's knowledge or agreement. Targeted data in spyware usually contains phone contact list, call and browser history, location, and camera images (Khan et al. 2015)(Sujithra 2012)(La Polla et al. 2013).

3 RELATED WORK

In order to detect sensitive user inputs in smartphones' applications, analyzing the context and semantics of UIs is required (Nan et al. 2015). A direct approach is to consider all user inputs are sensitive (Arzt et al. 2014), which is obviously will cause overload, more false positives and impartially poor results due to combining sensitive user inputs that we emphasis on with a plenty of additional sources we do not care.

To the best of our knowledge, there is no in-depth study of user inputs privacy discourses on Android phones that alert only suspicious discourses. Thus, the discussion of related work is divided into three types: taint analysis, text analysis and privacy source detection approaches.

3.1 Taint Analysis approaches

There are a lot of researches on identifying, understanding and evaluating privacy disclosures which focus on predefined sensitive data sources on several mobile OS. From technical point of view, there are two main approaches for identifying data-flows of privacy disclosures within an app; static (Arzt et al. 2014)(Huang et al. 2014)(Fahl et al. 2012), (Gibler et al. 2012) and dynamic taint analysis.

The first approach is static taint analysis which emphases on finding the potential privacy leak route based on tracking analysis and program slicing. It can analyze large numbers of applications. Besides, they can detect malware or vulnerabilities in Android

apps. FlowDroid (Arzt et al. 2014) is one of pioneer works in static analysis that only offer alternatives to taint all user inputs as sensitive sources and offers a limited form of sensitive input fields (password fields). AsDroid (Huang et al. 2014) includes static analysis tool that link the app behaviour from API call sites to a top level function through related UI to detect malicious behaviour. Mallodroid (Fahl et al. 2012) is a static analysis tool aimed to detect apps that potentially use SSL/TLS inadequately or incorrectly and thus are potentially vulnerable to Man-in-the-Middle attacks. The static analysis can separate input data from app UI and finds vulnerable elements in the program code of an application (Nan et al. 2015). Using this approach in detecting sensitive user inputs will lead to mark all the user inputs are sensitive which is not acceptable because it introduces a lot of false positives and cannot differentiate if the operations are intended or not by users due to the lacking of user intention and context information. Furthermore, as ssmartphones have limited ability in handling and storing data, protecting all user inputs will take more time and resources on smartphones which is obviously overkill (Yang et al. 2013).

The second approach is dynamic taint analysis such as TaintDroid (Enck et al. 2010) and AppIntent (Yang et al. 2013) which can track sensitive data at runtime. In another word, they can capture the context of runtime environment such as configuration variables and user input. More significantly, code may be obscured to hinder static analysis, either deliberately or accidentally. The issues are either reducing overhead in deployments, or increasing code coverage in dynamic testing (Rastogi et al. 2013)(Yang et al. 2013). TaintDroid (Enck et al. 2010) is one of the innovator researches which uses dynamic taint analysis to identify privacy leaks by tracking the flow of sensitive data over third-party applications. AppIntent (Yang et al. 2013) is another dynamic program analysis aimed to discriminate user-intended sensitive data transmissions from other transmissions to improve identifying privacy leaks. However, dynamic taint approaches cannot be applied in marketplaces for detecting privacy leaks in marketplaces because they report a leak only if such risky transmission occurs in the execution time (Rastogi et al. 2013).

However, most research works in such dynamic approaches often (Arzt et al. 2014), (Enck et al. 2010), (Yang et al. 2013), (Xu et al. 2012) manually identify the contents that need protection in the apps. They depend on users, developers or designer i.e. necessitates human interference which is not suitable for evaluating privacy threads on large-scale apps' analysis. Moreover, they cannot differentiate if the operations are valid or not from systematic point of view without relying on human factor.

Moreover, all the above-mentioned taint analysis approaches do not try to analyze or to understand whether the detected privacy disclosures has any relationship with the app's functionality. Instead, the security experts or the users have to understand such relationship manually. Our approach utilizes peer voting mechanism (Lu et al. 2015) to identify suspicious privacy disclosures and benefit the users in selecting more traditional and safe applications.

3.2 Text Analysis in Android Apps

Diverse researches exploit UI text analysis for various security goals. AsDroid (Huang et al. 2014) detects tricky behaviors in Android applications by checking conflicts between program behavior and anticipated behavior through analyzing UI textual. However, their UI analysis is based on few textual keywords, which could be insufficient. CHABADA (Gorla et al. 2014) checks application behaviors against application descriptions through Natural Language Processing (NLP) approach called "topic modelling". It aimed to detect malicious behaviours in Android applications by collecting applications that are alike to each other in their textual descriptions. Then detects application which used APIs differently from the regular use of the APIs in the same collection. CHABADA cannot exactly detect privacy disclosures as it only recognizes sensitive APIs without tracking the data-flows among their sources and sinks. Our approach identifies suspicious privacy disclosures and benefit the users in selecting more traditional and safe applications.

Whyper (Pandita et al. 2013) take advantages of a NLP technique (Stanford Parser) to check if the application description justifies the permission usage. Although our approach provides an enhanced complementary approach in assessing the validity of privacy disclosure, we might possibly leverage their methods to produce more comprehensive privacy-related texts for identifying user inputs data.

3.3 Privacy Source Detection

Some studies concentrate on mapping between Android permissions and APIs such as PScout (Wain et al. 2012). It gets the specific permissions from the Android OS source code to accomplish permission-to-API mapping through static analysis. Cashtags (Mitchell et al. 2015) protects users sensitive inputs that are displayed on the screen by and replacing them

with non-sensitive data elements. It aims to prevent shoulder surfing threats but it simply gathers all sensitive data in one data repository without any protection techniques.

SUPOR (Huang et al. 2015) and UIpicker (Nan et al. 2015) are the most related works addressing the problem of our paper. They intend to automatically identify sensitive user inputs in android application. They present different approaches to cover user inputs that are not protected with smartphone OS and APIs access permissions. UIpicker take advantage of supervised learning approach to train a classifier based on the extracted features from the UI elements' texts and layout descriptions. It furthermore considers the texts of the sibling elements in the layout file.

SUPOR detects sensitive user inputs using UI rendering, geometrical layout analysis and NLP techniques. Since UIPicker rely on sibling elements in the layout file to associate the description text for a UI field, this could simply include unrelated texts as features. SUPOR eliminate such problems by selecting only the text labels that are actually close to input fields in the monitor, imitating how users look at the UI, and uses the labels' text to determine the sensitiveness of the input fields. SUPOR focuses on a definite type of UI elements (EditText) whereas UIPicker includes more UI elements such as dropdown lists (Spinners), RadioButton, CheckBox besides (EditText). However, UIPicker techniques in extracting privacy-related texts could complement

Table 1: Comparison between Sensitive Users' Inputs Detection Systems.

	SUPOR	UIPicker
Data scope	Identity, account, personal, credential, financial and health information	Account credentials and user Profiles, location (plain text format) and financial data
Contribution	Propose SUPOR, first static mobile app analysis tool for detecting sensitive user inputs in Android apps that are not permission protected, and evaluate its performance.	Measure the spreading of private user input data based on 17,425 applications, propose UIPicker framwork for automatic identification of user inputs data in large scale within Android apps and offer runtime security improvement based on UIPicker.
Benchmark	First research in this field	First large-scale analysis of apps' privacy risks.
Result	Detect 355 apps with false positive rate 8.7% from 16,000 popular Android apps	Detect 6,179 (35.46%) of 17,425 popular Android apps
Manual Evaluation	Based on 40 randomly selected apps (Precision: 97.3% & Recall: 97.3%)	Based on 200 randomly selected apps (Precision: 93.6% & Recall: 90.1%)
System Components	Layout analysis, UI sensitiveness analysis and UI – code Binding.	Pre-processing, Privacy text analysis, Classifier and Program behaviour filtering
Strength	• Boarder detection converge in user text fields using synonyms and Google Translate. • Better scalability (shorter time analysis)	• Boarder detection converge in many types of user inputs. • Secure user data inputs by preventing sending them as plaintext and checking SSL risks using integration with MalloDroid • Dynamic generation of privacy features to expand private semantic data using a few privacy-related seeds.
Weaknesses	• Fixed privacy related dataset besides excluding address from their dataset. • Insufficient context to identify sensitive keywords • Deal with one specific type of UI elements (EditText) • Cannot differentiate between valid and suspicious privacy disclosers	• Limitation in privacy feature extraction mechanisms that introduces false positive and false negative results. • Cannot differentiate between valid and suspicious privacy disclosers

NLP techniques in SUPOR for enhancing the construction of keyword dataset. Table 1 provides a comparison between these related approaches. However, SUPOR and UIPicker cannot distinguish between valid and suspicious privacy disclosers which we focus on in this paper.

Several research works (Arzt et al. 2014), (Enck et al. 2010), (Yang et al. 2013)(Han et al. 2013)(Gibler et al. 2012), (Mitchell et al. 2015) emphasis on smartphone privacy disclosure of sensitive data sources. Our approach identifies legitimate (valid) sensitive user inputs and may enable most of the current privacy researches to be functional on sensitive user inputs. Therefore, our research compliments the existing works.

4 PROBLEM STATEMENT

In this section, we first display a motivating example of users' sensitive inputs in a UI screen, then we examine challenges in recognizing such data and clarify our data scope of users' input.

4.1 Motivating Example

Figure 2 displays a UI of mobile app that hold some vital sensitive information that is often required in a many shopping apps. The user has to input the credit card records to complete the payment process. As the developers might be unaware of the possible threats on the exposes of such sensitive information, the credit card credentials are sent in plain text over an insecure channel which accidentally compromises users' privacy. Several apps in smartphones necessitates sensitive information for many functional goals. This information is mostly private and personal which users are not comfortable in revealing them without proper security.

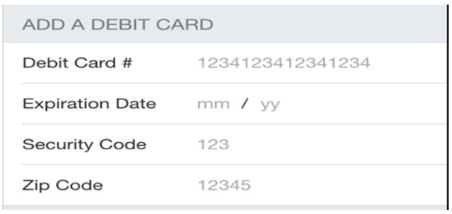

Figure 2: Example of sensitive user inputs in UI.

Sensitive user inputs are the dominant type of sensitive data that has been almost ignored in previous researches. Even though user inputs can be greatly security-sensitive and have risky consequences once it is exposed, little has been done so far to detect them at a large-scale (Nan et al. 2015). The main issue here is how to automatically discriminate sensitive user inputs from other inputs in UIs. This is challenging because of the content's lankness of fixed constructions that obstruct recovering them without semantics exploration. Another issue is to determine if the privacy disclosure of the identified sensitive user inputs is valid or suspicious.

Previous privacy disclosure analysis approaches often limit the options to mark all user inputs as sensitive sources and cannot determine the validity of discovered data-flow privacy leak i.e. whether it is caused by functional privacy exposures in benign app or not. Exploring in this method would produce absolutely bad results dues to combining sensitive user inputs that we emphasis on with a plenty of additional sources we do not care and due to combining valid and invalid data-flow privacy leaks. Furthermore, such approaches obviously will cause more overload; more false positives in detecting theses sensitive user inputs and overestimated privacy exposures covering numerous false warnings (e.g., benign or functional privacy exposures).

Similar problem also occurs in runtime user input protection. For example, if an app is trying to transmit sensitive user inputs outside the mobile, an appropriate defence method is to inform users to prevent such transmitting. It is overkill to alarm users of all inputs whether it is sensitive or not, including their valid and suspicious privacy exposures. Since these warnings are not true user privacy violations, they cause distractions and interruptions which will greatly annoy users. Moreover, this will reduce the usability since (1) many usual inputs do not require to be treated as sensitive inputs and (2) most privacy exposures of sensitive user inputs are valid and do not require to annoy user with it.

4.2 Challenges

Sensitive users' input data can be simply identified by human but it is very challenging for the machine to automatically recognize this data on large-scale using existing approaches. These sensitive input data cannot be recognized during runtime checking because they are extremely unstructured which make it very difficult to use predefined expressions for matching when users input them. Moreover, private user inputs, similar to any usual inputs, are spread in several UIs in a particular app. Mostly such UIs require login or composite trigger situations which makes it very challenging for automatic testing tools to navigate such interfaces comprehensively without manual interference. It also impractical to detect these data by static analysis approaches because, within

program code's, there is no clear difference between sensitive user input and other inputs. Apps accept all these input data, then transferred out or kept in local storage in an identical way, which makes it hard to differentiate them using static analysis approaches (Nan et al. 2015).

Another issue is to determine if the privacy disclosure of the identified sensitive user inputs is valid or not. This requires to track the related data flows of these inputs privacy disclosures to various sinks which help users make further informed judgments. Many existing privacy exposes detection approaches were designed to expose apps' activities that trace confidential data (source) to a confidential channel (sink), but this might be not expressive enough to users or developers (Lu et al. 2015). Current taint analysis approaches alert users regard any app privacy violations even if it is part of the app's core functions. Thus, whenever evaluating a valid navigation app or a doubtful calculator app with a 3rd-party library tracing users, current approaches alert users with the identical kind of reports that show possible privacy exposes.

This paper proposes enhanced approach for suspicious privacy disclosure detection of users' inputs in smartphones. which includes: (1) automatic recognition of the private sensitive content the user enters into smartphone apps and based on it (2) distinguish only suspicious data flows of privacy disclosures for that user inputs to various sinks. Our goal is primarily to detect sensitive data in benign apps. The results can be also used for security analysis or protection of such data. Instead of focusing on malicious privacy leakages that intentionally avoid detection, our approach targets efficient large-scale examination of apps, some of which might not appreciate user privacy and aggressively misuse user privacy in exchange of profits. Malware detection is out of scope of this paper. There are many works for detecting malware apps (Gorla et al. 2014; Pandita et al. 2013), however, our approach do not deal with malicious apps that deliberately avoid our analysis (Nan et al. 2015), e.g., malware that creates its layout dynamically or uses images as labels to prompt users to input their sensitive data.

Our approach is designed to discover concealed privacy leakage user-inputs data flows in a specific Android app that could not be explained by the common functions of the app in order to maintain low false positive rate.

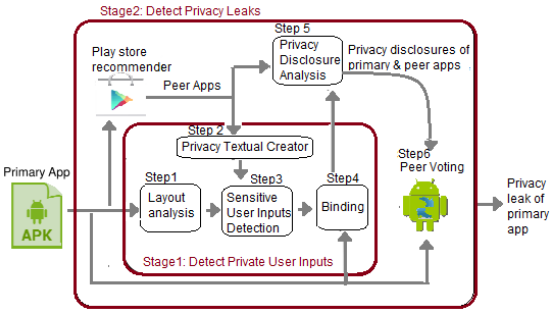

Figure 3: Overall Architecture of our Approach.

5 IDENTIFICATION APPROACH

In this section, we provide a summary of our approach and describe the key units that we propose in our identification framework.

5.1 Overview

Figure 3 shows the overall workflow of our approach. It is made up of six key units to identify suspicious leaks of data-flows which contain sensitive users' inputs. These key units are: layout analysis, privacy textual creator, sensitive users' inputs detection, privacy disclosure analysis, binding and peer voting. They can be split into two stages: private users' inputs identification and privacy leaks recognition.

In the first stage (Step 1,2,3,4), our approach takes a set of apps to identify sensitive user inputs from their textual semantics and associated field location in GUI. In the second stage (Step 5,6), peer voted mechanism (Lu et al. 2015) scans the resulted privacy disclosures from both the primary app and peers apps (Step 5) to identify truly suspicious privacy disclosures (Step 6).

During the private users' inputs identification stage, the layout analysis unit takes an app in a form of APK file, deconstructs the layout files within this APK file, and delivers the layout files holding input fields. Then sensitive users' inputs detection unit combines text labels with input fields inside the resulted layout files in order to prepare for the discovery of private user inputs. Privacy textual creator extracts privacy textual elements from a subset of specific layouts in peer (similar) apps to generate keywords dataset. Based on the resulted sensitive keywords (from step 2), the words of the text labels (from step 3) is compared to determine the privacy of the input fields. The binding unit then examines the primary app's source code to find the variables associated with the value of the sensitive

input fields. After variable binding, the second stage initiates the process of privacy leak recognition.

Throughout the privacy leak recognition stage, existing research efforts in detecting privacy disclosure (step5) of smartphones' sensitive data sources can be utilized for sensitive user inputs. Thus privacy disclosure analysis generates data-flows report of users' inputs privacy disclosures using any static taint analysis tool such as flowDroid (Arzt et al. 2014). After that, peer voted mechanism detects the exposes of sensitive user inputs that are truly suspicious. Next we describe each unit in detail.

5.2 Architecture Units

This section describes the key units in our identification architecture that detects truly suspicious user inputs privacy leaks in Android apps that harmfully impact end-users.

5.2.1 Layout Analysis

This unit aims to provide user inputs fields of a primary app, and obtain the details of input fields such as types, hints, and coordinates. Such details are used for the users' inputs detection unit (step 3) for detecting sensitiveness of user inputs fields. Based on the user perception, text label must be physically close to the input field in the UI. Thus layout analysis unit extracts the users' inputs fields from layout files imitating how users see the UI. Then it computes the distances between text labels and input fields based on their coordinates to find the finest expressive text label for every input field (Huang et al. 2015).

5.2.2 Privacy Textual Creator

Our approach leverages NLP methodologies used in related work (Huang et al. 2015) (Nan et al. 2015) to enhance the privacy textual keywords dataset construction method. As a start, all texts in the resource files of all similar (peer) apps, including the primary one, is gathered in a form of a list of noun phrases. Then we extract words in this list to create a list of words and sort both lists based on their frequency. We can adapt Stanford parser as NLP technique to extract nouns and noun phrases from the top frequent text. As privacy textual words show together in specific layouts, they are semantically related to users' sensitive data. Such layouts can be utilized to extract privacy textual keywords. Therefore, another NLP technique (chi-Square test) extracts sensitive keywords from a subset of specific layouts. It tests whether a particular word in the user-input layout is sensitive or not according to its occurrences in keyword dataset. Also, we can enlarge the keyword dataset by refining the known words and leverage WordNet(Princeton 2010) as well as Google Translate to include synonyms and translated keywords in several language. The outcome of this unit is privacy textual dataset that contain sensitive keywords.

5.2.3 Sensitive Users' Inputs Detection

The privacy determination of an input field is based on three levels: its type, hint or text label that were collected in the layout analysis (step 1). At first, the type of user inputs field is tested to find whether it is a password type. If not, the second test compares the hint of input field with the privacy-related keywords dataset to check if is sensitive or not. If not, the third test compares text label of input field with the privacy-related keywords dataset to check if is sensitive or not but this test requires finding the related text label that expresses the goal of the input field.

We can leverage SUPOR correlation scores algorithm to associate a text label to each input field based on their distances and relative positions (Huang et al. 2015). The outcome of this unit is a list of sensitive user inputs fields in the primary app.

5.2.4 Binding

This step aims to bind the sensitive input fields that were detected earlier with related variables that store their values in the primary application source code. The binding unit do this using context-sensitive analysis which identifies each sensitive input field with a contextual ID. In order to distinguish the variables' names in many layouts within the same application, each contextual ID contains layout ID and input field ID to reference any input field on application level. Such binding mechanism enables the initiation of privacy leak recognition (Huang et al. 2015) (Nan et al. 2015).

5.2.5 Privacy Disclosure Analysis

To assist users in making more informed decisions, this unit was built to discover apps' behaviours that propagate user sensitive input data (i.e., source) to a sensitive channel (i.e., sink). The privacy disclosure analysis unit generates privacy disclosure reports for all potential users' inputs flows inside a specific app as well as the related similar apps since they all have similar privacy disclosures. These flows include both valid and invalid privacy leakages for similar apps which are included to expand the scope of detecting sensitive data flows for the next step 6. In order to achieve high coverage and scalability, we will use

static taint analysis tool which primarily aim to discover all data-flows from the sensitive users' inputs data (sources) to the defined channels (sinks). Such static analysis tool can achieve high performance as well as high coverage capacity to detect more complete privacy disclosure data-flows. The outcome of this unit includes all kinds of data-flow privacy leakages that originate from users' inputs fields in similar mobile applications.

5.2.6 Peer Voting

After detecting privacy disclosures for a given primary app, suspicious privacy leakages should be distinguished using peer voting mechanism. Since applications with the identical or similar functionality (called peer apps) have similar privacy disclosures, this insight formulates the peer voting mechanism. The peer voting unit identifies the legitimacy of a particular privacy disclosure spotted in an app (called primary app) by asking its peer apps regard their profiles of privacy disclosures. Peer apps are expressed from the viewpoint of users, which covers the apps that are operational alike with the primary app. Thus, users can consider a peer app as a substitute of the primary app. In order to implement this unit, we need (1) a method to find operational similar apps as well as (2) an automated approach to distinguish the extremely suspicious privacy disclosures of user inputs (Lu et al. 2016).

Before initiating peer voting mechanism, the peer apps should be selected first. There are several opportunities to do this such as keyword-based similarity, classification based on category or permissions, or recommendation systems. Recommendation systems can be either a fundamental property of app stores or a standalone service provided by third parties. Such systems return a list of apps that are operational similar or associated to the primary app. The resulted list can contain apps that are frequently installed or used together. To take advantage of users' experience in using apps, we choose to utilize existing app recommendation systems such the one delivered by Google Play, and occupy a NLP method named semantic similarity to enhance refining similar apps. The peer apps generated from the existing Google Play recommender can produce similar apps lists drew from the users' experience i.e. user views and installation patterns. Although the recommendation details are ambiguous, the resulted ranking lists offers useful selection regard choosing functionally similar apps. Then we filter out irrelevant apps (minor outlier) using same-category rule between a primary app and its peer apps and using semantic similarity on app descriptions which excludes apps with long semantical distance from the other peers (Lu et al. 2015).

Then peer voting mechanism takes peer apps as input and decides the validity of a specific privacy disclosure in the primary app by checking the privacy disclosures of the peer apps. Each peer app can vote for it: each peer app is checked to detect whether it has this privacy disclosure or not. If yes, the peer votes for 1; if not, vote for 0. If the majority of the peers has the same privacy disclosure, it is counted as an essential for the core functions of primary apps. Otherwise, the privacy disclosure is possible to be caused by replaceable or unpredicted element of the primary app, and consequently will be considered as a suspicious privacy leakage. However, privacy leakage should be measured to decide to what extent it can be valid. This validity can be modelled in equation 1 where Votes# is the number of peer apps who has the same privacy disclosure and Peers# in the number of peer apps for a specific primary app (Lu et al. 2015).

$$\text{Privacy Validity} = \text{Votes\#} / \text{Peers\#} \quad (1)$$

The privacy validity represents "how likely the privacy disclosure is legitimate in the primary app" and should not exceed the validity threshold which is usually set around 2% (Lu et al. 2015) (Lu et al. 2016). If the primary app compared to the peer does not agree with the voting, then this unit reports a probable violation of use input privacy.

6 CONCLUSION

In this paper, we present some application-based threads and propose an enhanced framework architecture to detect truly suspicious user-inputs privacy leaks in Android apps that harmfully impact end-users. We plan to implement this framework as future work. Sensitive user inputs detection is important to improve protection but mostly ignored as a sensitive source in mobile apps. Our approach conducts specialized privacy leaks detection considering user inputs data source. We leverage NLP methodologies to enhance the privacy textual keywords dataset construction method. It can achieve significant improvements to static analysis work in terms of validity. By excluding the valid and legitimate disclosures, our approach exposes only suspicious privacy leaks that cannot be linked with apps' core functions. As a result, privacy leaks detection rate can be greatly increased, and simultaneously, the manually works needed from security analysts or end-users will be decreased.

REFERENCES

Arzt, S. et al., 2014. FlowDroid : Precise Context , Flow , Field , Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *PLDI '14 Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp.259–269.

Enck, W. et al., 2010. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *Osdi '10*, 49, pp.1–6.

Fahl, S. et al., 2012. Why eve and mallory love android: an analysis of android SSL (in) security. *Proc. of ACM CCS*, pp.50–61.

Gibler, C. et al., 2012. AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7344 LNCS, pp.291–307.

Gorla, A. et al., 2014. Checking app behavior against app descriptions. *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, pp.1025–1035. Available at: http://dl.acm.org/citation.cfm?doid=2568225.2568276.

Han, J. et al., 2013. Comparing Mobile Privacy Protection through Cross-Platform Applications. *Network and Distributed System Security Symposium*, pp.1–15. Available at: http://www.liaiqin.com/hanjin/%5Cnpapers3://publication/uuid/EDE08F21-0175-4B99-B31B-86FC339DAFB4.

Huang, J. et al., 2014. AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. *ICSE 2014: Proceedings of the 36th International Conference on Software Engineering*, (March). Available at: https://ece.uwaterloo.ca/~lintan/publications/asdroid-icse14.pdf.

Huang, J. et al., 2015. SUPOR: Precise and Scalable Sensitive User Input Detection for Android Apps. *24th USENIX Security Symposium (USENIX Security 15)*, pp.977–992. Available at: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/huang.

ITU, 2016. ITU Statistics. Available at: http://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx.

Khan, J., Abbas, H. & Al-Muhtadi, J., 2015. Survey on Mobile User's Data Privacy Threats and Defense Mechanisms. *Procedia Computer Science*, 56(Csdi), pp.376–383. Available at: http://www.sciencedirect.com/science/article/pii/S1877050915017044.

Lu, K. et al., 2015. Checking More and Alerting Less: Detecting Privacy Leakages via Enhanced Data-flow Analysis and Peer Voting. *Symposium on Network and Distributed System Security (NDSS)*.

Lu, K. et al., 2016. DuLeak: A Scalable App Engine for High-Impact Privacy Leaks. , p.16. Available at: https://www.google.com/patents/US9245125.

Mitchell, M., Wang, A.-I.A. & Reiher, P., 2015. Cashtags: Protecting the Input and Display of Sensitive Data. *24th USENIX Security Symposium (USENIX Security 15)*, pp.961–976. Available at: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/mitchell.

Nan, Y. et al., 2015. UIPicker : User-Input Privacy Identification in Mobile Applications This paper is included in the Proceedings of the.

Pandita, R. et al., 2013. Whyper: Towards Automating Risk Assessment of Mobile Applications. *USENIX Security Symposium*.

La Polla, M., Martinelli, F. & Sgandurra, D., 2013. A Survey on Security for Mobile Devices. *IEEE Communications Surveys & Tutorials*, 15(1), pp.446–471.

Princeton, U., 2010. WordNet:An Electronic Lexical Database. *Princeton University*. Available at: http://wordnet.princeton.edu.

Rastogi, V., Chen, Y. & Enck, W., 2013. AppsPlayground : Automatic Security Analysis of Smartphone Applications. *CODASPY '13 (3rd ACM conference on Data and Application Security and Privac)*, pp.209–220.

Sujithra, M., 2012. Mobile Device Security : A Survey on Mobile Device Threats , Vulnerabilities and their Defensive Mechanism. , 56(14), pp.24–29.

Wain, K. et al., 2012. PScout : Analyzing the Android Permission Specification. *CCS '12 Proceedings of the 2012 ACM conference on Computer and communications security*, pp.217–228. Available at: http://www.eecg.toronto.edu/~lie/papers/PScout-CCS2012-web.pdf%5Cnhttp://dl.acm.org/citation.cfm?id=2382222.

Xu, R. et al., 2012. Aurasium: Practical Policy Enforcement for Android Applications. *Proceedings of the 21st USENIX conference ...*, p.27. Available at: https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final60.pdf%5Cnhttp://dl.acm.org/citation.cfm?id=2362793.2362820.

Yang, Z. et al., 2013. AppIntent: analyzing sensitive data transmission in android for privacy leakage detection. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pp.1043–1054. Available at: http://dl.acm.org/citation.cfm?doid=2508859.2516676.