Received April 30, 2020, accepted May 20, 2020, date of publication May 25, 2020, date of current version June 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2997335

Recommending Security Requirements for the Development of Android Applications Based on Sensitive APIs

YUZHOU LIU[1,2], LEI LIU[1,3], HUAXIAO LIU[1,3], SHANQUAN GAO[1,3], AND GUOHANG SONG[1,3]

[1]College of Computer Science and Technology, Jilin University, Changchun 130012, China
[2]College of Electronic Science and Engineering, Jilin University, Changchun 130012, China
[3]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, China

Corresponding author: Huaxiao Liu (liuhuaxiao@jlu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1003103, and in part by the Natural Science Research Foundation of Jilin Province of China under Grant 20190201193JC.

ABSTRACT App stores allow anyone to sell his products to millions of potential users. However, limited by the resources and time, some developers often focus on the functionalities of their Apps without well-rounded considering security problems, which are more and more important for a successful product. In this paper, we propose an approach to help developers elicit security requirements by recommending related information gained from existing Apps in the marketplace. Firstly, we construct a feature framework to summarize functionalities of Apps by mining their descriptions with the method proposed in our previous work. Then, the sensitive APIs used in these Apps are extracted from their APK files and mapped with App features. Finally, we establish relationships between permissions and functionalities by taking sensitive APIs as a bridge, and design a recommendation framework to show information according to developers' demands from two aspects: the security requirements for the whole App and the ones for the given functionality. We evaluate our approach with 580 Apps from 5 categories on Google Play. The results confirm the usefulness of our approach, especially it can help new developers without experience initialize the security requirements and give mature developers supplementary information to elicit security requirements completely.

INDEX TERMS Requirements engineering, sensitive permissions, APIs analysis, information recommendation.

I. INTRODUCTION

Today, the appearance of App (application) stores brings a new era for the software development, anyone, whether a big or small company or even an individual developer, can access these distribution infrastructures to sell the products to millions of potential users [1], [2]. This is a good phenomenon for promoting the industry of Apps, but also leading potential risk that some products are developed without well-rounded considering the security problems [3]–[5]. As people trends to do almost everything at their fingertips [6], especially sensitive activities such as bank transfers and e-business, an insecurity App may disclosure private information of users and result in serious consequences. Thus, developers are required to give not only good functionalities but also safe Apps [7].

Requirements are the first stage of the software development and it is a good time to consider the security problems of Apps [8]. This is not only for reducing the cost of repairing the products, but also because once users believe an App insecure they would not trust it again, such opinions are almost unchangeable. However, eliciting security requirements is not an easy job for developers because they may not have enough background knowledge to predict the risks in their products, especially for the small companies and individuals, and the competitive App market does not give them enough time to learn from scratch. Luckily, there always exists products sharing similar functionalities with the App to be developed

The associate editor coordinating the review of this manuscript and approving it for publication was Shuiguang Deng.

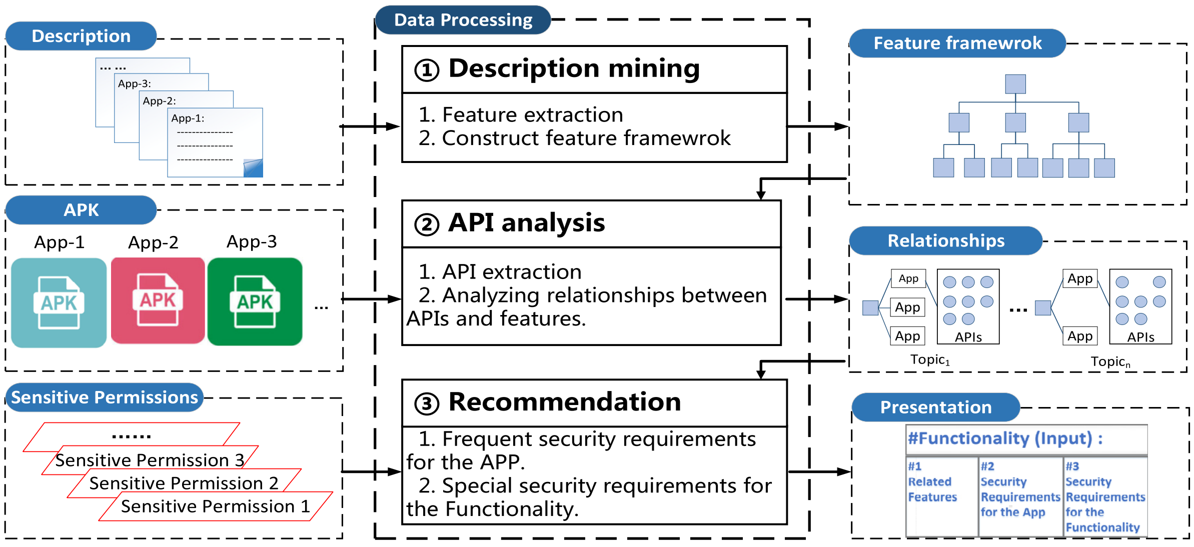


FIGURE 1. Overview of proposed approach.

in the App stores, and they provide valuable data resource for solving this problem.

In fact, for one App, if it wants to access to sensitive resources or private data from smartphones for achieving some functionalities, it needs to employ the corresponding sensitive permissions [9]. Considering such security mechanism of smartphones, we could take the security requirements elicitation of a new App as a process of predicting the permissions it would apply. From this angle, if we can summarize the relationships between the App functionalities and sensitive permissions from existing Apps, we could use such reusable knowledge to recommend the security requirements when developing a new product with similar functionalities.

Based on the above idea, we propose an approach to recommend security related information to developers by mining data in App stores. As Android system is opener than IOS and Android Apps have more than 60% market share, we take Android Apps as our research objects in this paper. Figure 1 shows the process of our approach:

Firstly, we extract features of each App from its description, a common texts data resource for introducing the products in App stores, by using the method given in our previous work [10], and then summarize the information into a feature framework to describe the functionalities of Apps in one domain.

Second, to solve the problem that the permissions of Apps are stated for the whole product rather than functionalities, we use the Application Programming Interfaces (APIs) [11] to bridge the gap between features and permissions. We extract the APIs used in the Apps from the APK files, and establish the relationships between them and the feature framework by calculating two parameters: one is "term frequency, TF", aiming at analyzing the APIs often used by the Apps contain a certain functionality reflected by one node in the framework; the other is "inverse document frequency, IDF", focusing on analyzing the APIs used especially for a certain functionality.

Thirdly, combining with the relationships between APIs and permissions given in PScout [12], we give the information of sensitive permissions related to a given functionality for helping developers elicit security requirements and define a recommendation framework to visualize the results.

Overall, our major contributions include following 2 points:

1) By taking sensitive APIs as a bridge, we give an approach to gain relationships between permissions and functionalities from data resource in App stores, and use the results to help developers elicit security requirements from two aspects:
- The permissions may be employed for developing the Apps having the given functionality as the security requirements for the whole App;
- The permissions should be considered for achieving the given functionality as the security requirements especially for the functionality.

2) We thoroughly evaluate our approach with data of 580 Apps from 5 categories on Google Play from two angles:
- On one hand, we evaluate each step of our approach quantitatively, the results show that our feature framework can summarize the features of Apps reasonably and we can establish relationships between functionalities and APIs accurately.
- On the other hand, we conduct a survey on 60 students and 10 developers to evaluate our recommended information, the responses show that the participants confirm the usefulness of our approach, especially it can help new developers without experience (students) initialize the security requirements quickly and give

mature developers supplementary information to elicit more complete security requirements.

The paper is organized as follow. Section 2 presents the work related to our approach. In section 3, we further state the problem clearly. Section 4 gives the methods for summarizing functionalities from the descriptions to construct feature framework; Second 5 introduces the process for gaining and analyzing the APIs based on the feature framework; Section 6 presents the method for gaining recommended security requirements according to a given functionality, and the introduce the recommendation framework. Finally, the experiments and the conclusion are shown in Section 7 and 8 respectively.

II. RELATED WORK

Our approach aims at helping developers gain complete security requirements at the early stage of App development, and it involves two main research fields: requirements elicitation and security analysis. Thus, we analyze existing work from these 2 directions.

A. REQUIREMENTS ELICITATION OF APPS

Different from traditional software, the requirements elicitation of Apps has its unique characteristics, and we summarized them from following two points.

(Requirements reuse) As there always exists millions of products in the marketplaces, the requirements of these software system can be reused for developing new Apps, and some methods are proposed to achieve this goal. Hariri *et al.* [14] give a method that analyzes online product listings with data mining techniques to discover common features, which can be also used in the products in one domain. Similarly, Ferrari *et al.* [15] mine not only commonalities but also variabilities from the brochures of a group of vendors, and use them to compare existing features in the market so that developers can give a more competitive product. Yu *et al.* [16] gain features from web repositories and organize them into a hierarchical rEpository of software feature (HESA), then recommend relevant and high-quality features to stakeholders. Davril *et al.* [17] provide an approach to construct feature models from publicly available product descriptions found in online product repositories and marketing websites, and the model can be utilized for the domain analysis of new product. In addition, Lian *et al.* [18] present MaRK (Mining Requirements Knowledge) to identify and retrieve requirement knowledge from the documents that contain descriptions of functional features, and such knowledge can be reused for developing a new product. These researches show that the information of exiting Apps can be useful for requirements engineering. However, most of them only focus on functional requirements but ignore non-functional ones. However, these researches seldom consider non-functional requirements, especially there is no research aiming at reusing the security information in requirements elicitation to our knowledge. In our previous works [10], [13], we have given a method to mine part of non-functional features from App descriptions. In this paper, we go one step forward to gain security-related information from existing products and reuse them for requirements elicitation.

(Review mining) Apps highlight user experiences, and this makes reviews become an important data resource for eliciting new requirements. There are a lot of researches on review mining and we simply summarize part of them from two aspects.

On one hand, as users like to express their sentiments in reviews, some research paid attention to mine user preference for guiding the development of Apps. Bin *et al.* [19] propose WisCom, a system can help developers identify users' major concerns and the reasons why they like or dislike a given app. Guzman and Maalej [20] extract fine-grained app features from reviews and mine user sentiments on them to tell developers how do users like this feature. Similarly, Gu and Kim [21] design a review summarization framework SUR-Miner, which classifies reviews and extracts aspects from them to help developers identify what parts of your Apps are loved by users. Differently from above research, Khalid *et al.* [22] focus on the complaints in the reviews and summarize 12 types of complaints usually given by users, this gives developers an insight into the user-raised issues. Similar, Li *et al.* [23], [24] proposed an approach to analyze the impact of an app's release by analyzing the changes of users' sentiment from reviews before and after it.

On the other hand, users give their concrete demands or problems on the Apps in reviews, and there are also many researches aiming at extracting such information for software evolution. Villarroel *et al.* [25] introduce CLAP to categorize and cluster user reviews based on their contents, then prioritize the clusters of reviews for planning the app release. Gao *et al.* [26], [27] design a framework in prioritizing and discovering emerging App issues for developers by tracing reviews over versions. More specifically, Palomba *et al.* [28] introduce CHANGE ADVISOR to gain requirements by analyzing the structure, semantics, and sentiments of reviews and then help developers to identify the set of source code components need to be changed. Furthermore, Yu *et al.* [29] improve CHANGE ADVISOR and give a tool ReviewSolver to locate the problematic code more effectively. Mining reviews is a hot research question and there are many researches on it [30].

However, for the App to be developed, there is no data of reviews for it. Although we can collect reviews from related Apps [31], they contain too many questions special for their own apps. Thus, the usefulness of reviews is limited at the early stage of development. Different from these works, we use the descriptions and APK files of Apps as the data resource. We not only give a method to mine useful information from them, but also define a reasonable structure to integrate and organize the results for supporting the information recommendation.

B. SECURITY ANALYSIS OF APPS

The importance of security for Apps is undoubted, and many researchers devote themselves to this research field.

Here, we only summarize parts of their works related to us from two aspects.

(Description-to-behavior fidelity) Many researches analyze the security of apps by inspecting whether their behavior is exactly same as they claim. Qu *et al.* [32] present a system AutoCog to relate descriptions with permissions by employing techniques in natural language processing and learning-based algorithm, and it helps to analyze whether the description reflects the need for permission. Gorla *et al.* [33] give CHABADA check implemented App behavior against advertised App behavior to discover potential risky products. Motivated by CHABADA, Zhang *et al.* [34] revisit the study of checking app behavior against app description in the context of TPLs, and it can identify more than 50% of new outliers. User interface can be taken as another kind of App descriptions, Avdiienko *et al.* [35] extract Android APIs invoked in Apps as well as the text shown on their screen, and use such association to detect whether user interface elements do the action they suggested. Zhang *et al.* [36] apply sensitivity analysis to assess an app's privacy risks by checking whether a requested sensitive resource would contribute to any user-perceivable app features. Considering that only descriptions or permissions could not declare all sensitive operations, Yu *et al.* [37] introduce app's privacy policy and its bytecode to enhance the description-to-behavior fidelity, and develop a new system TAPVerifier for carrying out investigation of individual software artifacts and conducting the cross-verification. All these researches show a fact that the sensitive behavior reflected by permissions or APIs should be consistent with the functionalities given in the descriptions. Our work is based on this fact, but on the opposite, we use APIs as a bridge to mine the mapping between permissions and App functionalities (behaviors) from existing products, and use the knowledge to help developers elicit security requirements related to their functionalities at the early stage of development.

(Detection of Malicious Apps) Some researches analyze the security of apps from the angle of detecting whether they are malicious. Zhou *et al.* [38] use a system called DroidRanger to detect malicious apps, it uses a permission-based behavioral foot printing scheme to detect new samples of known Android malware families and applies a heuristics-based filtering scheme to identify certain inherent behaviors of unknown malicious families. Arp *et al.* [39] propose DREBIN, which performs a broad static analysis, gathering as many features of an app as possible and embedding them in a joint vector space, in this way, it can use machine learning techniques to identify malwares automatically. Considering Permission control is one of the major Android security mechanisms, Wang *et al.* [40] explore the permission-induced risk in Android apps for malicious detection, and they evaluate the usefulness of risky permissions for mal-app detection with support vector machine, decision trees, as well as random forest. In addition, some researches take the detection of malicious Apps as a classification problem and solve it by giving classifier. Tao *et al.* [41] mine hidden patterns of malware and extract highly sensitive APIs that are widely used in Android malware by studying real-world Android apps, and give a system MalPat to distinguish malicious and benign Android Apps. Koli [42] gain the features from random collected samples of goodware and malware apps, and use them to train the classifiers as a machine learning-based malware detection system for Android platform. All above researches could assist Android app marketplaces to fight against malware, but they could not help developers to develop a benign product. With different purpose from these researches, we assume that most of Apps in the market places are safe, and mine sensitive permissions as well as APIs employed by their functionalities. This information can be used as the background knowledge to predict the risk in the Apps to be developed, and help developer elicit security requirements efficiently and completely.

III. PROBLEM STATEMENT

In the development of a new App, the coding often started after functional requirements have been identified but the security requirements are not well elicited yet. This is because the risks are not obvious and developers may not have enough background knowledge and time to consider them well, especially for small companies and individual developers. However, with the progresses of development, new security requirements emerge continually and developers have to spend extra time to modify the Apps. Even worse, some risks may be not noticed before the release of the App.

To better illustrate the problem statement, let us consider a common scenario. Karl and his team want to develop a new App for social communication, they have had general ideas of the new App and proposed some requirements. Then, Karl thinks it is ready to begin coding. However, when realizing some functionalities, Karl finds that the App needs to get sensitive data from users and they need to add protection strategy to keep the product as safe as required by the users. For example, for the functionality "share photo", it needs to scan the files in the phone by employing the permissions related to "STORAGE", this means the privacy information could be disclosed to malicious activity, such as deleting important documents (this is an important problem often complained by users). Thus, Karl must continually add new security requirements and modify the corresponding functionalities or even the whole product. The extra time cost is not the only problem, what worries Karl more is the potential risks they have not noticed yet.

Our work aims at helping Karl elicit the security requirements at the beginning according to the knowledge mined from existing products in the App stores. To achieve this goal, we require Karl download the description texts and APK files of the Apps in the category "Social" to establish the dataset AppDataset = $\{\mathrm{App}_1, \ldots, \mathrm{App}_n\}$, and each App can be specified as a 3-tuple, $\mathrm{App}_i = (Name, Description, APK)$, where:

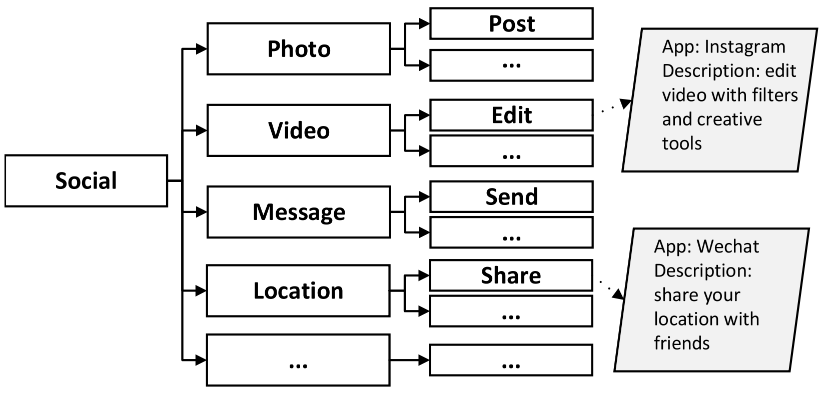

Social
Photo
Video
Message
Location
...
Post
Edit
Send
Share
App: Instagram
Description: edit video with filters and creative tools
App: Wechat
Description: share your location with friends

FIGURE 2. An example of feature framework.

- Name is the unique identifier of the App;
- *Description* is the texts introduced the App;
- *APK* is the install document of the App.

From the AppDataset, our approach not only can to summarize the features of Apps as we do before [10], [13], but also gains the permissions employed for them as the security requirements should be considered in the development process. The approach consists of three steps as shown in Figure 1 and we give the detailed introduction of them in the subsequent sections.

IV. MINING DESCRIPTIONS OF APPS

App description is the introduction of production provided in kinds of market platforms. Although the incomplete nature makes only one App description cannot provide enough information for understanding all the functionalities, the ones collected from related (similar) products together can well cover most features in a particular domain. Thus, we summarize the App features in descriptions as the basis for developers to search the information they need.

A. FEATURE EXTRACTION

As the App descriptions are user-oriented and given in natural language, they contain kinds of information besides App functionalities. In our previous work [10], we have given a method to extract App features from descriptions by giving a set of feature extraction rules. The method achieves good performance in our experiments so we also use it here and give a brief introduction.

For each *Description* of App in AppDataset, we extract the information of App features by following four steps:

Preprocess. We remove the non-English and non-texts parts (such as special characters, e-mail address) from the *Description*, and spilt it into a sequence of sentences.

Syntax analysis. We analyze the syntax of each sentence in *Description* in turn with Stanford Parser, and gain a set of parsing trees.

Parsing Tree analysis. For each parsing tree, it is analyzed in a top-down traversal strategy and matched with the tree transform rules we pre-defined. The useless structures (such as the structure only for representing the tense of sentences) are cut from the tree and the pronouns are replaced with their anaphors according to their context.

Phases extraction. Each node in the parsing tree is further matched with the information extraction rules we pre-defined by summering the relationships between structures of sentences and features (for example, verb-object phrase is the most common structure to express functional features of the App). If one node satisfies the rules, the phases describing the features are gained from the descendant nodes. We define the description of a feature f contains two parts: the verb part $f.Des^{verb}$ and the noun part $f.Des^{noun}$.

B. FEATURE SUMMARIZATION

After extracting features from the descriptions of Apps in AppDataset, we need to further summarize them for two reasons: on one hand, there are many features describe similar functionalities of Apps, and we need to group them together for using easily; on the other hand, there is much detailed and scattered information, and we need to integrate them into high-level for improving the understandability.

Traditionally, the features are usually summarized with the methods of topic modeling [33], [43], [44], such as LDA [33], [45], but the meanings of topics are often unclear. Thus, we define a three-level framework according to the structure of phases describing the features for summarizing them. Figure 2 is part of the feature framework summarized from the features of Apps in the category of "Social" on Google-Play, we use it as an example to illustrate the construction method briefly.

The First Level Is Root: There is only one root node and it gives the category of Apps in the market places.

The Second Level Is Noun Level: The nodes in this level are summarized from the noun parts of features and they describe the functionalities in a high-level. For example, the node "Photo" represents the functionalities related to the photo, including "post photo", "search photo" and so on. As one feature may be described by different nouns, such as "picture" is similar as "photo", we cluster them and take one cluster as one node. Here, we use a gradual clustering process

TABLE 1. Summarization of mapping between sensitive permissions and APIs.

Permission	API	Permission	API	Permission	API	Permission	API
WRITE_EXTERNAL_STORAGE	160	CALL_PHONE	32	READ_CALENDAR	948	READ_PHONE_STATE	3734
ACCESS_COARSE_LOCATION	3369	READ_SMS	848	SEND_SMS	1853	RECORD_AUDIO	526
ACCESS_FINE_LOCATION	1580	GET_ACCOUNTS	2770	WRITE_CONTACTS	2693	READ_CONTACTS	3024
PROCESS_OUTGOING_CALLS	312	WRITE_CALENDAR	732	RECEIVE_SMS	495	CAMERA	440
READ_EXTERNAL_STORAGE	26	WRITE_CALL_LOG	59	READ_CALL_LOG	241	USE_SIP	90
RECEIVE_WAP_PUSH	30	RECEIVE_MMS	25	**Total (Sensitive APIs): 23987**			

rather than classical K-means because the K-value is difficult to determine:

1) We firstly collect the nouns in all features to establish a set Set^{Noun} and sort them according their appearance frequency, then the top-k nouns are used as the initial cluster centers, here we require the similarity between the centers need to be smaller than the threshold α, otherwise they are merger together.
2) Secondly, all the nous in Set^{Noun} are compared with each center in turn. Suppose that one noun W has the biggest similarity with the center C, if this value is larger than α, W is clustered with C, otherwise, W is defined as a new cluster center.

In this process, the similarity between two words is calculated by word2Vec, and the node is presented by the center word of a cluster.

The Third Level Is Verb Level: The nodes in this level are summarized from the verb parts of features and they with their parent nodes together can describe concrete functionalities, such as the node "Share" with its parent node "Location" presents a common functionality. After generating the noun nodes, we group the features with nouns in one cluster together, and further cluster their verb parts in the same way as the noun parts to generate the child verb nodes. In addition, the information of the App containing the feature represented by the node as well as related sentence in the descriptions are reserved, for example, from the node "Edit" under the node "Video" in Figure 2, we can see that the App "Instagram" has this functionality from its description sentence "edit video with filters and creative tools".

The feature framework organizes the functionalities of Apps from different abstract levels, and it can be used as an effective index for developers to search the information they need.

V. API ANALYSIS

Our goal is to help developers elicit security requirements in the early stage by identifying the sensitive permissions they need to employ according to the functionalities, and we use APIs as the bridge to establish the relationships between functionalities and permissions. Thus, we firstly analyze the sensitive APIs based on feature framework to identify their relationships with App functionalities.

A. INITIALIZING RELATIONSHIPS BETWEEN FEATURE FRAMEWORK AND SENSITIVE APIs

PScout [11] has established mapping relationships between permissions and APIs, this provides us a basis to identify the APIs need to be extracted. In the website of PScout, there are 10 different versions. To better cover APIs used in the existing Apps, we collect all the 10 versions and summarize the relationships between sensitive permissions and APIs. As shown in Table 1, a sensitive permission is mapped with more than one API, for example, there are 160 different APIs require the permission "WRITE_EXTERNAL_STORAGE". Totally, we identify 23987 sensitive APIs as the objects to be analyzed.

For one App in AppDataset, we decompile its APK file by using apktool, a popular tool for analyzing the codes of Android App, and then examine the sensitive APIs used in the custom code. In this way, we can establish the relationship between one App and sensitive APIs it uses, that is for each App $\in$ AppDataset, we have a mapping $App \rightarrow \{SenApi_1, \ldots, SenApi_n\}$, where $SenApi$ is the API related to sensitive permissions summarized from PScout.

Meanwhile, we have the mapping between nodes in the feature formwork and Apps in AppDataset: for an App $\in$ AppDataset and a noun node N_{noun} in the second level of feature formwork, if there exists a feature f extracted from $App.Des$ and the noun part of f belongs to the cluster of N_{noun}, we have the relationships between the App and N_{noun}, $N_{noun} \rightarrow \{App\}$; furthermore, N_{noun} must have a child node N_{n-verb}, which is a verb node in the third level of feature framework, and the verb part of f belongs to the cluster of N_{n-verb}, we have the relationships between the App and N_{n-verb}, $N_{n-verb} \rightarrow \{App\}$. These mappings can be established easily when constructing the feature framework.

Based on the above analysis, we can initialize the relationships between the feature framework and sensitive APIs: for each node N in the framework, we have $N \rightarrow Set^{N}_{app} = \{App_1, \ldots App_n\}$, Set^{N}_{app} is the set of Apps having the functionality reflected by N, and $App_i \rightarrow Set^{App_i}_{API} = \{SenApi_1, \ldots, SenApi_m\}$, $Set^{App_i}_{API}$ is the set of sensitive APIs used in the App_i.

Note that, the noun node in the framework gives the functionality in a high-level and it can be broken down to a set of concrete functionalities represented by the verb nodes, so the Apps related to the brother verb nodes together are the set of Apps related to their parent noun node. Suppose that a noun node N_{noun} and the set of its children are $N_{verb_1}, \ldots, N_{verb_n}$, we have:

$$Set^{N_{noun}}_{app} = \bigcup_{i=1}^{i=n} Set^{N_{verb_i}}_{app}.$$

Meanwhile, different nodes in the framework can share same Apps because one product can have diverse functionalities.

B. RELATIONSHIPS ANALYSIS

By analyzing the sensitive APIs used in Apps and the features of these Apps, we identify the relationships between APIs and functionalities initially, the sensitive APIs related to the node N is the summarization of APIs used in the Apps related to N, that is:

$$N \to Set_{API}^{N} = \bigcup_{i=1}^{i=n} Set_{API}^{App_i}.$$

However, this is not enough for eliciting security requirements because such relationships are too abstract to use. Thus, we further distinguish the roles of these APIs and introduce two parameters often used in NLP to quantify their values for the subsequent recommendation, suppose that there is a node N and a sensitive API $SenApi$.

- $\mathrm{TF}(N, SenApi)$: term frequency, the frequency of $SenApi$ appearing in the Apps that have the functionality represented by N, and it is calculated by the following formula,

$$\mathrm{TF}\,(N, SenApi) = \frac{|\left\{App_i | App_i \in Set_{app}^{N} \cap SenApi \in Set_{API}^{App_i}\right\}|}{|N \to Set_{app}^{N}|}.$$

- $\mathrm{TF}-\mathrm{IDF}(N, SenApi)$, inverse document frequency, it reflects whether the $SenApi$ is used for achieving the functionality represented by N rather than other functionalities of Apps,

$$\mathrm{TF}-\mathrm{IDF}\,(N, SenApi) = \mathrm{TF}\,(N, SenApi) \times IDF\,(N, SenApi)\,,$$

where,

$$\mathrm{IDF}\,(N, SenApi) = \lg\left(\frac{|Set_{app}^{\mathrm{Parent}(N)}|}{|\left\{App_i | App_i \in Set_{app}^{\mathrm{Parent}(N)} \cap SenApi \in Set_{API}^{App_i}\right\}|}\right).$$

Here, $\mathrm{Parent}(N)$ represents the parent node of N. As the feature framework organizes the functionalities of Apps in a level structure, the child nodes are analyzed in the range of products related to their parent node.

The two parameters can be used to evaluate the relationships between APIs and functionalities, analyze whether they are related directly (the API is used for the functionality) or indirectly (the API is used for the App). This provides us the basis for prioritizing the information recommended from different angles.

VI. INFORMATION RECOMMENDATION

Having the relationships between functionalities and APIs, we can identify permissions they required based on the mapping between APIs and permissions given by PScout. However, the large-scale information summarized from Apps in one domain is not easy to use directly, so we extend the recommendation framework designed in our previous work [13], which aims at recommending functional requirements, to help developers get security requirements according to their functionality demands, meanwhile, and the two-kind information can be combined together for better supporting the development process.

A. RETRIEVE INFORMATION

At the early stage of development process, the developers often only have a general idea or some key points on the functionalities of the App, and we use them to retrieve information from the feature framework. Suppose that developers give a functionality in natural language, this process consists of three main steps.

Firstly, extracting the keywords from the developers' demands. Similar as the descriptions of Apps, we also analyzed the syntax of the sentence given by the developers and used the pre-defined rules to extract the words expressing the feature f contained in it. The keywords are separated into two sets according to their POS: the set of nouns $\mathrm{Set}_{\mathrm{noun}}$ and the set of verbs $\mathrm{Set}_{\mathrm{verb}}$.

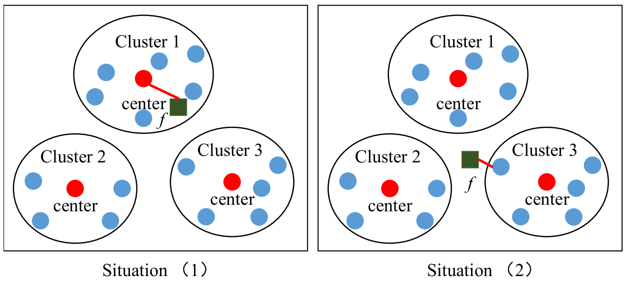


FIGURE 3. **Two situations for retrieving nodes in feature framework.**

Secondly, retrieving the related nodes in the feature framework based on the keywords. The feature framework is analyzed in a top-down strategy. Firstly, the nodes in the noun level (the second level) are compared with the keywords in $\mathrm{Set}_{\mathrm{noun}}$, and there are two situations as shown in Figure 3:

- if the $\mathrm{Set}_{\mathrm{noun}}$ of f is in the cluster of one node, that is the similarity between the center word of the cluster and the word in $\mathrm{Set}_{\mathrm{noun}}$ is larger than the threshold α, the node is identified as the one related to f, denoted by N^f;
- otherwise, if f is out of the range of all the clusters, we further to find the cluster nearest to f as the related node N^f by calculating the similarity between each word in the cluster and $\mathrm{Set}_{\mathrm{noun}}$.

After identifying N_{noun}^{f}, we further retrieve the related verb node N_{verb}^{f} from its child nodes in the verb level (the third level), the process is the same as before but if f

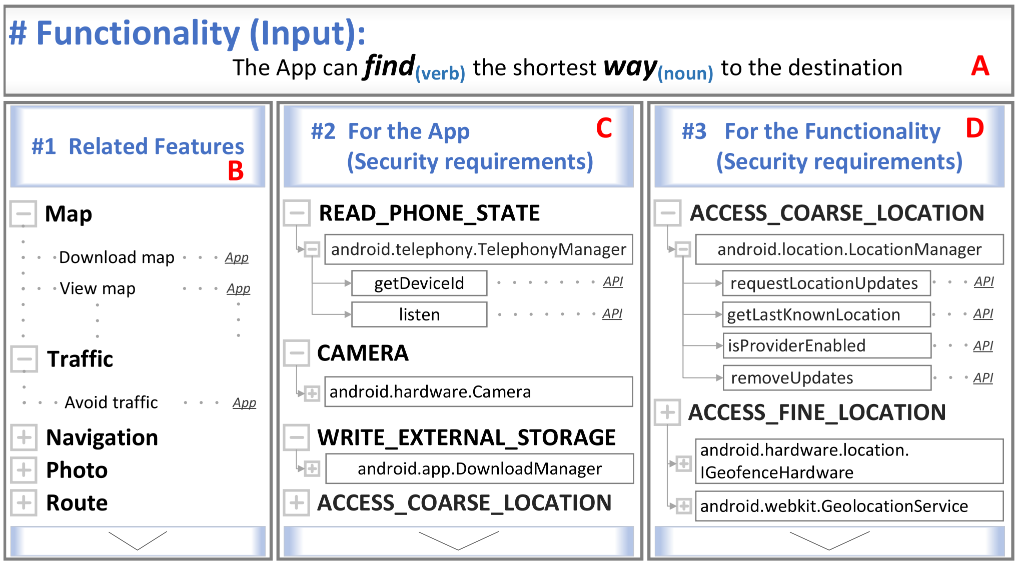


FIGURE 4. Examples of recommended information.

is a high-level feature and Set$_{\text{verb}}$= ∅, there would not exit related N^f_{verb}.

Thirdly, gaining related APIs as well as permissions. For each node N in the feature framework, we have gained its related sensitive APIs, $N \rightarrow Set^N_{API} = \{SenApi_1, \ldots, SenApi_n\}$, in which each API has two evaluation parameters $\text{TF}(N, SenApi)$ and $\text{TF} - \text{IDF}(N, SenApi)$. So we also have the related APIs to f according to N^f_{noun} and N^f_{verb}. Meanwhile, we could also have the permissions related to f according to PScout, if there exists a mapping between $SenApi$ and a Permission and $SenApi$ is related to N^f_{noun} or N^f_{verb}, we have the Permission is related to f.

B. THE RECOMMENDATION FRAMEWORK

In order to help developers understand the security-related information clearly, we design a recommendation framework to represent the retrieving results. Figure 4 gives an example that recommends the information summarized from 40 Apps in the "Social" according to one functionality demand, we use it to introduce each part of the framework.

PART A (Functionality): In the part A of the framework, developers can input a functionality of the App to be developed, and our method gains its keywords for the recommendation. For example, the functionality f consisting of the noun "*way*" and the verb "*find*" is extracted from the sentence "*The App can find the shortest way to the destination*".

PART B (Related Features): Part B gives the features that often appear together with the given functionality in existing Apps. The information can be gained with the method proposed in our previous work [13] and it is helpful for the developers to understand the Apps having the given functionality, so that they can better understand the recommended security requirements of the new App. According to the structure of feature framework gained from the dataset, the information in this part has two levels: the first level is the high-level features represented by the noun nodes in the feature framework, and developers can overall know the related features, such as "*Map*" shows that the operations on the "*map*" are also needed for realizing the functionality "*find way*"; then, the word in the first level can be further unfolded into concrete features represented by the verb nodes, such as "*Download map*", so that the developers can understand the information clearly. In addition, an example App is given behind each related feature to show it intuitively, and developers can go to its homepage to further gain more detailed information.

We sort the information in this part based on their recommendation values calculated by the evaluation method given in our previous work [13].

PART C (Security Requirements for the App): Part C recommends the security requirements often applied by the Apps contain the functionality f by giving the permissions they usually employ. From the feature framework, we retrieve the node N related to f, and gain the related sensitive APIs $Set^N_{API} = \{SenApi_1, \ldots, SenApi_n$, we sort these APIs according to TF $(N, SenApi)\}$, which represents their frequency in the existing Apps, then the permissions related to the top-k $SenApi$ are recommended in this part. Although these permissions may be not employed for achieving the f, they are often employed by existing Apps so the developers should also consider them for developing the new App.

The recommended permissions are sorted according to TF $(N, SenApi)$ of their related APIs. Suppose that the APIs related to one $Permission$ are $\{SenApi_1, \ldots, SenApi_k\}$, we quantify the priority of the $Permission$ as $\sum_{i=1}^{i=k}$ TF $(N, SenApi)$. Furthermore, for each recommended permission, we also represent its related APIs and organize them by using their classes and methods. These APIs not only can help developers better understand why the permission

TABLE 2. Detailed information of our dataset.

	DataSet1				DataSet2
Category	Navigation	Photograph	Music	Travel	Social
Number of Apps	45	45	45	45	400
Number of Sentences	795	1035	682	834	6983

TABLE 3. Participants of the experiments.

	Numbers	Age	Major	Development Skills
Doctors	3	25~30	Requirements engineering	More than 1 year
Students	60	20~23	Computer science and technology	Null
Developers	10	28~40	Software Engineering	More than 5 years

is needed but also can be used in the new App. For example, the sensitive permission "READ_PHONE_STATE" is important for the Apps in the "social" and it is recommended firstly, then it can be unfolded into the class API "android.telephony.TelephonyManager" with two concrete methods "getDeviceId" and "listen", which are the APIs used frequently. Meanwhile, we also give the link of the introduction on each API so that developers can gain more information if needed.

PART D (Security Requirements for the Functionality): Part D recommends the security requirements applied for realizing the functionality f according to $\mathrm{TF}-\mathrm{IDF}(N, SenApi)$, which reflects the specificity of the API for f. The recommended sensitive permissions as well as their related APIs are organized in the same way as Part C but sorted by $\mathrm{TF}-\mathrm{IDF}(N, SenApi)$. The information is more specific to the functionality f, and it reflects the potential security risk should be paid attention to when developing f. Note that, the information in Part C could also appear in this part, because the permission would be important for both the App and f, especially when f is the core functionality of the App. For example, the permission "ACCESS_COARSE_LOCATION" are required by the functionality "*find way*", meanwhile, it is common for the Apps in "*social*" to get users' location information.

In Figure 4, the functionality given by the developers is a concrete functionality and expressed by verbs and nouns, so the recommended information is gained from the third-level (verb node) of the feature work. However, there is another situation that the input f is a high-level functionality, that is its verb part $\mathrm{Set}_{\mathrm{verb}} \neq \emptyset$, we gain the recommended information from the second-level of the feature framework. In this way, we can give more targeted information according to developers' different kinds of demands.

VII. EXPERIMENTS

To evaluate our approach, we conducted a series of experiments and surveys to answer the following two questions:

- RQ1: Whether our approach can summarize and organize the information from data of Apps reasonably?
- RQ2: Whether the recommended information by our approach is useful for eliciting the security requirements at the early stage of the development?

Specially, RQ1 focused on inspecting the performance of each step of our approach to ensure the information can be mined efficiently. RQ2 aims at analyzing the usefulness of our approach in practice.

A. DATASET AND PARTICIPANTS

We chose Apps from 5 categories as the subjects of our experiments and collected their data, including descriptions as well as APK files, from Google Play. Two datasets are created as shown in Table 2:

DataSet1 was for RQ1 and it consists of 180 Apps covering 4 categories ("Navigation", "Photograph", "Music", and "Travel"), so that it can better check the effectiveness of our approach in different kinds of products.

DataSet2 was for RQ1 and it included 400 Apps in "Social", which provides us a relevant large-scale resource to gain the recommended information for evaluating the usefulness of our approach.

The participants of our experiments and surveys are from the university and software companies. The detailed information is shown in Table 3:

1) 3 doctors, majoring in software engineering, especially requirements engineering. All of them have more than one year industrial experience in App development.
2) 60 students, they came from computer science and technology in Jilin University. All of them had taken course on software engineering, so we take them as new developers without much experience.
3) 10 App developers, they were from 5 different companies and have more than 5 years software development experience, so we treated them as mature developers to evaluate the recommended information.

B. EXPERIMENTS FOR RQ1

As the approach is based on our previous work [10], [13], the performance of some steps had been evaluated, including extracting App features from descriptions and clustering features. Meanwhile, we also use some existing methods in

TABLE 4. Comparison between feature framework and LDA.

	Understandability						Representation					
	Students			Developers			Student			Developer		
	FAM	LDA	P-V	FAM	LDA	P-V	FAM	LDA	P-V	FAM	LDA	P-V
Navigation	4.04	3.24	0.001	4.60	3.20	0.010	3.80	3.84	0.743	3.10	3.20	0.792
Photograph	3.87	3.02	0.001	4.30	3.30	0.015	4.13	3.89	0.132	3.60	3.50	0.739
Music	4.00	3.74	0.07	4.50	4.10	0.157	4.41	4.07	0.012	3.90	3.70	0.608
Travel	3.85	2.87	0.001	3.90	3.20	0.020	3.72	3.83	0.429	3.10	3.10	1.000

our approach, such as PScout, apktool, and they are widely used and we do not inspect them either. Thus, we answer RQ1 by evaluating the rest two main steps with two sub-questions as follow:

- Sub-Question1: Whether the feature framework can summarize the features gained from App descriptions reasonably?
- Sub-Question2: Whether our approach can establish relationships between functionalities and APIs effectively?

1) COMPARISON BETWEEN FEATURE FRAMEWORK AND LDA

LDA [45] is a classical topic modelling method and often used to summarize the App descriptions to gain high-level functionalities, its reasonability has been illustrated. Thus, we take LDA as the baseline to evaluate our feature framework for summarizing features and answer Sub-Question1.

a: EXPERIMENT DESIGN

We firstly extracted features from descriptions of Apps in one category, then summarized them by the proposed method and LDA separately, the results, including a feature framework and set of topics, are taken as the materials in the experiment. For the feature framework, we sort the nodes in each level according to the number of words in the corresponding clusters, and choose the top-10 noun nodes and their top-10 child verb nodes to construct a simplified model to be evaluated. For the topics generated by LDA, their number is adjusted from 5-15 and we choose the best results manually as the object to be compared.

In the experiment, we gave the participants (60 students and 10 developers) both the feature framework and the topics, and asked them to evaluate the summarization results from two aspects separately: (understandability)whether the result is easy to be understood; and (representation) whether the result can well represent the functionalities of Apps in one category. The responses were given on scale 1 to 5, where "1" indicates "absolutely not" and 5 "totally yes".

Two-tailed statistical tests were used for comparisons in the four categories of Apps because of the non-directionality of the hypotheses. The null hypothesis H0: there is no difference between the feature framework and topics generated by LDA. The probability of 5% was accepted as the threshold to reject the null hypothesis [46]. In addition, as not all the students took the job seriously, we filter the responses of the students who gave same scores to all questions to reduce the bias. On the opposite, we reserved all the responses from developers because we thought they were more responsible.

b: RESULTS

We finally gained 46 pairs of effective responses from students and 10 pairs from developers. The responses from students are analyzed by Student's t test by assuming they were distributed normally, while Wilcoxon signed ranks test was adopted to analyze the responses from developers [46].

The results of comparisons from two aspects are shown in Table 4, where FAM represents the feature framework and column P-V is the p-value. We can get two main observations as follow:

Firstly, the participants recognize that our feature framework is easy to be understood, the responses are from 3.85 to 4.04 for students and from 3.90 to 4.60 for developers. Meanwhile, it can be seen that the understandability of the feature framework is higher than the topics generated by LDA, and such differences are significant in all the categories (P-value <0.001) except "Music".

Secondly, with respect to the representation, the responses from students are also high, from 3.72 to 4.13, while the ones from developers declined a little but still higher than 3.0. It indicates that our feature framework can well represent the functionalities of Apps in one category to some extent. However, the differences between the feature framework and LDA are not significant (P-value >0.05) except one condition the P-value <0.05 for the responses given by students to the category "Music".

According to above observations, we can see that the results for the category "Music" are different from others, so we further analyze it to better understanding the performance of our approach. We had discussions with a part of students and developers and found that: as the functionalities of Apps in "Music" is relevantly simpler than the Apps in other categories, the words in such condition are easier to be understood, whether in the feature framework or topics gained by LDA. In addition, we think it is good for our feature framework to have similar performance as LDA on representation because we only used a simplified model extracted from the whole framework in the experiment.

In summarize, the hypothesis H0 can be rejected, and we conclude that our feature framework is better than LDA,

TABLE 5. The performance of our method for mapping functionalities with APIs.

	For the APP (TF)		For the functionality (TF-IDF)	
	Noun nodes	Verb nodes	Noun nodes	Verb nodes
Navigation	0.82	0.76	0.53	0.49
Photograph	0.70	0.69	0.63	0.51
Music	0.66	0.65	0.68	0.50
Travel	0.75	0.71	0.55	0.51
Average	73.25%	70.25%	59.75%	50.25%

especially when the functionalities of Apps are complex, and it can summarize the features gained from App descriptions reasonably.

2) MAPPING FUNCTIONALITIES WITH APIs

In our approach, we use APIs as a bridge to establish the relationships between functionalities and sensitive permissions, so a key step is mapping functionalities with APIs, and we give an experiment to evaluate its performance (Sub-Question2).

a: EXPERIMENT DESIGN

We use the simplified feature frameworks (including top-10 noun nodes and their child top-10 verb nodes) generated in the experiment for Sub-Question1 as the materials for answering Sub-Question2. As our approach distinguishes two kinds of relationships between functionalities and APIs, we also evaluate the performance from two aspects: on one hand, whether the API *SenApi* related to the functionality N according to the TF $(N, SenApi)$ is frequently used in the Apps having the N; on the other hand, whether the API *SenApi* related to the functionality N according to the $\mathrm{TF}-\mathrm{IDF}\,(N, SenApi)$ is useful for the Apps achieving the functionality N. Answering the two questions requires professional knowledge, so we only ask the 10 developers participant in the experiment.

In the experiment, we give the noun nodes as well as its one random verb nodes in one simplified feature framework to the developers and confirm that they understood the functionalities representing by the nodes, from high-level to concrete-level. Then, top-10 related APIs identified according to TF and TF-IDF are given to the developers separately, and we asked them to evaluate whether the APIs are adapted to their relationships with the nodes. The response could be positive "YES", negative "NO" or "I do not know".

We collected the responses from each developer. For an API related to a node N, only when all the developers responded "YES" to the relationship between the API and N, we confirmed the relationship is right. According to the responses to the 10 APIs related to N for one kind relationship, we define the precision of the mapping as:

$$\text{Precision} = \frac{Num(\text{YES})}{Num\,(YES) + Num(NO)},$$

where $Num(\text{YES})$ is the number of APIs that the relationship between them and the corresponding node is right. For example, suppose that we have a noun node "Photo", and 8 of its top-10 related APIs identified according to TF are confirmed as usually used in the Apps having the functionalities for handling "photo", the precision of the mapping is 80%.

b: RESULTS

As there are two kinds of relationships, we also analyze the responses from two aspects: relationships for the whole App, and the relationships for the functionalities. In addition, our feature framework has two main levels, noun nodes give the functionalities in high-level, and verb nodes give concrete functionalities, so we analyzed the responses to nodes in different levels separately for each aspect. Table 5 summarizes the results, which show the average precision of nodes in each category.

With respect to the APIs for the App, the precision is above 70% on average, both for the noun nodes and verb nouns in different categories, that is more than 70% of APIs given by our approach are confirmed having right relationships with the node, and they are commonly used in the Apps having corresponding functionalities. In fact, almost all the given APIs had received "YES" as responses, but not all participants think all of them are not necessary for developing the Apps and give answer "No", so the precision is declined as we use very strict standard.

With respect to the APIs for the functionality, it can be seen that the precision is much lower, just above 50%. We think such performance is reasonable because mapping functionalities and their APIs is a difficult task, and our approach is based on statistical analysis without deeply mining the semantic meaning of the APIs and functionalities, so it could not finish the task accurately. However, the purpose of our approach is to recommend sensitive permissions required by the functionalities as the security requirements, as one permission is mapped with many APIs, the influence of inaccurate APIs would be alleviated when recommending high-level requirement information.

Based on the analysis above, we believe the performance of our approach is acceptable and it can establish relationships between functionalities and APIs effectively, especially for finding the APIs often used by the Apps sharing one certain functionality.

C. SURVEY FOR RQ2

It is difficult to evaluate the usefulness of the recommended information as it is subjective. Thus, we conducted a survey for answering RQ2.

TABLE 6. Summarization of responses in the survey.

Questionnaire			Strongly Disagree	Disagree	Neither	Agree	Strongly Agree	Total
Section	Q	Participant						
S1		Students	0	0	11	21	14	46
		Developers	0	0	2	7	1	10
S2	1)	Students	1	2	5	22	16	46
		Developers	0	1	2	5	2	10
	2)	Students	2	2	9	29	4	46
		Developers	1	1	4	4	0	10
	3)	Students	1	5	8	19	13	46
		Developers	0	2	3	4	1	10
S3	1)	Students	2	5	8	20	11	46
		Developers	0	2	1	6	1	10
	2)	Students	3	5	11	18	9	46
		Developers	1	1	2	5	1	10

1) DESIGN OF THE SURVEY

The survey consisted of two parts: one is a questionnaire and the other is an open discussion. We mined the DataSet2 automatically with our approach as the basis of the survey. Then, we randomly chose 5 nouns and 5 related verbs from the feature framework to gain 6 recommendation instances, and use them as the materials, and designed a questionnaire to evaluate each part of information in our recommendation framework (as shown in Figure 4) respectively.

The first section (S1) of the questionnaire evaluates the information shown in Part B. The information is gained by the approach given in our previous work and has been evaluated, so we only give one simple question: whether the information is useful for generally understanding the Apps having the given functionality.

The second section (S2) of the questionnaire evaluates the information given in Part C, and it consists of three main questions: 1) whether the recommended permissions can reflect the security requirements needed to be considered; 2) whether the APIs are useful for understanding the related permissions; 3) whether the information can help you elicit a more complete security requirements for developing a new App.

The third section (S3) of the questionnaire focuses on Part D and it gives questions from the angle of the functionality: 1) whether the recommended information is useful for understanding the potential risk that may be caused by the given functionality; 2) whether the recommended information is useful for adding the security requirements of the functionality.

For each question, we gave the participants five options, from 1 to 5 represent strongly disagree to strongly agree. In addition, an open discussion can be given behind each question and the participants can give any comments to our approach freely.

2) FEEDBACK OF THE SURVEY

This survey was conducted after the experiment for comparing between feature framework and LDA, so that the participants already had a general understanding about our work. The students participated in the survey at class, while the developers could give responses with e-mail according to their own schedules. Meanwhile, the feedback of 14 students was identified invalid, because their responses were filtered in the experiment for evaluating our feature framework. Thus, we totally got 46 valid questionnaires from students and 10 from developers, Table 6 summarizes the responses from the participants and Figure 5 is the boxplot of the results.

With respect to the information presented in Part B of our recommendation framework, most of participants agreed that these features are helpful for them to generally understand the Apps. 35 in all 46 students (76.09%) and 8 in all 10 developers (80%) gave positive responses for the question in Section 1. The results are consistent with the experiments in our previous work, so we believe this Part of framework is useful and bridge the functional requirements and the security requirements at the early stage of App development.

The Part C of the recommendation framework gives the security-related information for the whole App, and the Section 2 of the questionnaire not only aims at evaluating the usefulness of the information (question 3) in this part but also checking the effectiveness of the organization of the information (question 1 and 2):

- From the responses of question 1) in S2, it can be seen that both students and developers believe the sensitive permissions can reflect the security requirements to some extent, while only 2 students and 1 developer gave negative feedback. It indicates that the sensitive permissions are useful information for security requirements elicitation.

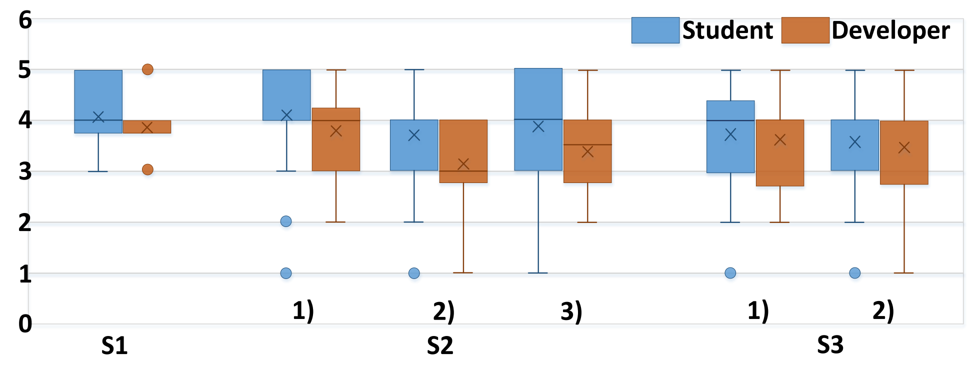


FIGURE 5. **Boxplots of the distributions of the responds in the survey.**

- For the question 2), we want to know whether giving APIs under permissions is a good organization. The responses of students and developers are quite different: 71.74% of students believed the APIs are useful for understanding the permission while only 40% of developers gave positive responses and none of them gave "strongly agree". According to the open feedback of the question, we found the main reason for such difference is that: students have less background knowledge and more related information can help them understand meaning of the permission; on the opposite, developers have known the permissions well and they do not need APIs as supplementation, but they said such information may be more useful for the stage of coding.
- For the question 3), 32 students and 5 developers thought the recommended information is helpful for eliciting a more complete security requirement for developing a new App. The reasons for students and developers giving negative feedback are different: for the students, the limitation is that they had little knowledge on permissions and did not know the meaning of the recommended information; but for the developers, they think the recommended permissions are common for the whole App are simple and they had already known, so the effectiveness for enriching the security requirements is limited.

The Part D of the recommendation framework gives the permissions may need to be considered when developing the demanded functionality. As the information is organized in the same way as Part C, the questions in S3 only evaluate the usefulness of the information.

- From the responses for question 1), 67.37% of students and 70% developers agreed that the information is useful for understanding the potential risk of the given functionality, and only 7 students and 2 developers gave negative feedback while 8 students and 1 developer hold neutral opinion. It indicates that the information is related to the demanded functionality and reflects security requirements to some extent.
- For the question 2), the responses are a little lower than the ones for the question 1), but there are still 27 students and 6 developers think such information can be used to supplement the security requirements. The reasons of the negative feedback are similar as the ones for the question 3) in S2. Thus, we believe that the participants had accepted the usefulness of the recommended information in Part D, and only seldom of them rejected such conclusion.

Besides the questionnaire, we tried our best to have a discussion with the participants to gain their opinions on our approaches. In general, the opinions often include usefulness and limitations of the recommended information.

As the representative of new developers without much experience, students express that the recommended information gives them an easy way to understand the permissions required by the App or the functionality, and this helps them to initialize the elicitation of security requirements and enrich them gradually. As one student said:

"When I get the functionality, I totally have no idea where to start eliciting the security requirements, but from the given information on permissions as well as APIs, I know about the potential risks and can give corresponding requirements".

At the same time, some students indicate that the background knowledge is the most important hinder for them to use the recommended information: "*I really do not know the meaning of these sensitive permissions, it costs much time to ready their introductions to understand them firstly.*"

The concerns of mature developers are different from students, they show great interest in the information that can inspire them with the security requirements "*not easy to be though but indeed exit*", and they said that the recommended information could remind them some important points they had ignored. *"Although most of the given information is common and I have known it, there always exists unexpected one and I really need to consider them for developing a safe App",* a developer said. In addition, the developers paid unexpectedly attention to the information of recommended APIs on their usefulness for developing the Apps rather than for understanding corresponding permissions, they said *"it would be good to give a more precise relationships between* *the functionality and APIs and I can use them for the coding",* and we think it is a good research direction for our future work.

Summary: The participants confirm the usefulness of our approach in practice but from different aspects: it is helpful for new developers to overall construct knowledge structure on the security requirements of the App and break them down into details; and it gives mature developers supplementary information to search missing risk, so that they can elicit more complete security requirements.

D. THREATS TO VALIDITY

Although the results of the experiments and the responses of the survey were relatively good, the validity of our work suffers several threats, we analyze them from the following two aspects.

Threats to the validity of methods in our approach. There are three main threats in this aspect focusing on the limitations of methods themselves in our approach.

Firstly, our approach gets mapping between APIs and permissions from PScout. Although PScout launches new versions continually, it could not provide complete relationships, especially when new APIs appear frequently. However, establishing such relationships is another research question and not easy to be solved, so we did not give new method in this paper.

Secondly, our approach utilizes statistical analysis to identify the APIs related to a given functionality, so it could not give the relationship precisely. However, as the functionality of API is often fine-grained while the functionality in requirements is coarse-grained, it is difficult to map them with semantic analysis. In the future, we consider to use the introductions of APIs into our approach as a new resource and wish to solve this problem.

Thirdly, according to the response of the survey, an important limitation for new developers using our recommendation information is that they could not understand the meaning of permissions well without enough background knowledge. In our method, we have given each permission with the link of their home page, and this can alleviate the limitation to some extent but may be not enough. So, we plan to extract core information from the introduction of the permission and add a brief illustration after the recommendation information, so that users can understand them easily.

Threats to the validity of experiments. The threats from this aspect concern the factors affecting the validity of our experiments, especially the survey.

The first threat stems from the participants of our experiments, their knowledge and skills would impact the results. Hence, we choose two kinds of participants to evaluate our approach. One kind is students with similar background, they are from the same major and same grade; the other kind is developers come from different companies, so that our approach can be analyzed from different angles. Furthermore, we had conducted an open discussion with the participants to better understand their opinions.

Secondly, another threat of our experiments depends on the subjective opinions of participants, either students or developers. It is because there are no objective criteria for evaluating our approach and we could not get an oracle for the experiments. Meanwhile, we do not have conditions to use the approach in practice at present. We wish to cooperate with companies to further evaluate our approach.

VIII. CONCLUSION AND FUTURE WORK

Today, the security becomes more and more important for the success of Apps, and it is deserved to be well considered at the stage of requirements for reducing the cost of software development. In this paper, we propose an approach to help developers elicit security requirements. In our approach, we construct a feature framework by mining App descriptions to express the functionalities of Apps in the marketplaces, and then extract sensitive APIs used in these Apps as the bridge to establish relationships between functionalities and sensitive permissions. We define a recommendation framework for developers getting useful information according to their demands from two aspects: security requirements for the whole App and the ones for the given functionality. Our evaluation experiments show that the feature framework can summarize features in a reasonable way and our method can map functionalities and APIs in an acceptable accuracy. Furthermore, the survey on students and developers also confirms the usefulness of the recommended information in practice.

In the future, we will consider introducing semantic analysis for establishing relationships between APIs and functionalities more accurately, and add the illustration of permissions to the recommendation framework so that new developers can understand them more easily. Also, we wish to use our approach in large-scale cases to further evaluate it.

REFERENCES

[1] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 817–847, Sep. 2017.
[2] S. McIlroy, N. Ali, and A. E. Hassan, "Fresh apps: An empirical study of frequently-updated mobile apps in the Google play store," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1346–1370, Jun. 2016.
[3] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.
[4] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu, "Why are Android apps removed from Google play?: A large-scale empirical study," in *Proc. 15th Int. Conf. Mining Softw. Repositories (MSR)*, 2018, pp. 231–242.
[5] E. Knauss *et al.*, "Supporting requirements engineers in recognising security issues," in *Proc. Requirements Eng., Found. Softw. Qual. Work. Conf.* Berlin, Germany: Springer-Verlag, 2011.
[6] S. Chen, T. Su, L. Fan, G. Meng, M. Xue, Y. Liu, and L. Xu, "Are mobile banking apps secure? What can be improved?" in *Proc. ESEC/SIGSOFT FSE*, Oct. 2018, pp. 797–802.
[7] Z. Fang, W. Han, and Y. Li, "Permission based Android security: Issues and countermeasures," *Comput. Secur.*, vol. 43, pp. 205–218, Jun. 2014.
[8] B. Nuseibeh and S. M. Easterbrook, "Requirements engineering: A roadmap," in *Proc. Int. Conf. Softw. Eng.*, 2000, pp. 35–46.
[9] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. SOUPS*, 2012, pp. 1–14.
[10] Y. Liu, L. Liu, H. Liu, X. Wang, and H. Yang, "Mining domain knowledge from app descriptions," *J. Syst. Softw.*, vol. 133, pp. 126–144, Nov. 2017.

[11] R. Robbes, M. Lungu, and A. Janes, "API fluency," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., New Ideas Emerg. Results (ICSE-NIER)*, May 2019, pp. 97–100.

[12] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proc. Acm Conf. Comput. Commun. Secur.*, 2012, pp. 217–228.

[13] Y. Liu, L. Liu, H. Liu, and S. Li, "Information recommendation based on domain knowledge in app descriptions for improving the quality of requirements," *IEEE Access*, vol. 7, pp. 9501–9514, 2019.

[14] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher, "Supporting domain analysis through mining and recommending features from online product listings," *IEEE Trans. Softw. Eng.*, vol. 39, no. 12, pp. 1736–1752, Dec. 2013.

[15] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta, "Mining commonalities and variabilities from natural language documents," in *Proc. 17th Int. Softw. Product Line Conf. (SPLC)*, Aug. 2013, pp. 116–120.

[16] Y. Yu, H. Wang, G. Yin, and B. Liu, "Mining and recommending software features across multiple Web repositories," in *Proc. 5th Asia–Pacific Symp. Internetware (Internetware)*, Oct. 2013, pp. 1–9.

[17] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," in *Proc. 9th Joint Meeting Found. Softw. Eng. (ESEC/FSE)*, Aug. 2013, pp. 290–300.

[18] X. Lian, M. Rahimi, J. Cleland-Huang, L. Zhang, R. Ferrai, and M. Smith, "Mining requirements knowledge from collections of domain documents," in *Proc. IEEE 24th Int. Requirements Eng. Conf. (RE)*, Beijing, China, Sep. 2016, pp. 156–165.

[19] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2013, pp. 1276–1284.

[20] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of app reviews," in *Proc. IEEE 22nd Int. Requirements Eng. Conf. (RE)*, Aug. 2014, pp. 153–162.

[21] X. Gu and S. Kim, '"What parts of your apps are loved by users?'(T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 760–770.

[22] H. Khalid, E. Shihab, and A. E. Hassan, "What do mobile app users complain about? A study on free iOS apps," *IEEE Softw.*, vol. 32, no. 3, pp. 70–77, May/Jun. 2015.

[23] X. Li, Z. Zhang, and K. Stefanidis, "Mobile app evolution analysis based on user reviews," in *Proc. SOMET*. 2018, pp. 773–786.

[24] X. Li, Z. Zhang, and K. Stefanidis, "Sentiment-aware analysis of mobile apps user reviews regarding particular updates," in *Proc. ICSEA*, 2018, p. 109.

[25] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *Proc. 38th Int. Conf. Softw. Eng. (ICSE)*, May 2016, pp. 14–24.

[26] C. Gao, B. Wang, P. He, J. Zhu, Y. Zhou, and M. R. Lyu, "PAID: Prioritizing app issues for developers by tracking user reviews over versions," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 35–45.

[27] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, pp. 48–58.

[28] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 106–117.

[29] L. Yu, J. Chen, H. Zhou, X. Luo, and K. Liu, "Localizing function errors in mobile apps with user reviews," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2018, pp. 418–429.

[30] N. Genc-Nayebi and A. Abran, "A systematic literature review: Opinion mining studies from mobile app store user reviews," *J. Syst. Softw.*, vol. 125, pp. 207–219, Mar. 2017.

[31] N. H. Bakar, Z. M. Kasirun, N. Salleh, and H. A. Jalab, "Extracting features from online software reviews to aid requirements reuse," *Appl. Soft Comput.*, vol. 49, pp. 1297–1315, Dec. 2016.

[32] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "AutoCog: Measuring the Description-to-permission fidelity in Android applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2014, pp. 1354–1365.

[33] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, 2014, pp. 1025–1035.

[34] C. Zhang, H. Wang, R. Wang, Y. Guo, and G. Xu "Re-checking app behavior against app description in the context of third-party libraries," in *Proc. SEKE*, 2018, pp. 664–665.

[35] V. Avdiienko, K. Kuznetsov, I. Rommelfanger, A. Rau, A. Gorla, and A. Zeller, "Detecting behavior anomalies in graphical user interfaces," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 201–203.

[36] L. L. Zhang, C.-J.-M. Liang, Z. L. Li, Y. Liu, F. Zhao, and E.-H. Chen, "Characterizing privacy risks of mobile apps with sensitivity analysis," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 279–292, Feb. 2018.

[37] L. Yu, X. Luo, C. Qian, S. Wang, and H. K. N. Leung, "Enhancing the description-to-behavior fidelity in Android apps with privacy policy," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 834–854, Sep. 2018.

[38] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," in *Proc. NDSS*, 2012, pp. 50–52.

[39] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. E. R. T. Siemens, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, 2014, pp. 23–26.

[40] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in Android applications for malicious application detection," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.

[41] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.

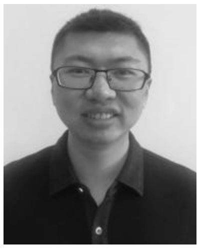
[42] J. D. Koli, "RanDroid: Android malware detection using random machine learning classifiers," in *Proc. Technol. Smart-City Energy Secur. Power (ICSESP)*, Mar. 2018, pp. 1–16.

[43] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua, "New and improved: Modeling versions to improve app recommendation," in *Proc. 37th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2014, pp. 647–656.

[44] S. Vakulenko, O. Müller, and J. V. Brocke, "Enriching iTunes App Store categories via topic modeling," in *Proc. Int. Conf. Inf. Syst. (ICIS)*, 2014, pp. 1–11.

[45] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

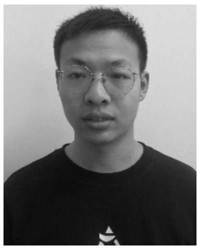
[46] C. Wohlin, P. Runeson, M. Höst, and M. C. Ohlsson, *Experimentation in Software Engineerin*. Berlin, Germany: Springer-Verlag, 2012.



YUZHOU LIU received the bachelor's degree in optical information science and technology from the Beijing Institute of Technology, in 2010, and the Ph.D. degree in computer science and technology from Jilin University, China, in 2019. Then, he worked as a Software Engineer at China Unicom, until 2014. During the period of work, he is mainly engaged in data extraction and data analysis. He is currently a Teacher with the College of Computer Science and Technology, Jilin University. His current research concerns on requirements engineering, data mining, and natural language process.

LEI LIU received the master's degree in computer science from Jilin University, China, in 1985. He is currently a Doctoral Supervisor with the College of Computer Science and Technology, Jilin University. The central themes of his research are programming language and its realization technology, software security and cloud computing, the semantic web and ontology engineering, and knowledge representation and reasoning. At Jilin University, he has held responsibilities for more than 30 projects as a lead person in the area of computer science. He has authored numerous articles and technical reports on various international journals and conferences.

HUAXIAO LIU received the Ph.D. degree in computer science from Jilin University, China, in 2013. He is currently an Assistant Professor with the College of Computer Science and Technology, Jilin University. The central theme of his research is improving software quality, and his recent research concerns the software requirements engineering, software cybernetics, and formal methods of software development. More specifically, he develops techniques to verify aspect-oriented requirements model based on ontology.



SHANQUAN GAO received the master's degree in software engineering from Jilin University, China, in 2018, where he is currently pursuing the Ph.D. degree with the College of Computer Science and Technology. His current research concerns on requirements engineering and data mining.

GUOHANG SONG received the master's degree in computer science from Jilin University, China, in 2013, where he is currently pursuing the Ph.D. degree with the College of Computer Science and Technology. He is mainly engaged in data mining and data analysis. His current research concerns on recommendation system.