AUTOREB: Automatically Understanding the Review-to-Behavior Fidelity in Android Applications

Deguang Kong
Samsung Research America
doogkong@gmail.com

Lei Cen
Purdue University
lcen@purdue.edu

Hongxia Jin
Samsung Research America
hongxia@acm.org

ABSTRACT

Along with the increasing popularity of mobile devices, there exist severe security and privacy concerns for mobile apps. On Google Play, user reviews provide a unique understanding of security/privacy issues of mobile apps from users' perspective, and in fact they are valuable feedbacks from users by considering users' expectations. To best assist the end users, in this paper, we automatically learn the security/privacy related behaviors inferred from analysis on user reviews, which we call review-to-behavior fidelity. We design the system AUTOREB that automatically assesses the review-to-behavior fidelity of mobile apps. AUTOREB employs the state-of-the-art machine learning techniques to infer the relations between users' reviews and four categories of security-related behaviors. Moreover, it uses a crowdsourcing approach to automatically aggregate the security issues from review-level to app-level. To our knowledge, AUTOREB is the *first* work that explores the user review information and utilizes the review semantics to predict the risky behaviors at both review-level and app-level.

We crawled a real-world dataset of $2,614,186$ users, $12,783$ apps and $13,129,783$ reviews from `Google play`, and use it to comprehensively evaluate AUTOREB. The experiment result shows that our method can predict the mobile app behaviors at user-review level with accuracy as high as 94.05%, and also it can predict the security issues at app-level by aggregating the predictions at review-level. Our research offers an insight into understanding the mobile app security concerns from users' perspective, and helps bridge the gap between the security issues and users' perception.

Categories and Subject Descriptors

D.4.6 [**Operating System**]: Security and Protection; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Security, Privacy, Algorithm

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

DOI: http://dx.doi.org/10.1145/2810103.2813689.

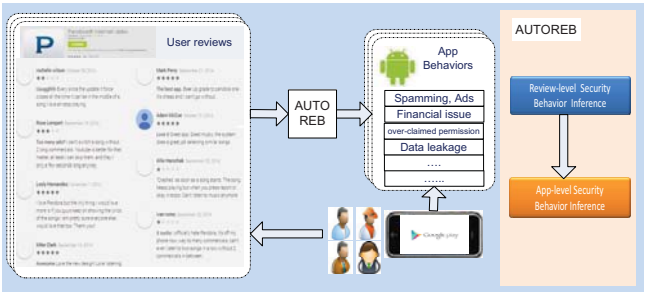


Figure 1: Infer the security-related behaviors from users' reviews. Overview of the framework of AUTOREB : (a) Engine 1: Review-level security behavior inference engine; (b) Engine 2: App-level security behavior inference engine.

Keywords

Android; App; Risk; Information Retrieval; Perception; Learning

1. INTRODUCTION

Nowadays, people spend more time using mobile apps on smart phones and tablets because of the convenience they bring to people's daily life. Personalized service (such as targeted advertising) is possible on mobile devices when users' personal information (*e.g.,* contacts, user locations), is accessible by mobile apps. However, disclosing personal information to mobile apps could lead to serious security and privacy concerns.

On `Google play store`, user reviews are public to all users (see Fig. 1). The user reviews describe how the users think about the app, and give an idea of the security-related behaviors during the running of apps, which we call "*review-to-behavior fidelity*[1]". For mobile app end users, whether an app has exposed severe security risk or not actually depends on how they think about it. User review provides such information and can help people make a decision. The user review perspective has been applied in other applications, such as restaurant recommendations and online product purchase.

With the belief that the user reviews and apps' behaviors should be consistent, we present AUTOREB, a system that automatically identifies the user reviews that reflect the security-related behaviors both at review-level and app-level. This is the first work on app risk analysis from real users' reviews. In this work, we extend many techniques to the context of mobile app review analysis,

[1]In this paper, the "behavior" only refers to the security and privacy related behavior of mobile apps. We focus on four categories of security-related behaviors extracted from user reviews, which are demonstrated in Table 1.

Category	Behavior Descriptions
`Spamming`	Ads in notification bar, ads via email, ads via SMS, pop-up ads, fishing, *etc.*
`Financial Issue`	Paid for *In-App-Purchase (IAP)*, but do not get the item, free to premium, *etc.*
`Over-privileged Permission`	Request too much permissions than users' expectations
`Data leakage`	Access privacy data without users' acknowledgement, *e.g.*, users' account, contact, location, *etc.*

Table 1: Security/Privacy-related behaviors

including information retrieval, text mining, and machine learning. AUTOREB can be applied in the following scenarios.

• End users can automatically infer whether the app has security-related behaviors from other users' experiences and expectations.

• Mobile app developers are alert of users' complains from users' feedbacks, and are aware of the security-related behaviors that the mobile app has displayed.

• App security behavior analysis is helpful for the risk assessment of mobile apps, which can be used to improve the credibility of app rating scores in app markets.

However, in practice, in order to build a system that can automatically solve the review-to-behavior fidelity problem, we need to address the following challenges (C1-C3):

C1: Review semantics Most of the reviews are short, probably with wrongly spelled words or even made-up words. Moreover, different words and expressions can be used for the same purpose. Therefore, it requires to gather enough semantics from user reviews to understand the security-related behaviors.

C2: Users' security concerns Not all reviews reflect security/privacy-related concerns. Recent study from the distribution of user ratings [12] shows that users tend to give positive feedback. Some reviews include vague and pure emotional comments, and some reviews may complain about the quality and the high cost of an app, which have few relations with security and privacy issues. Therefore, how to infer the security-related behaviors from the crowded users' reviews is still very challenging.

C3: Credibility of users The user reviews can be noisy and diverse. User reviews are from different users. Some users may not be responsible for their reviews. Some users are not likely to report the security problems. In other words, not all reviews are informative. How to distinguish different types of users, and make the app-level results more reliable?

To address the above problems in the context of review-to-behavior fidelity, we design and implement a system called AUTOREB, which automatically infers the security-related behaviors by considering the semantics of user reviews and aggregating the security-related behaviors from review-level to app-level via crowdsourcing.

The contribution of this paper is summarized as follows.

• To address the challenge of *review semantics* (C1), at user review level, we use a feature augmentation approach by exploring the "*relevant feedback*" technique, where the correlations and co-occurrences of different reviews are automatically taken into account.

• To address the challenge of *users' security concerns* (C2), we build a classifier to automatically learn the four categories of security-related behaviors (Table 1) using machine learning technique. As long as enough training user reviews with labels are given, our system can automatically infer the security behaviors of mobile apps with accuracy as high as 94.05% at the user-review level.

• To address the challenge of *user credibility* (C3), with the extracted review-level semantic features, we aggregate the behavior annotation results from review-level to app-level based on the expertise of different users. We do not put full confidence on all the users, instead, the wisdom of crowds are automatically learned via crowdsourcing.

We believe that the AUTOREB system provides a generic and universal framework for the analysis of user reviews with enhanced semantic understandings of security-related behaviors. Besides, this framework could be easily extended to analyze the security and privacy issues from the reviews appeared of websites, *e.g.*, `Yelp`, etc. Furthermore we have the following interesting observations.

- *Evaluation* As the first work that infers the security behaviors from user reviews, we made great efforts to collect and label the user reviews. The evaluation of our system on $19,413$ reviews and $3,174$ apps demonstrates that AUTORBF can accurately predict the review-level security behaviors with accuracy as high as 94.04%. Compared to the baseline using keyword-based approach, our work excels a large margin with 51.36% in accuracy, which validates the effectiveness of our system design and deployment.

- *Insight* We list the 50 apps that have user complains about security issues (Table 8). We discuss the relations (in Section 6.4) between the examined four behaviors abstracted from user reviews and those from program analysis. We find that user reviews can reveal how the users think about the security issues, but not necessarily the exact behaviors of apps, which provides an important source to understand mobile app security problems complementary to program analysis.

The remainder of paper is organized as follows. Section 2 states the problem. We present the detailed design of AUTOREB system in Section 3, followed by the design of review-level security behavior inference engine in Section 4. Section 5 presents the design of app-level behavior inference engine. Section 6 presents the experimental evaluation of our system, followed by the discussions of relations with program analysis. Section 7 discusses the related work and finally Section 8 concludes the paper.

2. PROBLEM STATEMENT

On the Android platform, users are known to be very concerned about the security-related behaviors of mobile apps [10]. As is sharply observed in Felt *et al.* [11], automatically understanding the security behaviors from meta data is not an easy job for most of the users. Our goal is to predict the mobile app security-related behaviors from crowded users' reviews, which we call "*review-to-behavior*" fidelity. As compared to mobile app permission analysis or static/dynamic analysis ([7], [46], *etc.*), inferring the security behaviors from the user reviews is still lacking. User reviews provide users' perception on app behaviors, from which we can identify the most pertinent information about users' concerns.

In this paper, we aim to design and implement a system (namely AUTOREB) to address the following problems: *given the reviews from different users on mobile apps, can* AUTOREB *automatically infer the security-related behaviors of apps? And how accurate is it?* This system is expected to help users be aware of security issues, and accurately predict app behaviors both at review level and app level. Besides, given enough training user reviews and

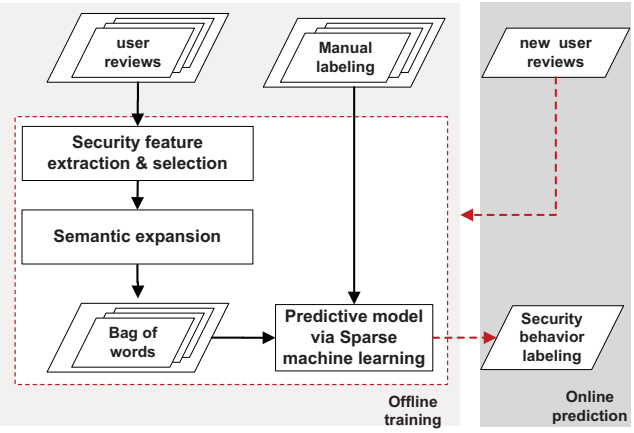


Figure 2: The framework of review-level security behavior inference engine (solid lines are for the training process, and dashed line for the process of annotating new user reviews).

labeled behaviors, the whole process is expected to perform automatically without any human intervention. The developed system can be deployed as an individual app or deployed at `Google play` markets. It summarizes users' concerns about security issues, and can help to improve the trustiness of overall app markets and make the android echo-system better.

An illustration We show several user review examples from `Google play`, and explain the labeled security behaviors.

Review 1: `Seemed ok, but the morning after signing in my account spammed everyone on my contact list with junk mail from me...could be a coincidence...`
Anonymized User 1
Labels: `Data leakage, Spamming`

Review 2: `Why does this need to send sms messages and make calls? Updates not that timely. Uninstalling.`
Anonymized User 2
Labels: `Financial Issue, Over-privileged permission`

For each review, its labels are given automatically by AUTOREB as one or several security behaviors listed in Table 1. The scope of this paper is to infer the "*security*" behaviors from the user reviews. More specifically, we focus on the four categories of security behaviors in Table 1. Since most of the users are not professional security experts, their reviews indicate the understandings of security issues from ordinary users in a vague way. The four categories of security behaviors are summarized from raw user reviews after manual checking. Each category of the behaviors corresponds to a coarse-grained security behavior, which covers a group of fine-grained behaviors illustrated in Table 1.

3. OVERVIEW OF SYSTEM DESIGN

Fig. 1 illustrates the overview of AUTOREB design, which includes two key engines: (a) review-level security behavior inference engine (RLI), and (b) app-level security behavior inference engine (ALI). Fig. 2 and Fig. 3 give the detailed design of each engine, respectively.

Given the user reviews, RLI automatically generates review-level security behavior labels (*i.e.,* four categories of security behavior in Table 1) corresponding to each review using machine learning.

Then the review-level security behavior labels are fed into ALI. ALI automatically generates app-level security behavior labels in consideration of users' credibility. Although it appears similar to the review-level security behavior labeling process and also denoted as the four categories of security issues shown in Table 1, it labels the behaviors of an app by aggregating all different reviewers' opinions via utilizing the wisdom of crowds. We will illustrate the detailed design of each engine in Section 4 and 5, respectively.

4. REVIEW-LEVEL SECURITY BEHAVIOR INFERENCE ENGINE (RLI)

Given enough training user reviews, the goal of review-level inference engine (RLI) is to automatically label a new user review to a particular security behavior category. The design of RLI is depicted in Fig. 2, which consists of two key phases: training phase (denoted in solid line in Fig. 2) and testing phase (denoted in dash line in Fig.2). In the training phase, a classifier is trained using the labeled user reviews, and in the testing phase, new user reviews are fed into the classifier and automatically labeled.

4.1 Training vs. Testing phases

The **training phase** includes the following three key steps.

Step 1: Security-related feature extraction and selection. To infer the security-related behaviors, we first extract the features (including words and phrases) that have close relations with the four categories of security concerns. This is to *address the challenge C2*, since it has been observed from the user review dataset that the complains about the functionality, quality and attractiveness of mobile apps account for the majority of users' concerns. We use a keyword-based approach to select the security-related words and phrases appeared in user reviews. After step 1, only security related features are preserved and selected, and will be used to build a classifier for labeling user reviews.

Step 2: Semantic expansion. To *address the challenge C1*, this step concerns how to understand the user reviews described in different words and phrases but for the similar or the same meaning. By taking advantage of the "*relevance feedback*" technique [44] in information retrieval, we find the words/phrases relevant to security-related features, and expand the original review features by adding new "relevant" words and phrases. This process is iterated until all the "relevant" features are added to the review feature sets. This process is also known as "*feature augmentation*". We aim to capture the semantics of user reviews as much as possible by utilizing the relations among different user reviews. After this step, each user review is abstracted into a feature vector denoted as a bag of word (BOW).

Step 3: Training classifier using sparse machine learning. To *address the challenge C1*, once we have obtained the BOW features after semantic expansion, we train the classifier for prediction of each category of security-related behaviors. Our framework is open to any classifier that requires the BOW feature as the training samples to classify a new user review. Such classifiers include the kNN classifier, SVM classifier, *etc*. In our approach, we use machine learning classifier (*i.e.,* sparse SVM) by exploring the structural sparsity of feature space, which has shown the state-of-the-art performance [41].

Testing phase The output of the training phase is an automated user review classifier. Given new user reviews, we first extract and select the word/phrase features related to security concerns,

and then generate BOW features. Next, we feed the features to the trained sparse machine learning classifier and automatically determine the security behavior category.

Our framework does not require any human intervention. In a dynamic environment where there are new user reviews, the process repeats by retraining the sparse machine learning classifier. Next we will elaborate on the technical details of each step.

4.2 Security feature extraction and selection

In this subsection, we focus on the extraction of the security features. Proper pre-processing is applied to user reviews before feature extraction, including removing stop words and stemming. Since many user reviews are not related to security concerns of mobile apps, a necessary step we need to take is to narrow down the huge number of user reviews to a more feasible set for further analysis and annotation. To achieve this, a coarse-grained filtering is firstly applied to create a subset of suspect user reviews. We adopt a keyword-based approach to preserve a set of the suspect user reviews, *i.e.,* any user review that contains at least one of the predefined keywords will be put into the suspect set.

The key words are manually picked in an iterative way. The initial set of the key words are set to {*security*, *privacy*}. Then new key words are selected from those that have high co-occurrence with current key words. The co-occurrence of key words in user reviews are computed based on the user review corpus L (see Section 6.1) we collected on Google Play. If the co-occurrence of the word-pairs exceeds a large threshold and one of the word-pair is in the suspect set, we add the other word into the suspect set. The rationality behind this approach is that these key words reflect users' concerns on the security and privacy issues. This process is iterated until no more new key words are added into the suspect set.

To avoid mismatch, not only the synonyms are considered, other forms of the keywords (e.g., antonyms, mismatch words, etc.) are also taken into account. For example, *insure* and *unsure* are added for key-word "*security*", *stole* and *stolen* are added for key-word "*steal*", *etc*. We call these words as "security-related" features which capture the semantics of the security behaviors. In this step, instead of using the traditional *black-box* feature selection methods such as F-statistics [6] and mutual information, which rely heavily on the statistical property of the data, our *semantics-aware* feature selection approach identifies the security key words by considering the semantic meaning and correlations among different keywords in a white-box way.

4.3 Semantic expansion

User reviews have some properties which have significant differences from the properly edited documents. User reviews are normally short, probably have wrongly spelled words (as mentioned in Section 4.2), or even made-up words (such as "*privasy, nonsecure*"). Moreover, different words or expressions may be used for the same meaning. These properties may harm the performance of classifier if the extracted keywords are directly used for features.

Our idea is from information retrieval domain. In information retrieval, the query submitted to the search engine could be short, misspelled and vary in the choice of the words, which is very similar to our situation. For the same motivation, a traditional technique called query expansion with pseudo relevant feedback [44] in information retrieval is used to make reviews expansion. Originally, this technique uses the top documents in the retrieval ranking list with respect to the original query as "*relevant*" documents, and these documents are further used to expand the original query, generating a new query resembling the "*relevant*" documents. This query expansion is reported to always have a positive effect on the retrieval performance [44]. A similar process is adopted for review semantics expansion as follows. The process consists of two steps: **(1)** find the "*relevant*" reviews; **(2)** augment the "*original*" reviews with "*relevant*" reviews. The relevance between reviews is evaluated using the similarity in information retrieval model, *i.e.,*cosine similarity between the tf-idf features.

An interesting question would be the scope of the retrieval. Should the candidate "*relevant*" reviews be picked from all other reviews? Or just the reviews from the same app or the same category of apps? Besides, the "*relevant*" reviews should only be those posted before the reviews for expansion, so we are not using the "future" reviews to extend the current reviews. A sufficiently long ranking list from the retrieval engine will be returned, and the scopes and restrictions will be applied to this list to pick the "*relevant*" reviews afterwards.

With a set of "*relevant*" reviews, the review expansion can be conducted by making a sum of the original reviews and the mean of the set of "*relevant*" reviews on feature level as follows:

$$\mathbf{f}_{\text{new}} = (1-\alpha)\mathbf{f}_{\text{old}} + \alpha\frac{1}{|R|}\sum_{\mathbf{f}_{\text{expand}}\in R}\mathbf{f}_{\text{expand}},$$

where $\mathbf{f}_{\text{old}}$ is the Bag of Words (BOW) feature of the original reviews, $\mathbf{f}_{\text{new}}$ is the feature vector after the expansion of user reviews, $\mathbf{f}_{\text{expand}}$ is the feature vector for the expanded user reviews, and R is the set of the "*relevant*" reviews with respect to the reviews for expansion. α serves as a tunable parameter for the degree of effect of the expansion, and can be tuned for the best performance.

In summary, review expansion is an efficient way to explore the relations among reviews, and incorporate *semantics* into the review understanding process. The review expansion also relies on the retrieval engine, the cost of which is normally constant. Hence, the total cost of review semantic expansion is linear to the number of user reviews, which makes it applicable in practical problems.

4.4 Sparse machine learning classifier

After obtaining the BOW feature from the above steps, now we are ready to present the classifier designed to classify user reviews to the security behavior categories defined in Table 1. More formally, given the BOW feature and the annotated four categories, the suspect user review datasets can be represented as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$, where $\mathbf{x}_i \in \Re^p$ denotes the BOW feature vector for each user review, and n is the number of user reviews. Let $\mathbf{Y} = [\mathbf{Y}_{ik}](1 \le i \le n, 1 \le k \le K)$ be the label vectors for user reviews, and $Y_{ik} \in \{0, 1\}$ with $Y_{ik} = 1$ indicating the presence of the k-th label for user review i.

In our problem, more than one security behavior categories can be simultaneously assigned to one user review. This is known as multi-label learning in the machine learning community. Meanwhile, we hope our approach can work well on large-scale user review datasets. To achieve this goal, we use the linear classifier, where the output classification decision is simply a weight vector that separates the data points in the high-dimensional space. Spare linear Support Vector Machine (SVM) has been widely used in large-scale malicious web-site detection [27] and drive-by-download attack detection [38] due to its simplicity, scalability and interpretability. We adopt sparse SVM for solving the user review labeling problem for the exactly same reason.

In sparse SVM model [4], we use hinge loss (1st term in Eq. 1) and enforce the structural sparsity on feature space with LASSO regularization [41] to eliminate feature correlations. Meanwhile we

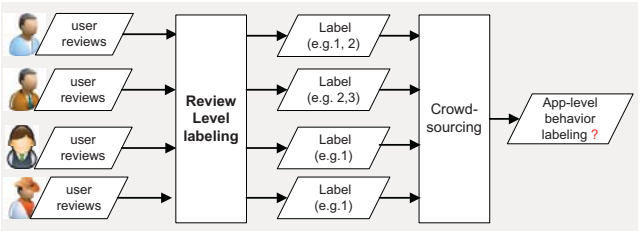


Figure 3: The framework of App-level security behavior inference engine: inferring the app-level labeling via crowdsourcing.

consider the label correlations among different behavior categories. Given a set of n labeled data points $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$, the goal of training spare SVM is to find the projection matrix $\mathbf{W} \in \Re^{p\times K}$ such that:

$$\min_{\mathbf{W}} \sum_{i=1}^{n} (1 - \mathbf{w}_{y_i}^T \mathbf{x}_i + \max_{m\neq y_i} \mathbf{w}_m^T \mathbf{x}_i)_+ + \alpha\Omega_1(\mathbf{W}) + \alpha\Omega_2(\mathbf{W}), \quad (1)$$

and $\Omega_1(\mathbf{W}) = \sum_{m=1}^{k} \|\mathbf{w}_m\|_1$,

$$\Omega_2(\mathbf{W}) = \sum_{ij} (\mathbf{w}^i)^T (\mathbf{D} - \mathbf{S})_{ij} \mathbf{w}^j = \mathrm{Tr}(\mathbf{W}^T \mathbf{L}\mathbf{W}),$$

where $i = 1, 2, \cdots, n$, $m = 1, 2, \cdots, k$, $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_k]$, $(x)_+ = \max\{x, 0\}$. The 1st term in Eq. 1 measures how data fit into the model given the separating surface $\mathbf{W}$, which is directly minimizing the training error. The 2nd term $\Omega_1(\mathbf{W})$ is a penalty for projection $\mathbf{W}$ using $\ell_{1,1}$-norm that tends to yield sparse solutions for feature selection purpose. The 3rd term $\Omega_2(\mathbf{W})$ is to eliminate the label correlations among different categories since some security behaviors occurred simultaneously. As in other classification problems, the iterative shrinkage method and L-BFGS quasi-Newton method [1] are applied for solving the optimization problem of Eq.(1). The convergence of the algorithm can be rigorously proved as that in [1].

5. APP-LEVEL SECURITY BEHAVIOR INFERENCE ENGINE (ALI)

The review-level security behavior inference model has annotated each review as one or several security behaviors shown in Table 1. Different users may have labeled the same app to different labels. A natural question that follows is: *what is the final label at app-level in consideration of all users?* We suggest using crowdsourcing technique to aggregate the security labels from user review level to app level.

Crowdsourcing [37] is a technique to infer the true labels of a given item from the annotations of multiple works, where the annotations are assumed to be low quality and may contradict with each other. Crowdsourcing has been used for the improvement of the product quality, semi-automatic generation of software programs [3], etc. Although there may be substantial disagreement among different annotators, we aim to learn from "*crowds*" to annotate the app by considering different opinions. We model the mapping of labels from user-review level to app-level as a crowdsourcing problem, treating user review behaviors as labels from works/annotators.

More formally, the app-level annotation problem by different users can be described as follows. For each user u with respect to app i, its reviews $\mathbf{d}_{u,i} \in \{0, 1\}^r$ is denoted as a r-dimensional set (r = 4 in our case) corresponding to the four categories of security behaviors shown in Table 1, i.e., $\mathbf{d}_{u,i} = \{d_{u,i}^1, \cdots, d_{u,i}^r\}$, where r is the size of label set, and $d_{u,i}^\ell$ is the binary auto annotation result of the ℓth classifier for the review given by user u to app i, i.e.,

$$d_{u,i}^\ell = \begin{cases} 1; \text{if user } u \text{ labels app } i \text{ as label } \ell \\ 0; \text{otherwise} \end{cases} \quad (2)$$

Given all the above labels $d_{u,i}$ generated from different reviewer u with respect to each app i, we need to learn a mapping which automatically projects the review-level labels $\mathbf{d}_{u,i} = \{d_{u,i}^1, \cdots, d_{u,i}^r\}$ to the app-level security labels $\mathbf{y}_i = [y_i^1, y_i^2, \cdots, y_i^r]$, i.e., $\{d_{u,i}\} \rightarrow \mathbf{y}_i$, where $y_i^\ell = \{0, 1\}, 1 \leq \ell \leq r$. If security behavior ℓ appears in app i, $y_i^\ell = 1$; otherwise, $y_i^\ell = 0$.

5.1 Why not majority voting?

One naive way is to treat all the users equally, *i.e.,* simply trust all the users and combine all the reviews' comments no matter they are trustful or not. For example, if most of the users say that this app has "*data-leakage*" issue, we say this app has "*data-leakage*" issue. This is known as majority voting [37] in crowdsourcing.

To treat each user equally by averaging over the labels of all the users, the final label for each app i computed from users $\{1, 2, \cdots, j, \cdots, m\}$ will be

$$y_i^\ell = \frac{1}{m} \sum_{j=1}^{m} d_{j,i}^\ell.$$

By thresholding y_i^ℓ, this method provides binary crowdsourcing result. One major problem with this method is that it treats each user equally and hence the contribution of experts would be overwhelmed by the crowds' less valuable opinions. In practice, some users are more trustable while others may not be very responsible for their reviews, or even fraudulent and deceptive.

5.2 Crowdsourcing by giving more credit to trustworthy users

We use a two-coin [37] model to annotate the apps from review-level to app-level, which pays more credit on trustworthy users. More specifically, the probability that a worker labels an app correctly is assumed to follow *bernoulli* distribution, one for the true positive label, and the other for negative. The advantage is that we can give a more accurate prediction of the security behavior of each app by taking into account the credibility of difference users.

There are two cases. For a specific app i, we use α_u^ℓ to denote *sensitivity*, *i.e.,* the probability that a user u would label an app with security behavior ℓ under the condition that the security behavior ℓ really exists; and we use β_u^ℓ to denote *specificity*, *i.e.,* the probability that a user u would label an app with no security behavior ℓ under the condition that the security behavior does not exist. Mathematically,

$$\alpha_u^\ell = \Pr(d_{u,i}^\ell = 1 | y_i^\ell = 1), \quad (3)$$

$$\beta_u^\ell = \Pr(d_{u,i}^\ell = 0 | y_i^\ell = 0), \quad (4)$$

where i $(1 \leq i \leq n)$ refers to each app, u $(1 \leq u \leq m)$ refers to each user, and ℓ $(1 \leq \ell \leq r)$ refers to each behavior label. Parameters $\alpha = [\alpha_u^\ell], \beta = [\beta_u^\ell]$ can be learned from he training data using EM algorithm according to Maximum Likelihood Estimation (MLE), and y_i^ℓ is then computed via Bayesian rules.

According to the maximum likelihood principle, this approach maximizes the following objective function:

$$J(\theta, \mathbf{y}) = \max_{\theta, \mathbf{y}} \Big[\ln \Pr([d_{u,i}] | \theta, \mathbf{y}) + \ln \Pr(\theta) \Big],$$

where $\theta = \{\alpha, \beta\}$ is the parameter as in Eqs.(3, 4), and

$$\Pr([d_{u,i}]|\theta, \mathbf{y}) = \prod_{i=1}^{n}\prod_{l=1}^{r}\Big[\Pr(y_i^\ell = 1|\theta)\prod_{u=1}^{m}\Pr(d_{u,i}^\ell|y_i^\ell = 1;\theta) + \Pr(y_i^\ell = 0|\theta)\prod_{u=1}^{m}\Pr(d_{u,i}^\ell|y_i^\ell = 0;\theta)\Big]; \quad (5)$$

$$\Pr(d_{u,i}^\ell|y_i^\ell = 1;\theta) = (\alpha_u^\ell)^{d_{u,i}^\ell}(1-\alpha_u^\ell)^{1-d_{u,i}^\ell}, \quad (6)$$

$$\Pr(d_{u,i}^\ell|y_i^\ell = 0;\theta) = (\beta_u^\ell)^{1-d_{u,i}^\ell}(1-\beta_u^\ell)^{d_{u,i}^\ell}. \quad (7)$$

After derivations using EM algorithm [5], finally the app-level behavior indicator y_i^ℓ is expressed as:

$$y_i^l = \frac{a_i^\ell \Pr(y_i^\ell = 1|\theta)}{a_i^\ell \Pr(y_i^\ell = 1|\theta) + b_i^\ell \Pr(y_i^\ell = 0|\theta)}, \quad (8)$$

where a_i^ℓ is the likelihood of app i getting label ℓ, b_i^ℓ is the likelihood of app i not getting label ℓ, *i.e.,*

$$a_i^\ell = \prod_{u=1}^{m}(\alpha_u^\ell)^{d_{u,i}^\ell}(1-\alpha_u^\ell)^{1-d_{u,i}^\ell}, \quad (9)$$

$$b_i^\ell = \prod_{u=1}^{m}(\beta_u^\ell)^{1-d_{u,i}^\ell}(1-\beta_u^\ell)^{d_{u,i}^\ell}, \quad (10)$$

and $\Pr(y_i^\ell = 1|\theta)$ is the prior probability [2] for app i labeled ℓ, and $\Pr(y_i^\ell = 1|\theta) = 1 - \Pr(y_i^\ell = 0|\theta)$.

5.3 Determining app-level behavior via y_i^ℓ

However, y_i^ℓ is not directly useful for determining the app-level behavior. Although we utilize crowdsourcing technique to aggregate review-level labels to app-level labels, the review-level labels are not really users' annotations, but auto annotations by a trained classifier (in Section 2). The reviews given by users are not originally annotations for the security issues. Therefore, the prior probability of finding a user review mentioning one of the four security issues is quite low, even given the fact that the security issues exist for the app. This is easy to understand since the user may not actually encounter the security issue when using the app, or may not give a review after encountering the issue, or the review-level classifier fails to recognize the *semantic* meaning in the review. As a result, y_i^ℓ may not be taken as the probability of the app having the security issues, but more as a security-risk ranking score for comparing the security risks of the app having the issues against others.

In order to get a clear output of whether the app has the security behavior or not, a threshold is needed to be tuned for each label to provide the binary prediction. This threshold can be determined through active manual annotation, which is discussed in Section 6.

6. EXPERIMENT

We present the experimental results of AUTOREB. AUTOREB aims to bridge the gap between users' understanding of apps and apps' actual behaviors. To evaluate the effectiveness of AUTOREB, we compare the behaviors inferred at both review-level and app-level against those labeled by human beings, and then make a quantitative study on the performance of the system. More specifically, we design the experiment and answer the following two questions:

- **RQ1:** What is the prediction performance using review-level security behavior inference model?
- **RQ2:** Can we get the app-level security behavior annotation results via crowdsourcing? What is the credibility of different users? Will users report the security problems?

6.1 Data Collection

Our dataset was collected from Google Play. On Google Play, user reviews about Apps are publicly available. Once we obtain the Google ID of a user, we can locate all the Apps that the user has reviewed. For each retrieved App, we crawled its reviews from Google Play. The crawler was written in python. One dataset L was collected during November 2013, containing $19,413$ user reviews on $3,174$ apps from Google play. We use this dataset to validate the review-level security behaviors. This dataset was annotated manually[3] by several mobile app professionals in two months. Each user review was given a label by three annotators. If three annotators reached a consensus, the user review was labeled as a specific label. Otherwise, the three annotators would discuss and reach a consensus on the controversial user reviews. Each user review was labeled with either one, several or none of the above labels described in Table 1. The annotation was conducted in consideration of the meanings of the reviews rather than the actual app behaviors. The statistics of dataset L is listed in Table 2.

Dataset	#app	#Review	Mean	Max	Min
L	3,174	19,143	6	4,500	1

Table 2: Dataset L details. The mean, max and min are statistics of for the number of reviews per app. The mean number of reviews per app is small because we have already filtered out the reviews with over 3 ratings, which are the majority of them.

Moreover, we collected another dataset D to validate the effectiveness of our approach on app-level security behavior inference. This dataset was collected through December 2013 to May 2014, containing $12,783$ apps[4] with $13,129,783$ reviews from $2,614,186$ users. Dataset D has no intersection with the apps in dataset L. However, we did not hire enough labors to manually label dataset to get the ground truth of security behaviors with respect to each app. Table 3 summarizes the statistics of dataset D. In addition, Fig. 4a shows the distribution of the number of apps over the number of reviews, in which the peak at 4000 is artificial, due to the fact that our crawler is set to only crawl the first $4,000$ reviews for each app. Fig. 4b shows the distribution of apps over the 5 rating values from the reviews. It is clear that most users tend to give high ratings to the apps once they decide to give them reviews. This, however, also implies that most of the reviews are not valuable for our purpose of detecting security issues.

Dataset	#app	#Review	#User	Max	Min
D	12,783	13,129,783	2,614,186	4,000	1

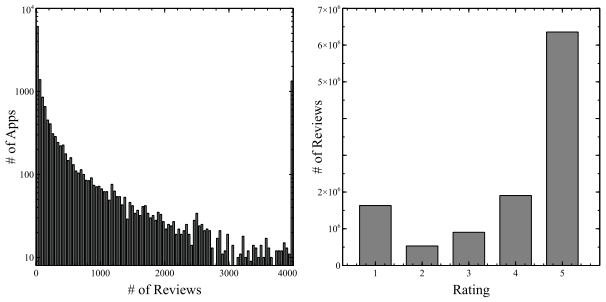
Table 3: Dataset D details. The mean, max and min are statistics of the number of reviews per app. The max number of reviews $(4,000)$ is artificial, because our web crawler is set to crawl only the first 4000 reviews for each app.

We next describe the evaluation results of AUTOREB in identifying review-level and app-level security behaviors.

[2] Initialization using $\Pr(y_i^\ell = 1|\theta) = \frac{1}{m}\sum_{j=1}^{m} d_{j,i}^\ell$, α, β represent the probability of a binary event, thus a natural choice is the Beta distribution. *i.e.,* $\Pr(\alpha|a_1, a_2) = \text{Beta}(\alpha|a_1, a_2)$, $\Pr(\beta|b_1, b_2) = \text{Beta}(\beta|b_1, b_2)$, where a_1, a_2, b_1, b_2 are the parameters for Beta distribution.

[3] The ground-truth of review labels was from the manually labeling mobile app reviews.

[4] The apps were selected in a random manner.



(a) Histogram for the number of apps *vs.* the number of its reviews in the collected dataset.

(b) Histogram for the number of reviews with different ratings in the collected dataset.

Figure 4: Statistics of the collected dataset D.

Label	Keywords
`Spamming`	ad, spam, notif, advertis, advert, spammi, add, money, secur, push, etc
`Financial Issue`	text, deduct, sm, bought, paid, took, privat, txt, taken, charg, purchas, etc
`Over-privileged Permission`	permiss, access, money, read, info, privaci, regist, camera, need, want, contact, requir, ask, locat, data, internet, request, credit, email, call, necessari, etc
`Data leakage`	permiss, privaci, info, hack, access, contact, money, fool, lie, id, ground, inform, lockscreen, steal, wit, wall, camera, data, requir, phish, internet, licenc, locat, etc

Table 6: Sample keywords used in "keyword based approach".

6.2 RQ1: Review-level security behavior inference

Experiment setting The evaluation of the proposed review-level classification is conducted on the annotated dataset L. As a supervised method, training dataset is required for training the model. The whole data set is randomly split in a $50\%/50\%$ manner into a training set with $10,893$ reviews and a testing set with $8,520$ reviews. This splitting is at app-level, so the reviews for the same app can only be either in the training or testing set, and the number of reviews in the two sets are not even.

In solving the optimization problem of security behavior inference, a five-fold cross validation method is adopted for finding the best αs in the training set. For semantic expansion, *tf-idf* features with cosine similarity is adopted for the retrieval model and Lemur[5] is used as the actual tool for the word retrieval. Time constraint is enforced to prevent review expansion with "*future relevant*" reviews. The indexes of retrieval model are built for the two sets separately so that the model parameters such as document number and IDF values will not interfere between the sets. The mixture ratio α and the size of "relevant" document set R for the expansion are fixed by using five-fold cross validation in the training set for each label along with the scope. The size of R is selected from $\{1, 3, 5, 10\}$.

Evaluation measurement The metrics used for evaluation are precision, recall and F1 value. Let the values of true positives, false positives, true negatives, and false negatives be TP, FP, TN and FN, respectively w.r.t. each label. TP indicates that AUTOREB correctly labels the review as a security issue if the review is. FP indicates that AUTOREB incorrectly labels a review as a security issue if the review is not. TN indicates that AUTOREB correctly labels a review as a non-security issue if the review is not. FN indicates that AUTOREB incorrectly labels a review as a non-security issue if the review is.

The precision metric is defined as Precision $= \frac{TP}{TP+FP}$, and the recall metric is Recall $= \frac{TP}{TP+FN}$. The F_1 measure is the harmonic mean of precision and recall, i.e., $F_1 = \frac{2TP}{2TP+FP+FN}$, and the accuracy is Accuracy $= \frac{TP+TN}{TP+FP+TN+FN}$. Larger values of these metrics suggest better classification results. Thus the values of these metrics indicate how well the security behavior inference results match with the human annotated results.

Experiment result analysis Table 4 shows the performance of our method in identifying different categories of security issues. The results indicate that out of 8,520 reviews, AUTOREB effectively labels the security-related behaviors with average precision, recall, F1 score and accuracy of 80.10%, 82.46%, 81.26% and 94.05%, respectively. We are aware that the high value of TN is due to few user reviews on security issues, which generally produces a very high accuracy. After carefully looking at different categories of security-related behaviors, we find that AUTOREB can identify "spamming", "over-privileged permission" and "data leakage" issues with average precisions of 83.84%, 78.25%, 77.97%, respectively, and with average accuracies of 91.96%, 95.99%, 93.46%, respectively. This confirms the effectiveness of our method.

An exceptional case is the detection of security issues related to "finance", the precision of which is not as high as the above three categories. When we check the user reviews, we find that the false positives and false negatives are higher than those computed from the other three security categories. We find that many users actually do not complain about the "financial issue", but their reviews are labeled as "financial issues". Also, some users complain about the "financial issues" without using any words related to "*buy, purchase, pay, paid, etc.*", which misleads AUTOREB in making the decision.

Comparisons against keyword-based method For comparison, we use keyword-based method as a baseline to show the necessity of using the proposed semantic expansion techniques and advanced machine learning method in AUTOREB. We agree that keyword-based approach can be extended to be more advanced. As the most fundamental and intuitive method, we evaluate the performance of keyword-based method. A number of keywords are manually selected for each security label as in Table 6, and this approach predicts all the reviews that contain at least one of the keywords to be positive for the underlining label. These keywords are stemmed [6] in pre-processing, so that the words from the same origin may be matched to each other. Following the same evaluation measurement used in AUTOREB, we summarize its results in Table 5.

We then compute the performance improvement using AUTOREB against the keyword-based approach. Let ∇Precision, ∇recall, ∇F1 and ∇ACC be the performance differences between our approach and the keyword-based approach corresponding to different metrics. Table 7 shows the performance differences in identifying security issues between AUTOREB and the keyword-based approach. For all the four categories and overall performance, AUTOREB gains significant performance

[5] `http://www.lemurproject.org/`

[6] `http://en.wikipedia.org/wiki/Stemming`

Label	# size	TP	FP	FN	TN	Precision	Recall	F1	ACC
Spamming	2,788	2,574	496	214	5,236	83.84%	92.32%	87.88%	91.66%
Financial Issue	578	337	179	241	7,763	65.31%	58.30%	61.61%	95.07%
Over-privileged Permission	711	511	142	200	7,667	78.25%	71.87%	74.93%	95.99%
Data leakage	1,258	977	276	281	6,986	77.97%	77.63%	77.82%	93.46%
Sum	5,335	4,399	1,093	936	27,652	N/A	N/A	N/A	N/A
Average	N/A	N/A	N/A	N/A	N/A	80.10%	82.46%	81.26%	94.05%

Table 4: Evaluation on different metrics of AUTOREB. # size denotes the number of positive samples with respect to the label, and ACC denotes accuracy.

Label	# size	TP	FP	FN	TN	Precision	Recall	F1	ACC
Spamming	2,788	2,780	5,619	8	113	33.10%	99.71%	49.70%	33.96%
Financial Issue	578	549	3,282	29	4,660	14.33%	94.98%	24.90%	61.14%
Over-privileged Permission	711	706	6,012	5	1,797	10.51%	99.30%	19.00%	29.38%
Data leakage	1,258	1,172	4,490	86	2,772	20.70%	93.16%	33.87%	46.29%
Sum	5,335	4,399	1,093	936	27,652	N/A	N/A	N/A	N/A
Average	N/A	N/A	N/A	N/A	N/A	21.16%	97.60%	34.78%	42.69%

Table 5: Evaluation on different metrics using keyword-based method. # size denotes the number of positive samples with respect to the label, and ACC denotes accuracy.

Label	▽Precision	▽ Recall	▽F1	▽ACC
Spamming	50.74%	−7.39%	38.18%	57.70%
Financial Issue	50.98%	−36.68%	36.71%	33.93%
Over-privileged Permission	67.74%	−27.43%	55.93%	66.61%
Data leakage	57.27%	−15.53%	43.95%	47.17%
Average	68.94%	−15.14%	46.48%	51.36%

Table 7: Performance differences between our approach and keyword-based approach. ▽ indicates the performance difference in terms of different metrics.

improvement in terms of precision, F1 and accuracy. We observe that the recall of our approach is generally lower than that of the keyword-based approach because our approach has higher false negatives. However, the keyword-based approach has extremely high false positives since many user reviews that include sensitive words do not necessarily cause security-related issues. Considering the tradeoff between the precision and recall, our approach outperforms the keyword-based approach significantly, *i.e.,* an average of 46.48% performance improvement in terms of F1, and an average of 51.36% performance gain in terms of accuracy.

It does not make much sense to evaluate each technique in our system separately for the best system performance since different components jointly contribute to the improvement of the overall system performance. Only one technique is not sufficient to improve the system performance.

6.3 RQ2: App-level security behavior inference

Experiment setting With the trained classifier from the review level experiment, *e.g.,* the classifiers evaluated in Table 4, all the reviews in dataset D are automatically annotated with the four labels. We apply the crowdsourcing algorithm introduced in Sec. 5 to generate app-level result $\mathbf{Y}$ for each app in D while evaluating the credibility of the users in D.

Experiment results In order to determine the thresholds for the four labels, we labeled about 50 apps for each label and tuned the thresholds to best classify those labeled apps. These apps are randomly selected from the apps whose (y_i^ℓ) values are within the range of $[\text{mean}(y_i^\ell) - \text{std}(y_i^\ell), \text{mean}(y_i^\ell) + \text{std}(y_i^\ell)]$, where $\text{mean}(y_i^\ell)$ is the average values of y_i^ℓ, and $\text{std}(y_i^\ell)$ is the standard deviation of y_i^ℓ.

In this way, we get all the security labels for each app at app-level. For example, CallToPark is a web app that aims to make personal payment and account management faster. Its y_i^ℓ computed from crowsourcing approach (Sec. 5) is

$$[0; \quad 0.997; \quad 0.002; \quad 0.003],$$

corresponding to the four categories of security issues of *spamming*, *financial issue*, *over-privileged permission* and *data leakage*. The results indicate it has 3 security issues (*i.e.*, *financial issue, over-privileged permission and data leakage*) out of the four. The serious degrees of security issues are reflected by the values of these corresponding numbers. The greater values indicate the more serious security issues from user perspective, such as more complains and dissatisfaction. We show a number of apps that have severe security problems using our analysis tool[7] in Table 8.

Analysis of user credibility A by-product of crowdsourcing in AUTOREB is the learned parameters for each user, indicating how reliable the users' reviews are in our system. Figs. 5, 6 show the distributions of the number of users based on the trained parameters. The distribution of the number of users over the learned α values shows three peaks (Fig. 5).

The *middle* peak is around 0.5, which is the manually set prior value. In fact, this peak is the expectation of the distribution, indicating the behavior of majority users.

The *lower* peak shows the users that are less likely to report the issues in reviews, hence are given less credibility, and contribute

[7]We are aware that the performance of AUTOREB is not perfect, and some of the labels may have false positives or false negatives. For example, the voicerecorder app (APK name: com.tokasiki.android.voicerecorder) actually has spamming issue, however, very few users complain ads, and AUTOREB does not label it as "spamming".

App Name	Spamming	Financial Issue	Over-privileged Permission	Data Leakage
com.chirpme.swipe	✓	✓	✓	✓
com.BeatMakerMicProFree	✓	×	✓	✓
com.usablenet.android.aetna	×	×	✓	✓
org.dyndns.devesh.flashlight	×	×	✓	✓
org.geometerplus.zlibrary.ui.android.squid	×	×	✓	✓
com.Vertifi.Mobile.P211391825	×	×	✓	✓
shipmate.norwegian	×	×	✓	✓
com.seattletimes.android.SeattleTimesMobileNews	✓	×	×	✓
com.sicecommentr.buttonfootball	✓	×	×	✓
org.dmfs.android.contacts	✓	×	×	✓
com.grainger.mobile.android	×	×	✓	✓
com.netpatia.android.filteredcompass	×	×	✓	✓
pl.pawelbialecki.smartsysteminfo	×	×	✓	✓
com.qwapp.android.netping	×	×	✓	✓
com.comerica.mobilebanking	×	×	✓	✓
com.livewallpaper.livewallpaper.garmaplelef	×	×	✓	✓
org.usaswimming.deckpass	×	×	✓	✓
poker.hands.order	×	×	✓	✓
com.foundersfcu.mobile	×	×	✓	✓
jp.picolyl.led_light	×	×	✓	✓
com.uvukeclub.fretboard	×	×	✓	✓
com.funtrigger.mp3tag	×	×	✓	✓
jp.gr.java_conf.hanitaro.tide	×	×	✓	✓
com.calories.burned.calculator	×	×	✓	✓
com.kakapo.slotsfarm	×	✓	×	✓
com.mdt.doforms.android	×	✓	×	✓
com.tokasiki.android.voicerecorder	×	✓	✓	×
la.droid.periodic	×	×	✓	✓
com.electronmagic.animalnumbers	×	×	✓	×
de.j4velin.vibrationNotifier	✓	✓	×	×
com.kosenkov.alarmclock	×	×	✓	✓
com.qxmd.calculate	×	×	✓	✓
cx.hell.android.pdfview	×	✓	✓	×
com.leumi.leumiwallet	×	×	✓	✓
com.fallacystudios.zombiehandbooklite	×	×	✓	✓
com.giraone.encmanlite	×	×	✓	✓
com.fiberlink.maas360.android.control	×	×	✓	✓
com.imprezzio.android.CallToPark	×	✓	×	✓
com.punyweakling.skins.storm	×	×	✓	✓
com.snowbee.colorize.hd	×	×	✓	×
com.ap.postdispatch	×	×	✓	×
com.upstartmobile.Cabulous	×	✓	×	×
com.raycom.wxix	✓	×	×	×
com.mobidia.android.mdm	×	×	×	✓
com.ifs.mobilebanking.fiid5334	×	×	×	✓
com.handcent.nextsms	×	✓	×	×
com.picobrothers.whcf	×	×	✓	×
com.ihunda.android.hiit	×	✓	×	×
com.kemsoft.myconsultant	✓	×	×	×
com.rollcallz.fbcheckin	×	×	×	✓
com.imprezzio.android.CallToPark	×	✓	✓	✓
fm.gigbeat.android	✓	×	×	×

Table 8: A list of 50 apps that have the security problems labeled by our method at app level, where ✓ denotes that the app has the corresponding security problem and × indicates that there is no corresponding security problem.

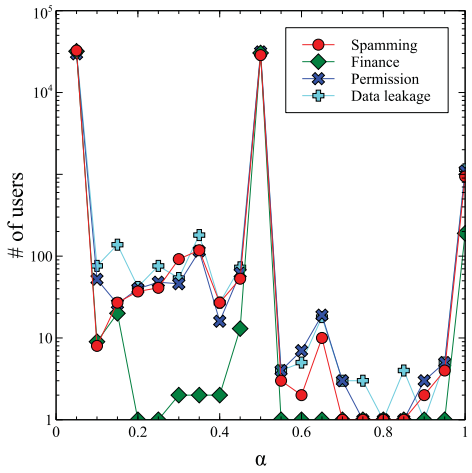


Figure 5: Distribution of # of users (log scale) over the learned α.

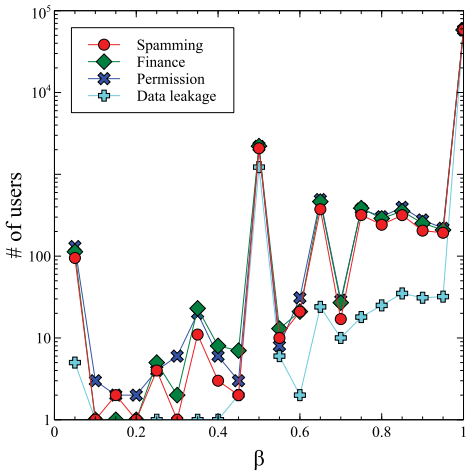


Figure 6: Distribution of # of users (log scale) over the learned β.

less in the crowdsourcing process. These users may more focus on the functionality and attractiveness over the security issues of the app, or simply do not want to make the efforts of reporting the issues.

The *higher* peak shows the users that are much more likely to report a security issue. They are considered as experts and their reviews are considered with high weights in the crowdsourcing process. These users are sensitive to those four security issues and are more willing to report them once they find them.

The distribution of users over the learned β values (Fig. 6) shows not much information, compared to that over α. Although α and β look symmetric in Eqs.(3, 4), in fact they are quite different in data. Users are not obligated to report security issues in reviews, and their default behavior is to report nothing. Therefore, β, the value of the probability of reporting nothing when the app does not have the issues, has a very high expectation with lower variance. As a result, the distribution of β has a clear tendency towards 1, and the three peaks are less obvious than those over α (note that the number of users is in log scale, and the peak near 1 is actually much more significant).

It is worth noting that the parameters are learned based mainly on how well the users' opinions match the majority. It does not reveal the exact reason why the credibility is low for some users. So we are not considering those users with low credibility as fraud users. It is just that their opinions carry less value for these four underlining security problems, and it is perfectly fine for the users not reporting any of the issues in reviews.

This also shows why we use the two-coin crowdsoucing model instead of the simple majority voting in the crowdsourcing process. With the distinction of different types of users, the app-level results would be much more reliable. User credibility can be used to detect the low quality annotators or spammers who assign labels randomly (*e.g.*, without actually looking at the apps).

6.4 Discussion

Utility As demonstrated in the experiment results, generally users have different psychological expectations for different apps, and do not necessarily concern about the same security issue [22]. We believe that it could benefit end-user security[8]. On the other hand, if market owners implement such a feature in app store, it is potentially counterproductive for the revenue, since some apps have security issues!

Relations with program analysis We acknowledge that program analysis (*e.g.,* [8], [15], [2], [14], [31], [17], [7], [46]) reveals the security behaviors that actually happened. However, this paper focuses on a different perspective. For the sake of completeness, we discuss the differences between the behaviors from user reviews and those from program analysis regarding the four security behavior categories.

The first behavior, ***spamming***, has different meanings from the ads libraries detected from program analysis. Spamming refers to both foreground and background ads and spamming behaviors, including ads in the notification bar, ads via email, ads via SMS, pop-up ads, fishing, *etc*. Program analysis can easily find ads libraries [16] used in mobile apps. Ads are present in most free Android apps, so it would be surprising if users would consider all ads as spamming. In other words, spamming can not be simply conflated with ad libraries. Given the ads library used by apps, why the user complains about some apps but not others? The ads detection task defined in ads analysis from program analysis may not comply with what users really feel about the ads.

The second behavior, ***financial issue***, is quite difficult to be detected using program analysis. Financial issues include complains about In-App-Purchase (IAP), free app to premium update and others that cause the users actual money lost in external contexts other than the running apps. For example, users may pay for IAP or upgrade but get nothing as described, and a bank app may transfer money for users but only deduction happened without deposit. These issues may be caused by program bugs but also by intentional fraud. These issues bother users' experience and cause security concerns that are beyond the running apps in users' devices.

The third behavior, ***over-privileged permission***, has different meanings for users and static analysis. For users, it means that the user does not believe that an app should request a permission for its intended functions. For static analysis, it means that the app requests permissions not required by its API calls. We can expose this in the following example. When an alarm clock app requests the camera permission, the user would always deem it as over-privileged permission, no matter how the app is written. However,

[8]Even so, different users may react differently after seeing such labels. For example, an attacker may even intentionally use security keywords to make the app have a security label; another attacker may begin to write more negative reviews.

the app can fly under the radar of static analysis by invoking the camera API. We agree that the latter is more dangerous because the app actually uses the dangerous permission instead of merely requesting it.

The last behavior, ***data leakage***, refers to accessing sensitive information without user's acknowledgement. For example, in an app, users' location information can be accessed and sent to the third party without users' awareness. Our detection is more from users' perspective. We are aware that user review analysis can not capture sensitive data flow and how data are sent out of phone as static or dynamic analysis does. However, we do know whether the users feel comfortable if their personal data are "*leaked*".

In summary, user review analysis is an important complement to program analysis due to the facts that:

• *The user reviews reveal how the users feel about their experiences. Users' expectations play a big role on how much the users can tolerate the apps' behaviors. Similar app behaviors may receive different responses from users.*

• *External factors of the app behaviors when using are much easier revealed by user reviews.*

• *Privacy issues are relative and personal. The borderline between privacy-intruding and tolerable misbehaviors is fuzzy and highly dependent on users' subjective expectations. Hence user reviews may provide an important complementary source for determining the threshold with the objective app behaviors revealed by program analysis.*

6.5 Limitations

Firstly, we need human efforts to label whether each review states about the four categories of security-related behaviors. The operation of AUTOREB depends on the availability of user reviews of mobile apps. When new categories of security-related behaviors are added, we can refer to the semi-supervised learning technique [43] to free people of labeling large amount of new training samples.

Secondly, the four categories of security behaviors are defined based on common sense. Since this is the first work about app risk analysis using user reviews, no previous works have discussed about what kind of behaviors users are most concerned with. As a future work, we will address this limitation by conducting a more thorough user study to support other categories of risk behaviors.

Finally, similar to other machine learning based systems, thresholds are used for feature learning and classification of user reviews. Determination of these thresholds generally needs domain knowledge and cross validation.

7. RELATED WORK

Machine learning for security Since the initial intrusion detection work done by Lee *et al*. [21], machine learning has been used to solve security problems, such as worm signature generation [32], malware classification [20] [45], software plagiarism detection [24], *etc*. Recently, machine learning has been used to understand the mobile app permissions and behaviors through analysis on the meta-data such as descriptions [33] [36], and contextual API dependency graphs [48], *etc*. In our work, we treat the user review understanding as a machine learning problem, and our approach is unique in that we consider the semantics of users' reviews when automatically inferring the security behaviors of mobile apps.

Mobile app risk assessment using meta-data Most (if not all) existing works on the analysis of mobile app risks from meta data focus on the descriptions of mobile apps [19] [36] which are provided by developers. WHYPER [33] predicts the risk assessment of mobile apps based on the analysis of the descriptions of apps from natural language processing perspective using first-order logic. Autocog [36] uses natural language technique and machine learning based approach to access the description-to-permission fidelity of mobile apps. Recently, Lin *et al.* [22] introduce a new model for privacy by capturing users' expectations via crowdsoucing, where the users' expectations of mobile apps are captured from user study rather than the user reviews by millions of users across the world as in AUTOREB. Sentiment analysis [23] has advantages in extracting and identifying subjective information from text corpus, which could be used to find more fine-grained user behaviors.

Annotations/Comments from users can be seen as decision aids that provide how others feel about the content. Muralidharan *et al.* [28] found that annotations are most helpful when they are from people known to have expertise in the current domain. Nelson *et al.* [30] found that expert annotations improved learning scores for exploratory learning tasks. User reviews can be viewed as "human annotations", which reflect how users feel about the apps. Existing studies show that if the app risk is too high, users decide not to install the app [42]. Moreover, the quality of mission critical app is not very high [18]. Unfortunately, users are unaware of app risks. As a new perspective, user reviews provide cues to help people evaluate the risks of apps.

Mobile security using code analysis There exist many works on permission analysis [9] [49], code analysis [15] [2], [47], and run-time behavior analysis [17] [7] [46] for security and privacy enhancement on mobile phones. Livshits *et al.* [25] proposed a two-prong static analysis algorithm for correct resource access prompt placement in smartphones. Nan *et al.* [29] proposed a run-time security enhancement mechanism for user input data protection on smart phones. However, these works are orthogonal to ours. AUTOREB explores the user reviews to understand users' real concerns for security issues, which can be used as a complementary tool to program analysis for improving user experiences and interactions on mobile apps.

NLP and data mining in software engineering DynaMine [26] was developed to find common error patterns by mining software revision histories. Qi *et al.* [35] point out that the earlier well-known advanced search-based technique for automatic bug repairing doesn't even beat a naive random deletion technique. Gegick *et al.* [13] leverage the existing basic NLP and text mining techniques to identify security bug reports among all bug reports. Tan *et al.* [40] take the first step in automatically analyzing comments written in natural language and use them to detect inconsistencies between comments and source code, indicating either bugs or bad comments. Pandita *et al.* [34] analyze natural language API documents to extract specifications that target towards generating code contracts. Slankas *et al.* [39] combine the techniques of information extraction and machine learning to discover patterns that represent access control rules in sentences.

There are several key distinctions between our approach and those works. Firstly, the texts in user reviews are much more free and messy compared to code comments and documentations in these works. The advanced NLP method such as POS tagging, syntax parsing and ontology tend to fail on the short, unstructured pieces of texts with misspelling and made-up words, since they are not written to be documented. Secondly, we adopt pseudo relevant feedback in information retrieval, due to the similarity between the freely conscripted query string and the messy review text. We do not claim there is no appropriate NLP technique that can improve the performance. Finally, we apply the techniques in machine learning and information retrieval to solve the problems which

may be hard to solve with other methods, *e.g.,* evaluating users' credibility on certain security issues from their reviews.

8. CONCLUSION

In this paper, we propose the system AUTOREB that understands the review-to-behavior fidelity in Android apps. AUTOREB can infer the mobile app security behaviors from the apps' reviews from different users via crowdsourcing. The machine learning based algorithm and crowdsourcing techniques are able to mine the relations between user reviews and app security behaviors. To our knowledge, AUTOREB is the first work that has the capability to accurately detect the apps' security behaviors from user reviews. Our system achieves the average accuracy as high as 94.05% in inferring the security behaviors from user reviews. We also get credibility for different users at app-level. Our study provides valuable insights and quantitative analysis in understanding the app behaviors from the users' perspective.

Acknowledgement We thank the anonymous reviewers and our shepherd Ben Livshits for their valuable comments.

9. REFERENCES

[1] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck. Mast: Triage for market-scale mobile malware analysis. In *WiSec*, pages 13–24, 2013.

[3] R. Cochran, L. D'Antoni, B. Livshits, D. Molnar, and M. Veanes. Program boosting: Program synthesis via crowd-sourcing. In *POPL*, Jan. 2015.

[4] A. Cotter, S. Shalev-shwartz, and N. Srebro. Learning optimally sparse support vector machines. In *ICML*, volume 28, pages 266–274, 2013.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[6] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Willey & Sons, New Yotk, 1973.

[7] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.

[8] W. Enck, D. Octeau, P. Mcdaniel, and S. Chaudhuri. A study of android application security. In *USENIX Security Symposium*, 2011.

[9] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *CCS*, pages 627–638, 2011.

[10] A. P. Felt, S. Egelman, and D. Wagner. I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns. In *SPSM*, pages 33–44, 2012.

[11] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *SOUPS*, 2012.

[12] 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 *KDD*, pages 1276–1284, 2013.

[13] M. Gegick, P. Rotella, and T. Xie. Identifying security bug reports via text mining: An industrial case study. In *MSR*, pages 11–20, May 2010.

[14] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *TRUST*, pages 291–307, 2012.

[15] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. In *MobiSys*, pages 281–294, 2012.

[16] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *WISEC*, pages 101–112, 2012.

[17] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications. In *CCS*, pages 639–652, 2011.

[18] H. Huang, K. Chen, C. Ren, P. Liu, S. Zhu, and D. Wu. Towards discovering and understanding unexpected hazards in tailoring antivirus software for android. In *AsiaCCS*, pages 7–18. ACM, 2015.

[19] D. Kong and H. Jin. Towards permission request prediction on mobile apps via structure feature learning. In *SDM*, pages 604–612, 2015.

[20] D. Kong and G. Yan. Discriminant malware distance learning on structural information for automated malware classification. In *KDD*, pages 1357–1365, 2013.

[21] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *USENIX Security Symposium*, pages 79–94, 1998.

[22] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *UbiComp*, pages 501–510, 2012.

[23] B. Liu. Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca*, 2010.

[24] C. Liu, C. Chen, J. Han, and P. S. Yu. Gplag: Detection of software plagiarism by program dependence graph analysis. In *KDD*, pages 872–881, 2006.

[25] B. Livshits and J. Jung. Automatic mediation of privacy-sensitive resource access in smartphone applications. In *USENIX Security*, pages 113–130. USENIX, 2013.

[26] B. Livshits and T. Zimmermann. Dynamine: Finding common error patterns by mining software revision histories. In *FSE*, ESEC/FSE-13, pages 296–305, New York, NY, USA, 2005. ACM.

[27] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious urls. In *KDD*, pages 1245–1254, 2009.

[28] A. Muralidharan, Z. Gyongyi, and E. H. Chi. Social annotations in web search. In *CHI*, pages 1085–1094, New York, NY, 2012.

[29] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. Wang. Uipicker: User-input privacy identification in mobile applications. In *USENIX Security*, pages 993–1008, 2015.

[30] L. Nelson, C. Held, P. Pirolli, L. Hong, D. Schiano, and E. H. Chi. With a little help from my friends: Examining the impact of social annotations in sensemaking tasks. In *CHI*, pages 1795–1798, New York, NY, USA, 2009. ACM.

[31] M. Neugschwandtner, P. M. Comparetti, G. Jacob, and C. Kruegel. Forecast: Skimming off the malware cream. In *ACSAC*, pages 11–20, 2011.

[32] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *2005 IEEE Security and Privacy (S&P)*, pages 226–241, 2005.

[33] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security*, pages 527–542, 2013.

[34] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language api descriptions. In *ICSE*, pages 815–825, Piscataway, NJ, USA, 2012. IEEE Press.

[35] Z. Qi, F. Long, S. Achour, and M. Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *ISSTA*, pages 24–36, 2015.

[36] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *CCS*, pages 1354–1365, 2014.

[37] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *The Journal of Machine Learning Research*, 11:1297–1322, 2010.

[38] K. Rieck, T. Krueger, and A. Dewald. Cujo: Efficient detection and prevention of drive-by-download attacks. In *ACSAC*, pages 31–39, 2010.

[39] J. Slankas, X. Xiao, L. Williams, and T. Xie. Relation extraction for inferring access control rules from natural language artifacts. In *ACSAC*, pages 366–375, New York, NY, USA, 2014. ACM.

[40] L. Tan, D. Yuan, G. Krishna, and Y. Zhou. /*icomment: Bugs or bad comments?*/. In *SOSP*, pages 145–158, New York, NY, USA, 2007. ACM.

[41] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[42] N. Wang, B. Zhang, B. Liu, and H. Jin. Investigating effects of control and ads awareness on android users' privacy behaviors and perceptions. In *MobileHCI*. ACM, 2015.

[43] R. Wang, W. Enck, D. Reeves, X. Zhang, P. Ning, D. Xu, W. Zhou, and A. M. Azab. Easeandroid: Automatic policy analysis and refinement for security enhanced android via large-scale semi-supervised learning. In *USENIX Security*, Washington, D.C., 2015. USENIX Association.

[44] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR*, pages 4–11, New York, NY, USA, 1996. ACM.

[45] G. Yan, N. Brown, and D. Kong. Exploring discriminatory features for automated malware classification. In *DIMVA*, pages 41–61, 2013.

[46] L. K. Yan and H. Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *USENIX Security Symposium*, 2012.

[47] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In *WISEC*, pages 25–36. ACM, 2014.

[48] M. Zhang, Y. Duan, H. Yin, and Z. Zhao. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. In *CCS*, Scottsdale, AZ, November 2014.

[49] 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 *NDSS*, 2012.