Claim What You Need: A Text-Mining Approach on Android Permission Request Authorization

Mingkui Wei
North Carolina State University
Raleigh, NC, 27606

Xi Gong
Cisco Systems. Inc
Research Triangle Park, NC, 27709

Wenye Wang
North Carolina State University
Raleigh, NC, 27606

***Abstract*—Android is one of the most popular mobile operating systems nowadays, whose popularity, however, also attracts even more crafty developers to develop malicious softwares, or malwares, to exploit illegitimate means for profit. As a basic countermeasure, Android enforces the *permission request scheme*, in which an application (App) is required to present to the user the system resources (permissions) it will access, and ask user's approval before installation. However, this approach has been proven ineffective as it delegates the whole responsibility of decision-making to the user, who usually lacks the professional knowledge to comprehend the interpretation of a permission. Alternatively, many current researches focus on identifying potential malwares based on attributes of individual Apps, such as inspecting their source code, which, unfortunately, fall in another extreme which tend to make the decision for the user. Nevertheless, from the user's perspective, a satisfactory solution should be an approach which *assists users to make the decision of the App installation on their own, by providing them with lucid reasons and requiring minimum professional knowledge*. Based on the observation that the *description* of an App is the most direct interface to communicate its functionality to the user, in this paper we are motivated to explore the relationship between the *description* and the *requested permissions* of an App, and further build a model to predict proper permissions based on its description. Our evaluation with Apps collected from the *Google Play Market* shows that our prediction can achieve as high as 87% accuracy. In this regard, provide a user has full understanding of the description of an App, our model can act as an effective reminder to the user if the App tries to stealthily request permissions that are inconsistent with its description, which is a major character commonly exploited by malwares.**

I. Introduction

Android system is by now one of the most popular operating systems on smartphones and mobile devices, as its open source nature provides great convenience for mobile application development. Meanwhile, the rapid growth of various applications (*Apps*) on Android system also attracts attackers to develop malicious softwares (malwares) and exploit illegitimate benefits. For instance, typical behaviors of a smartphone malware can be probing the user's privacy such as photos or even bank accounts, or sending text message to subscribe services run by the crafty authors for monetary purposes.

As the basic means to thwart such misbehavior, Android enforces a simple permission request strategy, in which system resources, such as photo gallery or SD card content, are mapped into multiple permissions, and before an App is to be installed, it must first present to the user all permissions it is going to request, and ask the user's approval. However, this approach has long been criticized and shown very ineffective in the real world [6]. This is because this approach delegates the responsibility to decide whether to install an App solely to the user, who usually lacks necessary professional knowledge to comprehend the interpretation of a permission [8]. Even for competent users, they tend to overlook this process after being asked the similar questions for multiple times [7].

To find alternative countermeasures without intensively rely on the user's reaction, many researchers put their effort on studying how to identify a potential malware based on an App's individual attributes. The primary directions include inspecting source code to identify potential mobile malwares [5], [12] and studying permission requests patterns of a batch of Apps and matching it to individual App [9], [10]. These approaches, however, fall into another extreme, which tend to make the decision for the user, i.e., by telling the user whether this App is a malware or not. Essentially, these approaches left user even more uninformed, because the reasons that an App is regarded as a malware (e.g., suspicious code or statistically inconsistent) are made even more complicated.

As a matter of fact, from the perspective of human being, i.e., the user, it is difficult to tell if an App is a malware or not without considering the user's intention. For instance, an App could request as many permissions as it wants, as long as the user is aware of and acknowledges this request, it shall not be regarded as malicious to this user. On the other hand, an App can request just one permission and still be think as malicious, if the user does not know or approve it. Therefore, we believe in the regard of App installation and malware identification, the user's interaction is indispensable, and the satisfactory solution should be the approach which neither let users to make the decision alone, nor make the decision for users, but *assist users to make the decision on their own, by providing lucid reasons while requiring as less professional knowledge as possible*.

In this paper, we propose a *text-mining based permission request authorization* scheme for Android applications to solve above problem. Our motivation is based on the key observation that the *description* of an App is the major and the most direct interface to communicate its functionality to the user. Therefore, provide the user fully understands the description of an App, which exposes much less requirement on the user's capability compared with understanding the meaning of a permission, he/she should be aware of the function and

possible resources this App will need. For example, if an App describes itself as an email client, it is easy to know it will need Internet access, while if an App's description reads it is a photo editor, then it will almost surely request to access user's photo gallery. Our proposed scheme mimics this *human intuition* process, which identify the links between the *description* and the *requested permissions* of an App, and alarms the user if it finds inconsistencies, i.e., identifies permissions which are not proper to be requested based on the description, which provides more comprehensible information to assist the user to make a judicious decision.

We collect a data set from *Google Play*, the official and biggest Android application market, and validate our scheme by comparing the actually requested permissions of Apps, with the predicted permissions based on these Apps' description by our scheme. Our results show that, for sensitive permissions such as *send SMS messages*, our prediction accuracy can be as high as 87%, which indicates that our scheme can act as an effective reminder to the user that the to-be-installed App is stealthily requesting excessive permissions, which is one of the major characters commonly find in malwares.

The remainder of this paper is organized as follows. In Sec. II we introduce related work on identifying mobile malwares. In Sec. III we demonstrate our model by providing detailed description and validation. In Sec. IV we evaluate our proposed model with data set obtained from real Android App market, and in Sec. V we conclude our work.

II. Related Work

Recently, many researchers have focused on identifying potential malwares through studying the permission request patterns [4]–[6], [9], [10], [12]. Enck et al. [5] develop a system named *Kirin* which produces meaningful security rules based on security analysis of Android. Then they manually select the blacklisted patterns to represent dangerous sets of permissions. With the help of *Kirin* and their rules, the authors successfully detect applications which implements risky functionality. Their work provides us incentives about the potential risk brought by the combination of dangerous permissions. As an improvement, in our research we aim to leverage machine learning tools to find potentially dangerous permissions automatically.

Frank et al. [9] use a data mining tool of Boolean matrix factorization to find overlapping clusters of permissions for both Android applications and Facebook users. Their goal is to understand the difference of permission requests between high-reputation and low-reputation applications. From their study, they observe a significant difference between these two groups of applications. Moreover, their results indicate that the permission request pattern can be used to evaluate the quality of applications. However, they only consider one attribute, i.e., the *reputation*, of applications to find its potential correlation with permission pattern, which is insufficient considering an App has various attributes. In addition, although reputation is a critical factor, it's not always related to security concerns. For instance, an App with benign purpose but crashes a lot is very likely to get negative reviews.

Peng et al. [10] aim to provide a risk ranking for Android applications, which can help identifying malicious or risky applications more effectively. The proposed risk scoring scheme is based on the permissions an application requests. Using combination of permission requests as the measurement, the authors derive a comparative risk information based on the application communities. If some application is far from the most benign applications (outlier in the distribution) in the market in terms of the defined measure, its risk is probably high. Then, they can generate a rank to help user make decisions whether to proceed the installation. However, the complicated model make users unlikely to understand the reasons behind the rankings.

Another line of work is to study the system and source code. Zhou et al. [12] focus on the repackaged applications and implement an application similarity measurement system called *DroidMOSS* to localize and detect the changes from application-repackaging behavior. Via further manual investigation on these repackaged applications, they are able to identify several malicious behaviors such as re-routing ad revenues, planting back doors or malicious payloads. Their work shows effectiveness by successfully detecting many malicious applications. However, this approach strongly relies on the source code of Apps, which are hard to obtain most of the times. Moreover, comparisons among a large amount of applications have to be conducted, which is time consuming and may not be feasible for the fast release of new applications.

III. System Model

The model of our proposed scheme is shown by the flowchart in Fig. 1, which comprises 3 modules, namely, the *Keyword Extraction* module, the *score calculation and grading rule identification* module and the *Permission Authorization* module. Our model needs a training set before performing permission authorization. The training set contains a set of Apps with description and permissions information. The description is imported to the *keyword extraction* module for keyword selection. The keywords and permissions are then sent to the *score calculation and grading rule identification* module, which analyze the relationship between keyword-permission pairs, and generate a set of rules to "grade" each pair by giving it a value. For an new App, the *score calculation and permission authorization* module will grade each permission according to the identified rules, and based on the score to decide whether to alarm the user that the permission may be requested improperly.

A. Keyword Extraction

Although our goal is to explore the relationship between the description and the permissions of an App, it is difficult, if not impossible, to identify the link directly. This is because although the permissions always follow fixed format and are limited in number, the description is composed with human language, which can not easily be comprehend by computers.

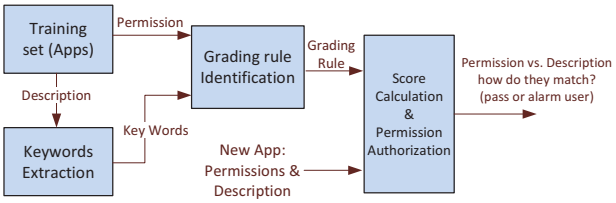


Fig. 1. Description Example

To tackle this problem, we exploit the text-mining approach and extract keywords from the voluminous description.

Keyword extraction is one type of machine learning [11] which has been undergone intensive study in recent years, and there has been many software/applications available. In our study, we tested three candidates: *AlchemyAPI* [1], a natural language processing service provided by AlchemyAPI. Inc; *Yahoo Content Analysis* [3], which is provided by Yahoo! and could detect entities/concepts, categories, and relationships within unstructured content; and *KEA* [2], a keyword/keyphrase extractor which is developed by the University of Waikato. The comparisons are briefly listed below.

1) AlchemyAPI is more tend to pick out keywords which are related to existing objects. It classifies the keywords into categories (e.g., by Company or by Technology), and find words which belong to them. As our aim is to find words which are related to a permission, this classification method cannot provide benefit to our study.
2) Yahoo Content Analysis derives results which are more meaningful for our study purpose. However, because Yahoo Content Analysis does not require training set, its results can not be controlled and oriented to exactly fit our purpose.
3) KEA requires training sets before perform keyword extraction. And because of it, we can take advantage of human wisdom to pick out the most related words from Apps' descriptions, and use them as the training sets. As shown by later evaluations, KEA is able to generate satisfactory results based on very few human-picked text-keyword pairs.

Based on above comparison, we choose KEA as the keyword extraction module in our framework.

B. Grading Rule Identification

The grading rule identification module is the most critical part in our framework, which quantitatively measures the relationship between the keywords and the permissions. This subsection is further divided into 2 parts. In the first part, we take a data set from *Google Play*, analyze the patterns between the keywords and the permissions, and propose the grading rules. And in the second part, we do verification based on the data set and the derived rule.

1) Exploring keywords and permissions:

i). Obtaining the data set. The data set we used in our study is a set of 480 Apps retrieved directly from *Google Play*, under the category "top paid business". Note that our goal is to identify the relationship between keywords and permissions, therefore to confine the samples within one category provides better similarity which facilitate the identification process, and we choose this category over others is because compared with other categories such as "entertainment", the functionality (and thus permissions) of "business" is more limited and the description is more focused, which will facilitate our study to identify relationship between the two. Note that all applications were taken as of May 2013, the content of *Google Play* may have dramatically changed since then.

ii). Keyword extraction. Remind the keyword extraction of KEA is a machine learning process, which needs to be trained with training set. Seemingly onerous, KEA can actually achieve satisfactory result based on very few samples. During our experiment, we generate the training set manually, i.e., we randomly choose an App from the data set, read its description and pick the keywords based on our understanding, also we limit the maximum keywords per App by 10. We gradually increase the training set and run the keyword extraction each time we add a new sample, and find that when the training set is 13, KEA can generate keywords which are well related to the corresponding description. Also note that during our keyword choosing process, we choose the keywords that we think mostly reflect the content of the description, we *do not* compare the description to the permissions and intentionally pick out keywords according to which.

iii). Keyword post-processing. With further inspections of the extracted keywords, we find that there are some keywords share the same meaning and should be combined, which generally fall in 2 scenarios. The first scenario is the same word with different tenses or singular/plural forms. For example, "send", "sends" and "sent" appear as different keywords. For this scenario, we restore all verbs to their base form, e.g., we replace "sends" and "sent" with "send". The second scenario is synonym. For example, "email" and "e-mail", or "short message" and "text message". These words are also combined and replaced by one word. During our study, we do the inspection and replacement manually, this is because we do not have a database which provides us verb tense and synonym information. For large volume process, a database can be built to provide automatic keywords synthesis and replacement.

iv). Keyword-permission relationship exploration. We first do a statistical analysis and rank both the permissions and keywords by their appearance frequency among the 480 Apps. For the 480 Apps, there are totally 88 different permissions are requested, and we identified 2400 different keywords. However, we find the frequency of both, especially the keywords decays very quickly, i.e., among the 2400 keywords, there are 1871 keywords (78%) only appear once and 2101 keywords (88%) appear less or equal to two times.

In Fig. 2 we show the top list of the ranking of both permissions and keywords. From the result we can observe that although the functionality and purpose of the 480 applications vary, there exists a statistical trend for the keywords and the permissions. More interestingly, this trend follows our common sense. For instance, in Fig. 2(b), we see that *email*

Ranking	Frequency	Permission name
1	349	full network access
2	300	modify or delete the contents of your USB storage/SD card
3	294	test access to protected storage test access to protected storage
4	205	view network connections
5	130	read phone status and identity
6	83	control vibration
7	81	read your contacts
8	79	prevent tablet from sleeping prevent phone from sleeping
9	72	view Wi-Fi connections
10	72	read call log
11	57	run at startup
12	55	take pictures and videos
13	53	find accounts on the device
14	51	precise location (GPS and network-based)
15	41	approximate location (network-based)

(a) Permission ranking.

Ranking	Frequency	Keyword	Ranking	Frequency	Keyword
1	110	email	16	17	speak
2	94	business	17	16	learn
3	74	file	18	15	share
4	66	contact	19	15	sd card
5	47	send	20	14	tasks
6	39	text	21	14	outlook
7	24	office	22	14	audio
8	24	calendar	23	13	call
9	23	sms	24	13	sales
10	23	download	25	13	sync
11	22	app	26	13	tool
12	21	notes	27	13	scan
13	19	location	28	12	it
14	18	calculator	29	12	video
15	17	phrases	30	12	dropbox

(b) Keyword ranking.

Fig. 2. Ranking of permissions and keywords of 480 applications.

ranks as the most frequently appeared keyword among these Apps, which conforms our intuition that business-related Apps are very likely to access email. Other than *email*, we also see other keywords which fit our understanding about the scope of "business", such as *business, contact*. For the permission requests shown by Fig. 2(a), we also see the same phenomenon. Except the first few permissions such as *full network access* which are almost "default" permissions requested by any type of Apps, we see *control vibration* and *read your contacts* rank on the top of the list, which are reasonable as a business App may need to vibrate the phone to remind the user, and access contacts to keep the user linked with other business partners.

The observation from Fig. 2 naturally leads to these further questions: *what is the keyword frequency pattern for a certain permission? Does it follow the same trend?* To this end, we further analyze the dataset in the following manner. We first select one permission from the list, and retrieve all Apps which have requested this permission. Then, we gather all the keywords of the set of Apps instead of the 480 Apps and re-rank the keywords. In Fig. 3 we present one of the results for the permission *read your contacts*, a popular permission which is requested by 81 Apps.

Ranking	Frequency	Keyword	Ranking	Frequency	Keyword
1	38	contact	16	6	caller
2	27	email	17	5	photos
3	19	text	18	5	outlook
4	19	business	19	5	sync
5	18	sms	20	5	secure
6	18	file	21	4	card
7	17	send	22	4	events
8	10	notes	23	4	usb
9	10	call	24	4	scan
10	9	calendar	25	4	invoice
11	8	location	26	4	gps
12	7	download	27	4	battery
13	7	tasks	28	4	box
14	6	office	29	4	sending
15	6	exchange	30	3	text message

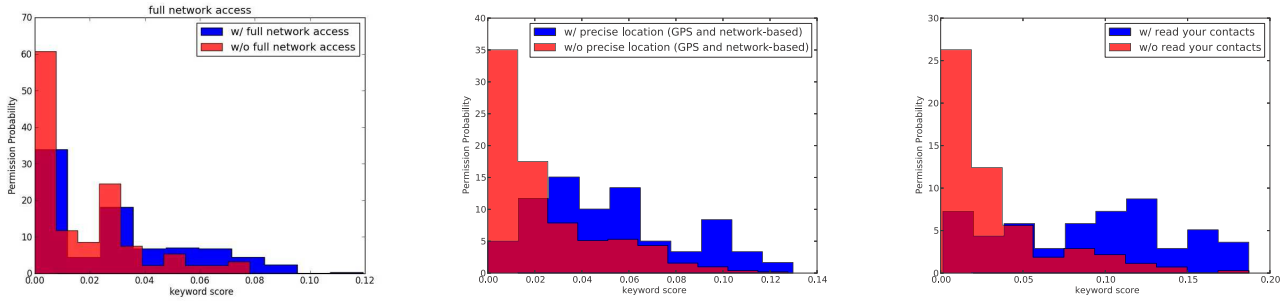
Fig. 3. Keywords frequency for *read your contacts*.

Compare Fig. 3 with Fig. 2(b), the difference can be easily identified. For example, for the 480 Apps, keyword *email* ranks the first and appears 110 times, while for the 81 Apps requested *read your contacts*, the most frequently appeared keyword is *contact*, and *email* has been degraded to the second place. Reversely, the keyword *contact* only ranks at the 4th position in Fig. 2(b). Beside the keyword ranking position change, which is obvious, we also observe the change on their relative frequencies. For instance, the ratio of the frequency of *email* and *contact* in Fig. 2(b) is $110/66 = 1.67$, which has been dramatically reduced to $27/38 = 0.71$ as shown in Fig. 3. Based on these observations, we can make the conclusion that there exists a relationship between permissions and keywords, and therefore it is possible to predict the permissions an App is going to request based on the keywords extracted from its descriptions. For instance, if an application contains the keywords *contact, email, text, business* and *sms*, it is highly likely that this application will request the permission *read your contacts*, or in other words, it is legitimate for this application to request this permission, because *it reveals this implication to the user*. On the contrary, if an application contains nothing but *secure*, then the probability or the legality of this application to have *read your contact* is low, and if this permission is ever requested, the App may probability abuse this permission, and the user should be alarmed.

Inspired by this observation, we propose to grade a permission based on keywords as follows. For each permission, we first collect all Apps which requested it. We rank the keywords of this set of Apps according to their frequency, such as shown in Fig. 3. We then assign a weight to each keyword as the ratio of its frequency to all keywords, i.e., a value with the keyword's frequency as the numerator, and the summation of frequency of all keywords as the denominator. Then for a new-coming App which requests this particular permission, we extract the keyword from its description, and then grade it by adding the weight of all keywords that appeared in its description. This approach, although straightforward, can effectively differentiate and predict the permissions requested by an App based on its description, shown by the verification provided in the following part.

2) Verification of the Grading Rule: To verify the validity of the grading rules, we conducted the following experiment.

1) Choose a permission
2) Extract keywords from the Apps which requested this



(a) Full Network Access (b) Precise Location (c) Read Your Contacts

Fig. 4. Score distribution for Apps with and without a certain permission.

permission, and rank them according to their frequency.

3) Assign weight to each keywords identified in step 2.
4) For each of the 480 Apps, calculate its score based on the weight of the keywords in step 3, no matter whether it requested this permission or not.
5) Separate the 480 Apps into two control groups based on whether they have requested this permission, and identify the distribution of the score for both groups.

The reasoning behind this experiment is as follows. If a permission does not have any relationship to certain keywords, i.e., the frequency of these keywords does not depend on whether this permission is requested, then for Apps which requested this permission and for those who did not, the score distribution should be similar. However, the results of this experiment shows clear distinctions between the two control groups, which veto this assumption and indicates the existence of such relationship. In Fig. 4 we provide the results for 3 permissions rank at different place shown in Fig. 2(a).

We first look at Fig. 4(a), which displays the score distribution for Apps with or without the permission *full network access*. From this figure we see that the distinction between the two group is not significant. Although the Apps with the permission tend to have higher score, the trend of the two group are similar. This figure actually shows a weak link between the permission *full network access* and the keywords, and the reason is because *full network access* is almost an "default" permission, meaning the network access is too generic a function for most Apps to be explicitly expressed in their description.

We then look at Fig. 4(b), which shows the score distribution for *precise location*. In this figure, the distinction between the two group begin to appear. In particular, for the Apps without this permission, the score follows a normal-like distribution with zero mean, and about 65% of the Apps have score less than 0.02. However, for those with this permission, the majority of them clustered between 0.02 and 0.06. The reason of this distribution is because *precise location* is not as popular as *full network access*, therefore Apps who requested it are likely to have GPS location functions and advertise them in their descriptions. However, because GPS function is not an imperative necessity for business Apps, this permission only ranks 14 in the permission list, and the distinction between the two groups are still not very significant.

At last we look at Fig. 4(c), which shows the score distribution for *read your contracts*, one of the mostly requested permissions and is also tightly related to the "business" category. In this figure we are able to observe a clear distinction between the two groups. The score distribution of the Apps without the permission almost keeps unchanged from previous two figures, but the distribution of those with this permission gathers at a much higher value and clearly distinguish themselves.

All three figures endorsed our grading rule. The first figure show a counterexample that if the link between a permission and keywords are weak, then the scores of Apps with and without this permission are tend to be similar. While the last 2 figures clearly shows that the more relevant the permission and the keyword, the more significant the distinction between the two groups.

C. Score Calculation and Permission Authorization

This module makes the final judgment on whether it is proper for an application to request a certain permission, and the flexible part lies in the setting of the threshold, i.e., the score beyond which the App will be regarded as benign.

Intuitively, we can set the threshold wherever the two group intersect (which are chosen as the threshold in our evaluation in Sec. IV), for instance, 0.05 shown in Fig. 4(c). Particularly, if an App requests *read your contact* but gets a score less than 0.05, we think this permission request is improper and notify the user. But this threshold can also be adjusted to fit different purpose. For instance, a value higher than 0.05 indicates a more strict policy, which indicates a benign App are more likely to be regarded as a potential malware.

IV. Evaluation

A. Metrics

To evaluate our scheme, we define two metrics, *Prediction Confidence* and *False Positive Rate*, to depict the difference between the permissions actually requested by Apps and predicted by our model. *Prediction Confidence* aims to compute the rate our prediction is consistent with the actual permission request. We only measures the *false positive rate* is because

Permission Name	Confidence	Dev of Conf	False Positive	Dev of FP
send SMS messages	0.8708334	0.055381922	0.09583307	0.039236188
read your contacts	0.7937467	0.033374069	0.1583337	0.038367978
directly call phone numbers	0.7645767	0.048662075	0.19791667	0.049695528
precise location	0.6687466	0.064632198	0.2708304	0.061627182
full network access	0.4965796	0.055806821	0.0854135	0.011675425
modify USB/SD storage	0.6083297	0.06458	0.022403112	0.012542182

TABLE I
EVALUATION RESULTS

for malware detection, we concern more on whether our model can help user to identify potential malwares. Therefore, the false positive error, i.e., we allow a permission but it is not requested by the App, bears higher risk.

Definition 1: Given an application A_i and a permission P_j, let $p_{i,j} \in \{0,1\}$ and $\hat{p}_{i,j} \in \{0,1\}$ be indicators which respectively denote the *actual* and *predicted* result of whether A_i requested P_j, where 1 means A_i requested P_j, and 0 means otherwise. For a permission P_j and a set of applications $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$, we can derive the set of permission requests $\mathcal{P} = \{p_{1,j}, p_{2,j}, \ldots, p_{n,j}\}$ and the set of *predicted* permission requests $\hat{\mathcal{P}} = \{\hat{p}_{1,j}, \hat{p}_{2,j}, \ldots, \hat{p}_{n,j}\}$, define the *prediction confidence* as:

$$\frac{|\{(p_{i.j}, \hat{p}_{i.j}) | p_{i,j} = \hat{p}_{i,j}, p_{i.j} \in \mathcal{P}, \hat{p}_{i,j} \in \hat{\mathcal{P}}\}|}{|\mathcal{A}|} \quad (1)$$

Definition 2: Provided the same denotation as in Definition 1, define the *false positive rate* as:

$$\frac{|\{(p_{i,j}, \hat{p}_{i,j}) | p_{i,j} = 0, \hat{p}_{i,j} = 1, p_{i,j} \in \mathcal{P}, \hat{p}_{i,j} \in \mathcal{P}'\}|}{|\mathcal{A}|} \quad (2)$$

B. Evaluation Setup and Summary

We choose six permissions as the subjects in our evaluations. Among the chosen permissions, four are considered the "sensitive" ones, which are *send SMS messages*, *read your contacts*, *precise location* and *directly call phone numbers*, besides which, we also select two widely requested permissions, i.e., *full network access* and *modify or delete the contents of your USB storage modify or delete the contents of your SD card*, such that to allow us to observe the effectiveness of our scheme on different permissions.

For each experiment, we randomly choose 432 out of the 480 Apps as the training set to identify the relationship pattern between keywords and permissions, and apply the score assignment on the other 48 applications. To get a more stable result, we further introduce a *cross validation* mechanism, in which we run 10 trials and in each trial the training set and test set are randomly chosen. The results of this experiment are shown in Table I.

From Tab. I we can find that for the permissions which are highly related to the category, our predictions can achieve high accuracy. Meanwhile, we also observe that their deviations are very small, which guarantees a steady prediction. Similar as being found in Sec. III, lower accuracy is found on less relevant permissions. For *full network access*, the confidence is close to 50%, which basically provides zero information.

V. CONCLUSION

In this paper, we conduct an empirical study on identification of improper permission request of Android Apps by exploring the relationship between their two important attributes, i.e., the *description* and the *permission*. Based on text-mining and machine learning approach, we design a score-based classification model to investigate the suitability of permissions authorization of an App based on its description, and evaluate the effectiveness with Apps obtained from *Google Play*. As one of the efforts in studying the *permission* issue of mobile applications, our approach brings an innovative perspective to handle this problem. Our contributions not only limit to the proposed classification model, but also lie in as the first to take semantic analysis on literal attributes of Android Apps and establish relationship to their permissions. In the future, we will work on enhancing the performance of our model in terms of improving the effectiveness of keyword extraction, designing more comprehensive grading mechanism and testing with larger and more diversified data sets.

REFERENCES

[1] Alchemyapi. http://www.alchemyapi.com/.
[2] Kea. http://www.nzdl.org/Kea/index.html.
[3] Yahoo content analysis. http://developer.yahoo.com/contentanalysis/.
[4] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 73–84. ACM, 2010.
[5] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009.
[6] Z. Fang, W. Han, and Y. Li. Permission based android security: Issues and countermeasures. *computers & security*, 43:205–218, 2014.
[7] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM, 2012.
[8] M. Frank, B. Dong, A. P. Felt, and D. Song. Mining permission request patterns from android and facebook applications. In *ICDM*, pages 870–875, 2012.
[9] M. Frank, B. Dong, A. Porter Felt, and D. Song. Mining permission request patterns from android and facebook applications. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 870–875. IEEE, 2012.
[10] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM, 2012.
[11] C. Wu, M. Marchese, J. Jiang, A. Ivanyukovich, and Y. Liang. Machine learning-based keywords extraction for scientific literature. *J. UCS*, 13(10):1471–1483, 2007.
[12] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326. ACM, 2012.