APP Vetting Based on the Consistency of Description and APK

Weili Han[1,2,3(✉)], Wei Wang[1], Xinyi Zhang [1], Weiwei Peng[1], and Zheran Fang[1]

[1] Software School, Fudan University, Shanghai, China
[2] Key Lab of Information Network Security,
Ministry of Public Security, Shanghai, China
[3] Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
wlhan@fudan.edu.cn

Abstract. Android has witnessed a substantial growth over the years, in the market share as well as in the number of malwares. In this paper, we proposed a novel approach to detect potentially malicious applications, based on the semantic relatedness between the applications' descriptions and the apk files. We gathered an application database of 7,570 valid applications for training and testing, finding that about 16.6 % of the tested applications exhibit a lack of relatedness between the apk files and descriptions, due to either inadequate embedded text in apk file, too short a description, unsuited description, or being a malicious application. In additions, there are 4 % of applications unjustly deemed as unrelated. Our study showed that the semantic based approach is applicable in terms of malware detection and in judging the soundness of descriptions.

Keywords: Android security · Malware · NLP · APK · Description

1 Introduction

Recent years, smart phones and mobile devices have become more and more popular. A recent report from International Data Corporation [5] showed that the number of Android smart phones reached 81.1 % in the first quarter of 2013. And the number of Android applications is rapidly growing. According to a report from AppBrain [1], the official Android Market held a total of 1,316,773 applications by July 30, 2014.

Security on Android has become a hot topic over the years, with a growing number of malicious applications threatening the privacy and financial security of users. The automated detection of malicious applications was put forward by Google in the form of Google Bouncer, which, according to RiskIQ [2], was able to detect 60 % of the malicious applications in the official android market, Google Play. However, the detection rate has drastically decreased with time, by the year 2013, it was able to detect only 23 % of the malicious applications. This shows a serious need for new methods of malware detection.

Such security problem is due to Android's open platform and unrestricted application market, using which any developer, professional, amateur or even

M. Yung et al. (Eds.): INTRUST 2014, LNCS 9473, pp. 259–277, 2015.
DOI: 10.1007/978-3-319-27998-5_17

malicious, are able to develop and sell their work to the world [7]. Such a low barrier attracts many developers unskilled in the English language, flooding the market with poorly described application, making it possible for malicious applications to hide among them.

In light of the lacking in malware detection methods and descriptions writing aids with the android market, we intend to develop a tool, being the first to utilize the relatedness between the application descriptions and the embedded text in apk files to achieve malicious application detection and to discover poorly written descriptions.

Following this innovative approach we designed a framework to analyze the relatedness between an application's description and the embedded text in the apk file. We are able to achieve a recall of 91.2 % in the most tolerant case. We further analyzed the applications with its descriptions and apk file deemed unrelated, find that 77 % of them falls into one of the following categories, (1) Inadequate Embedded Text, (2) Short Description, (3) Unsuited Description, (4) Malicious Application.

The rest of the paper is organized as follows. In Sect. 2, we introduce the background of the Android System and some tools used in our work, NLTK and ESA. Section 3 defines the problem and the objective of our system, while Sect. 4 outlines the framework of our system, the acquisition of data and rationalize some of the design choices. Section 5 evaluates the system and analyzed the evaluation results. We then discuss the strength and weakness of the system in Sect. 6, the related works in Sect. 7. At the end, we conclude the paper in Sect. 8.

2 Background Knowledge

2.1 Android System

Android applications are mainly written in Java, with some configurations and resources defined in XML format. Developers need to register all the Activities the application use in the Manifest file. Activity in Android is an application component that provides a screen with which users can interact in order to perform specific operations, generally, the appearance of an Activity is determined by a layout XML file (e.g. main.xml), where a hierarchy of viewable widgets is defined. Some of the texts in an Activity are statically determined, like the text on a button or some description words, these texts can be defined in three ways: (1) developer declares a string resource in string.xml, and includes the string resource in the view's text field within layout XML file, (2) developer directly assigns a string value to the text field of a view within layout XML file, (3) developer assigns a constant string to the text fields of a view's object in Java code.

By coding convention, text to be displayed to users are written mostly through the first and the second way. Thus, we can extract the displayed string from the string.xml file and layout XML file.

2.2 Text Sanitization

Natural Language Toolkit (NLTK) is a leading platform providing advanced language processing tools which are used in our system.

RegexpTokenizer Used to break sentences into words.
tag.pos_tag Used to tag words by part of speech.
RegexpParser Based on a grammar, given by regular expression, used to parse a sequence of words into a tree structure.
PorterStemmer Proposed by Porter in 1980, Porter Stemming algorithm [14] is used to stem words, by removing word suffix, uniforming tense, converting plurals and singular noun.
WordNetLemmatizer This stemmer utilize the `morphy` function of WordNet [8], and preform no action if the word is not recognized by WordNet.

2.3 Explicit Relatedness Analysis (ESA)

ESA [9] is used to calculate the relatedness between text. The main idea behind ESA is to construct a multidimensional semantic space based on a given set of literature, and then put the vector to be analyzed into the semantic space, turning the semantic relatedness between two pieces on text into the cosine value of the angle between two vectors projected into the semantic space.

It is called explicit semantic analysis because each dimension in the semantic space is an explicit article in the literature. While the projection into the semantic space is based on the overlapping of words between the text.

The reason we choose ESA as the tool of semantic relatedness analysis is because that ESA has its own literature as the building blocks, greatly decreasing the size of the data needed for training. In comparison to synonym based tools like WordNet, ESA is more focused on the relatedness on a broader scale and provides a quantitative representation of the semantic relatedness.

We use `esalib` [13], the best maintained open-source ESA library available, which uses 2005 wikipedia as the literature set.

3 Problem Definition

We define the semantic vector for the embedded text of application *app* as below:

$$T_{app} = [v_1, v_2, v_3, ..., v_n]^T$$

$$\text{where } v_i = \begin{cases} 1 & \text{if } w_i \in MT[app] \\ 0 & \text{otherwise} \end{cases}$$

Where $MT[app]$ is the list of semantically significant phrases in the embedded text of *app*.

Similarly, semantic vector of *app*'s description is defined as:

$$D_{app} = [v_1, v_2, v_3, ..., v_n]^T$$

$$\text{where } v_i = \begin{cases} 1 & \text{if } w_i \in MD[app] \\ 0 & \text{otherwise} \end{cases}$$

Where $MD[app]$ is the list of semantically significant phrases in the description of *app*.

Our objective is to find a relatedness function $\hat{rel}$ approximating the actual relatedness between the description and embedded text as closely as possible.

$$\hat{rel}(T, D) = \begin{cases} 1 & \text{if } T \text{ is deemed related to } D \\ 0 & \text{otherwise} \end{cases}$$

We then need to introduce a few assumptions.

First, we suppose there is an ideal function rel.

$$rel(T, D) = \begin{cases} 1 & \text{if } T \text{ is actually related to } D \\ 0 & \text{otherwise} \end{cases}$$

Second, suppose that for most applications, its description and embedded text is related.

$$\sum_{app \in App} rel(T_{app}, D_{app}) \approx |App|$$

App here means the set of all applications in the dataset.

Third, there exist some similar applications, among which these descriptions and embedded text points to similar functionalities and thus relate to each other. Given an application, the number of applications similar to it grows linearly with the size of the dataset.

More specifically, given an X uniformly distributed on the set *App*, there exist α and β, such that the following equation is satisfied.

$$E(\sum_{app \in App} rel(T_X, D_{app}) + \sum_{app \in App} rel(T_{app}, D_X)) \approx \alpha \cdot |App| + \beta$$

The expectation is calculated as:

$$\sum_{X \in App} \sum_{app \in App} rel(T_X, D_{app}) + \sum_{X \in App} \sum_{app \in App} rel(T_{app}, D_X)$$

$$\approx \alpha \cdot |App|^2 + \beta \cdot |App|$$

$$2 \cdot \sum_{app_1 \in App} \sum_{app_2 \in App} rel(T_{app_1}, D_{app_2}) \approx \alpha \cdot |App|^2 + \beta \cdot |App|$$

$$\sum_{app_1 \in App} \sum_{app_2 \in App} rel(T_{app_1}, D_{app_2}) \approx \alpha' \cdot |App|^2 + \beta' \cdot |App|$$

We then introduce two measurements, precision and recall, to examine the effectiveness of our approximation $\hat{rel}$. To avoid the introduction of subjective error, we take the fact of whether the texts are from the same application as ground truth, which is to say, we deem a piece of embedded text and a description as matched if and only if they are from the same application. Hence, precision and recall should be in forms as below.

$$precision = \frac{\sum_{app \in App} \hat{rel}(T_{app}, D_{app})}{\sum_{app_1 \in App} \sum_{app_2 \in App} \hat{rel}(T_{app_1}, D_{app_2})}$$

$$recall = \frac{\sum_{app \in App} \hat{rel}(T_{app}, D_{app})}{|App|}$$

Recall stands for the portion of applications that have their own embedded text and description deem related, which, according to assumption two, should be approximate to 1. On the other hand, precision stands for the portion of embedded text and description pairs that actually come from the same application, among all that are deemed related, which, according to assumption number three, is approximately $\frac{1}{\alpha' \cdot |App| + \beta'}$.

In order to avoid the presence of $|App|$ in *precision*'s ideal value, we define *precision*$'$ as a replacement of *precision*. We define *precision*$'$ as the expected *precision* when each piece of embedded text is compared with two descriptions, one from the applications itself, and another randomly chosen among all test samples, which gives,

$$\begin{aligned} precision' &= \frac{\sum_{app \in App} \hat{rel}(T_{app}, D_{app})}{\sum_{app \in App} (E_X(\hat{rel}(T_{app}, D_X)) + \hat{rel}(T_{app}, D_{app}))} \\ &\approx \frac{|App|}{\alpha' \cdot |App| + \beta' + |App|} \\ &\approx \frac{1}{\alpha' + 1} \qquad \text{when } |App| \text{ is large} \end{aligned}$$

α' is dependent on the real life condition. Our goal is to let *recall* be close to 1, which *precision*$'$ is close to $\frac{1}{\alpha'+1}$; thus the objective function is defined as,

$$F_1 = \begin{cases} 2 \cdot \frac{precision' \cdot recall}{precision' + recall} & \text{if } precision' \leq \frac{1}{1+\alpha'} \\ 2 \cdot \frac{\frac{1}{1+\alpha'} \cdot recall}{\frac{1}{1+\alpha'} + recall} & \text{otherwise} \end{cases}$$

And the problem becomes to find a $\hat{rel}$ to make F_1 as high as possible.

For example, suppose the total number of applications is 1,000, with a recall of 90 % and a precision of 10 %, then the adjusted F_1 score should be 94 % when α' is less than 1 %.

This applies when in the test set, each application has its embedded text matched against all 1,0000 descriptions. However, we now want to reduce the workload by randomly selecting 49 descriptions, in addition the one description from the applications, to matched against the embedded text. Suppose in this case, the recall is still 90 % which the precision become 12 %, the adjusted F_1 score should be 88 % with *precision*$'$ as 86 %.

4 System Design

4.1 Framework

Our system takes in an application and a description, after a serious of processing and decision making, decides whether these two are related semantically.

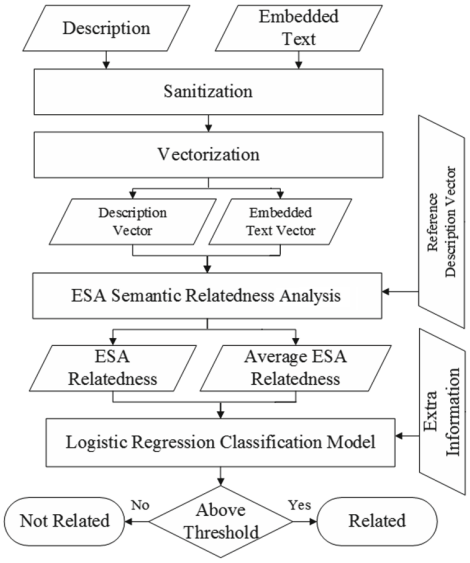

Fig. 1. System implementation outline

As shown in Fig. 1, the system is designed into the following modules.

Sanitization. In sanitization, we remove filter and clean the texts to extract strings with semantic relevance. Using the tools of `NLTK`, we split the sentences according to regular expression, tag words with their part of speech, combine words into phrases, stem words and remove stopwords, obtain a list of phrases in the ends.

Vectorization. Using 0 and 1 to represent the existence of a word in semantic vector. The involved vectors includes,

- v_1 The vector representing the given description
- v_2 The vector representing the embedded text in the given application
- V_3 The reference semantic vectors generated before hand from 50 randomly chosen descriptions.

ESA Semantic Relatedness Analysis. Using v_1 and v_2 generated in Vectorization, ESA relatedness are calculated. Average ESA Relatedness are the average of the relatedness between v_2 and the 50 vectors in V_3.

Logistic Regression Classification. Using the classification function trained before hand, applying which to input vector given by the current ESA relatedness, average ESA relatedness, description, and embedded text, we obtain a result in the range of 0 to 1. Based on the threshold, the system then gives a conclusion on whether the two pieces of text are deemed related.

4.2 Data Setup

Obtaining Data. To obtain the description information of Android applications, we use an open-source third party script, android-market-api [3]. Since Google Play strictly restricts the number of requests to 500 applications per user, denying to reply useful information when the limitation is exceeded, we have to use multiple google account to accomplish the crawling of application description as well as apk file. Even then, we are only able to obtain about 10,000 applications' information. To extract semantic information from the apk file, we use the tool `apktool` [10] provided by Google to obtain string file `strings.xml`, and layout files. Since by coding convention, the text displayed in user interface would be stored in these files, we choose these as the source of embedded text information.

Data Sanitization. After gathering the application descriptions, we then sanitize them to extract key phrases for further processing.

First, we substitute the non English characters to their closest English counterparts, such as converting Lo$\ddot{w}$is to Lowis.

Second, we customize the `RegexpTokenizer` to deal with cases of abbreviation (e.g. U.S.), combined-words and percentage, currency, and numbers (e.g. 10.2 %). Then we split the sentence according to punctuation and spaces. The specific regular expressions used are as listed.

Abbreviation ([A-Z])(\.[A-Z])+\.?
Combined-words \w+(-\w+)*
Percentage, currency and numbers \$?\d+(\.\d+)?%?
Ellipse \.\.\.

Third, we use the tool `tag.pos_tag` to perform part of speech tagging.

Fourth, we use `RegexpParser` to extract phrases, focusing mainly on noun phrases, including single noun, noun+noun, adjective+noun which make up 84.8 % (need change) of the noun phrases in our dataset.

In addition, we decide to keep all the verbs that is identified. These two types of words are what we believe contains most semantic meaning.

Noun Phrases <NN.*|JJ>*<NN.*>
Verb <VB.*>

At last we stem all the extracted words, remove stopwords, duplicates or words that are too long or too short.

Embedded Text. In dealing with embedded text, we need to first remove spacing characters such as "$\backslash n$" and "$\backslash t$", line numbers and html tags. Texts embedded in apk file can be classified into two type, phrases and sentences. Phrases are fragments of words lacking a full grammatical structure, usually used as texts on buttons, or options; while sentences are used in introduction of the application, copyright information, documentations, etc. In this case, we roughly assume that word sequences consisting of more than four words are sentences.

Sanitization are performed to these sentences in ways similar to those in descriptions.

Data Model. The semantic information we processed in Sect. 4.2 is then stored in `json` file, with a mapping relationship as below.

$$MD[app] = [w_{i_1}, w_{i_2}, ..., w_{i_n}]$$

In the definition, *app* means the name of the given application, n means the total number of phrases in the application description. While W is the list of all phrases extracted from descriptions, with w_i meaning the ith words in W. MD stores the phrases mapping of descriptions.

Embedded text are stored in similar ways.

$$MT[app] = [w_{i_1}, w_{i_2}, ..., w_{i_n}]$$

In the case, MT stores the mapping of embedded text.

4.3 Classification Model

Since the $\hat{rel}$ function to be generated is a classification function, we use logistic regression, one of the most popular forms of binary regression [15], to derive the model.

Preliminary Analysis. We give an preliminary analysis of the dataset, which consists of 7,570 applications.

First, we analyze the difference in relatedness across various categories. According to Fig. 2, there are actually observable difference in description text relatedness across categories.

Applications in Categories like "Media and Video", "Personalization", "Productivity" and "Tools" have a higher relatedness on average. The number of applications in "Media and Video" is rather low, taking up on 52 among the total of 7,570 applications. Typical applications in this category include `hdplayer`, `videoeditor`, and `utorrent`, all require the technique of video encoding or fast decoding, some even need a high bandwidth to provide video content to the client. All these functionalities are demanding on the technical side, usually supported by well developed software company or website, with considerable number

of staffs and users. It is quite natural for these kind of applications to have well written descriptions.

"Personalization" includes mobile theme, mobile font management applications, lock screens, etc. Descriptions and embedded texts in this category tend to be short, functionalities tend to be focused; thus the descriptions and texts do not diverge due to the simplicity.

"Productivity" and "Tools" are mostly used to provide technical service (such as file management and anti-virus), which are also mostly provided by established companies.

On the other hand, categories like "Books &References", "Education" and "Libraries &Demo" seems to have a lower relatedness on average. With "Libraries &Demo", applications in this categories are under development, the lack of relatedness seems quite natural. For "Education" and "Book &References", these two categories are mostly intended as a learning aid. Some are originally intended to meet the developer's own need, without a marketing team to write the description.

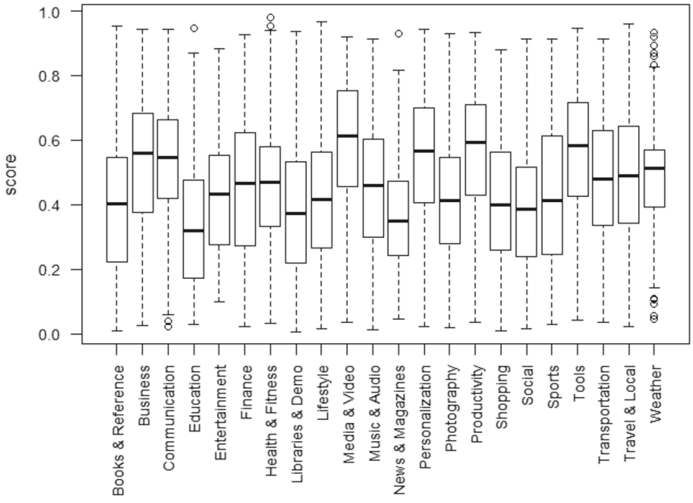

Fig. 2. ESA relatedness distribution across categories

From Fig. 3, we can reenforce our claim that "Media and Video", "Productivity" and "Tools" are mostly developed by established teams since they all have a longer description length on average; while "Libraries &Demo" have a very small average description length.

We then go on to analyze whether the relatedness between an application's embedded text and its own description is actually higher than that between others. As the reference point, we randomly selected 50 descriptions; for a given embedded text, we calculated the average relatedness between it and the selected descriptions.

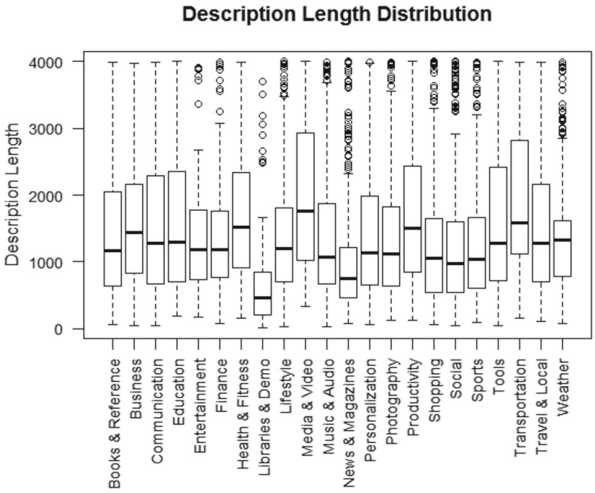

Fig. 3. Description length distribution across categories

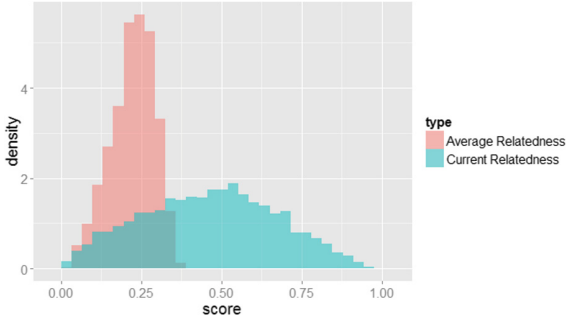

Fig. 4. Relatedness and average relatedness distribution

From Fig. 4, we can observe that texts from the same applications generally have a higher relatedness than the average. However, there is no distinct boundary between these two, which is to say, we will not able to find a threshold that could perfectly tell apart whether the texts are from one application. Say, we set the threshold to 0.2, 87.7 % of the applications can have its own embedded text and description correctly related; while 66.1 % of the average relatedness would be beyond the threshold, giving a `precision'` of 56.9 %.

A sampled average of relatedness is compared to the relatedness from same applications. As shown in Fig. 5, the dots in the lower triangle stands for applications whose embedded text has a relatedness to its own description lower than the average of sample texts, taking up 11.5 % of all applications. This means that using average relatedness as aid to classification can improve the recall over threshold-

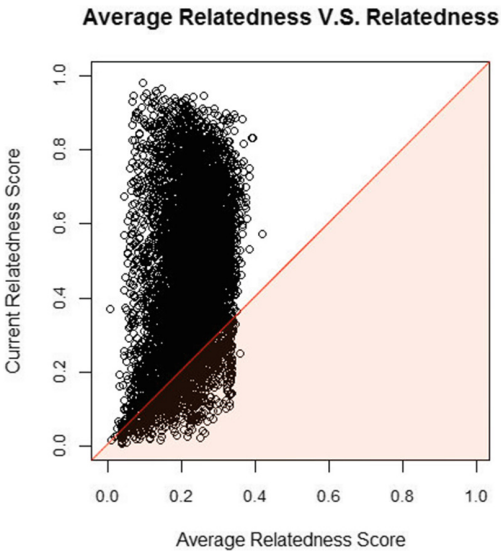

Fig. 5. Relatedness against average relatedness

based classification; however, there is still 46.6 % of the average relatedness above that of the applications own relatedness, giving a `precision'` of 65.0 %.

Beyond the usage of relatedness and average relatedness we develop a more detailed model to perform the classification.

Logistic Regression. First we examine the factors that would possibly reflect the actual relatedness.

score the ESA relatedness value between the embedded text and description to be classified
avgScore the average ESA relatedness between the given embedded text and 50 randomly chosen descriptions
descLen the number of phrases in the given description
textLen the number of phrases in the given embedded text

The effect of average relatedness had been discussed in Sect. 4.3. Considering this `avgScore` is compared against `score`, we use `reScore` which is $\frac{score}{avgScore}$ to replace it.

In addition, in `descLen` and `textLen` affect the dimensions of the semantic vector they are kept as influencing factors.

Running Logistic Regression on the factors listed gives result are shown in Table 1.

Using the model, and ground truth classifications we can then project the four factors on to the region (0, 1), making the projected value for texts from the same application close to 1 while that for texts from different applications

Table 1. Weights in logistic regression

	Estimate	Standard Err	Z value	p value
Intersect	-7.625e+00	4.214e-02	-180.948	< 2e-16
score	6.860e+00	1.427e-01	48.077	< 2e-16
reScore	1.803e+00	3.301e-02	54.628	< 2e-16
descLen	-5.570e-04	1.653e-05	-33.699	< 2e-16
textLen	6.077e-07	1.965e-07	3.092	0.00199

close to 0. As shown in Fig. 6, most of the projected value for not matched pairs are hold in a small region.

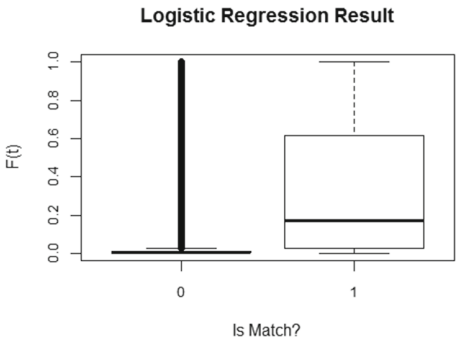

Fig. 6. Logistic regression classification

As was described in Sect. 3, given α' we want to find the threshold, such that F_1 is the highest.

In Fig. 7, we plot *recall* against *precision*$'$, depending on α' different threshold is selected, detailed values are listed in Table 2.

5 Evaluation

5.1 Prototype Implementation

All codes in `python` are written and run in `Python2.7`, with `Windows 8` Operation System, 4-Gigabytes of memory. Functionalities implemented through python includes text sanitization and storage, handling of length information, determining threshold, and part of the graphs. ESA relatedness calculation is implemented in `Java`, with `OpenJDK 7u51` on Operating System `Ubuntu 13.10`, 16-Gigabytes of memory.

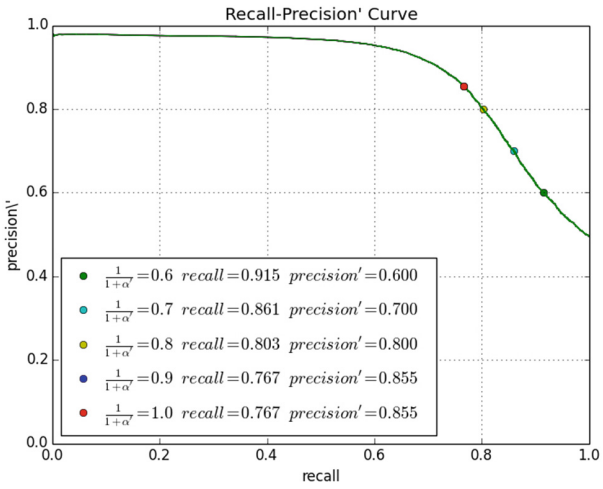

Fig. 7. *Recall* against *Precision*′

5.2 Experiment Setup

There are 7,570 valid applications obtained and used in our work, whose apk files and descriptions are crawled from Google's official Android market, Google Play. Although the default language is English, due to some intentional of unintentional error, some of the applications' descriptions are not in English, which are discarded.

In order to evaluate our system, we divide the applications into two sperate parts, performing as the training set and the test set. The training set consists of 347,428 applications while the test set consists of 38,602 applications.

Based on different choice of α' the measurements are calculated, as is shown in Table 3.

Table 2. Threshold and corresponding F_1 score

$\frac{1}{1+\alpha'}$	Threshold	Recall	*Precision*′	F_1
0.6	0.0050	0.915	0.600	0.725
0.7	0.0090	0.861	0.700	0.772
0.8	0.0143	0.803	0.800	0.801
0.9	0.0203	0.767	0.855	0.809
1.0	0.0203	0.767	0.855	0.809

We compare our method, based on Logistic Regression (annotated as LR), with the straight forward threshold on ESA relatedness approach (annotated as TH). As shown in Table 4 and illustrated in Fig. 8, logistic regression has

Table 3. Evaluation results

$\frac{1}{1+\alpha'}$	TP	FP	FN	Recall	Precision	Precision'	F_1
0.6	662	23743	64	0.912	0.0271	0.576	0.706
0.7	616	14255	110	0.848	0.0414	0.674	0.751
0.8	575	7716	151	0.792	0.0694	0.776	0.784
0.9	544	4837	182	0.749	0.101	0.835	0.790
1	544	4837	182	0.749	0.101	0.835	0.790

Table 4. Comparison with Threshold-based approach

Recall	*precision*		*precision'*		F_1	
	LR	TH	LR	TH	LR	TH
0.749	0.101	0.0605	0.835	0.752	0.790	0.750
0.792	0.0694	0.0478	0.776	0.705	0.784	0.746
0.848	0.0414	0.0351	0.674	0.637	0.751	0.728
0.912	0.0271	0.0234	0.576	0.539	0.706	0.678

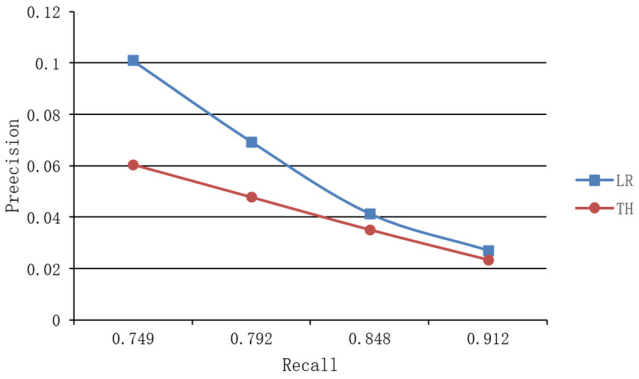

Fig. 8. Method precision comparison

made noticeable improvement in comparison to the threshold-based approach, especially when recall is not that high.

5.3 Our Findings

We examined the applications that have their own embedded text and description deemed unrelated, and classifies the reason behind such misclassification. We find that there are about 23 % applications deemed related by manual examination, containing adequate text length and indicative sentences. However, for the rest 77 % there are four reasons behind our "misclassification", the distrib-

Table 5. Distribution of reasons behind "Misclassification"

Type	Percentage(%)
Related	23
Inadequate Embedded Text	50
Short Description	26
Unsuited Description	12
Malicious Application	1

ution of which is as shown in Table 5. Note that some applications actually fall in multiple categories.

Inadequate Embedded Text. From Fig. 9, we can observe that some of the applications deemed unrelated have much less embedded text then normal ones.

The lack of embedded text is normally the result of two conditions. Either the application has very few functionalities so not much text is displayed in the user interface, a typical example of which is `Muzika`. This application only provides the functionality to search and download copyleft music, therefore, the only application related words in the embedded text are "search", "length", "size" and "bps", which can not set up an effective correlation with its description.

Another condition is when the developer do not follow the coding convention of putting interface text into the string.xml or layout file, but instead hardcoded it or put it in the database, leaving little useful information for our system. One typical example of this is `ANT+`, it has only a few entries in layout file and even fewer in string.xml, however, the magnitude of the strings hardcoded in the apk file actually approaches 72 kilo-bytes.

The inadequacy of embedded text can be somewhat mitigated by the introduction of hardcoded string as embedded text, however, it would introduce much noise to the system, considering that we should encourage the adherence to coding convention, we decide not to take in the hardcoded string into consideration.

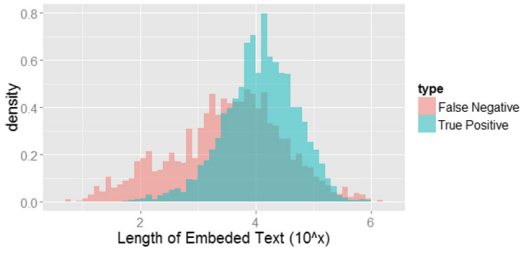

Fig. 9. Distribution of embedded text length

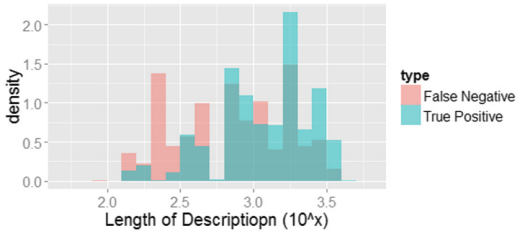

Fig. 10. Distribution of description length

Short Description. From Fig. 10, some of the applications deems unrelated has much shorter description then normal ones. This lack to description makes it harder to establish a connection between the application and the description. Typical applications that fall into this category is either made by individuals instead of a company or that it relies on introduction channel other the application market.

A typical example is `Subway Surfers News`, as a fan-made application displaying news, videos for the game `Subway Surfer`, the application is spread through fan groups of the game. Hence, its description is rather simple, only pointing out the functionality of news and video display, with no reference to functionalities implemented such as, QR-code scanning, initiating events, building albums, audio recording and coupons.

Inadequate descriptions like this could drive away potential users. Our system would warn the developers when such things happen, thus help the developers to better promote their application.

Unsuited Description. Some descriptions, although pretty long, failed to cover the actual functionalities of the application. Some elaborate on the introduction of their company (e.g. `DHgate Mobile`, some quoted many reviewers opinion (e.g. `Nexercise`), but talk little about the functionality.

All these lacks a strong correlation with the application, thus leading to a rather low relatedness. Our system can spot such problem, and suggest the developers to amend their description to a more informative state, making better impression on the potential users.

Although fetch from the Google Play in U.S., some applications still use a foreign language as its description (e.g. `Weibo`), these also pose as unsuited descriptions. Since some English words are mingled in the description they are not filtered out during preprocessing.

Malicious Application. Some applications due to reason unknown, use the application content for other, put on a different name and get into the market (e.g. `Friend Locator`, who stoles content from `360live`), which leads to difference between the application description and the embedded text.

6 Discussion

Based on the relatedness between embedded text and description, we study the state of description writing currently on the Android market. We implemented an innovative system to decided whether the description is in accordance with the application, helping the developers and Android market administrator to evaluate the quality of the description automatically.

Different from related works, our study focus on the semantic relatedness,using text shown to the user during usage to judge the description, which is an approach never attempted before.

6.1 Weaknesses

In calculating text-to-text relatedness, because the limit training set size we currently possess, we use ESA as the tool, which relies on external knowledge base in calculation. However, since new words are continually emerging in this age, external knowledge base may become outdated, failing to catch the meaning of new brand names, product names, etc. This may reduce the accuracy of our classification over time.

In terms of description aid, we are only able to alert the developers of unjust descriptions, without detailed suggestions on how it may be improved.

7 Related Work

The closest related work is AutoCog [16], which examines the consistency between an application's description on Google Play and the permissions it requires, they also leverage NPL to understand the functionality of an application, while they tried to fetch semantic features from "what the developer tells the user", i.e. the description, we get this information from "what the user actually see", i.e. the actual semantic strings, we argue that semantic strings extracted from user interfaces of applications contains more straightforward and more detailed information about what this application would do.

There are many previous works that focuses on these kinds of problems in Android permission system. Han et al. [11] proposed a framework of Collaborative policy administration, where the malicious applications can be detected due to abnormal permission configuration. The descriptions of applications may be used to measure the similarity of two applications, then help identify the abnormal permission configuration [17]. The descriptions of applications can be used to help role mining algorithms too. Enck et al. [6] observed that some specific combinations of permissions could be used as signature of malware, so they introduced a set of security rules and a tool called Kirin to check the application's permission requirements against these rules. Kirin cares about application's permission request only but does not examine whether these permission are really need by the application. Kathy et al. [4] moved one step further, they introduced

PScout to statically check the source code of Android and built a set of mappings between Android Framework APIs and the permissions each API needed. With this set of mappings at hand, one can easily verify that whether an application over claimed privileges that it would not use, but they cannot handle the permission misuse problem nor the situation where repackaged malware invoked additional APIs that are irrelevant to the functionality of the origin application to do malicious things.

8 Conclusion and Future Work

Our system takes in apk files and descriptions; uses natural language processing to normalize the text into semantic vectors; calculate the relatedness between vectors from the given text; taking into consideration size of embedded text, description length, and relatedness, decide whether the apk file and the description are related.

Assuming tested applications all have their descriptions and embedded text related, we performed a test over 7,570 applications, and achieved a recall of 91.2 % in the most tolerant scenario. Under a stricter standard, 25.1 % of the applications are misclassified, by manual examination, we find that among these 25.1 % of applications, about 77 % fall into one of the four categories: (1) Inadequate Embedded Text, (2) Short Description, (3) Unsuited Description, (4) Malicious Application.

We hope to dealt with the problem of inadequate embedded text, finding the threshold, below which the developers should be notified that the given application contains too little information. And if we detect that there is many semantic information hardcoded in the program, we should suggest the developers to perform a reconstruction.

For the to-be-outdated knowledge base, as the training set grows larger we consider using the training set itself, instead of the external knowledge base, to generate text-relatedness measurement. Online learning [12] can then be used to keep the measurement updated.

In the future we may design a better storage schema for the semantic vectors, so that pieces semantically close to the given text can be quickly located and used for description suggestion.

Acknowledgement. This paper is supported by 12th Five-Year National Development Foundation for Cryptography (MMJJ201301008), Key Lab of Information Network Security, Ministry of Public Security (C13612), Natural Science Foundation of Shanghai (12ZR1402600). We thanks anonymous reviewers for their comments.

References

1. Number of android applications. Technical report, AppBrain (2014)
2. Research also shows steady and significant drop in number of malicious apps being removed in past three years. Technical report, RiskIQ (2014)

3. An open-source api for the android market. https://code.google.com/p/android-market-api. Accessed 2014
4. Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D.: Pscout: Analyzing the android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 217–228. ACM, New York (2012)
5. Chau, M., Reith, R., Ubrani, J.: Worldwide quarterly mobile phone tracker. Technical report, International Data Corporation (2014)
6. Enck, W., Ongtang, M., Mcdaniel, P.D.: On lightweight mobile phone application certification. In: ACM Conference on Computer and Communications Security, pp. 235–245 (2009)
7. Fang, Z., Han, W., Li, Y.: Permission based android security: issues and countermeasures. Comput. Secur. (COSE) **43**, 205–218 (2014)
8. Fellbaum, C.: WordNet An Electronic Lexical Database (1998)
9. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: International Joint Conference on Artificial Intelligence, pp. 1606–1611 (2007)
10. Google. android-apktool. https://code.google.com/p/android-apktool. Accessed 2014
11. Han, W., Fang, Z., Yang, L.T., Pan, G., Wu, Z.: Collaborative policy administration. IEEE Trans. Parallel Distrib. Syst. (TPDS) **25**(2), 498–507 (2014)
12. Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the EM algorithm. In: International Symposium on Neural Networks (1993)
13. Knoth, P., Zilka, L., Zdrahal, Z.: Cross-lingual link discovery in wikipedia using explicit semantic analysis. In: The 9th NTCIR Workshop Meeting, pp. 6–9, Tokyo, Japan, December 2011. Knowledge Media Institute
14. Porter, M.: An algorithm for suffix stripping. Program-electron. Libr. Inf. Syst. **14**, 130–137 (1980)
15. Pregibon, D.: Logistic regression diagnostics. Ann. Stat. **9**, 705–724 (1981)
16. Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z.: AutoCog: measuring the description-to-permission fidelity in android applications. In: ACM Conference on Computer and Communications Security (2014)
17. Zhang, X., Han, W., Fang, Z., Yin, Y., Mustafa, H.: Role mining algorithm evaluation and improvement in large volume android applications. In: Proceedings of the First International Workshop on Security in embedded systems and smartphones (SESP 2013), conjunction with ASIACCS 2013 (2013)