VALIDATING ANDROID PERMISSION REQUESTS BY ANALYZING APP DESCRIPTIONS

A Thesis

by

M. Kaan Şahin

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
July 2016

VALIDATING ANDROID PERMISSION REQUESTS BY ANALYZING APP DESCRIPTIONS

Approved by:

Assistant Prof. Barış Aktemur, Advisor
Department of Computer Science
Özyeğin University

Associate Prof. Murat Şensoy
Department of Computer Science
Özyeğin University

Assistant Prof. Ali Çakmak
Department of Computer Science and Engineering
Istanbul Şehir University

Date Approved: 20 July 2016

To my loved ones

ABSTRACT

Android applications can access sensitive user data if they are granted certain permissions. Android security model gives users the responsibility to approve permission requests of applications. For this, a user needs to decide whether an app's functionality justifies the permission request. To aid users in their decision, we propose and evaluate a methodology that analyzes the description of an app and identifies unusual and unjustified permission requests. In contrast to existing techniques, our approach is not limited to a fixed set of permissions and does not need source code or binary app data to operate; yet, it is on par with the current state of the art in terms of accuracy.

ÖZETÇE

Android uygulamaları, bazı izinler kullanıldığı takdirde hassas kullanıcı bilgilerine erişebilmektedirler. Android'in güvenlik modeli ise, uygulamaların izin isteklerini onaylama sorumluluğunu kullanıcıya bırakmaktadır. Bu sebeple, kullanıcıların uygulamaların işlevlerinin istenen izinlerle örtüşüp örtüşmediğine karar vermeleri gerekmektedir. Kullanıcılara bu kararlarında yardımcı olmak için, verilen bir uygulamanın tanımlamasını inceleyerek izin isteklerinin geçerliliğini belirleyen yeni bir yöntemi bu çalışmada sunup değerlendirmekteyiz. Var olan diğer yöntemlerden farklı olarak; kullandığımız yöntem belli bir izin kümesiyle sınırlı değildir ve inceleme için kaynak koduna yada derlenmiş uygulama koduna ihtiyaç duymamaktadır, yine de mevcut yöntemlerle doğruluk açısından denk seviyede bulunmaktadır.

ACKNOWLEDGEMENTS

I would like to thank to Özyeğin University for letting this work to be possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Android operating system runs on millions of personal mobile devices. Android applications access and utilize personal data such as phone call history, address book, location, etc. to offer their services. The popularity and the ability to access personal data attract the attention of unethical developers who attempt to make profits by violating users' privacy, e.g. by collecting sensitive information and selling the data to advertisement companies.

In the Android security model, an application has to be granted permissions to access parts of the device and data. For instance, the `READ EXTERNAL STORAGE` permission allows an application to read the contents of the SD card of the device; `READ CONTACTS` allows access to the address book of the user. Although strong, this security model has a serious Achilles' heel: the users. When installing an application, Android displays the list of permissions that the app asks for. It is the user's responsibility to confirm or refuse installation of the app. The problem is, most of the users do not read what permissions the application is requesting; and even if they read, they do not know the meaning of permissions and their implications [1]. Privacy-violating applications use this fact to their advantage; they ask for many sensitive permissions that would normally be considered unrelated to their seeming purpose. Because many users simply approve the permissions right away, applications get excess permissions that they use later on to collect private data and perform harmful/unethical operations. If the users were more careful and they approved permission requests only if the request was justified, a majority of privacy violation problems could be prevented. A typical user can easily notice that it is natural for a calendar application to request

the `READ CALENDAR` permission, but it is unusual for it to ask for permission to send SMS messages. However, it could as well be that the application asks for the SMS permission to be able to send event reminders to other people for meetings. At this point, the user needs to make a judgement about whether the application they are about to install justifies the permission requests it is making. If the user is convinced that there is justification, they may go ahead and approve the permission requests, otherwise reject.

The way an application can convince the user is by means of the application meta-data, and in particular, the app *description*. If there is need for a permission that would normally be considered unusual, the app may include reasoning in its description about why it is asking for that specific permission. The reasoning may be explicitly stated, or implicit from the features of the app. For example, consider a banking application that is requesting access to the camera. Intuitively, this is a suspicious demand. Why would a banking app need to use the camera of the device? It might be that the application has a QR-code feature to perform some operations conveniently for their customer. The app might be requiring access to the camera to be able to read QR codes. In the app description, this feature may be conveyed to the user explicitly, as in "We ask for camera access to read QR codes", or it may be implicit from the description of features, as in "Use the QR codes on our daily bulletin to easily access our investment guide." From the perspective of the user, judging whether an app rightfully asks for a permission is likely to be time-consuming and boring. With this motivation, our goal in this work is to help the users with their decisions by making them skip ordinary/expected permission requests and instead focus on the suspicious ones. To this end, we propose and evaluate a methodology that analyzes the description of an app and identifies unusual/unjustified permission requests. Our **contributions** are as follows:

- Our method is oblivious of the specifics of permissions; it works with any permission, not with a fixed set.
- Experimental results show that our method competes with or is more accurate than the state of the art tools.
- Our approach does not need app source code or binary code.
- Our data set is publicly available (`https://github.com/m-kaan-s`).

This paper is organized as follows: In Chapter 2 we discuss the related work. Chapter 3 presents our solution approach. In Chapter 4, we experimentally evaluate our tool. Finally, in Chapter 5 we give our conclusions.

CHAPTER II

RELATED WORK

In this chapter we discuss related work in two groups. In Section 2.1, we evaluate existing literature regarding privacy issues of Android applications. In Section 2.2, we discuss similarity measures and their appropriateness for our context.

2.1 Android App Privacy

The two most closely related work are WHYPER [2] and AutoCog [3]. We used these two tools in our evaluation in Chapter 4.

WHYPER extracts sentences from descriptions and uses natural language processing (NLP) methodologies to associate sentences with particular permissions. For this, WHYPER compares the description sentences with data collected from API documents, based on the intuition that the permission-related phrases in the API documents will be similar to those in the app descriptions. This way, descriptions are marked relevant or irrelevant for a particular permission. As previously noted by others [3], WHYPER is limited by (*i*) lack of automation, (*ii*) lack or incompleteness of API documents for some permissions, and (*iii*) limited semantic information present in the API documents.

AutoCog uses advanced NLP and machine learning methodologies to analyze descriptions at the level of sentences, and identifies whether a sentence indicates the application's intent to use a particular permission. In contrast to WHYPER, AutoCog is automated and does not depend on API documents. However, like WHYPER, the set of permissions covered by AutoCog is fixed. AutoCog analyzes 11 permissions, WHYPER analyzes 3 (in Android 5.0, there are 43 readily-usable permissions that are deemed "dangerous"). Our approach does not restrict the analyzed permissions.

The granularity of our tool's output is coarser than WHYPER and AutoCog; we mark applications to be relevant/irrelevant to permissions at the whole description level, WHYPER and AutoCog give the results at the sentence level.

There is a group of related work that involves analysis of the application's binary. Gorla et al. [4] use both the description and the binary of an application to analyze the app for permission usage. They first categorize applications by processing their metadata using NLP and then clustering. Applications in each cluster are analyzed according to their API usage, which is extracted from the application binary. Applications that utilize API methods in unusual ways compared to the other apps in the same cluster are marked as suspicious. Detection rates for malware in this work are usually less than 60%. Avdiienko et al. [5] process application binary to find data flows that are different than the flows in other applications. While this approach may provide strong evidence of privacy violation, it has weaknesses against reflection, native code execution, and self decrypting applications. Chen et al. [6] also perform analysis of application's executable. They base their work on the motivation that a valid mobile application sends some data via the network or SMS to a remote receiver, then obtains a response, and gives the user some feedback by e.g. altering a visual component on the screen. A malware usually transmits sensitive data to remote servers without receiving a response or not providing a sensible feedback to the user. Again, this approach provides strong evidence of violation, but it requires a heavyweight runtime analysis. PUMA [7] and automatic categorization of applications [8] extract features from application binaries, and process the features via machine learning methods. PUMA categorizes permission usage to find patterns to detect malware. Automatic categorization approach [8] extracts embedded strings from applications for this purpose. These work were the earlier research on permission analysis and have been substantially improved by more recent research.

Zhang et al. [9] propose a method to generate security-centric app descriptions

by analyzing the app code. We consider this line of research complementary to ours.

Methods that utilize metadata processing have the ability to pre-emptively avoid harmful application installations, and also keep a blacklist of applications. Sarma et al. [10] use permission usage frequencies by categories, rare permission usages in some categories, and dangerous permission pairs. However, they do not utilize the information in application descriptions. Frank et al. [11] add price distribution and user ratings to this approach.

Benton et al. [12] focus on user behaviour and effectiveness of permission request methods instead of providing harmful application detection methods. It gives valuable information about how an application installation system must be designed to maximize user awareness.

2.2 Similarity Measures

In this section, we provide some information about some similarity methods which can be used in a similar manner to our approach.

First, we want to consider Euclidean distance [13]. In this method, simply the dissimilarity of two entities are calculated. Since this approach uses all properties of the entities which are to be compared, it is not suitable for detecting irrelevant permission usage in Android app descriptions. An Android app description may include words that justify a permission. But also, it may include totally unrelated words. So, it will get a low score. Even worse, another Android app description that does not contain any word which can justify a permission can get the same dissimilarity score. This means, we cannot use the reversed dissimilarity score, and this method is not appropriate in our context.

Another widely used method is the Jaccard index or Jaccard similarity coefficient [14]. This method basically examines relation of common properties of two entities to all properties of two entities. Problem in this method is that apps with small

descriptions or apps with very long descriptions will have poor similarity scores. So, this method is not appropriate for detecting irrelevant permission usage, but can be used to catch different apps with the same or greatly similar descriptions.

Along with these methods, Pearson correlation coefficient [15] is also used in similar works. This methods looks for linear correlation between two entities. But it is again dependent on the number of properties of two entities, and a small app description that justifies a permission usage will not get a good score when compared with a long app description that also justifies a permission usage.

Finally, there is cosine similarity method, which calculates a form of angular similarity between two entities. While it is very promising and widely used in text processing, it does not perform better than our method in our context. Details of this method's usage within tf-idf [16] and its evaluation is presented in Chapter 4.

CHAPTER III

SOLUTION APPROACH

A permission request of an app can be rationalized either because the permission is implicitly associated with the functionality provided by the app, or it is explicitly justified in the description of the app. In either case, the metadata of the app should be analyzed. There are various ways to approach this problem as discussed in Chapter 2. Our work takes a permission-centric approach, based on the assumption that a majority of the apps that request a particular permission rightfully ask for the permission, and only a minority do not provide justification.

We begin with a large set of descriptions of apps that request a particular permission p. We analyze the distribution of words that occur in these descriptions. This stage is similar to the training phase of machine learning algorithms. At this stage we analyze the application descriptions to gather information about what words are typically used by apps that request the permission p. After this training phase, when we face with the description of a new app that asks for p, we compare the words of the description to the previous analysis result. If there is sufficient similarity between the two, we conclude that the app justifies its request of p, otherwise the request is suspicious. This is repeated for each permission that the app requests.

Our training phase is straightforward, and the details are as follows: Starting from a set of app descriptions, we pre-process each description using a word filter that uses white-space and punctuation marks as word delimiters, and removes non-informative words like “at”, “in”, “on”. Stemming is not used since algorithm does not care about meanings of the words and actually needs original forms of words during cache building for ordering of words. Results of stemming enabled operation

$$p \in P \qquad \textit{(permissions)}$$

$$w \in W \qquad \textit{(words)}$$

$$d \in D := \text{multiset of words} \qquad \textit{(descriptions)}$$

$$D_p = \{d \in D \mid \text{app with description } d \text{ requests } p\}$$

$$\Omega_w^d = \frac{\text{Multiplicity of } w \text{ in } d}{|d|} \qquad \textit{(weight of } w \textit{ in } d\textit{)}$$

$$\Omega_w^p = \frac{\sum_{d \in D_p} \Omega_w^d}{|D_p|} \qquad \textit{(weight of } w \textit{ in } p\textit{)}$$

$$\Omega_{max}^p = max\{\Omega_w^p \mid w \in W\} \qquad \textit{(max word weight in } p\textit{)}$$

Figure 1: Formal definitions of the training phase.

can be seen at Chapter 4. For each app, we end up with a *multiset* of words as the app's description. We then calculate the weight of each word in each description. We prefer weights to raw frequencies to prevent long descriptions from dominating the analysis results. Then, we calculate the weight of each word w for each permission p, by averaging the weight of w over all the descriptions that request p. We store the weights in a *permission cache* to use later during app analysis. The formal definitions of the training phase are given in Figure 2. Note that training is **unsupervised**; it does not require manual examination.

After the training phase, comes the app analysis phase. Given a new app description, we apply the same word filtering that we did in the training phase. This gives us a multiset of words, d', as the description of the new application. We calculate the weights of words in d' in the same way we did during the training phase. After we have the word weights in d', we compare these weights to the weights in the permission cache for each permission p that the new application requests. The similarity between the two gives us the *conformance level* of d' to p. We use this value to decide whether the new app rightfully requests p.

The distinguishing feature of our work is in how we define the conformance level

Words in d' that have lower or equal weight than w, denoted $L_w^{d'}$, is defined as

$$L_w^{d'} = \{w' \in d' \mid \Omega_{w'}^{d'} < \Omega_w^{d'} \ \vee \ (w' >_{d'} w \ \wedge \ \Omega_{w'}^{d'} = \Omega_w^{d'})\}$$

where, $w' >_{d'} w$ is the order of appearance in d'.

Raw conformance of d' to p, denoted $R_p(d')$, is

$$R_p(d') = \sum_{w \in d'} \sigma(w, d', p)$$

where $\sigma(w, d', p)$, the score of w in d' with respect to p, is

$$\sigma(w, d', p) = \begin{cases} \Omega_w^{d'}, & \text{if } L_w^{d'} = \emptyset \\ \Omega_w^{d'} \times \sum_{w' \in L_w^{d'}} \dfrac{\kappa(w, w', p)}{|L_w^{d'}|}, & \text{otherwise} \end{cases}$$

where the coefficient $\kappa(w, w', p)$ is determined as follows:

$$\kappa(w, w', p) = \begin{cases} 0.8, & \text{if } \Omega_w^p = 0 \\ 1, & \text{if } \Omega_w^p > 0 \wedge \Omega_w^p \geq \Omega_{w'}^p \\ \Omega_w^p / \Omega_{w'}^p, & \text{otherwise} \end{cases}$$

Finally, $C_p(d')$, the adjusted conformance of d' to p, is

$$C_p(d') = R_p(d') \times max\{\Omega_w^p / \Omega_{max}^p \mid w \in d'\}$$

Figure 2: Calculating the conformance of a description d' to a permission p.

of a description to a permission, shown in Figure 2. Each word w that occurs in a description d' contributes to the raw conformance level of the description to a permission p. We calculate w's contribution by multiplying its weight in d' with a coefficient. The coefficient can be determined via various methods. We have decided to use the following intuitive and efficient definition: We find the words in d' that have the same or lower weight than w. We then compare each such word's weight in p with w's weight in p. If w does not exist in the permission cache, we use a coefficient of 0.8 to limit the contribution of unknown words to the overall result; we selected this coefficient value experimentally. If the weight of w in p is equal or higher

Word	Weight (%) in	
	Permission π	Application δ
A	30	35
B	10	25
C	35	-
D	9	25
E	-	08
F	16	07

Table 1: Sample word weight data.

(i.e. the relative order of the weights of the words in d' and in p are the same), the coefficient is 1. Otherwise, the coefficient is the ratio of w's weight in p to the other word's weight. Suppose words are ordered according to their weights. If a word's position in a description based on this ordering is similar to the word's position in a permission, the coefficient will be close to 1; a word that does not obey the ordering will be penalized with a lower coefficient, and will not make a significant contribution to the raw conformance level of the description. A hypothetical description where the words have exactly the same relative weight-ordering with respect to the permission cache would thus have a raw conformance score of 1.

While a raw conformance value gives a fine approximation to determine permission relevance, we use an extra step to adjust the conformance according to the usage of the most important words. Words with high weights in p are important and their absence in the analyzed application description hints that the application deviates from the majority of the applications that request p. With this intuition, we adjust the raw conformance value as shown in Figure 2. If the analyzed app uses the permission's most popular word in its description, the adjusted conformance value is the same as the raw value.

A description is said to conform to a permission if the adjusted conformance level is above a threshold value. We set this threshold to 0.5 to equally divide the conformance level space. This threshold value can be adjusted experimentally.

An artificial example:

To see the conformance calculation in action, we give a sample data in Table 1, for a fictional permission π and app description δ. Based on these data, the score of each word is calculated as follows:

A: Each word that has a lower weight than A in the app also has a lower weight in the permission. So, the coefficients are 1, and the score of A is equal to its weight in the app: 0.35.

B: Words D, E, and F are in L_B^δ. In the permission, F's weight (0.16) is higher than B (0.10); thus its coefficient is 0.10/0.16. D's weight is lower than B's weight; so, D's coefficient is 1. E does not exist in the permission (i.e. its weight in π is 0), bringing a coefficient of 1. In total, B's score is $0.25 \times (0.10/0.16 + 1 + 1)/3 = 0.219$.

C: This word does not exist in δ, hence has zero score.

D: Words E and F are in L_D^δ. F's weight in π is higher than D's weight in π. Hence, F's coefficient is 0.09/0.16. E does not exist in the permission, bringing a coefficient of 1. So, the score is $0.25 \times (0.09/0.16 + 1)/2 = 0.195$.

E: There is only one word, F, in L_E^δ. Because E does not exist in the permission cache, a constant coefficient of 0.8 is used, giving a score of $0.08 \times 0.8 = 0.064$.

F: Word F does not have any lower ranked words in the application. Therefore, its weight (0.07) is its score.

From these word scores, the raw conformance of δ to π is calculated as $0.35 + 0.219 + 0.195 + 0.064 + 0.07 = 0.898$. The application does not contain the most popular word of π (i.e. C). Instead, it contains the second most popular word of π (i.e. A). The adjustment to the raw conformance is thus 0.30/0.35. This gives us a conformance value of $0.898 \times (0.30/0.35) = 0.77$.

Word	Weight (%) in	
	Permission	Application
flashlight	3.04	13.33
turns	0.13	13.33
camera	0.80	13.33
flash	0.63	13.33
mobile	1.43	6.67
screen	0.70	6.67
white	0.05	6.67
completely	0.04	6.67
bug	0.38	6.67
fix	0.10	6.67
devices	0.33	6.67

Table 2: Word weights for the `CAMERA` permission and a flashlight application.

A real example:

For a real case short enough to be included here, let us look at the description of the net.dalar.android.flashlight app:

> *FlashLight*
>
> *Use you mobile as a flashlight. Turns your screen white completely. Turns camera flash on. Bug fix for devices with no camera/flash.*

The cache of the `CAMERA` permission and the weights of the words in the description are given in Table 2. The permission cache is calculated using a data set of 762 applications for the `CAMERA` permission. The words are sorted according to their weights in the application and the order of appearance. Based on these data, some of the word scores are calculated as follows:

flashlight: Each word that has a lower or equal weight than "flashlight" in the application also has a lower or equal weight in the permission. So, the coefficients are 1, and the score of "flashlight" is equal to its weight in the app, 0.133.

camera: There are 8 words in L_{camera}. Among these, "mobile" is the only one

whose weight in the permission is higher than "camera"s weight. So, word score for "camera" is $13.33\% \times (7 \times 1 + (0.80/1.43))/8 = 0.126$.

white: The words that are in L_{white} are "completely", "bug", "fix", and "devices". Only "completely" has a lower weight than "white" in the permission. So the word score is $6.67\% \times (1 + 0.05/0.38 + 0.05/0.10 + 0.05/0.33)/4 = 0.030$.

devices: This word is the last entry in the table, and has no lower-ranked words. So, its weight (0.067) is its score.

Combining the word scores, we get a raw conformance value of 0.777. The application's #1 word, "flashlight", also happens to be the most popular word in the permission cache. Therefore, the adjusted conformance value is the same as the raw value.

CHAPTER IV

EVALUATION

We evaluate our approach by comparing it with two state-of-the-art tools, WHYPER [2] and AutoCog [3], whose data are available online. We also compare our method with the standard term frequency-inverse document frequency (tf-idf) method. We then provide an analysis for the `SEND SMS` permission, which is not covered by either Autocog or WHYPER. We finally evaluate the sensitivity of our method to the size of the training data.

4.1 Comparison with Whyper

WHYPER's data contain manually examined app descriptions for three permissions: `READ CALENDAR` (195 apps), `READ CONTACTS` (190 apps), and `RECORD AUDIO` (200 aps). When going over their data, we noticed several cases where there was a structural error (i.e. inconsistent data with missing parts), or there was a manual examination error (i.e. a sentence was marked as relevant to a particular permission while it was irrelevant, or vice versa). We corrected the errors, and as a result, the permission-relevancy changed for 35 applications in the `READ CALENDAR` group, 8 in `READ CONTACTS`, and 26 in `RECORD AUDIO`. We provide the corrected data at our website.

We ran WHYPER and our tool on WHYPER's (corrected) data. For testing, we used 10-fold cross validation in the following way: For each permission, we shuffled and split the applications of that permission into 10 groups. For each group, we used the other 9 groups to train the tool. Then we used the result of the training phase to check what the tool predicts for permission-relevancy of each application in the first group.

Method	TP	TN	FP	FN	Pr	Re	Fs	Acc
READ CALENDAR								
Our tool	95	59	17	24	84.8	79.8	82.3	79.0
WHYPER	86	70	6	33	93.5	72.3	81.5	80.0
READ CONTACTS								
Our tool	104	49	30	7	77.6	93.7	84.9	80.5
WHYPER	90	71	8	21	91.8	81.1	86.1	84.7
RECORD AUDIO								
Our tool	134	25	30	11	81.7	92.4	86.7	79.5
WHYPER	107	47	8	38	93.0	73.8	82.3	77.0

Table 3: Comparing our tool with Whyper. Precision (Pr), recall (Re), F-score (Fs), and accuracy (Acc) values are given as percentages (%).

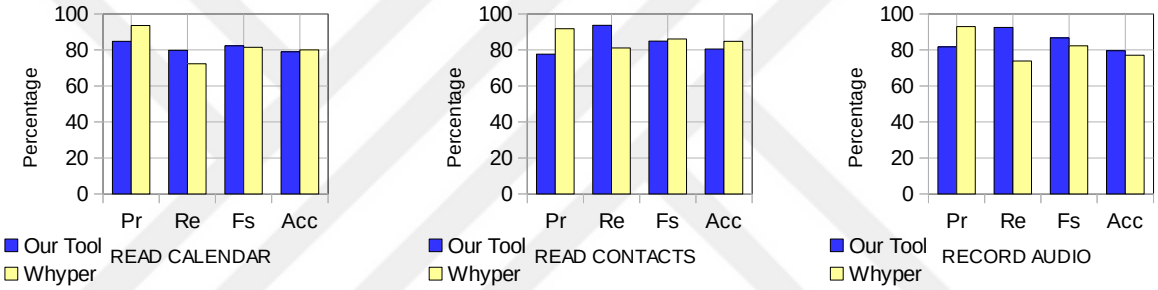


Figure 3: Bar chart for WHYPER evaluation

The results are shown in Table 3, where we give true positives (TP), true negatives (TN), false positives (FP), false negatives (FN), precision (Pr), recall (Re), F-score (Fs), and accuracy (Acc) values. These values have the definitions below. We illustrate the Pr, Re, Fs, and Acc values in Figure 3 for easier viewing.

TP: An application provides justification for the permission, and the tool has correctly identified the app to do so.

TN: An application does *not* provide justification for the permission, and the tool has correctly identified the app not to do so.

FP: An application does *not* provide justification for the permission, and the tool has incorrectly identified the app to do so.

FN: An application provides justification for the permission, and the tool has incorrectly identified the app not to do so.

Pr: TP / (TP + FP)

Re: TP / (TP + FN)

Fs: (2 × Pr × Re) / (Pr + Re)

Acc: (TP + TN) / (TP + TN + FP + FN)

Our approach and WHYPER give comparable F-score and accuracy values. The major differences are in precision and recall; while our recall is much better than WHYPER, the opposite holds for precision. Our approach does not have WHYPER's limitations listed in Chapter 2; our approach is fully automated and does not depend on API documents.

As a last step, we wanted to see the effect of stemming on our approach. For this, we have used porter stemming on the app description words. In the end, we have obtained lower scores when stemming was used. For `READ CALENDAR`, `READ CONTACTS` and `RECORD AUDIO` permissions, we have obtained 43, 43 and 81 invalid results, respectively. As stated in Chapter 3, our approach uses original forms of words as an extra property and stemming causes loss of that property.

4.2 Comparison with AutoCog

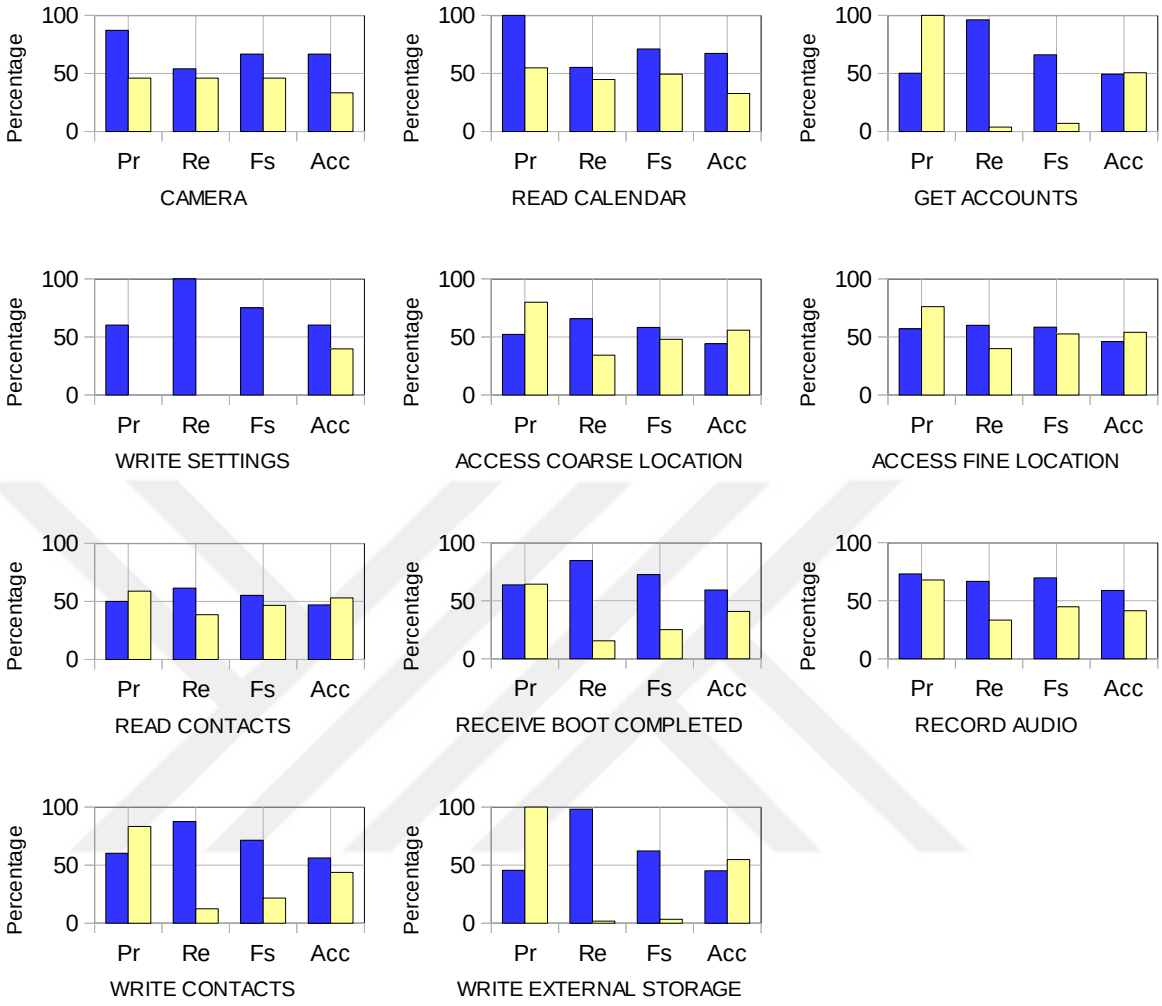
We compared our approach with another state-of-the-art tool, AutoCog [3]. AutoCog relates description sentences with zero or more permissions (out of a fixed set of 11 permissions). For evaluation, we could not use the manually examined data that were used for WHYPER, because many app descriptions have since been changed or removed from AutoCog's database. So, we downloaded arbitrary app descriptions from AutoCog's web site. If AutoCog's output for an app includes at least one sentence marked for a particular permission p, we concluded that AutoCog reports the app to be relevant to p. For our tool, we again did 10-fold cross-validation. We

Perm.	Dis.	Method	TP	TN	FP	FN	Pr	Re	Fs	Acc
1	81	Ours	27	27	4	23	87.1	54.0	66.7	66.7
		AutoCog	23	4	27	27	46.0	46.0	46.0	33.3
2	52	Ours	21	14	0	17	100.0	55.3	71.2	67.3
		AutoCog	17	0	14	21	54.8	44.7	49.3	32.7
3	158	Ours	78	0	77	3	50.3	96.3	66.1	49.4
		AutoCog	3	77	0	78	100.0	3.7	7.1	50.6
4	113	Ours	68	0	45	0	60.2	100.0	75.1	60.2
		AutoCog	0	45	0	68	-	0.0	-	39.8
5	59	Ours	23	3	21	12	52.3	65.7	58.2	44.1
		AutoCog	12	21	3	23	80.0	34.3	48.0	55.9
6	63	Ours	24	5	18	16	57.1	60.0	58.5	46.0
		AutoCog	16	18	5	24	76.2	40.0	52.5	54.0
7	49	Ours	16	7	16	10	50.0	61.5	55.2	46.9
		AutoCog	10	16	7	16	58.8	38.5	46.5	53.1
8	91	Ours	49	5	28	9	63.6	84.5	72.6	59.3
		AutoCog	9	28	5	49	64.3	15.5	25.0	40.7
9	80	Ours	38	9	14	19	73.1	66.7	69.7	58.8
		AutoCog	19	14	9	38	67.9	33.3	44.7	41.3
10	64	Ours	35	1	23	5	60.3	87.5	71.4	56.3
		AutoCog	5	23	1	35	83.3	12.5	21.7	43.8
11	124	Ours	56	0	67	1	45.5	98.2	62.2	45.2
		AutoCog	1	67	0	56	100.0	1.8	3.4	54.8

For each permission, 200 app descriptions are used.
Dis: Number of apps for which AutoCog and our tool disagree.
Corr: Apps among the disagreed ones for which our tool gives the correct, Autocog gives the incorrect answer.
Permissions: 1. `CAMERA`, 2. `READ CALENDAR`, 3. `GET ACCOUNTS`, 4. `WRITE SETTINGS`, 5. `ACCESS COARSE LOCATION`, 6. `ACCESS FINE LOCATION`, 7. `READ CONTACTS`, 8. `RECEIVE BOOT COMPLETED`, 9. `RECORD AUDIO`, 10. `WRITE CONTACTS`, 11. `WRITE EXTERNAL STORAGE`.

Table 4: Comparing our tool with AutoCog (small set).

counted the apps where AutoCog and our tool disagree. We did the experiment with a small (Table 4) and a large data set (Table 5) separately. Figure 4 is an illustration of Table 4 for easier viewing. For the small data set, we manually examined whether AutoCog or our tool gives the correct answer for each of the disagreed apps. For the larger data set, we reused these manual examination results to evaluate the disagreed apps. In general, our tool provided better results than AutoCog.



Legend: Blue (dark) bars are our tool's results, yellow (light) bars are Autocog's results.

Figure 4: Bar chart for Autocog evaluation.

Evaluation shows that, for some permissions, there may be a high degree of disagreement. This is mostly caused by the fact that the permissions do not provide rigid borders for their usage, and completely unrelated applications from different categories can legally use them. A clustering approach (e.g., CHABADA [4]) to group similar applications into categories may improve the results of both AutoCog and our tool. Due to the variation in the ratio of disagreements, we performed a sensitivity analysis of our tool on the training data set size, given in Section 4.5.

Permission	Apps	Dis.	Dis./Apps	Common	ComCorr.
CAMERA	4723	1607	34%	29	18 (62%)
READ CALENDAR	617	258	42%	28	24 (86%)
GET ACCOUNTS	8512	2908	34%	40	21 (53%)
WRITE SETTINGS	2195	607	28%	58	41 (71%)
ACCESS COARSE LOCATION	11399	6965	61%	45	24 (53%)
ACCESS FINE LOCATION	11484	6795	59%	42	24 (57%)
READ CONTACTS	3938	1129	29%	37	18 (49%)
RECEIVE BOOT COMPLETED	7281	2456	34%	68	39 (57%)
RECORD AUDIO	3165	1440	45%	49	34 (69%)
WRITE CONTACTS	1628	566	35%	54	32 (59%)
WRITE EXTERNAL STORAGE	22115	11707	53%	98	47 (48%)

Common: Number of disagreed apps in *both* small and large data set evaluation.
ComCorr.: Apps among the *Common* ones for which our tool gives the correct, AutoCog gives the incorrect answer.

Table 5: Comparing our tool with AutoCog (large set).

4.3 Comparison with tf-idf

In Chapter 3, we explained how we process descriptions to calculate word weights for a permission, which we called the *permission cache*. The app descriptions we process are documents. The permission cache that we build can also be considered a document, although an artificial one. Hence, it is intuitive to think that we could have used a standard document similarity method to see if a description conforms to a permission. For that purpose, we evaluate using the term frequency-inverse document frequency (tf-idf) method [16] as the permission conformance formula instead of the one we presented in Chapter 3. This way, we obtain a "master document" that represents a document group; in our case, the descriptions for a permission p. When we are given a new description, we find the similarity of this document to the master document using cosine similarity. If the similarity value is above a threshold value, we say that the description is relevant to the permission p. We set the threshold value experimentally. To evaluate how the tf-idf approach performs, we used the same

Permission	Total Apps	Examined	TP	TN	FP	FN	Pr	Re	Fs	Acc
SEND SMS	1310	131	60	48	14	9	81.1	87.0	83.9	82.4

Examined: Number of apps that have been manually examined.

Table 6: Evaluation of our tool with an arbitrary permission

manually examined data set we had for WHYPER comparison. We again applied 10-fold cross-validation methodology. For the READ CALENDAR permission, most of the similarity values were in the range of 0–0.2. Setting the threshold value to 0.1 gave the best result. For the READ CONTACTS and RECORD AUDIO permissions, we found the best threshold values to be 0.057 and 0.058, respectively. Using these values, the tf-idf approach incorrectly identified 67 applications for READ CALENDAR, 62 for READ CONTACTS, and 46 for RECORD AUDIO. When using the approach we propose in Chapter 3, the incorrect answers are fewer (see Table 3): 41, 37, and 41, respectively. So, our approach performs better than a standard tf-idf method in our context.

4.4 Analysis for a Permission not Covered by Autocog or Whyper

In this section, we show the results of our tool for a permission that is not covered by either Whyper or Autocog. We picked a potentially dangerous permission: SEND SMS. For the analysis, we arbitrarily downloaded descriptions of 1310 applications that request the SEND SMS permission. We again used the 10-fold cross-validation approach. We manually examined 10% (131) of the applications (chosen randomly) for evaluation. Results are shown in (Table 6). We see that for 82% of the applications (108 out of 131), a correct result was found by our tool.

4.5 Sensitivity to the Training Data Set

To evaluate how our tool reacts to the changes in the training data set, we arbitrarily selected the WRITE EXTERNAL STORAGE, WRITE SETTINGS, READ CONTACTS, and

RECORD AUDIO permissions. For each permission, we picked a fixed set of 50 descriptions that is always excluded from the training set. We then ran our tool with training data of increasing sizes, starting at 100. The results are in Figure 5 and 6. In Figure 5, at each test, we compared the results for the fixed 50 apps to the results reported by the preceding (i.e. smaller) data set test, and noted the percentage for which there is agreement between the two tests. E.g.: For WRITE EXTERNAL STORAGE permission, 94% of the results when using a training data size of 600 are the same as when using a data size of 400. In Figure 6, we provide the accuracy results obtained for each test. We see a fluctuation for the WRITE SETTINGS permission; however, the results are relatively stable for size 400 and larger. This analysis indicates that the evaluations we performed where we had training data sizes of ∼200 are reliable, yet the results may be improved with larger training data sets.

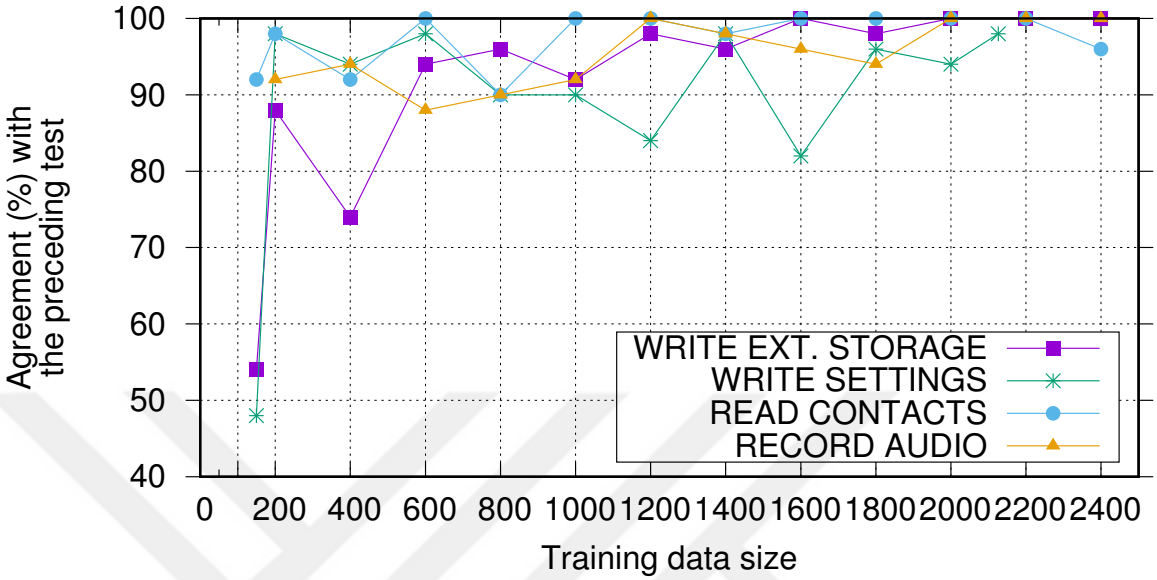


Figure 5: Sensitivity Analysis for the Training Data Set Change.

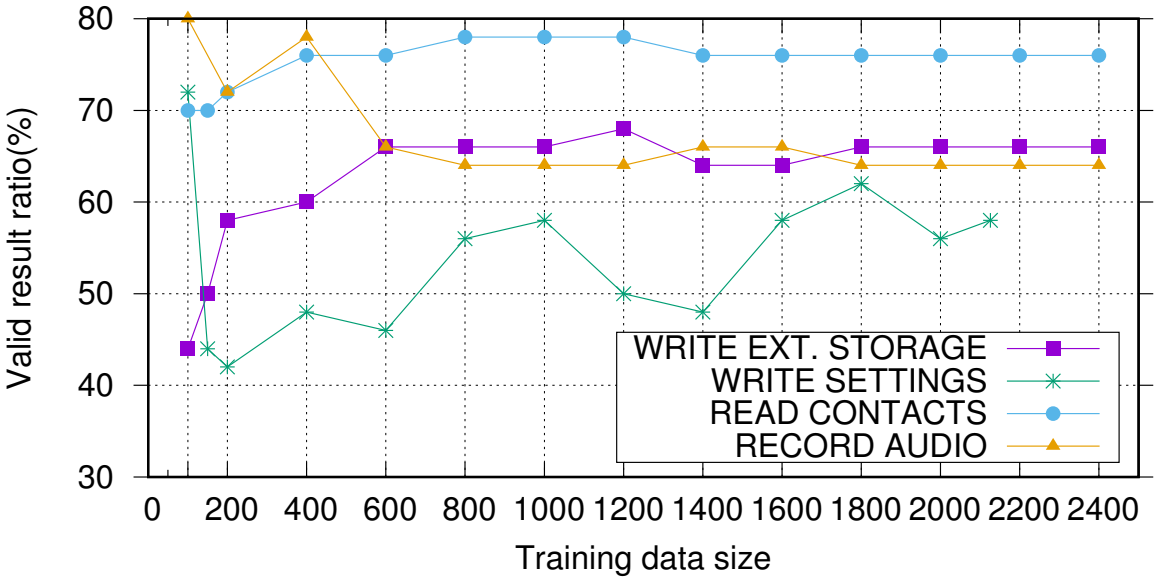


Figure 6: Sensitivity Analysis for the Training Data Set Change - Valid Result Ratios.

CHAPTER V

CONCLUSION

We proposed and evaluated a methodology to automatically detect Android applications' relevance to the permissions they request. We showed that our approach is better or on par with two state of the art tools. Our approach is not limited to a fixed set of permissions and does not depend on the application source code, binaries or API documents; it processes the application metadata only. We also provided a sensitivity analysis to evaluate how the results change with larger training data sets. As a final note, our approach is flexible for improvement. For example, user comments for particular apps can be put into a similar training stage and used during permission relevance analysis. This will also add an extra level of protection against app developers who fill their app descriptions with words to get around similar security systems. Our data are publicly available at `https://github.com/m-kaan-s`.

APPENDIX A

DEFINITIONS

API: Application Programming Interface. A set of interface methods and routines used for software development.

k - fold: A cross - validation method. Original data is equally divided to k randomly generated partitions. A single partition is used as validation data and k - 1 partitions are used as training data.

Malware: Malicious software. This term is usually used to describe software or software components which can steal private data, access data or components which may cause harm to owner.

NLP: Short for Natural language processing. It is used for human or natural language input processing via computer tools and software to obtain data to use in computer software and automated tools.

QR code: A certain kind of two dimensional barcode which is widely used in mobile device applications.

SD card: A non - volatile compact memory card which is widely used in mobile devices.

SMS: Short message service. A text messaging service which is widely used especially in mobile devices.

TF - IDF: Term frequency - inverse document frequency. A numerical statistic which shows the importance of a word in a document of a group or corpus.

Bibliography

[1] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *SOUPS 2012*, pp. 3:1–3:14.

[2] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *SEC 2013*, pp. 527–542.

[3] 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 2014*, pp. 1354–1365.

[4] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *ICSE 2014*, pp. 1025–1035.

[5] V. Avdiienko, K. Kuznetsov, A. Gorla, and A. Zeller, "Mining apps for abnormal usage of sensitive data," in *ICSE 2015*, pp. 426–436.

[6] X. Chen and S. Zhu, "Droidjust: Automated functionality-aware privacy leakage analysis for android applications," in *WiSec 2015*, pp. 5:1–5:12.

[7] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *CISIS - ICEUTE - SOCO 2012 Special Sessions*, pp. 289–298, 2013.

[8] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas, "On the automatic categorisation of android applications," in *CCNC 2012*, pp. 149–153.

[9] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," in *CCS 2015*, pp. 518–529.

[10] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in *SACMAT 2012*, pp. 13–22.

[11] M. Frank, B. Dong, A. P. Felt, and D. Song, "Mining permission request patterns from android and facebook applications," in *ICDM 2012*, pp. 870–875.

[12] K. Benton, L. J. Camp, and V. Garg, "Studying the effectiveness of android application permissions requests," in *PERCOM Workshops 2013*, pp. 291–296.

[13] H. Ding, G. Trajcevski, P. Scheuermann, Z. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measures," in *VLDB 2008*, pp. 1–11.

[14] V. Thada and V. Jaglan, "Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm," in *International Journal of Innovations in Engineering and Technology*, Volume 2 Issue 4, Aug 2013, pp. 202–205, 2013.

[15] A. T. Maturi and A. Elsayigh, "A comparison of correlation coeffi cients via a three-step bootstrap approach," in *Journal of Mathematics Research*, Volume 2 No 2, May 2010, pp. 3–10, 2010.

[16] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining Text Data*, pp. 77–128, Springer, 2012.

[17] E. Cambria and B. White, "Jumping nlp curves: A review of natural language processing research," in *IEEE Computational Intelligence Magazine*, Volume 9 Issue 2, May 2014, pp. 48–57, 2014.

[18] G. Sidorov, A. Gelbukh, H. Gomez-Adorno, and P. David, "Soft similarity and soft cosine measure: Similarity of features in vector space model," *Computación y Sistemas*, vol. 18, no. 3, pp. 491–504, 2014.

VITA

Mustafa Kaan Şahin, Electronics and Communication Engineer. Born in Şile, Istanbul and obtained bachelor's degree in Electronics and Communication Engineering field from University of Kocaeli. Developed commercially deployed high end payment systems, encrypted mailing systems and various embedded device configuration and usage tools. Currently working on Android and Linux based set top box DVB and IPTV products. Also apart from all these, has special interest for real time full computer system simulators/ emulators.