Re-checking App Behavior against App Description in the Context of Third-party Libraries

Chengpeng Zhang[1], Haoyu Wang[1,2*], Ran Wang[1], Yao Guo[3], Guoai Xu[1*]
[1] Beijing University of Posts and Telecommunications, Beijing, China, 100876
[2] Beijing Key Lab of Intelligent Telecommunication Software and Multimedia
[3] Peking University, Beijing, China, 100871
Email: {buptkick, haoyuwang, wangran2015, xga}@bupt.edu.cn, yaoguo@pku.edu.cn

***Abstract*—Recent research suggested promising approaches that identify potential malware by checking the inconsistence between app description and actual behavior of the app. However, state-of-the-art approaches have ignored the impact of third-party libraries (TPLs) when detecting outliers, which could affect the detection results greatly in two folds. On one hand, most Android apps would not list the functionality of TPLs in app description, which could cause false positives, as many apps that use TPLs will be identified as outliers. On the other hand, it is important to separate TPLs from custom code when analyzing the sensitive behaviors, otherwise the malicious behaviors of custom code will be obscured by TPLs. In this paper, we revisit the study of checking app behavior against app description in the context of TPLs. Experiment results on more than 400K Android apps suggest that more than 54% of apps are no longer identified as outliers after filtering TPLs, and we could identify roughly 50% of new outliers. Furthermore, removing the impact of TPLs could help to identify malware and pinpoint the malicious behavior of custom code. Out results shed a light on applying the TPL analysis to enhance a variety of mobile app analysis tasks.**

I. Introduction

Mobile malware is rapidly becoming a serious threat in recent years. The number of Android malware has risen steadily. Recent report [1] shows that the number of Android malware achieves almost 3.5 million in 2017, which has affected billions of devices.

A wide range of research has thus proposed approaches to analyze and detect malware, which could be roughly categorized into static analysis methods [2]–[5] and dynamic analysis methods [6], [7]. However, these approaches are usually not help against new malware families [8], as in many cases, it is difficult to differentiate between malware and benign apps because they may share the same/similar sensitive behaviors. For example, automated tools might detect that a flashlight app and a map app both use users' location information, while the map app is more legitimate than the flashlight app from the users' perspective. As a result, the difference between the users' expectations and the actual behavior of the app should be an important indicator for detecting a malicious app.

Recent research suggested promising approaches that identify potential malware/outliers by checking whether it behaves as advertised [9]–[11]. For example, WHYPER [9] and AutoCog [10] use natural language processing (NLP) techniques to infer the apps expected behaviors from app descriptions, and compare with the actual behavior extracted from the requested permissions. CHABADA [11] uses Latent Dirichlet Allocation (LDA) on app descriptions to identify the main topics of each app, and then clusters apps based on related topics. By extracting sensitive APIs used for each app, it can identify outliers which use APIs that are uncommon for that cluster. For example, for a group of 20 wallpaper apps, the app that has uncommon behaviors (e.g., access location and contacts) is probably up to malware.

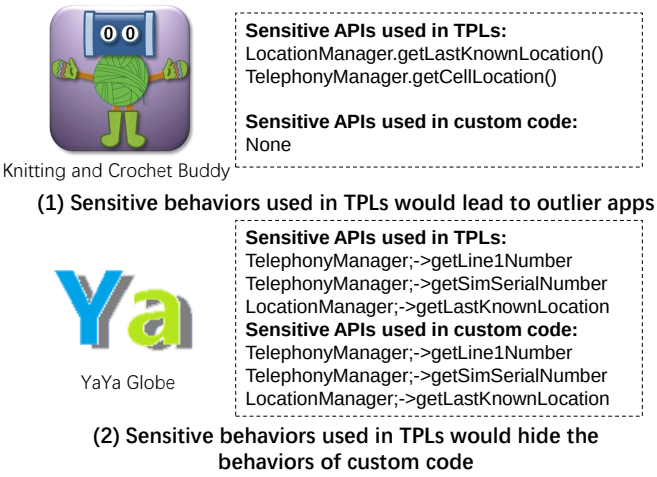

Fig. 1. Motivating Examples

Our work is motivated by CHABADA [11]. Although their experiment results are promising, they have ignored an important issue: *the impact of third-party libraries (TPLs)*. We argue that TPLs should be considered separately when checking app behavior against app description for two reasons.

First, TPLs (e.g., ad library, third-party analytics, social networking libraries, etc.) are widely used in Android apps. Previous work [12] suggested that more than 60% of the code in Android apps belongs to third-party libraries on average. However, **the use of TPLs could affect the outliers detection due to the reason that most Android apps would not list the functionalities of TPLs in their descriptions**. In our initial experiment on 100 popular Android apps, we found that

*Co-corresponding authors
DOI reference number: 10.18293/SEKE2018-180

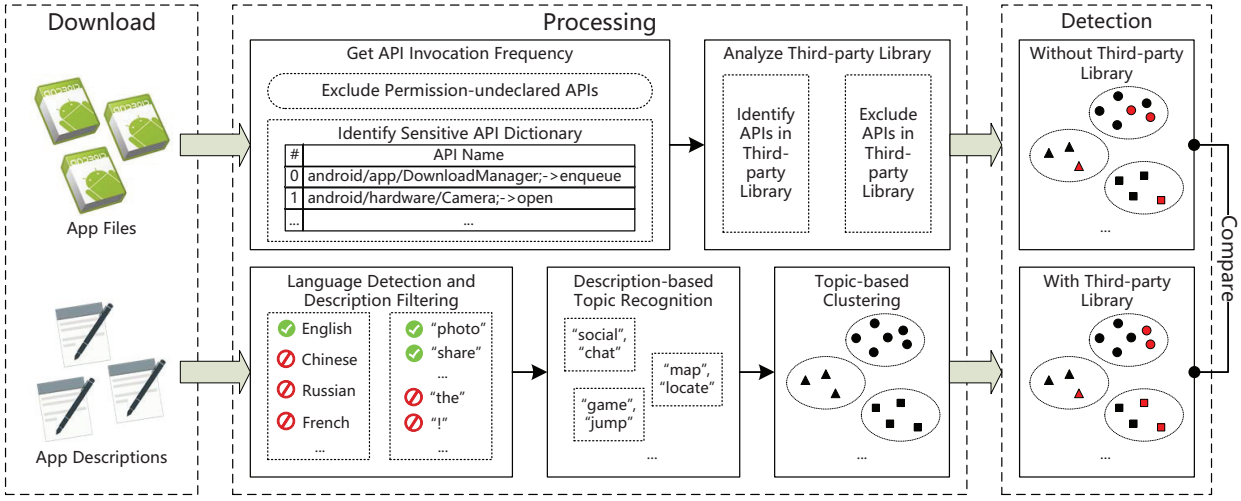

Fig. 2. Study Methodology

95% of them do not explicit show the functionalities of TPLs in app descriptions. For instance, as shown in Figure 1 (1), the app (androiddeveloperjoe.knittingbuddy) uses the Amazon Ads and InMobi library to display ads. It was identified as outliers because several sensitive APIs introduced by TPLs, although there are no sensitive behaviors exist in its main code.

Second, we argue that **it is important to separate TPLs from app custom code, which could be then used to determine whether the sensitive behaviors are introduced by custom code or not.** Otherwise, the malicious behaviors of core functionalities will be obscured by TPLs. For example, for a group of 20 wallpaper apps, the malicious app that tracks user's location information in its main code will be identified as normal app if most of these apps embed advertisement library that would access location data for targeted advertising. Indeed, more than 70% of popular apps in Google Play use advertising libraries [13]. The purposes of sensitive behaviors used in custom code and TPLs are usually different [14], [15]. As shown in Figure 1 (2), the sensitive behaviors used in TPLs would dilute the behaviors of custom code.

It is worth noting that some TPLs also show aggressive and malicious behaviors, as reported by many previous work [16]–[19]. However, the key idea of this paper is to separate TPLs from app custom code and determine whether the sensitive behaviors are introduced by custom code. The malicious behaviors introduced by app developers in the core functionalities of the app will be more obvious if we remove the noisy introduced by TPLs.

Hence, in this paper, we revisit the study of checking app behavior against app description in the context of TPLs. We take the first step to examine the impact of TPLs in pinpointing outlier apps. Experiment results on more than 400k apps (an order of magnitude higher than CHABADA) from Google Play suggest that TPLs introduce great impact on the outlier detection of Android apps, with more than 54% of apps are no longer identified as outliers after excluding TPLs. Besides, we could identify more than 50% of new outliers because *removing the impact of TPLs could sharp the sensitive behaviors of the main functionalities*.

II. Study Methodology

A. Crawling the dataset

We have downloaded more than 400 thousand free Android apps from Google Play in March, 2015. We crawled the meta-data of these apps, including the app names, app categories, app ratings, the number of installs, etc. We also downloaded all the apk files of these apps.

B. Overall Architecture

The overall architecture of our study is shown in Figure 2. For the crawled apps, we first apply natural language processing (NLP) techniques to the app descriptions for filtering and stemming. Then we take advantage of LDA to identify topics from app descriptions, and each app is represented as a topic vector. Apps with similar topic vectors will be grouped in the same cluster. For apps in the same cluster, we extract sensitive APIs used in them. Apps with unusual API used patterns will be labeled as outliers. Note that to explore the impact of TPLs on the outliers detection, we use a clustering-based approach to identify TPLs used in these apps and compare the results with and without TPLs.

C. Description-based Clustering

1) App Description Preprocessing: Considering that a too short description cannot represent the functionality of the app well, we first exclude the apps with the length of descriptions less than 10 words in our dataset. Then we use the language-detector [20] tool to detect the language of app description. Note that we only consider the apps whose descriptions are written in English in this work. Take advantage of Mallet [21], we build a list to filter out stop-words. Then we use the Snowball [22] to turn the words into stem form. At last, after app description preprocessing, we have 276,333 app samples left in the dataset.

TABLE I
THE 30 TOPICS EXTRACTED FROM THE 276,333 APP DESCRIPTIONS

#	Topic Name	Topic Keywords
0	sociality	share facebook twitter social chat app friend messag email send network post love peopl free photo creat sms featur easi
1	picture	pictur photo imag camera beauti make color design fashion galleri dress girl hair style effect frame choos creat app share
2	religion	church god prayer bibl christian book chapter app india indian islam lord read audio listen quran vers holi hindi day
3	racing	race car game speed play free slot drive win machin coin real simul truck park excit fun featur spin experi
4	workout	workout app weight number calcul exercis time train fit bodi measur result unit simpl convert track math system program base
5	shopping	shop servic offer search find book app store price order deal featur product locat custom direct inform home special mobil
6	child	child kid babi children game learn fun play anim app color draw pictur cat dog pet educ age free sound
7	jumping	play game jump run level shoot control fli fun enemi score tap collect challeng featur bird zombi power avoid world
8	sport	sport team score footbal player club world golf soccer countri leagu match app game live cup play includ flag fan
9	geography	citi nation state art year south area counti north west american countri includ san world award histori cultur york local
10	theme	theme icon font keyboard launcher instal download app appli set free select screen widget home press android function phone support
11	video	video mobil watch app movi free download content youtub connect updat channel stream copyright offici disclaim latest devic share
12	scenery	beauti natur enjoy fish water christma magic world tree sea room flower beach night day time light blue hous halloween
13	finance	money account mobil manag app bank credit check pay bill view payment onlin access secur transact transfer balanc servic inform
14	cooking	cook food recip make easi eat drink kitchen step fruit cake ingredi app love delici healthi cream ice chicken restaur
15	business	busi compani manag servic provid product develop market custom client industri experi profession design technolog work job offer
16	audio	audio music song radio listen play record app download station stream artist lyric free favorit rock danc player live featur
17	life	life time peopl make thing good day work person start love feel give tip find learn mind fact back don't
18	mobile	mobil phone call android devic app applic contact connect number messag user sms network wifi control data send password secur
19	calendar	calendar event inform school schedul app mobil view student class date access featur offici news confer connect find updat communiti
20	accounting	list real properti applic estat inform licens agreement licensor provid term data termin sale user mls servic state relat right
21	ringtone	rington sound quot app free applic android phone alarm set funni friend make fun joke inspir sleep effect download notif
22	dictionary	dictionari word english learn translat languag studi app test question answer chines letter quiz text search spanish french phrase featur
23	system	set button time screen widget app tap click mode display press select devic start phone batteri light version chang featur
24	puzzle	puzzl game play score level point fun time challeng match mode player card number free move block simpl bubbl ball
25	news	news read latest updat magazin app issu star content articl inform access stori world featur free download subscript www page
26	wallpaper	wallpap background live imag set free screen galaxi phone app applic devic support download home pictur anim samsung tablet
27	health	healt medich app treatment inform mortgag care help patient doctor profession complet emerg diseas time featur easi access free
28	map	map citi locat app guid inform travel hotel place find search navig weather gps tour rout time restaur featur trip
29	documenter	file manag android app version devic support applic note data featur search user list creat card googl record save code

2) Identifying Topics from App Descriptions: A "topic" consists of a cluster of words that frequently occur together in app descriptions. To identify sets of topics, we resort to topic modeling using LDA based on Mallet [21]. Note that we choose the same number of topics (n=30) to be identified by LDA as CHABADA [11] for comparison. Table I shows the result of 30 topics (with top 20 keywords of each topic) that we have identified from the 276K app descriptions. Note that the "topic name" in the second column is the abstract concept we manually assigned to that topic, which is a meaningful word that can represent the corresponding topic. As a result, each app is represented as a topic vector, and each dimension of the topic vector represents the probability that an app belongs to the corresponding topic.

3) App Clustering: To identify the clusters of apps with similar descriptions, we take advantage of K-means++ algorithm[1] for app clustering based on the topic vectors.

Note that the K-means++ algorithm needs to be given either some initial potential centroids, or the number K of clusters to identify. Thus one challenge here is to identify the number of clusters that should be created. One straight-forward idea is to run the algorithm multiple times and each time with a different K number. Based on a set of clustering results, we would then be able to identify the best one.

We propose to use Genetic Algorithm (GA) [23] combined with K-means++ to determine the best number of clusters, which was shown to be effective in previous work [24]. GA is suitable to this problem space because the selection, crossover and mutation steps of GA could help to choose the optimal value of K. With the generations evolving, the better individuals with higher fitness values will emerge [24]. As a result, take advantage of GA, we will get the best value of K.

We used the silhouette coefficient[2] as the fitness value to measure the effectiveness of clustering results. Note that each cluster is represented by a silhouette coefficient, which is based on the comparison of its tightness and separation. This silhouette coefficient shows how closely each element is matched to the other elements within its cluster, and how loosely it is matched to other elements of the neighboring clusters. The range of the value of silhouette coefficient is -1 to 1. When the value of the silhouette coefficient of an element is close to 1, it means that the element is in the appropriate cluster. Thus, we compute the average of the silhouette coefficient for each solution using K as the number of clusters, and we select the solution whose silhouette coefficient was closest to 1.

In this paper, we use the scikit-learn [25] to implement K-means++ algorithm and the calculation of silhouette coefficient. The implementation of GA is based on Pyevolve [26], an evolutionary computation framework. As a result, to achieve the highest silhouette coefficient on average, we choose 29 as the best number of clusters should be created. The clustering result for the 267,333 apps we analyzed is listed in Table II.

[1]https://en.wikipedia.org/wiki/K-means%2B%2B

[2]https://en.wikipedia.org/wiki/Silhouette_(clustering)

TABLE II
THE RESULT OF APP CLUSTERING.

#	# Apps	Most Important Topics
0	5825	sport(94.2%), puzzle, jumping
1	11791	business(95.3%), accounting, finance
2	11508	shopping(98.5%), cooking, picture
3	8660	workout(98.6%), finance, racing
4	4711	cooking(99.8%), geography, theme
5	7967	audio(96.9%), geography, video
6	5079	racing(99.8%), jumping, geography
7	5800	health(98.3%), geography, mobile
8	7853	dictionary(99.4%), theme, puzzle
9	18498	puzzle(99.2%), jumping, racing
10	10449	mobile(99.5%), finance, theme
11	5817	ringtone(97.9%), jumping, wallpaper
12	10084	calendar(99.5%), geography, finance
13	6567	finance(100.0%)
14	12485	documenter(95.6%), mobile, finance
15	7025	scenery(78.2%), jumping(5.4%), wallpaper(3.7%)
16	5954	theme(100.0%)
17	10791	system(95.7%), theme, wallpaper
18	9084	map(98.8%), geography, calendar
19	7952	picture(99.7%), wallpaper, geography
20	8539	sociality(94.3%), mobile, picture
21	17604	jumping(99.9%), geography, wallpaper
22	28138	geography(20.7%), accounting(5.5%), jumping(4.8%)
23	6410	religion(98.5%), geography, wallpaper
24	8194	child(97.0%), puzzle, picture
25	7088	video(99.4%), jumping, wallpaper
26	8069	news(95.4%), geography, video
27	9518	life(88.8%), jumping, cooking
28	8873	wallpaper(99.8%), theme, scenery

Each cluster contains apps whose descriptions contain similar topics, as shown in Column Most Important Topics. The percentages reported in the last column represent the weight of the dominant topic within each cluster.

D. App Processing

1) Identifying the Sensitive APIs: To identify outliers regarding their actual behavior, we need to identify the sensitive APIs used in each app. CHABADA uses a set of sensitive APIs derived from STOWAWAY [27]. The dataset was published in 2011 and it contains a list of sensitive APIs of Android version 2.2, which is outdated and incomplete. In this work, We use a list of permission-related APIs from PScout [28], which contains 680 sensitive APIs.

As shown in previous work [29], APIs with the same name but different parameters always share the similar functionalities. Thus we only take account of the name of the API, i.e., the APIs with the different parameters but the same name are identified as the same API. At last, there are only 428 APIs left from the 680 permission-sensitive APIs.

For each app, we generate an API feature vector where each dimension represents the invocation frequency for the corresponding API. Note the previous work [30] suggested that there are unreachable APIs in Android apps, i.e., these apps do not declare the corresponding permissions. Thus, we extract the declared permissions for the manifest of each app and filter the APIs that use undeclared permissions.

2) Identifying Third-party Libraries: One key idea in this paper is to measure the impact of third-party libraries in description-based outlier detection. Thus, we first need to identify the code that belongs to third-party libraries, and then we generate the API feature vector for each app with and without third-party libraries respectively.

In this paper, we have implemented a clustering-based approach which is shown to be effective in LibRadar [31]–[33] to identify the common frameworks used in apps. We extract the API call features at the package level and then we enforce strict comparison here to cluster all the features into groups, which means that only when the features of two packages are exactly the same can they be clustered. We choose the 128 as the threshold to identify third-party libraries, which means that as long as the package has occurred in more than 128 apps, we will regard it as the common library. With this threshold, we are able to find that roughly 70% of the code belongs to TPLs. Then we check the usage of sensitive APIs in TPLs to generate the feature vectors without TPLs as comparison.

E. Outliers Detection

In this paper, we use the Isolation Forest [34] algorithm to identify outliers, which was shown to be effective in identifying anomalous points in a large number of data. Besides, Isolation Forest algorithm has low memory requirements and linear time complexity, so it is more suitable for processing high-dimensional data. Note that the Isolation Forest algorithm needs two important parameters, one is the amount of trees, the other one is the amount of samples. In our study, we use the default value of sklearn, i.e., the amount of trees is set as 100 and the amount of samples is set as 256.

III. EVALUATION

A. The Impact of Third-party Libraries

First, we want to answer the research question: *what is the impact of TPLs in pinpointing outliers?* For this purpose, we compare the results of outliers detection using Isolation Forest algorithm with and without TPLs respectively. The result of outliers detection is shown in Table IV.

Surprisingly, it turns out that only 721 (46.5%) apps are still identified as outliers while excluding TPLs. Moreover, 723 apps (50.1%) are newly identified as outliers after excluding TPLs in our experiment. Thus we further analyzed the reasons that leading to the inconsistence between the results.

1) "False Positive" Outliers: For the apps that are no longer identified as anomalies after excluding TPLs, we randomly picked 5 apps for each cluster (145 apps in total) and manually inspect the code for understanding the reason.

At a result, we found that almost all these apps belong to the reason that *TPLs have introduced sensitive APIs that are not rarely used in the custom code*. For example, the app named "Puzzle Code" (with package name "com.yiqusoft.puzzle") uses two ad libraries named "domob" and "adsmogo". These two ad libraries use several sensitive APIs (e.g., "getLastKnownLocation") to display customized ads. However, these sensitive APIs are never invoked in the custom code. Thus, this app was identified as outliers without filtering TPLs.

TABLE III
THE RESULT OF MANUALLY ANALYSIS.

Cluster No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	Total
With TPLs	1	0	2	1	2	6	0	6	1	2	3	6	1	2	0	0	9	2	0	7	3	2	2	6	2	0	0	6	9	81
W/O TPLs	2	0	4	4	2	10	2	10	2	4	4	6	2	4	0	0	10	2	0	8	4	4	4	10	4	0	0	10	10	122

TABLE IV
THE RESULT OF OUTLIERS DETECTION.

	# Outliers	# Common	% Percent
Before	1,551	721	46.5%
After	1,444	721	49.9%

2) "False Negative" Outliers: For the apps that are newly identified as outliers, we randomly picked 5 apps for each cluster (a total number of 145 apps) and manually inspect the code to explore the reason.

We found that the anomalous behaviors introduced by app developers in the core functionalities of the app will be more sharp if we remove the impact of TPLs. For instance, the app "com.appgame7.fruitsbreak" uses a certain amount of sensitive APIs to collect user's phone number and location information in both custom code and TPLs. Most of the apps in the same cluster have embedded the ad libraries that share similar sensitive behaviors. The malicious behaviors in the custom code will be more obvious after removing the impact of TPLs.

B. The Behaviors of Outliers

We further explore the research question: *whether our approach could be used to identify malicious behaviors of Android apps?* For this purpose, we first manually inspect the top outliers as produced by our technique and classified them as malicious or normal. Then we upload all the identified outliers (2,274 apps in total) to VirusTotal[3] to check how many of them are flagged as malicious.

1) Manually Inspection: Only human can interpret properly what is in an app description. Therefore, we manually inspect the top 10 outliers for each cluster (290 apps in total). We first examine their descriptions, the list of sensitive APIs used, and the corresponding decompiled code. Then we install them on a real smartphone (nexus 5) and manually test it. Here we use TCPDump[4] to collect and analyze the network traffic.

Note that some outliers may not be malicious due to the inadequate descriptions. We would classify an app as malicious if: (1) the app collects user's privacy information (e.g., phone number and location) and uploads it to a remove server without explicit advertise the sensitive behaviors either in its description or in its privacy policy. For example, the app "com.appblast.popclock" is a simple alarm clock app, whereas it collects user's phone number and uploads it to a remote server. Another Android app "com.yoursite.lockfingerscanner" claims that it is able to achieve finger unlock function, whereas it is a grayware that only contains download links of other apps. (2) the app is reported as malware by the pre-installed anti-virus tool[5] if we install it on the smartphone. For example, after the app named "TouchNPaint" (with package name "game.child.paint") is installed, the AVL engine will report it as malware with family name "a.gray.mfpad". The result of manually analysis is shown in Table III. After eliminating TPLs, we could identify more malicious outlier apps.

TABLE V
DETECTION RESULT OF VIRUSTOTAL.

	"False Positive" Apps	"False Negative" Apps	Common
Flagged by VT	205	359	434
Total	830	723	721
Percentage	24.70%	49.65%	60.19%

2) Detection Result of VirusTotal: We then upload all the identified outliers (with and without TPLs, 2,274 apps in total) to VirusTotal to check how many of them are flagged by current anti-virus engines. As shown in Table V, 639 apps are labeld as malicious by at least one engines before filtering TPLs, and the number rised to 793 apps if we removed the impact of TPLs. It is also interesting to note that, although roughly 25% of the "false positive" apps are flagged by VirusTotal, most of them are labeld as "AdWare", which means that the malicious behaviors are introduced by ad libraries that embedded in the app. This result suggests that *removing the impact of TPLs could help to better pinpoint the malicious behaviors of custom code*.

IV. DISCUSSION

In this paper, we revisit the study of checking app behavior against app description in the context of TPLs. We use several heuristic methods similar with that used in CHABADA [11] to identify the topics and clusters, which could be potentially improved. Moreover, we mostly rely on manually efforts to analyze the results, which may exist bias.

Our study suggests that TPLs play an important role in mobile app analysis, which could be potentially used to enhance a variety of mobile app analysis tasks, e.g., malware detection and app clone analysis [12], [35].

V. RELATED WORK

A large mount of related work focus on bridging the gap between app description and user expectation. WHYPER [9] and AutoCog [10] use NLP techniques to infer permission use from app descriptions. Yu et al. [30] revisited this approach and revealed that using description and permission will lead

[3]virustotal.com

[4]http://www.tcpdump.org

[5]AVL for Android, com.antiy.avl

to many false positives. Thus they proposed exploiting the privacy policy and its bytecode to enhance the malware detection based on app description. Our work is motivated by CHABADA [11], which uses LDA on app descriptions to identify the main topics of each app, and then clusters apps based on related topics. By extracting sensitive APIs used for each app, it can identify outliers which use APIs that are uncommon for that cluster. Ma et al. [29] extended CHABADA by proposing an active and semi-supervised approach to detect malware using both known benign and malicious apps. Wang et al. [15], [36], [37] proposed to infer the purpose of permission use. However, all of the previous studies do not consider the impact of TPLs when checking whether the app behaves as it advertised.

VI. Concluding Remarks

In this paper, we present a study to show that TPLs play an important role in pinpointing the inconsistence between app description and app behavior. Based on the extensive experiment on more than 400K apps, we show that more than half of the apps are no longer identified as outliers after filtering TPLs, and we could identify more new outliers. The results in this paper could shed a light on a new perspective for researchers that TPLs could be potentially used to enhance a variety of mobile app analysis tasks.

VII. Acknowledgement

This work is supported by the science and technology project of State Grid Corporation of China: "Research on Key Technologies of Security Threat Analysis and Monitoring for Power Mobile Terminals" (Grant No. SGRIXTKJ[2017]265).

References

[1] "2018 malware forecast: the onward march of android malware," https://nakedsecurity.sophos.com/2017/11/07/2018-malware-forecast-the-onward-march-of-android-malware/, 2018.

[2] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *MobiSys 2012*, pp. 281–294.

[3] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Asia JCIS 2012*. IEEE, 2012, pp. 62–69.

[4] M. Liu, H. Wang, Y. Guo, and J. Hong, "Identifying and analyzing the privacy of apps for kids," in *Proceedings of the HotMobile 2016*, pp. 105–110.

[5] Z. Kan, H. Wang, G. Xu, Y. Guo, and X. Chen, "Towards light-weight deep learning based malware detection," in *The 42nd IEEE International Conference on Computers, Software & Applications (COMPSAC 2018)*.

[6] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, vol. 32, no. 2, p. 5, 2014.

[7] L.-K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis." in *USENIX security symposium*, 2012, pp. 569–584.

[8] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu, "Why are Android apps removed from google play? a large-scale empirical study," in *15th International Conference on Mining Software Repositories (MSR 2018)*.

[9] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in *USENIX Security 2013)*, pp. 527–542.

[10] 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.

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

[12] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Wukong: A scalable and accurate two-phase approach to android app clone detection," in *ISSTA 2015*, pp. 71–82.

[13] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 221–233, Jun. 2014.

[14] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and Purpose: Understanding Users' Mental Models of Mobile App Privacy Through Crowdsourcing," in *UbiComp '12*, 2012, pp. 501–510.

[15] H. Wang, Y. Li, Y. Guo, Y. Agarwal, and J. I. Hong, "Understanding the purpose of permission use in mobile apps," *ACM Trans. Inf. Syst.*, vol. 35, no. 4, pp. 43:1–43:40, Jul. 2017.

[16] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in Android ad libraries," in *MoST 2012*.

[17] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *WISEC'12*, pp. 101–112.

[18] F. Dong, H. Wang, L. Li, Y. Guo, G. Xu, and S. Zhang, "How do mobile apps violate the behavioral policy of advertisement libraries?" in *Proceedings of the HotMobile 2018*, pp. 75–80.

[19] H. Wang, Y. Guo, Z. Tang, G. Bai, and X. Chen, "Reevaluating android permission gaps with static and dynamic analysis," in *Proceedings of GlobeCom*, ser. GlobeCom'15, 2015.

[20] "Language detection library for Java," 2016. [Online]. Available: https://github.com/optimaize/language-detector

[21] "Mallet: A machine learning for language toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu

[22] "Snowball: A language for stemming algorithms," http://snowballstem.org, 2001.

[23] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, Oct 1988.

[24] X. Yang, D. Lo, L. Li, X. Xia, T. F. Bissyand, and J. Klein, "Characterizing malicious android apps by mining topic-specific data flow signatures," *Information and Software Technology*, vol. 90, pp. 27 – 39, 2017.

[25] "Scikit-learn: Machine learning in Python," http://scikit-learn.org, 2011.

[26] "Pyevolve: A complete genetic algorithm framework written in pure Python," 2009. [Online]. Available: http://pyevolve.sourceforge.net

[27] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of SPSM 2011*, pp. 3–14.

[28] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in *CCS 2012*, pp. 217–228.

[29] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," in *COMPSAC 2015*, pp. 179–184.

[30] L. Yu, X. Luo, C. Qian, S. Wang, and H. K. Leung, "Enhancing the description-to-behavior fidelity in android apps with privacy policy," *IEEE Transactions on Software Engineering*, 2017.

[31] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: Fast and accurate detection of third-party libraries in android apps," in *ICSE '16*, 2016, pp. 653–656.

[32] H. Wang and Y. Guo, "Understanding third-party libraries in mobile app analysis," in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17, 2017, pp. 515–516.

[33] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Automated detection and classification of third-party libraries in large scale android apps," *Journal of Software*, vol. 28, no. 6, pp. 1373–1388, 2017.

[34] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, pp. 3:1–3:39, Mar. 2012.

[35] W. HaoYu, W. ZhongYu, G. Yao, and C. XiangQun, "Detecting repackaged android applications based on code clone detection technique," *SCIENCE CHINA Information Sciences*, vol. 44, no. 1, pp. 142–157, 2014.

[36] H. Wang, J. I. Hong, and Y. Guo, "Using text mining to infer the purpose of permission use in mobile apps," in *The 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2015)*, pp. 1107–1118.

[37] H. Wang, Z. Liu, Y. Guo, X. Chen, M. Zhang, G. Xu, and J. Hong, "An explorative study of the mobile app ecosystem from app developers' perspective," in *Proceedings of the 26th International Conference on World Wide Web (WWW 2017)*, pp. 163–172.