Checking App Behavior Against App Descriptions

Yudong RAO
North Carolina State University
Raleigh, North Carolina
yrao3@ncsu.edu

ABSTRACT

How do we know a program does what it claims to do? After clustering Android apps by their description topics, Gorla et al.[2] identify outliers in each cluster with respect to their API usage. A "weather" app that sends messages thus becomes an anomaly; likewise, a "messaging" app would typically not be expected to access the current location. Applied on a set of 22,500+ Android applications, our CHABADA prototype identified several anomalies; additionally, it flagged 56of novel malware as such, without requiring any known malware patterns.

KEYWORDS

Android, malware detection, clustering, description analysis

1 INTRODUCTION

It is a longstanding problem for developers to check if a program does what it claims to do. And now it has been a problem for computer and cellphone users too. Whenever a new app is installed, users will face the risk of the app being "malware", especially for Android users.
Instead of focusing on whether an app's behavior matches a specific pattern or not(a common practice for current research and industry), Gorla et al.[2] looks at whether an app behaves as advertised and proposed an approach called CHABADA.
Android apps are chosen as the domain of this paper. Natural language description from the Google Play Store is used as a proxy for the advertised behavior of an app. And the set of Android application programming interfaces (APIs) that are used from within the app binary is used as a proxy for its implemented behavior. The key idea is to associate descriptions and API usage to detect anomalies. CHABADA takes five steps, as illustrated in Figure 1. We are diving into the details of these steps in the following sections.

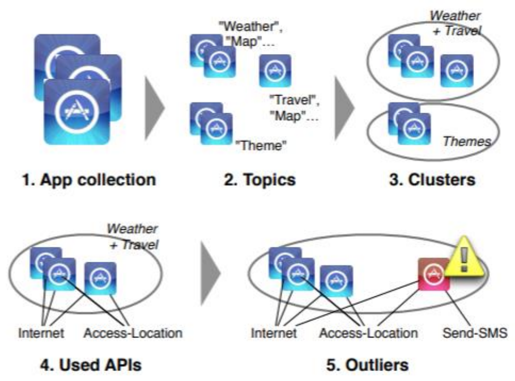


Figure 1: Detecting applications with unadvertised behavior. Starting from a collection of "good" apps (1), we identify their description topics (2) to form clusters of related apps (3). For each cluster, we identify the sentitive APIs used (4), and can then identify outliers that use APIs that are uncommon for that cluster (5).

2 CLUSTERING APPS BY DESCRIPTION

CHABADA is short for "Checking App Behavior Against Discriptions of Apps". This method starts with collection of Android Apps. Then it identifies topics of app descriptions and clusters the apps based on common topics.

2.1 Collecting Apps

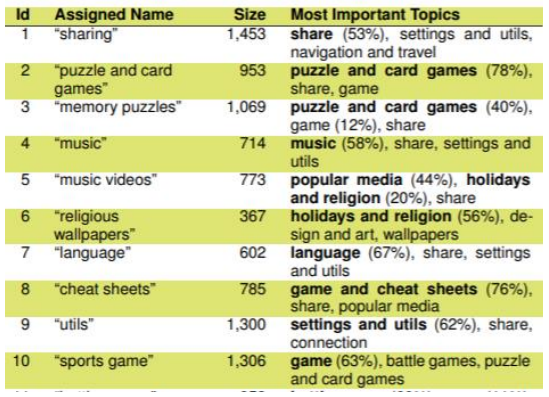
A large set of applications were collected from the Google Play Store. With automated collection scripts, Gorla et al.[2] downloaded the top 150 free applications in each of the 30 categories in the Google Play Store during the Winter and Spring of 2013. Finally a total of 32126 apps were obtained. As CHABADA is set to identify outliers before they get released to the public, it only uses name and description.

2.2 Preprocessing Descriptions with NLP

Standard techniques of natural language processing were applied for filtering and stemming of descriptions. Google's *Compact Language Detector* was used to detect and remove non-English paragraphs of app descriptions. Then *stop words*(such as "the", "is", "which",...) were removed and *stemming* was applied on all descriptions.
With that, Gorla et al.[2] eliminated those applications whose description would have less than 10 words after NLP preprocessing and those without any sensitive APIs (see Section 3 for details). This resulted in a final set of 22,521 apps.

2.3 Identifying Topics with LDA

To identify sets of topics for the apps under analysis, *Latent Dirichlet Allocation*(LDA)[3] is used for topic modeling.
LDA relies on statistical models to discover the topics that occur in a collection of unlabeled text. A "topic" consists of a cluster of words that frequently occur together.
Implementation of this paper feeds output of NLP pre-processing into the Mallet framework[1]. With LDA, 30 topics were generated and an app would belong to at most 4 topics. An app is considered related to a topic only if its probability for that topic is at least 5%.



Id	Assigned Name	Size	Most Important Topics
1	"sharing"	1,453	**share** (53%), settings and utils, navigation and travel
2	"puzzle and card games"	953	**puzzle and card games** (78%), share, game
3	"memory puzzles"	1,069	**puzzle and card games** (40%), game (12%), share
4	"music"	714	**music** (58%), share, settings and utils
5	"music videos"	773	**popular media** (44%), **holidays and religion** (20%), share
6	"religious wallpapers"	367	**holidays and religion** (56%), design and art, wallpapers
7	"language"	602	**language** (67%), share, settings and utils
8	"cheat sheets"	785	**game and cheat sheets** (76%), share, popular media
9	"utils"	1,300	**settings and utils** (62%), share, connection
10	"sports game"	1,306	**game** (63%), battle games, puzzle and card games

Figure 2: Example of resulting app clusters.

2.4 Clustering Apps with K-means

Topic modeling characterized each application by a vector of affinity values(probabilities) for each topic. Then we use this vector to do K-means clustering.

2.5 Finding the Best Number of Clusters

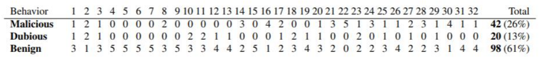
Authors used the *elements silhouette*[4], which is the measure of how closely the element is matched to the other elements within its cluster, and how loosely it is matched to other elements of the neighboring clusters to identify the best number of clusters. The average of the elementsâĂŹ silhouette is computed for each solution using K as the number of clusters, and the solution whose silhouette was closest to 1 is selected.

2.6 Resulting App Clusters

Figure 2 shows the partial list of clusters that were identified for the 22,521 apps analyzed.

3 IDENTIFYING OUTLIERS BY APIS

Now that the app clusters are generated. Authors began searching for outliers regarding their actual behaviors.
For each Android app, the binary APK file was extracted, and with a *smali* diassembler, all APO invocations were extracted.
Using all API calls as features would induce overfitting in later stages. Therefore, authors selected a subset of APIs, namely the sensitive APIs that are governed by an Android permission setting (access of camera, sending messages etc.).
The next step is to identify outliers, that is, those applications whose API usage would be abnormal within their respective topic cluster. One-Class Support Vector Machine learning (OC-SVM) is used to achieve such an objective.
With the sensitive APIs as binary features, CHABADA trains an OC-SVM within each cluster with a subset of the applications in order to model which APIs are commonly used by the applications in that cluster. The resulting cluster-specific models are then used to identify the outlier applications. By ranking the elements (i.e. apps) by their distance to the OC-SVM hyperplane, we can identify which ones have behaviors that differ the most from what has been commonly observed.



Behavior	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	29	30	31	32	Total
Malicious	1	2	1	0	0	0	0	2	0	0	0	0	0	3	0	4	2	0	0	1	3	5	1	3	1	1	2	3	1	4	1	1	**42** (26%)
Dubious	1	2	1	0	0	0	0	0	0	2	2	1	1	0	0	0	1	2	1	1	0	0	2	0	1	0	1	0	1	0	0	0	**20** (13%)
Benign	3	1	3	5	5	5	5	3	5	3	3	4	4	2	5	1	2	3	4	3	2	0	2	2	3	4	2	2	3	1	4	4	**98** (61%)

Figure 3: Manual assessment of the top 5 outliers, per cluster and total.

4 EVALUATION

4.1 Outlier Detection(RQ1)

Gorla et al.[2] ran CHABADA on all 32 clusters, as described in Section 3. Following K-fold validation, they partitioned the entire set of 22,521 applications in 10 subsets, and used 9 subsets for training the model and 1 for testing. They ran this 10 times, each time considering a different subset for testing. Out of the whole list of outliers identified in each run, they identified the top 5 outliers from the ranked list for each cluster. These 160 outliers were then assessed whether they would really exhibit suspicious behavior by manual assessment. And the result is shown in Figure 3.

Top outliers, as produced by CHABADA, contain 26% malware; additional 13% apps show dubious behavior. So CHABADA effectively identifies anomalies.

4.2 Malware Detection(RQ2)

A dataset of 1200 known malicious Android apps by Zhou et al.[5] was used for evaluation of Malware Detection. The OC-SVM model is used as a classifier here. Classification was performed in three ways. First was classification with topic clusters. CHABADA succeeded in detecting the majority of malware. Next was classification without clustering, which yielded more false negatives. Gorla et al. also tried classification using giving categories as indicated in Google Play Store. And the result showed that clustering by description topics is superior to clustering by given categories, as the precision of malware classification was 56% for topic clusters and 47% for given categories.

5 CONCLUSION

By clustering apps by description topics, and identifying outliers by API usage within each cluster, Gorla's[2] CHABADA approach effectively identifies applications whose behavior would be unexpected given their description. Several examples of false and misleading advertising were identified and as a side effect, a novel effective detector for yet unknown malware was obtained. Just like mining software archives has opened new opportunities for empirical software engineering, we see that mining apps and their descriptions opens several new opportunities for automated checking of natural-language requirements.

REFERENCES

[1] Mallet: A machine learning for language toolkit. http://malet.cs.umass.edu. (????). 2002.
[2] Florian Gross Andreas Zeller Alessandra Gorla, Ilaria Tavecchia. 2014. Checking App Behavior Against App Description. In *ICSE 2014 Proceedings of the 36th International Conference on Software Engineering*. ACM, 1025–1025.
[3] Michael Jordan David M. Blei, Andrew Y. Ng. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.
[4] P. Rousseeuw. Silhouettes. 1987. a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20, 1 (1987), 52–65.
[5] X. Jiang Y. Zhou. 2012. Dissecting Android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy (SP)*. IEEE.