Android App Classification and Permission Usage Risk Assessment

Yidong Shen[1(✉)], Ming Xu[1], Ning Zheng[1], Jian Xu[1], Wenjing Xia[1], Yiming Wu[2], Tong Qiao[2], and Tao Yang[3]

[1] Internet and Network Security Laboratory of Hangzhou Dianzi University, Hangzhou, China
{151050043,mxu,nzheng,jian.xu,161050051}@hdu.edu.cn
[2] School of Cyberspace, Hangzhou Dianzi University, Hangzhou, China
{ymwu,tong.qiao}@hdu.edu.cn
[3] Key Lab of the Third Research Institute of the Ministry of Public Security, Shanghai, China
yangtao@stars.org.cn

Abstract. With Android6.0, users can decide whether to grant an app runtime permission. However, users may not understand the potential negative consequences of granting app permissions. In this paper, we investigate the feasibility of using an app's requested permissions and the intent-filters, app's category and permissions requested by other apps in the same category to better inform users about whether to install a given app and the risk scores associated with granting each of the app's required permissions. In an evaluation with 10,979 benign and 3,205 malicious apps, we demonstrate the effectiveness of the proposal approach.

Keywords: Android · Runtime permission · Risk score · Category

1 Introduction

More than seven million malicious mobile apps were identified in 2016 [10]. However, smartphones remain vulnerable to malware. Nevertheless, there is an increasing number of advanced benign apps that provide intelligent and personalised services, such as location-based services and social sharing services, even though such apps may have potential security and privacy risks. For instance, users may not expect their locations (e.g. home locations, workplaces) and other privacy information (e.g. contact lists, SMS records) to be spied by the third party apps [19].

The Android platform provides several security measures such as the Android permission system, that reduce security and privacy risks. The Android operating system requires apps to request permissions before they can use certain system data and features [7]. Users must decide whether to grant such permissions. However, this approach has been proven ineffective. Studies have shown that

I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 567–577, 2018.
https://doi.org/10.1007/978-3-030-00916-8_52

more than 70% of smartphone apps request permission to collect data that is not directly required by the app's primary functionality. However, typically, users rush through the installation process and tend to grant all permissions [6]. In addition, even if users pay attention to the permission request, most do not have sufficient knowledge to understand permissions and their possible harm. Effective permission recommendations can provide additional information that can help users grant permissions appropriately to prevent privacy breaches [14].

Recent studies about Android permission recommendations can be found in the literature. Several studies have focused on expert recommendations based on crowdsourcing [9,13,14], utilising an app's runtime behaviour to evaluate the risks associated with granting permissions [12] and learning user behaviour to propose rules with different levels of abstractions [11]. Prior to Android 6.0, all permissions had to be accepted in order to install an app; therefore, these studies proposed modifying an Android source code to modify its permission system. They also could not consider the permission group mechanism in Android6.0.

With Android 6.0, the system permissions in Android are divided into two categories, normal and dangerous (runtime). Normal permissions do not directly risk the user's privacy. If an app lists a normal permission in its manifest, the system grants the permission automatically. Apps request runtime permissions as required at runtime. Android 6.0 has nine runtime permission groups with 24 runtime permissions related to personal user information and critical system resources. If a permission associated with a permission group has been granted, future permission requests in the same group are granted automatically for the given app. For example, if an app had previously requested and been granted the READ_CONTACTS permission, the WRITE_CONTACTS permission will be granted automatically because they are both in the Contacts permission group.

In this study, we consider the intended functionality of the app, i.e. what the app is expected to do, and which runtime permissions are requested by apps with a similar functionality to create an effective permission risk score. Currently, Google Play divides apps into 49 categories. Apps in the same category provide similar functionalities. The risk score measures whether a risk is commensurate with the benefit when granting runtime permission to an app in a given category. The higher the risk score, the greater the risk. Conversely, a low risk score indicates that granting permission would be beneficial and the associated risk is not significant.

We also utilise intent-filter to assess the risks with installing an app. Intent is a complex messaging system in the Android platform, and is considered as a security mechanism to hinder apps from gaining access to other apps directly. To receive intents, apps must define what type of intent they accept in the AndroidManifest.xml file, as intent-filter.

Therefore, we have designed an automatic analysis framework that does not require modifying the Android permission system to provide users with additional information prior to installation of an app. The proposed framework consists of malicious app detection and permission recommendation. Malicious app detection assesses the risks associated with installing an app, and permission

recommendation assesses the risk score of each requested permission. First, we collect permissions and intent-filters from AndroidManifest.xml and then feed this extracted data to a Random-Forest-based classifier [8]. After the classifier has been trained using a training set, it can classify apps as benign or malicious. Utilising the frequency of occurrence of a runtime permission in different categories, the framework provides risk scores for the requested runtime permissions of benign apps relative to the app's category.

The contributions of this study can be summarised as follows.

1. To provide users with additional information prior to installing an app, we propose an automatic framework that combines Android malware detection and permission recommendations.
2. We introduce an effective risk score for each requested app permission relative to the given app's category.
3. For efficiency, we utilise lightweight features that enable extracting features in reasonable time.

The remainder of this paper is organised as follows. Related work is discussed in Sect. 2. Section 3 describes the datasets, and our methodology is presented in Sect. 4. We describe experiments in Sect. 5, and conclusions are presented in Sect. 6.

2 Related Work

In this section, we outline the state-of-the-art studies that focus on Android malware detection, Android app recommendation and Android permission recommendation.

2.1 Android Malware Detection

DREBIN [5] is notable among early machine-learning-based malware detection approaches. It performs a broad static analysis, gathering features including permissions, intent-filters, API calls, strings, etc., and then utilises support vector machines to classify apps. It also provides explainable detection. However, DREBIN does not consider the benefit of granting a permission to an app in a specific category. For example, the permission SENS_SMS has the same risk value in the Communication and Sport app categories.

Sokolova et al. [16] modelled permission-based decision support systems to classify apps. They modelled permission requests by different categories using graphs, obtained patterns for each category and verified the performance of different patterns in order to select the most descriptive patterns. Their results demonstrate that they could detect 80% of the malware with a low false-positive rate (FPR).

Sarma et al. [15] proposed a risk warning system based on the occurrence of 24 overall permissions (manually identified as dangerous) in each app category. A risk signal is triggered if an app requires critical permissions that are used by less than a given percentage of apps in the corresponding category. Understanding the reason for a risk signal helps the user determine whether to install an app.

2.2 Android Permission Recommendation

DroidNet [14], Rashidi et al. [13] and Jha et al. [9] utilised crowdsourcing technology to recommend whether to grant permissions. An evaluation of DroidNet demonstrated that, given a sufficient number of experts in the network, it can provide accurate recommendations and cover most app requests given a small coverage from a small set of initial experts.

Oglaza et al. [11] learned user behaviours to propose rules with different levels of abstractions. Such rules can help users protect their privacy more easily.

Rashidi et al. [12] proposed XDroid, an Android app resource access risk estimation framework that employs a hidden Markov model. XDroid utilises an app's runtime behaviors to evaluate risks associated with granting an app's permissions. However, XDroid cannot provide additional information about requested permissions prior to app installation.

2.3 Android App Recommendation

Taylor et al. [17] proposed SecuRank, a contextual permission analysis framework, to recommend functionally-similar alternative apps that require less sensitive access to the device. They discovered that up to 50% of apps can be replaced by a preferable alternative. They noted that alternatives are more likely to be available for free and popular apps.

Zhu et al. [19] proposed a recommendation system that considers app ratings and app-related privacy issues, such as required permissions. They developed an app risk score based on the number of required permissions and the occurrence of such permissions in a given app category, i.e. a lower penalty is incurred if a permission is used more frequently in a given category, and as more permissions are required by an app, its associated risks increase.

3 Datasets

In this section, we describe the datasets used in our study. In our experiments, we used two datasets consisting of 14,184 and 6,400 apps. The number of app categories in each dataset was 32. The former dataset included 10,979 benign and 3,205 malicious apps. The latter dataset contained 6,400 benign apps (200 apps per category). All apps in the datasets were from Androzoo [4], which is a growing collection of Android apps collected from several sources, including the official Google Play app market. We determined the app categories in our datasets from Google Play. To prepare reliable ground truth data, each app was uploaded to VirusTotal [3] and scanned by tens of different Antivirus product. If no malicious content was detected, the app was considered benign. If more than four VirusTotal scanners detected a malicious content, the app was considered malicious. This procedure ensures that our data is (almost) correctly split into benign and malicious apps, even if several scanners falsely labels a benign app as malicious.

4 Methodology

The proposed framework involves three main steps: (1) extracting features from apps, (2) classifying apps using machine learning and (3) assessing the risk score of runtime permissions requested by apps classified as benign and recommending permissions by providing the risk score. Here, the overall objective is to provide users with additional information to help them make decisions.

4.1 Feature Extraction

Every Android app must include a manifest file called AndroidManifest.xml which contains information of package, including requested permissions and components of the app such as activities, services, intents, intent-filters, content providers etc.

To obtain permissions and intent-filters, we first used Androgurad [1], which is an open-source project for the static analysis of Android apps, to obtain the AndroidManifest.xml file. Then, we utilised the Python Beautiful Soup [2] library to extract features from the obtained file. In this study, we use permissions and broadcast receivers, a type of intent-filter, as features. A broadcast receiver is used to receive broadcast intent sent by an Android system. Finally, the extracted features including normal permissions, runtime permissions and broadcast receivers are embedded in feature vectors that are used to classify the apps, and runtime permissions are used to assess the permission risk scores.

4.2 App Classification Using Machine Learning

We used a machine learning method to automatically train a model to distinguish between benign and malicious apps. The feature vectors were used to train a classifier. Thus, manual analysis was not required to construct corresponding malware detection rules. We selected a Random-Forest-based [8] classifier to classify apps.

4.3 Permission Risk Assessment

In this section, we present our methodology for assessing the risk score of runtime permissions requested by a given app that is classified as benign. We focus on creating an effective risk score. An effective risk score should have a simple semantic meaning that is easy to understand. Understanding the risk score will help users determine whether to grant each requested permission. Most malware might be simply repackaged versions of official apps. Evidence of the widespread use of repackaging by malware writers is provided in MalGenome [18], a dataset in the Android security community, where 80% of the malicious samples are known to be built via repackaging other apps. If the malicious code interacts with more sensitive resources than the original app, new permissions must be requested in the Manifest file. Therefore, malicious apps request more permissions with high risk scores than benign apps. When a user sets a threshold

to filter out permissions with high risk scores (e.g. greater than 0.75), a small percentage of permissions requested by benign apps will be denied, and many permissions requested by malicious apps will be denied.

We are inspired by the observation described in the literature [15,19]. The observation states that, if a permission is used more frequently in a given category, the penalty incurred by this permission in this category will be lower. [15] use this observation to detect malware, a risk signal is triggered by an app if it requests critical permissions used by less than a certain percentage of apps in the given app category. [19] employ this observation to calculate the risk score of an app and recommend apps by considering both an app's popularity and the user's security preferences. [15,19] prove the validity of this observation in malicious and benign apps, respectively.

We calculate a permission's risk score for apps that belong to different categories according to the frequency of occurrence of the permission in these categories. A smaller permission risk score indicates that the permission is likely required and the risks associated with allowing the app are smaller. We used a dataset that contains 6,400 benign apps (200 apps in each category) to calculate the risk score. $RiskScore_{ij}$ denotes the risk score of granting permission j to an app in category i. $RiskScore_{ij}$ is defined as follows.

$$RiskScore_{ij} = \frac{\sum_{c \in C_i} f_{cj}}{\sum_{c \in C} f_{cj}}$$

Here, f_{cj} is the number of apps in category c that request permission j. C is the category set (32 categories), and C_i is a subset of C. All the categories that the number of apps request permission j greater than f_{ij} are in C_i. Note that a permission's risk score decreases as it is used more frequently in the apps of a given category. $RiskScore_{ij}$ has a simple semantic meaning, i.e. the risk score of permission j requested by apps in category i is less than or equal to $1 - RiskScore_{ij}$ percent of risk scores of permission j requested by apps that request this permission in the category balance dataset.

In terms of the permission group mechanism in Android 6.0, we show users all of the risk scores of an app's permissions requests and use the highest risk score of requested permissions in a permissions group to represent the risk score of the given group.

5 Experiments

Here, we evaluate the performance of the proposed approach. All experiments were conducted on a 4.2-GHZ four-core CPU, 32 GB main memory personal computer. These experiments were conducted to evaluate the performance of the malware detection and permission recommendation system in the proposed approach.

5.1 Evaluation of Malware Detection

In our first experiment, we evaluated malware detection performance. A dataset with 10,979 benign and 3,205 malicious apps was used in this experiment. We used 10-fold cross validation, which randomly selects parts of the data for training and the rest for testing. This was repeated 10 times to obtain a reasonable result. The two criteria utilised in our experiments were the true-positive rate (TPR) and FPR. The TPR is the proportion of actual positives correctly identified as such, and the FPR is the proportion of actual negatives falsely identified as positive. Here, positive indicates a malicious app, and negative indicates a benign app. The TPR and FPR values of the proposed approach were 86.8% and 0.99%, respectively. The results demonstrate that our machine-learning-based malware detection can detect 86.8% of the malicious apps with only a few false detections.

All the features we used in our experiments are lightweight features that can extracted from AndroidManifest.xml. It took 127 min to extract permissions and intent-filters from 14,184 apps. The average time required to extract features was 0.5 s. From this evaluation, we conclude that the proposed framework is capable of performing this task in practical time.

5.2 Evaluation of Permission Recommendation

In our second experiment, we evaluated the performance of permission recommendation based on the risk score. Note that apps classified as benign are used in the proposed approach's permission recommendation system. Here, 10,870 benign and 422 malicious apps were considered. We focus on the following permission groups: Phone, SMS, Location and Contacts. Permissions in the Location group are used to obtain the user's location information. Permissions in the Phone group are used to obtain the state of the device, IMEI, the call log and the ability to initiate a phone call. Permissions in the Contacts group are used to access the list of accounts in the Accounts Service and read/write contact information. Permissions in the SMS group are used to receive, read and send text messages.

We evaluated the performance of the risk score by setting a threshold range from 0.05 to 0.9. If the risk score of a permission group is greater than $1 - threshold$, the permissions in this group will be denied. This evaluation was configured as follows. The deny ratio of a permission group is defined as the number of permissions denied in the given permission group divided by the number of permission requests by apps in the same group. The overall deny ratio is defined as the number of permissions denied in the four permission groups divided by the number of permission requests by apps in these groups.

The performance of the proposed approach is shown in Table 1. We list the performance of the four permission groups and overall performance. Here, 'B' denotes benign, and 'M' denotes malicious. For example, Contacts B indicates the deny ratio of the Contacts group for benign apps. The results show that the deny ratio of each permission group for benign apps was less than the deny ratio

for malicious apps. When the threshold for each permission group is less than or equal to 0.25, the deny ratio shown an obvious distinction. We recommend that users deny a permission if the risk score of the permission group it belongs to is greater than 0.75. If the user denies permissions for which the risk scores of the corresponding permission groups are greater than 0.75, the overall deny ratio for benign and malicious apps are 27.3% and 56.7%, respectively. The results show that the risk score we provide is effectiveness.

On the basis of user preferences, users can also deny a permission according to the risk score of the corresponding permission group, and users can set different threshold values to filter out permissions in different permission groups. For example, a user is very concerned about leaking their location information; thus, they can deny permissions in the LOCATION group for which the Location risk score is greater than θ (e.g., 0.3).

Table 1. Deny ratio of the proposed approach

Threshold	Contacts B	Contacts M	Location B	Location M	Phone B	Phone M	SMS B	SMS M	Overall B	Overall M
0.05	3.5%	9.3%	1.9%	14.2%	7.9%	16.7%	8.4%	35.5%	4.2%	16.3%
0.1	11.0%	26%	5.5%	24.6%	13.5%	17.7%	16.9%	45.5%	9.8%	25.1%
0.15	16.7%	34.9%	7.5%	35.8%	24.6%	29.2%	25.6%	56.4%	15.6%	35.8%
0.2	18.3%	42.8%	18.2%	45.8%	28.8%	40.7%	37.1%	74.5%	22.0%	46.8%
0.25	23.6%	55.3%	24.1%	59.2%	32.7%	46.9%	43.8%	78.2%	27.3%	56.7%
0.3	28%	60.5%	30.6%	64.2%	41.1%	55.7%	48.1%	78.2%	33.4%	62.3%
0.4	42.3%	68.4%	44.3%	80.2%	50.3%	64.3%	56.9%	82.7%	45.9%	72.9%
0.5	52.3%	79.1%	48.8%	81%	54.7%	71.8%	63%	84.5%	52.2%	78.1%
0.6	65.5%	83.3%	64.5%	88.8%	54.7%	71.8%	69.3%	85.5%	62.7%	82.0%
0.7	70.1%	85.1%	72.5%	89.4%	75.6%	80.7%	82.3%	91.8%	73.2%	86.0%
0.8	81.8%	90.7%	78.3%	93.6%	80.6%	83%	84.2%	91.8%	80.3%	89.5%
0.9	93.2%	96.3%	80.8%	96.1%	90.5%	93.8%	84.2%	91.8%	87.1%	94.9%

To the best of our knowledge, no related study has examined permission recommendation in order to automatically provide a risk score relative to granting an app's permission prior to installation of the app. We compared the proposed approach to a previously reported approach [15] that related to malware detection. For each app category, the existing approach refers to any critical permission (i.e. a runtime permission in Android 6.0) requested by less than θ% of the apps in this category as a θ-Rare Critical Permission ($\theta - RCP$) for this category. Here, CRCP (θ) denotes the Category-based Rare Critical Permission signal. Any app that requests one of the $\theta - RCP$'s in its category triggers $CRCP(\theta)$. On the basis of the permission group mechanism, we set the θ range from 5% to 30% to evaluate the performance of denying permissions that trigger $CRCP(\theta)$. The CRCP performance is shown in Table 2.

The frequency of occurrence of permissions shown an obvious difference for benign apps. For example, the average frequency of occurrence of

Table 2. Deny ratio of CRCP

θ	Contacts B	Contacts M	Location B	Location M	Phone B	Phone M	SMS B	SMS M	Overall B	Overall M
5%	12.4%	19.5%	1%	8.4%	12.8%	22%	66.7%	84.5%	11.5%	23.5%
10%	26.9%	35.8%	2.4%	19.0%	16.2%	26.6%	86.2%	95.5%	18.5%	33.5%
15%	31.6%	46%	7.5%	35.8%	26.0%	28.5%	87.1%	95.5%	24.3%	42.4%
20%	39.8%	58.6%	8.0%	36.9%	43.0%	34.4%	95.8%	100%	31.6%	47.9%
25%	48.4%	74.4%	21.8%	57.3%	49.4%	42.0%	98.7%	100%	41.4%	61%
30%	63.5%	86.0%	34.1%	70.1%	63.0%	66.2%	100%	100%	54.1%	75.7%

ACCESS_FINE_LOCATION in all categories is 26.4%; however, the frequency of occurrence of READ_SMS in Communication is 18%, and the frequency of occurrence of READ_SMS in Communication is greatest in all categories. Therefore, when setting θ to deny permissions, it may perform well for some permission groups and not perform well for other permission groups. For example, when θ is 10%, the deny ratios of the Location group for benign and malicious apps are 2.4% and 19.0%, respectively; however, the deny ratios of the SMS group for benign and malicious apps are 86.2% and 95.5%, respectively. Even though the frequency of occurrence of permissions has an obvious difference in a permission group. As a result, this may not perform well for some permission groups. For example, when θ is 25%, the deny ratios of Phone for benign and malicious apps are 49.4% and 42.0%, respectively. There are two permissions in the Location group, and the average frequencies of occurrence are 26.4% and 27.6%. Therefore, in the Location group, the CPCR performance is comparable to the proposed approach. Thus, owing to the obvious difference relative to the frequency of occurrence of different permissions, the user cannot understand the risk of granting a permission using the frequency of occurrence of this permission for the given app category. The results show that the CRCP perform not well in permission recommendation.

6 Conclusion

We have proposed a method that combines Android malware detection and Android permission recommendation to provide users with additional information about apps to help make appropriate decisions about app installation and usage. We investigated the feasibility of using app permission requests, the intent-filters, the app's category and which permissions are requested by other apps in the same category to better inform users about whether to install a given app and the risk scores of granting each required permission. This risk score has a simple semantic meaning that is easily understood. The results of a malware detection experiment indicate that the proposed approach can extract features that we used from an Android app in 1 s and that our malware detection can detect 86.8% of malicious apps with only a few alarms (less than 1%). The results

of a permission recommendation experiment show that, when a user denies a permission for which the risk score of the corresponding permission group is greater than 0.75, 27.3% and 56.7% of permissions for benign and malicious apps will be denied, respectively.

Acknowledgment. This work is supported by the National Key R&D Plan of China under grant no. 2016YFB0800201, the Natural Science Foundation of China under grant no. 61070212 and 61572165, the State Key Program of Zhejiang Province Natural Science Foundation of China under grant no. LZ15F020003, the Key research and development plan project of Zhejiang Province under grant no. 2017C01065, the Key Lab of Information Network Security, Ministry of Public Security, under grant no. C16603.

References

1. Androgurad. https://github.com/androguard
2. Beautifulsoup. https://pypi.python.org/pypi/beautifulsoup4
3. Virustotal. https://www.virustotal.com/zh-cn/
4. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: AndroZoo: collecting millions of android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 468–471. IEEE (2016)
5. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: DREBIN: effective and explainable detection of android malware in your pocket. In: NDSS (2014)
6. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: user attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and Security, p. 3. ACM (2012)
7. Google: Android system permission 2015 (2015). https://developer.android.com/guide/topics/permissions/index.html
8. Ho, T.K.: Random decision forests. In: Proceedings of the Third International Conference on Document Analysis and Recognition, vol. 1, pp. 278–282. IEEE (1995)
9. Jha, A.K., Lee, W.J.: Analysis of permission-based security in android through policy expert, developer, and end user perspectives. J. UCS **22**(4), 459–474 (2016)
10. McAfee: Mcafee labs threats report April 2017 (2017). https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-mar-2017.pdf
11. Oglaza, A., Laborde, R., Benzekri, A., Barrère, F.: A recommender-based system for assisting non-technical users in managing android permissions. In: 2016 11th International Conference on Availability, Reliability and Security (ARES), pp. 1–9. IEEE (2016)
12. Rashidi, B., Fung, C., Bertino, E.: Android resource usage risk assessment using hidden Markov model and online learning. Comput. Secur. **65**, 90–107 (2017)
13. Rashidi, B., Fung, C., Vu, T.: Dude, ask the experts!: Android resource access permission recommendation with recdroid. In: IFIP/IEEE International Symposium on Integrated Network Management, pp. 296–304. IEEE (2015)
14. Rashidi, B., Fung, C., Nguyen, A., Vu, T.: Android permission recommendation using transitive Bayesian inference model. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016, Part I. LNCS, vol. 9878, pp. 477–497. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45744-4_24

15. Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Android permissions: a perspective combining risks and benefits. In: Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 13–22. ACM (2012)
16. Sokolova, K., Perez, C., Lemercier, M.: Android application classification and anomaly detection with graph-based permission patterns. Decis. Support. Syst. **93**, 62–76 (2017)
17. Taylor, V.F., Martinovic, I.: Starving permission-hungry android apps using SecuRank. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1850–1852. ACM (2016)
18. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: IEEE Symposium on Security and Privacy, pp. 95–109 (2012)
19. Zhu, H., Xiong, H., Ge, Y., Chen, E.: Mobile app recommendations with security and privacy awareness. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 951–960. ACM (2014)