Permission-based Analysis of Android Applications Using Categorization and Deep Learning Scheme

Hamidreza Alimardani[1,*], and *Mohammed* Nazeh[1]

[1] Centre of Postgraduate Studies, Limkokwing University of Creative Technology, Jalan Teknokrat 1/1, Cyberjaya, 63000 Cyberjaya, Selangor, Malaysia

Abstract. As mobile devices grow in popularity, they have become indispensable in people's daily lives, keeping us connected to social networks, breaking news, and the entire Internet. While there are multiple competing platforms, Google's Android is currently the most popular operating system for mobile devices. This popularity has drawn attention of hackers as well. Thus far, research works have analysed Android permissions individually, which makes analysis complex and time consuming. In this work, we propose categorizing Android permissions based on Google's recommendation and perform LSTM analysis on data. The used datasets are Drebin and AndroZoo, which are the most complete and well-respected among research community. The experiment results show that LSTM achieved 91% of true positive rate.

1 Introduction

As mobile devices grow in popularity, they have become indispensable in people's daily lives, keeping us connected to social networks, breaking news, and the entire Internet. While there are multiple competing platforms, Google's Android is currently the most popular operating system for mobile devices.

Google takes an open stance toward Android. The third-party applications are not rigorously scrutinized before they are made available to users on the Play Store. This open approach has led to security challenges and privacy concerns, and Google relies on community reviews (e.g., user ratings and application flagging) to mitigate security threats. Google can also remove any detected or reported malware from the Play Store and remotely from affected devices [1].

To better defend against malware, Google launched Bouncer [2], a server-side security service that performs a series of analyses to detect hidden, malicious behaviour when applications are uploaded to Google Play. However, it is found that Bouncer may be bypassed by sophisticated malware from knowledgeable adversaries [3]. More recently, Google launched a new application scanning service as an extension to Bouncer with the release of Android 4.2. This new security service is built into the platform to detect potentially malicious or harmful code when side-loading applications onto devices.

To implement rich features and achieve better user experience, applications frequently collect and compile sensitive data available on mobile devices. Many of these applications access users' personal information, such as geolocation, device identifiers and contact lists. Ideally, such accesses should be controlled to protect user privacy.

Although applications must ask for permissions to use sensitive information, it may not be clear to the end-user how such information is actually used after granting access. For instance, a photo-sharing application needs to access the network to transmit photos and acquire the user's location for geotagging the photos. In this example, the user needs to trust that the application only uses this privacy-sensitive information as indicated, which is not always the case. If the application is ad-supported, it may send the user's location along with other privacy-related data to advertisers' servers to display targeted ads. To mitigate this issue, recent legislation mandates that any application that collects personal data from users must conspicuously post a privacy policy describing how such information is collected, used and shared [4]. Unfortunately, privacy policies often contain broad, vague legal language, and can be inaccurate or out-of-date [5, 6]. Therefore, permission is our selected feature for this work.

The use of Android permission in this work differs from other works, since we group requested permissions based on Google's categories, as normal and dangerous. This way, analysis of data is faster and more effective. Additionally, this work analyses 100,000 Android

* Corresponding author: hra874@gmail.com

applications, which is more than other related works and makes the results more reliable.

2 Related works

We know that the Android operating system has a Linux core, from which it inherits important parts of the Linux security architecture. Prior to installation of an application, the Android provides a list of requested permissions to the user. Upon the permissions being granted, the application installs itself on the device. There are 130 official Android permissions [7]. Google categorizes them into four groups, namely, normal, dangerous, signature, and signatureOrSystem [8]. Researchers take different approaches in analysing Android permissions.

Android builds a part of its security on a "permission restricted access model" on sensitive sources (e.g., SD card, contacts). This means that applications, to gain access to such resources, should declare in the manifest the appropriate permissions, which users should grant during the installation. However, in such a model application might "manipulate" the requested permissions and gain access to private information, without users' consent at all.

A typical example is that of those applications that request more permissions than what they actually need, named as over-privileged [9, 10]. Such applications can be transformed silently into malware, whenever an operating system or an application update occurs. Furthermore, Android OS permission number expansion from its first version (100 permissions) to the latest version (170), as Figure 1 illustrates, is indeed making, in a way, larger the attack surface exposed to an adversary.

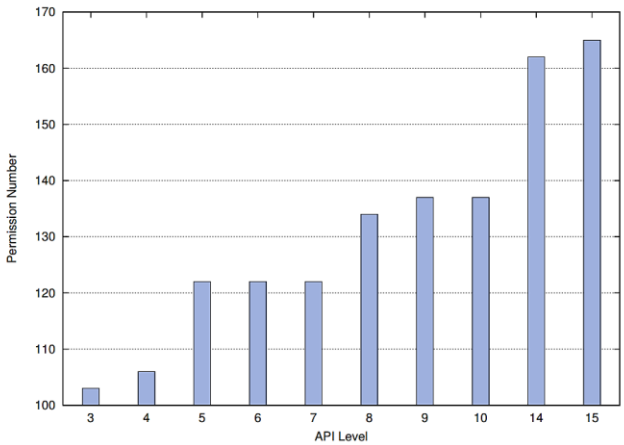

Fig. 1. Android permission evolution

RiskMon [11] introduces an automated service to assess the security and privacy risk of a given application taking into account legitimate users normal behaviour. RiskMon leverages on (a) machine learning and (b) trusted applications different run-time features to build user's legitimate model. RiskRanker [12] detects zero day related Android malware by analysing whether a particular application exhibits dangerous behaviour based on static analysis.

DroidAnalytics [13] develops a solution to scrutinize Android application at the byte code level, and generates the corresponding signatures that can be used by anti-virus software. In the same direction, Shahzad et al. rely on bi-grams sequences of op-codes retrofitting in machine learning classifiers to detect malware [14], while Permlyzer analyses application's permission usage based on both static and dynamic analysis [15]. Barrera et al. accomplish a permission analysis based on Self-Organising Map (SOM) [16], while Xuetao et al. study Android's permissions evolution [17].

Other solutions such as Whyper [18] reason about the necessity of requesting an access to specific permission. To do so, Whyper relies on Natural Language Processing (NLP) by extracting information from the keywords and description defined in the application. Similarly, TatWing et al. build a permission based abnormal model leveraging on application description and its permission [19].

Yajin et al. introduce a tool for the systematic study of applications that might passively leak private information, due to vulnerabilities stemming from built-in Android components, such as read/write operations to content provider [20].

Applications are statically analysed to identify such data flows. Analogously, Liang et al. propose a malware detection engine that relies on the semantic analysis of an examined application [21]. Sbirlea et al. develop techniques for statically detecting Android application vulnerabilities to attacks aiming at obtaining unauthorized access to permission-protected information [22].

Thus, we believe that permissions and other related information (i.e., APIs) residing in applications should be considered as an important information source for detecting malicious applications. In this work, we introduce an analysis approach to enhance the performance of anomaly machine learning based techniques used to assess whether an application is malicious or not, based on applications permission related information, elaborating on previous research works [23-25] in the direction of achieving higher accuracy. To do so, we group permissions based on Google's documentation[a].

3 Dataset analysis

This section consists of introducing our dataset and its descriptions. In addition, we present some insights on the dataset using statistical analysis.

[a]http://developer.android.com/guide/topics/manifest/permission-element.html

3.1. Datasets

Each investigation requires a dataset in view of which the creators assess their proposed framework. Android malware is a moderately new research zone. The principal Android malware was found in 2010 [26]. At first, scientists did not have a strong and standard dataset to work with.

The Drebin data sample was published in 2014 by Arp et al. [24]. It is a collection of 5,560 Android malware categorized into 179 different families. It was collected between August 2010 and October 2012. The authors scanned the Drebin with antivirus applications. They report that while the best scanners detected over 90% of the malware, others detected less than 10% of the data sample. The Drebin was well-accepted among researchers [27, 28]. We acquired it for this study.

Not at all like the specified data samples, AndroZoo is a growing collection of Android applications from a few sources, including the official Google Play. As of composing this work, AndroZoo contains more than five million Android applications. Not exclusively does this data sample contain Android malware, yet it contains clean applications too [29]. Crawling different sources began in late 2011 and has proceeded with from that point onward.

The 14 sources incorporate Google Play, Anzhi, AppChina, 1mobile, AnnGeeks, Slideme, HiApk, ProAndroid, and so on. The AndroZoo sends all the downloaded applications to the VirusTotal for checking. The quantity of antivirus programming that identify an application as malignant is put away in the metadata document, as vt_detection.

The metadata record is accessible on the AndroZoo site and is refreshed consistently. Thus, if vt_detection is zero, at that point the application is perfect. Else, it is considered as malware. This element enables analysts to utilize AndroZoo as a malware store, as well as a clean application archive. Thus, we chose this data sample and the Drebin for our experiment.

3.2. Data exploration

In this section, we analyze our dataset in terms of static analysis and in particular, permissions. Permissions are classified according to protection levels. The purpose of a protection level is twofold. First, it characterizes the general risk that is implied by the respective permission. Second, it determines how the Android platform handles the process when applications request the respective permission.

In this work, we use four levels to categorize each permission into respective group. For each application, we determine requested permissions to be in each of the four groups. Then, we analyze the application to identify it as clear or malware. This contribution makes analysis of Android applications faster, since we just deal with four groups of permission levels, versus all 130 Android permissions.

Based on our analysis, the applications requested 6,079 permissions, which is an average of 5.5 permissions per application. However, this set includes many duplicates, meaning that many applications tend to request similar permissions. In total, the installed applications requested 283 different permissions.

However, more than 40% of them were only requested by just one of the installed applications at a time. This is due to the fact that many permissions are vendor respectively application specific. For example, the permission com.symantec.permission.ACCESS_NORTON_SECURITY was only requested by a single app in the whole dataset.

In contrast to those permissions that are only requested once, some permissions are requested far more often compared to others. The ten most frequently requested permissions are listed in Table 1.

Table 1. Top Ten Most Frequently Requested Permissions in the Dataset

Permission	Frequency (%)
INTERNET	57
ACCESS_NETWORD_STATE	47
WRITE_EXTERNAL_STORAGE	45
WAKE_LOCK	27
READ_PHONE_STATE	23
VIBRATE	22
ACCESS_WIFI_STATE	20
ACCESS_FINE_LOCATION	17
ACCESS_COARSE_LOCATION	15
BLUETOOTH	13

As expected, the INTERNET permission is requested most often. An interesting observation could be made in terms of requested permissions and their protection levels. Most of the requested permissions have a protection level of “dangerous”, indicating that they are potentially harmful.

4. Experimentation

This section focuses on performing experiment and evaluation of the proposed method. First, the deep learning scheme is discussed. Then, results of the experiment is presented, and comparison with related works is conducted.

4.1. Deep learning Scheme

Since this work proposes usage of deep learning scheme in conducting experiments, it is beneficial to explain

deep learning history, its structure, and different available algorithms.

The recurrent neural networks and LSTM are proposed to deal with series of related data. One of the appeals of RNNs is the idea that they are able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame [30].

General and basic structure of deep learning scheme is shown in Figure 2. It consists of input layer, which is input data to algorithms. There are hidden layers in which the algorithm makes numerous mathematical calculations. Finally, the output layer is results of the calculations of the algorithms.

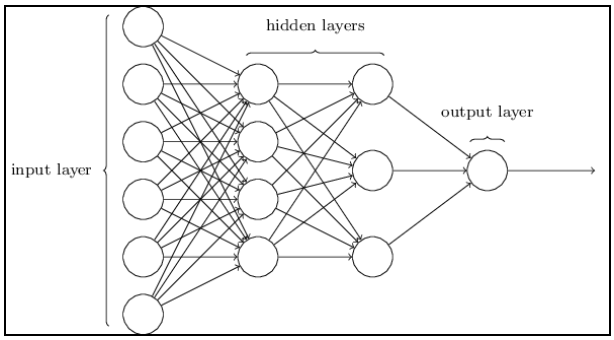

Fig. 2. Structure of Deep Learning Scheme

As stated before, the concept of deep learning has been around for decades. However, they gained momentum in recent years, for two reasons. First, availability of massive data. Deep learning algorithms work well when dealing with massive amount of data, such as credit card transactions, geospatial data. The more data is available, the better they are trained. In case of this study, we gather network traffic and permission of millions of applications, which is considered good input for deep learning algorithms.

The second reason is availability of computing power. In recent years, appearance of the cloud computing has dramatically increased the computing power. It is possible to rent up to 64 cores of CPU from Google Cloud or Amazon AWS services at affordable price. The deep learning algorithms are able to process terabytes of data using massive computing power to achieve high accuracy.

In this work, we experiment with two types of deep learning algorithms. The classic deep neural network (DNN) and the recurrent neural network (RNN). The DNN follows the architecture depicted in Figure 4.1, by having input, hidden, and output layers. The hidden layer is where the calculations happen. Each neural network involves the following functions and calculations.

- The first step in a neural network is the forward propagation in which the inputs are propagated across the layers. In addition, the network predicts the output based on inputs. Based on the explanations, the following functions are defined in the forward propagation.

$$z_1 = xW_1 + b_1 \quad (1)$$

$$a_1 = \tanh(z_1) \quad (2)$$

$$z_2 = a_1 W_2 + b_2 \quad (3)$$

$$y_2 = softmax(z_2) \quad (4)$$

The equations z_1 and z_2 are functions that take x as input and use W and b as weight and bias respectively. The $tanh$ is an activation function that takes z_1 as input and passes the result to the next layer. In the output layer, the $softmax$ function is used to calculate y_2 that is the prediction of the neural network.

4.2. Results and Discussion

This experiment starts with extracting permissions from Android applications. This process is done using APKtools. The permissions are extracted for both normal and malicious applications. The end result is a dataset of all permissions labelled as normal or malicious.

- The next stage is to associate permissions to normal or dangerous. This way, the processing time and, consequently, the detection time is faster. This method is contribution of this study.

The data is divided into training set and test set. The training set is used to train an algorithm. The test set is used to test the learned algorithm. Researchers commonly dedicate 70% of data to training and 30% to testing. We followed the same procedure to employ standard methods. Then, the training data is fed to deep learning algorithms.

Table 2 shows results of the experiment, in form of true positive rate (TPR), false positive rate (FPR), and loss.

Table 2. Result of the Experiment on Permissions

Algorithm	TPR	FPR	Loss
DNN	82.17%	17.83%	0.1533
LSTM	91%	9%	0.0048

The table shows loss as 0.0048. The loss is calculated as difference between actual value and predicted value by the algorithm. Its optimum value is zero, and as it is closer to zero, it is considered a good performance.

The results show that the LSTM algorithm performed better than DNN. It is because of the structure of LSTM in which decision making at each layer involves data from previous layer. This way, the data is treated more

analogous to human brain in which decisions are based on data from early encounters.

In order to establish contribution of this work, it is necessary to compare the results to most related research works. Table 3 shows the comparison of recent works with this study.

Table 3. Comparison with Related Works

Reference	Results	Dataset Size
[31]	89.91%	2,130
[32]	90.26%	1,621
[33]	87.40%	9,512
[34]	89.04%	2,808
This study	91%	100,000

As it can be seen from Table 3, this study achieved high detection rate compared to other related works. Although the difference between this study and other related works looks small, it is necessary to consider dataset size as an important factor.

As dataset size grows, the results of experiment look generalized and more promising, since it includes more clean and malicious data. This work used 50,000 clean applications and 50,000 malicious application. It provides more general results compare to other related works.

On the other hand, this work focuses on complexity of data analysis and detection. In order to analyze and develop a detection method for 100,000 application, we proposed categorizing Android permission based on the Google suggestions. This way, the effectiveness of detection method is better, as the results in Table 3 present.

5. Conclusion

In this work, we proposed an analysis methods for Android applications based on permissions. This method uses categorization suggested by Google, to facilitate analysis of Android application data. This way, it is easier to analyze 100,000 application, since we categorize their permissions to normal and dangerous. The dataset was gathered from Drebin and AndrZoo datasets, and both clean and malicious applications are included. The experiment was performed using deep learning scheme, and particularly the LSTM algorithm, since it is capable of making decisions based on previous data much like human brain. The results show that the experiment achieved 91% of true positive rate.

References

[1] R. Cannings. (2010, 1st September). *Exercising Our Remote Application Removal Feature*. Available: https://android-developers.googleblog.com/2010/06/exercising-our-remote-application.html

[2] H. Lockheimer. (2012, 1st September). *Android and Security*. Available: http://googlemobile.blogspot.com/2012/02/android-and-security.html,

[3] J. Oberheide and C. Miller, "Dissecting the android bouncer," in *SummerCon*, New York, USA, 2012.

[4] K. D. Harris. (2012, 1st September). *Secures Global Agreement to Strengthen Privacy Protections for Users of Mobile Applications*. Available: https://oag.ca.gov/news/press-releases/attorney-general-kamala-d-harris-secures-global-agreement-strengthen-privacy

[5] L. Xu, *Techniques and Tools for Analyzing and Understanding Android Applications*: University of California, Davis, 2013.

[6] A. Tilton. (2012, 1st September). *VLINGO MAKES OFFICIAL STATEMENT: SUCCESS AT CUSTOMERS' EXPENSE*. Available: https://www.androidpit.com/vlingo-user-data-leak

[7] V. Moonsamy, J. Rong, and S. Liu, "Mining permission patterns for contrasting clean and malicious android applications," *Future Generation Computer Systems,* vol. 36, pp. 122–132, 2013.

[8] Google. (2014, 1st April). *permission*. Available: http://developer.android.com/guide/topics/manifest/permission-element.html

[9] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *18th ACM Conference on Computer and Communications Security*, Chicago, Illinois, USA, 2011, pp. 627-638.

[10] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, Chicago, Illinois, USA, 2011, pp. 3-14.

[11] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Riskmon: Continuous and automated risk assessment of mobile applications," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, 2014, pp. 99-110.

[12] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," in *10th International Conference on Mobile Systems, Applications, and*

Services, Low Wood Bay, Lake District, UK, 2012, pp. 281-294.

[13] M. Zheng, M. Sun, and J. Lui, "DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware," in *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Melbourne, Australia, 2013, pp. 163 - 171.

[14] R. K. Shahzad and N. Lavesson, "Veto-based malware detection," in *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, 2012, pp. 47-54.

[15] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in android applications," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, 2013, pp. 400-410.

[16] D. Barrera, H. G. Kayacik, P. C. v. Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *17th ACM Conference on Computer and Communications Security*, Chicago, Illinois, USA, 2010, pp. 73-84.

[17] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 31-40.

[18] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: towards automating risk assessment of mobile applications," in *22nd USENIX Security Symposium*, Washington, D.C, USA, 2013, pp. 527-542.

[19] J. Zhu, Z. Guan, Y. Yang, L. Yu, H. Sun, and Z. Chen, "Permission-based abnormal application detection for android," in *International Conference on Information and Communications Security*, 2012, pp. 228-239.

[20] Y. Z. X. Jiang and Z. Xuxian, "Detecting passive content leaks and pollution in android applications," in *Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)*, 2013.

[21] S. Liang, M. Might, and D. Van Horn, "Anadroid: Malware analysis of android with user-supplied predicates," *arXiv preprint arXiv:1311.4198,* 2013.

[22] D. Sbîrlea, M. G. Burke, S. Guarnieri, M. Pistoia, and V. Sarkar, "Automatic detection of inter-application permission leaks in Android applications," *IBM Journal of Research and Development,* vol. 57, pp. 10: 1-10: 12, 2013.

[23] W. Dong-Jie, M. Ching-Hao, W. Te-En, L. Hahn-Ming, and W. Kuo-Ping, "DroidMat: Android Malware Detection through Manifest and API Calls Tracing," in *Seventh Asia Joint Conference on Information Security (Asia JCIS)*, Tokyo, Japan, 2012, pp. 62-69.

[24] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *2014 Network and Distributed System Security (NDSS) Symposium*, San Diego, USA, 2014, pp. 1-15.

[25] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," in *9th International Conference on Security and Privacy in Communication Networks*, Sydney, Australia, 2013, pp. 86-103.

[26] Lookout. (2010, 21st April). *Security Alert: Geinimi, Sophisticated New Android Trojan Found in Wild*. Available: https://blog.lookout.com/blog/2010/12/29/geinimi_trojan/

[27] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder*, et al.*, "DroidScribe: Classifying Android Malware Based on Runtime Behavior," in *Mobile Security Technologies (MoST 2016)*, San Jose, USA, 2016, pp. 1-12.

[28] M. Varsha, P. Vinod, and K. Dhanya, "Identification of malicious android app using manifest and opcode features," *Journal of Computer Virology and Hacking Techniques,* pp. 1-14, 2016.

[29] K. Allix, T. F. Bissyand, J. Klein, and Y. L. Traon, "AndroZoo: collecting millions of Android apps for the research community," in *13th International Conference on Mining Software Repositories*, Austin, USA, 2016, pp. 468-471.

[30] A. Mikami, "Long Short-Term Memory Recurrent Neural Network Architectures for Generating Music and Japanese Lyrics," Ph.D., Computer Science Department, Boston College, 2016.

[31] H.-J. Zhu, T.-H. Jiang, B. Ma, Z.-H. You, W.-L. Shi, and L. Cheng, "HEMD: a highly efficient random forest-based malware detection framework for Android," *Neural Computing and Applications,* pp. 1-9, 2017.

[32] C. Bae and S. Shin, "A collaborative approach on host and network level android malware detection," *Security and Communication Networks,* vol. 9, pp. 5639-5650, 2016.

[33] K. Sokolova, C. Perez, and M. Lemercier, "Android application classification and anomaly detection with graph-based permission patterns," *Decision Support Systems,* vol. 93, pp. 62-76, 2017.

[34] B. Sanz, I. Santos, X. Ugarte-Pedrero, C. Laorden, J. Nieves, and P. G. Bringas, "Instance-based Anomaly Method for Android Malware Detection," in *10th*

International Conference on Security and Cryptography,
Reykjavík, Iceland, 2013, pp. 387-394.