Automated Rogue Behavior Detection for Android Applications

Shuangmin Zhang, Ruixuan Li✉, Junwei Tang, Xiwu Gu
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China
E-mail: {shmzhang, rxli, jwtang, guxiwu}@hust.edu.cn

***Abstract*—There are a large number of third-party application markets that provide application download services for Android users. In order to improve users' satisfaction, major application markets urgently need an automated solution to avoid some rogue behaviors that affect users' experience, such as rogue advertisements that induce users to click, rogue pop-up boxes that cannot be closed normally, and rogue floating windows that affect users' experience. To address such problems, we propose a rogue behavior detection framework. We use the object detection approach to identify advertisements in screen views, the random forest method to identify the pop-up views, and then combine image analysis, natural language processing and heuristic methods to detect rogue behaviors. The proposed framework can also record evidences of rogue behaviors in applications, so that application markets can ask developers to rectify. The experimental results show that the precision of rogue application detection reached 96.7% and the recall reached 90.1%.**

I. Introduction

In the last decade, it has entered the era of mobile devices. According to the mobile operating system market share in March 2020, Android system has the highest share at 72.26% [1]. The huge user base of the Android system has also spawned a large number of feature-rich Android applications. Users with high security awareness generally choose to download applications from large application markets, because most of them perform virus detection before allowing the applications to be online. However, there are still a large number of problematic applications that can cause trouble to users in application markets.

Research on Android applications mainly focuses on safety issues that pose a significant risk, such as malware detection [2], privacy leak detection [3] and ad fraud detection [4]. Little attention has been paid to user experience of using Android applications. For the detection of rogue ads (a kind of rogue behavior), the detection of ads in the screen views is a key point. Ad detection methods mainly adopt traffic analysis method [4] [5]. These methods can only identify the screen views containing ads, but cannot give the specific location of the ads. Hence, an efficient method is needed to detect the location of the ads in screen views. In order to improve users' experience and satisfaction, an automated solution is urgently needed to identify rogue behaviors that affect users' experience in Android applications, including some rogue behaviors that only affect users' experience without causing major security issues.

DOI reference number: 10.18293/SEKE2020-089

In this paper, we propose a framework to find rogue behaviors that affect users' experience. The contributions of our work are as follows. We propose an object detection approach based on deep learning approach to detect ads in screen views. It can be used to serve rogue ad detection. We present a classifier that can divide the screen views into those containing and not containing pop-up boxes based on random forest. It can be used to serve rogue pop-up box and rogue ad detection. We implement the rogue ad detection module based on natural language processing and heuristic rules, the rogue pop-up box detection module based on heuristic rules, and the rogue floating window detection module based on image analysis and heuristic rules.

II. Definition and Categories of Rogue Behaviors

A. Definition of Rogue Behavior

Rogue Behavior: A kind of application behavior that indirectly affects the user's mobile device, making the user unable to use the mobile device conveniently, and bringing potential threat to the user's mobile device. It has no direct damage to the system after the execution, nor causing the infringement of user's personal information and fees [6].

B. Categories of Rogue Behavior

1) Rogue Ad: **Ads with contents that induce users to click.** Some ads often display some inductive information to induce users to click. As shown in Fig. 1(a), the bottom of the left screen view is an ad. The "Clear Memory" and "Close" in the ad view are fake buttons. Such ads can easily mislead users to think that their devices need to clear memory. When a user clicks on the green fake button, the installation package starts to be downloaded. **Ads that overlay clickable components.** To improve the click rate of ads, applications may pop up an ad pop-up box immediately after a normal pop-up box. The original intention of the user is to click the button of the normal pop-up box, but at this time, the ad pop-up box pops out suddenly, and the user is highly likely to click on the ad by mistake, as shown in Fig. 1(b). **Ads that appear**

before application exit. After a user's click on the exit key, the application will usually pop up a prompt box to confirm if the user really wants to exit. When the user clicks the "Exit" button, the user's intention is to leave the application. However, some applications show an ad after clicking the "Exit" button, as shown in Fig. 1(c).

2) Rogue Pop-up Box: Some developers may use pop-up box inappropriately, causing trouble to users. As shown in Fig. 1(d), the prompt pop-up box of the application only provides one button means "update", without the button to close the pop-up box. Through manual testing, even clicking the back key of Android device still cannot exit, which will cause great trouble to users.

3) Rogue Floating Window: Some applications misuse the floating window to place ads for profit. Because a floating window is floating on the normal application screen view, it will cover part of the contents in the application screen view, which will affect users' experience. Android application developers generally like to design the floating window used for advertising purposes as red-envelope-style to attract users to click on the ad, as shown in Fig. 1(e).

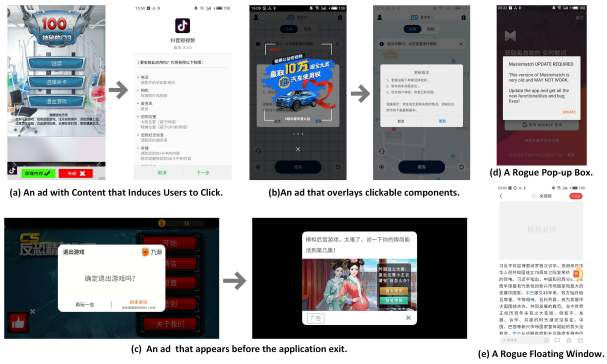

Fig. 1. Rogue Behaviors.

III. METHOD

The overall scheme of rogue behavior detection for Android applications is shown in Fig. 2. The Android application traversal module automatically runs applications and records information based on Appium[1]. The information of the screen views and the events that caused the views' transition are recorded during the traversal process, as shown in Fig. 3. The main purpose of the Android application screen view classification module is to classify the application screen views. This module will train a deep learning-based object detection model and a random forest-based classifier. The object detection model is used to identify ads in the screen views. The random forest classifier is used to divide the screen views into views containing pop-up box and those not containing pop-up box. The rogue ad detection module, the rogue pop-up box detection module and the rogue floating window detection module are used to identify rogue behaviors in Android applications.

[1]Appium. http://appium.io/

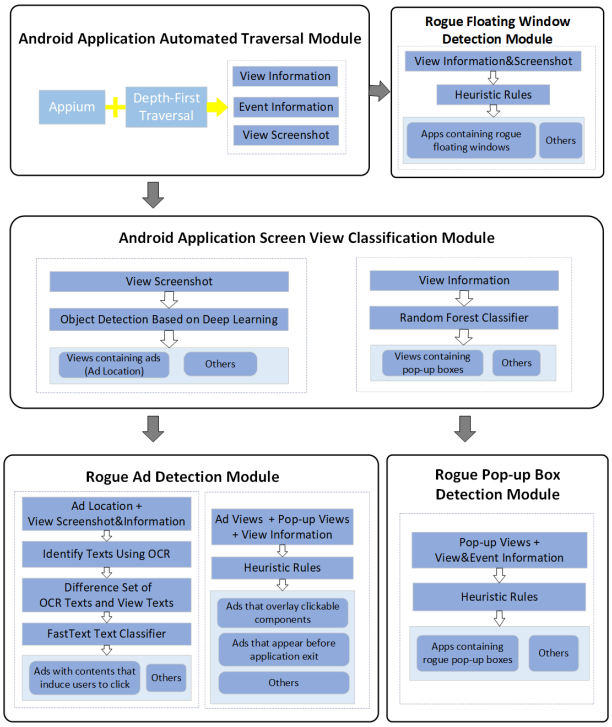

Fig. 2. Overall Scheme of Rogue Behavior Detection.

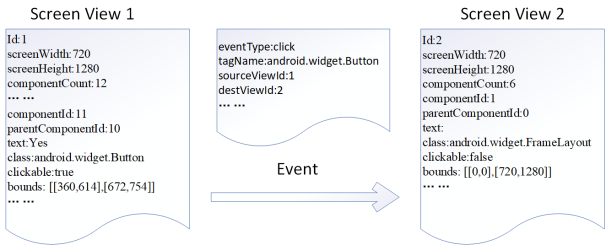

Fig. 3. Recorded Information.

A. Ad Detection

We design a lightweight network based on RetinaNet [7], as shown in Fig. 4. The left is backbone, the medium is Feature Pyramid Networks (FPN) structure [8], the right are classification and bound regression subnetworks. Each of L1 and L2 is a convolution. L3, L4 and L5 are made up of 4, 8 and 4 cells respectively, and each cell is a residual module composed of three deep separable convolutions [9]. By FPN structure, features of different scales are fused to combine the high-level information with the low-level information. Each of the classification and bound regression branches contains two residual modules consisting of two convolutions. Finally, a convolution is added as the output layer. We design 24 anchors of different scales and aspect ratios at each location. For the classification branch, each anchor is classified as an ad or non-ad, so the number of the output channels is 48. For the bound regression branch, every four numbers make up the upper-left and lower-right coordinates of the border, so the number of the output channels is 96.

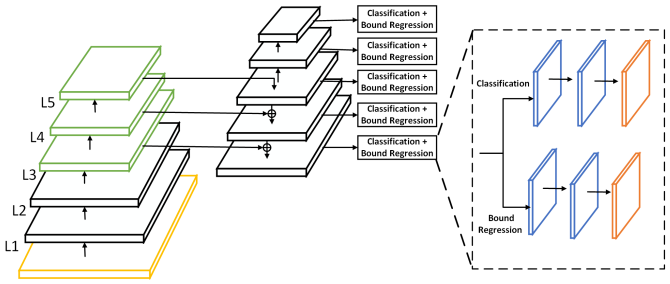

Fig. 4. Ad Detection Model.

B. Pop-up View and Non-Popup View Classification

We divide the Android application screen views into views containing pop-up box and those not containing pop-up box. We analyze the information of 1,000 Android applications, and summarize a series of features. These features include some 0-1 features as well as continuous value features. These features can be roughly divided into the following categories: component size features, component quantity features, component location features, keyword features and the proportions of components with specific attributes. The extracted features are suitable for decision tree classifier, so we use a random forest classifier based on decision tree. The random forest classifier has higher accuracy by combining the results of multiple base classifiers.

C. Heuristics-based Detection of Rogue Behaviors

1) Rogue Ad Detection: **Detection of ads with contents that induce users to click.** We first get the ad images cropped from the screenshots according to the ad detection result, and then we use the Optical Character Recognition (OCR) API provided by Baidu[2] to extract the texts in ad images. Then, we analyze the semantics of the advertising texts. In order to remove the interference of the texts outside the ad, we adopt difference set of OCR texts and view texts. Due to Chinese has multiple expressions of the same meaning, we need to understand the degree of similarity between different words. We adopt neural network word vectors for word representation, so we can identify other inductive sentences of similar meaning. The detection can be considered as an advertising text classification task. We adopt FastText [10] as the classification model, because it is based on word vectors and maintains high accuracy while training and testing time are greatly reduced. **Detection of ads that overlay clickable components.** In order to overlay the clickable components, the ad view should have an ad that occupies more than 30% of the screen space. The previous screen view of the ad view is a pop-up view that contains at least one clickable component. **Detection of ads that appear before the application exits.** Generally, before exiting the application, users will be asked if they really want to quit. There are two situations at this time. One is that the user clicks the ”Exit” button, then will lead to exit. In normal circumstances, this prompt view should be the last view in the traversal process. If an ad view appears after that, this ad is a rogue ad. Another situation is that the user clicks the ”Cancel” button, then will go back to the previous view. Because the previous view may contain ads, the ads that appear at this time should not be considered as rogue ads.

2) Rogue Pop-up Box Detection: The detection of pop-up boxes that force applications to upgrade requires the cooperation of the automated traversal module. In automated traversal process, we remove events of update buttons. If a pop-up view contains ”update”, ”upgrade”, ”download” or other texts with similar meanings, and the event executed on this view is ”Back” type event, and the ID of the next view is equal to this view, the view is considered to have a pop-up box that forces users to update the application.

3) Rogue Floating Window Detection: The most intuitive feature of rogue floating windows is that they have red-envelope-style small icons that are approximately square. We consider components with an aspect ratio between 0.8 and 1.2 to be regarded as approximate square, and extract the pixel value of each pixel for such components. The screen views that contain such components with visual red pixels accounting for more than 20% and visual yellow pixels accounting for more than 3% are considered as those likely to have red-envelope-style floating windows. It is found that rogue floating windows often exist in multiple screen views. Therefore, we further consider the context information of the current screen view. If one of the surrounding screen views also contains red-envelope-style icon, we believe it contains rogue floating window.

IV. EXPERIMENTAL EVALUATION

A. Dataset

The dataset used in our work is from the application market named ”Mango Download Station”[3]. We collected a total of 4,000 applications. For Android application ad detection, 500 applications were randomly selected as the dataset. For the classification of views containing pop-up box and views not containing pop-up box, 1,000 applications were randomly selected as the dataset. For the detection of rogue behaviors, the remaining 3,000 applications were selected as the test set, which did not include the training set used for ad detection and the dataset used for the classification of screen views.

B. Result of Ad Detection

We use 300 applications as the training set, and the remaining 200 applications as the validation set. The validation results are shown in Fig. 5. Fig. 5(a) shows the precision curve. Fig. 5(b) shows the recall curve. Fig. 5(c) shows the F1 curve. Fig. 5(d) shows the precision-recall curve. The abscissa represents the recall under different score thresholds, the ordinate represents the corresponding accuracy, and the area under the curve represents the Average Precision (AP). Here, the score threshold means that during evaluation, a predicted box with a score lower than the threshold will be

[2]Baidu OCR. https://ai.baidu.com/tech/ocr/general

[3]Mango Download Station. http://www.90370.com

thought as a negative case, otherwise as a positive case. The evaluation was conducted with an Intersection Over Union (IOU) of 0.75, which means that the predicted box and the ground truth box are considered to be matched if their IOU is greater than 0.75. Based on the results and combined with the need to identify ads in our work, the score threshold with higher recall rate is selected under the condition that the precision and F1 value are not too low. The score threshold we use for ad detection is 0.42. The AP is 0.808 on the validation set. The effect of ad detection is good enough to be used in our work to serve rogue ad detection.

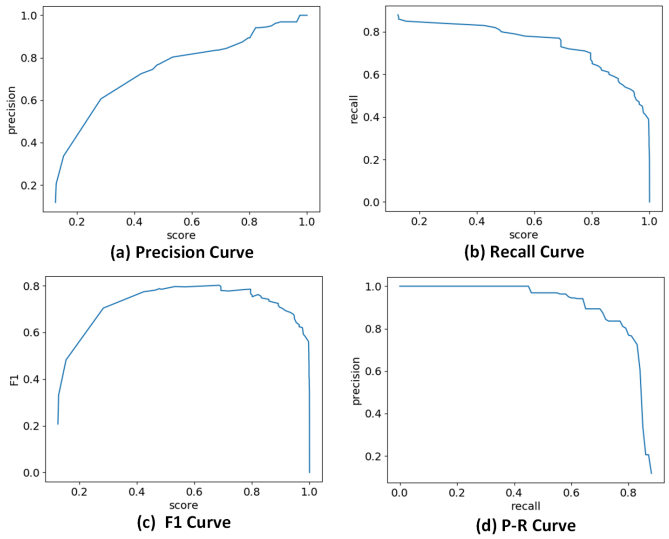

Fig. 5. Precision, Recall, F1 and P-R Curve.

C. Result of Pop-up View and Non-Popup View Classification

In this experiment, 1,000 applications were randomly selected as the dataset. For the evaluation of the classifier, the 10-fold cross-validation method is used. We randomly divided the data into 10 parts, one of which was taken as the validation set and the other 9 parts as the training set. The experiment was conducted for 10 times, and the arithmetic mean value of the results was taken as the final result. The results show that the average accuracy of the classification is 98%. The effect of the classifier is good enough to be used in our work to serve rogue pop-up box detection and rogue ad detection.

D. Result of Rogue Behavior Detection

By manually marking the data set, 131 of the 3000 applications in the test set were marked as having rogue behaviors. Using our method to test the 3,000 applications, the results of rogue behavior detection are shown in Table I. The results show that 122 applications have rogue behaviors, of which 118 applications do have rogue behaviors, and 4 do not. 13 applications with rogue behaviors were not identified. In summary, the precision of rogue behavior detection in our work is 96.7%, and the recall is 90.1%.

TABLE I
RESULT OF ROGUE BEHAVIOR DETECTION.

		Predict Labels	
		Rogue Apps	Normal Apps
Actual Labels	Rogue Apps	118	13
	Normal Apps	4	2865

V. CONCLUSION AND FUTURE WORK

In this paper, we propose an effective rogue behavior detection framework for Android applications. We implement ad detection, screen view classifier, rogue ad detection, rogue pop-up box detection, and rogue floating window detection. Using the methods we proposed, 118 Android applications containing rogue behaviors were found in 3,000 applications. The results show that the precision of rogue application detection reached 96.7% and the recall reached 90.1%.

We only identify five types of rogue behaviors, and other rogue behaviors may exist in reality. Meanwhile, new rogue behavior is still emerging. The methods proposed in this paper is scalable, and this study can be easily expanded to new rogue behaviors.

VI. ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program of China under grants 2016YFB0800402 and 2016QY01W0202, National Natural Science Foundation of China under grants U1836204 and U1936108.

REFERENCES

[1] statcounter, "Mobile operating system market share worldwide," https://gs.statcounter.com/os-market-share/mobile/worldwide, 2020.

[2] Y. S. Sun, C.-C. Chen, S.-W. Hsiao, and M. C. Chen, "Antsdroid: automatic malware family behaviour generation and analysis for android apps," in *Australasian Conference on Information Security and Privacy*. Springer, 2018, pp. 796–804.

[3] 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 (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[4] F. Dong, H. Wang, L. Li, Y. Guo, T. F. Bissyandé, T. Liu, G. Xu, and J. Klein, "Frauddroid: Automated ad fraud detection for android apps," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 257–268.

[5] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014, pp. 123–134.

[6] C. C. S. Association, "Description format for mobile internet malicious code," http://www.ptsn.net.cn/standard/std_query/show-yd-3983-1.htm, 2013.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[8] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2det: A single-shot object detector based on multi-level feature pyramid network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9259–9266.

[9] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[10] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.