Statistical Feature-Based Android Malware Detection Method (Postprint)

Authors: cold wave, Jianbin Li

Date: 2018-05-24T00:00:00+00:00

Abstract

To address the problem of neglecting the statistical significance of features in Android malware detection, we propose a detection method for Android malicious applications based on statistical features. This method extracts statistical features of applications as the training dataset and utilizes a clustering algorithm to preprocess the malicious dataset, thereby mitigating the influence of individual differences on experimental results. Furthermore, the method establishes detection models by combining features with various machine learning algorithms (e.g., linear regression, neural networks, etc.). The accuracies of the two models proposed by this method both exceed 95%, and the detection time is substantially reduced compared to comparative experiments. Experimental results indicate that statistical features of applications can effectively distinguish benign from malicious applications, and that data preprocessing via clustering algorithms can enhance detection accuracy.

Full Text

Preamble

Journal: Computer Application Research (ChinaXiv Cooperative Journal)
Article URL: http://www.arocmag.com/article/02-2019-09-041.html
Title: Android Malicious Application Detection Method Based on Statistical Features

Authors: Leng Boa, Li Jianbinb†
Affiliations:
a School of Information Science & Engineering,
b Information Security & Big Data Research Institute,
Central South University, Changsha 410083, China

Abstract

Aiming at the problem of ignoring the statistical significance of features in Android malicious application detection, this paper proposes a detection method based on statistical features. This method extracts statistical characteristics from applications as training data and employs clustering algorithms to preprocess malicious datasets, thereby reducing the impact of individual differences on experimental results. Furthermore, the method combines these features with multiple machine learning algorithms (such as linear regression, neural networks, etc.) to establish detection models. The accuracy of both proposed models exceeds 95%, with detection time substantially reduced compared to baseline experiments. Experimental results demonstrate that statistical characteristics of applications can effectively distinguish between benign and malicious applications, and that preprocessing data through clustering algorithms can improve detection accuracy.

Keywords: statistical features; machine learning; individual differences; malware detection

---

0 Introduction

According to International Data Corporation (IDC) [1], Android smartphones dominated the global smartphone market share from Q1 2016 to Q1 2017, holding an overwhelming advantage over other platforms. The rapid proliferation of smartphones has fundamentally transformed daily life. Smartphones are no longer mere communication devices; they enable web browsing, video streaming, shopping, chatting, office work, and more. Simultaneously, smartphones have become microcosms of user information, containing nearly all personal data such as identity information, banking details, and contact lists. The consequences of loss or theft by malicious actors would be catastrophic. Concurrently, as Android devices become increasingly ubiquitous, convenient network access and high-value personal information have attracted considerable interest from malware developers. 360 Security Center [2] reports that the Android platform sees an average of 21,000 new malicious applications daily. Therefore, detecting Android malware and preventing user harm is both urgent and highly challenging.

Current research on Android application detection primarily focuses on combining features with machine learning algorithms to verify application maliciousness. By analyzing Android application source code or binary information, researchers have proposed detection models trained on permission features [3], component features [4], dex features [5], and smali features [6]. Permission and component features are both derived from the AndroidManifest.xml file. Application-requested permissions and components are contained within specific tags that the Android system reads and configures during installation. Dex features refer to characteristics parsed from the dex files extracted after decompressing applications. Dex files are Dalvik virtual machine executables with a

fixed format. Smali features originate from smali files obtained through application decompilation. Detection methods built upon these static features can accurately identify malicious applications.

However, these methods focus on understanding the semantic meaning of each feature while neglecting individual particularities. Moreover, detection time for malicious applications has not received sufficient attention. Therefore, this paper proposes an Android malicious application detection method based on statistical features, utilizing permissions, components, dex information, and smali information combined with multiple machine learning algorithms (such as SVR, MLP, etc.) for malware detection. The method comprises two models. Each model analyzes and extracts statistical features to minimize the impact of individual differences on application detection. Model 1 aims for faster detection, while Model 2 provides higher detection precision. The contributions of this paper are: (a) analyzing Android applications from a novel perspective by extracting statistical features from apk code rather than focusing on feature semantics; (b) pioneering the use of clustering algorithms to reduce individual difference effects and selecting the clustering result with minimal volatility across cluster sizes to mitigate random clustering impacts; (c) employing a dual-layer learning model to achieve better accuracy; and (d) providing two models for different scenarios—Model 1 for faster detection and Model 2 for more precise detection.

Static analysis methods offer fast detection rates and high accuracy but cannot detect novel malware. Dynamic detection methods can identify new malware but are particularly resource-intensive. To leverage the advantages of both approaches while compensating for their shortcomings, researchers have proposed hybrid static-dynamic Android malware detection methods [18]. Hybrid detection employs static analysis first; if an application is detected as malicious, dynamic analysis is unnecessary. Otherwise, the application is submitted for dynamic analysis.

1 Related Work

Android malicious application detection methods are primarily categorized into static detection, dynamic detection, and hybrid approaches.

Static detection methods analyze Android application source code or binary information, extracting permissions, components, function calls, Intents, ICC (Inter-Component Communication), opcodes, and other content for analysis with machine learning algorithms. Permission and component analysis examines the permission, activity, service, provider, and receiver declarations in AndroidManifest.xml [7]. Permissions represent application capabilities and effectively reflect potential behaviors. Other component information serves as application entry points and reflects execution flow. Function call analysis involves structurally processing APIs used in application source code to construct function call graphs. These graphs begin execution from each application' s MainThread, undergo complex logical processing, and ultimately maintain application state

by the system. Generated function call graphs comprehensively reflect application execution processes [8]. Android Intents serve as carriers for component communication, transmitting data across component boundaries, and nearly all malicious applications utilize Intents. Feizollah et al. [9] demonstrated the effectiveness of Android Intents as features for detecting malicious applications. Idress et al. [10] constructed a framework for identifying Android malware using permissions and intents. ICC refers to data transmission among the four major Android components, and analyzing ICC usage enables malware detection [11]. Android applications are essentially binary files—sequences of 0s and 1s. Analyzing their format and extracting operation instructions can effectively identify malicious behaviors [12].

Dynamic analysis employs sandbox technology to simulate user click events, automatically completing application installation, execution, and uninstallation while recording information to detect malware. Application behavior in sandbox environments generates models, with different models belonging to different categories. When an application is detected as not belonging to benign categories, it is classified as malicious [13]. Altering click event sequences can produce different application behaviors for comparison to determine maliciousness [14]. System data used during application runtime can be tagged and tracked to analyze information flow and detect anomalous behavior [15]. Monitoring network traffic generated during application execution also effectively reflects application behavior [16]. Some malicious applications detect whether they are running on actual devices to hide their behavior. Therefore, Salehi et al. [17] created a host-based lightweight detection system that operates on mobile devices and can reconstruct application behavior.

The aforementioned detection methods represent the current focus of Android malware detection research. However, these approaches emphasize feature content while neglecting statistical significance and fail to adequately address individual difference impacts. The method proposed in this paper offers valuable insights for Android malware detection.

2 Design and Implementation

The Android malicious application detection method based on statistical features is a static detection approach comprising three components: data collection, feature extraction, and model training (Figure 1 [Figure 1: see original paper]).

2.1 Data Collection

This paper collected extensive malicious and benign samples. Malicious samples were obtained from the Arp D [19] dataset, containing 5,560 malicious applications across 179 malware families, including trojans, adware, spyware, and information-stealing applications. Benign samples were sourced from the Google Play Store—the official Android smartphone application market with

strict application vetting. Applications on Google Play undergo multiple checks by developers, code reviewers, and metadata validators before publication, making them reliably benign. The benign sample dataset contains 3,000 applications from 27 Google Play categories.

2.2 Feature Extraction

Apktool [20] can decompile Android apk files and decode resources to their nearest original form. Unzip [21] can decompress apk files. This paper utilizes both tools for feature extraction. Decompiling yields numerous folders and files, as shown in Figure 2 [Figure 2: see original paper].

The extracted statistical features include permission components, smali, and dex features. These features are closely related to application execution, with some clearly reflecting executable behaviors, making them meaningful for detection.

AndroidManifest.xml declares all permissions and components requested by an application. Apktool.xml contains additional information such as version details. The Res folder contains various resources. The Smali folder contains smali files derived from Java bytecode, while the original folder contains binary information. Figure 3 [Figure 3: see original paper] shows an example of a decompressed apk file.

This paper analyzed AndroidManifest.xml from Figure 2 and extracted numerous features, such as the number of activities in an apk. Table 1 displays permission and component features.

DexFormat [22] specifies the format for .dex files. By analyzing this format, this paper collected the features listed in Table 2 .

All smali files from the same application are stored in the smali folder after decompilation. Each .smali file follows a fixed syntax format. This paper records the quantity of feature information from smali files, such as static fields and protected fields. Table 3 lists partial statistical features.

2.3 Model Construction

The learning and training module shown in Figure 4 [Figure 4: see original paper] constitutes a crucial component of this paper. The training module consists of two layers.

The first layer employs a clustering algorithm to reduce the impact of individual differences among malicious applications on experimental results. Since clustering algorithms exhibit randomness when selecting centroids, this paper selects the clustering result with minimal volatility across multiple experiments (10 runs) to determine the final clustering. Subsequently, benign samples are added to each malicious cluster to form training sets, which are then trained using multiple machine learning algorithms: linear regression, support vector regression, neural networks, random forest, K-nearest neighbors, ridge regression,

and Bayesian ridge regression. Finally, partial malicious and benign samples from the remaining dataset construct test sets to evaluate each model, obtaining the AUC (Area Under ROC Curve) for each algorithm per cluster. The model with the highest AUC value for each cluster becomes the final cluster model.

In the second training layer, this paper first constructs temporary training and test sets. The first-layer cluster models then compute these temporary sets, with each model producing a column of results. With four cluster models, four columns are obtained as follows:

$$y_i = \text{Model}_{\text{Cluster}_i}(\text{temporary-set}) \quad (i = 1, 2, 3, 4)$$

These four columns of data are then combined with the label column:

$$Z = \{y_i, \text{label}\} \quad (i = 1, 2, 3, 4)$$

Finally, similar to the first layer, multiple algorithms are executed on the training set, and the test set is used to obtain the AUC for each algorithm. The algorithm with the best AUC is selected as the final model for the second layer.

This paper provides two models to meet different scenario requirements. Model 1 combines permission, component, and smali features. Model 2 combines permission, component, and dex features. The following discussion proceeds by model.

3 Experimental Results and Analysis

3.1 Clustering Analysis

KMeans is one of the most popular clustering algorithms. It implements clustering by randomly selecting centroids and assigning samples to the nearest centroid class. Consequently, random centroids produce clusters of varying sizes. When centroid numbers vary from 3 to 5, this paper runs experiments 10 times, with results shown in Figure 5 [Figure 5: see original paper]. In each subplot, the x-axis represents the number of centroids, while the y-axis represents each cluster' s size. Both Model 1 and Model 2 maintain relatively stable cluster sizes at four centroids. Therefore, the clustering algorithm indicates that four clusters yield the most stable clustering effect.

3.2 Cluster Models

Based on multiple algorithms, this paper records the True Positive Rate (TPR) for each algorithm per cluster as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where TP represents true positive samples (positive samples correctly detected as positive), and FN represents false negative samples (positive samples incorrectly detected as negative).

Additionally, the False Positive Rate (FPR) for each algorithm per cluster is calculated as:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

where FP represents false positive samples (negative samples incorrectly detected as positive), and TN represents true negative samples (negative samples correctly detected as negative).

Furthermore, this paper records the accuracy for each algorithm per cluster as the ratio of correct predictions to all results. The AUC for each cluster algorithm is then obtained by plotting ROC curves.

This paper records the optimal algorithm AUCs, as shown in Table 4 and Table 5 .

3.3 Final Model

This paper uses each cluster model to transform temporary training and test sets into final sets. Each model computes the temporary set to obtain a column of data. With four cluster models, four columns are obtained:

$$y_i = \mathrm{Model}_{\mathrm{Cluster}_i}(\text{temporary-set}) \quad (i = 1, 2, 3, 4)$$

The four columns of data are then merged with the label column:

$$Z = \{y_i, \mathrm{label}\} \quad (i = 1, 2, 3, 4)$$

Subsequently, different algorithms are executed on the final training and test sets for Models 1 and 2. The experimental results are recorded in Table 6 and Table 7 .

The results show that MLPRegressor achieves the best AUC in both Model 1 and Model 2. Therefore, this paper uses MLPRegressor as the final algorithm and constructs the final model based on this algorithm.

3.4 Comparative Analysis

In DREBIN [19] from the Arp D dataset, researchers combined SVM (Support Vector Machine) algorithms with numerous Android application features to distinguish malicious from benign applications. Since this paper' s malicious applications are sourced from the DREBIN dataset, the proposed method is

compared with DREBIN in terms of time efficiency and accuracy, with results shown in Figure 6 [Figure 6: see original paper] and Figure 7 [Figure 7: see original paper].

Figure 6 demonstrates that Model 1 significantly reduces detection time using the same test samples, while Figure 7 shows that Model 2 achieves the highest accuracy.

4 Discussion

The above sections introduced the innovation, experimental analysis, and comparative results of this work. Compared to previous research, this paper offers advantages in detection efficiency and accuracy. However, limitations remain, which are briefly discussed below along with future research directions.

The proposed Android malware detection method based on statistical features extracts features from samples for model training and detection. The extracted feature dimensions remain low—only four dimensions—which cannot fully reflect the value of statistical features in Android malware detection. Additionally, since no suitable benign sample set currently exists in research, the selection of benign samples in this paper is not perfectly matched, requiring alternative solutions. Finally, the overall sample capacity needs expansion to further improve detection accuracy.

5 Conclusion

Existing research has extracted data from multiple aspects of Android applications as features for malware detection, including permissions, components, dynamic logs, and network traffic. However, current studies only analyze the practical significance behind features (e.g., permissions represent operations an application can perform) while neglecting their statistical significance. Furthermore, nearly all detection methods suffer from inadequate detection times. This work employs clustering algorithms to minimize individual difference impacts and combines multiple machine learning algorithms for model training. The results demonstrate that the proposed models require less time to detect application sets while maintaining high detection accuracy. Future work will further explore the correlation between features and individuals and extract features from different perspectives to improve Android malware detection accuracy.

References

[1] Simon B, Melissa C, Peggy C, et al. Smartphone OS [EB/OL]. (2018-04-20). https://www.idc.com/promo/smartphone-market-share/os.

[2] 360 烽火实验室, 360 互联网安全中心. 2017 年 Android 恶意软件年度专题报告 [EB/OL]. (2018-03-02). http://zt.360.cn/1101061855.php?dtd101061451&did=491056914.

[3] Liang Shuang, Du Xiaojiang. Permission-combination-based scheme for Android mobile malware detection [C]//Proc of IEEE International Conference on Communications. 2014: 2301-2306.

[4] Li Li, Bartel A, Bissyandé T F, et al. IccTA: detecting inter-component privacy leaks in Android apps [C]//Proc of IEEE//ACM International Conference on Software Engineering. 2015: 280-291.

[5] Maiorca D, Mercaldo F, Giacinto G, et al. R-PackDroid: API package-based characterization and detection of mobile ransomware [C]//Proc of Symposium on Applied Computing. 2017: 1718-1723.

[6] Tang Junjie, Cui Xingmin, Zhao Ziming, et al. NIVAnalyzer: a tool for automatically detecting and verifying next-intent vulnerabilities in Android apps [C]//Proc of IEEE International Conference on Software Testing, Verification and Validation. 2017: 492-499.

[7] Bhattacharya A, Goswami R T. DMDAM: data mining based detection of Android malware [C]//Proc of Icic2 Intelligent Computing and Communication. 2017.

[8] Narayanan A, Liu Yang, Chen Lihui, et al. Adaptive and scalable Android malware detection through online learning [C]//Proc of International Joint Conference on Neural Networks. 2016: 157-175.

[9] Feizollah A, Anuar N B, Salleh R, et al. AndroDialysis: analysis of Android intent effectiveness in malware detection [J]. Computers & Security, 2017, 65(C): 121-134.

[10] Idrees F, Rajarajan M, Conti M, et al. PIndroid: a novel android malware detection system using ensemble learning methods [J]. Computers & Security, 2017, 68: 36-46.

[11] Xu Ke, Li Yingjiu, Deng R H. ICCDetector: ICC-based malware detection on Android [J]. IEEE Trans on Information Forensics & Security, 2016, 11(6): 1252-1264.

[12] Yan Jinpei, Qi Yong, Rao Qifan. LSTM-based hierarchical denoising network for Android malware detection [J]. Security & Communication Networks, 2018, 2018: 1-18.

[13] Saracino A, Sgandurra D, Dini G, et al. MADAM: effective and efficient behavior-based Android malware detection and prevention [J]. IEEE Trans on Dependable & Secure Computing, 2018, PP(99): 1-1.

[14] Tam K, Khan S J, Fattori A, et al. CopperDroid: automatic reconstruction of Android malware behaviors [C]//Proc of Network and Distributed System Security Symposium. 2015.

[15] Chen Li, Zhang Mingwei, Yang C Y, et al. POSTER: semi-supervised classification for dynamic Android malware detection [C]//Proc of ACM Sigsac Conference on Computer and Communications Security. 2017: 2479-2481.

[16] Zulkifli A, Hamid I R A, Shah W M, et al. Android malware detection based on network traffic using decision tree algorithm [C]//Proc of International Conference on Soft Computing and Data Mining. Cham: Springer, 2018: 405-414.

[17] Salehi M, Amini M. Android malware detection using Markov chain model of application behaviors in requesting system services [EB/OL]. (2017). arXiv preprint arXiv:1711.05731.

[18] Narayanan A, Chandramohan M, Chen L, et al. A multi-view context-aware approach to Android malware detection and malicious code localization [J]. Empirical Software Engineering, 2017(6): 1-53.

[19] Arp D, Spreitzenbarth M, Hübner M, et al. DREBIN: effective and explainable detection of Android malware in your pocket [C]//Proc of Network and Distributed System Security Symposium. 2014.

[20] Connor T, Ryszard W. A tool for reverse engineering Android apk files [EB/OL]. (2017-09-21). https://ibotpeaches.github.io/Apktool/.

[21] Ed G, Christian S, Mike W, et al. Info-ZIP [EB/OL]. (2008-07-07). http://www.info-zip.org/.

[22] Google company. dalvik executable format [EB/OL]. (2017-06-20). https://source.android.com/devices/tech/dalvik/dex-format.

Note: Figure translations are in progress. See original paper for figures.