A Metadata-Based Android Malware Detection Method (Postprint)

Authors: Li Jianghua, Qiu Chen

Date: 2018-08-13T00:00:00+00:00

Abstract

Android applications commonly exhibit more functionalities than those inherent to their designated categories, thereby necessitating the acquisition of additional permissions, which may introduce certain security vulnerabilities when present in excess. To address these concerns, we propose an Android malware detection methodology based on meta-information. Initially, LDA topic extraction is performed on Android application descriptions to achieve data dimensionality reduction, followed by the application of the K-means clustering algorithm to group applications according to functional categories. Subsequently, for all applications within the same functional category, permission information is extracted and, with permission features serving as the research focus, the kNN algorithm is employed for the classification and detection of Android malware. Experimental results demonstrate an average accuracy of 94.81%, thereby validating the effectiveness and high accuracy of the proposed method.

Full Text

Preamble

Title: An Android Malware Detection Method Based on Meta-Information

Authors: Li Jianghua†, Qiu Chen (School of Information Engineering, Jiangxi University of Science & Technology, Ganzhou, Jiangxi 341000, China)

Abstract: Android applications often provide more functionalities than their designated categories suggest, requiring additional permissions that may introduce security risks. To address this issue, we propose an Android malware detection method based on meta-information. First, we extract LDA topics from Android application descriptions to reduce dimensionality and group applications by functionality using the K-means clustering algorithm. Then, for applications within the same functional category, we extract their permission information and employ the kNN algorithm for malware classification. Experimental results

demonstrate an average accuracy of 94.81%, confirming the effectiveness and high accuracy of our approach.

Keywords: Android malware detection; meta-information; application description; permission features

1. Related Research

Android application meta-information refers to the details users see before downloading and installing an app, such as descriptions and ratings. This meta-information encompasses numerous data types, and the specific information displayed varies across different app markets due to differences in management, business models, and design approaches.

Application descriptions serve to introduce app functionalities. Notably, many applications offer more features than their categories indicate, which is only evident from their descriptions. Since executing these additional functions requires corresponding permissions, apps with extended functionalities often request more permissions than typical for their category, potentially creating security vulnerabilities. Determining whether such apps are malicious requires further analysis.

Current Android malware detection research primarily focuses on static features (permissions, Java code) and dynamic features (system calls, network traffic), with approaches combining both static and dynamic characteristics gaining popularity among researchers. Zhou Yujuan et al. proposed a detection method using permission information and privilege escalation threats as features. Enck et al. introduced Kirin security rules for Android malware detection. Felt et al. argued that all application behaviors relate to API calls, which can be mapped to permissions. They used an automated tool to detect Android API calls and construct permission mappings for identifying excessive permissions, experimentally demonstrating that one-third of applications over-request permissions. Yang Huan et al. employed association rule mining to discover relationships between permission combinations and malicious apps, achieving 87% detection accuracy. Wen Weiping et al. also proposed dangerous permission combination rules, such as RECEIVE_{{{BOOT}}*{{COMPLETED}}}* + *INTERNET* + *ACCESS*{{{COARSE}}_{{LOCATION}}}, which could leak user location. However, a sports game app named "Puppet Football Fighters-Steampunk Soccer" contains this exact combination yet was classified as benign by the authoritative VirusTotal detection site, indicating that dangerous permission combination rules lack universal applicability.

Hardware and software limitations constrain the practical deployment of some effective methods. Consequently, malware detection based on Android application meta-information has attracted researcher attention. Teufl et al. analyzed meta-information collected from Google Play using complex knowledge

discovery processes and statistical methods combined with machine learning algorithms. Martín et al. explored the ADROIT method using multiple meta-information types, reporting 93.67% accuracy. However, processing numerous meta-information types proves challenging. Therefore, Gorla et al. proposed a method using fewer meta-information types, determining application themes from descriptions (with theme words manually defined) and identifying API call anomalies related to those themes, correctly identifying 56% of malicious apps in the MalGenome dataset.

To improve detection rates while using minimal meta-information and determining whether functionally-extended apps are malicious, this paper proposes a novel Android malware detection method based on meta-information, focusing on Android app meta-information for malware detection research.

---

2. Proposed Method

Our method examines Android application descriptions and permissions—both forms of meta-information—through two main steps:

a) Determine application functional types from Android app descriptions and group applications accordingly. Application descriptions introduce app functionalities. By extracting keywords describing these functions and applying clustering algorithms, we can identify functional types and group applications by function.

b) For all applications within the same functional category, extract permission information and use permission features as the research object for Android malware classification detection.

Permissions signify an application' s capability to access specific functions. Therefore, applications of the same functional type should have similar permission lists. If an app possesses extraneous permissions, it may be malicious. Analyzing these permissions helps determine maliciousness.

Illustrative Example: Consider four applications A, B, C, and D with functionalities shown in . After clustering, they are grouped by function as shown in , where each group is named by its primary function. For the group represented by function 6 containing applications B, C, and D, these three apps constitute the sample set. This set is divided into training and test subsets: B and C form the training set, while D forms the test set. Assume B is malicious and C is benign. After vectorizing permission information, if B' s permission vector is $\langle p_1, p_2, ..., p_m \rangle$, C' s is $\langle p'_1, p'_2, ..., p'_m \rangle$, and D' s is $\langle p''_1, p''_2, ..., p''_m \rangle$. For test sample D, we calculate its similarity to training samples B and C, denoted as η_{BD} and η_{CD}. If η_{BD} meets a preset threshold, D shares B' s nature and is malicious; if η_{CD} meets the threshold, D shares C' s nature and is benign.

Permission Information Vectorization Process:

a) For all applications in the same functional category, count distinct permission names and their frequencies in the permission lists.

b) Randomly assign or follow a predetermined order (e.g., dictionary order) for permission names.

c) For each application, compare its permission list against the ordered permission list, marking "1" for matches and "0" otherwise. The sum of "1" s and "0" s equals the number of distinct permissions, yielding an n-dimensional vector for that application' s permission information. Repeating this produces permission feature vectors for all applications.

d) All applications in the group form the sample set, which is split into training and test subsets.

e) After vectorizing permission information, calculate similarity between permission vectors from the training set and each test sample. If a training sample' s permission vector meets a similarity threshold with the test sample, they share the same nature. A malicious training sample indicates a malicious test sample; a benign training sample indicates a benign test sample.

---

3. Experiments

3.1 Experimental Setup

Our experiments were conducted on a laptop with an Intel CORE i3 processor, Windows 7 64-bit operating system, and 4 GB RAM. The software environment comprised PyCharm 5.0.3 as the IDE, Python as the programming language, and Python 3.5 as the compiler.

3.2 Experimental Data

All data were collected from the Aptoide app market using web crawlers. Aptoide provides comprehensive Android application meta-information, including app descriptions, ratings, compatibility (Android version), permissions, app names, sizes, version numbers, release dates, minimum screen requirements, supported processor types, package IDs, MD5 values, SHA1 signatures, developer codes, organizations, locations, countries, provinces, and more. We randomly crawled 17,812 app meta-information records.

As an open market, Aptoide accepts submissions from developers worldwide, resulting in app descriptions in various languages. Many applications also lacked partial information. To avoid unpredictable impacts from missing data, we removed records with incomplete information and those without app descriptions or with non-Chinese/English descriptions, retaining only records with Chinese or English descriptions. After this filtering, approximately 5,000 records remained,

containing app descriptions, permissions, and app names. We further removed applications whose permission lists contained only Normal-class permissions, as these are all benign. This yielded our final experimental dataset.

3.3 Data Preprocessing

Application descriptions are textual data requiring processing before computational analysis. After cleaning, we obtain a corpus of app descriptions and apply the LDA topic model for topic extraction. LDA processing yields relationships (affinities) between documents and topic words, expressed as probabilities (percentages), as shown in .

LDA topic extraction not only captures document-topic word relationships but also reduces dimensionality. However, the data still requires conversion for K-means processing. We proceed by vectorizing topic words, converting them into feature vectors.

Example: Consider three sentences to be vectorized: a) Apple is delicious. b) I love apple. c) Apple is delicious and I love apple.

After cleaning (removing special characters, stop words, etc.), these become as shown in . The first sentence vectorizes as $[1, 0, 1, 0]$, the second as $[1, 1, 0, 1]$, and the third as $[1, 1, 1, 1]$, completing the vectorization.

While direct vectorization of the cleaned corpus would suffice, we chose LDA topic extraction first, then vectorized the topic words. This is because clustering algorithms perform better with fewer features. Converting descriptions to topics is crucial for achieving better clustering results.

After LDA extraction, we obtain document-topic and topic-word relationships and can output the extracted topic words.

3.4 Clustering Analysis

Typically, an application has at most five different functional types, so we set a maximum of five topics per document, though these could be identical, indicating no extra functionality. After LDA processing, we obtain affinity vectors between app description documents and topics, which serve as input for the K-means clustering algorithm. We set the number of cluster centroids k in the range 2-30, finding that $k = 14$ yields optimal clustering results. The resulting application groups are shown in .

Analyzing these apps, in China, "connection"category commonly includes "WiFi Master Key," "social" includes "WeChat" and "QQ," and "shopping" includes "Taobao"and "JD.com."These apps have massive user bases, making them prime targets for malware developers.

The "game"category actually comprises "challenge games,""sports games,""battle games,""racing games,"etc. These different game types differ only in gameplay or operations and are clustered into the same category. In , "theme and wallpaper"

is not grouped with "wallpaper." Unlike games, changing wallpaper only alters the screen background, while changing themes modifies icons, background, and overall colors—fundamentally different application types.

3.5 Discriminant Detection

Before conducting detection experiments, we used VirusTotal to determine each app' s nature (malicious or benign). Among the 5,000+ applications, 624 were malicious, distributed across categories as shown in .

Malware primarily concentrates in "manager," "player," "news and sharing," and "advertisement" categories, with smaller numbers in other types. The sum of values in the second and third columns of exceeds the actual sample and malware counts because some applications have multiple functions and are assigned to multiple categories. Malicious apps may also be counted multiple times due to their functional diversity.

3.5.1 Handling Sample Imbalance Sample imbalance significantly impacts results. For instance, if the training set contains two classes but one class substantially outnumbers the other, kNN's majority voting principle may misclassify samples because correct labels are outnumbered by incorrect ones among the k nearest neighbors.

To address misclassification from imbalance, we process the training data using undersampling and oversampling methods. Undersampling reduces the majority class to match the minority class by discarding samples, while oversampling replicates minority class samples to achieve balance.

In ensemble methods, bootstrap sampling creates new training datasets by randomly sampling with replacement from the original dataset. We combine undersampling, oversampling, and bootstrap sampling to create new training datasets where the majority class size approaches or equals the minority class size, preventing misclassification from imbalance. Since undersampling might exclude valuable samples, we perform multiple sampling experiments.

Let dataset A represent the majority class and dataset B the minority class. If A contains t times more samples than B:

a) If t is an integer, randomly sample without replacement from A to split it into t new datasets. Each combines with B to form t training datasets for kNN. For a given k, kNN runs t times, producing t test results for the same data. Majority voting determines the final classification.

b) If t is not an integer, split A into $\lfloor t \rfloor$ datasets as in (a). For the remaining data, randomly select additional samples without replacement from the original dataset to supplement the $\lfloor t \rfloor + 1$-th dataset. Each combines with B to form $\lfloor t \rfloor + 1$ training datasets. For a given k, kNN runs $\lfloor t \rfloor + 1$ times, with majority voting determining the final result.

3.5.2 Results and Analysis For "manager," "player," "news and sharing," and "advertisement" categories, we split benign and malicious samples into training and test sets at a 2:1 ratio, as shown in .

The data in reveals significant imbalance, with benign samples far outnumbering malicious ones: "manager" has 3.6× more benign samples, "player" has 7.12×, "news and sharing" has 2.85×, and "advertisement" has 3.6×. To prevent error rate increases from this disparity, we applied the Section 3.5.1 approach to create multiple training datasets. This yielded 4, 8, 3, and 4 training sets for "manager," "player," "news and sharing," and "advertisement," respectively, with test sets unchanged. The restructured training sets are shown in through .

After vectorizing permission features from each training set, we input them to the kNN algorithm for classification testing. To avoid ties in nearest neighbor category assignment, we selected odd values for k. The k range was unified across the four categories as $\{1, 3, 5\}$.

Let TP denote correctly classified benign apps, FP denote malicious apps misclassified as benign, TN denote correctly classified malicious apps, and FN denote benign apps misclassified as malicious. The kNN classification results for each functional category are shown in and .

Analysis reveals that the optimal k value differs across categories: "manager" achieves highest accuracy (95.74%) at $k = 3$; "player" reaches 95.34% at $k = 1$; "news and sharing" attains 95.28% at $k = 5$; and "advertisement" achieves 93.49% at $k = 3$. The average accuracies are 94.81%, 93.79%, 94.69%, and 91.51%, respectively.

compares our method with recent related work. Methods in [11,14] use permission features but require app decompilation first. Methods in [16,17] use meta-information directly extractable from markets, avoiding decompilation. However, [17]' s model, built primarily on benign apps, suffers high false positives due to limited malicious samples. Compared to ADROIT [16], which also uses Aptoide meta-information, our detection rate is comparable yet improved, and significantly higher than methods using only permission features. This validates our approach' s effectiveness and high accuracy.

---

4. Conclusion

Android malware detection is a critical research area. Current studies predominantly focus on static and dynamic feature-based methods. This paper proposes an Android malware detection approach from the meta-information perspective of application descriptions. Experimental validation confirms our method' s effectiveness. Using classic clustering and classification algorithms, detection accuracy depends on their performance and offers room for improvement—representing our future work direction.