International Conference on Advances in Materials, Computing and Communication Technologies
In Association with International Journal of Scientific Research in Science and Technology
Volume 9 | Issue 1 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)

Android Malware Detection

Mrs Hamsareka S[1], Krishnamoorthy S[2], Prasanth T[2], Purusothaman R[2], Santhosh Kumar A[2]

[1]Assistant Professor, Department of Computer science and engineering, KSR Institute for Engineering and Technology, Tiruchengode, Tamil Nadu, India

[2]Department of Computer science and engineering, KSR Institute for Engineering and Technology, Tiruchengode, Tamil Nadu, India

ABSTRACT

Malicious applications pose a threat to the security of the Android platform. The growing amount and diversity of these applications render conventional defenses largely ineffective and thus Android smartphones often remain un-protected from novel malware. In this paper, we propose DREBIN, a lightweight method for detection of Android malware that enables identifying malicious applications directly on the smartphone. As the limited resources impede monitoring applications at run-time, DREBIN performs static analysis abroad, gathering as many features of an application as possible. These features are embedded in a joint vector space, such that typical patterns indicative for malware can be automatically identified and used for explaining the decisions of our method. In an evaluation with 123,453 applications and 5,560 malware samples DREBIN outperforms several related approaches and detects 94% of the malware with few false alarms, where the explanations provided for each detection reveal relevant properties of the detected malware. On five popular smartphones, the method requires 10 seconds for an analysis on average, rendering it suitable for checking downloaded applications directly on the device.

Keywords: Android, Malware, Risk Ranker, mobile security, permissions

I. INTRODUCTION

Android is one of the most popular platforms for smartphones today. With several hundred thousands of applications in different markets, it provides a wealth of functionality to its users. Unfortunately, smart phones running Android are increasingly targeted by attackers and infected with malicious software. In contrast to other platforms, Android allows for installing applications from unverified sources, such as third-party markets, which makes bundling and distributing applications with malware easy for attackers. According to a recent study over 55,000 malicious applications and 119 new malware families have been discovered in 2012 alone [18]. It is evident that there is a need for stopping the proliferation of malware on the Android market and smartphones. The Android platform provides several security measures that harden the installation of malware, most notably the Android permission system. To perform certain tasks on the device, such as sending a SMS message, each application has to

explicitly request permission from the user during the installation. However, many users tend to blindly grant permissions to unknown applications and thereby undermine the purpose of the permission system. As a consequence, malicious applications are hardly constrained by the Android permission system in practice.A large body of research has thus studied methods for analyzing and detecting Android malware prior to their installation. These methods can be roughly categorized into approaches using static and dynamic analysis. For example, TaintDroid [11], DroidRanger [40] and DroidScope [37] are methods that can monitor the behavior of applications at run-time. Although very effective in identifying malicious activity, run-time monitoring suffers from a significant over-head and can not be directly applied on mobile devices. By Contrast, static analysis methods, such as Kirin [13], Stow-away [15] and RiskRanker [21], usually induce only a small run-time overhead. While these approaches are efficient and scalable, they mainly build on manually crafted detection patterns which are often not available for new malware instances. Moreover, most of these methods do not provide explanations for their decisions and are thus opaque to thepractitioner.In this paper, we present DREBIN, a lightweight method for detection of Android malware that infers detectionterns automatically and enables identifying malware directly on the smartphone. DREBIN performs a broad static analysis, gathering as many features from an application's code and manifest as possible. These features are organized in sets of strings (such as permissions, API calls and network addresses) and embedded in a joint vector space. As an example, an application sending premium SMS messages is cast to a specific region in the vector space associated with the corresponding permissions, intents and API calls. This geometric representation enables DREBIN to identify combinations and patterns of features indicative for malware automatically using machine learning techniques.

For each detected application the respective patterns can be extracted, mapped to meaningful descriptions and then provided to the user as explanation for the detection. Aside from detection, DREBIN can thus also provide insights into identified malware samples.
Experiments with 123,453 applications from different markets and 5,560 recent malware samples demonstrate the efficacy of our method: DREBIN outperforms related approaches [13, 26, 33] as well as 9 out of 10 popular anti-virus scanners. The method detects 94% of the malware samples with a false-positive rate of 1%, corresponding to one false alarm in 100 installed applications. On average the analysis of an application requires less than a second on a regular computer and 10 seconds on popular smartphone models. To the best of our knowledge, DREBIN is the first method which provides effective and explainable detection of Android malware directly on smart phone devices .In summary, we make the following contributions to the detection of Android malware in this paper:

- Effective detection.

We introduce a method combining static analysis and machine learning that is capable of identifying Android malware with high accuracy and few false alarms, independent of manually crafted detection patterns.

- Explainable results.

The proposed method provides an explainable detection. Patterns of features indicative for a detected malware instance can be traced back

- Lightweight analysis.

For efficiency we apply linear time analysis and learning techniques that enable detecting malware on the smart phone as well as analyzing large sets of applications in reasonable time.We need to note here that DREBIN builds on concepts of static analysis and thus cannot rule out the presence of obfuscated or dynamically loaded malware on mobile devices. We specifically discuss this limitation of our approach in Section 4. Due to the broad analysis of features however, our method raises the bar for attackers to

infect smartphones with malicious applications and strengthens the security of the Android platform, as demonstrated in our evaluation .The rest of this paper is organized as follows: DREBINand its detection methodology are introduced in Section 2.Experiments and a comparison with related approaches are presented in Section 3. Limitations and related work are discussed in Section 4 and Section 5, respectively. Section 6 concludes the paper.

II. RELATED WORK:

The analysis and detection of Android malware has been a vivid area of research in the last years. Several concepts and techniques have been proposed to counter the growing amount and sophistication of this malware. An overview of the current malware landscape is provided in the studies of Felt et al. [16] and Zhou & Jiang [39].(1)

Detection using Static Analysis The first approaches for detecting Android malware have been inspired by concepts from static program analysis.Several methods have been proposed that statically inspect applications and disassemble their code [e.g., 12, 13, 15,21]. For example, the method Kirin [13] checks the permission of applications for indications of malicious activity. Similarly, Stowaway [15] analyzes API calls to detect overprivileged applications and RiskRanker [21] statically identifies applications with different security risks. Common open-source tools for static analysis are Smali [17] andAndroguard [10], which enable dissecting the content of applications with little effort.Our method DREBIN is related to these approaches and employs similar features for identifying malicious applications, such as permissions, network addresses and API calls. However, it differs in two central aspects from previous work: First, we abstain from crafting detection patterns manually and instead apply machine learning to analyze information extracted from static analysis. Second,the analysis of DREBIN is optimized for effectivity and efficiency, which enables us to inspect application directly on the smartphone.(2)

Detection using Dynamic Analysis

A second branch of research has studied the detection of Android malware at run-time. Most notably, are the analysis system TaintDroid [11] and DroidScope [37] that enable dynamically monitoring applications in a protected environment, where the first focuses on taint analysis and the later enables introspection at different layers of the platform.While both systems provide detailed information about the behavior of applications, they are technically too involved to be deployed on smartphones and detect malicious software directly. As a consequence, dynamic analysis is mainly applied for offline detection of malware, such as scanning and analyzing large collections of Android applications. For example, the methods DroidRanger [40], AppsPlayground [29], and CopperDroid [31] have been successfully applied toStudy applications with malicious behavior in different Android markets. A similar detection system called Bouncer Is currently operated by Google. Such dynamic analysis systems are suitable for filtering malicious applications from Android markets. Due to the openness of theAndroid platform, however, applications may also be installed from other sources, such as web pages and memory sticks, which requires detection mechanisms operating on the smartphone.ParanoidAndroid [28] is one of the few detection systems that employs dynamic analysis and can spot malicious activity on the smartphone. To this end, a virtual clone ofthe smartphone is run in parallel on a dedicated server and synchronized with the activities of the device. This setting allows for monitoring the behavior of applications on the clone without disrupting the functionality of the real device.The duplication of functionality, however, is involved with millions of smartphones in practice operating ParanoidAndroid at large scale is technically not feasible.(3)

Detection using Machine Learning

The difficulty of manually crafting and updating detection patterns for Android malware has motivated the application of machine learning. Several methods have been proposed that analyze applications automatically using learning methods [e.g., 2, 26, 33]. As an example,the method of Peng et al. [26] applies probabilistic learning methods to the permissions of applications for detecting malware. Similarly, the methods Crowdroid [4], DroidMat [36], Adagio [20], MAST [5], android DroidAPIMiner [1]analyze features statically extracted from Android applications using machine learning techniques. Closest to our work is DroidAPIMiner [1] which provides a similar detection performance to DREBIN on recent malware. However, DroidAPIMiner builds on a k-nearest neighbor classifier that induces a significant runtime overhead and impedes operating the method on a smartphone. Moreover,DroidAPIMiner is not designed to provide explanations for its detections and therefore is opaque to the practitioner.Overall, previous work using machine learning mainly focuses on an accurate detection of malware. Additionalaspects, such as the efficiency and the explainability of the detection, are not considered. We address these aspects and propose a method that provides an effective, efficient and explainable detection of malicious applications.(4) System Call Monitoring. Systems such as [30,33,34] detect malware by monitoring and analysis of system calls. A fundamental shortcoming of such approaches is the semantic gap between the system calls and specific behaviors. DroidScope [37] is designed to reconstruct both OS-level and Java-level semantics. Their dynamic analysis approach is limited by path exploration challenges.(5)

Android Permission Monitoring studied security of Android apps by analyzing the permissions registered in the top official Market apps [21]. Stowaway [23] and COPES [16] are designed to find those apps that request more permissions than they need. PScout [15] analyzes the usage trend of permissions in Android apps. Kirin [22] detected malicious Android apps by finding permissions declared in Android apps that break “predefined” security rules. More recent work also detected malicious Android apps by designing several classifiers, whose features were built primarily on the application categories and permissions [29]. A concern with these approaches is false positives stemming from the coarse-grained nature of permissions and the highly common nature of benign apps to over-claim their set of required permissions. Mario etal. [24] presented their studies of permission request patterns of Android and Facebook applications.Framework API Monitoring. DroidRanger [41] and Pegasus [18] detect malicious Android apps by statically matching against “pre-defined” signatures (permissions and Android Framework API calls) of well-known malware families. Such approaches require semi-manual analysis of suspicious system calls and manual selection of heuristics (or detection patterns). Thus, they are not systematic and not robust to the evolution of malware. In [36,14] the frequencies of API calls were used as detection features,and more recently in [12], the names and parameters of APIs and packages were used as detection features.

Online Malware Detection Service. We intend to make DroidMiner available as a public web service for Android malware analysis and detection. Similar public services include AndroTotal [27] which allows users to submit applications and have them simultaneously analyzed by various mobile antivirus systems and CopperDroid [32] which performs system-call centric dynamic analysis.Due to space limit, we leave more detailed comparisons and discussions in [38].(6)

Android Platform Security Defense and Analysis Existing studies have also developed several security extensions to defend against specific types of attacks. TaintDroid [20] detects those apps that may leak users’ privacy information. However, it is not designed to detect other types of malicious behaviors such as stealthily sending SMS. RiskRanker[42] detects

malicious apps based on the knowledge of known Android system vulnerabilities, which could be utilized by malicious apps, and several heuristics. Dendroid [35] is a static analysis tool which specializes in text mining of android malware code. Quire [19] prevents confused deputy attacks. Bugiel et al. [17] proposed a security framework to prevent both confused deputy attacks and collusion attacks. AppFence [25] protects sensitive data by either feeding fake data or blocking the leakage path. Apex [28] allows for the selection of grantedpermissions, and Kirin [22] performs lightweight certification of applications. Paranoid Android [30], L4Android [26] and Cells [13] utilize the virtual environment to secure smartphone OS. SmartDroid [39] automatically finds UI triggers that result in sensitive information leakage.(7)

III. PROPOSED SYSTEM

The proposed system also focuses on Significant Permission Identification (SIGPOD). In addition identification of dangerous, benign as well as shutdown enabled permission list is also carried out. Feature reduction is also carried out. SVM classification for both all permission lists as well as feature reduced data sets is include.

MODULES:

The following modules are present in the proposed application.

1. Data set collection
2. Finding dangerous permissions list
3. Finding benign permissions list
4. Permission ranking with negative rate
5. Permission mining with association rule
6. Svm classification
7. Features reduction
8. Svm classification in features reduced data set
9. Association rule mining
10. Mutual information
11. Pearson correlation coefficient.

Data set collection:

All permission details of the app are saved in a single Excel workbook as records. This is the input for the project.

Finding dangerous permissions list:

Certain permission values such as READ_SMS, WRITE_SMS and the like are checked for values with '1' so that the apps are declared as dangerous and listed.

Finding benign permissions list:

Certain permission values such as BIND_SERVICE, and the like are checked for values with '1' so that the apps are declared as benign and listed.

Permission ranking with negative rate:

This module referred to as PRNR, provides a concise ranking and comprehensible result. The approach operates on two matrices, M and B. M represents a list of permissions used by malware samples and B represents a list of permissions used by benign apps. Mij represents whether the jth permission is requested by the ith malware sample, while "1" indicates yes and `0"indicatesno. Bij represents whether the jth permission is requested by the ith benign app sample.

Before computing the support of permissions from matrices M and B, it first checks their sizes. Typically, the number of benign tends to be much larger than number malicious apps; therefore, the size of B is much larger than the size of M. With this ranking scheme, it prefers the dataset on the two matrices to be balanced.The PNR algorithm is used to perform ranking of the datasets.In the formula above, R(Pj) represents the rate of the jth permission. The result of R(Pj) has a value ranging between [–1, 1]. If R(Pj)=1, this means that permission Pj is only used in the malicious dataset, which is a high-risk permission. If R(Pj) =–1, this means that permission Pj is only used in the benign dataset, which is a low-risk permission.

If R(Pj)=0, this means that Pj has a very little impact on malware detection effectiveness.

Permission mining with association rule:

In this module, after pruning some permission by using PNR and SPR with the PIS, it can remove non-influential permissions even more. By inspecting the reduced permission list that contains some significant permissions, it finds three pairs of permissions that always appear to get her in an app. For example, permission WRITE_SMS and permission READ_SMS are always used together. They also both belong to Google's "dangerous" permission list. Yet, it is unnecessary to consider both permissions, as one of them is sufficient to characterize certain behaviors. As a result, we can associate one, which has a higher support, to its partner. In this example, we can remove permission WRITE_SMS. In order to find permissions that occur together, it proposes a PMAR mechanism using the association rule mining algorithm.

Svm classification:

In this module, 70% of the data in the given data set is taken as training data and 30% of the data is taken as test data. The model is trained with training data and then predicted with test data. Of which, most of the apps are classified as Benign and fewer apps are classified as Suspicious.

Features reduction :

In this module, each column values are taken and find the number of '1's and '0' and their percentage is calculated. If any one of the percentages is above 95%, then the column is treated as non-sensitive and can be eliminated.

Svm classification in features reduced data set:

In this module, 70% of the data in the given data set is taken as training data and 30% of the data is taken as test data but with the columns after feature reduction. The model is trained with training data and then predicted with test data. Of which, most of the apps are classified as Benign and fewer apps are classified as Suspicious.

Association rule mining:

In this module, all the permissions are iterated in for loop and three columns are taken to find permission value '1' along with the next fourth column with permission value '1'. If the count of three columns values matched with the count of the fourth column then it is found out there is an association rule and printed out. The iteration continues for all 216 permissions.

Mutual information

In this module, mutual information is found out as follows: Let X denote a permission variable and C be the class variable. The relevance of X and C can be measured by mutual information of them as

$$I(X,C)=\sum_{x_i}\sum_{c_j}P(X=x_i,C=c_j)\log\frac{P(X=x_i,C=c_j)}{P(X=x_i)P(C=c_j)} \quad (1)$$

where P(C = cj) is the frequency count of class C with value cj, P(X = xi) is the frequency count of permission X with value xi, and P(X = xi,C = cj) is the frequency count of X with value xi in class cj. In this paper, the class C has binary values, c0 for benign apps and c1 for malicious apps. Each permission X is a boolean variable with value 1 or 0. I(X, C) is nonnegative in [0, 1]. I(X, C) = 0 indicates no correlation, while I(X,C) = 1 means that C is completely inferable by knowing X.

Pearson correlation coefficient:

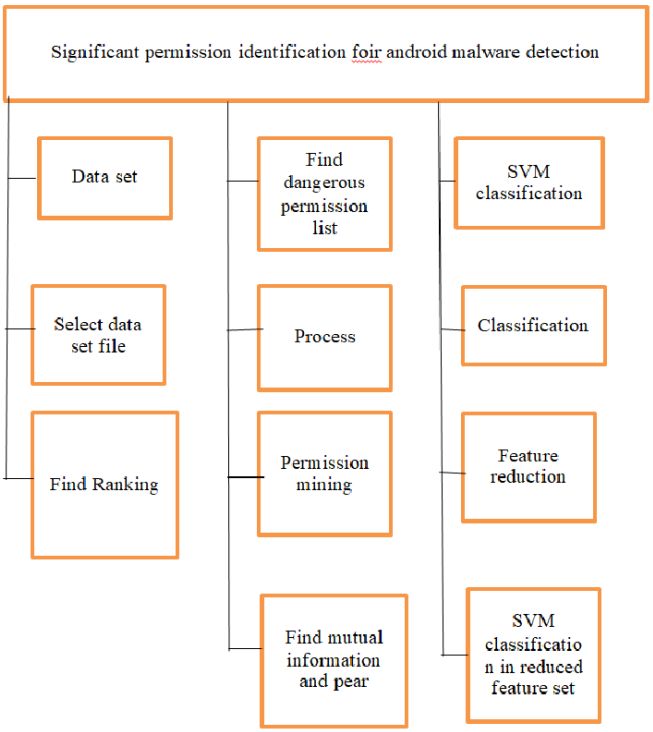
In this module, pearson correlation coefficient is found out as follows: Pearson Correlation Coefficient measures the relevance of X and C by

$$R(X,C)=\frac{\sum_{n=1}^{N}(X_n-\bar{X})(C_n-\bar{C})}{\sqrt{\sum_{n=1}^{N}(X_n-\bar{X})^2\sum_{n=1}^{N}(C_n-\bar{C})^2}}$$

where ¯ X (resp. ¯ C) is the average of all sample values of X (resp. C), Xn (resp. Cn), n = 1...N. R(X,C) has a value in [−1, 1], where R(X,C) = 0 indicates the

independency of X and C, R(X,C) = 1 indicates the strongest positive correlation of them and R(X,C) = –1 indicates the strongest negative correlation. R(X,C) = 1 means that permission request of X makes apps highest risky, while R(X,C) = –1 means that permission request of X makes apps lowest risky.

ARCHITECTURE:



IV. CONCLUSION

This proposed framework demonstrated how it is possible to reduce the number of permissions to be analyzed for mobile malware detection, while maintaining high effectiveness and accuracy. It has been designed to extract only significant permissions through a systematic three-level pruning approach. The existing system considers 22 permissions for malware apps but the proposed system analyzes 47 permissions are malware apps for the given data set. The difference is due to the non-sensitive permission features reduction. By adjusting the unique percentage in values of particular permission, the malware surety would be raised or lowered.

There are several directions for future research. The current investigation of classification is still preliminary. Furthermore, the algorithm consistently outperformed all the tested classification and methods under different conditions.The future enhancements can be made with still more permission sets.

V. REFERENCES

[1]. M.Grace, Y.Zhou, Q.Zhang, S.Zou and X.Jiang, “RiskRanker: Scalable and accurate zero-day android malware detection,” inProc.10thInt.Conf. Mobile Syst., Appl., Services, 2012, pp. 281–294.

[2]. A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in Proc. 18th ACM Conf. Comput. Commun. Security, 2011, pp. 627–638.

[3]. W. Enck et al., “TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones, ”ACMTrans. Comput.Syst., vol. 32, no. 2, 2014, Art. no. 5.

[4]. D. Arp, M. Spreitzenbarth, M. H¨ubner, H. Gascon, K. Rieck, and C. Siemens, “DREBIN: Effective and explainable detection of android malware in your pocket,” presented at Annu. Symp. Netw. Distrib. Syst. Security, 2014.

[5]. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, “DroidMiner: Automated mining and characterization of fine-grained malicious behaviors android applications,”inProc.Eur.Symp.Res.Comput.Security,2014, pp. 163–182.

[6]. Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent. http://www.gartner.com/it/ page.jsp?id=1848514.

[7]. Android Market. http://www.android.com/market/.

[8]. Amazon Appstore for Android. http://www.amazon. com/mobile-apps/b?ie=UTF8&node=2350149011.

[9]. APPLE,INC. Apple's App Store Downloads Top Three Billion. http://www.apple.com/pr/library/2010/ 01/05appstore.html, January2010.