|| Volume 7 || Issue 8 || October 2023 || ISSN (Online) 2456-0774
INTERNATIONAL JOURNAL OF ADVANCE SCIENTIFIC RESEARCH
AND ENGINEERING TRENDS

Malicious Pattern Detection from android API's using Machine Learning and Deep Learning

Tejal Sanjay Navarkar[1], Prof. R. V. Patil[2]

Student, Department Computer Engineering, S.S.V.P.S.'s B.S. Deore College of Engineering, Dhule, India
Professor, Department of Computer Engineering, S.S.V.P.S.'s B.S. Deore College Of Engineering, Dhule, India

Abstract: - **Android application security is based on a consent approach that limits access to important resources on an Android device by third-party Android apps. Before proceeding with the installation, the user must approve the set of rights that the programmer needs. This process aims to inform users of the risk of trying to install and using an implementation on their device; however, even when the permission system is well comprehended, users are often unaware of the threat posed, and instead trust the app store or the prominence of the app, and acknowledge the insertion without questioning the developer's motivations. Machine learning and Deep learning classifiers are increasingly being used to categories malware based on permissions, either separately or associatively. The goal of this research is to look at strategies for characterization and detection of malware in the literature based on the preceding elements. We do so by illustrating and describing the limits of previous research as well as potential future research topics.**

Keywords: Android applications, malware detection, permission-related APIs, random forests, software security

I INTRODUCTION

Permissions are the foundation of the Android security concept. Permission is a security feature that restricts access to a portion of the code or data on the device. The restriction is in place to safeguard sensitive data and code from being abused to distort or harm the user experience. Permissions are used to provide or deny access to APIs and resources that are restricted. The Android INTERNET permission, for example, is needed for applications to execute network communications; hence, the INTERNET permission restricts the establishing of a network connection. In order to read entries in a user's phonebook, an application must also have the READ CONTACTS permission. It is the responsibility of the developer to determine the permissions an application needs. Many users are unaware of what each permission entails and approve them without thinking, enabling the programme to access sensitive information about the user. Another fault is that the user cannot choose which rights to give and which to refuse. Numerous consumers will still approve the installation of an app even if it requests a questionable permission among many apparently valid ones. Android has surpassed iOS as the most popular operating system for smartphones and tablets, with an estimated market share of 70% to 80%. With 1 billion Android devices expected to be shipped in 2017 and over 50 billion cumulative app downloads since the first Android phone was introduced in 2008, cyber criminals have naturally turned their attention to mobile platforms. According to mobile security specialists, there was an astonishing growth in Android malware from 2012 to 2013, with the number of harmful applications found ranging from 120.000 to 718.000. Many efforts have gone into analysing the characteristics of smartphone platforms and their apps in the last decade in order to properly identify malware from official and third-party sources. Bouncer, a Google tool, checks applications for potentially harmful activity. Bouncer automatically tests applications submitted to the Android Market by running them in a virtual Android environment hosted on Google's cloud infrastructure. Although the amount of malware downloads has dropped after Bouncer was installed, this solution does not offer protection against newer attack methods. The permission system is used by the Android platform to limit application privileges in order to protect users' sensitive data. To access the privacy-relevant resources, an application must get a user's consent for the needed rights. As a result, the permission system was created to protect users from intrusive programmes, although its success is greatly dependent on the user's understanding of permission approval.

When an unfamiliar app is published, the system advises checking the Play Store Marketplace to see whether it is dangerous. For each new application or updated version of an existing app, generate a risk score during runtime and categorise it according to a predefined threshold. Implementing a malware detection system in a real-time Android application environment is a success.

II LITERATURE REVIEW

Permissions necessary and requested are coupled with six additional characteristics from the manifest and the disassembled code in DREBIN [1]. Machine learning algorithms are used to understand the difference between dangerous and benign programmes automatically. Once trained offline on a dedicated machine, the Support Vector Machine is just transmitted the learnt model to the smartphone for identifying dangerous apps.

Huang et al. [2] investigate the feasibility of detecting fraudulent Android apps using permissions and 20 features from app bundles. According to their findings, a single classifier can identify around 81 percent of fraudulent programmes. It may be a fast filter to detect more suspect apps, according to them, by integrating findings from several classifiers.

Liu and Liu [3] use requested and necessary permissions by an application in a similar way. Machine learning algorithms and permissions are employed in this system to categorise an app as benign or harmful.

Sanz et al. [4] propose a novel approach for detecting fraudulent Android apps using machine learning methods and examining the application's extracted permissions. The existence of tags uses-permission and uses-feature in the manifest, as well as the amount of permissions granted to each application, are utilised to categorise them.

According to [5] is a way for building Machine Learning classifiers and detecting malware by extracting numerous properties from the Android manifest. These features are the specific permissions sought and the uses-feature>> tag.

Aung and Zaw [6] present a framework for developing a machine learning-based malware detection system for Android in order to identify malware apps and improve Smartphone users' security and privacy. This system collects numerous permission-based characteristics and events from Android apps and analyses them using machine learning classifiers to determine if the app is benign or malicious.

Shabtai et al. [7] divide Android apps into two categories: utilities and games. Successful separation between games and tools, in their opinion, should offer a good indicator of such systems' capacity to learn and model Android benign programmes and possibly identify malware files using Machine Learning (ML) techniques on static attributes collected from Android programme files.

The writers of these books limit their research to the most often requested permissions (or a certain selection of permissions) [8]. However, depending on the assault, permissions like READ LOGS might be just as dangerous as others (like INTERNET). Every permit should be carefully assessed as having the potential to be dangerous when paired with another. This method of selection, according to [9], produces considerably skewed results. Machine learning-based detection techniques are recognised to have two drawbacks: they have a high rate of false alarms, and choosing which characteristics should be learnt during the training phase is a difficult task. The procedure of picking datasets for training is therefore a crucial stage in these systems. The performance of the classifier improves with time: for a particular month Mi, whose apps were used for the training datasets, the resultant classifier becomes less and less capable of identifying all malware in subsequent months. Mk,k is greater than i.

To represent the applications, the majority of these efforts extract a feature set. The information conveyed by such characteristics varies depending on the job. There is no evidence to prove which attributes provide the greatest detection results, however each research takes needed permissions into account. Moonsamy et al. [10] are interested in using permissions as the sole feature to characterise programmes and identifying certain permission patterns to distinguish between clean and malicious apps.

Machine learning algorithms and permissions are used to identify an application as dangerous or benign, however only the data accessible to the user is examined before the programme is downloaded; the source code of the programmes is not considered. This implies that hidden flaws such as permission escalation attacks and capability leaks (explained in detail in [11]) cannot be identified.

III MOTIVATION

Many real-time APIs include harmful material as well as unlawful device access, making it difficult to revoke such access and recommending to the Play Store Marketplace if an unknown programme is harmful or not. For each new application or updated version of an existing app, generate a risk score during runtime and categorise it according to a predefined threshold. Implementing a malware detection system in a real-time Android application environment is a success.

IV PROPOSED SYSTEM DESIGN

The malware detection techniques are proposed by the system. We used I the permission ranking-based feature selection technique (ii) the similarity-based permission feature selection (iii) the association rule mining technique. The permission ranking-based feature selection strategy and the permission feature selection technique based on similarity rate the characteristics based on frequency. The permissions are deleted using the association rule mining technique, which is popular in malware and benign applications. Furthermore, we increase the deep learning algorithm's detection accuracy for permission-induced malware. We develop an enhanced RNN technique that

comprises fewer but more critical characteristics by comparing essential and non-essential characteristics.

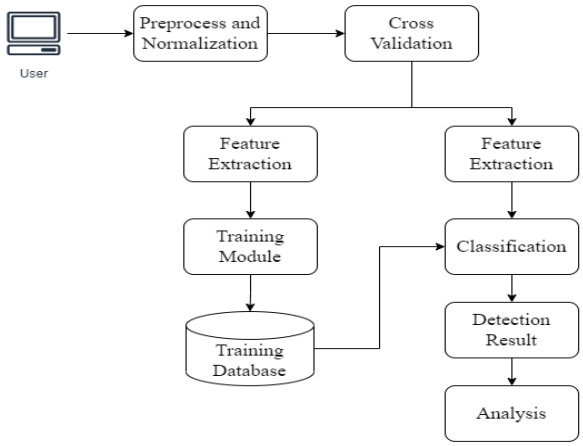

Figure 1: System Architecture

Problem Definition

In this research to design and implement real Android applications throughout the globe may mine secret malware patterns and extract extremely sensitive APIs that are extensively utilised in Android malware. We also use MalPat, an automated malware detection technology, to combat malware and help Android app markets deal with unknown dangerous apps.

Objectives

- To design developed an system for detect the malicious contents from third party API's which is generally used for android application development.
- To implement a machine learning or deep learning algorithm to mine the API codes.
- To validate the entire API's using background Knowledge which works like supervised learning approach.

Advantages

- Single Category features: The advantages of single category features are easy to extract, and low power computation. The limitations associated with this method are code obstruction, imitation attack and low precision.
- Multiple categories of Features: The advantages of multiple category features are easy to extract, and high accuracy.
- The main benefits of machine learning base analysis are to perform the highest accuracy as compared to static and dynamic analysis.

Limitations

- Highest complexity;
- Framework requirement to combine the static and dynamic features;
- More resource use; and
- Time-consumption.

Applications

- Android Malware classification systems
- Android bug tracker for API's and web services
- SecureRank Application for finding malicious API's.

Algorithm

A recurrent neural network (RNN) is a structure or piece of hardware that mimics the functions of every emotional brain. Recurrent neurons or processing components make up an artificial neural network. It is divided into three levels: input layer, concealed layer (which may include several layers), and output layer.

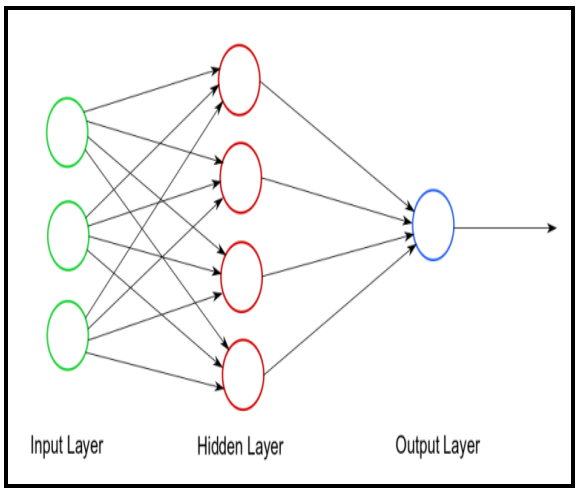

Figure 2: RNN Layers

Training Process

Input: Training dataset Train-Data [], Many activation functions [], Threshold Th

Output: Extracted Features Feature set[] for a trained module that has been finished.

Step 1: Set the data input block d[], the activation function, and the epoch size.

Step 2: Features-pkl ← Feature-Extraction (d[])

Step 3: Feature-set [] ← optimized (Features-pkl)

Step 4: Return Feature-set []

Testing Process

Input: Extracted features of testing instances set Data [i.....n], Train data policies PSet[41].....T[n]

Output: Normal or attack.

Steps:

1. For each (Data [i] into Data) choose n attributes from Data [i] using below formula,

$$\text{Treeset}(k) = \sum_{k=1}^{n} \text{attribute } [D[i]k \ldots . . D[n]n]$$

2. For each (PSet [i] from PSet),

$$\text{Train}[m] = \sum_{m=1}^{n} \text{attribute } [T[i]k \ldots . . T[n]n]$$

3. Evaluate train and test instances using below formula,

$$\text{Treeset}[k].\text{weight} = \text{similarity}(\text{Treeset}[k] \sum_{m=1}^{n} \text{Trainset } [n]$$

4. If (Treeset[k]: weight > Th),

Treeset[k].class ← Train[m]: class

Break;

5. Return Treeset[k].class

V RESULT

The implementation process was completed in a Java open-source setting. The device operates on the Java 3-tier analytics platform with a distributed INTEL 3.0 GHz i5 CPU and 4 GB RAM. Whether an email is spam or not has been determined uses the APK dataset. We have performed experiment analysis on ensemble machine implementation to verify the outcomes.

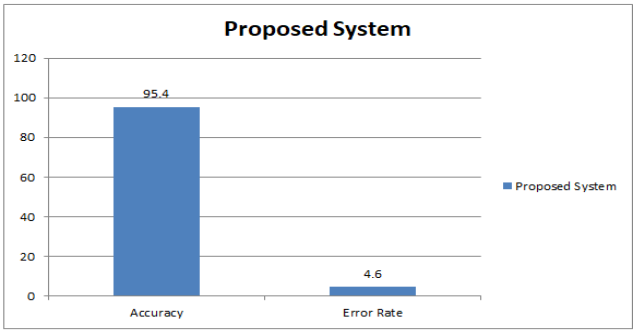

Figure 3: Accuracy of system analysis

Figure 6 shows the suggested system's classification accuracy and a comparison to several state-of-the-art systems. The figure above shows the detection accuracy of APK in malicious or not detection using different machine learning and deep learning classifications. The suggested classifier has been used to identify APK in malicious or not, with a high accuracy rate of up to 95.40%.

To ensure higher accuracy, our model was trained using many classifiers, which were then checked and compared. The user will get each classifier's assessed results. The user may compare the result with other results to determine if the data is APK in malicious or not when all classifiers have returned their findings to them. For easier comprehension, graphs and tables will be used to display each classification result.

Table I. Comparison Table

Classifiers	Accuracy	Precision	Recall	F-score
SVM	80.50	100	89.50	85.40
NB	81.40	100	87.40	82.40
RNN	95.40	96.50	94.10	97.15

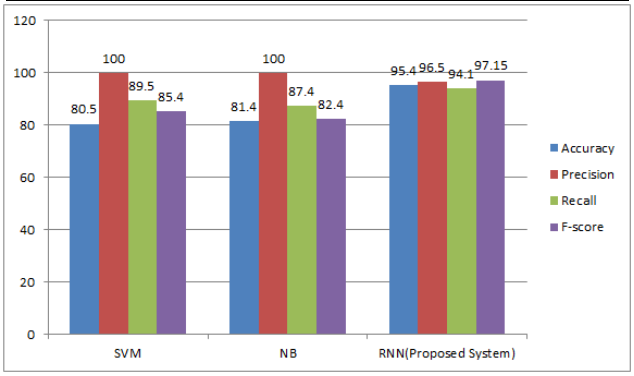

Figure 4: Comparison of Existing and Proposed System algorithms

VI CONCLUSION

The number of Android smartphones connecting to the Internet has recently increased. For connectivity and data sharing, the majority of these Android devices rely on Android Apps. The Android platform's permissions system limited the apps' access. Permission may be used as a feature in Android apps to distinguish between good and bad apps. Our efforts, on the other hand, lowered the number of permissions required to maintain accuracy and efficiency. In this project, harmful and benign Android applications are used in the real world to uncover hidden malware patterns. Previous research has mostly focused on permissions, sensitive resources, intents, and the like, with relatively few attempts to solve the malware detection issue from an API standpoint. To close this gap, we compare the behaviour of malicious and benign applications when it comes to API use. We can mine malware patterns and retrieve extremely sensitive APIs by training the random forests classifier with fine-grained features. We propose an automated malware detection method using a machine learning and deep learning algorithm to help Android app markets.

References

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective ¨ and Explainable Detection of Android Malware in Your Pocket," 2014.

[2] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," in Advances in Intelligent Systems and Applications-Volume 2, pp. 111–120, Springer, 2013.

[3] X. Liu and J. Liu, "A Two-Layered Permission-Based Android Malware Detection Scheme," in Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on, pp. 142–148, IEEE, 2014.

[4] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Alvarez, "Puma: Permission ´ usage to detect malware in android," in International Joint Conference CISIS12-ICEUTE´ 12-SOCO´

[5] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Alvarez Mara ´ n˜on, ´ "MAMA: Manifest Analysis for Malware Detection in Android," Cybernetics and Systems, vol. 44, no. 6-7, pp. 469–488, 2013

[6] Z. Aung and W. Zaw, "Permission-based android malware detection," International Journal Of Scientific & Technology Research, vol. 2, no. 3, 2013.

[7] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying Android applications using machine learning," in Computational Intelligence and Security (CIS), 2010 International Conference on, pp. 329–333, IEEE, 2010.12 Special Sessions, pp. 289–298, Springer, 2013.

[8] R. Sato, D. Chiba, and S. Goto, "Detecting Android Malware by Analyzing Manifest Files," Proceedings of the Asia-Pacific Advanced Network, vol. 36, pp. 23–31, 2013.

[9] K. Allix, T. F. D. A. Bissyande, J. Klein, and Y. Le Traon, "Machine Learning-Based Malware Detection for Android Applications: History Matters!," 2014.

[10] 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, 2014.