International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-10 Issue-5, June 2021

Blockchain Based Detection of Android Malware using Ranked Permissions

Siddhant Gupta, Siddharth Sethi, Srishti Chaudhary, Anshul Arora

Abstract: ***Android mobile devices are a prime target for a huge number of cyber-criminals as they aim to create malware for disrupting and damaging the servers, clients, or networks. Android malware are in the form of malicious apps, that get downloaded on mobile devices via the Play Store or third-party app markets. Such malicious apps pose serious threats like system damage, information leakage, financial loss to user, etc. Thus, predicting which apps contain malicious behavior will help in preventing malware attacks on mobile devices. Identifying Android malware has become a major challenge because of the ever-increasing number of permissions that applications ask for, to enhance the experience of the users. And most of the times, permissions and other features defined in normal and malicious apps are generally the same. In this paper, we aim to detect Android malware using machine learning, deep learning, and natural language processing techniques. To delve into the problem, we use the Android manifest files which provide us with features like permissions which become the basis for detecting Android malware. We have used the concept of information value for ranking permissions. Further, we have proposed a consensus-based blockchain framework for making more concrete predictions as blockchain have high reliability and low cost. The experimental results demonstrate that the proposed model gives the detection accuracy of 95.44% with the Random Forest classifier. This accuracy is achieved with top 45 permissions ranked according to Information Value.***

Keywords: Blockchain, Intrusion Detection, Mobile Malware, Mobile Network, Mobile Security.

I. INTRODUCTION

There are millions of Android users in the world and a great subset fall prey to the malware present in applications that they use in their day-to-day lives. Android, due to its open-source nature has fallen prey to many malware attacks. Malware detection is extremely sought for, considering the increasing attacks on the open-source Android platform. Citing the example of one such Android application, *CamScanner*, one of the most used Android applications globally, was alleged by a team of Kaspersky researchers to possess malicious traits in several of its modules and libraries [1]. Android malware pose serious threats such as system damage, leakage of information stored on the device, financial loss to the users, etc. According to Kaspersky [2], more than 5 million malicious applications were detected on the Android platform. Hence, keeping these threats in mind, in this paper, we attempt to present a machine learning-based framework in the Blockchain environment to detect malicious behavior in such Android applications.

Contributions: The main contributions of this work are summarized below:

1. We ranked the permissions using the concept of information value and weight of evidence to identify the useful and distinguishing permissions that can efficiently detect Android malware.
2. Thereafter, we converted the permissions obtained from each application into a sentence by concatenating them and separating them by spaces. TF-IDF was applied to the above-formed sentences to convert them to numeric values.
3. We further applied machine learning algorithms namely Random Forest, Naive Bayes, and Extremely Randomized Trees on the above-obtained numeric values.
4. We also applied deep learning architecture using a word embedding and LSTMS on the set of numeric permissions.
5. Lastly, a blockchain architecture was developed based on consensus to give better and accurate predictions to the problem described before.

Organization: The remainder of the paper is structured as follows. We discuss the related work in the field of Android malware detection in Section II. The detailed methodology of the proposed work is discussed in Section III. We review the results obtained from the proposed model in Section IV and conclude with future work directions in Section V.

II. RELATED WORK

We discuss the related work in two subsections. First, we review the blockchain-related works, i.e., the applications/areas in which blockchain has been applied. Next, we discuss the related works in the field of Android malware detection. We discuss these related works in the upcoming two subsections.

Manuscript received on May 07, 2021.
Revised Manuscript received on May 15, 2021.
Manuscript published on June 30, 2021.
* Corresponding Author
Siddhant Gupta*, Discipline of Mathematics and Computing, Delhi Technological University, Delhi, India, Email: siddhant1999gupta@gmail.com
Siddharth Sethi, Discipline of Mathematics and Computing, Delhi Technological University, Delhi, India, Email: sid.sethi31@gmail.com
Srishti Chaudhary, Discipline of Mathematics and Computing, Delhi Technological University, Delhi, India, Email: srishtic27@gmail.com
Anshul Arora, Discipline of Mathematics and Computing, Delhi Technological University, Delhi, India, Email: anshul15arora@gmail.com

A. Blockchain Related Works

Cui et al. [3] proposed a blockchain-based multi-WSN (Wireless Sensor Network) authentication scheme for distributed IoT systems. Hasan et al. [4] proposed a blockchain-based creation process of DTs(Digital Twins). DT is a digital representation of a real-world physical component product or equipment. The proposed model aimed to guarantee secure and trusted traceability, accessibility, and immutability of transactions, logs, and data provenance. The authors in [5] constructed a novel secure mutual authentication system that can be applied in smart homes and other applications by integrating blockchain, group signature, and message authentication code to provide reliable auditing. The authors in [6] proposed blockchain-based smart parking with fairness, reliability, and privacy protection. The group signatures, bloom filters, and vector-based encryption were leveraged to protect the users' privacy. The authors in [7] proposed a new data governance fashion that was built upon the blockchain-based decentralized services computing paradigm. The core principle was that data owners should be able to publish their data as a set of services that can be deployed independently from the application systems where the data were born. The authors in [8] proposed a Peer-to-Peer (P2P) energy trading scheme for a Virtual Power Plant (VPP) by using Smart Contracts on the Ethereum Blockchain Platform. Jaiman et al. [9] proposed a blockchain-based data-sharing consent model for access control over individual health data. They used smart contracts to dynamically represent the individual's consent over health data and to enable data requesters to search and access those data. The dynamic consent model extended to two ontologies: the Data Use Ontology (DUO) which modeled the individual consent of users and the Automatable Discovery and Access Matrix (ADA-M), which described queries from data requesters.

Guo et al. [10] proposed a blockchain-inspired event recording system for autonomous vehicles. They designed the mechanism of "Proof-of-Event"' with a dynamic federation consensus to achieve indisputable accident forensics by providing trustable and variable event information. They proposed a dynamic federation consensus scheme to verify and confirm the new block of event data in an efficient way without any central authority. The authors in [11] proposed a new voting protocol based on the blockchain technology, a new encryption mechanism to guarantee that nobody can decrypt the votes, but everyone can verify the validity of the votes as well as the outcome of the tallying process by using the homomorphic property of the encryption. This ensures the validity of the submitted votes in the counting process and at the same time maintains confidentiality. The authors in [12] proposed an Intelligent Transportation Systems(ITS) oriented, seven-layer conceptual model for blockchain and address the key research issues in B2ITS. Blockchain is considered one of the secured and trusted architectures for building newly developed parallel transportation management systems. The authors in [13] proposed a comprehensive concept, market design, and simulation of a local energy market between 100 residential households with the approach of a distributed information and communication technology using private blockchain which underlines the distributed nature of local markets. Turkanovic et al. [14] proposed a global higher education credit platform named EduCTX which is based on the concept of the European Credit Transfer and Accumulation System(ECTS). Based on a globally distributed peer-to-peer network, EduCTX processed, managed, and controlled ECTX tokens, which represent credits that students gain for completed courses, such as ECTS. Higher Education Institutions(HEI's) were the peers of the blockchain network. The authors in [15] proposed a Blockchain-based digital content distribution system that has a decentralized and peer-to-peer authentication mechanism that can be considered as an ideal rights movement mechanism.

B. Android Malware Detection

In this subsection, we discuss the works proposed in the literature for Android malware detection. Broadly, detection mechanisms for Android malware are divided into three categories, namely, Static, Dynamic, and Hybrid, depending upon the features used for detection. We discuss these techniques next.

Static Detection

Static Techniques aim to detect malicious Android apps without running the apps. Such techniques analyze static components like permissions, intents, Java source code, etc., where the apps are not required to be executed on smartphones. Several static-based detection works have been reported in the literature. The authors in [16] calculated a specific score of each permission in which the number of malware samples containing that particular permission was divided by the total number of malware present in the dataset. This permission score was used to detect malicious applications. Moonsamy et al. [17] analyzed the harmful patterns of permissions defined within the malicious applications. The authors in [18] used Information Gain and T-Test to rank and further detected malware using machine learning algorithms. Idrees et al. [19] detected malware by identifying the significant intents as well as permissions that are defined in malicious samples and normal applications. The authors in [20] identified the similarity between malware and non-malware applications by applying Hamming Distance using the static features of permissions, intents, and APIs. The authors in [21] applied a linear SVM algorithm on the static features extracted from manifest files to detect the malware applications. Similarly, authors in [22] have also used various static features including manifest components, and have applied machine learning algorithms for detection. In [23], the authors ranked permissions and intents for detecting malware. Sanz et al. proposed a model named MAMA [24] which used various manifest file components like hardware components and permissions for detecting malware. Similarly, authors in [25] and [26] detected malicious applications using various machine learning algorithms on static features. Authors in [27] proposed Apposcopy that used API calls to analyze the control flow and data flow properties to detect malware. Wang et al.[28] used string features such as permissions and intents along with API calls and their function call graphs to detect malware in Android applications. The authors in [47] analyzed permissions in pairs for malware detection.

None of the above-discussed words have aimed to use the Blockchain platform on the ranked permissions for effective Android malware detection. In this work, novel from others, we have applied the Blockchain model on ranked permissions to detect Android malware.

Dynamic Detection

Solutions involving static detection mechanisms might not be able to detect the malicious component primarily because they do not execute the applications, and by not doing so, while updating, these malicious components might be downloaded. This called for the proposition of dynamic solutions for detecting malware in Android by the researchers. Some research works around the detection of malicious apps which used Android OS-based features are as follows. In [29], analysis of systems calls and API was done to identify malicious Android apps. In [30], malware detection was done by Shabtai et al. using dynamic features like the percentage of Central Processing Unit usage, active processes, along with the number of network traffic packets being communicated. In [31], it was deduced that Java, JNI, or native code execution initiated the potential source of unwanted malicious behavior in the malware applications by observing system calls of malicious Android apps. In [32], system calls of apps were analyzed to differentiate benign from malware ones. In [33], malicious apps were detected by Iqbal et al. by analyzing several dynamic features such as memory consumption, CPU usage, and system call events.

Now, we review the related works that have used Internet traffic features for Android malware detection. In [34], the existing traffic patterns present in benign apps were analyzed. Further, to figure out deviations and variations from benign traffic patterns in the malicious traffic patterns, the authors used various machine learning classification techniques. In [35], malware detection was done by Wang et. al by applying Natural Language Processing techniques on the HyperText Transfer Protocol (HTTP) headers. In [36], malicious activity in the network traffic was detected by extracting network-level features and applying multiple classifiers. In [37], malware network traffic was detected by Igor et al. by observing patterns of 14 features from the TCP / IP headers of the malicious and normal files. The authors in [48] and [49] analyzed several network traffic features for malware detection in Android.

The dynamic features extraction, for example, system calls and network traffic, is computationally complex and has huge overheads as compared to static techniques, therefore, in this paper, only a static detection model is presented.

Hybrid Detection

To combine the advantages of both static and dynamic solutions, few hybrid solutions exist in the literature wherein both static and dynamic features can be combined to propose a hybrid detection model. In [38], the authors analyzed the components of the manifest files such as permissions and dynamic features of run-time Dalvik code loading for malicious Android apps detection. In [39], machine learning techniques were applied after extracting important features like apps rating, dynamic API calls, permissions, number of users who downloaded the app. In [40], the authors presented the 'AdDroid model' that detects malicious activities on a device by analyzing Android actions such as uploading of a file to a server, internet connections, installing packages on the device, etc. In [41], Android malware was detected by observing run-time-related events along with sensitive APIs and permissions by Zhu et al. Apart from this, the authors in [42] and [43] proposed two absolutely different hybrid detection techniques by merging and combining the permissions with network traffic features. On similar lines, the authors in [44], [45], and [46] worked on malware detection by analyzing the combination of static and dynamic features.

Since hybrid detection involves both static as well as dynamic features, they involve computational overheads too, similar to dynamic mechanisms. Therefore, we have aimed to propose a static detection in our model.

III. PROPOSED METHODOLOGY

In this section, we present the proposed approach for detecting malicious Android apps. We obtained Android APK files, extracted permissions from the AndroidManifest.xml files, and used them as a distinguishing feature to characterize and differentiate malware files from non-malware files using machine learning and deep learning techniques. Further, we have boiled down the problem to a natural language processing task by creating sentences from the extracted permissions for each file and concatenating them which are further used to train the machine learning and deep learning models. We obtained the non-malware files (in APK format) from apkpure.com and malware files (also in APK format) from Genome, Drebin, and Koodous. Then, we ran a python script to extract permissions from the Android manifest files which were used as a feature for building the predictive model. We discuss this methodology in detail in the upcoming sections.

A. Dataset Construction

From the applications in the APK format, we extracted the Android manifest file using apktool which contains information such as permissions the application wants from the user, the intents of the applications, etc. Permissions are certain accesses that the application asks the user to grant in order to function. Some of them are as follows:·a). Access to photos in the device, b) Access to use location services, c) Access to use the browser, etc. Figure 1 summarizes one of the instances of permissions present within the manifest file of an app.

Fig. 1: A snippet of the Android manifest file from which permissions were extracted.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

Permissions extracted from the APK files, obtained as described above, were used as a sole distinguishing feature between the 2 classes (malware and non-malware). We had a total of 477 permissions. For each file, from the various permissions obtained, we constructed a sentence by concatenating them and separating them by spaces. We have a total of 2853 samples with a number of non-malware and malware files being 1451(in blue) and 1402(in orange) respectively as shown below. Thus, after doing this for each file, we obtained a data frame of 2853 rows and 2 columns, one for the permission sentences and the other for the labels (1 for malware and 0 otherwise). Figure 2 summarizes a data frame showing a few rows.

	permissions	labels
2738	internet write_external_storage mount_unmount_...	1
1041	internet access_network_state read_phone_state...	0
834	receive_boot_completed call_phone read_contact...	0
547	internet write_external_storage write_settings...	0
401	access_network_state internet get_accounts wak...	0
...	...	...
1683	read_phone_state access_network_state send_sms...	1
633	access_network_state internet write_external_s...	0
2017	internet access_fine_location	1
1705	send_sms receive_sms internet receive_boot_com...	1
2176	access_network_state access_wifi_state change_...	1

2853 rows × 2 columns

Fig. 2: A snippet of the data frame showing a few rows.

B. Permissions Ranking With Information Value

Information Value, abbreviated as IV, is closely connected with Weight of Evidence (WOE). This method evolved from logistic regression to select variables for building the base model and turned out to be very useful for the binary classification of APK files under consideration. We used information value as a metric to rank the most important permissions that enabled us to distinguish between the two classes, the malware files from the non - malware ones. The following equation defines the concept of Information Value.

$$IV = \sum(\%non - events - \%events) \times ln\left(\frac{\%non - events}{\%events}\right)$$

Non-malware files are represented by non-events and malware files are represented by events. Variables whose IV comes out to be more than 0.5 can be considered as those variables which have a very strong impact on determining which of the two classes a data point will belong to, and are thus given higher priority during model development. Such strong predictors can significantly enhance the accuracy of the model and prove to be extremely useful.

C. Machine Learning Framework for Classification

TF - IDF stands for Term Frequency-Inverse Document Frequency, a fairly popular statistical measure associated with natural language processing that gives information about the importance of a word in a corpus. The permissions' sentences cannot be fed to machine learning algorithms as it is in the form of strings. To convert the permissions feature to numeric form, we encode them using the TF - IDF technique. For each permission, we found its TF - IDF score.

Term Frequency

It is defined for a word taking into account its corresponding document. Thus,

$$TF = \frac{Number\ of\ times\ a\ word\ appears\ in\ a\ document}{Number\ of\ words\ in\ the\ document}$$

Inverse Document Frequency

It is defined for a word taking into account the entire corpus. Thus,

$$IDF = log\left(\frac{Total\ number\ of\ documents}{number\ of\ documents\ that\ contain\ the\ word}\right)$$

Thus,

$$TF - IDF = Term\ Frequency \times Inverse\ Document\ Frequncy$$

TF - IDF was chosen in our proposed model because of the following reasons:

1. It gives more importance to rarer words in the corpus, i.e., IDF will be high in this case.
2. It gives more importance if a word is frequent in a particular document, i.e., TF will be high in this case.

After obtaining the encoded data, it was fed to the machine learning models. As the data was fairly balanced, accuracy was used as a metric to evaluate the performance of models.

D. Machine Learning Classifiers

In our model, we applied three machine learning classifiers namely, Naïve Bayes, Random Forest, and Extremely Randomized Trees. The following discussion gives details on three classifiers.

Naïve Bayes: Naïve – Bayes is a probabilistic, supervised machine learning algorithm that is used for classification problems by applying the famous Bayes theorem. The assumption is that every pair of features is conditionally independent, and the value of the class variable is known. This conditionally independent nature makes the computations and consequently the entire algorithm a lot simpler, but still capable of generating powerful results. It is closely associated with Natural Language Processing in the context of text classification in domains such as sentiment analysis and spam detection and filtration. The Bayes theorem determines the probability of the occurrence of an event under the condition that another related event has already occurred prior to the event under consideration and the probability of the occurrence of the other event is known. The basic governing formula is given below:

$$P\left(\frac{A}{B}\right) = P\left(\frac{B}{A}\right) \times \frac{P(A)}{P(B)}$$

In a nutshell, we used the Maximum A Posteriori (MAP) estimation to determine which class does the data point under consideration most likely belongs to.

Random Forest: Random Forest is a very popular and efficient ensemble model which uses Decision Trees as its base model. Decision Trees are one of the most highly interpretable algorithms as they seem to be just like nested if-else statements and therefore are highly favorable while solving machine learning problems. But Decision Trees of reasonable depth are prone to overfitting which leads to bad results on testing data. Also, they exhibit low bias and are characterized as high variance models. Random forests are good in preserving the low bias and reducing the high variance, thus, making the final ensemble low bias and low variance. Random forest achieves this by a technique called Bagging (Bootstrap-Aggregation). In this, we construct datasets that are subsets of the original dataset using bootstrap sampling with replacement and train each decision tree of reasonable depth with different subsets. We then aggregate the result from all the decision trees by either majority voting which is in the case of classification task or by taking the mean of the predictions which is in case of regression. In our hyperparameter tuning, we have used the number of iterators or decision trees as 200 and we have allowed the trees to grow to maximum depth. The random state was set to 5.

Extremely Randomized Trees The core idea behind this variation of Random Forest is to further assist in reducing variance by randomizing the process. In most cases, it works better than random forest as it takes a step further from the random forest. In a random forest, a decision tree tries to split each value of a numerical feature by sorting the feature which is considerably time-consuming. When working with extra tree classifiers, we select the samples uniquely, without replacement for the decision trees which is not the case in random forests. Since extremely randomized trees do not split at each value, but a subset of those values, they randomize the process even further consequently, thus, reducing the variance significantly. Unlike random forests, it does not take substantial time to choose the best split. A random splitting point is chosen which further helps in reducing the variance. The catch here is that it may increase a slight bias at times. The estimators used were 200 and the random state was set to 100.

E. Deep Learning Framework for Classification

The permission sentences were converted to sequences and to make same length documents they were padded with zeros using pre-padding. The max length was taken to be 13 which was the length of the largest sequence, hence no truncation was performed. Then, the above-obtained sequences were passed through a word embedding layer to convert the permissions to word vectors. The dimension of a vector representing each permission was taken to be twelve and pre-trained word vectors were not used, instead, the word embedding layer weights were taken to be random and the layer was set to trainable during model training.

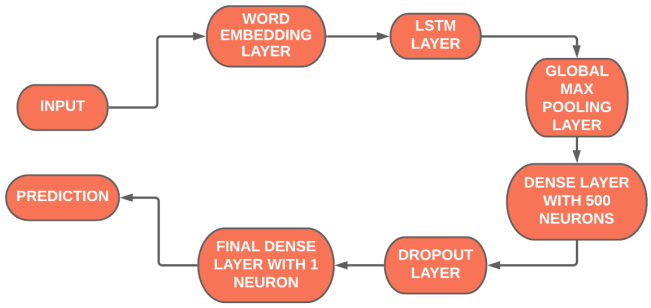

Fig. 3 : A flowchart illustrating the deep learning model flow.

Figure 3 highlights the flowchart of deep learning model flow. The first layer consists of the input layer followed by the word embedding layer as described above. Then an LSTM layer is used with the number of units (output dimension) taken as 8 and return sequences set to true. LSTM performs much better than the standard Elman RNN unit (simple RNN) as it can learn long dependencies and prevent the vanishing gradient problem. LSTM layer is followed by the global max-pooling layer. Further, a dense layer of 500 neurons is used with an activation function set to ReLU. To avoid overfitting, a dropout layer is added with the fraction of neurons in the previous layer to drop set to 0.4. The last layer is the final dense layer with an activation function as sigmoid.

F. Implementation on Blockchain

A node initially takes the input file from the central server which it receives from the user and then sends this file to the peer-to-peer (P2P) local network which contains other nodes. Each node will be given a unique predictive model to perform the classification task. Each node having its machine learning model tries to classify whether the current file is malignant or benign. The network then comes to a consensus regarding whether the file is malicious or not. This begins with each node adding their predicted probabilities to the blockchain as a transaction after hashing with its key. This is followed by the node which broadcasts the file traversing the blockchain and taking a weighted average of the predicted probabilities, thus, obtaining the final verdict. The weights will be equal to the accuracy score of the models of each node. A unique predictive model will be given to any new incoming node in the network by the central server. The accuracy score of a node's model acts as a measure of the node's trust.

Using the blockchain ensures that the information or data stored doesn't get tampered with. If the information regarding a node is changed, then it will change the hash code for the node. The previous node stores the hash code for the next node so that the chain-like structure is maintained. If the current node's hash code gets changed then it will not match with the previous node and would be considered invalid. If a modification is suggested by an unauthorized source to the blockchain in a node, it will be dismissed instantly since the hash of the block will vary as compared to the subsequent ones. Thus, one cannot tamper with the results produced by the modes.

IV. RESULTS AND DISCUSSION

In this section, we review the results obtained from the proposed model. Firstly, we highpoint the top-ranked permissions obtained from the Information Value. Table 1 summarizes the top 5 permissions ranked with the Information Value. As can be seen from the table, "Receive" permission is the top-ranked permission with a larger information value. Top-ranked features help in improving the detection accuracy of the proposed model.

Table 1: Top Ranked Permissions

Permission Name	Information Value
Receive	2.202
C2D Message	1.689
Get Tasks	1.109
Use Credentials	0.848
Read Phone State	0.805

A. Detection Results With Machine Learning and Deep Learning Algorithms

Now, we present the detection results of our proposed approach. Table 2 summarizes the detection results when we use the Naïve Bayes algorithm for detection. We have used a different set of permissions at each iteration. As the table summarizes, we get an accuracy of 62.12% when we use the top 10 ranked permissions sorted with Information Value. Similarly, other entries of the table can be understood. We observe that we get the highest accuracy of 91.59% with the set of top 45 ranked permissions. When we further increase the number of permissions, the detection accuracy did not increase. Hence, we can argue that we get the best accuracy of 91.59% with the Naïve Bayes classifier.

Table 2: Detection Results with Naïve Bayes

Number of Permissions Ranked According to IV	Detection Accuracy (in %)
10	62.12
15	71.20
20	76.14
25	81.00
30	84.10
35	87.57
40	89.24
45	91.59

Table 3 summarizes the results when we use the Random Forest classifier for detecting malicious Android apps. Again, we review the results with the different number of top-ranked permissions. We observe that we get the highest accuracy of 95.44% on top 45 ranked permissions. Similar to the Naïve Bayes classifier, we get the highest accuracy with the top 45 ranked permissions and the accuracy does not increase further when we increase the number of permissions.

Table 3: Detection Results with Random Forest

Number of Permissions Ranked According to IV	Detection Accuracy (in %)
10	70.27
15	75.87
20	82.14
25	86.02
30	89.14
35	91.21
40	93.87
45	95.44

Table 4 highlights the detection results when we use Extremely Randomized Trees for Android malware detection. Similar to previous results, we get the highest accuracy of 95.09% with top 45 ranked permissions. Again, the accuracy did not increase when we include more permissions in the analysis.

Table 4: Detection Results with Extremely Randomized Trees

Number of Permissions Ranked According to IV	Detection Accuracy (in %)
10	70.00
15	74.92
20	81.23
25	85.54
30	88.40
35	90.12
40	94.14
45	95.09

Table 5 highlights the detection results when we use LSTM (Long Short Term Memory) algorithm for Android malware detection. Similar to previous results, we get the highest accuracy of 95.27% with top 45 ranked permissions. Again, the accuracy did not increase when we include more permissions in the analysis.

Table 5: Detection Results with LSTM Algorithm

Number of Permissions Ranked According to IV	Detection Accuracy (in %)
10	70.94
15	76.57
20	82.14
25	86.13
30	88.90
35	91.31
40	94.90
45	95.27

Summary: If we compare Tables 2, 3, 4, and 5, we observe that the Random Forest classifier gives the highest accuracy of 95.44% as compared to other classifiers. All the classifiers give their best results with top 45 ranked permissions. Further increasing the number of permissions did not improve the detection accuracy. Hence, we can argue that ranking the permissions with the concept of Information Value is useful in effectively detecting Android malware.

B. False Results Analysis

We observe that few normal samples downloaded from Play Store were detected as malicious by the proposed model, hence leading to false positives. The reason lies in the fact that these apps were related to blocking the phone calls / SMS on the device. Numerous apps exist in the Play Store that aim to block incoming phone calls / SMS when installed on the mobile device. Because this functionality matches with the behavior of malicious apps, hence, these samples are falsely detected as malware. On the other hand, few malware samples like AnserverBot, Geinimi, Plankton, etc., get detected as normal, hence, leading to false negatives.

Some of these malicious samples contained a very less number of permissions, i.e., only 1 or 2 permissions within their manifest file.

Hence, it becomes difficult to detect such malicious samples with a low number of permissions. Moreover, few malware samples are stealthier in the way that they download malicious components at update time. Permissions-based static techniques cannot detect such stealthier samples. Samples such as BaseBridge download malicious components at update time, hence, are undetected by the permissions-based approach. Keeping these challenges in mind, in our future work, we will target to incorporate dynamic features as well for analysis like system calls and network traffic. We will also look to implement our approach on recent and stealthier Android malware samples.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a static model to detect Android malware based upon permissions analysis in the blockchain environment. There are numerous permissions in Android, hence, we aimed to rank the permissions based upon the Information Value. Such ranking helps in eliminating the irrelevant permissions for improving the detection accuracy. Further, we applied the machine learning and deep learning algorithms on the top-ranked permissions. The experimental results demonstrated that the proposed model was able to achieve an accuracy of 95.44% with top 45 ranked permissions. In our future work, we will aim to include more static and dynamic features for analysis, such as other manifest file components, system calls, and network traffic, CPU and memory usage, etc.

REFERENCES

1. Malicious Android app had more than 100 million downloads in Google Play. Available Online. https://www.kaspersky.co.in/blog/camscanner-malicious-android-app/16595/
2. Mobile malware evolution 2020, Available Online. https://securelist.com/mobile-malware-evolution-2020/101029/
3. Z. Cui *et al*., "A Hybrid BlockChain-Based Identity Authentication Scheme for Multi-WSN," in IEEE Transactions on Services Computing, vol. 13, no. 2, pp. 241-251, 2020.
4. Hasan et al. , "A Blockchain-Based Approach for the Creation of Digital Twins", IEEE Access, vol. 8, pp. 34113-34126, 2020.
5. C. Lin, D. He, N. Kumar, X. Huang, P. Vijayakumar and K. R. Choo, "HomeChain: A Blockchain-Based Secure Mutual Authentication System for Smart Homes," IEEE Internet of Things Journal, vol. 7, no. 2, pp. 818-829, 2020.
6. C. Zhang et al., "BSFP: Blockchain-Enabled Smart Parking With Fairness, Reliability and Privacy Protection," IEEE Transactions on Vehicular Technology, vol. 69, no. 6, pp. 6578-6591, 2020.
7. X. Liu, S. X. Sun and G. Huang, "Decentralized Services Computing Paradigm for Blockchain-Based Data Governance: Programmability, Interoperability, and Intelligence," IEEE Transactions on Services Computing, vol. 13, no. 2, pp. 343-355, 2020.
8. S. Seven, G. Yao, A. Soran, A. Onen, and S.M. Muyeen "Peer-to-Peer Energy Trading in Virtual Power Plant Based on Blockchain Smart Contracts" IEEE Access, vol. 8, pp. 175713-175726, 2020.
9. V. Jaiman, and V. Urovi, "A Consent Model for Blockchain-Based Health Data Sharing Platforms" , IEEE Access, vol. 8, pp. 143734-143745, 2020.
10. H. Guo, W. Li, M. Nejad and C. C. Shen, "Proof-of-Event Recording System for Autonomous Vehicles: A Blockchain-Based Solution", IEEE Access, vol. 8, pp. 182776-182786, 2020.
11. X. Yang, X. Yi, S. Nepal, A. Kelarev and F. Han, "Blockchain voting: Publicly verifiable online voting protocol without trusted tallying authorities", Future Generation Computer System, vol. 112, pp. 859-874, 2020.
12. Y. Yuan and F. Wang, "Towards blockchain-based intelligent transportation systems," in IEEE 19th International Conference on Intelligent Transportation Systems, pp. 2663-2668, Brazil, 2016.
13. E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer and C. Weinhardt, "A blockchain-based smart grid: towards sustainable local energy markets", Computer Science-Research and Development, vol. 33, pp. 207-214, 2018.
14. M. Turkanovic, M. Holbl, K. Kosic, M. Hericko, and A. Kamisalic, "EduCTX: A Blockchain-Based Higher Education Credit Platform", IEEE Access, vol. 6, pp. 5112-5127, 2018.
15. J. Kishigami, S. Fujimura, H. Watanabe, A. Nakadaira and A. Akutsu, "The Blockchain-Based Digital Content Distribution System," IEEE Fifth International Conference on Big Data and Cloud Computing, pp. 187-190, China, 2015.
16. K. Talha, D. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system", Digital Investigation, vol. 13, pp. 1- 14, 2015.
17. 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.
18. W. Wang et al., "Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection", IEEE Transactions on Information Forensics and Security, vol. 9, pp. 1869-1882, 2014.
19. F. Idrees, and M. Rajarajan, "Investigating the Android Intents and Permissions for Malware detection", 7th International Workshop on Selected Topics in Mobile and Wireless Computing, 2014.
20. R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features", Future Generation Computer Systems, vol. 105, pp. 230-247, 2020.
21. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket",NDSS,2014.
22. H. Fereidooni, M. Conti, D. Yao and A. Sperduti, "ANASTASIA: Android mAlware detection using STatic analySIs of Applications," 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Larnaca, 2016.
23. K. Khariwal, J. Singh and A. Arora, "IPDroid: Android Malware Detection using Intents and Permissions," 4th World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, United Kingdom, pp. 197-202, 2020.
24. B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. Bringas, and G. Lvarez, "MAMA: manifest analysis for malware detection in android", Cybernetics and Systems, vol. 44, pp. 469-488, 2013.
25. S. Feldman, D. Stadther and B. Wang, "Manilyzer: Automated Android Malware Detection through Manifest Analysis," IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems, Philadelphia, PA, pp. 767-772, 2014.
26. M. Kumaran and W. Li, "Lightweight malware detection based on machine learning algorithms and the android manifest file," IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, pp. 1-3, 2016.
27. Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics based detection of android malware through static analysis", 22nd ACM SIGSOFT Symposium on Foundations of Software Engineering, 2014.
28. W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu and X. Zhang, "DroidEnsemble: Detecting Android Malicious Applications With Ensemble of String and Structural Static Features," IEEE Access, vol. 6, pp. 31798-31807, 2018.
29. V.M. Afonso et al., "Identifying Android malware using dynamically obtained features", Journal of Computer Virology and Hacking Techniques, vol. 11, pp.9-17,2015.
30. A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: a behavioral malware detection framework for android devices", Journal
of Intelligent Information Systems, vol. 38, pp. 161-190, 2012.
31. A. Reina, A. Fattori, and L. Cavallaro, "A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors", 6th European Workshop on System Security, 2013.

32. M. Jaiswal, Y. Malik and F. Jaafar, "Android gaming malware detection using system call analysis," 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, pp. 1-5, 2018.
33. S. Iqbal and M. Zulkernine, "SpyDroid: A Framework for Employing Multiple Real-Time Malware Detectors on Android," 13th International Conference on Malicious and Unwanted Software (MALWARE), Nantucket, MA, USA, pp. 1-8, 2018.
34. A. Shabtai et al., "Mobile malware detection through analysis of deviations in application network behavior", Computers & Security, vol. 43, pp. 1-18, 2014.
35. S. Wang, et al., "Detecting Android Malware Leveraging Text Semantics of Network Flows", IEEE Transactions On Information Forensics And Security, vol. 13, pp. 1096-1109, 2018.
36. J. Feng, L. Shen, Z. Chen, Y. Wang and H. Li, "A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic," IEEE Access, vol. 8, pp. 125786-125796, 2020.
37. I. J. Sanz, M. A. Lopez, E. K. Viegas and V. R. Sanches, "A Lightweight Network-based Android Malware Detection System," IFIP Networking Conference (Networking), Paris, France, pp. 695-703, 2020.
38. M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection", 10th ACM Mobisys, 2012.
39. A. Mahindru, A. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques", Neural Computing & Applications, 2020.
40. A. Mehtab et al., "AdDroid: Rule-Based Machine Learning Framework for Android Malware Analysis", Mobile Networks and Applications, vol. 25, pp. 180–192, 2020.
41. H. Zhu et al., "HEMD: a highly efficient random forest-based malware detection framework for Android," Neural Computing & Applications, vol. 30, pp. 3353–3361, 2018.
42. A. Arora, and S. Peddoju, "NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions", 17th IEEE TrustCom, 2018.
43. A. Arora, S. Peddoju, V. Chauhan, and A. Chaudhary, "Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learning", 24th ACM MobiCom, 2018.
44. S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song and H. Yu, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," IEEE Access, vol. 6, pp. 4321-4339, 2018.
45. Q. Fang, X. Yang and C. Ji, "A Hybrid Detection Method for Android Malware," IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, pp. 2127- 2132, 2019.
46. L. Taheri, A. F. A. Kadir and A. H. Lashkari, "Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls," International Carnahan Conference on Security Technology (ICCST), Chennai, India, pp. 1-8, 2019.
47. A. Arora, S.K. Peddoju, and Mauro Conti, "PermPair: Android Malware Detection using Permission Pairs", IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1968-1982, 2020.
48. A. Arora, and S.K. Peddoju, "Minimizing Network Traffic Features for Android Mobile Malware Detection", 18th ACM International Conference on Distributed Computing and Networking, Hyderabad, India, 2017.
49. A. Arora, S. Garg, and S.K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices", 8th IEEE Next Generation Mobile Apps Services and Technologies, Oxford UK, 2014.

AUTHORS PROFILE

Siddhant Gupta, is a final year student pursuing B.Tech in mathematics and computing at Delhi Technological University . he has keen interest in problem solving and machine learning specially in fields of natural language processing and computer vision. he has previously interned with KPMG and Snapdeal and will be joining Amazon as software development engineer post his graduation.

Siddharth Sethi, is a final year student pursuing B.Tech in Mathematics and Computing at Delhi Technological University. He has keen interest in machine learning and has enthusiastically participated in making the project. He is currently placed in Accenture as an Application Analyst.

Srishti Chaudhary, is a final year student at Delhi Technological University pursuing B. Tech in Mathematics and Computing. She has keen interest in problem solving and Machine Learning and has shown great dedication towards this project. She has interned with HMEL and Algo8 prior to finally getting placed at Microsoft as a Software Engineer.

Anshul Arora, is currently working as Assistant Professor in Discipline of Mathematics and Computing, Delhi Technological University Delhi, India. He has pursued Masters and Ph.D. from Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India. His areas of research include Mobile Security, Mobile Malware Detection, Network Traffic Analysis, and Blockchain.

Retrieval Number: 100.1/ijeat.E25930610521
DOI:10.35940/ijeat.E2593.0610521
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication