Detecting Malware on Android Using Machine Learning Techniques

P. Siva Srinivasa Rao[1],
PG Scholar
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GVR&S COLLEGE OF ENGINEERING AND TECHNOLOGY,
Budampadu-522017, Guntur (Dt), A.P., India.
Thrishiva123@gmail.com[1]
Dr. SK.Sajeedha Parveen[2]
Professor
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GVR&S COLLEGE OF ENGINEERING AND TECHNOLOGY,
Budampadu-522017, Guntur (Dt), A.P., India
shaiksajeedaparveen@gmail.com[2]

ABSTRACT:

The Android platform became one of the most vulnerable targets for cyberattacks in recent times due to a rapid surge in malware embedded apps. Researchers have investigated various machine learning techniques for Android malware detection but most of these techniques are inefficient against the novel malware. The various problems like code obfuscation, the requirement of device root privileges, simulated and small size datasets pose serious flaws to the existing solutions. This work evaluates several machine learning models for mitigating these issues using low privileged monitorable features sampled in the SherLock dataset. The findings of this research conclude that the XGBoost clas- sifier is the most accurate in detecting the malware compared to other classifiers with 93% overall values of precision, recall, and accuracy. In terms of FNR values, which sig- nify the undetected malware, the XGBoost classifier also performs better than the other algorithms with values of 7.0%.

Keywords: Android Malware Detection, Machine Learning, XGBoost Classifier, SherLock Dataset, Code Obfuscation, Low-Privileged Features, Cybersecurity, False Negative Rate (FNR), Malware Embedded Apps, Android Vulnerabilities, Precision, Recall, Accuracy, Root Privileges, Novel Malware

1. INTRODUCTION

In recent times, smartphones turn out to be an indispensable part of human life as most of the day to day life computation is shifting towards smartphones. The smartphones are equipped with a variety of sensors to provide several applications to the end-users and generates a huge amount of sensitive and confidential data. Android OS, due to its opensource distribution, emerged as blazing popularity for smartphones in the last few years. This predominant operating system platform has established itself not only in the mobile world but also in the Internet of Things devices and turns out to be the most common operating system for smartphones with a market share of 86.1% by the end of 2019 [1].

Malware can also attack an android device by performing a privilege escalation attack [2] by which an unauthorized person can gain control of the phone via the backdoor. Furthermore, malware can perform other attacks like phishing and ransomware as well as it can affect the device by draining the battery and infecting the network interface card. So, there is a requirement of efficient security mechanisms for malware detection in Android devices. Google play store follows a very simple security mechanism known as Bouncer [3]. It is a third-party program that continuously scans the google play store repository for identifying malicious apps. However, this may reduce the number of uploaded malicious apps but still, it fails to detect most of the vulnerable apps uploaded on the play store. The other security mechanisms such as the Android permission system, integrated with the Android OS, control the permission to access the resources by giving the individual permissions to apps statically at the time of installation. But the issue with the Android permission system is that almost all end-users blindly grant permission to apps during installation. Botha et al. [4] have applied the security mechanism used for PC to smartphones and claim that smartphones fail to perform well with these methods due to extensive resource utilization. Hence there is an increasing need for sophisticated, advanced, robust, and automated malware detection systems to detect malicious applications. Machine learning methods are very recent and well-established techniques for malware detection on the Android platform. Researchers have extensively classified the study of Android malware detection using machine learning techniques into two ways, static and dynamic analysis based on the features set acquired from the apps [5].

2. Literature survey

2.1 Android Platform Architecture

Android is an open-source software platform based on Linux, built for a wide range of devices and form factors. The key components of the Android platform is shown in be- low diagram. In this figure, violets items are modules written in native machine code (C/C++), while green items are modules interpreted and executed by the Dalvik Virtual Machine(DVM). The bottom red layer contains the components of the Linux kernel and executes in kernel space. Using a bottom-up approach, we briefly address the different abstraction layers in the following subsections.

2.2 The Linux Kernel

The Android platform is based on the Linux kernel. The Android Runtime (ART), for example, depends on the Linux kernel for basic functionalities, including multi-threading and low-level memory management. The Linux kernel helps Android to take advantage of core security features and encourages handset makers to build hardware drivers for well-known kernel. Android uses a sophisticated version with some special additions to the Linux Kernel. These includes wake-locks, a memory protection scheme that is more proactive in saving memory, the Binder IPC driver, and other functionality that are essential for a mobile embedded platform such as Android.

2.3 Native C/C++ Libraries

Many core components and services of the Android system, such as ART and HAL, are developed from native code, which require native libraries written in C and C++. Mostly these are external libraries with only slight improvements such as OpenSSL, WebKit and bzip2. The Android platform provides Java framework APIs to expose applications to some of those native libraries' functionality. You can use the Android NDK to use any of these native application libraries directly from your native code if you are creating an app that requires C or C++ code.

2.4 Android Application

Mobile apps are delivered as APK files. APK files are signed ZIP files that comprises the byte code of the application along with all its data, tools, third-party libraries and a manifest file describing the detail functionality of the application. Permissions for files in an application are then registered so that only the application itself can able to access them. In addition, every program is given its own Virtual Machine when it is started which ensures code is segregated from other applications

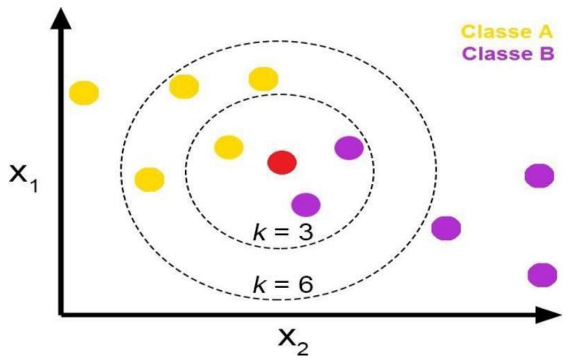

Fig 1: K-nearest neighbour

K-nearest neighbour is a distance based classification technique in which new data points are classified by looking at their k- number of neighbours. The new data point is being associated to the class which has majority among the entire k neighbours. Thus decision boundary of majority class is improved by some margin. This process continues till all the points are being classified. KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

3. Proposed System

3.1 System Analysis:

The literature contains various methods which adapt different strategies to detect malware applications. Basically most of the Android malware detection work can be grouped in two categories, static analysis and dynamic analysis. Both approaches have their advantages and disadvantages as Static analysis is unrealistic and vulnerable to obfuscation where as dynamic analysis is generally faster as compared to static analysis and less resource intensive but requires change in kernel of the operating system. Some other methods combine static and dynamic analysis, known as hybrid analysis, to improve detection accuracy. Along with this, there are some new approaches to malware detection using device-based low-monitorable features. Below, we give a quick review of existing approaches that belong to these categories. In this segment, we also address the work done using the identification of behavioral and signature based malware.

Chin et al. proposed a method called Comdroid which is used for detecting the malicious application using communication-based vulnerabilities in Android. In addition to an open API, the Android operating system also features a rich inter application message passing system for transferring messages between applications. This promotes cooperation between applications and reduces the burden on developers by encouraging reuse of the components. Unfortunately, the passing of message is also the subject of an application attack. So Comdroid utilize this concept and analyze the interaction between Android application and determine Security threats in application modules.

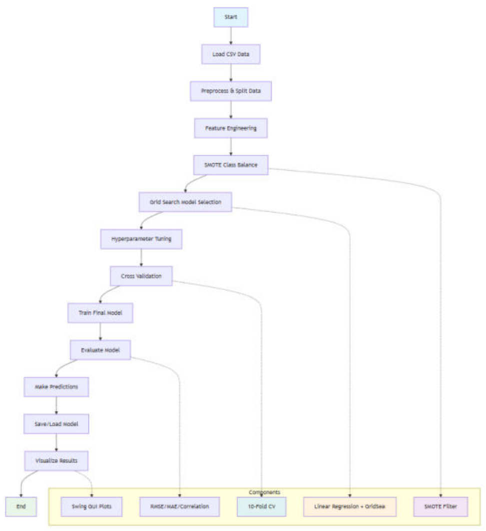

Fig 2: Architecture Diagram

The fig 2 represents a comprehensive machine learning workflow pipeline designed to ensure robust model development and evaluation. The process begins with loading CSV data, followed by data preprocessing and splitting into training and test sets. Feature engineering is then applied to enhance the dataset, after which the SMOTE (Synthetic Minority Over-sampling Technique) method is used to balance class distributions in cases of imbalanced datasets. Next, model selection is carried out through grid search, which systematically tests different combinations of hyperparameters. Hyperparameter tuning and cross-validation, particularly 10-fold cross-validation, are employed to fine-tune the model and ensure it performs well on unseen data. Once the optimal model is selected, it is trained on the full training dataset. The model is then evaluated using metrics such as RMSE (Root Mean Square Error), MAE (Mean Absolute Error), and correlation scores. After evaluation, the model makes predictions, which can be saved or reused later. The results are visualized using plots, and a final analysis is presented using swing-out plots to display key findings. The entire workflow integrates key components such as linear regression (including ordinal regression), SMOTE filtering for class balance, and advanced evaluation techniques, making it a reliable and efficient end-to-end machine learning process.

3.2 **Evaluation Metrix:z**

The recall or true positive rate (TPR) is defined as

$$\text{Recall}=\frac{TP}{TP+F} \quad (1)$$

The accuracy is defined as

$$\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN} \quad (2)$$

The F1-score is defined as

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

3.3 Dataset:

The dataset used in this work is provided by the University of Ben-Gurion, called SherLock dataset. This significant smartphone data is generated from a continuing long- term data collection experiment by providing Samsung Galaxy S5 to 50 volunteers. The two Smartphone agents are involved in this data collection experiment: SherLock and Moriarty.

Sherlock: SherLock is a data collection agent which captures various device metrics (such as battery usage, CPU usage and memory usage etc.) from a wide range of sensors and applications at a high sampling rate.

Moriarty: Moriarty perpetrates varied cyber attacks on the user and records its activities in order to provide labels to SherLock dataset.

The primary objective of the dataset is to help safety professionals and research groups to develop a innovative methods to detect malicious behavior in smartphones implicitly in those devices where sensor data can be accessed without the privileges of the superuser(root).

The Sherlock dataset is organized into form of a probe. A probe is a data table in which multiple sensors sharing the same interval are grouped to collect the data. The dataset is decomposed into 15 different probes. Of the 15 probes, 8 are PULL probes, which captures sensor data in fixed time interval and 7 are PUSH probes which captures sensor data as soon as the new information arrives. PULL probes are those table that are sampled at regular frequency. For example system metrics is collected in every 5 seconds and therefore recorded in T4 PULL probe. The various PULL

probes, which provide different information, are explained below.

- T0 probe: Contains Telephone, System, and Hardware Information.
- T1 probe: Contains Location, Cell tower, Wi-fi, and Bluetooth scan information.
- T2 probe: Contains Accelerometer, Gyroscope, and Magnetic field information.
- T3 probe: Contains information about audio and light data.
- T4 probe: Contains CPU, memory, Network traffic, I/O interrupts etc. information.
- App probe: Contains information like memory, CPU, etc. for each running app.

PUSH probes are those table which are sampled at the occurrence of specific events. For example, as soon as a new SMS message arrived it was recorded by the SMS PUSH probe instantly. There are various PUSH probes which provides different information are explained below.

- App package : Contains information about installed, updated and removed apps.
- Broadcast : Contains information about Broadcast intent.
- Call log : Contains information about calls.
- Moriarty : Contains information about Malware actions and Malware sessions.
- SMS log : Contains information about SMS status.
- Screen status: Contains information about screen on/off.

4. Results:

Table 1: Comparison of the performance of classification algorithms

Model	TPR	FPR	FNR	Precision	F1-Score	Accuracy	AUC
Naive Bayes	62.5%	21.3%	37.5%	73.65%	67.62%	70.79%	78.11%
KNN	83.65%	15.23%	16.34%	83.95%	83.80%	84.22%	91.48%
Decision Tree	87.74%	12.25%	12.25%	87.21%	87.47%	87.74%	87.72%
Random Forest	92.42%	8.01%	7.57%	91.65%	92.04%	92.19%	97.56%
XGBoost	93.02%	6.52%	7.0%	93.14%	93.08%	93.25%	97.87%

In order to evaluate the features obtained after the feature selection process, we apply the five selected classification models in this experiment as stated earlier, and examines their effectiveness using various performance metrics introduced earlier. The experiment intends to improve the results of Memon et al. and performed using a balanced dataset having 70% of data for training and 30% of data for testing the models. The results of the experiment, for all five classifiers, are presented in Table 2. The percentage accuracy achieved by all the five classifiers is presented in Fig. which shows that the XGBoost attains the highest classification accuracy of 93.25%, Random Forest attains the second- highest accuracy of 92.19%, and the Decision Tree achieves the third-highest accuracy of 87.74%. The respective accuracy sore of KNN and Naive Bayes classifier is 84.22% and 70.79%. The methods used by our system and the one presented by Memon et al. are very similar. Their system uses the features selected using the Chisquare method having a p-value > 0.05, whereas our system uses the top 40 features selected using the mutual information gain method. We show that the futures selected using the mutual information gain method gives the better result for the Android malware detection on the SherLock dataset as the highest accuracy achieved by our system using the XGBoost (Extreme Gradient Boosting) classifier is 93.25%, whereas the highest accuracy achieved by Memon et al. [6] using Gradient Boosted Trees is 90.24%. Our methods show the better result for the Random Forest also with an accuracy of 92.19% as compared to the accuracy of 89.67% reported by the same classifier in Memon et al. . However, the Decision Tree used by Memon et al. obtained better results in terms of accuracy value 89.72% and FPR value 9.65% as compared to our approach having an accuracy value 87.74% and FPR value 12.25% using the same algorithm. FPR and FNR denotes the total misclassified records but FPR is not considered as critical as FNR because the latter directly indicates the undetected malware, whereas FPR indicates the benign apps that have been detected as malicious. So a low value of FPR and FNR is desirable to make our detection system more accurate and less time-consuming. Fig. shows the percentage value of FPR and FNR observed by all the classifiers. XGBoost has achieved the lowest FPR and FNR values compared to other algorithms, which is 6.5% and 7% respectively. The second- best performance is achieved by the Random Forest with an FPR of 8% and an FNR of 7.6%.

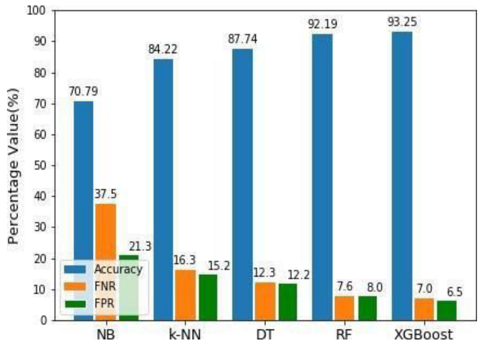

Figure 3: Measure of Accuracy, FNR, and FPR for all five classifiers

The FNR values achieved by the other classification models as shown in Fig. 3 is 37.5%, 16.3%, and 12.3%, respectively for Naive Bayes, KNN, and Decision Tree classifier. Similarly, the FPR values achieved by the other classification models are also shown in Fig.3, which are 21.3%, 15.2%, and 12.2% respectively for Naive Bayes, KNN, and Decision Tree. From the results shown in Table 2, we conclude that the XGBoost classifier outperforms the other classification models in terms of the highest accuracy and the lowest FPR and FNR values, as it reduces the risk of undetected malware and lowers the false notification of malware detection. Comparing to Memon et al. our approach shows a better performance in terms of FPR also as the lowest FPR value achieved in our proposed work is 6.52% using XGBoost classifier, whereas they achieved the lowest FPR of 9.2% using Gradient Boosted Trees. The Random Forest in our approach also shows a better FPR value of 8% compared to the FPR value of 10.16% achieved by the same algorithm in Memon et al.

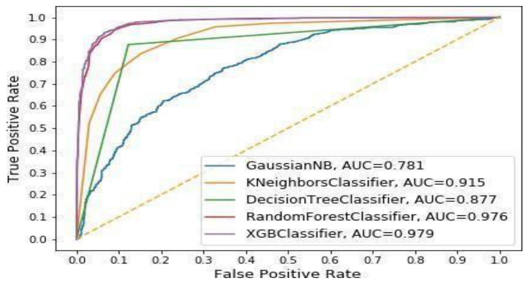

Fig 4 : ROC curves for all five classifiers

The Fig 4 represents the Further to assess and compare the detection performance of every selected classifier, the ROC curve and AUC values are computed and presented in Fig. 4 The ROC curves indicate that XGBoost and Random Forest have similar detection performance with a high value of TPR at a low value of FPR. The AUC values of the Random Forest and XGBoost classifier as shown in Fig. 4 are 97.6% and 97.9% respectively which are almost equal and hence both approaches have a similar detection performance. The detection performance of XGBoost shows a TPR of 0.93 at an FPR of 0.065 and Random Forest shows a TPR of 0.92 at an FPR of 0.08. From Fig. 4, we conclude that the XGBoost and Random Forest have better detection performance in terms of ROC-AUC values compared to other classifiers. The performance of Random Forest is very effective with this dataset due to two reasons . First, the use of out-of-bag error as an estimate.

For generalizing the error improves its performance. Second, being an ensemble classifier it prevents the overfitting of data as it yields the limited value of generalization error even after adding more trees to Random Forest. On the other hand, the performance of XGBoost is very effective, as it avoids the overfitting of data, reduces error rate, and performs faster than other classifiers, due to regularization nature and parallel processing implementation .

5. Conclusion and Future Work

In this work, we have investigated the performance of various classifiers like naive bayes, knn, decision tree, random forest, and xgboost for android malware detection using low- privileged monitorable features. To train our model, we have used system-specific features selected using the mutual information gain method from the t4 probe sampled in the sherlock dataset. The results obtained in this work shows that the futures selected using the mutual information gain method gives the better result for the android malware detection on the sherlock dataset. The findings of this research show that the xgboost and random forest are the top two performers with the respective accuracy of 93.25% and 92.19%. Xgboost classifier is the most accurate in detecting the malware with 93% overall values of precision, recall, and accuracy. In terms of fnr and fpr values, xgboost also outperforms the other classifiers with respective values of 7.0% and 6.52%. Naive bayes, knn, and decision tree are not that effective as their accuracy score is 70.79%, 84.22%, and 87.74% respectively. If we consider the roc curves, the detection performance of xgboost is best with a tpr of 0.93 at an fpr of 0.065 and random forest has the second-best detection rate with a tpr of 0.92 at an fpr of 0.08, which indicates that xgboost has the better classification power. So in this research, we employ various machine learning techniques and conclude that xgboost and random forest classifiers are the top two performers for android malware detection. After all there is always some scope for the improvement in any work. In this work there is also a scope of improving the results by utilizing the large amount of data. Along with this there is also a scope of selecting the set of good features which helps in improving the detection accuracy. Further, data pre-processing can help to achieve a better result by removing the anomalous data.

REFERENCES:

[1] “The android software stack,” https://developer.android.com/guide/platform, accessed: 2020-10-29.

[2] “K-nearest neighbors,” https://towardsdatascience.com/knn-k-nearest-neighbors-1 - a4707b24bd1d, accessed: 2020-10-29.

[3] “Classifying data with decision trees,” https://elf11.github.io/2018/07/01/python-decision-trees-acm.html, accessed: 2020-10-29.

[4] “Random forests,” https://towardsdatascience.com/random-forests-and-decision-t reesfrom-scratch-in-python-3e4fa5ae4249, accessed: 2020-10-29.

[5] “Smartphone market share,” 2020, (Accessed : July 2020). [Online]. Available: https://www.idc.com/promo/smartphone-market-share/o

[6] Memon, N., Anwar, Z., Rahman, M. A., & Khan, S. (2019). SherLock: An Efficient and Effective Approach for Android Malware Detection Using Machine Learning Techniques. *Journal of Information Security and Applications*, 47, 236–245. https://doi.org/10.1016/j.jisa.2019.04.010