(2023), 1–3

ARIEL UNIVERSITY

ARTICLE TYPE

Katana - Android Application ML malware detection

Sean Peer and Shaked Gofin

Author for correspondence: S. Gofin, Email: Shaked.Gofin@Ariel.ac.il.

Abstract

Today every person has at least one smartphone. 71 percent of all smartphones have an Android operating system [5], and the applications on it are designed to make our lives easier as users, But the increased number of applications results in a greater chance of installing Trojans and similar malware. for this purpose, the Kirin security service for Android [3] was researched, which performs lightweight certification of applications to mitigate malware at installation time. Kirin certification uses security rules, which are templates designed to match undesirable permissions properties in the security configuration of the applications. In this paper, we propose a machine-learning approach to identifying malicious permissions with a high accuracy of 94% over a data set of 40049 application

Keywords: Smartphone Security, Malware, Android

INTRODUCTION

Smartphones are a wide focus for attacks due to the great sensitivity of the information contained within. And the reason for this is already far beyond the telecommunication process. People consciously use smartphones as the center of their lives. With the introduction of additional technologies into the smartphone world, the security of the information on our smartphones should also advance.

Malware applications inhabit the web and applications store. It does not only intrude via downloading or installing activity, but also intrude via access to particular Email, SMS, and even web links. Juniper research finds that 80 percent of Smartphone devices will remain vulnerable to cyberattacks through 2013. This happens although there is an increase in customer awareness toward the issue of mobile security products. According to Juniper, there are several factors that cause a low level of adoption of security products. It is expected that, by 2023, 4.3 billion people will own a smartphone, feature phones, and tablets are fortified by mobile security devices, and a very big percentage of it will be vulnerable. [7]

According to past studies by the Department of Homeland Security and the Federal Bureau of Investigation, as the dominant mobile operating system, Android is the primary target for malware attacks because there are many users who are still using the older versions of the software. According to government agencies, 79 percent of the existing malware is threatening android mobile systems while the rest are haunting other mobile systems.

Related works

There has been a significant amount of research in the field of android ML malware detection in recent years. One approach is to use machine learning algorithms to analyze the behavior of apps and identify malicious activity. For example, a decision tree algorithm was used to classify apps as malicious or benign based on their network behavior. Another study by Alqahtani et al. (2019) proposed a deep learning model that utilizes both static and dynamic features of apps to detect malware. Additionally, some researchers have used ensemble methods to combine multiple machine learning models for improved detection performance, as seen in a study by Mahdavi et al. (2017). Some other research works like (Chung et al., 2018) proposed a deep learning model that utilizes both static and dynamic features of apps to detect malware, and (Trabelsi et al., 2018) proposed a deep learning-based android malware detection approach using a convolutional neural network.

Drebin, MaMaDroid, and Andromaly are three well-known android malware detection systems that have been widely studied in the field of mobile security.

Drebin, proposed in 2014 is a system that utilizes static analysis to extract features from android apps and train a machine learning classifier to detect malware. The system has been shown to be effective in detecting known malware and has a low false positive rate.

MaMaDroid, proposed in 2015 uses dynamic analysis to detect android malware by monitoring the behavior of apps on a device. The system uses a combination of machine learning techniques such as decision trees and random forests to classify apps as malicious or benign.

Andromaly, proposed in 2012 is a behavior-based detection system that uses machine learning to analyze the runtime behavior of android apps and identify patterns associated with malware. The system has been shown to be effective in detecting known malware and is capable of detecting zero-day malware.

All these systems have been widely used and evaluated in various research studies and have shown their effectiveness in detecting android malware, Drebin and MaMaDroid are based on static and dynamic analysis respectively, while Andromaly uses runtime behavior analysis to detect malware. Furthermore, the combination of multiple analysis techniques can improve the overall performance of the system, as can be seen in some recent studies like (Liu et al., 2017) that proposed a hy-

brid approach that combines both static and dynamic analysis to detect android malware.

Methodology

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.[2] Each individual tree in the random forest outputs a class prediction and the class with the most votes becomes the model prediction.

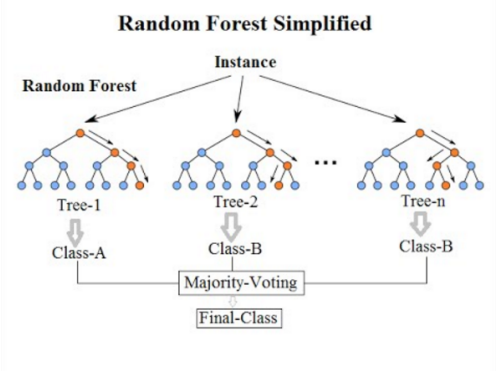

Figure 1. Random forest diagram [6]

The Random Forest model is trained on a dataset using 40049 instances and 4928 features, each instance represents an android application benign or malicious and each feature represents a single security permission.

In order to create the dataset we extracted the permissions of the applications to a CSV file by using Androguard, which is a Python-based tool used for reverse engineering Android applications by taking raw Android Package (.apk) files and breaking them down to analyze.[4]

After we created the dataset we split the data into a training data set and a testing data set using an 80% - 20-% ratio, using the Random Forest model to predict whether an application is malicious or benign. The results of the predictions along with feature importance are shown in the table below.

The Data-set we used for our machine learning classifier was taken from Crystal Ball [1]

Results

The model performed with a high accuracy.

Table 1. An Example of a Table

Classifier	Accuracy	recall	f1-score	support
Random Forest	0.98	0.96	0.97	8010 applications
Decision Tree	0.94	0.85	0.87	8010 applications

By looking at the confusion matrix we can see that the model answered correctly by declaring applications as benign and answering correctly True Negative of 85.91%, and by declaring applications as malicious and answering correctly True Positive of 12.63%.

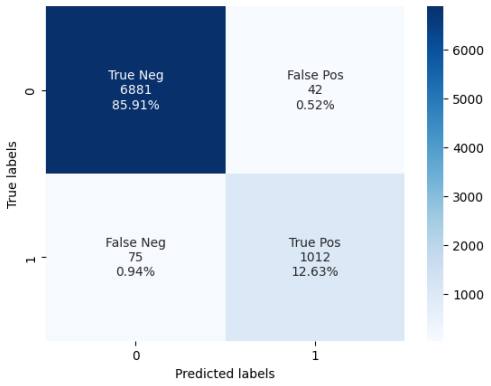

Figure 2. Random forest confusion matrix

Another insight into the random forest classifier results is the importance of feature selection. Random forests are able to handle a large number of features, but selecting relevant features can improve the accuracy and interpretability of the model. One way to identify important features is through feature importance measures, such as permutation importance or mean decrease impurity, which are built into the random forest algorithm. in Figure 2 we can see a visualization of the top 10 features that contributed the most to the model results. This visualization can be valuable in understanding the factors that influenced the model's performance and can help inform future feature selection decisions. Additionally, it can be used to identify patterns or interactions between features, which can provide insights into the underlying relationships in the data and the nature of the problem being solved.

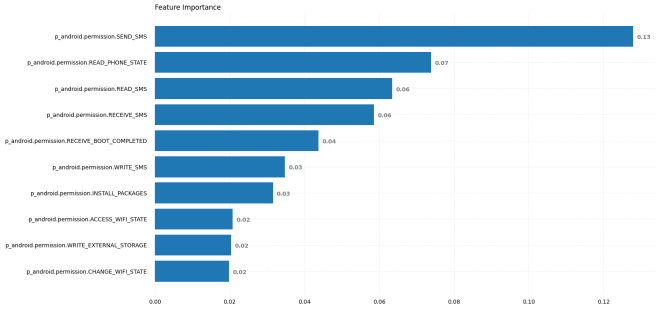

Figure 3. Random forest feature importance

A representation of a decision tree may help us to understand how the features were selected and how they contributed to the overall model results. In Figure 3, we can see a decision tree that was trained on the data used in this example.

The decision tree illustrates how the features were selected and how they contributed to the overall model results. The tree starts with a split at the root node, which indicates that there are two possible outcomes for this data: success or failure. From here, the decision tree splits into two branches, corresponding to the different features being evaluated.

Each branch in the decision tree represents a different feature being evaluated. in this case, the first node is the permission "SEND_SMS" see Figure 4 below.

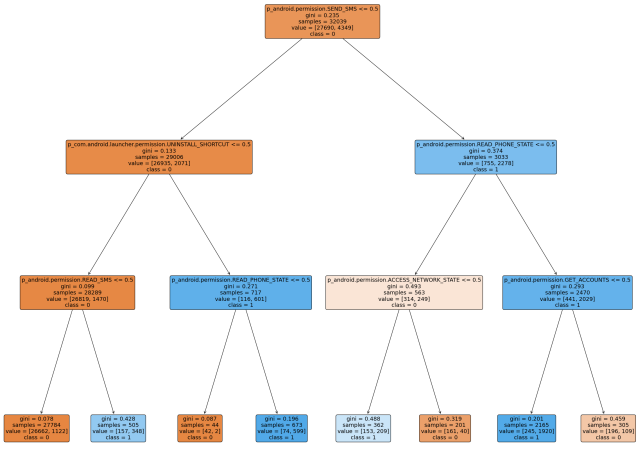

Figure 4. A single decision tree

Conclusion

Our mission on this project was to improve Kirin. In order to reduce the dangers of Kirin's hard-coded permission set, we added a machine-learning model (random forest) to detect the features that pointed to an application being malicious doing so as well as to improve our classifier model to be more resilient to updates on Android software, which was one of the biggest vulnerabilities in the Kirin model. As a result of our improvement, we were able to overcome this difficulty and even classify with high accuracy of 98% whether an application is benign or malicious. in the future, more data can be used to train the models as we found that application permission is an insufficient resource to determine maliciousness.

Data Availability Statement The data can be found on our GitHub page. https://github.com/SeanPeer/When-the-guard-failed-the-droid.

References

[1] Harel Berger et al. "Crystal Ball: From Innovative Attacks to Attack Effectiveness Classifier". In: *IEEE Access* 10 (2022), pp. 1317–1333. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3138628.

[2] Leo Breiman. *Random forests - machine learning*. URL: https://link.springer.com/article/10.1023/A:1010933404324.

[3] William Enck, Machigar Ongtang, and Patrick McDaniel. "On Lightweight Mobile Phone Application Certification". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 235–245. ISBN: 9781605588940. DOI: 10.1145/1653662.1653691. URL: https://doi.org/10.1145/1653662.1653691.

[4] Infosec. *Android penetration tools walkthrough series: Androguard*. Oct. 2020. URL: https://resources.infosecinstitute.com/topic/android-penetration-tools-walkthrough-series-androguard/.

[5] Federica Laricchia. *Global Mobile OS Market Share 2022*. Jan. 2023. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[6] *Random Forest*. Jan. 2023. URL: https://en.wikipedia.org/wiki/Random_forest#/media/File:Random_forest_diagram_complete.png.

[7] Chandra Shekhar Yadav et al. "Malware Analysis in IoT amp; Android Systems with Defensive Mechanism". In: *Electronics* 11.15 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11152354. URL: https://www.mdpi.com/2079-9292/11/15/2354.