Webology (ISSN: 1735-188X)
Volume 17, Number 3, 2020

Predicting Unexpected Permission Authorization Of Android Application Using Machine Learning

Manisha Patil[1] and Dhanya Pramod[2]

[1]Faculty of Computer studies, Symbiosis International (Deemed University) (SIU), Lavale, Pune, 412115, Maharashtra, India. Indira College of Commerce and Science.

[2]Symbiosis Centre for Information Technology (SCIT), Symbiosis International (Deemed University), Hinjewadi, Pune, 411057, Maharashtra, India.

Abstract

Purpose–One of the leading and popular operating system (OS) for mobile phones and tablets is Android OS. Since Android is an open-source platform, there is tremendous growth in its users and this has appealed to the attackers to target the platform. This paper focuses on the work done in the area of mobile app security analysis, by using permission-based static analysis of Android apps and further proposes a model to detect unauthorized permissions. Machine learning algorithms are used to train the models and further predict unexpected permissions in applications. Finally, results are compared with MalDAE frameworks results.

Design/methodology/approach — this study examines more than two hundred smart phone apps, from the Play store, to detect unexpected permission authorization. Apps from various categories like Games, Entertainment, Education, Shopping, Tools, and the most popular free APKs have been studied. Also, reverse engineering was administered to all APK files and permission from each APK was extracted using the self-built tool: AndRev and Safe features were calculated using Virus Total tool. Further, extracted features were stored in a .csv file, and analysis is carried out using machine learning algorithms to predict unexpected permission authorization that may exploit the user's privacy.

Findings –Three algorithms, namely, SMO, J48, and Decision Tree are applied on a Final dataset of 100 applications. Out of the three learning algorithms, the Decision Tree classifier gave the best results with a classification accuracy of 83%. Further, the study extended and six algorithms, namely, Logistic regression model, J48, SMO, Random Forest and Decision Table, Bagging Model with cross-validation, are applied on 250 applications. Out of the three learning algorithms, Bagging Model is giving the best results, with a classification accuracy of 88%.

Originality/value—in this study, free mobile apps were analyzed for unexpected permission authorization. In the past several years, authors have analyzed these issues, but in this work, a tool to extract features from the apk file was designed and stored in a .csv file to ease analysis. Feature extraction tool can be used by data scientists for conducting future research. Also, most popular free mobile apps (more than a hundred) were analyzed and proposed machine learning predictive models for predicting user's privacy using permissions data set of free mobile apps. Observations obtained from this security analysis would be useful to future android app developers.

Keywords: Privacy, Mobile apps, Static Analysis, Permission, Android APK.

1. Introduction

Nowadays, users do many of their routine tasks like online shopping, net banking, travel booking, movie ticket booking, mobile learning, and so on using smartphones for which internet access is required. With the popularity of smartphones and mobile apps, to do browsing, e-mail checking, and similar kind of internet services, users use default apps installed on smartphones or search for various apps from the play store and download apps to the device Abah et al.(2015) [1].

However, in all these situations, access is required to smartphone utilities, specific storage units, and the risk is that personal data is available on the same device. Some smartphone apps also collect very detailed personal information like location data and store videos, photos, multimedia messages, text, and so on. The account credentials for services often used by user has created issues associated with user's privacy. In addition to these users are not aware of this privacy coercion linked with the installation of apps to the device. The risk here is that even though the basic functioning of the apps does not require access to certain utilities, the permission for accessing them might be enabled by default in the app by mistake or due to the oversight of the programmer. Bakour et al.(2019) [11].

In this study, two hundred free android mobile apps available from Google and the other trusted play stores are analyzed to find such unexpected permission authorization. The paper also addresses related issues, such as access to specific permission, the necessity of permission to perform basic functionality of Apps. The proposed method also predicts whether it is safe for a normal user to use those apps.

The paper begins with a review of the related work by the earlier researchers using static and dynamic analysis approaches used to check user's privacy and detection of malicious behavior. In the next section, we present the methodology used to collect, clean the dataset of android apk, reverse engineering to extract, and aggregate features are depicted. The next section covers the machine learning approach for the prediction of unexpected permission authorization of free mobile apps. The paper further presents the results and related discussions and ends with conclusions.

2. Literature Review

The static analysis uses the code segments analysis technique to detect unexpected behavior. Static analysis is carried out on the Android device without running an application. The benefit of static analysis is the low computational cost, low resources, and less time consumption. Signature-based and Anomaly detection are two different ways of Static Analysis.

The signature-based detection technique is also called Misuse Detection. Feng et al. 2014 [15], presented a semantic language Appopscopy which is used for detecting malicious Android applications based on signatures. In this approach, Signature matching was carried out using the inter-component call graphs for control flow properties and static taint analysis for data flow properties. In the experiments conducted by Fuchs et al. 2009 [16], they use Scandroid, where the security-related permissions were extracted and further data flow was analyzed with the malicious signatures. In another study, Zhou et al. 2012 [42], proposed a method for extracting permissions and further detecting Android application malware using heuristic filtering. All three authors have their way towards a research study based on signature matching for control flow, data flow, and permission properties to detect Android application malware.

The anomaly Detection process has been widely accepted in detecting malicious behavior and generally uses machine learning algorithms. Features extracted from the known malicious app were used for training the learning model and predict malware that was not known before. Machine learning approaches have been used by many studies to train and classify applications. The static features of safe and unsafe applications have many commonalities. The combination mode features on which static analysis is based such as API calls, components and permissions often have high false positive rates, while hybrid and dynamic analysis are sensitive towards time and have more requirements for the operating platform. Android Malware detection method Li. J. et al. 2019 [27], is based on static analysis, using pattern matching with Naïve Bays algorithm. The methodology of frequent pattern mining and API correlation helps to keep features independent which is the ideal precondition for the Machine learning technique used. This method results not only because of the timeliness detection but also because of the low false positive rate and high accuracy.

Abah et al.2015 [1], tried machine learning with K-Nearest Neighbor classifier to train the model. Aung et al. 2013 [10], proposed a framework in which a permission learning model was designed based on machine learning algorithms to classify applications. These two author's study signifies that trained learning models are used for predicting malicious behavior of mobile apps. Similarly, Shabtai et al. 2011 [8], proposed Android based framework - Adnromaly for anomaly detection using Machine learning models and continuous monitoring of information obtained from mobile devices. At that time due to the unavailability of real malware authors could not test their proposal.

Dynamic analysis is a process of executing applications in the real environment. The advantage of dynamic analysis is to record the application's runtime behavior and dynamic code loading. Dynamic analysis is harder to implement than static analysis, due to the app execution overhead.

Mobile-Sandbox [33], first performs static analysis then uses the obtained results to perform dynamic analysis through extended code coverage execution. It further extracts API

calls and applies machine learning algorithms for the detection of malicious applications. The intelligent mobile malware detection method, Bakour et al. 2019 [11], uses permission requests with API calls. The proposed method not only detects malware effectively and achieves an F-measure of 94.3 % but also supports malware forensic investigation significantly for mobile applications.

Afonso et al. 2015 [3], proposed a dynamic analysis technique to record the system and for detecting the malware call frequency, an API based feature was used. However in this case the application's certain API level acts as the restriction. Taindroid 2010 [44], is a tool that was proposed for malware analysis and it used network data for application analysis. In his experiment, Maline 2006 [30], uses a tool, which utilizes system calls for classification and explores machine learning algorithms to detect malware. The study mentioned above is based on dynamic analysis of system call, API call, and network data as input features. In addition to this, Markus Miettinenet al. 2006 [30], studied how malware-based malicious activities are carried out in applications. One of the effective protections suggested by authors is to consider the mobile device itself as a test object. Finally, the authors proposed a model - unified intrusion detection which did not have any precise and viable scheme. Schmidt et al. 2008 [2], recommended a framework that monitors Linux-kernel level events. The authors could not test the project properly due to the unavailability of real Android devices at that time.

In Bakour et al.2018 [12], research gaps in the existing Android malware detection framework were discussed and future research trends were proposed with Schematic Review Model. This review fails to conclude which approaches and features are used for malware analysis. In Alazab et al. 2020 [4], developed an abstract model which will be executed in the background on the mobile device without intervening in the user's other activities by protecting the user's and developer's community from a harmful and malicious app on the mobile device. During this study, 380 user's response was considered with uninformed decision making while installing an android app on the device. The study is about the android user-based security model that breaks the OS aspect of keeping applications isolated.

One of the primary methods, Inter-Component Communication Analysis is used for detecting malware to avoid the disadvantages of the signature-based approach. Shabtaiet et al. 2011 [8]. It tracks the information transferred between components and checks for data leakage or privacy of data access. Android applications are programmed using Java programming and further compiled to byte code, and translated into Dalvik Virtual Machine format referred to as DEX. To analyze the application's data flow, it decompiles Android APK files to a byte code or a source code of java. The Inter-Component Communication analysis consists of permission, API calls, control flow, data flow, structural and semantic analysis.

In Li. L et al. 2017 [28], static analysis techniques, challenges of malware detection techniques proposed between 2011 to 2015 are reviewed in detail but it does not mention features used for analysis purposes. By Alazab et al. (2020) [5], about 80 static analysis existing techniques are studied in-depth, challenges are discussed, one case study was conducted to check the power of anti-malware systems against malicious applications. The common thing observed in all three

systematic literature reviews conducted for android malware detection techniques, the trend of research using visualization for Malware analysis is not discussed.

Over a period of time hybrid methods that combine static and dynamic techniques evolved. Jeong et al. 2008 [25], argued that a system call sequence in a binary executable can be used to generate a code graph that can further detect malicious activities. Inference from the system call graphs from execution traces could further derive differences between normal and malicious applications. Naïve Bayes (NB), Decision Trees, Bayesian Networks (BN), etc are the machine learning classifier techniques applied by Shabtai et al. 2011 [8], to classify Android Applications. The total features collected are around 22,000 and further reduced to 50 features during the study for classification. Detection of repackaged applications using DroidAnalyt [7], a signature-based system. The major drawback of these techniques is difficult to implement in practice.

Enck et al 2011 [44] used static analysis techniques for the research study. Analyses 1100 free applications from the official Android Market, de-compiled all apps to discover security-related metrics and their association. The authors concluded that sensitive information is extensively leaked. Pridgen & Wallach 2013 [38], examined a sample of 114,000 apps and found that a privacy risk to users is increasing and consequently the number of permissions required by apps is also increasing. Felt et al. 2011[6], and Kelley et al. 2012 [7], suggested that Android users are not aware of the Android permissions system – although it's not sufficient in providing ample user privacy. However, this is mainly focusing on work based on the Android permission system and the user privacy risk associated with it.

In a dissimilar approach proposed [29], Kern & Sametinger 2012 argued that by allowing the use of individual permissions as a per-app basis so that each Android app would have listed permissions explicitly and the user can choose request as per requirement. Studies have also proposed apps to detect permission-based risk issues and recommended framework [46], which monitors and controls an app's access to sensitive permissions. Berthon et al. 2012 [32], came up with a set of two apps, Security Monitor, which has to be installed on the device, and Security Reporter, injected into an app to monitor the targeted app and then reports to the Security Monitor with resource requests details. Moreover, these studies have app implementations to resolve permission-based user's privacy risk issues.

Juanru, Dawu&Yuhao 2012 [26], done Android malware research using a technique of decompiling Android apps. Xu, Saïdi& Anderson 2012 [46], came up with a solution to enhance user privacy by automatically repackaging Android apps. Aurasium tools also have abilities like sandboxing and policy administration.

Dynamic analysis is performed by monitoring a running mobile application. In this era of smartphone usage growth, users store a huge amount of data including sensitive data on their devices. It is very essential to analyze the usage of such sensitive data and TaintDroid [44], is a tool that uses dynamic taint tracking for the same on Android devices. Any information that was sent from another application needs tainting. TaintDroid notes the data originated and received from such sources and identifies the paths of the flow. A sensitive data leak is reported if the data are used by other applications. However, it did not give any sort of alert messages for considerable

numbers when checking was done using some of the evaluated malware samples.

Kirin [38], an application that provides certification by checking permissions at the time of installation. Kirin extracts security configurations to check it against the rule concerning the security-related policy which the app was having at the time of app installation. Kirin may decide to delete the app or notify the user as it could not meet all the security policy rules.

In another work runtime techniques data shadowing, blocking outside communication were proposed to protect mobile user's privacy of AppFence. In data shadowing blank or dummy data is replied to a personal data request of a non trusted application. The second technique does not allow, sharing app information at runtime by intercepting network stack calls as well as monitors if it is shared with sockets. During such situations, alert messages are released. Both are phone-based approaches so will increase CPU overhead.

There have been efforts to avoid exploitation of transitive permission properties and XManDroid- a real-time communication monitoring framework.[34] is a milestone in this direction. The framework was verified by adding 7 types of attacks with various scenarios in an available dataset. Integrating this model with the existing Android permission framework may be attempted by the authors in the future. Portokalidis et al. 2010 [19], proposed Paranoid – a security model based on remote servers to do a continuous security check. The information needed to execute an app on the device is transferred to a remote server. The app is then re-executed again on the remote server to keep minimum computational and battery usage of the device.

TaintDroid[44] and Vet-Droid [48], both the Framework are using dynamic taint analyses for tracking information flow. However, many Apps transmit sensitive data as basic functionality, especially a few Apps like WhatsApp and other popular social media apps like Instagram or Facebook, and do the same. To illustrate, in WhatsApp, one user clicks a photo, then sends it to one of own friend by accessing the contact list. In such cases, the App may access sensitive information and communicate over the network across the globe, which is expected to notify the user as the part of dynamic analysis technique task and hence considered as malicious behavior. Thus, it shows that dynamic taint analysis techniques are not effective to identify malicious code efficiently.

Apps need to be installed first in case of dynamic analysis process then it can be analyzed but in the other static analysis process, App is analyzed and notification is given to the user beforehand and not after installation.

MalPat [37], an automated technique to detect malware is designed using permission-related APIs and RF classifier. Reverse engineering apk features are extracted and unique patterns of a set of permission-related patterns are then constructed, which is further used for classification. Results are then compared by the author with existing models.

Matina T. et al. 2015 [39], studied the impact of the evolutionary increase in the app permissions requests from users and developers and reports the results, that today's smartphone OS lacks the required level of protection to personal data of concerned. The study also proposed few remedies against the erosion of user privacy like Data transfer, storage, usage- time, and its security in a data

protection framework that has an impact on transparency and user consent, in this case, is required. The author's future intention is to study user's awareness and investigate mentioned suggestions with a focus on user's privacy concerns.

Spyros E. et al. 2017 [35], analyses thousands of free and paid apps from the Google app store with respect to extent of request for access to personal data. Further Co-related quantitative factors like Installations, reviews, ratings or reviews, types of data access required. The authors argued that the level of privacy protection of personal data in smartphone apps leaves a lot of room for improvement. On mobile devices, the privacy protection is so weak that privacy leakage is possible by manipulating the existing set of permission. Access to the gyroscope sensor is obtained through a request of permission on the mobile device and recovered a human speech by G.Andre et al. 2013 [17].

Most of the permission-based studied work is performed using feature defined with <uses-permission>tag, in Talha K. A. et al. 2015[36], the proposed algorithm consist of a signature database, android client, and server. The logistic regression algorithm is used for the classification of an app as malware or not with an 88% of detection accuracy rate. A static analysis-based framework was proposed Spreitzenbarth et al. 2015 [33], where each used permissions sensitive APIs, permission rate and monitoring systems have been used as a feature for testing and training the classifier. PCA was adopted for preprocessing the extracted features and rotation forest RF which is an ensemble that was used for the classification of the apps into safe or unsafe. The outcomes are then compared with some other existing models. Features considered for static analysis of malware detection method Wang W et al. 2018 [40], are distrustful API calls, properties of hardware used, permissions, intents, and patterns related to code.

Excess set of permission or over-privileged permission detection in Android app is studied by Stowaway [18]. Research shows that almost one-third of the total available android apps this issue is noted. In Paranoid[19], it is found that after the approval related permissions such as "WRITE_MEDIA_STORAGE" and "READ_EXTERNAL_STORAGE" allow to access all important photos, notes, and videos stored in the SD card of the mobile device. Thus, it has become a consent that permissions need to be controlled. In the malware characterizations study, David Barrera et al. 2011 [17], studied more than 1000 popular Android Apps and concluded that the set of permissions requested by various categories of apps are different, and commonly requesting permissions are Internet and READ_PHONE_ STATE.

The research of H.T.T. Truong et al. 2013 [21], shows that permission like "sendTextMessage" is used by android malware to send a message without notifying the user. DroidSwan [22], shows "WRITE_EXTERNAL_STORAGE", "INTERNET" , "WRITE_APN_SETTINGS", and few more permissions are used for malicious purpose. J. Oberheide et al. 2008 [23], proposed an Android malware detection technique that uses a permission-based approach, in which features are extracted from. apk files and a dataset is created. It uses machine learning approaches to classify the samples. Results are not ideal due to the small dataset, arbitrary arguments, and limited algorithms.

L. Juanru et al. 2012 [26], proposed a lightweight method based on static analysis to detect

Android malware. In this method, features are extracted from the manifest file, hardware components, permissions, components, intents, and disassembled code. Learning-based detection is applied which only uses Support Vector Machines. M. Kern et al. 2012 [29], used permissions, control flow graphs as the features and Zhou et al. 2012 [42], used permissions and Intents for message passing. K-means is used to classify clusters and Singular Value Decomposition mainly is used on the low-rank approximations. Other than these drawbacks, one more observation is noted that in practice rarely the learning type like unsupervised is applied by researchers for completing static analysis study.

Research till now states, the limitations in Signature-based detection techniques and expects to develop new techniques which can handle the polymorphic nature found in malware. Further researchers have proved the efficiency of machine learning and data mining algorithms in the process of detecting malware. Most of the researchers have used supervised models for learning and very few of them used k-means algorithm, unsupervised clustering method.

The method of malware detection is a binary (safe or unsafe) classification problem, hence researchers are using the SVM (Support Vector Machine) model. This model is a non-probabilistic one that is based on finding a relevant hyperplane that separates data sets with expected margins. Another important classification model used is Naive Bayes, for classifying binary as well as multi-class problems. This model is based on the Bayes theorem and predicts the probability of feature without using correlation with other features of the dataset Chumachenko K et al. 2017 [13]. Random variables carry information about another variable and mutual information (MI) is used for measuring the amount of information. This definition is useful for quantifying the relevance as it is within the context of feature selection that gives a feature subset concerning the output vector. MI used in Zhu H-J et al. 2018[49], along with two other feature selection methods. Initially, the author came across Chi-square, based on the feature ranking of chi-square scores and then selecting the top-ranked features for training the model. Secondly, it underwent one-way analysis of variance – ANOVA, which was based on ranking the features utilizing one-way ANOVA F-test statistics and selecting top-ranked features for the training model. MalDAE [20], malware detection framework, unlike earlier methods gives understandable explanation for familiar types of malware and also provides predictive support for resisting malware.

In recent research, very few authors have come with a deep learning approach that uses (NN) neural network with input, output, multiple hidden layers distributed by a large number of neurons. Best approach due to its ability to extract features automatically. In Karbab EB et al. 2017 [24], the semantic vector is created to train neural networks from raw API calls of the android apk dataset. In Nix Ret al. 2017 [31], (Convolution – NN) CNN, (Recurrent –NN) RNN, (Long short term memory) LSTM are implemented to reduce training time for similar pattern matching process used in signature-based detection techniques.

The various attempts made by researchers to analyses applications are summarized in table I.

Author	Technique Used	Analyses
Feng et al. [15] {Appopscopy- a semantic language }	Signature based static analysis	Inter component call graphs for control flow properties
		Static taint analysis for data flow properties
Fuchs et al. [16] {Scandroid }		Security related permissions are extracted, data flow is analyzed
Zhou et al. [42] {heuristic filtering }		Extracts permissions
DroidAnalyt[7]		repackaged applications
Abah et al. [1]	Anomaly Detection based static analysis	Machine learning with K-Nearest Neighbor classifier
Aung et al. [10]		Permission learning model is designed based on machine learning algorithms to classify applications
Shabtai [8] {Adnromaly}		Machine learning models and continuous monitoring of information obtained from the mobile device
Afonso et al. [3]		Records system and API call frequency
Taindroid [44]		Uses network data
Li.. J. et al. [27]		frequent pattern mining, API correlation, Naïve Bays algorithm
Maline [30]		System calls, Machine Learning Model
Markus Miettinen [30]		consider mobile device itself as a test object, unified intrusion detection model
Schmidt [2] et al		monitors Linux-kernel level events
Jeong et al.[25]	Inter-Component Communication Analysis	System call sequence and generates code graph.
Berthome et al. [32]		system call graphs, Security Monitor and Security Reporter

Table 1 : Related work done on various techniques used for android mobile app security

Author	Technique Used	Analyses
Shabtai et al.[8]	Static analysis	Machine learning classifier
Enck et al [43]		Security-related metrics and its association
Pridgen& Wallach [38]		Privacy risk to users
Felt et al. [6] and		
Kelley et al. [7]		Users are not aware of Privacy Risk
Kern &Sametinger [29]		Listed permissions explicitly
Mobile-Sandbox [33]		first performs static analysis then uses the obtained results to perform dynamic analysis through extended code coverage execution
Bakour et al. [11]		Intelligent mobile malware detection , supports malware forensic investigation significantly for mobile applications.
Zhou et al. [33]		Monitors and controls an app's access to sensitive permissions
Juanru, Dawu &Yuhao [26]		Decompiling App
Xu, Saïdi& Anderson [46] {Aurasium}		Sandboxing and policy administration then Repackaging
TaintDroid [44]	Dynamic analysis	Usage of sensitive data, "Taint Sources", "Taint Flow"
Kirin [38]		Extracts security configurations to check it against the security policy and notify accordingly
Bakour et al. [12]		Schematic Review Model
Alazab et al. [4]		abstract model
Li. L et al. [28]		reviewed static analysis techniques
Alazab et al. [5]		one case study was conducted to check the power of anti-malware systems against malicious applications

Afonso [3]		Data shadowing, blocking outside communication to protect mobile user's privacy
Bugiel et al. [34] XManDroid		A real-time communication monitoring framework to avoid exploitation of transitive permission properties
Portokalidis et al. [19] {Paranoid}		A security model based on remote servers to do a continuous security check
TaintDroid [44] and Vet-Droid [34]		Dynamic taint analyses to track information flow

Table 1 continued : Related work done on various techniques used for android mobile app security

Author	Technique Used	Analyses
Matina T. et al. [39]	Privacy Protection Study	Impact of evolutionary increased in app's permissions requests from both user's and developer's point of view
Spyros E. et al. [35]		Co-related quantitative factors like
		Installations, reviews, ratings or reviews, types of data access required
G.Andre et.al. [17]		Access to the gyroscope sensor through a request of permission and recovered a human speech on a mobile device
Stowaway [18]		over-privileged permissions detection
MalPat [37]		an automated technique to detect malware is designed using permission related APIs and RF classifier
Paranoid [19]		Access to SD card of mobile through approval of app permission.
David Barrera's [22]		apps request a different set of permissions
H.T.T.Truong et.al. [21]		malware uses permissions without notifying to user
DroidSwan [22]		suspicious permissions
Talha K. A. et al. [36]		logistic regression algorithm

Spreitzenbarth et al. [33]	Malware Characterizations Study	PCA was adopted for preprocessing the extracted features, rotation forest RF was used for the classification
Wang W et al. [40]		distrustful API calls, properties of hardware used, permissions, intents, and patterns related to code.
Chumachenko K et al. [13]		model is based on Bayes theorem and predicts the probability of feature without using correlation
Zhu H-J et al. [49]		Chi-square, ANOVA
Han et al. [20]		correlation , fusion of the static and dynamic API
J. Oberheideet..al. [23]		permission-based approach
Karbab EB et al. [24]		semantic vector is created to train the neural network
Nix R et al.[31]		Convolution – NN, Recurrent –NN, Long short term memory used to reduce training time
L.Juanru et.al [26]		Features like hardware components, permissions, intents and disassembled code, Learning-based detection which uses Support Vector Machines
M.Kern et.al. [29]		permissions, control flow graphs
W.Zhou et,al. [42]		uses permissions and Intents for passing messages. K-means to classify clusters. Unsupervised learning technique.

Table 1 continued : Related work done on various techniques used for android mobile app security

3. Methodology

Google have created Android application format known as APK – Android Application Package, used for distribution and installation of app on user's mobile device. Following is the proposed system Block diagram.

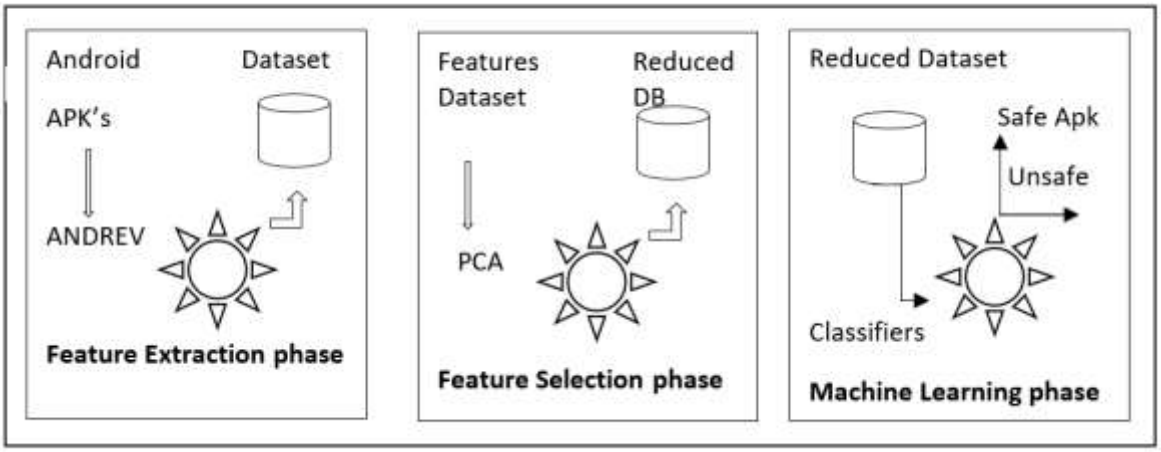

Figure 1: The proposed System Block Diagram

Google play store is the main data source to get free Android APKs required for this research study. There is no other way for the normal user to validate the user's privacy and app security, so the user is simply dependent on the review system provided by Google Play Store which then becomes a trusted data source for the user to download apps. There are different categories of apps available that affect the distribution of permissions but for these study categories of apps like – Games, Education, Entertainment, Shopping, General/Tools are considered.

Considering a situation, where an App requesting permission (INTERNET) access for online shopping is treated normal, but in the case of a General Category app like Calculator, it will be an unexpected permission authorization. The authors decided to download 50 top apps from each category to download from the mentioned data source. As a result, 250 free Apps, which is a good sampling of Apps. The next step is to reverse engineer the APKs to extract permissions.

Drebin dataset [50] provides text files with the list of features and permissions but, apk files are not provided. The dataset does not provide information on whether the app is safe or not safe. Therefore, there is a need to build research dataset with apks.

3.1 Feature extraction phase - APK Decompiling

The APK file is a compressed folder consisting of – DEX, Manifest file, images, layouts called raw resources all are bundled together. For reverse engineering of Android APK files, tools are available like ApkTool, Dex2jar, JD-GUI. This research study utilized ApkTool which is powerful and easy to scale. ApkTool decompiles only one APK file at one run so designed AndRev tool: Batch script in Windows platform to invoking ApkTool to decompile 50 or group of APK files. It creates a folder for each APK file which was the storage issue for the device used for reverse engineering so once the permissions are extracted all folders are deleted. The time for reverse engineering one apk file is not more than one minute depending on the size of a file. The reverse engineering is done with a windows desktop machine. We noticed around 20 – 30 APKs that could

not be reverse engineered due to incorrect file names (e.g. Space in words, extension mistakes, etc.)

Finally, decompiled 250 valid APK files and extracted feature vector – Dataset from those reverse engineered apps.

3.2 Feature selection phase - Feature Aggregation and Data Cleaning

Manifest file extracts permissions and stored those to .csv file – features dataset, after extracting features; noticed more than 91 permissions were getting repeated out of 491. So, removing duplicate permissions from the dataset got unique permissions in the dataset. Using Virus-Total Tool calculated the output variable Safe for all 100 apps. Almost 90 % of apps are safe initially.

In the second phase, applied Principle Component Analysis to analyze the complexity of features and reduced data set / DB to 100 features. The permission extraction result for a typical App is as below.

1 1 1 1 1 0 0 0 1 1 0 0 1 0 . . . 0 0 1

To avoid such a situation, one has to reduce the dimension of feature space which refers to dimensionality reduction. To achieve dimensionality reduction feature elimination and feature extraction are the two ways available. In this study feature extraction method is used as it drops the least important permissions i.e. permissions that are independent of the output variable. The purpose of PCA is to reduce the number of variables, it helps to identify variables to completely remove from the dataset, it ensures variables are independent of one another, and makes independent variables less interpretable.

3.3 Machine Learning phase

Machine learning algorithms are then applied to these feature vectors – reduced dataset to detect unexpected permissions in applications, which results in safe or unsafe app as a predicted output of classifiers used. Three algorithms, namely, J48, Decision Tree (DT) and SM0 are applied on an initial dataset of 100 applications.

4. RESULTS

After the feature vectors are created, the accuracy of unexpected permission authorization prediction has been calculated by applying three supervised machine learning algorithms namely J48, Naïve Bayes and SMO. Out of the 100 applications 60 applications were used as the training data and 40 as test data. To evaluate the efficiency of the method used, a confusion matrix is constructed to measure the accuracy of the experiments.

The matrix uses the following metrics:

TP = Number of APKs correctly classified as unexpected permission authorization.

TN = Number of APKs correctly classified as expected permission authorization.

FP = Number of APKs incorrectly classified as unexpected permission authorization.

FN = Number of malware applications that have not been detected by the classifier.

Accuracy = TP + TN / (TP + FP + TN + FN)

Precision = TP / (TP + FP) and Recall = TP / (TP +FN)

F-Measure = 2 * [(Precision * Recall) / (Precision + Recall)]

The best results were obtained by using the Decision Tree classifier with a correct classification accuracy of 83%.

Table 2-4 give the confusion matrices obtained. Figure 2 gives a comparison of the three classifiers.

Table 2. Confusion Matrix for SMO

SMO	Safe	Not Safe
Safe	55	11
Not Safe	12	22

Table 3. Confusion Matrix for J48

J48	Safe	Not Safe
Safe	53	13
Not Safe	10	24

Table 4. Confusion Matrix for Decision Tree

Decision Tree	Safe	Not Safe
Safe	54	12
Not Safe	11	23

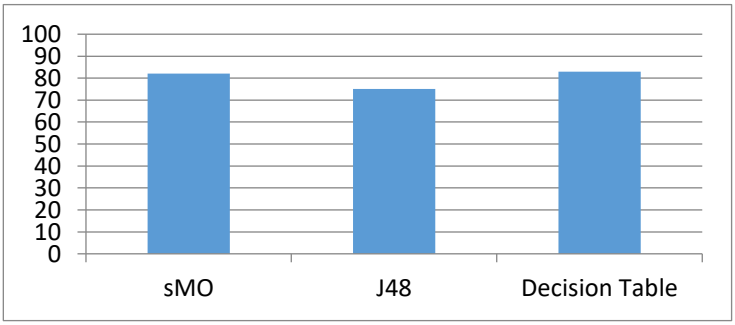

Figure 2 gives a comparison of the three classifiers for Data set of 100 APKs.

Further the experiment is extended with 250 APKs feature vectors are created, the accuracy of unexpected permission authorization detection has been calculated by applying 2 more supervised machine learning algorithm namely Logistic regression model, J48, SMO, Random Forest and Decision Table , Bagging Model with cross validation. Out of the 250 applications 200 applications were used as the training data and 50 as test data. To evaluate the efficiency of the method used, a confusion matrix is constructed to measure the accuracy of the experiments, as depicted below-

Table 5. Confusion Matrix for Logistic Regression, SMO, J48, Random Forest Model for data set of 250 APKs

Confusion Matrix for Logistic Regression Model				
	Cross Validation 10 folds		**Percentage Split (66)**	
LogRegre	Safe	Unsafe	Safe	Unsafe
Safe	5	44	0	18
Unsafe	20	181	0	67
Correct / Incorrect	74.40%	25.60%	78%	21%
Confusion Matrix for SMO Model				
SMO	Safe	Unsafe	Safe	Unsafe
Safe	4	45	0	18
Unsafe	15	186	0	67
Correct / Incorrect	67%	24%	78.82%	21.17%
Confusion Matrix for J48 Model				
J48	Safe	Unsafe	Safe	Unsafe
Safe	0	49	0	18
Unsafe	1	200	0	67
Correct / Incorrect	80%	20%	78.82%	21.17%
Confusion Matrix for Random Forest Model				
RandomForest	Safe	Unsafe	Safe	Unsafe
Safe	1	48	0	18
Unsafe	3	198	0	67
Correct / Incorrect	79.60%	20.40%	78.82%	21.17%

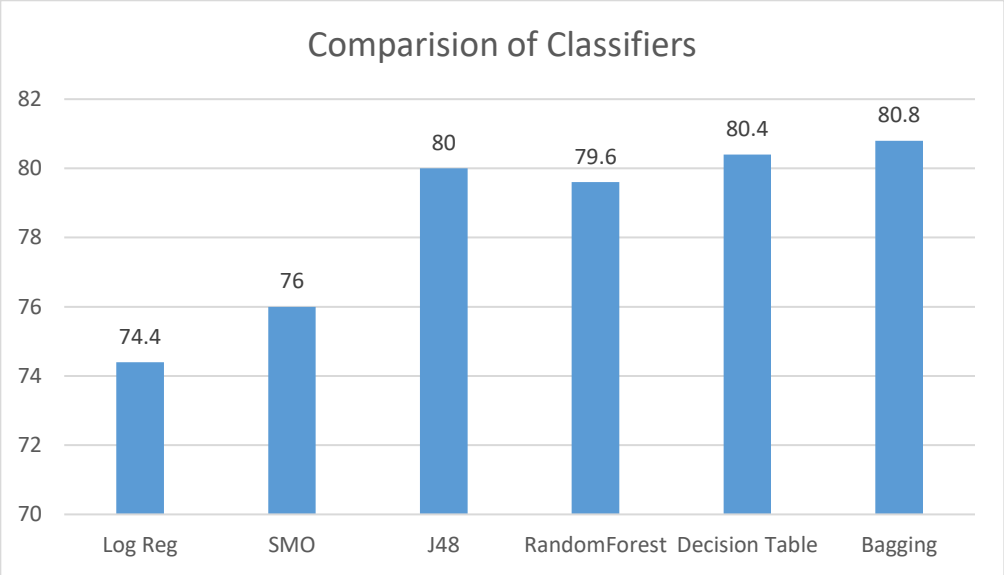

Figure 3 gives a comparison of the six classifiers for data set of 250 apks

Table 6. Confusion Matrix for Decision Table, Bagging Model with Cross-Validation

Confusion Matrix for Decision Table Model						
			PercSplit 80%		PerSplit 90%	
Decision Table	Safe	Unsafe	Safe	Unsafe	Safe	Unsafe
Safe	0	49	0	8	0	3
Unsafe	0	201	0	42	0	22
Correct / Incorrect	80.40%	19.60%	84%	16%	88%	12%
Confusion Matrix for Bagging Model						
			PercSplit 80%		PerSplit 90%	
Bagging	Safe	Unsafe	Safe	Unsafe	Safe	Unsafe
Safe	1	48	0	8	0	3
Unsafe	0	201	0	42	0	22
Correct / Incorrect	80.80%	19.20%	84%	16%	88%	12%

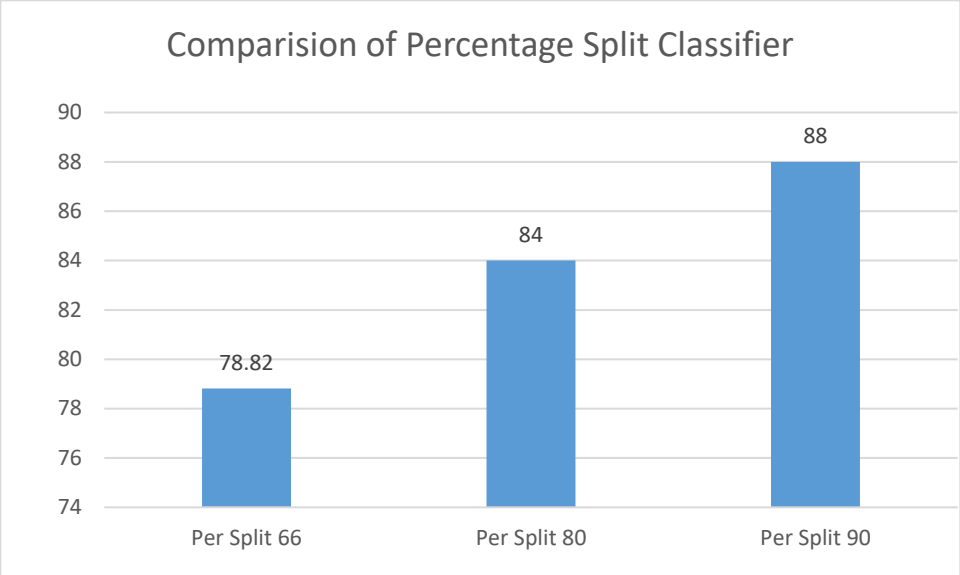

Figure 4 gives a comparison of the percentage split classifier for data set of 250 APKs.

Table 7. Performance for Bagging and Random Forest Model

Performance of Ensemble Learning Models					
	TP Rate	FP Rate	Precision	Recall	F-Measure
Bagging	0.76	0.323	0.755	0.76	0.756
Random Forest	0.83	0.244	0.827	0.83	0.826

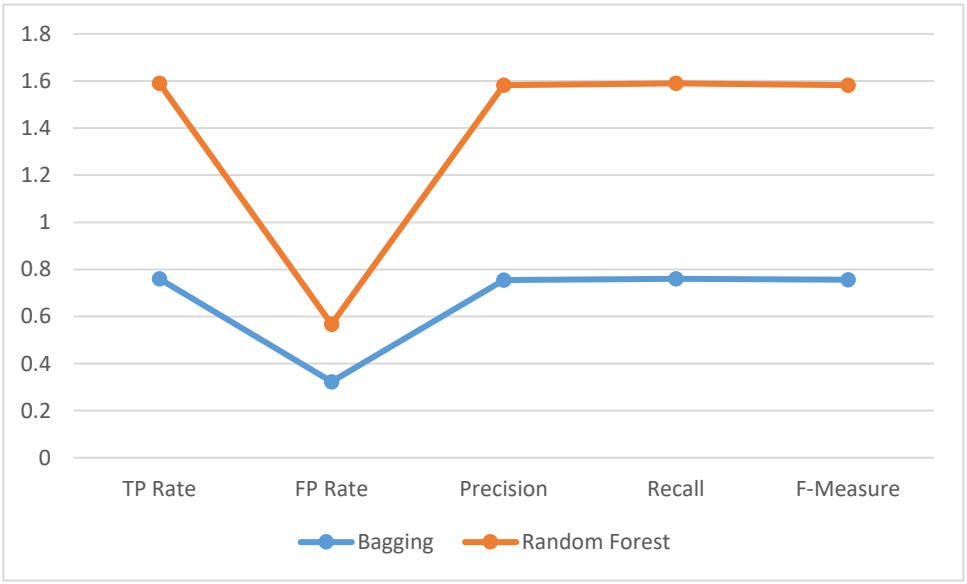

Figure 5 gives a comparison of the Ensemble Learning Model

	Classification based on static Features and ML Model		Classification based on static API sequence- MalDAE	
	Random forest	Bagging	Random forest	Decision Tree
Recall	83.00	76.00	81.08	78.01
F1 Score	82.60	75.60	82.09	79.14
Accuracy	83.00	76.00	84.21	81.4
Precision	82.70	75.50	84.03	80.98

Table 8 – Comparative Analysis between proposed and MalDAE framework

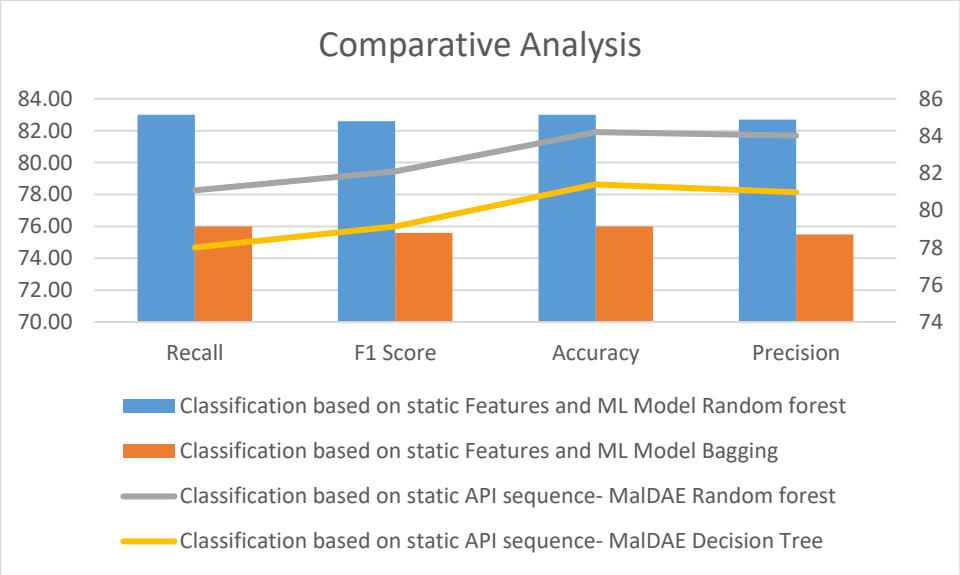

Figure 6 Comparative Analysis between proposed and MalDAE framework

5. Conclusion

Smartphone usage brought general user's lives to a different level. Mobile apps provide unexpected permission authorization which is not safe as per the security point of view to the user and hard to discover the vulnerabilities' root cause. The android operating system is popular among the users, Android app developers as well as malware developers because of its open-source feature. The research shows valuable approaches that are proposed by many researchers but they have limitations to protect user's privacy completely. There is an evolutionary increment in malware for Android OS.

Therefore, it is high time to come with useful security solutions for Android smartphones, like signature-based anti-virus technology, device-level firewall, Identity access management, and control mechanisms, and Intrusion Detection Technology that is lightweight. This paper shows

how android permission-based techniques are used to detect unexpected permission authorization.

The study contributes to the outcomes mentioned as below:

A huge data set of the app was collected, reverse engineered them using a self-designed feature extraction tool: AndRev to extract features and saved the features in comma-delimited file. This feature extraction tool can be used by data scientists for conducting future research Feature dimension is reduced using Principal Component Analysis, which also helped to analyze the complexity and understand features better.

Ensemble learning algorithms like Bagging and Random Forest models are used for the analysis of unexpected permission authorization. Precision recall and F-measure are calculated to measure performance and compared for both models.

Research has focused on applying machine learning algorithms to a dataset of the top 250 apks and explored new ways to use machine learning to detect unexpected permission authorization. By introducing a machine learning layer within a framework, creates the potential to deal with the continuous update in a dataset, as new apps are deployed on the google app store every day. This will allow users the opportunity to use safe apps and monitor unsafe apps.

Comparative analysis using Drebin dataset resulted in a similar performance measures as mentioned in Table 7 and Figure 5 while MalDAE framework resulted in better performance measures as mentioned in Table 8 and Figure 6.This research performs a base for further research in the field of malware analysis with machine learning methods.

Our future work includes:

User's secured apps can be predicted with more depth logic of permissions usage. Practical implementation of work is the future attempt.

Data availability statement

Dataset 1 : The data that support the findings of this study are openly available at https://www.sec.cs.tu- bs.de/~danarp/drebin/download.html

Daniel Arp, Michael Spreitzenbarth, Malte Huebner, Hugo Gascon, and Konrad Rieck "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket", 21th Annual Network and Distributed System Security Symposium (NDSS), February 2014

Michael Spreitzenbarth, Florian Echtler, Thomas Schreck, Felix C. Freling, Johannes Hoffmann, "MobileSandbox: Looking Deeper into Android Applications", 28th International ACM Symposium on Applied Computing (SAC), March 2013

Dataset 2 : The data that support the findings of this study are available from the corresponding author, upon reasonable request.

References

[1]Abah, J., V, W. O., B, A. M., M, A. U., and S, A. O. (2015). A machine learning approach to anomaly-based detection on Android platforms. https://arxiv.org/abs/1512.04122

[2] A.-D. Schmidt and S. Albayrak, (2008) "Malicious Software for Smartphones," TechnischeUniversit¨at Berlin - DAI-Labor, Tech. Rep. TUBDAI 02/08-01, February 2008, http://www.dai-labor.de.

[3] Afonso, V. M., de Amorim, M. F., Gregio, A. R. A., Jun- ´ quera, G. B., and de Geus, P. L. (2015). Identifying Android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques, 11(1):9–17.

[4]Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., & Awajan, A. (2020). Intelligent mobile malware detection using permission requests and API calls. Future Generation Computer Systems: the international journal of grid computing: theory, methods and applications, 107, 509-521. https://doi.org/10.1016/j.future.2020.02.002

[5] A. Ngobeni and S. Mhlongo, "Towards Enhancing Security in Android Operating Systems – Android Permissions & User Unawareness," 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2019, pp. 1-6.

[6] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner,(2011) "Survey of Mobile Malware in the Wild,". [Online]. Available:http://www.eecs.berkeley.edu/∼afelt/malware.html

[7] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin & D. Wagner, (2012) "Android permissions: User attention, comprehension, and behavior", SCOUPS, p. 3.

[8] AsafShabtai, Uri Kanonov, Yuval Elovici, ChananGlezer, and Yael Weiss. (2011) "Andromaly: a behavioral malware detection framework for android devices". Journal of Intelligent Information Systems, pages 1–30, 10.1007/s10844-010-0148-x

[9] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Jan Clausen, Kamer Ali Y¨uksel, Osman Kiraz, Ahmet Camtepe, and SahinAlbayrak.(2008) "Enhancing security of linux-based android devices". In in Proceedings of 15th International Linux Kongress. Lehmann.

[10] Aung, Z. and Zaw, W. (2013). Permission-based Android malware detection. International Journal of Scientific & Technology Research, 2(3).

[11] Bakour, K., Ünver, H.M. & Ghanem, R. The Android malware detection systems between hope and reality. SN Appl. Sci. **1,** 1120 (2019). https://doi.org/10.1007/s42452-019-1124-x

[12] Bakour K, Ünver HM, Ghanem R (2018) The android malware static analysis: techniques, limitations, and open challenges. In: 2018 3rd international conference on computer science and engineering (UBMK). IEEE

[13] Chumachenko K (2017) Machine learning methods for malware detection and classification. http://urn.fi/URN:NBN:fi:amk-20170 31031 55. Accessed 13 Mar 2019

[14] C. A. Castillo, Android Malware Past, Present, and Future, Tech. rep.,Mobile Working Security Group

[15] Feng, Y., Anand, S., Dillig, I., and Aiken, A. (2014). Apposcopy: Semantics-based detection of Android malware through static analysis. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 576–587.

[16] Fuchs, A. P., Chaudhuri, A., and Foster, J. S. (2009). SCanDroid: Automated security certification of Android applications. https://www.cs.umd.edu/~avik/ papers/scandroidascaa.pdf.

[17] G. Andre, P. Ramos, BOXER SMS Trojan, Tech. rep., ESET Latin American Lab (2013).

[18] G. Portokalidis et al., (2010)"Paranoid Android: Versatile Protection for Smartphones," Proc. Ann. Computer Security Applications Conf. (ACSAC 10) ACM, pp. 347-356.

[19] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, (2010) "Paranoid Android: versatile protection for smartphones," in 'Proceedings of the 26th Annual Computer Security Applications Conference', Austin, Texas, pp. 347-356.

[20] Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., & Mao, L. (2019). MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. Computers and Security, 83, 208–233. https://doi.org/10.1016/j.cose.2019.02.007

[21] H. T. T. Truong, E. Lagerspetz, P. Nurmi, A. J. Oliner, S. Tarkoma, N. Asokan, S. Bhattacharya, (2013) The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators, arXiv preprintarXiv:1312.3245.

[22] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, (2011) "Crowdroid: Behavior-Based Malware Detection System for Android," Proc. ACM Workshop Security and Privacy in Mobile Devices (SPMD 11), ACM, pp. 15-26.

[23] J. Oberheide, E. Cooke, and F. Jahanian, (2008)"CloudAV: N-Version Antivirus in the Network Cloud," Proc. 17th Conf. Security Symp., Usenix, pp. 91-106

[24] Karbab EB et al (2017) Android malware detection using deep learning on API method sequences. arXiv preprint arXiv:1712.08996 . https ://arxiv .org/abs/1712.08996 v1

[25] K. Jeong and H. Lee, (2008) "Code graph for malware detection. In Information Networking." ICOIN. International Conference on,.

[26]L. Juanru, G. Dawu& L. Yuhao, (2012) "Android Malware Forensics: Reconstruction of Malicious Events", ICDCSW 2012, pp. 552-558.

[27] Li J., Wu B., Wen W. (2019) Android Malware Detection Method Based on Frequent Pattern and Weighted Naive Bayes. In: Yun X. et al. (eds) Cyber Security. CNCERT 2018. Communications in Computer and Information Science, vol 970. Springer, Singapore

[28] Li L et al (2017) Static analysis of android apps: a systematic literature review. Inf Softw Technol 88:67–95. https ://doi.org/10.1016/j.infso f.2017.04.001

[29] M. Kern, & J. Sametinger,(2012) "Permission Tracking in Android", UBICOMM 2012, pp. 148-155.

[30] M.Miettinen and P. Halonen.(2006) "Host-based intrusion detection for advanced mobile devices".

[31] Nix R, Zhang J (2017) Classification of android apps and malware using deep neural networks. In: 2017 international joint conference on neural networks (IJCNN). IEEE

[32] P. Berthome, T. Fecherolle, N. Guilloteau& JF. Lalande, (2012) "Repackaging Android Applications for Auditing Access to Private Data", ARES 2012, pp. 388-396.

[33] Spreitzenbarth, M., Schreck, T., Echtler, F., et al.: Mobile-Sandbox: combining static and

dynamic analysis with machine-learning techniques. Int. J. Inf. Secur. 14(2), 141–153 (2015)

[34] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, and Ahmad-Reza Sadeghi, (2011) "XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks," Technical Report, TechnischeUniversit Darmstadt.

[35] S. Polykalas, G. Prezerakos, F. Chrysidou and E. Pylarinou,(2017) "Mobile apps and data privacy: when the service is free, the product is your data", In Proceedings of International Conference on Information Intelligence Systems Applications (IISA 2017), IEEE, Larnaca, Cyprus.

[36] Talha KA, Alper DI, Aydin C (2015) APK auditor: permissionbasedandroid malware detection system. Digit Investig 13:1–https ://doi.org/10.1016/j.diin.2015.01.001 er varable.

[37] Tao G et al (2018) MalPat: mining patterns of malicious andbenign android apps via permission-related APIs. IEEE TransReliab 67(1):355–369. https ://doi.org/10.1109/tr.2017.27781 47

[38] T. Book, A. Pridgen& DS. Wallach, (2013) "Longitudinal Analysis of Android Ad Library Permissions", arXiv preprint arXiv:1303.0857

[39] Tsavli, M., Efraimidis, P., Katos, V., Mitrou, L.: (2015) Reengineering the user: privacy concerns about personal data on smartphones. Inf. Comput. Secur. **23**(4), 394–405

[40] Wang W, Zhao M, Wang J (2018) Effective android malware detection with a hybrid model based on deep auto encoder and convolutional neural network. J Ambient Intell Humaniz Comput. https ://doi.org/10.1007/s1265 2-018-0803-6

[41] Wang X et al (2017) Characterizing android apps' behavior for effective detection of malapps at large scale. Future Gener Comput Syst 75:30–45. https ://doi.org/10.1016/j.future.2017.04.041

[42] W. Zhou, Y. Zhou, X. Jiang, P. Ning, (2012) Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces, in: Proceedings of the second ACM conference on Data and Application Security and Privacy, CODASPY '12, ACM, New York, NY, USA, 2012, pp.317–326. doi:10.1145/2133601.2133640. URL http://doi.acm.org/10.1145/2133601.2133640

[43] W.Enck, M. Ongtang, and P. McDaniel, (2009) "On Lightweight Mobile Phone Application Certification," Proc. 16th ACM Conf. Computer and Communications Security (CCS 09), ACM, , pp. 235-245.

[44] W. Enck, P. Gilbert, B.-G. Chun, L. Cox, J. Jung, P. McDaniel and A. Sheth,(2010) "Taintdroid: an information-flow tracking system for real-time privacy monitoring on smartphones," In 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 1-6,.

[45] W. Enck, D. Octeau , P. McDaniel and S. Chaudhuri, (2011) "A study of Android application security," 20th Usenix Security Symposium.

[46] Xu R., H. Saïdi& R. Anderso,(2012) 'Aurasium: Practical policy enforcement for android applications', 21st USENIX conference on Security symposium, pp. 27-27.

[47] Y. Zhou, X. Zhang, X. Jiang & V. Freeh,(2011) "Taming information-stealing smartphone applications (on Android)", TRUST 2011, pp. 93-107

[48] Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X. S., and Zang, B. (2013). Vetting undesirable behaviors in android apps with permission use analysis. In Sadeghi, A.-R., Gligor, V., and Yung, M., editors, The 2013 ACM SIGSAC conference, pages 611– 622

[49] Zhu H-J et al (2018) DroidDet: effective and robust detectionof android malware using static analysis along with rotationforest model. Neurocomputing 272:638–646. https ://doi.org/10.1016/j.neuco m.2017.07.030

[50] The data that support the findings of this study are openly available at https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html