Machine Learning-Based Android Malware Detection with only Nine Permissions

ABSTRACT

The use of mobile phones, particularly smartphones, has been growing exponentially in recent times. From 2016 to 2021, smartphone users increased by more than 70 percent [28]. With the increase in the popularity of smartphones, smartphones have become the prime target for criminal hackers. As a result, android malware samples are coming to the market at an alarming rate. A study shows that there are more than 4 million malicious android apps in the market, and each day around 11 thousand new malwares add to this number [22]. To combat this mass number of malware, we need a malware detection system that is efficient in detecting malicious android apps. There are numerous existing malware detection systems, but most of them require countless features from both dynamic and static analysis. Thus, they are not scalable, lightweight, and efficient in detecting malware. Additionally, most studies that used limited features like only permission data, had done their research on much older dataset. Hence, there is a need for new research on this topic. In this paper, we build a permission-based malware detector for android application with a new dataset and significantly less permissions. Initially, we used support vector machine (SVM) and all the extracted permission data as features to build our classification model. The model accuracy, precision, recall and f1 score were 97.41 percent, which is higher than the other state of the art similar approaches done on an older dataset. Next, we replicated this similar study with a few different machine learning algorithms: random forest, decision tree and logistic regression, and observed they all give similar results. However, tree-based algorithm performs a little better than the other algorithms. Finally, to achieve a lightweight malware detection system, we reduced the number of permissions or the features on a two-step process, and found only a slight difference in results. In the first step, even after reducing the number of permissions by about 94 percent, the accuracy dropped by only 2.7 percent. In the second step, we further reduced the number of features or permissions and observed the difference in results. We managed to prune to 9 permissions while maintaining accuracy of 93 percent, which is lower than technique mentioned in other literature to reduce features.

KEYWORDS

malware, android, android permissions,data mining, machine learning, static analysis

1 INTRODUCTION

There are more than three billion smartphone users worldwide, and this number is expected to rise in the coming years [17]. According to a recent report, smartphone shipments worldwide are expected to reach 1.48 billion units by the end of 2023 [18]. The report also mentions that 46.45 percent of the world population currently has smartphones. The rise of smartphone usage is more prominent in the android platform. Android owns the majority of the smartphone market. According to a study, android current market share as a mobile operating system is more than 70 percent [16]. Due to its massive popularity as an operating system, android is also the prime target for criminal hackers. According to a recent article, 98 percent of mobile malware targets android devices [20]. Due to being an open-source mobile operating system, android is vulnerable in nature. Unlike iOS, android allows users to download and install applications from unauthorized sources. As a result, there is a significant risk for users downloading malicious android apps from the internet. There are numerous types of android malware, including bank trojans, ransomware, crypto mining, stalkware, ads click fraud, and many more. Many of these result in personal information leaks, use of hardware components for crypto mining, banking information leaks, and etc. According to a new report, 11 thousand new android malware samples come to the market every single day. Due to this rapid increase in malware size and types, detecting malicious apps is becoming very difficult. Therefore, there is an urgency for building a malware detector that can detect malicious applications effectively and efficiently.

Table 1: Static vs Dynamic

Static	Dynamic
The application is not run	The application is run and monitored
Permission-based approach	Anamoly-based detection
Signature-based approach	Behavior-based detection
Not suitable for complex malware detection	Suitable for complex malware detection

There are two types of analysis for malware detection. One is the static analysis and the other is the dynamic analysis as shown in table 1. In static analysis, the source code of the application is used in the analysis process. It analyzes the functionalities of the app to determine whether the given app is vulnerable or not. However, this is not a practical solution since we completely ignore the behavior of the application during the analysis. On the contrary, in dynamic analysis, the application is run, and the behavior of the app is used for the analysis. This is also not an effective solution since some behavior may be missed during the analysis process. The app's requested permissions on the other hand, tells both the functionalities and behaviors of the apps. Hence, permissions are very good features for android malware detection. There are plenty of studies that used permission data for building their malware detection system. Drebin [1], for instance, used permissions as well as many other features extracted with static analysis to develop their detection model. However, increased features increase the model complexity and deteriorate the model's performance. As a result, there is a need for detection system with less features for efficient performance and scalability. According to Google [9], there are 30 permissions that are considered as dangerous permissions. We can utilize this list of permissions to reduce our feature number and build an efficient malware detection system.

In this paper, we introduce a permission-based android malware detector with significantly fewer features or permissions. Our approach extracts the app's requested permissions and then only

considers the significant permissions to build our classifier model. We propose a method to reduce the app's requested permissions on a two-step process. First, we consider only Google's recommended dangerous permissions and discard the remaining permissions. Lastly, we propose a novel way of further reducing the permissions using the benign and malware matrices. As a result of pruning the permissions, we were able to build a detection system that is efficient and effective. As stated earlier in the paper, android malware samples are rising at an alarming rate. Thus, an efficient malware detector is a necessity.

First, our approach involves extracting all the app's requested permissions from a new dataset containing only the latest android malware samples. We used all the extracted permissions as features to build the first classifier model using SVM. The result of our initial approach shows that our model's accuracy, precision, recall, and f1 score was 97.4 percent. Compared with other state of the art approaches, for example, Drebin, Arslan et al. [3], and Talha et al. [26], our results show that our approach outperforms them in terms of malware detection accuracy. In next step, our approach involves replicating the same experiment with different machine learning algorithms, namely logistic regression, random forest, and decision tree. Our results show that all the different machine learning algorithms have similar accuracy, which is between 96-97 percent. In our last step of our approach, we successfully reduced the number of app's requested permissions on a two-step process. Firstly, we discarded all the permissions except the Google's recommended dangerous permissions. After the pruning, there were only 23 permissions left, which is about a 94 percent reduction in terms of permission's number. The reduced permissions are then used as features for training our SVM machine learning model. With a 94 percent drop in permission, our model accuracy dropped by only 2.7 percent. In the second step of our process, we further pruned the permissions using the difference of benign and malware matrices. Our model's accuracy dropped on each permission reduction. However, the drop was not significant. Even with only 9 permissions our model's accuracy was above 93 percent.

This paper has made the following contributions:

(1) We build an android malware detecting system with a new dataset containing only the latest malware samples. We only used the app's requested permissions as features to build our model. Thus, our model is simple and efficient. Even with only permission features, our model outperforms the existing state of the art approaches like Drebin, Arslan et al. and Talha et al. in terms of model's accuracy.
(2) We found out that different machine learning models do not significantly influence the results. We replicated our study with a few other machine learning models: logistic regression, decision tree, and random forest. Our results show that they all have similar model accuracy, which is between 96 – 97 percent.
(3) We successfully reduced the number of permissions on a two-step process. Pruning the app's requested permissions causes only a small drop in accuracy. In the first step, reducing the permissions by 94 caused a decrease in accuracy by only 2.7 percent. In the second step, even after pruning the number of permissions to only 9, the accuracy was still above 93 percent, which outperforms other state of the art approaches, for example, Li et al [13] managed to reduce 22 permissions for maintaining accuracy above 90.

The rest of the paper is organized as follows: Section 2 discusses the related papers around the topic. Section 3 talks about the data collection procedure and how the data is used. Section 4 introduces the methodology used in this paper. Section 5 discusses three research questions and the results. Section 6 mentions the threats to validity and limitation. Finally, Section 7 concludes the paper.

2 RELATED WORKS

Android malware detection is a very trendy research topic. With the rise of global smartphone users, particularly android users, android malware is also on the rise. Due to this rapid growth of malware, android malware research has become a prominent research field.Several researchers have already invented several innovative methodologies to address the issue of android malware.The approaches can be broadly categorised into three categories: static analysis, dynamic analysis and machine learning based on features extracted from static and/or dynamic analysis.

2.1 Static Analysis

Numerous researches have been done in the field of static analysis for android malware detection.Static analysis based on signature is a widely used approach for detecting malware applications. The process involves extracting and storing signatures from test applications. The signatures are then compared with benign and malicious applications to see which one is benign and which one is harmful. One drawback of this approach is that it cannot detect unseen new malware. In a study by Kang et al. [11], they proposed a signature-based method that detects and classifies malicious android applications by incorporating the creator's information as features. Their results showed that their system could detect android malware faster with the help of the creator's information like the serial number of certificates. They achieved a detection accuracy of 98 per cent. One drawback of their approach is that it has a high false-positive rate. The author believes that they can overcome the high false-positive rates by adding dynamic analysis to their detection mechanism. Another signature-based approach by Faruki et al. [8], introduced a method for detecting malware using statistically robust features. The proposed system is suitable for detecting unknown malware. Their method can classify malware with 60 per cent accuracy. Another study by Song et al. [25] used the filtering method for detecting malicious applications. They used four layers of filtering mechanisms, including MD5 values, the combination of malicious permissions, dangerous permissions and dangerous intents. Their evaluation results showed a high accuracy of 99 percent.

Static analysis also involves permission-based analysis. Several researchers have explored this area of research. Rovelli et al. [21] introduced a permission-based malware detection system (PMDS) that views requested permissions as behavioral markers. They build the machine learning classifier based on these markers to classify malware. Their system successfully detected 92 – 94 percent of unseen malware accurately. Another study by Wu et al. [30] proposed a system that characterizes an application based on many static

information such as permissions, intent message passing, API calls etc. They used a clustering algorithm, namely K-means, to further enhance the modelling capacity. Compared to Androguard, their system is faster since it takes half the time than Androguard to predict android malware and benign applications.

2.2 Dynamic Analysis

In dynamic analysis, the behavior of the application is analyzed to detect malicious and benign applications. The application is run to observe the behavior. For example, in a study by Vidal et al. [29], the authors used boot sequences to determine the malicious application. The authors only analyzed the system calls sequences during the booting process to identify a pattern for malware detection. One drawback of their approach is that their system has high false-positive rate. In another study by Shankar et al. [24], the authors introduced *Anti-Hijacking*, which is a malware detector that uses honey pot to detect malware at runtime with accuracy as high as 96 percent. The authors basically focused on two types of attacks: intent-based hijacking and authenticated session hijacking. Chaba et al. [7] used system call logs to observe the behavior of the malicious app while running it in the host system. The system call logs are then used to build a dataset that the authors used to detect unknown applications as benign or malicious. They used their dataset on three classification algorithms: Naïve Bayes, Random Forest and Stochastic gradient descent and observed accuracy of greater than 93 percent in all three algorithms. One advantage of their approach is that they detect malicious applications in a shorter time than other approaches. Sing et al. [4] introduced android application sandbox (AASandbox). AASandbox performs both static and dynamic analysis for malware detection. In static analysis, AASandbox scans applications for malware patterns without installing them. During dynamic analysis, it installs the application in a fully isolated environment and looks for malware patterns for malware detection. Several studies utilized anomaly behavior monitoring in their dynamic analysis approach. A study by Saracino et al. [23] proposed a malware detecting system based on anomaly behavior monitoring. Although this method could able to detect unknown applications, the false positive rate is high.

2.3 Machine Learning

Drebin [1] introduces a lightweight android malware detection method that can identify malicious android applications directly on the phone. It analyzes several features extracted from broad static analysis to detect and identify malware. Compared to other state-of-th- art approaches, it successfully detected android malware with 94 percent accuracy with few false positives. However, the dataset they used contains only 5560 malware samples. Arslan et al. [3] propose a malware detection system based on the app's requested permissions and app's used permissions. The requested permissions are extracted from the manifest.xml file, and the used permissions are obtained from the source code. If there is a mismatch between the requested permissions and the used permissions, the app is considered suspicious, and a suspicion value is assigned. An app with a suspicion value greater than the mean is deemed to be malicious. Their classification accuracy was 91.95 percent. APK auditor [26] is a signature-based android malware detector that uses static analysis for detecting malicious android application. It consists of three components: 1) A central database for storing the results of the static analysis of apps. 2) Android client application for the user to grand the analysis request. 3) Central server for maintaining the communication between the database and client.The server is also responsible for the analysis task. APK auditor's malware detection accuracy was 88 percent. However, it has a high false-positive rate of 0.9 percent compared to other approaches. It only utilizes the requested permissions for malware detection. If the source code is also analyzed to extract used permissions, their results could be improved.

McLaughlin et al.[15] propose a novel deep learning based approach for android malware classification. The classification is performed on raw opcode sequences obtained from the static analysis of application samples. Their approach can simultaneously extract features and classify applications. This eliminates the need for hand-engineered feature extraction. Another study conducted by Yuan et al. [33] utilizes deep learning-based approach to build their malware classifier. They used 200 features extracted from both static and dynamic analysis to feed their model. Their model accuracy was 96 percent. However, their study did not mention their false positive and false negative rates. In another paper, Yerima et al. [32] proposed a machine learning-based technique using a Bayesian classifier for android malware detection. In this paper, the authors used reverse engineering to analyze and collect 58 different features using the APK analyzer tool. They further reduced and ranked their features. Out of 58 properties, they found that only 15 to 20 properties are required for optimal classification performance. MalDozer [12] is an automatic, efficient, and effective malware detection system that relies on deep learning-based approach. MalDozer uses sequences of API calls to learn patterns for detecting malicious android applications. It was evaluated on multiple datasets containing benign and malicious applications. MalDozer successfully detected 96 – 99 percent malware with meager false-positive rates. SAMADroid [2] is a novel hybrid android malware detection model that uses both static and dynamic analysis. It also consists of a local and remote host to identify malicious applications. Its accuracy is higher (99 percent) since it consumes less storage and power. However, its performance is reduced since its communication is dependent on the server. Xiao et al. [31] propose an approach based on application system call sequences that use a backpropagation neural network to identify malicious applications. They used system call pattern for the classification task. Their f1 score was 98.2773 percent. Another study conducted by Tobiyama et al. [27] used recurrent neural networks and convolution neural networks to detect android malware. It is a two-step process. The features for the classifications are extracted using a pre-trained recurrent neural network, and the classification of the feature images is done next using a convolution neural network. They evaluated their approach with AUC, and their result was AUC = 0.96. However, they evaluated their results on a very small dataset. Hasegawa et al. [10] proposes a lightweight malware detection method based on 1-D convolutional neural network. A small portion of the apk file is analyzed without decompression using CNN. They achieved a detection accuracy of 97 percent. Although, this approach captures the key features for the malware detection, but they can not be confirmed since apks are compressed file.

Brown et al. [5] introduced an approach that utilized the human immune system and information flow in applications to determine malicious and normal android applications. Their results showed that their approach could successfully detect 93.33 percent of malware with fewer false-positive rates. Despite the high accuracy, their approach gives a high false-negative rate. Also, they used a small dataset for the evaluation of their approach. In another study, Canfora et al. [6] propose two ways of detecting malicious android applications. First, they used the Hidden Markov model and then structural entropy to detect android malware. They used the source code for analyzing. They used structural pattern algorithm for malware detection. The results shows that they achieve a precision of 96 percent. However, the result could be further improved by using a larger dataset. Li et al. [14] introduced an approach for detecting malicious android applications using feature code. They extracted the function calls and system calls from malicious and benign applications. These features are then used for training and classification using machine learning and data mining algorithms. Finally, they created a feature library and model library. Their results showed that their approach could effectively detect new unknown malware. Their results showed high accuracy with low false-positive rates. However, in their study, they considered fewer behavioral features. Li et al citeli2018significant proposes a significant permission identification (SPI) approach for machine learning-based detection of Android malware. The proposed approach utilizes the permissions requested by an application to construct a feature vector that is used to train a machine learning model. The SPI approach selects a subset of the most significant permissions based on their relevance to malware behavior. The experimental results show that the SPI approach achieves high accuracy in detecting Android malware compared to other state-of-the-art methods, but they still used high number of features or permissions.

In summary, there is a need for new research for building light weight malware detector with as less features as possible.

3 DATA COLLECTION

Apk file is kind of a zip file with all the files and directories compressed in .apk format. All apk files contain manifest.xml file, which contains all the essential features and characteristics of the app like entry points, permissions, libraries, the minimum version of android OS needed to run the application, and many more. Many of these features can be extracted from the apk file in order to use them in the static analysis of malware. The asset directory contains all the assets and resources that the app will use during its execution. The lib directory contains all the native libraries that the app requires for this execution. The signature directory or the meta -inf contains the metadata such as app signatures. The file named resources.arsc has all the precompiled resources. Classes.dex file is the most critical file in the application as it contains all the class files compiled in .dex format. The res directory contains resources that are not compiled and put in the resources .arsc file. It basically contains natively compiled code. Figure 1 gives an overview of the android application file structure.

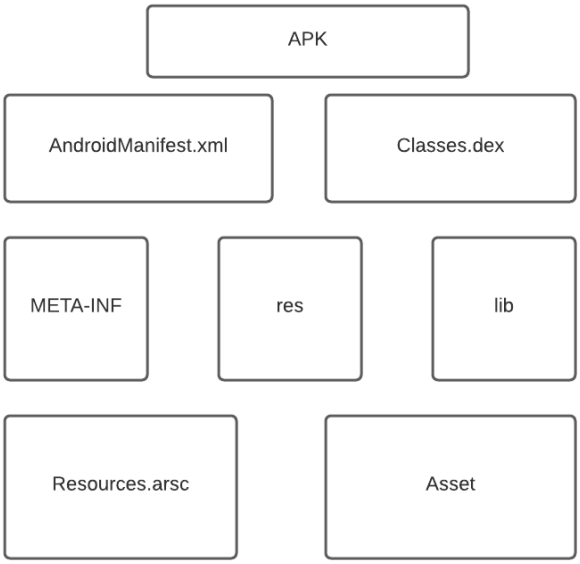


Figure 1: Apk File Structure

3.1 Data Source

The dataset used in this study is from CICMalDroid 2020 [19]. It is a new android malware dataset that is big, recent, diverse and comprehensive.It has more than 17 thousand android samples. CICMalDroid is a new dataset containing all the latest android malware sample up to the year 2018. It is a diverse dataset with malware from different categories including adware, banking, riskware, SMS and benign. The dataset also contains CSV files of several different features extracted with the help of static and dynamic analysis. However, for this particular study, we ourselves extracted the permission features from the android application files.

The dataset is collected from several sources such as VirusTotal service, Contagio security blog, AMD, MalDozer and other sources used by recent research papers. It contains a wide variety of categories. An overview of each category is as follows:

- **Adware:** Mobile adware displays third-party advertising material (i.e. ads) on the victim's phone. Adware hides in a normal application, and they may look harmless, but once installed, the adware may continuously display pop up ads even if the victim tries to force close the application. It may also be responsible for personal information leaks.
- **Banking:** Mobile banking malware or bank trojan malware refer to the malware that is specially designed to mimic the original user interface of the bank. This way, such malware trick victim to put their login information in the fake app. The hacking technique used in this process is called phishing.
- **SMS:** SMS malware is specially designed to exploit the SMS service of the victim's phone to steal sensitive and confidential information from the victim's phone. The attackers send malware SMS to the victim's phone, tricking the victim into opening the attached malware or clicking a malicious link.

- **Riskware:** Mobile riskware refers to such applications that are not strictly malicious but have the potential to turn into malicious applications in the form of adware or ransomware. In simple words, applications are called riskware whose installation poses a security risk to a device or the stored data.
- **Benign :** This category of applications contains normal apps that do not pose any security risk to the users and the stored data. They are safe and free from any malware.

3.2 Data Preparation

In this study, we are only concerned about the permission data. For each application file (apk), the app's requested permissions are extracted from the manifest file. The "pyaxmlparser" python library is used to extract the permissions. Figure 2 shows an overview of the extraction procedure.

Figure 2: Data Preparation

All the extracted permissions are used to build the permission table. The columns in the permission table represent the different types of permissions, and the rows of the table represent the different applications. Each row represents the list of permissions present in that respective application. The last column of the permission table contains the class value where 1 represents the malicious application, and 0 represents benign applications.

- **1:** Malware application
- **0:** Benign application

Figure 3 illustrates a snapshot of the permission table. The permission table is then used to feed our machine learning model, where we used all the permissions as features for our classification of benign and malicious applications.

permission.READ_CALENDAR	permission.RECEIVE_WAP_PUSH	permission.CAMERA	permission.PROCESS_OUTGOING_CALLS	permission.CALL_PHONE	class
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	0	0	1
0	0	1	0	0	1
0	0	0	0	0	1

Figure 3: Permission Table

4 METHODOLOGY

The main aim of this paper is to build a lightweight android malware detector using permission features. To achieve this goal, initially, we used all the extracted permissions as features to feed our model. For our initial model, we used support vector machine (SVM) for the classification of malware and goodware. We used SVM as the initial model because it is one of the most robust and accurate algorithms among the other classification algorithms. We used performance metrics such as accuracy, precision, recall, and f1 score for the evaluation. Figure 4 illustrates an architecture overview of our system

In the second phase of our study, we replicated the same experiment with other machine learning algorithms like logistic regression, decision tree classifier and random forest classifier. The aim of this phase is to find the best performing machine learning algorithm for the malware classification problem. For each of the machine learning algorithms, we used the default settings i.e. we kept the parameter same as default.

In the last phase of our study, the aim was to prune the permissions features without affecting the classification results. This approach, in effect, eliminates the need to analyze the app's requested permissions that do not affect the classification of malware and goodware. As a result, our classification would be more efficient and effective since we now need to analyze significantly fewer permissions to determine malicious android applications. In summary, our approach prunes the permission on a two-step process:

4.1 Google Dangerous Permission

Google considers some permissions as dangerous permissions since these permissions might give the app the permission to access sensitive information and perform restricted actions that might affect the system or other application [9]. There are 30 such permissions as listed in the table 2. We reduced the permissions' number using this list of permissions. We only considered the dangerous permissions to build our new permission table. The rest of the permissions are discarded. As a result, the permission table is now much smaller containing only the dangerous permissions. The reason for pruning the permission with Google's dangerous permission list is that dangerous permissions significantly influence the effectiveness of malware classification compared to the other permissions.

4.2 Permission Ranking

In the second step of pruning, we divide our permission table into two matrices: benign matrix and malware matrix. The benign matrix consists of the permission matrix of only benign applications, and the malware matrix consists of the permission matrix of malware applications. For the matrix creation, we only used the reduced permission table obtained after the first pruning approach, since in the second step, our main aim is to reduce the permission features further. We compared the two matrices in terms of permission features for the pruning process. For each permission feature, we compared the presence of that respective permission between malware samples and benign samples and assigned a significance score. For instance, as shown in the figure 5, two matrices are present: benign matrix and malware matrix. These two matrices are made using the reduced permission table. Each permission is considered and compared between the matrices. For example, when *P1* is compared, we calculate the presence of *P1* in benign matrix and the malware matrix i.e. we calculate the number of ones in the *P1* column between the two matrices and compare the result. According to the above example, *P1* is present same number of times in both benign and malware matrix i.e. one time. Hence, *P1* will not be significant in classifying malware and normal applications. However, if we consider *P3*, it is present three times in malware samples compared to the benign samples. Thus, *P3* is a significant permission for determining malicious android applications.

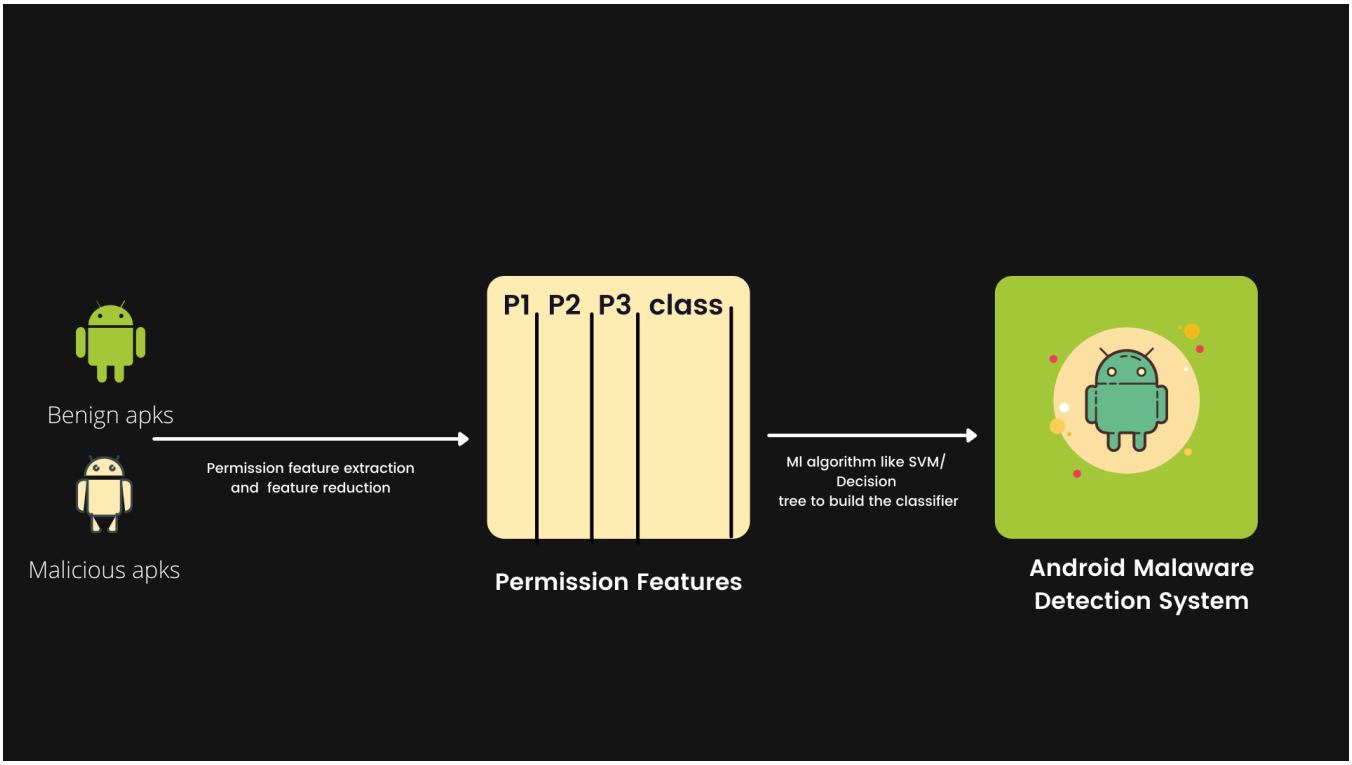


Figure 4: Architecture Overview

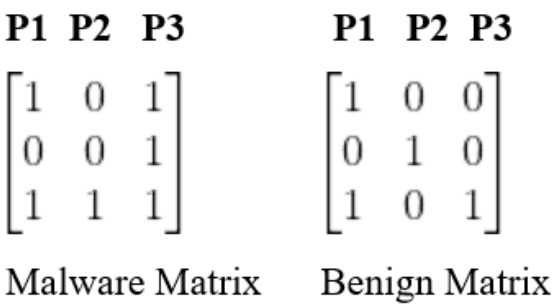


Figure 5: Malware Matrix and Benign Matrix

The significant score is based on the difference in presence of permission between malware samples and benign samples. For instance, a high significance score of a permission means that there is high difference in presence of this permission between the malware and goodware applications. The significant score is calculated with the following equation:

$$S(P)_j = \frac{size(B_j)}{size(M_j)} \times |\sum_i B_{ij} - \sum_i M_{ij}| \quad (1)$$

where B represents the benign matrix and M represents the malware. $S(P)_j$ denotes the significant score for j_{th} permission, $size(B_j)$ and $size(M_j)$ refer to the size of the benign and malware matrix respectively. We are considering the ratio of benign and malware matrix because our dataset is imbalanced. It has different number of benign samples and malware samples.

The aim of our pruning is to create a ranked list of permissions. The rank is based on the significance score. Higher the significance score of the permission, higher its rank in the ranked list. In our approach, based on our rank list, we reduced the permission feature one by one and fed it to the machine learning model to calculate the accuracy with fewer permission features.

5 THREE PHASES

In this section, we aim to discuss and explain the three phases of our study.

5.1 Malware detection with only permission features

5.1.1 Motivation. During the literature review of android malware detection using machine learning techniques, we came across numerous brilliant studies with state-of-the-art methodologies and excellent empirical results. However, most of the recent studies around that topic used a combination of different features obtained from both dynamic and static analysis of applications for their detection model. Drebin [1] , for instance, used several features in combination with permission features to build their detection

Table 2: Dangerous Permissions according to Google

Dangerous Permissions
WRITE_CALL_LOG
READ_CALENDAR
WRITE_CALENDAR
CAMERA
READ_CONTACTS
WRITE_CONTACTS
GET_ACCOUNTS
ACCESS_FINE_LOCATION
ACCESS_COARSE_LOCATION
RECORD_AUDIO
READ_PHONE_STATE
READ_PHONE_NUMBERS
CALL_PHONE
ANSWER_PHONE_CALLS
READ_CALL_LOG
WRITE_CALL_LOG
ADD_VOICEMAIL
USE_SIP
PROCESS_OUTGOING_CALLS
BODY_SENSORS
SEND_SMS
RECEIVE_SMS
READ_SMS
RECEIVE_WAP_PUSH
RECEIVE_MMS
READ_EXTERNAL_STORAGE
WRITE_EXTERNAL_STORAGE
ACCESS_MEDIA_LOCATION
ACCEPT_HANDOVER
ACCESS_BACKGROUND_LOCATION
ACTIVITY_RECOGNITION

model. Similarly, Yuan et al. [33] used 200 features obtained from both dynamic and static analysis of the application to detect malicious android applications. However, using a large number of features leads to increased modeling complexity. They also increase the computation overhead, which ultimately leads to poor performance of the model. Thus, a study is needed where only one type of feature i.e. the permission feature is used for malware detection.

5.1.2 Approach. As mentioned previously, we collected our dataset from CICMalDroid 2020 [19], which is a new open dataset released by the University of New Brunswick. The dataset contains android application files, also known as apk files, in five different categories: adware, banking, SMS, riskware and benign. Only the benign category contains safe applications. Other categories like adware, banking, and SMS contain malware samples. We discarded the riskware category since riskware are not strictly malware. They just have the potential to become malicious and so are considered risky.

In the first step, we extracted all the requested permissions from the manifest.xml file of an application. We extracted all the permissions of the apps and built a permission table with class label. We used all the permissions as features to feed our model. In total, after analyzing all the apk files except the riskware applications, we extracted 390 permissions. All of these permissions are used as features to feed our model. We used the Support Vector Machine (SVM) algorithm for the classification task as it is efficient and correct compared to other classification algorithms. We used ten-fold cross-validation for model training and testing and calculated the mean results. As far as the model's performance is concerned, we evaluated the performance of our model in terms of the following metrics:

- Accuracy
- Precision
- Recall
- F1 score

5.1.3 Results. We only used the SVM algorithm for malware classification in this phase. As mentioned previously, we used four evaluation metrics to evaluate our model: accuracy, precision, recall, and f1 score. After ten cross-validation, our model's mean accuracy, precision, recall, and f1 score are around 97.4 percent, as shown in the figure 6.

Accuracy	Precision	Recall	F1
97.747	97.745	97.747	97.743

Figure 6: SVM performance with all permissions used as features

Compared to other studies that used many different types of features for analysis, our simple permission-based detector performed better as shown in figure 7 .For instance, if we compare with Drebin [1], even after using more numbers and types of features than us, its accuracy was 94 percent, around 3 percent less than our approach.

Similarly, our model performed better than Arslan et al [3], who used a different methodology for detecting malicious android applications. Their method involves calculating suspicion scores based on the difference between requested and used permissions. Yuan et al. [33] who utilized 200 different features obtained from static and dynamic analysis achieved a lower accuracy score than our approach. In summary, in phase 1, we build a permission only classifier that outperforms other state of the art approaches.

Accuracy	Precision	Recall	F1
97.747	97.745	97.747	97.743

Figure 7: Accuracy comparison with other approaches

5.2 Comparing different machine learning algorithms in malware detection

5.2.1 Motivation. Each model or any machine learning algorithm has several features that process the data in different ways. Thus, they have a difference in performance. The challenge is finding the best-performing machine learning algorithm for a given type of dataset. This phase aims to solve this challenge and find the best-performing model that suits the permission data.

5.2.2 Approach. For phase 2, we replicated the same methodology used in phase 1, but for the classification of malware, we now used three other types of machine learning models, including logistic regression, decision tree classifier, and random forest classifier. Same as before, we used all the 390 permissions as features for malware detection. During training, for each machine learning model, we kept parameter in its default setting. Ten fold cross-validation was used for the training and the testing purpose, and the mean accuracy was recorded and compared between the different machine learning models.

5.2.3 Results. The results show that the other machine learning algorithms have similar performance. We only compared the accuracy between four machine learning models and observed that their accuracy was between 96 – 97 percent as per figure 8. The results show that logistic regression has the lowest accuracy of 96.09 percent, and the random forest has the highest accuracy of 97.61 percent.

Thus, in conclusion, different machining learning algorithms do affect the performance of the model. However, the difference in model performance is very minimal.

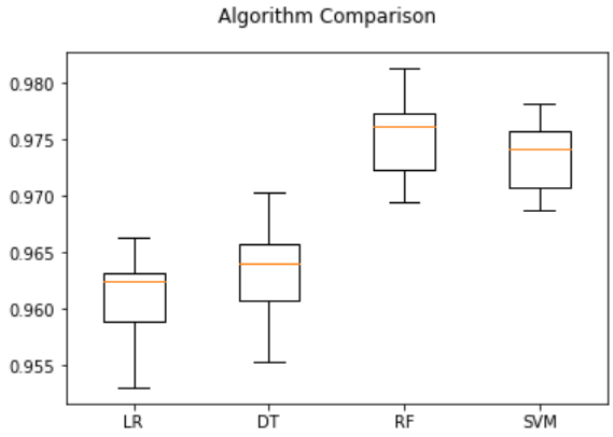


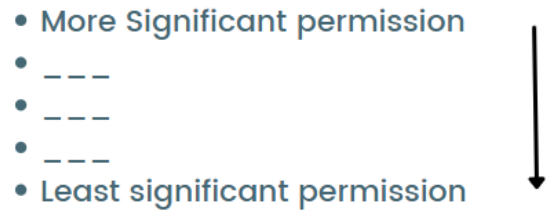
Figure 8: Machine Learning Algorithms comparison

5.3 Feature reduction or Permission pruning

5.3.1 Motivation. Smartphone popularity is increasing exponentially so does the malware types and numbers. With the largest market share of mobile operating systems, android has become the prime target for criminal hackers because it is an open-source OS with less security features than other OS like iOS. Due to the rapid increase in malware numbers, we need a malware detection system that is quick and efficient. Analyzing large number of features vectors for malware detection causes an increased computation overhead. More number of features means increased modeling complexity which leads to poor performance and inefficient system. As a result, feature selection and reduction are a must for efficient and faster performance. In this phase, we aim to determine if it is possible to prune the permission features without significantly affecting the model's performance.

5.3.2 Approach. For this phase, we reduced the permission features using a two-step process: Google dangerous permissions and permission ranking. Google [9] considers 30 permissions as dangerous permission since these permissions can allow the app to have additional access to restricted data and perform additional restricted actions. In our approach, we completely discarded the permissions except for the dangerous permissions. We only considered the dangerous permissions as features for detecting malicious apps because malware samples are more likely to have these 30 permissions than benign ones. After discarding all the other permissions, our permission table only had 23 permissions; all of them are dangerous permission as per Google. These 23 permissions are then used as features for android malware detection. Same as phase 1, 10-fold cross validation is used for the classification, and SVM is used as the classification algorithm.

In the second step of our multi-level pruning, we further reduced the permission features using a novel technique of permission ranking. We initially made two matrices from the permission table: benign matrix and malware matrix. As mentioned previously, for each permission feature, we compare the presence of it in the benign and the malware samples. Comparing the presence of permission in the benign and the malware apps, we assign a significance score for the permission. We assign a higher significance score for permission if it is present in different numbers in benign and malicious applications. The significance score is calculated as per equation 1 where M represents malware matrix, and B represents benign matrix. The ratio of benign to malware samples is considered because our dataset consists of imbalanced data, where we have more samples of malware applications than benign applications. The significance score is an indicator of whether a permission is significant in detecting malware or not. With the signification score, we rank the permissions from highest significant score to lowest significance score. Based on the ranked list of permissions, we pruned the



permission features one by one. The reduced permissions are then used as features for detecting malware applications. We reduced

the permissions features one after another and observed the results on the classification accuracy.

Using the list of dangerous permissions, we pruned permissions to only 23 permissions. The reduced 23 permissions features are then fed to the SVM model. The model achieved an accuracy, precision, recall, and f1 score of 94.77 percent, which is just a drop of 2.78 percent. Initially, we extracted around 390 permissions from the apk samples. After pruning, there were only 23 permissions left in the permission table. Therefore, there is a reduction of 94.1 percent in the number of permissions. Hence, a drop in accuracy of 2.78 percent for 94.1 percent drop in permission number is not that much.

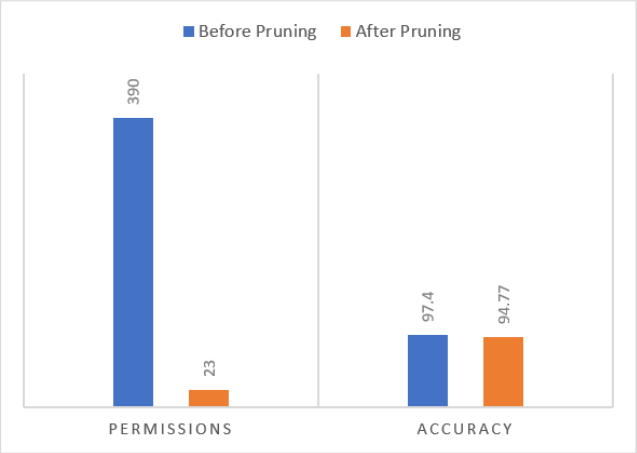


Figure 9: Accuracy drop vs Permission drop

In the second step of our pruning process, we calculated the significance score for each permission, as shown in figure 10. Here, *READ_PHONE_STATE* is the permission with the highest significance score, and *BODY_SENSORS* has the lowest significance score. This means that *BODY_SENSORS* is used by both malware and benign samples equally; thus, it is not a significant feature that can classify malware and benign applications.

```
[('permission.READ_PHONE_STATE', 3131.2973621103115),
 ('permission.SEND_SMS', 3034.438848920863),
 ('permission.RECEIVE_SMS', 2747.091926458833),
 ('permission.READ_SMS', 2098.601119104716),
 ('permission.WRITE_EXTERNAL_STORAGE', 1810.792965627498),
```

Figure 10: Top 5 permissions based on significance score

Based on the ranked list of permissions, we reduced permission one by one starting with the first permission *BODY_SENSORS*. Our results show that even after reducing the permissions to only 9 permissions, our dectection accuracy was still above 93 percent meaning it was only dropped by one percent. Figure 11 illustrates how pruning the permission based on significance scores affects the perform of the model. As we reduced the permission, our accuracy dropped. However, the drop is minimal. Even pruning permissions to only 9 permissions caused our accuracy to drop by only one percent.

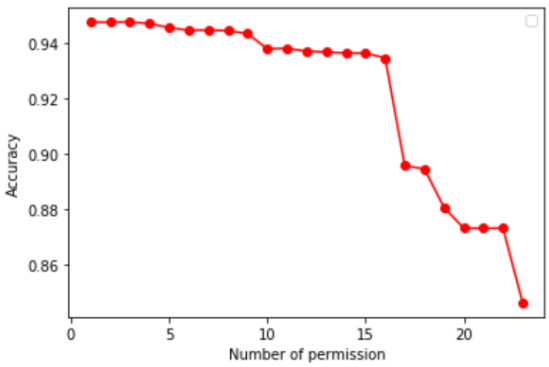


Figure 11: Accuracy on reducing less significant permissions

6 THREATS TO VALIDITY AND LIMITATIONS

In this section, we will discuss the threats to validity and limitations.

6.1 Deep Learning

In our approach, we only used traditional machine learning algorithms like SVM, logistic regression. There are many existing studies with good results based on deep learning algorithms. In our phase 2, for model comparison, the results would have been more reliable if you have replicated our study with a deep learning model and compared the results

6.2 Only used one type of feature

In order to achieve a light weight malware classifier, we only used one type of feature, that is permissions. There is a possibility of sophisticated malicious app with typical permissions but logic bomb. They cannot be detected.

6.3 Riskware

During data preparation, we completely ignored one of the categories in our dataset: riskware. We discarded riskware since they are not strictly malware, and thus it would have negatively affected our classification result.

6.4 Google's Dangerous Permissions

We used Google's dangerous permissions to prune the permissions. However, these permissions might exist in benign applications too. If that is the case then the classification performance is not optimal, and could be improved with other better approaches like the mentioned permission ranking approach.

7 CONCLUSION

Android malware samples are flowing to the market at an alarming rate. To tackle this surge, we need a malware detection system that is efficient, fast and scalable. Analysis with a small number of features is preferable since it is efficient and has less computation overhead. This paper aimed to use only the permission features to detect malware. The paper also introduced two pruning methods to further reduce the feature/permissions without affecting the model's performance significantly. Initially, we analyzed 12,796

application samples to extract the permission features. We successfully collect 390 different permissions. We used each of these permissions as feature to build our detection system. SVM model is used for the classification of malware and benign. We successfully detected malware with 97.40 percent accuracy, which is higher than other approaches that use many other features in addition to permission features.

The aim of our study is also to reduce features for classification for building a lightweight model. We introduced two methods of pruning: Google dangerous permissions and novel permission ranking. After reducing the permissions based on Google's dangerous permissions, we achieved a drop of 94.1 percent in permission numbers while only losing 2.78 percent inaccuracy. Lastly, based on novel significant permission ranking, we pruned permissions from 23 to 9 by only dropping one percent in accuracy, which is lower than other state of the art approaches like Li et al. [13] who managed to reduce to 22 permission for above 90 percent accuacy.

REFERENCES

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

[2] Saba Arshad, Munam A Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. 2018. Samadroid: a novel 3-level hybrid malware detection model for android operating system. *IEEE Access* 6 (2018), 4321–4339.

[3] Recep Sinan Arslan, İbrahim Alper Doğru, and Necaattin Barişçi. 2019. Permission-based malware detection system for android using machine learning techniques. *International journal of software engineering and knowledge engineering* 29, 01 (2019), 43–61.

[4] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. 2010. An android application sandbox system for suspicious software detection. In *2010 5th International Conference on Malicious and Unwanted Software*. IEEE, 55–62.

[5] James Brown, Mohd Anwar, and Gerry Dozier. 2016. Detection of mobile malware: an artificial immunity approach. In *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 74–80.

[6] Gerardo Canfora, Francesco Mercaldo, and Corrado Aaron Visaggio. 2016. An hmm and structural entropy based detector for android malware: An empirical study. *Computers & Security* 61 (2016), 1–18.

[7] Sanya Chaba, Rahul Kumar, Rohan Pant, and Mayank Dave. 2017. Malware detection approach for android systems using system call logs. *arXiv preprint arXiv:1709.08805* (2017).

[8] Parvez Faruki, Vijay Laxmi, Ammar Bharmal, Manoj Singh Gaur, and Vijay Ganmoor. 2015. AndroSimilar: Robust signature for detecting variants of Android malware. *Journal of Information Security and Applications* 22 (2015), 66–80.

[9] Google. 2021. *Permissions on Android.* Retrieved December 14, 2021 from https://developer.android.com/guide/topics/permissions/overview

[10] Chihiro Hasegawa and Hitoshi Iyatomi. 2018. One-dimensional convolutional neural networks for Android malware detection. In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE, 99–102.

[11] Hyunjae Kang, Jae-wook Jang, Aziz Mohaisen, and Huy Kang Kim. 2015. Detecting and classifying android malware using static analysis along with creator information. *International Journal of Distributed Sensor Networks* 11, 6 (2015), 479174.

[12] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. 2018. MalDozer: Automatic framework for android malware detection using deep learning. *Digital Investigation* 24 (2018), S48–S59.

[13] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. 2018. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics* 14, 7 (2018), 3216–3225.

[14] Yiran Li and Zhengping Jin. 2015. An Android malware detection method based on feature codes. In *Proceedings of the 4th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering*.

[15] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. 2017. Deep android malware detection. In *Proceedings of the seventh ACM on conference on data and application security and privacy*. 301–308.

[16] S. O'Dea. 2021. *Android - Statistics & Facts.* Retrieved December 14, 2021 from https://www.statista.com/topics/876/android/

[17] S. O'Dea. 2021. *Smartphone users worldwide 2016-2021.* Retrieved December 14, 2021 from https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[18] S. O'Dea. 2021. *Smartphones - Statistics & Facts.* Retrieved December 14, 2021 from https://www.statista.com/topics/840/smartphones/#dossierKeyfigures

[19] University of New Brunswick. [n. d.]. CICMalDroid 2020. https://www.unb.ca/cic/datasets/maldroid-2020.html

[20] Purplesec. 2021. *2021 Cyber Security Statistics The Ultimate List Of Stats, Data & Trends.* Retrieved December 14, 2021 from https://purplesec.us/resources/cyber-security-statistics/

[21] Paolo Rovelli and Ýmir Vigfússon. 2014. PMDS: permission-based malware detection system. In *International conference on information systems security*. Springer, 338–357.

[22] safeatlast. 2021. *20 Scary Mobile Malware Statistics to Keep in Mind.* Retrieved December 14, 2021 from https://safeatlast.co/blog/mobile-malware-statistics/

[23] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. 2016. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing* 15, 1 (2016), 83–97.

[24] Venkatesh Gauri Shankar and Gaurav Somani. 2016. Anti-Hijack: Runtime detection of malware initiated hijacking in android. *Procedia Computer Science* 78 (2016), 587–594.

[25] Jun Song, Chunling Han, Kaixin Wang, Jian Zhao, Rajiv Ranjan, and Lizhe Wang. 2016. An integrated static detection and analysis framework for android. *Pervasive and Mobile Computing* 32 (2016), 15–25.

[26] Kabakus Abdullah Talha, Dogru Ibrahim Alper, and Cetin Aydin. 2015. APK Auditor: Permission-based Android malware detection system. *Digital Investigation* 13 (2015), 1–14.

[27] Shun Tobiyama, Yukiko Yamaguchi, Hajime Shimada, Tomonori Ikuse, and Takeshi Yagi. 2016. Malware detection with deep neural network using process behavior. In *2016 IEEE 40th annual computer software and applications conference (COMPSAC)*, Vol. 2. IEEE, 577–582.

[28] Ash Turner. 2021. *HOW MANY SMARTPHONES ARE IN THE WORLD?* Retrieved December 14, 2021 from https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

[29] Jorge Maestre Vidal, Marco Antonio Sotelo Monge, and Luis Javier García Villalba. 2018. A novel pattern recognition system for detecting Android malware by analyzing suspicious boot sequences. *Knowledge-Based Systems* 150 (2018), 198–217.

[30] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia Joint Conference on Information Security*. IEEE, 62–69.

[31] Xi Xiao, Zhenlong Wang, Qing Li, Shutao Xia, and Yong Jiang. 2017. Back-propagation neural network on Markov chains from system call sequences: a new approach for detecting Android malware with system call sequences. *IET Information Security* 11, 1 (2017), 8–15.

[32] Suleiman Y Yerima, Sakir Sezer, Gavin McWilliams, and Igor Muttik. 2013. A new android malware detection approach using bayesian classification. In *2013 IEEE 27th international conference on advanced information networking and applications (AINA)*. IEEE, 121–128.

[33] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 371–372.