BRAIN. Broad Research in Artificial Intelligence and Neuroscience
ISSN: 2068-0473 | e-ISSN: 2067-3957
Covered in: Web of Science (WOS); PubMed.gov; IndexCopernicus; The Linguist List; Google Academic; Ulrichs; getCITED; Genamics JournalSeek; J-Gate; SHERPA/RoMEO; Dayang Journal System; Public Knowledge Project; BIUM; NewJour; ArticleReach Direct; Link+; CSB; CiteSeerX; Socolar; KVK; WorldCat; CrossRef; Ideas RePeC; Econpapers; Socionet.

2021, Volume 12, Issue 4, pages: 28-56 | https://doi.org/10.18662/brain/12.4/237

Deep Learning Based Malware Detection Tool Development for Android Operating System

Mahmut TOKMAK[1],
Ecir Uğur KÜÇÜKSİLLE[2],
Utku KÖSE[3]

[1] Isparta University of Applied Sciences, Gelendost Vocational School, Isparta, Turkey, mahmuttokmak@isparta.edu.tr
[2] Suleyman Demirel University, Department of Computer Engineering, Isparta, Turkey, ecirkucuksille@sdu.edu.tr
[3] Suleyman Demirel University, Department of Computer Engineering, Isparta, Turkey, utkukose@sdu.edu.tr

Abstract: *In today's world that called technology age, smartphones have become indispensable for users in many areas such as internet usage, social media usage, bank transactions, e-mail, as well as communication. The Android operating system is the most popular operating system that used with a rate of 85.4% in smartphones and tablets. Such a popular and widely used platform has become the target of malware. Malicious software can cause both material and moral damages to users.*

In this study, malwares that targeting smart phones were detected by using static, dynamic and hybrid analysis methods. In the static analysis, feature extraction was made in 9 different categories. These attributes are categorized under the titles of requested permissions, intents, Android components, Android application calls, used permissions, unused permissions, suspicious Android application calls, system commands, internet addresses. The obtained features were subjected to dimension reduction with principal component analysis and used as input to the deep neural network model. With the established model, 99.38% accuracy rate, 99.36% F1 score, 99.32% precision and 99.39% sensitivity values were obtained in the test data set.

In the dynamic analysis part of the study, applications were run on a virtual smartphone, and Android application calls with strategic importance were obtained by hooking. The method called hybrid analysis was applied by combining the dynamically obtained features with the static features belonging to the same applications. With the established model, 96.94% accuracy rate, 96.78% F1 score, 96.99% precision and 96.59% sensitivity values were obtained in the test data set.

Keywords: *Android malware analysis, static analysis, dynamic analysis, hybrid analysis, deep learning.*

How to cite: Tokmak, M., Küçüksille, E.E., & Köse, U. (2021). Deep Learning Based Malware Detection Tool Development for Android Operating System. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience, 12*(4), 28-56. https://doi.org/10.18662/brain/12.4/237

1. Introduction

Nowadays, smart phones have occupied an important position in many users' lives and have become an indispensable part. Because the smartphone can be used not only as a phone, but also as a portable computer that provides various services, such as short message service (SMS), email, internet, social media, maps, GPS, banking applications, games (Alshahrani et al., 2018; Alzaylaee et al., 2020).

According to the data of International Data Corporation (IDC) in June 2020, when examining the ratio of operating systems in the global smartphone market, Android's share of the market is 85.4% and it is expected to be 86.3% in 2023 according to the company's estimate (IDC, 2020).

As an open source and widely preferred technology around the world, Android has become the target of malware. These malware have the ability to send text messages to special charge numbers, gain access to personal data, and even install code that can download and apply additional malware to the user's device without the user's consent. Malware can also be used to create mobile botnets (Alzaylaee et al., 2020; Anagnostopoulos et al., 2016). In the last few years, the number of malware samples targeting the Android platform has increased significantly. According to a report by McAfee in 2018, more than 2.5 million new Android malware applications were detected in 2017, so the number of malware samples increased to nearly 25 million till 2017 (Alzaylaee et al., 2020; McAfee, 2018).

The widespread use of Android and the increase the number of malicious software make it necessary to study on malware detection. In this context, Google Play introduced a detection mechanism called Bouncer in 2012 to prevent the spread of malicious software. In order to detect any malware behavior, the Bouncer application was run in a virtual area for five minutes and tested to detect malicious behavior. However, it has been shown that this detection system can be circumvented by malware (Alzaylaee et al., 2020). In addition, Google announced Google Play Protect in 2017. Google Play Protect is a service that works constantly to keep data and apps safe and tries to provide mobile security by automatically scanning the device. At the same time, Google reported that more than 50 billion apps are scanned every day with this service, regardless of where they were downloaded (Google Play, 2018). However, in the report published by McAfee, they stated that the Google Play Protect service is not successful when tested against malwares that detected in the last 90 days (McAfee,

2018). As can be seen, in addition to the study of Google Play, various approaches and studies are still needed to combat malware.

Various approaches have been proposed in previous studies to detect Android malware. These approaches are called static analysis, dynamic analysis or hybrid analysis. In the method called static analysis, it is based on the principle of analyzing the written code without running the application. Dynamic analysis is based on the principle of determining the behavior of the application by running the application in a controlled environment such as an emulator (Android Virtual Device- "AVD") or on a real device. The hybrid analysis method is based on the principle of combined usage of static and dynamic analysis method.

In this study, it is aimed to detect malicious software that threatens Android platform. During the static analysis phase, the Java application that we named Kuzgun was developed and the feature extraction process was performed in 9 different categories. In this respect, contrary to studies such as Aafer et al. (2013), Au et al. (2012), Feizollah et al. (2017), Hou et al. (2016, 2017), Idrees et al. (2017), Rosmansyah and Dabarsyah (2015), Yerima et al. (2014), Yuan et al. (2016), the study has been analyzed in detail by using other features such as intents and system commands that can be obtained from Android applications, instead of focusing only on permissions and API calls. After the feature extraction process, size reduction was performed using Python and malware was detected using deep neural networks. As a result of the static analysis, 99.38% accuracy rate, 99.36% F1 score, 99.32% precision and 99.39% sensitivity values were obtained in the test data set.

2. Related Work

In this section, a general literature review has been made on research studies for the analysis of Android malware.

Au et al. (2012) made detailed mapping of Android permissions and API calls in their study called PScout. In particular, they have matched API calls that are not documented by Android with the necessary permissions to use them.

Wu et al. (2012) developed a system called DroidMat. Firstly, they obtained API calls related to the requested permissions, intent messages, activities, services, recipients and permissions by using DroidMat. As a result, they made a classification with the K Nearest Neighbor algorithm.

Aafer et al. (2013), in their work called DroidAPIMiner, static analysis was performed using API calls obtained from application code through reverse engineering.

Yerima et al. (2014) studied three methods of detecting Android malware based on data mining. By using a special application developed by Java, they made Bayesian classification by using data obtained from Android application packages by automatic reverse engineering techniques. In the first of the models that they perform static analyzis, they used the standard Android permissions obtained from Manifest files. In the second model, they used API calls to point out potential harmful they got from code features. In the third model, they made a classification by using the Android permissions and API calls they obtained in the first two models.

Arp et al. (2014) developed a static analysis-based tool called DREBIN, which enables detecting harmful applications directly on smartphones. They gathered the attributes they obtained from the Android samples into 8 groups. The attributes that the application obtained from the manifest file are hardware properties, requested permissions, application attributes, intent filters. The attributes they obtain from the source code of the application are restricted API calls, used permissions, suspicious API calls, and network addresses. They tried to detect malicious software using Support Vector Machines, one of the machine learning methods.

Rosmansyah and Dabarsyah (2015) used API calls in their study which they used the static analysis method. They made classification with Random Forest, J48, and Support Vector Machines algorithms.

Fereidooni et al. (2016) used static analysis method in their study called ANASTASIA and created a malware detection system using machine learning techniques. They use the intents, the permissions used, system commands, suspicious API calls and malicious activities that obtained from the Android application as attributes. They obtained the best result with the XGBoost algorithm from the data they tested with machine learning techniques such as Extreme Gradient Reinforcement (XGBoost), Random Forest, and Support Vector Machines.

Hou et al. (2016), in their study called Deep4MalDroid, they extracted the Linux kernel system calls of Android applications and used the weighted graph method for attribute representation. Deep Learning method was used in the study.

Yuan et al. (2016), in their study called DroidDetector, they extracted attributes by static analysis and dynamic analysis, and then detected malicious applications by characterizing the attributes with Deep Learning method. In static analysis, they obtained Android permissions and sensitive

API calls. In dynamic analysis, they obtained 192 attribute titles by using hooking methods to obtain file input/output, short message service, encryption and network operation attributes of application behavior characteristics.

Hou et al. (2017) opened APK (Android Application Package) by using APKTool and extracted smali codes from dex files. They created Deep Neural Networks by dividing API calls obtained from Smali codes into blocks. Also made a comparison using deep learning methods such as Deep Belief Network, Stacked AutoEncoders. They found the Deep Belief Networks method more successful than the Stacked AutoEncoders method.

Feizollah et al. (2017) evaluated Android intents as a distinctive feature to identify malicious applications. They stated that the intents contain semantically rich features in identifying malicious software compared to other well-studied features such as permissions. Bayesian Networks was usen in the tests.

Idrees et al. (2017) proposed a permission and intent-based detection method in their study called PIndroid. They argued that Android permissions and intents are related, and they tested the attributes that consist of permissions and intents using machine-learning techniques.

Milosevic et al. (2017) used 2 different machine learning methods in their studies. They made classification and clustering in the study where they used the static analysis method. They said that the results obtained using classification methods showed better performance than clustering methods.

McLaughlin et al. (2017) performed a static analysis using the opcode arrays they extracted from the manifest file and the dex file. They trained their datasets with convolutional neural networks.

Karbab et al. (2018) conducted static analysis using API calls in their study called MalDozer. They used the deep learning method in their detection mechanisms.

Alshahrani et al. (2018), in their study named DDefender, used static and dynamic analysis techniques to extract features from the user's device and then applied a deep learning algorithm to detect malicious applications. Firstly, they used the dynamic analysis method to extract system calls, system information, network traffic, and required permissions from an audited application, then used the static analysis method to extract important features such as components of the application from the audited application.

Yang et al. (2018) proposed a dynamic analysis method called DroidWard in order to characterize harmful behavior, increase the rate of detecting harmful applications, and extract the most relevant and effective

features. They gathered the attributes in 15 separate categories, and stated 9 of them as classical methods and 6 as new. They used DroidBox while extracting dynamic attributes. However, they stated that the data monitored by DroidBox was limited, and they simulated precise API calls with Monkeyrunner by making changes to the source code of DroidBox. They use machine learning techniques such as Support Vector Machines, Decision Tree, and Random Forest.

Sugunan et al. (2018) conducted a comparative study on the behavior of harmful and harmless applications using static and dynamic analysis. They extracted static attributes using APKtool and dynamic attributes using Droidbox APIMonitor. They stated that the combined use of static and dynamic analysis features and the results of their tests with methods such as machine learning algorithms RF, SVM, J48 and Naive Bayes were more successful.

Cordonsky et al. (2018) proposed a system that uses static and dynamic analysis methods together. Cuckoo recorded features such as API calls, network activities, and string sequences obtained from malicious applications they analyzed in the sandbox in JSON format and used them as input to Deep Neural Networks.

Alzaylaee et al. (2020) conducted a study called DL-Droid, which detects malicious Android applications with dynamic analysis method. They used the Deep Learning method in the study. They have extracted dynamic attributes using the application named DynaLog developed by Alzaylaee et al. (2016).

Contrary to the previous studies, in the static analysis part of our study, the attribute category that can be effective for Android malware detection was evaluated in 9 different categories instead of a few specific categories (API, Permissions, etc.).

In our static analysis, a total of 62.547 examples of Android harmful and harmless applications were used, again numerically more than other studies, and approximately 750.000 different features were obtained from these samples. The attributes are reduced to 900 by the IPCA method. In the hybrid analysis method, a data set was obtained by combining the features obtained by the static analysis method and the API calls obtained by the hooking method in the dynamic analysis method. When using the hooking method, API calls that are effective in Android malware detection are hooked. Again, 375.000 features were obtained by using 35.142 Android applications with a high numerical value and these attributes were reduced to 5000 using IPCA. The obtained features were trained and tested with DNN and the results were presented.

3. Materials and Methods

3.1. Android Software Analysis

Analysis methods are analyzed in 3 categories in the literature. These methods are categorized as static analysis, dynamic analysis, hybrid analysis methods (Alshahrani et al., 2018; Bhilvare & Manik, 2015; Idrees et al., 2017).

3.2. Static analysis method

The static analysis method involves analyzing the source code from the APK file of the applications using various reverse engineering techniques. In the static analysis method, applications are analyzed without running on an emulator or any device. APK files in compressed file format can be opened with compression programs such as Winzip, Winrar, and so, the dex, manifest and source files can be accessed. Tools such as apktool, dex2jar are used to analyze the dex file. For the analysis of the manifest file, applications such as AXML2jar, AXMLPrinter2.jar are used. Since the applications are not run while performing static analysis, there is no damage to the device. This is seen as an advantage of the static analysis method. However, if methods such as obfuscation are used in practice to make the codes difficult to understand, success cannot be achieved with the static analysis method, which is seen as the disadvantage of the static analysis method (Bhilvare & Manik, 2015; Türker & Can, 2019; Yerima et al., 2014).

3.3. Dynamic analysis method

Dynamic analysis method is based on the principle of running applications using a real device or a virtual device and following the behavior of the application at runtime. Since the application is analyzed by running the application in the dynamic analysis method, if the application is a harmful application, the device will be affected and damaged when it is run on real devices. For this reason, it is considered more appropriate to run applications in systems called sandbox (virtual device-emulator). Creating virtual, isolated systems is a costly and laborious method. At the same time, the dynamic analysis method takes a long time compared to the static analysis (Bhilvare & Manik, 2015; Kulkarni, 2018; Türker & Can, 2019).

3.4. Hybrid analysis method

The hybrid analysis method is a method in which the static analysis method and the dynamic analysis method are used together. Hybrid analysis method generally consists of two stages. These stages are the first stage where the static attributes are extracted and the second stage where the dynamically extracted attributes are obtained (Tong & Yan, 2017). The aim of the hybrid analysis method is to achieve correct results by avoiding the disadvantages of the static analysis method and dynamic analysis method.

3.5. Incremental Principal Component Analysis

Principal component analysis (PCA) is a statistical method used to reach the basic attributes that can represent the data within the data set. Considering the multivariate distributed data set, While considering a multivariate distributed data set, PCA aims to represent the data set with the least data loss and the least variable. While given a set of multivariate distributed data in the X-Y coordinate system, the PCA firstly finds the maximum variations of the original data sets. These data points are then projected onto a new axis called the U-V coordinate system. The direction of the U and V axis is known as the main components. The main direction in which the data changes is indicated by the U-axis, whose orthogonal direction follows the V-axis. As in Figure 1, if all data points on the V axis are very close to zero, the data set can be represented with only one U variable and variable V can be omitted (Ng, 2017).

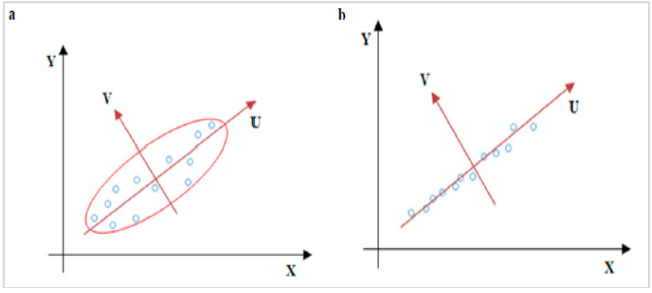

Figure 1. Principal component analysis (Ng, 2017)

Incremental principal component analysis (IPCA) is used instead of PCA if the size of the data set is too large. The large data set may cause insufficient system memory to run the algorithm. IPCA is a method developed to overcome this memory problem. In the IPCA method, which was first presented by Hall et al. and developed with subsequent studies,

instead of taking all training samples, it is based on the principle of dividing the training samples into the specified number and processing all training samples iteratively by processing this number of samples each time. Processing these training samples by dividing them into certain parts reduces the load on the memory and thus aims to prevent the memory deficiency problem (Hall et al., 1998; Ozawa et al., 2006).

3.6. Deep neural network

DNN can be expressed as a neural network with input layer, output layer and multiple hidden layers. The attributes extracted from the problem addressed in each layer of the DNN are learned and the attributes learned in that layer constitute the input values for the next layer. In this way, a network structure is created in which the attributes are learned from the first layer to the final layer (Tokmak & Küçüksille, 2019). A DNN with an input layer, two hidden layers and an output layer is shown in Figure 2.

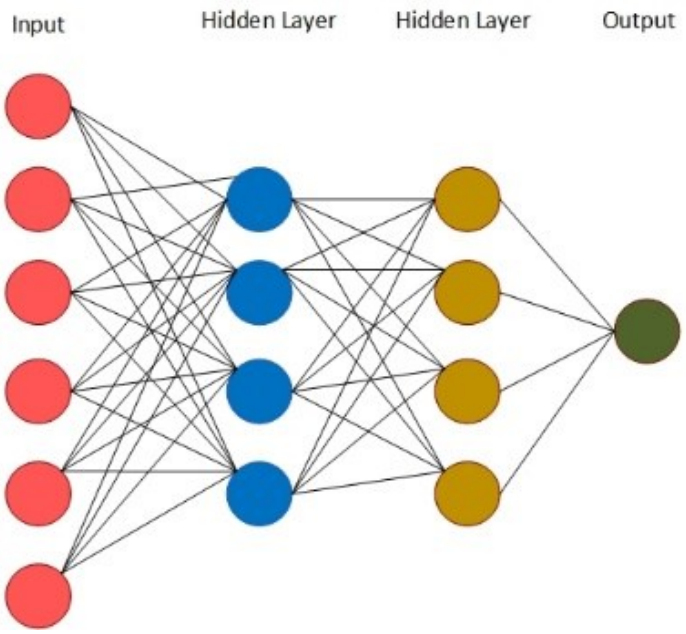

Figure 2. Deep neural network structure

Neurons represented by circles in the DNN structure in Figure 2 have weight value, bias and activation functions. Neurons produce an output value by adding the bias value after the input value and the weights are

multiplied. The activation function is used to control this output value, that is, to decide whether the neuron can be active or not.

DNN is used in many areas such as natural language processing, image processing, speech recognition, time series analysis, malware detection (Barros et al., 2017; Cui et al., 2018; Kolosnjaji et al., 2016; Mezgec et al., 2019; Qiu et al., 2017; Ranjan et al., 2017; Tokmak & Küçüksille, 2019; Zeyer et al., 2017).

4. Research Findings

In this study, static analysis method and hybrid analysis method were applied in the light of the literature in order to detect Android malware. The study conducted in this framework is explained under the headings of static analysis and hybrid analysis.

4.1. Static Analysis Application

For the analysis of Android applications with static analysis method using machine learning techniques, the attributes should be extracted and these extracted attributes should be given as input to the machine learning method. For this purpose, an application was developed in Java programming language during the creation of the attribute vector, and an application in Python programming language was developed at the stage of dimension reduction and DNN method for the created attribute vector.

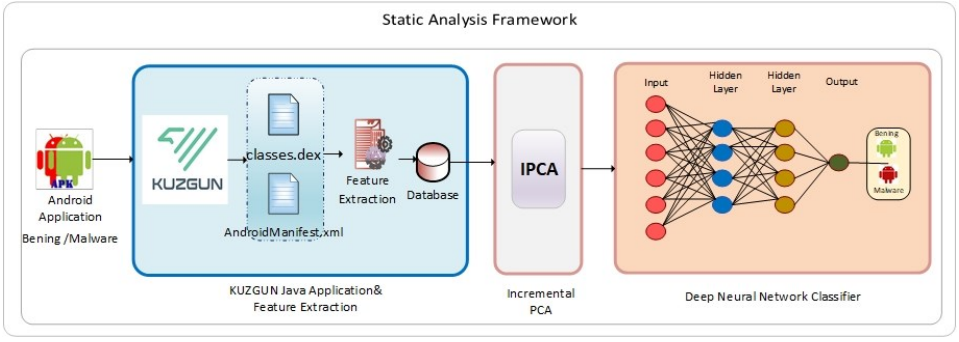

Figure 3. Static Analysis Framework

A console application named Kuzgun has been developed in Java programming language for the purpose of analyzing Android applications. An open source apkfile library was developed and used to parse the APK application files (Fenton, 2018). With this application, the attributes required for analysis were extracted from Manifest.xml, classes.dex files of all APK files in the folder whose folder path is specified, and these attributes are

saved to the MySQL database. After the attributes obtained from all APK files were saved in the database, the attribute vector was saved in the csv file format by reading from the database. The saved attribute file was subjected to IPCA dimension reduction with the Python module and the results were obtained with the DNN Deep Learning method. Static analysis framework is shown in Figure 3.

4.2. Data set

The data set used in the study has a large-scale structure in terms of numerical and diversity. Malicious Android applications in the data set were obtained from AMD (Android Malware Dataset, 2018), while harmless applications were obtained from commercial applications collected by a group of researchers from the Hong Kong University of Science and Technology (AMD, 2018; Android Wake Lock Research, 2018; Liu et al., 2016; Wei et al., 2017).

The AMD dataset contains 24.553 samples, and the disk space is about 60 GB. The AMD data set was collected between 2010 and 2016 and consists of malicious software included in 71 different malware families. 24.503 malware applications were used in the study due to errors such as certificate errors, signature errors and source table errors encountered during the parsing of malicious applications with Kuzgun software.

Non-harmful software was shared by the research group in blocks of 224 sections. The data set consists of 44,736 harmless applications. Due to the problems in the links given during the download and the errors encountered during the parsing of the applications with the developed Kuzgun software, 38.044 of these examples were used in the study, and these applications took up about 350 GB of space on the disk. The data set created includes 62.547 applications in total.

4.3. Static analysis application attributes

Data obtained from AndroidManifest.xml file and classes.dex file were used to detect Android malware while performing analysis with the static analysis method. When the studies were examined, it was revealed that the studies based on Android permissions are intense in detection mechanisms, but the permissions alone are not sufficient, and the need to address features such as API calls, intents, activity, etc. For this purpose, a wide range of features are used in the study. The attributes used are expressed with the Ö tag.

$Ö_1$ Requested permissions: It has an important place in Android security mechanism. Malware can gain access to the device and personal

information through the permissions they obtain and cause various damages. A software with CAMERA permission can access the camera and take photos and videos without the knowledge of the user. A software with SEND_SMS permission can send SMS messages, a software with RECEIVE_SMS permission can receive SMS messages. thanks to these permissions, messages sent and received without the knowledge of the user can cause high bills and unwanted subscriptions.

$Ö_2$ Intents: They provide information exchange and data transfer between Android application components. Starting an activity, switching between activities or starting a service is done through Intents. Typical example of malware-related Intent message BOOT COMPLETED is used to trigger malicious activity immediately after restarting the smartphone.

$Ö_3$ Android application components: They are represented in 4 different categories. These are activities, services, content providers, broadcast receivers. Every application declares these components in their manifest file. The names of these components can also help identify well-known malwares. DroidKungFu2, the second variant of the DroidKungFu malware can be identified by the broadcast receiver name com.eguan.state.Receiver and the service name com.eguan.state.StateService.

$Ö_4$ Restricted API calls: According to the Android Security Architecture, an Android application must define the necessary permissions in the manifest file to access critical API calls such as REBOOT, DELETE_PACKAGE. It is possible that the application is using these API calls unintentionally in the manifest file. The use of restricted API calls without asking for permission is an event that should be followed up in revealing harmful behavior. This may indicate that the malware uses root exploits to overcome the limitations imposed by the Android platform (Arp et al., 2014; Bhandari et al., 2015; Fereidooni et al., 2016).

$Ö_5$ Suspicious API calls: Some of the API calls can access sensitive resources or smartphone information. These types of API calls are often seen in malware samples and can cause malicious behavior. API calls such as getDeviceId() and getSubscriberId() used to access sensitive data, setWifiEnabled() and execHttpRequest() used to communicate over the network, sendTextMessage() for receiving and sending SMS, Cipher.getInstance() used for code scrambling are seen as suspicious (Arp et al., 2014; Seo et al., 2014).

$Ö_6$ Used permissions: Following API calls, they are defined in the attribute vector as both requested and actually used permissions. Analysis of the permissions used is also a determining factor as it determines application

behavior. To determine the permissions used, the study named PScout which maps the permissions required by API calls was used.

Ö$_7$ Unused permissions: In developed applications, it is defined in the manifest file, but this permission is defined as the permissions that are not used by the developed application. In order to make it difficult to understand the malicious behavior during malware analysis, some developers can define extra permissions to make the behavior analysis of the malware difficult. It is important to follow up the unused permissions for this purpose. Unused permissions are extracted using the shared API permission map in PScout.

Ö$_8$ System Commands: A set of commands can be used on the Android platform as in the Linux operating system. Thanks to these system commands, malicious applications can run exploit code, download and install executable files by taking root privileges (Fereidooni et al., 2016; Seo et al., 2014) listed the most frequently used system commands in their study. The commands in this list are based on attribute extraction.

Ö$_9$ Network addresses: A malicious software can regularly establish network connections in order to receive collected commands from the device or to send data out. Therefore, all IP addresses, hostnames, and URLs found in the application's code block are included in the attribute group.

In order to detect malicious software, the features categorized in 9 groups were expressed as vectors during DNN analysis. For this purpose, string values in 9 attributes sets are defined in Ö set. The Ö attributes set consists of approximately 750.000 different attributes. As in Equation 1, Ö can be expressed as a combination of 9 attributes groups.

$$\ddot{O} \coloneqq \ddot{O}_1 \cup \ddot{O}_2 \cup \ldots \cup \ddot{O}_9 \quad (1)$$

|Ö| dimensional vector space in which each dimension is 0 or 1 has been defined by using the Ö set. An application x is mapped to the vector φ (x) so that for each feature extracted from the x application, the corresponding dimension is mapped as 1 and the other dimension is 0. This mapping is shown in Equation 2 for X application.

$$\varphi: X \to \{0,1\}^{|\ddot{O}|}, \varphi(x) \mapsto \left(I(x, \ddot{o})\right)_{\ddot{o} \epsilon \ddot{O}} \quad (2)$$

$\boldsymbol{I(x, \ddot{o})}$ function is simply shown in Equation 3.

$$I(x,ö)=\begin{cases}1 & \text{If application x contains feature ö}\\ 0 & \text{If not}\end{cases} \quad (3)$$

$\boldsymbol{\varphi(x)}$ attribute vector is shown in Equation 4.

$$\varphi(x)\mapsto\begin{pmatrix}\cdots\\ 1\\ 0\\ \cdots\\ 1\\ 0\\ \cdots\end{pmatrix}\begin{matrix}\cdots\\ \text{android.permission.ACCESS\_NETWORK\_STATE}\\ \text{android.permission.READ\_PHONE\_NUMBERS}\\ \cdots\\ \text{getDeviceId}\\ \text{setWifiEnabled}\\ \cdots\end{matrix} \quad (4)$$

4.4. Static analysis incremental PCA

As a result of static analysis, a data matrix of 62.547 rows and 750.000 columns was obtained. Since it would be very difficult to analyze with such a large data set, PCA was tried to be performed at first. However, due to the size (120 GB) of the data set, this operation could not be performed with the existing hardware. For this reason, a machine with 32 Core and 400 GB RAM memory has been rented from Google Cloud servers. Ubuntu operating system was installed on this machine and Python libraries required for analysis were installed. Again, memory problems were encountered in the PCA process in one go and it was decided to perform IPCA. Python sklearn, h5py and numpy libraries were used during the IPCA process. As a result of various experiments, the data could be expressed with 900 features. Explanation rate of these 900 features to the total data was determined as 95.72%.

4.5. Static analysis deep neural network

The 900 features obtained were modeled with an established DNN. The DNN model was created using Python and the sklearn, numpy, keras pandas libraries were used. During the modeling, random 80% of the data is reserved for training and 20% for testing. The established DNN model includes one input layer, 2 hidden layers and 1 output layer. Hidden layers consist of 6 nodes.

As a result of the established model, the accuracy rate of the training set was 0.9974 and the accuracy rate of the test set was 0.9938. The success criteria obtained for the training set of the model are shown in Table 1, and the success criteria obtained for the test set are shown in Table 2.

Table 1. Static analysis training set, performance results

	Precision	**Recall**	**F1-score**	**Support**
Harmless application	0,9992	0,9965	0,9979	30.505
Harmful application	0,9946	0,9988	0,9967	19.532
Macro average	0,9969	0,9977	0,9973	50.037
Weighted average	0,9974	0,9974	0,9974	50.037
ROC AUC				**0,9976**
Accuracy				**0,9974**

Table 2. Static analysis test set, performance results

	Precision	**Recall**	**F1-score**	**Support**
Harmless application	0,9963	0,9935	0,9949	7539
Harmful application	0,9902	0,9944	0,9923	4971
Macro average	0,9932	0,9939	0,9936	12510
Weighted average	0,9939	0,9938	0,9938	12510
ROC AUC				**0,9939**
Accuracy				**0,9938**

4.6. Hybrid Analysis Application

The hybrid analysis method is a method performed as a result of static analysis method and dynamic analysis method of Android applications. Hybrid analysis framework is shown in Figure 4.

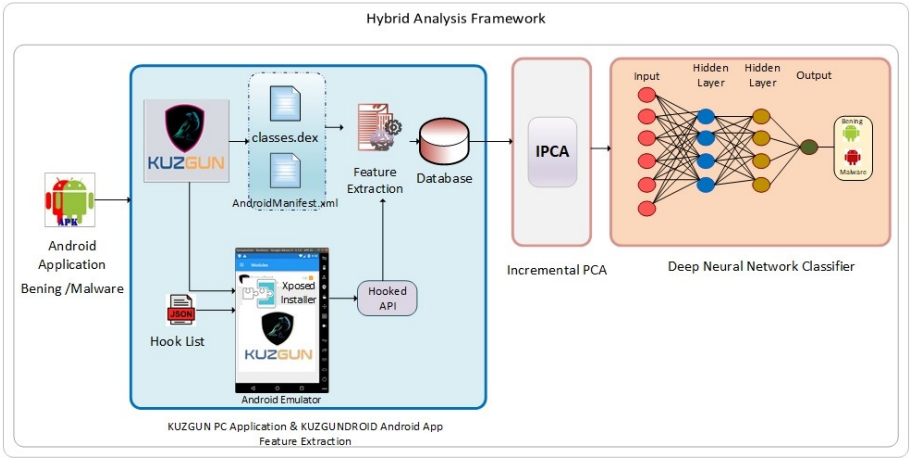

Figure 4. Hybrid Analysis Framework

The dynamic analysis phase of the hybrid analysis method is based on the principle of running Android applications on a real device or a virtual device and obtaining the necessary information and determining the behavior of the application. Considering the damages that malicious applications can cause on the real device, it seems more appropriate to perform the dynamic analysis in a virtual environment isolated from the real environment. For this purpose, the virtual Android emulator developed by Genymotion was used to perform the dynamic analysis part of this study. As a result of the analysis of the applications in the data set, Android 5.1 Lollipop API 22 version was used, aiming to run most of the applications considering the minimum sdk, maximum sdk, target sdk features. The emulator used is shown in Figure 5.

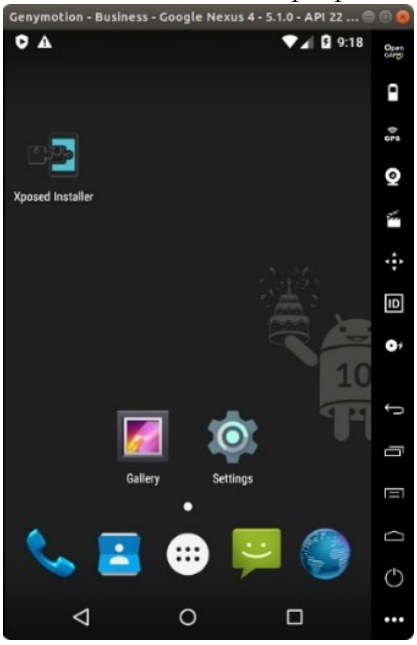

Figure 5. Android emulator

The Xposed application framework, as shown in Figure 6, was used to capture the methods to be hooked while running Android

applications. Xposed Framework 89 version was used with Xposed installer 3.1.5 version.

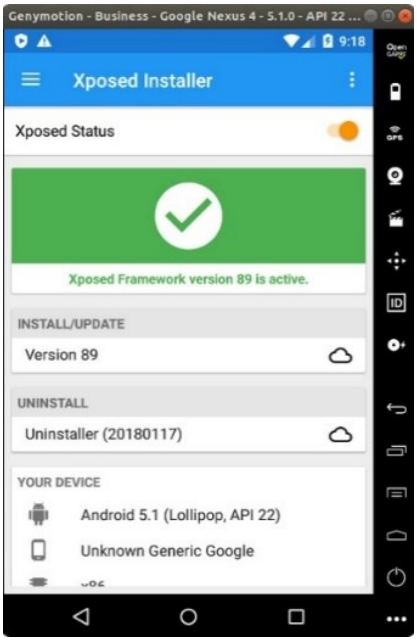

Figure 6. Android emulator and xposed application framework

In order to use the Xposed application framework, the necessary definitions for Xposed should be made in the manifest file while developing the Android application. As in Figure 7, the required definitions in the manifest file in our application are: "xposedmodule" defined as true, "Xposedminversion" defined as 42 and above, "xposeddescription" is labeled as "Dynamic Analysis Tool". When the application is run, Xposed will be integrated into the application framework as a module as in Figure 8 and it will be able to hook the desired methods and functions in the developed application based on these definitions.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/ap
    package="com.dinamikanaliz.kuzgundroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.
    <uses-permission android:name="android.permission.

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_iauncher"
        android:label="KuzgunDroid"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="xposedmodule"
            android:value="true" />
        <meta-data
            android:name="xposedminversion"
            android:value="42+" />
        <meta-data
            android:name="xposeddescription"
            android:value="Dinamik Analiz Aracı" />
    </application>

</manifest>
```

Figure 7. KuzgunDroid manifest file

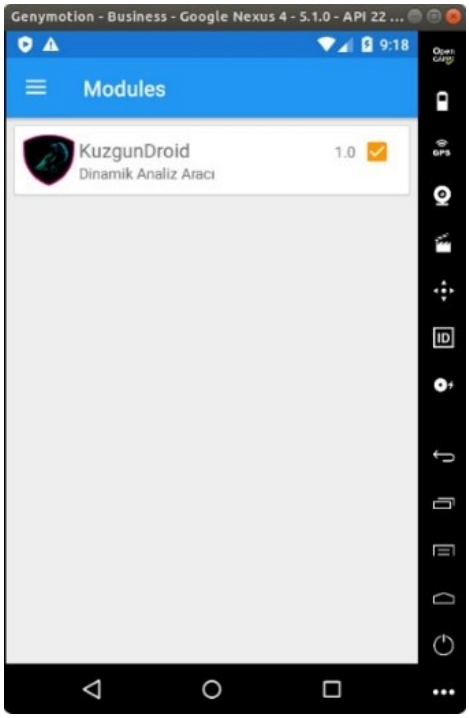

Figure 8. KuzgunDroid module

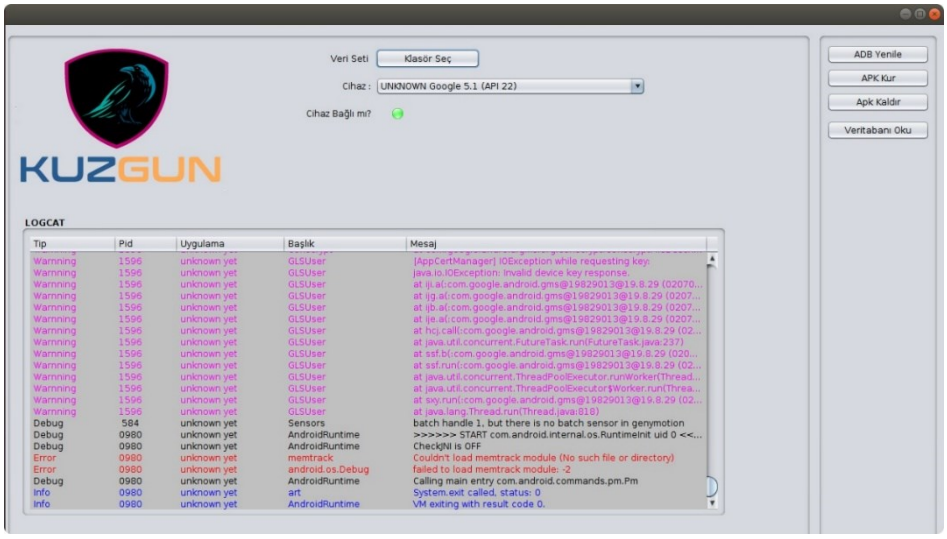

Figure 9. Kuzgun PC application

With the Kuzgun PC application shown in Figure 9, the database is recorded automatically by specifying the path to the folder containing harmful and non-harmful applications by obtaining application permissions and application components statically and obtaining hooked API names dynamically when the application is running. Communication between the emulator and the PC is done with ADB commands. After the application is installed on the emulator with the adb "install" command, it is run with the "am start" command and the application is monitored for 1 minute after the log message is received from the system that the application is running. While the 1 minute running time of the applications on the emulator was determined, tests were made with some applications selected from the data set. As a result of the experiments, it was concluded that 1 minute is sufficient for the hook list used. In this 1 minute, log records are read with the "logcat" command and API calls about the installed application are obtained and recorded in the database. If an error is encountered during installation or during operation, the registration for this application has not been made. These errors are generally caused by changes introduced in Android versions or by definitions made in the application. For example, Android introduced the ART concept after the 4th version and did not guarantee operation in applications written in previous versions. In addition, some applications do not work in the emulator we use because some API calls have been removed or permission levels have been changed. After running for 1 minute without receiving an error, the application was stopped with the "shell am force-stop" command and removed using the "pm uninstall" command and the application information was removed with the "pm clear" command.

4.7. Hybrid analysis data set

The data set used in the static analysis phase was also used in the hybrid analysis phase of the study. While the applications used in the data set are subjected to dynamic analysis, some applications have received an error due to the updates introduced in Android versions. In order to ensure the consistency of the analysis, the applications with errors were removed from the data set. A total of 35.142 Android apps were analyzed without any errors, 14.205 of which were harmful apps and 20.937 non-harmful apps.

4.8. Hybrid analysis attributes

In the hybrid analysis, API calls that are dynamically hooked as a result of running Android applications are used. The list of APIs selected for hooking is given in the appendices of the study. In addition, the statically

obtained permissions and Android components are used as attributes in the dynamic analysis method.

4.9. Hybrid analysis incremental PCA

As a result of the hybrid analysis processes, a data matrix of 35.142 rows and 375.000 columns was obtained. Since it would be very difficult to analyze with such a large data set, PCA was tried to be performed first, but due to the size of the data set (40 GB), this process could not be performed with the existing hardware. For this reason, a machine with 32 Core and 400 GB RAM memory has been rented from Google Cloud servers. Ubuntu operating system was installed on this machine and Python libraries required for analysis were installed. Again, memory problems were encountered in the PCA process in one go and it was decided to perform IPCA. Python sklearn, h5py and numpy libraries were used during the IPCA process. As a result of various experiments, the data could be expressed with 5.000 features. The disclosure rate of these 5.000 attributes to the total data was determined as 92%.

4.10. Hybrid analysis deep neural network

The obtained 5,000 features were modeled with an established DNN. During the modeling, random 80% of the data is reserved for training and 20% for testing. The established DNN model includes one input layer, 2 hidden layers and 1 output layer. Hidden layers consist of 6 nodes.

As a result of the established model, the accuracy rate of the training set was 0.9943 and the accuracy rate of the test set was 0.9694. The success criteria obtained for the training set of the model are shown in Table 3 and the success criteria obtained for the test set are shown in Table 4.

Table 3. Hybrid analysis training set, performance results

	Precision	**Recall**	**F1-score**	**Support**
Harmless application	0,9949	0,9957	0,9953	16.901
Harmful application	0,9936	0,9922	0,9929	11.212
Macro average	0,9942	0,994	0,9941	28.113
Weighted average	0,9943	0,9943	0,9943	28.113
ROC AUC				**0,9939**
Accuracy				**0,9943**

Table 4. Hybrid analysis test set, performance results

	Precision	**Recall**	**F1-score**	**Support**
Harmless application	0,968	0,9822	0,9751	4278
Harmful application	0,9717	0,9495	0,9605	2751
Macro average	0,9699	0,9659	0,9678	7029
Weighted average	0,9694	0,9694	0,9693	7029
ROC AUC				**0,9658**
Accuracy				**0,9694**

5. Discussion and Conclusions

In today's technology world, where Android malware threatens users, taking security measures, detecting and preventing the damages that malicious software can cause emerges as an important field of study. In the study conducted within the framework of this important issue, it is aimed to detect malicious applications targeting the Android operating system.

For the detection of Android malware, static analysis method and the hybrid analysis method were applied; hybrid analysis method is combined usage of static analysis and dynamic analysis methods. When looking at other applications in the literature, a data set that can be called big data in terms of quality and quantity was used. A detailed attribute vector was created on this data set, which is not available in other studies.

In the application of the static analysis method, a wide range of attributes determined in 9 different categories were extracted without running applications by using the software called Kuzgun that developed using Java programming language. Although one or a few of the features extracted in 9 different categories have been used in other studies, this study is the first in the literature in terms of using all of these features together. As given in the data set sections of the study, the number of attribute categories has been examined in a broad perspective for the behavior of the harmful variants, since the obtained Android applications are very diverse and the detection of more malicious applications is aimed. Such a wide feature analysis has brought analysis difficulties with it. That is to say, since the extracted feature category is numerically high, naturally the total number of features obtained was also high. In total, the space occupied by the features in the memory also prevented analysis with the existing hardware. For this purpose, by using the Google Cloud server leasing method, the IPCA

method and dimension reduction method was used on attributes at first. The data set consisting of approximately 750.000 attributes could be represented with 900 attributes by the IPCA method. These 900 attributes reduced by IPCA method had a representation rate of 95.72%. The attributes obtained in the IPCA method formed the inputs of the established DNN model. The established DNN model includes the input layer consisting of 900 nodes, the first hidden layer consisting of 6 nodes, the second hidden layer consisting of 6 nodes and 1 output layer. With the training carried out in the established model, 99.74% accuracy rate, 99.73% F1 score, 99.69% certainty, 99.77% sensitivity values were obtained in the training set. In the test data set, 99.38% accuracy rate, 99.36% F1 score, 99.32% precision, 99.39% sensitivity values were obtained.

In the hybrid analysis part of the study, unlike the static analysis, the behavior of the applications was taken as a basis by running applications and the API calls hooked during execution were added to the features obtained by static analysis. An Android application called KuzgunDroid was developed in order to implement the method called dynamic analysis based on the principle of running Android applications on a virtual device or a real device, and strategically important API calls were obtained by hooking method. At the same time, a Java application named Kuzgun has been developed that communicates with the Android device, sends applications to the emulator, runs, follows the results and removes the application installed on the emulator. The Kuzgun PC application has been developed with the ability to record these attributes in the Mysql database by combining the API calls obtained from the Android emulator with the statically obtained attributes, as well as the ability to read these attributes from database and then record with required format for the DNN model. Obtaining API calls involves a difficult and laborious process as stated in the studies in the literature. Because after each Android application is installed on the emulator, it must be run for a certain period of time so that the behavior of the application can be followed. As in this study, in a study with a large number of data, this situation requires large self-data in terms of both time and the study platform. Due to the disadvantage of existing Android virtual devices, Genymotion Android device was taken as a license and used in the study. In this way, convenience was provided during the compilation and testing of the Kuzgun applications we developed. Another disadvantage encountered while performing dynamic analysis is the total time spent to perform the analysis. According to the literature, each application runs for 1 minute on the emulator. Considering the number of data sets, the time spent for analysis is approximately 25 days. Considering these difficulties, other

studies usually limit the data set in numerical terms. However, in this study, these difficulties were ignored in order to detect more harmful application families without considering time and other difficulty constraints. As a result of the hybrid analysis method, 375.000 feature vectors were obtained from 35.142 applications. As with static analysis, our existing hardware was insufficient to analyze this data. For this purpose, by using the Google Cloud server leasing method, the IPCA method and dimension reduction method was used primarily on the attributes. The data set consisting of approximately 375.000 features was represented with 5.000 features by the IPCA method. These 5.000 features reduced with the IPCA method had a 92% representation rate. The attributes obtained in the IPCA method formed the inputs of the established DNN model. The established DNN model includes the input layer consisting of 5.000 nodes, the first hidden layer consisting of 6 nodes, the second hidden layer consisting of 6 nodes and 1 output layer. With the training performed in the established model, 99.43% accuracy rate, 99.41% F1 score, 99.42% precision and 99.4% sensitivity values were obtained in the training set. In the test data set, 96.94% accuracy, 96.78% F1 score, 96.99% precision, 96.59% sensitivity values were obtained.

Table 5. Results of some DL studies in the literature (Alzaylaee et al., 2020)

Studies	Analysis Method	Number of No-Harm Apps	Number of Harmful Apps	Accuracy	Precision	Sensitivity	F1- score
DroidDetector Yuan et al. (2016)	Static	880	880	89,03	90,39	89,04	89,76
DroidDetector Yuan et al. (2016)	Dynamic	880	880	71,25	72,59	71,25	71,92
DroidDetector Yuan et al. (2016)	Static & Dynamic	880	880	96,76	96,78	96,76	96,76
CNN McLaughlin et al. (2017)	Static	863	1.260	98	99	95	97
MalDozer Karbab et al. (2018)	Static	37.627	20.089	-	96,29	96,29	96,29
Deep4MalDroid Hou et al. (2016)	Dynamic	1.500	1.500	93,68	93,96	93,36	93,68
AutoDroid Hou	Static	2.500	2.500	96,66	96,55	96,76	96,66

et al. (2017) Ddefender Alshahrani et al. (2018)	Static & Dynamic	2.104	2.104	95,13	-	-	95,45
DL-Droid Alzaylaee et al. (2020)	Dynamic	19.620	11.505	94,95	94,08	97,78	95,89
DL-Droid Alzaylaee et al. (2020)	Static & Dynamic	19.620	11.505	95,42	95,31	97,19	96,24

Table 6. Results obtained with Kuzgun

Studies	Analysis Method	Number of No-Harm Apps	Number of Harmful Apps	Accuracy	Precision	Sensitivity	F1- score
KuzgunDroid	Static & Dynamic	20.937	14.205	96,94	96,99	96,59	96,93
Kuzgun	Static	38.044	24.503	99,38	99,32	99,39	99,36

Acknowledgement

This study is derived from the Dr.Mahmut TOKMAK's PhD thesis called 'Deep Learning Based Malware Detection Tool Development for Android Operating System'.

References

Aafer, Y., Du, W., & Yin, H. (2013). DroidAPIminer: Mining api-level features for robust malware detection in android. In T. Zia, A. Zomaya, V. Varadharajan & M. Mao (Eds.), *International conference on security and privacy in communication systems* (pp. 86-103). Springer. https://www.cs.ucr.edu/~heng/pubs/droidapiminer-securecomm13.pdf

Alshahrani, H., Mansourt, H., Thorn, S., Alshehri, A., Alzahrani, A., & Fu, H. (2018). Ddefender: Android Application Threat Detection Using Static and Dynamic Analysis. In *2018 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, USA*, (pp. 1-6).

https://drive.google.com/file/d/1CQfDO7VwqwYzkrYrWPdsbyST8pWqwalE/view

Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2016). Dynalog: An Automated Dynamic Analysis Framework for Characterizing Android Applications. In *2016 International Conference on Cyber Security and Protection Of Digital Services (Cyber Security), London, United Kingdom* (pp. 1-8). https://arxiv.org/ftp/arxiv/papers/1607/1607.08166.pdf

Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2020). DL-Droid: Deep Learning Based Android Malware Detection Using Real Devices. *Computers & Security*, *89*, 101663. https://doi.org/10.1016/j.cose.2019.101663

AMD. (2018). *Android Malware Dataset*. Retrieved May 10, 2018 from http://amd.arguslab.org/

Anagnostopoulos, M., Kambourakis, G., & Gritzalis, S. (2016). New Facets of Mobile Botnet: Architecture and Evaluation. *International Journal of Information Security*, *15*(5), 455-473. http://doi.org/10.1007/s10207-015-0310-0

Android Wake Lock Research. (2018). *Obtain the commercial Android apps*. Retrieved May 06, 2018, from http://sccpu2.cse.ust.hk/elite/downloadApks.html

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium*, *14* (pp. 23-26). The Internet Society. http://user.cs.uni-goettingen.de/~krieck/docs/2014-ndss.pdf

Au, K. W. Y., Zhou, Y. F., Huang, Z., & Lie, D. (2012). Pscout: Analyzing the Android Permission Specification. In T. Yu (Ed.), *Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh North Carolina, USA* (pp. 217-228). Association for Computing Machinery. http://dx.doi.org/10.1145/2382196.2382222

Barros, P., Parisi, G. I., Weber, C., & Wermter, S. (2017). Emotion-Modulated Attention Improves Expression Recognition: A Deep Learning Model. *Neurocomputing*, *253*, 104-114. http://dx.doi.org/10.1016/j.neucom.2017.01.096

Bhandari, S., Gupta, R., Laxmi, V., Gaur, M. S., Zemmari, A., & Anikeev, M. (2015). DRACO: DRoid analyst combo an android malware analysis framework. In O. Makarevich (Ed.), *Proceedings of the 8th International Conference on Security of Information and Networks* (pp. 283-289). Association for Computing Machinery. https://doi.org/10.1145/2799979.2800003

Bhilvare, A., & Manik, T. (2015). An Overview of Different Malware Analysis Techniques in Android. *IJSRD - International Journal for Scientific Research & Development*, *3*(1), 368-372. http://www.ijsrd.com/articles/IJSRDV3I1264.pdf

Cordonsky, I., Rosenberg, I., Sicard, G., & David, E. O. (2018). DeepOrigin: End-to-end deep learning for detection of new malware families. In *2018 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-7). IEEE. https://elidavid.com/pubs/deeporigin.pdf

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., & Chen, J. (2018). Detection of Malicious Code Variants Based on Deep Learning. *IEEE Transactions on Industrial Informatics*, *14*(7), 3187-3196. https://doi.org/10.1109/TII.2018.2822680

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). Androdialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Computers & Security*, *65*, 121-134. https://doi.org/10.1016/j.cose.2016.11.007

Fenton, C. (2018). *GitHub*. Retrieved July 6, 2018 from https://github.com/CalebFenton/apkfile

Fereidooni, H., Conti, M., Yao, D., & Sperduti, A. (2016,). ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)* (pp. 1-5). IEEE. https://doi.org/10.1109/NTMS.2016.7792435

Google Play. (2018). Retrieved December 11, 2018 from https://www.android.com/play-protect/

Hall, P. M., Marshall, A. D., & Martin, R. R. (1998). Incremental Eigenanalysis for Classification. In J. N. Carter & M. S. Nixon (Eds.), *Proceedings of British machine vision conference* (pp. 286-295). BMVC. http://www.bmva.org/bmvc/1998/pdf/p186.pdf

Hou, S., Saas, A., Chen, L., & Ye, Y. (2016). Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)* (pp. 104-111). IEEE. https://doi.ieeecomputersociety.org/10.1109/WIW.2016.040

Hou, S., Saas, A., Chen, L., Ye, Y., & Bourlai, T. (2017). Deep neural networks for automatic android malware detection. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017* (pp. 803-810). https://doi.org/10.1145/3110025.3116211

IDC. (2020). *International Data Corporation*. Retrieved December 2, 2020 from https://www.idc.com/promo/smartphone-market-share

Idrees, F., Rajarajan, M., Conti, M., Chen, T. M., & Rahulamathavan, Y. (2017). PIndroid: A novel Android Malware Detection System Using Ensemble Learning Methods. *Computers & Security*, *68*, 36-46. https://doi.org/10.1016/j.cose.2017.03.011

Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, *24*, 48-59. https://doi.org/10.1016/j.diin.2018.01.007

Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016). Deep learning for classification of malware system call sequences. In B. H. Kang & Q. Bai (Eds.), *Australasian joint conference on artificial intelligence* (pp. 137-149). Springer. http://cys.ewi.tudelft.nl/~zarras/files/AI_2016_Deep.pdf

Kulkarni, K. (2018). *Android Malware Detection through Permission and App Component Analysis using Machine Learning Algorithms* [Master Thesis, University of Toledo]. OhioLINK. http://rave.ohiolink.edu/etdc/view?acc_num=toledo1525454213460236

Liu, Y., Xu, C., Cheung, S. C., & Terragni, V. (2016, November). Understanding and detecting wake lock misuses for android applications. In T. Zimmerman (Ed.), *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 396-409). Association for Computing Machinery. http://cse.sustech.edu.cn/faculty/~liuyp/files/FSE2016.pdf

McAfee. (2018). Mobile Threat Report. Retrieved December 3, 2019 from https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., ... & Joon Ahn, G. (2017, March). Deep android malware detection. In G.-J. Ahn (Ed.), *Proceedings of the seventh ACM on conference on data and application security and privacy* (pp. 301-308). Association for Computing Machinery . https://dora.dmu.ac.uk/bitstream/handle/2086/16947/Deep-Android-Malware-Detection.pdf?sequence=1&isAllowed=y

Mezgec, S., Eftimov, T., Bucher, T., & Seljak, B. K. (2019). Mixed Deep Learning and Natural Language Processing Method for Fake-Food Image Recognition and Standardization to Help Automated Dietary Assessment. *Public health nutrition*, *22*(7), 1193-1202. https://doi.org/10.1017/s1368980018000708

Milosevic, N., Dehghantanha, A., & Choo, K.-K. R. (2017). Machine Learning Aided Android Malware Classification. *Computers & Electrical Engineering*, *61*, 266-274. https://doi.org/10.1016/j.compeleceng.2017.02.013

Ng, S. (2017). Principal Component Analysis to Reduce Dimension on Digital Image. *Procedia computer science*, *111*, 113-119. https://doi.org/10.1016/j.procs.2017.06.017

Ozawa, S., Pang, S., & Kasabov, N. (2006). An incremental principal component analysis for chunk data. In 2006 IEEE International Conference on Fuzzy Systems (pp. 2278-2285). IEEE. http://dx.doi.org/10.1109%2FFUZZY.2006.1682016

Qiu, X., Ren, Y., Suganthan, P. N., & Amaratunga, G. A. (2017). Empirical Mode Decomposition Based Ensemble Deep Learning for Load Demand Time Series Forecasting. *Applied Soft Computing*, *54*, 246-255. https://doi.org/10.1016/j.asoc.2017.01.015

Ranjan, R., Patel, V. M., & Chellappa, R. (2017). Hyperface: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(1), 121-135. IEEE. https://arxiv.org/pdf/1603.01249.pdf

Rosmansyah, Y., & Dabarsyah, B. (2015). Malware detection on android smartphones using API class and machine learning. In *2015 International Conference on Electrical Engineering and Informatics (ICEEI)* (pp. 294-297). IEEE. http://dx.doi.org/10.1109%2FICEEI.2015.7352513

Seo, S.-H., Gupta, A., Sallam, A. M., Bertino, E., & Yim, K. (2014). Detecting Mobile Malware Threats to Homeland Security Through Static Analysis. *Journal of Network and Computer Applications*, *38*, 43-53. http://doi.org/10.1016/j.jnca.2013.05.008

Sugunan, K., Kumar, T. G., & Dhanya, K. (2018). Static and Dynamic Analysis for Android Malware Detection. In E. Blessing Rajsingh, J. Veerasamy, A. H. Alavi & J. Dinesh Peter (Eds.), *Advances in Big Data and Cloud Computing* (pp. 147-155). Springer.

Tokmak, M., & Küçüksille, E. U. (2019). Detection of Windows Executable Malware Files with Deep Learning. *Bilge International Journal of Science and Technology Research*, *3*(1), 67-76. http://dx.doi.org/10.30516/bilgesci.531801

Tong, F., & Yan, Z. (2017). A Hybrid Approach of Mobile Malware Detection in Android. *Journal of Parallel and Distributed computing*, *103*, 22-31. https://doi.org/10.1016/j.jpdc.2016.10.012

Türker, S., & Can, A. B. (2019). Andmfc: Android malware family classification framework. In *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)* (pp. 1-6). IEEE. http://doi.org/10.1109/pimrcw.2019.8880840

Wei, F., Li, Y., Roy, S., Ou, X., & Zhou, W. (2017). Deep ground truth analysis of current android malware. In M. Polychronakis & M. Meier (Eds.), *14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 252-276). Springer, Cham. http://www.arguslab.org/documents/tech_reports/2017/amd_fgwei_2017.pdf

Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012). Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia Joint Conference on Information Security* (pp. 62-69). IEEE. https://doi.org/10.1109/AsiaJCIS.2012.18

Yang, Y., Wei, Z., Xu, Y., He, H., & Wang, W. (2018). Droidward: an Effective Dynamic Analysis Method for Vetting Android Applications. *Cluster Computing*, *21*(1), 265-275. https://doi.org/10.1007/s10586-016-0703-5

Yerima, S. Y., Sezer, S., & McWilliams, G. (2014). Analysis of Bayesian Classification-Based Approaches for Android Malware Detection. *IET Information Security*, *8*(1), 25-36. https://arxiv.org/ftp/arxiv/papers/1608/1608.05812.pdf

Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: Android Malware Characterization and Detection Using Deep Learning. *Tsinghua Science and Technology*, *21*(1), 114-123. https://doi.org/10.1109/TST.2016.7399288

Zeyer, A., Doetsch, P., Voigtlaender, P., Schlüter, R., & Ney, H. (2017). A comprehensive study of deep bidirectional LSTM RNNs for acoustic modeling in speech recognition. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 2462-2466). IEEE. https://www-i6.informatik.rwth-aachen.de/publications/download/1030/Zeyer-ICASSP-2017.pdf