Android Malware Detection through Machine Learning Techniques: A Review

https://doi.org/10.3991/ijoe.v16i02.11549

Oluwakemi Christiana Abikoye, Benjamin Aruwa Gyunka
University of Ilorin, Ilorin, Nigeria

Oluwatobi Noah Akande (✉)
Landmark University, Kwara State, Nigeria
olu_noah010@yahoo.com / akande.noah@lmu.edu.ng

Abstract—The open source nature of Android Operating System has attracted wider adoption of the system by multiple types of developers. This phenomenon has further fostered an exponential proliferation of devices running the Android OS into different sectors of the economy. Although this development has brought about great technological advancements and ease of doing businesses (e-commerce) and social interactions, they have however become strong mediums for the uncontrolled rising cyberattacks and espionage against business infrastructures and the individual users of these mobile devices. Different cyberattacks techniques exist but attacks through malicious applications have taken the lead aside other attack methods like social engineering. Android malware have evolved in sophistications and intelligence that they have become highly resistant to existing detection systems especially those that are signature-based. Machine learning techniques have risen to become a more competent choice for combating the kind of sophistications and novelty deployed by emerging Android malwares. The models created via machine learning methods work by first learning the existing patterns of malware behaviour and then use this knowledge to separate or identify any such similar behaviour from unknown attacks. This paper provided a comprehensive review of machine learning techniques and their applications in Android malware detection as found in contemporary literature.

Keywords—Machine Learning, Ensemble Learning, Android Malware, Android Malware Detection, Base Classifier, Static Analysis, Dynamic Analysis

1 Introduction

Research has shown that Android malware analysis can be done in three different ways: The first method involves the deployment of static [1] and dynamic [2]. Investigation of code of application in order to spot components that are malicious before loading the application into any device; The second method involve modification of the Android system in order to put in modules for monitoring and interception of abnormal behaviours that may occur on the device [3,4,5] while the third approach

involve engaging virtualization to implement the separation of domains ranging from lightweight isolation of an application on the device to running multiple instances of Android OS on the same device [6,7].

However, recent study has shown that machine learning or "anomaly detection" approaches have now emerged to become a leading and more effective approach for defeating Android malware [8, 9, 10, 11]. Unlike the static analysis techniques that involves the manual examination of the AndroidManifest.xml file, source files and the Dalvik byte code, and the Dynamic analysis that involves running an application in a controlled environment to study its behaviour, the Machine Learning approach involves learning the general rules and patterns from benign and malicious app samples and then allowing data-driven predictions of decisions, such as classification [12]. Machine learning methodologies largely depends on static attributes extracted from an application [13]. The static components of an Android application provide the baseline upon which machine learning approaches are anchored and these static features are carefully gotten through the process of reverse engineering.

Machine learning techniques have been applied widely for the classification of applications, focusing mainly on generic malware detection. The application of machine learning in Android malware detection helps eliminate the difficulty involved with manually crafting and updating detection patterns [8]. Machine Learning is a procedure that analyzes data using software techniques (algorithms) to create a model, as shown in Fig. 1, which is useful for finding patterns and regularities in datasets [14]. It is a process of making machines learn from past experiences (existing data) in order to make decisions on future occurring events or data instances. Feature vectors are very essential elements of Machine Learning and they are usually built for the specific task the Machine Algorithm intent to accomplish. The basic idea behind Machine Learning is to get the probability distribution of data.

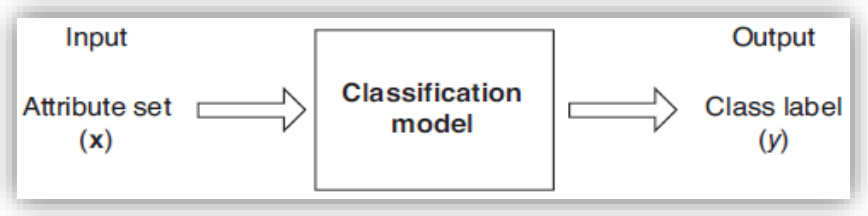

Fig. 1. Classification process in Machine Learning [15].

Machine Learning is divided into three main categories and they are Supervised Machine Learning [16, 17, 18] and unsupervised machine learning [18] and Re-enforcement Machine Learning [19]. Furthermore, there are three basic Learning Methods associated with each Learning Category; Classifications, Clustering, and Regression. Classification is the process used in Supervised Learning in which the data sets are well labelled into groups or classes; Clustering is the process used in unsupervised learning for unlabelled data sets; and Regression is best associated with Re-enforcement learning in which the expected end result is being ranked, graded or

estimated. A label is the name of the definite class or group the data instances belongs to. In machine learning, data are represented by a fixed number of features which can either be categorical, nominal, or continuous [20]. This paper gives a thorough review of different existing literatures in the field of Android malware detections using machine learning techniques.

2 Android Application Anatomy

All Android applications are created and compiled as Android Package (APK) file [21, 22]. The APK file is simply a ZIP format archive file that is renamed to have apk extension [23]. The content consists of a Dalvik executables, resources, native libraries and a manifest file; and is usually signed by the developer of the application using self-signed certificate [21, 22]. In particular, the APK file would usually contain two folders (META-INF and res) and three files (Classes.dex, AndroidManifest.xml, Resources.arsc) [21]. The classes.dex and AndroidManifest.xml are the most delicate and important components of the APK file which are usually the high targets of malware creators [24, 25].

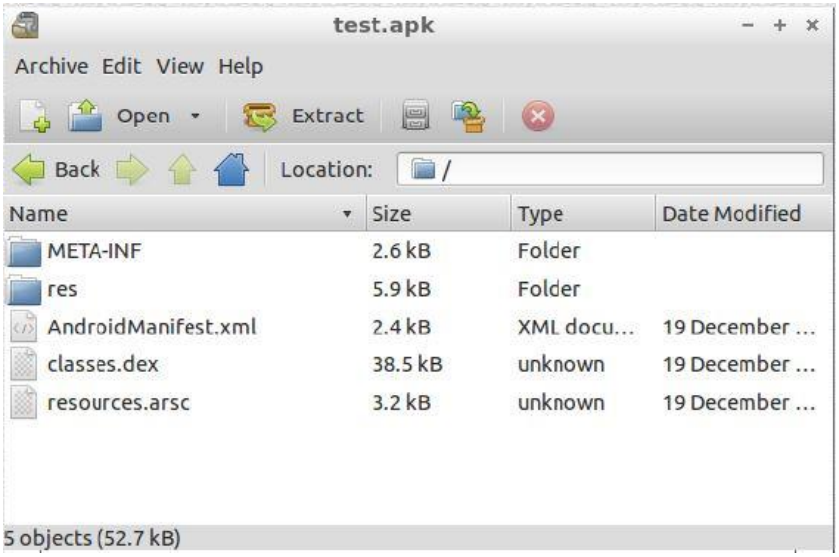

Fig. 2. An APK unzipped file displaying its contents [23].

The classes.dex is the dalvik Virtual Machine (VM) executable file which contains the main working code of the application. That is, the payloads of an application are created and defined in classes.dex file. The application code is compiled and stored in dex format. The AndroidManifest.xml provides a semantic-rich information about the application which includes the version and required permissions governing an app's operations. Attackers target these two files to either inject malicious code into classes.dex [25] or alter the permissions in AndroidManifest.xml for different nefarious

purposes. Therefore, for every Android application, the manifest file is of great importance for the purpose of malware analysis because it defines a list of the various application components which includes requested permissions, services, broadcast receivers, activities, package name, and SDK version [26]. Without this basic detailed information found in the manifest, an application cannot be installed or executed.

As noted earlier, the open nature of Android OS has attracted the interest of lots of developers but has also opened a very wide door for introduction and propagation of malicious applications. Google Play Store, Package Installer and adb Install are the three major mediators or mediums through which application packages can be installed in an Android device [22]. Installations via Google Play Store and Package Install are able to be monitored and scrutinized through the Permission model, but the adb install provides a more dangerous and quiet way of installing apps via USB which cannot be controlled since authentications and permissions are not deployed at this medium. [22] continued by noting that apps which are signed with the same certificate are able to share data between each other and may have the same UID and can even run the same process. This is important to note especially in knowing the origin of a set of malware or applications. Certificate is used for assurance that the code of the original application and its updates come from the same place, and to establish trust relationships between applications of the same developer. Although every application must be digitally signed before it is released and installed on a device, but the critical questions is, can the developer of an app and the digital signature be trusted by the end users and can the apps be trusted to be resistant to tempering once released or installed?

3 Android Malware Attack Trends

Authors in [27, 28] showed in their works that malware attack methods can be characterized as follows:

- **Information Extraction:** The malware in this category compromises a device and then steals personal information such as IMEI number, user's personal information and many more.
- **Automatic Calls and SMS:** This group of malware increases a user's phone bill by placing automatic calls and sending SMS to some premium numbers.
- **Root Exploits:** These set of malware seek to gain system root privileges in order to take control of the system and modify the system's configuration and other system information.
- **Search Engine Optimizations:** The malware here artificially searches for a term and simulates clicks on targeted websites in order to increase the revenue of a search engine or increase the traffic on a website.
- **Dynamically Downloaded Code:** This technique enables an installed benign application to download a malicious code and deploys it in the mobile devices without the user being aware.
- **Covert and Overt Communication Channels:** This is a vulnerability that is found in a device that facilitates the information leak between the processes that

are not supposed to share the information. This technique is seen as a highly sophisticated.

- **Botnets:** This is a network of compromised mobile devices with a Bot-Master which is controlled by a Command and Control servers (C&C). It carries out Spam delivery, DDoS (Distributed Denial of Service) attacks on the host devices.

Malware Authors use many techniques to evade detection. [29] pointed out these concealment techniques to include code obfuscation techniques, encryptions, unnecessary permissions which are not needed by the application, requesting for unwanted hardware, and download or update attacks in which a benign application updates itself or another application with malicious payloads.

4 Existing Literature

Authors in [30] proposed a novel classifier fusion approach called DroidFusion that is based on a multilevel architecture which enables effective machine learning algorithm combination in order to produce an improved accuracy. DroidFusion works by training the base classifiers at a lower level in order to create a model and then a set of ranking-based algorithms are applied on their predictive accuracies at the higher level so as to generate schemes for combination in which one was chosen to build a final classification model. Stratified 10-fold cross validation technique was applied on the training set that was used in training the base classifiers at the lower level in order to estimate their relative prediction accuracies. The authors utilized five base classifiers: J48, REPTree, Random Tree-100, Random Tree-9, and Voted Perceptron. DroidFusion was shown to have outperformed all the base classifiers on all the data sets provided and it also was shown to outperform Stacked Generalization.

A framework for the detection of Android malicious application that was based on Support Vector Machine (SVM) and Active Learning technologies was proposed in [31]. Dynamic data (feature) extraction was employed by the Authors and timestamps was attached to some of the features in order to use a novel time-dependent behavior tracking to significantly enhance the malware detection accuracy. In order to build an active learning model, the authors made use of expected error reduction query strategy so as to combine Android malware new informative instances and to retrain the model in order to be able to do adaptive online learning. To evaluate their model, the authors utilized the DREBIN benchmark malware dataset via a set of experiments and their findings revealed that their framework could detect new malware more accurately.

Authors in [10] conducted a survey with focus on showing the application and the use of ML methods in the analysis of malware. The Authors observed that machine learning is one of the most common techniques adopted in literatures for the analysis of complex malware. The study grouped learning algorithms into four main classes: signature-based (the matching of malicious signatures, the matching of malicious graph), classification (Bayes classifier, prototype-based classification, rule-based classifier, support vector machine, k-Nearest neighbors, decision tree, and artificial neural network etc.), clustering (k-Means clustering, hierarchical clustering, clustering

with locality sensitive hashing, density-based spatial clustering of applications with noise, clustering with distance and similarity metrics, self-organizing maps, prototype-based clustering) and others (learning with local and global consistency, expectation maximization, belief propagation).

Idrees et al. (2017) proposed PIndroid, which was a novel Android malware apps detection framework that uses permissions and intents features for the training of models and further employed classifier fusion technique to combine the classifiers together for an improved performance. The authors used 1745 app samples to conduct the experiments beginning with a comparison of performance between six classifiers: Multi-Lateral Perceptron (MLP), Decision Table, Decision Tree, Random Forest, Naïve Bayesian and Support Vector Machine (SVM) using Sequential Minimal Optimization (SMO). They combined the Decision Table, MLP, and Decision Tree classifiers using three fusion schemes which are Average of Probabilities, Product of Probabilities, and Majority Voting. The framework provided 99.8% True Positive detection accuracy rate, 1.1% False Positive detection rate and an F-measure of 99.7% through the Product of probability combination method.

Dong (2017) worked using permissions as his primary features to develop a novel detection system for Android malware. The collection of both benign and malicious Apps samples was done using a web crawler. To perform the reverse engineering process, the author developed a tool to decompile the Apps to source code and manifest files automatically. One of the findings was that distribution of permissions for Apps share a difference between malware dataset and benign datasets. The Author combined machine learning algorithms such as Logistic Regression Model, Tree Model with Ensemble techniques, Neural Network and finally an ensemble model to find the patterns and more valuable information.

Adebayo (2017) centred his work on building a robust malware detection system by improving apriori algorithm using particle swarm optimization and permission-based features of Android applications to improve the classification system and detection rate of malicious applications. He used the particle swarm optimization (PSO) to generate candidates (flagbearers) from the future set of Android applications for the improvement of Apriori algorithm and Apriori Association Rule (AAR). Afterward, the Author formulated rules from the generated candidates using the AAR. The detection model was developed by using the flagbearers and rules to train seven distinct classification algorithms; Bayesian Classifier, Naive Bayes (NB), PART, Decision Tree (J48), Random Forest (RF), Neural Network (NN), and Classification-based Multiple Association Rule (CMAR). The total application data used was a sample of 1500. The model developed from AAR-PSO performed better, having the best highest True Positive malware detection (TPR), lowest False positive (FPR), highest accuracy, and lowest error rate than any single model developed using the individual classification algorithm.

Authors in [32] proposed two approaches that are based on machine learning for static analysis of Android malware. The first approach depended on Android permission features (such as, manifest analysis) while the second was dependent on the analysis of Android source code using a bag-of-words representation model. The work was aimed at demonstrating the efficacy of making use of machine learning methods

in Android malware static analysis. Classification and clustering are the two machine learning approaches utilized by the Authors but classification method was mostly used for malware detection. Clustering was only used to infer the class of unlabeled data were only very few labels are present in the data set and the labels derived via clustering were further used to retrain the classification model with more data. The authors performed the experiment with ensembles that included odd combinations of three and five classifiers using a fusion method called Majority voting. Four experiments were performed in the study and they are; permission-based clustering, permission-based classification, source code-based clustering, and source code-based classification. A total of 400 applications (200 malicious and 200 benign) were used for the training and testing purposes for the models. Classification algorithms used for model building are Support Vector Machine (SVM), Decision Tree, C4.5, JRIP (Rule-based), Random Forest, Linear regression and Random Tree. Clustering algorithm used are Farthest First, Simple K-means and Expectation maximization (EM). The study showed that the permission-based model was computationally inexpensive compared to the source code analysis model but source code analysis model had an F-Measure of 95.1% against 89% for permission-based model.

An anomaly based malware detection framework for the Android platform was proposed in [33]. They adopted behavioural analysis procedure for both malicious and benign application by practically running them on a physical Android device in order to analyze their behavioral patterns. Machine learning algorithms deployed for malware classification are Decision Tree, K Nearest Neighbor, Logistic Regression, Multilayer Perceptron Neural Network, Naïve Bayes, Random Forest, and Support Vector Machine. Each of the algorithms were assessed using performance metric. Their findings revealed that Support Vector Machine and Random Forest provided the best outcomes for malware detection.

Authors in [34] proposed a composite classification model using a parallel combination of heterogeneous classifiers for Android malware detection which employed static features, extracted from 6,863 app samples, for the algorithms training. The classifiers deployed are Decision Tree (Tree based), Naïve Baye (probabilistic), Simple Logistics (function-base), PART (Rule-based) and RIDOR (Rule-based). Four classifier combination approaches were compared together, that is Average of Probability, Maximum Probability, Product of Probability, and Majority Vote, using the classification algorithms. The composite model was aimed at enabling an enhancing early detection model for Android malware which has improved accuracy and that can provide a quicker white box analysis by means of more interpretable constituent classifiers. The aim of the approach was to leverage the strengths of different kinds of supervised learning algorithms in order to produce a single classification verdict for new application. Static features (Permissions, APIs, and Android framework commands), collected from the Android application via reverse engineering procedure using a bespoke APK analysis tool written in Java, were utilized in their work for malware detection using machine learning. Their result showed that, for individual classifier, PART performed best while products of probabilities combination schemes showed best accuracy and TPR (True Positive Rate).

Similarly, a meta-ensemble technique for Android malware detection was proposed in [12]. They performed an intensive comparative analysis of different classification algorithms precision and then selected the best combination of them based on the ensemble precision obtained. Features selection technique utilized are Chi-squared and Relief for balanced and unbalanced datasets. The data sets were further divided into different new datasets, beginning with the original balanced datasets to different imbalanced datasets. All possible learning algorithms were applied, using their default parameters in Weka for binary attributes and nominal class, on each of the dataset formulated in order to find best performing classifiers and the best dataset, to enable best combination options. Bayesian Log Regression (BLR), Support Vector Machine (SVM) via Sequential Minimal Optimization (SMO), Random Committee (RC), and Random Forest (RF) record best performance with RC and RF performing more exceptionally amongst the four. Identifying RC and RF as the best meta-learners algorithms options, they were further combined together, using Random Forest with 200 trees as based classifier in Random Committee meta-ensemble to form the global composite model.

In a bit to overcome malware that uses obfuscation concealment methodologies to evade most state-of-the-art static based detection and analysis systems, DynaLog which is a dynamic Android malware analysis framework that characterizes Android applications was proposed in [35]. It is a behavioural based analysis system which depends mostly on the number of extensive dynamic features available in an application. It provides an automated environment which is able to massively analyze and characterize applications, thus rapidly identifying and isolating those that are malicious. The system is able to automatically accept huge amount of applications, serially start them on in an emulator, log the different dynamic behaviours and then take them out for further processing. DynaLog is built upon existing open source tools which gives it an advantage for a wide range scope for dynamic analysis of Android apps.

Authors in [13] conducted a research on the application of machine learning algorithm for the detection of Android malware. The research was focused on comparing different static features of applications in order to come out with the right static features combination that can produce the most effective Machine Learning Detection Model. Principally, the research aimed at introducing some feature sets known from the desktop systems (x86 domain), but never previously used in the Android ecosystem, into evaluating their effectiveness in aiding Android malware detections via Machine Learning. The new proposed features include Entropy of files in the resource-folder, Graphical User Interface (GUI) Layout, Number of methods and number of instructions per method. The Authors adopted WEKA Data Mining Software to enable them effectively classifies the different features. For the evaluation of the system, they randomly divided their samples into training set and test set in the proportion of 80 to 20. The test set does not contain any sample from the family of malware used in the training set. The comparative study was run to test both the individual and collective strength of the features. On individual bases, their results revealed that Android permissions are the best single predictor of the app's malignity with an accuracy of about 96%. Others which have accuracy above 90% include opcode frequency, opcode sequences, and app components. Random Forest appeared as the best per-

forming classifier. For the attributes or feature combination test, an accuracy of over 97% was gotten from features extracted from the Android Manifest which includes permissions, features, intents, and application components. Thus, the best performing attribute can be derived rather from an app's metadata stored in the Android Manifest than from the actual code of an app.

Authors in [36] utilized an off-device static and dynamic analysis for the purpose of Android malware detection. Features extracted from manifest files and disassembled codes were used by the Authors. The feature set, that are high-dimensional, includes permissions, file operations, intents, app developer Ids, API calls, components, network statistics, phone events, package serial numbers and lots other features. A linear classifier was used to detect malicious applications and assign a malicious score to the application using a scale of 3 to 4 being benign and being malicious. [28] focused attention on analyzing the common weaknesses found in existing Android malware analysis frameworks. The author noted that in order to create more resilient malware, malware authors are constantly studying the existing detection systems in order to know their detection techniques and principles which will enable them design malware that can evade them. The split-personality malware was the main focus because of their ability to evade most detection systems. The Author noted that most Android malware analysis frameworks are weak against malware characteristics such as code obfuscation which builds a very strong wall against static detections; fingerprinting characteristics renders especially the dynamic detection systems very useless; Application collusion characteristics which utilizes covert communication channels to initiate inter-process or application communications, defying the initial permissions granted an app at installation time. On defeating split-personality malware, the Author suggested the use of analysis frameworks like the BareCloud which adopts the bare-metal analysis approach.

Authors in [37] focused on how to identify and defeat the class of unknown malware known as the "0-day" malware. They developed a framework called SherlockDroid. To detect unknown malware, the SherlockDroid works by filtering masses of applications and only keeps the most likely to be malicious for future inspections. Apart from crawling applications from marketplaces, SherlockDroid extracts code-level features, and then classifies unknown applications using Alligator. Alligator is a classification tool that efficiently and automatically combines several classification algorithms. SherlockDroid considered features of Android applications extracted from a static analysis of the code. A lightweigth technique for Android malware called DREBIN that enables a direct malicious application detection on the Smartphone was proposed by [8]. DREBIN overcomes the limitations of lack of enough resources that hampers applications monitoring at run-time. It works by performing a wide static analysis, and gathering as many features (Permissions, API calls, hardware resources, app components, filtered intents and network addresses) of an application as possible. Joint vector space was used to embed these features such that typical patterns indicative for malware are automatically fiqured out. Support Vector Machine was the machine learning approach adopted for the malware classification. DREBIN was evaluated against other related approaches using 123,453 applications and 5,560 malware samples and the results showed that DREBIN performed better than the others with a

malware detection rate of 94% with very minimal false positive detection rate of 1%. DREBIN was tested on five different smartphones and recorded very high average speed of 10 seconds to run an analysis. Although their techniques made use of more features than applied in this study, they however only have 4.5% (5,560) malware samples out of the total 129,013 apps used, this percentage was too small to enable the system effectively learn malware patterns. Authors in [38] carried out an in-depth investigation of Android malware and their different analysis techniques. The study included the history, evolution, behaviour and the different methodologies available for the analysis of different Android threats. Having studied the activities of different malware, the authors revealed that at inception, malwares were mostly targeted at exploiting roots of devices for nefarious intents, but with the advancement in technology, malcode developers now discovered it is no longer necessary to exploit roots of the device before being able to get what they want. The Authors noted that once installed and executed, Android malware would most times rather exploit the different variance of permissions allowed for the app to perform different malicious activities. The research revealed that Android malware operates mostly as Trojans. The Authors stated that the analysis of Android malware involves looking for additional class files because it has become a common phenomenon to easily modify Android apps and add extra class file, such as a Trojan component, to its code. The research observed that external tools are better suited to combat malware rather than having tools that runs inside the device. The infected device will create an unreliable environment for the analysis tools by influencing their activities and decisions. The work gave a detail explanation on the different analysis methods which includes Static Analysis, Behavioural Analysis, and Dynamic Analysis through the Sandboxing Systems. They stated that strings are an essential part of any static malware analysis, possibly providing clues related to malware construction, functionality, authorship, and Command & Control (C & C). The research revealed that the most important strings of an app are found in classes.dex, the source code of apps, after they are unpacked.

Authors in [34] developed and analyzed proactive Machine Learning approaches based on Bayesian classification aimed at uncovering unknown Android malware via static analysis. They presented and analyzed three Bayesian classification approaches for detecting Android malwares. Permissions and code based properties such as API calls, both Java system based and Android system based, Linux and Android system commands were also extracted from the sampled applications. The three models were built by extracting the different features from a set of 1000 samples of 49 Android malware families together with another 1000 benign applications across a wide variety of categories. They employed a Java-based custom built APK analyzer to automate the reverse engineering of all the APK files. A list of top 20 permissions and top API calls used by benign and malicious applications were presented. The research findings showed that models developed using mixed-based and code property-based features are a better choice than the permissions-only models.

Authors in [39] investigated the effectiveness of antivirus solutions against Android malware. The aim of their study was to come up with results to help both corporate and private users to assess the real risk level imposed by Android malware on the one hand, and the protection level offered by antivirus software on the other hand.

The motivation of the research was anchored on the very high detection rates attested by various antivirus testing security reports in circulation. The authors noted that the general perception of Android security has been largely shaped by two classes of reports: the one given by antivirus vendors and the one given by magazines, companies, and institutes who publish test reports of antivirus products. The authors challenged the different reports on the effectiveness of antivirus support because most of these reports were all based on retrospective antivirus security testing results. To test a real world performance of antivirus against unknown malware signatures, the authors subjected about eleven anti-viruses, which were tagged as the "test candidates", chosen to represent a fair number of well-known companies, to an experiment. The authors only chose free or free-to-test versions of Android antivirus apps; they noted that the paid apps usually do not offer additional detection capabilities. The experimental test was setup to consider the ability to cope with typical malware distribution channels, infection routines, and privilege escalation techniques. The research revealed that it is very easy for malware to evade antivirus detection with only trivial alterations to their package files. For the experiment, the Authors developed a malware tagged as "Proof of Concept Malware" which demonstrates advanced functionality not found in most of the existing known Android malware. This proof of concept malware was totally strange to all anti-viruses as they are not familiar with its signatures.

The possibility of detecting malicious applications using permissions was explored in [40]. In order to retrieve the permissions, the authors disassembled APK packages, identified the invoked Android system functions and then reconstructed the permissions used. To evaluate their detection model, the authors used a dataset of 124,769 benign applications and 480 malicious ones, and 4 machine learning algorithms respectively: AdaBoost, Naive Bayes, Decision Tree and Support Vector Machine. However, in order to help the detection mechanism, the authors used several other features (like the number of particular file formats and both the number of under-privileged and over-privileged permissions) in addition to the permissions. The experimental finding showed that about of applications that are malicious can be detected using a single classifier. They however concluded that a permission-based mechanism can only be used as a quick filter to identify malicious applications but that it still requires a second analysis system or pass, as they called it, to make complete analysis to report malicious application. Similarly, [26] introduced a complete automated and a comprehensive analysis system called ANDRUBIS, which combines both static and dynamic analysis methods and it is publicly available. The Authors considered as less effective the approach adopted by most malware analyst which typically relies on analysis tools to extract characteristic information about an app in an automated fashion for the purpose of malware analysis. The challenge noted about this approach was that, though these analysis tools are very important, the resulting prototypes remain limited in terms of analysis capabilities and availability. Although a significant body of research uses both the static and dynamic analysis methods, none of them provide a comprehensive feature set for a sample. However, the Authors also suggested that post-analysis techniques such as clustering can produce more meaningful results if they are applied to a rich feature set. ANDRUBIS was built as an extension to the

public malware analysis sandbox called Anubis and it is for the analysis of unknown Android applications. However, the primary goal of ANDRUBIS as stated by the Authors is to provide researchers with a comprehensive static and dynamic analysis report of an application, not to automatically identify applications as goodware or malware. The data generated will then be used for malware analysis.

The use of machine learning approach for the detection of Android malware was proposed in [41]. The proposed concept involved the use of one-class Vector Support Machine for the training of only benign Android applications in an off-line manner. The training of the algorithm was done using the SciKit-Learn framework because of its ability to provide a convenient interface to LIBSVM. Features, which were solely based on permission (built-in and non-standard), gotten from the Manifest folder, and Control Flow Graphs (CFGs) which were both extracted using Androguard, which is an effective open source tool. They focused mainly on training the model based on only benign application for the reason that they believe benign Android application are far more readily accessible than the malicious ones. The One-Class Support Vector Machine (SVM) adopted in the research is a linear classifier that is in a high-dimensional feature space which is based on the constrained quadratic optimization problem. The detection is done by the model only when the classifier is able to classify an application (in this case, a training sample) as being sufficiently different from the benign class. Although this is a fast detection approach, but it is one sided, it does not include the malicious application which was added in this study.

Authors in [42], presented andromaly, a host-based behavioural framework for anomaly detection on Android devices. Andromaly continuously monitors various features and events obtained from the mobile device and then passing the data through anomaly detectors that use Machine Learning. The data that was collected can then be classified as either benign or malicious. The framework was implemented using small application which once installed, it samples various pieces of packets sent via cellular or Wi-Fi networks, total number of processes running, battery consumption and so on, and then analyses if the phone is functioning normally, or there are some anomalies in the collected data. The framework utilizes the idea that malware that have not yet been encountered can be detected by analyzing the similarities shown in the fluctuation of above mention system data with the introduction of already known malware. The framework is modular and can utilize various malware detection techniques using rules and algorithms besides its behavioural approach.

Authors in [43] proposed DroidRanger, a permission-based behavioural footprinting scheme to detect new samples of known Android malware families. In DroidRanger, applications are firstly filtered based on the Android permissions requird and, then, heauristcs-based filter is applied. This approach stands a better chance of combating zero-day malware as it is not based on signatures. This is because signature-based approaches are only reactive in nature, they can only work based on known signatures. The system uses an emulator for analysis which becomes a strong weakness that can be bypassed by the emulator-aware malware. This study therefore did not involve virtual environment for any of the components analysis. Similarly, a cloud-based security model, such as Android Application Sandbox (AASandbox) wasproposed in [2]. This system was able to perform both statistical and

dynamical analysis to automatically detect suspicious applications. AASandbox firstly performs a statistical analysis on the APK package in order to detect malicious patterns. Afterwards, the dynamical analysis is performed in a fully isolated environment. During the dynamical analysis, all the events occurring in the device are monitored. Though this system appears to be effective, especially that it mostly engages online scanning engines to scrutinize APKs, it cannot be totally free from error as it is cloud-based. It is also far away from the analyst who may have little or no control over the analysis and would thus accept any result generated by the system. This system too can be evaded by the split-personality malware since it is also virtualized.

5 Conclusion

Android Operating System remains an opened platform with lots of rising mobile device hardware manufacturers adopting it as the main OS for their devices. This has substantiated a constant exponential introduction of different applications developed for the platform. A huge drawback to this system is that most of the applications developed don't have a central control system for integrity check that would certify whether they are fit and secure to be released to the wider market or not. Due to this deficiency, lots of applications that were designed with good motive ends up becoming malicious in their behaviours because of poor designs or unprofessional developers that are either not careful about leaving bugs in the code of the apps nor follow the right security procedures in the development processes. The volumes of these poorly designed applications, coupled with the volumes of those that are deliberately created for malicious purposes, have combined to form a great security challenge for the Android platform. The advent of internet on these hand-held devices has massively created an enabling channel for increased proliferations of these malware. All these security threats targeted at the Android platform will continue to form the basis for which very many different researches will keep on being released with focus on the detection, analysis and remediation of evasive Android malware. Machine learning, which is a branch of Artificial Intelligent, has gained more popularity as the most suitable method for combating zero-day malware attacks. Hence, different novelty approaches for using machine learning methodologies will continue to be investigated and deployed in malware detections.

6 References

[1] Enck, W. (2011). Defending users against smartphone apps: Techniques and future directions. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7093 LNCS, 49–70. https://doi.org/10.1007/978-3-642-25560-1_3

[2] Bläsing, T., Batyuk, L., Schmidt, A. D., Camtepe, S. A., & Albayrak, S. (2010). An android application sandbox system for suspicious software detection. Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010, 55–62. https://doi.org/10.1109/malware.2010.5665792

[3] Backes, M., Gerling, S., Hammer, C., Maffei, M., & Philipp, von S.-R. (2012). AppGuard — Real-time policy en- forcement for third-party applications. Saarbrücken, Germany.
[4] Nauman, M., Khan, S., & Zhang, X. (2010). Apex. Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security - ASIACCS '10, 328. https://doi.org/10.1145/1755688.1755732
[5] Xu, R., Saïdi, H., Anderson, R., & Saıdi, H. (2012). Aurasium: Practical Policy Enforcement for Android Applications. In Proceedings of the 21st USENIX Security Symposium (pp. 539–552).
[6] Andrus, J., Dall, C., Hof, A. V., Laadan, O., & Nieh, J. (2011). Cells. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11, 173. https://doi.org/10.1145/2043556.2043574
[7] Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., & Peter, M. (2011). L4Android: A Generic Operating System Framework for Secure Smartphones. Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, 39–50. https://doi.org/10.1145/2046614.2046623
[8] Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., & Rieck, K. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Symposium on Network and Distributed System Security (NDSS) (pp. 23–26). https://doi.org/10.14722/ndss.2014.23247
[9] Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Roli, F. (2017). Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection, 1–14. https://doi.org/10.1109/tdsc.2017.2700270
[10] Ucci, D., Aniello, L., & Baldoni, R. (2018). Survey on the Usage of Machine Learning Techniques for Malware Analysis. Computers and Security, 1(1), 1–67. https://doi.org/10.1016/j.cose.2018.11.001
[11] Rescuers, V. (2018). How Cybercriminals became 'The New Mafia.' Retrieved January 31, 2018, from http://www.virusrescuers.com/how-cybercriminals-became-the-new-mafia/
[12] Coronado-De-Alba, L. D., Rodriguez-Mota, A., & Ambrosio, P. J. E.-. (2016). Feature selection and ensemble of classifiers for Android malware detection. In 2016 8th IEEE Latin-American Conference on Communications (LATINCOM) (pp. 1–6). https://doi.org/10.1109/latincom.2016.7811605
[13] Hahn, S., Protsenko, M., & Müller, T. (2016). Comparative evaluation of machine learning-based malware detection on Android. In Sicherheit 2016: Sicherheit, Schutz und Zuverl{ä}ssigkeit, Beitr{ä}ge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft f{ü}r Informatik e.V. (GI), 5.-7. April 2016, Bonn (pp. 79–88). https://doi.org/10.1007/978-3-662-01089-1_31
[14] Russell, I., & Markov, Z. (2017). An Introduction to the Weka Data Mining System. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE '17 (pp. 742–742). https://doi.org/10.1145/3017680.3017821
[15] Tan, P.-N., Steinbach, M., & Kumar, V. (2006). Classification : Basic Concepts , Decision Trees , and Model Evaluation Classification. Introduction to Data Mining, 1, 145–205.
[16] Brownlee, J. (2016). Supervised and Unsupervised Machine Learning Algorithms. Retrieved May 13, 2018, from http://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ https://doi.org/10.1007/978-3-030-22475-2_1
[17] Garg, B. (2013). Design and Development of Naive Bayes Classifier (Master Thesis). North Dakota State University of Agriculture and Applied Science.
[18] Namratha, M., & Prajwala, T. R. (2012). A Comprehensive Overview of Clustering Algorithms in Pattern Recognition. Journal of Computer Engineering, 4(6), 23–30.

[19] Scott, G. (2015). ML 101: Reinforcement Learning. Retrieved November 23, 2017, from http://scottge.net/2015/07/02/ml101-reinforcement-learning/
[20] Guyon, I., & Elisseeff, A. (2006). Feature Extraction, Foundations and Applications: An introduction to feature extraction. Studies in Fuzziness and Soft Computing, 207, 1–25. https://doi.org/10.1007/978-3-540-35488-8_1
[21] Rovelli, P. (2014). Developing a Next-Generation Mobile Security Solution for Android (Master Thesis). Reykjavík University.
[22] Zhauniarovich, Y. (2014). Android TM Security (and Not) Internals (ASANI Book) (1.01). Trento: asani.
[23] Fora, P. O. (2014). Beginners Guide to Reverse Engineering Android Apps. In RSA Conference (pp. 21–22). San Francisco: RSA.
[24] Shah, R. (2011). Analyzing and Dissecting Android Applications for Security defects and Vulnerabilities. Retrieved January 6, 2018, from https://www.helpnetsecurity.com/dl/articles/Blueinfy_Rushil_ScanDroid_Paper.pdf
[25] Tchakounté, F., & Dayang, P. (2013). System Calls Analysis of Malwares on Android. International Journal of Science and Technology, 2(9), 669–674.
[26] Weichselbaum, L., Neugschwandtner, M., Lindorfer, M., Fratantonio, Y., Veen, V. Van Der, & Platzer, C. (2012). ANDRUBIS : Android Malware Under The Magnifying Glass. Vienna University of Technology, Technical Report (Vol. TR-ISECLAB). Vienna Univeristy of Technology.
[27] Raveendranath, R., Rajamani, V., Babu, A. J., & Datta, S. K. (2014). Android malware attacks and countermeasures: Current and future directions. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT 2014, 137–143. https://doi.org/10.1109/iccicct.2014.6992944
[28] Richter, L. (2015). Common Weaknesses of Android Malware Analysis Frameworks. In IT Security Conference, University of Erlangen-Nuremberg during summer term 2015 (pp. 1–10). Erlangen.
[29] Baskaran, B., & Ralescu, A. (2016). A Study of Android Malware Detection Techniques and Machine Learning. In Proceedings of the 27th Modern Artificial Intelligence and Cognitive Science Conference 2016, Dayton, OH, USA, April 22-23, 2016. (pp. 15–23).
[30] Yerima, Suleiman Y., Sezer, S., & Muttik, I. (2016). Android Malware Detection Using Parallel Machine Learning Classifiers. 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, (Ngmast), 37–42. https://doi.org/10.1109/ngmast.2014.23
[31] Rashidi, B., Fung, C., & Bertino, E. (2018). Android malicious application detection using support vector machine and active learning. In 2017 13th International Conference on Network and Service Management, CNSM 2017 (Vol. 2018-Janua). https://doi.org/10.23919/cnsm.2017.8256035
[32] Milosevic, N., Dehghantanha, A., & Choo, K. K. R. (2017). Machine learning aided Android malware classification. Computers and Electrical Engineering, 61, 266–274. https://doi.org/10.1016/j.compeleceng.2017.02.013
[33] Ali, M. Al, Svetinovic, D., Aung, Z., & Lukman, S. (2017). Malware Detection in Android Mobile Platform using Machine Learning Algorithms. In International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017) (pp. 4–9). ADET: IEEE. https://doi.org/10.1109/ictus.2017.8286109
[34] Yerima, Suleiman Y., & Sezer, S. (2018). DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection. In IEEE Transactions on Cybernetics (pp. 1–14). IEEE. https://doi.org/10.1109/tcyb.2017.2777960

[35] 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 2016 (pp. 1–8). https://doi.org/10.1109/cybersecpods.2016.7502337

[36] Lindorfer, M., Neugschwandtner, M., & Platzer, C. (2015). MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis. In 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC) (Vol. 2, pp. 422–433). IEEE. https://doi.org/10.1109/compsac.2015.103

[37] Apvrille, L., & Apvrille, A. (2015). Identifying unknown android malware with feature extractions and classification techniques. Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015, 1, 182–189. https://doi.org/10.1109/trustcom.2015.373

[38] Dunham, K., Hartman, S., Morales, J. A., Quintans, M., & Strazzere, T. (2014). Android malware and analysis (1st ed.). New York: Auerbach Publications. https://doi.org/10.1201/b17598

[39] Fedler, R., Schütte, J., & Kulicke, M. (2013). On the Effectiveness of Malware Protection on Android. In Fraunhofer AISEC (p. 36). Berlin.

[40] Huang, C. Y., Tsai, Y. T., & Hsu, C. H. (2013). Performance Evaluation on Permission-Based Detection for Android Malware. In Smart Innovation, Systems and Technologies (Vol. 21, pp. 111–120). Berlin: Springer-Verlag Berlin Heidelberg.

[41] Sahs, J., & Khan, L. (2012). A Machine Learning Approach to Android Malware Detection. In Intelligence and Security Informatics Conference (pp. 141–147). Dallas: IEEE. https://doi.org/10.1109/eisic.2012.34

[42] Shabtai, A; Kanonov, U; Elovici, Y; Glezer, C; Weiss, Y. (2012). "Andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, 38(1), 161–190. https://doi.org/10.1007/s10844-010-0148-x

[43] Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. Proceedings of the Second ACM Conference on Data and Application Security and Privacy - CODASKY '12, 317–326. https://doi.org/10.1145/2133601.2133640

7 Authors

Oluwakemi Christiana Abikoye received a National Diploma (ND) in Computer Science from Kwara State Polytechnic, Ilorin in 1996, a B. Sc. degree in Computer Science from University of Ilorin in 2001, M. Sc degree in Computer Science from University of Ibadan, Ibadan in 2006 and Ph.D. degree in Computer Science in 2013.. She began her academic career at University of Ilorin, Department of Computer Science in 2004 as a Graduate Assistant and rose through the ranks. She is presently a Senior Lecturer. Oluwakemi is known for her innovative work in Computer/Communication Network Security, particularly on issues involving Security in computer, networks and Cash dispenser machines and transaction authentication system. Her research interests include Cryptography, Biometrics, Human-Computer Interaction, and Cybersecurity. She is also involved in the supervision of postgraduate (Masters/Ph.D.) students' research work in specialized areas of Computer (Information Security). She has several publications in Local, national and international journals.

Benjamin Aruwa Gyunka graduated with a Doctor of Philosophy degree (Ph.D.) in Computer Science from the University of Ilorin, Nigeria. He also holds a Bachelor of Science degree in Mathematics from the University of Jos and a Master of Science degree in Information Systems Security from Sheffield Hallam University, United Kingdom. Prior to this time, he had extensive experience as a Network Administrator while working with the National Open University of Nigeria (NOUN). His research interest lies mostly in Information Security, Cybersecurity, data mining, Android security, digital forensics, and machine learning.

Akande Noah Oluwatobi had B. Sc. and M. Sc. degrees in Computer Science from Ladoke Akintola University of Technology. He presently lectures in the Department of Computer Science, Landmark University, Omu-Aran, Nigeria. He is a member of Computer Professional (Registration Council) of Nigeria (MCPN), Member, Nigeria Computer Society (MNCS), and IAENG Society of Computer Science. His research areas include Data and Information Security, and Pattern Recognition (Medical Image Analysis).
Email: akande.noah@lmu.edu.ng / olu_noah010@yahoo.com

Article submitted 2019-08-20. Resubmitted 2019-09-17. Final acceptance 2019-09-21. Final version published as submitted by the authors.