Sustainable Android Malware Detection Scheme using Deep Learning Algorithm

Abdulaziz Alzubaidi
Computer Science Department,
College Computing in Al-Qunfudhah
Umm Al-Qura University

***Abstract*—The immense popularity of smartphones has led to the constant use of these devices for productive and entertainment purposes in daily life. Among the different operating systems, the Android system plays a very important role in the development of mobile technology as it is the most popular operating system. This makes it a target for cyberattack, with severe negative effects in terms of monetary and privacy costs. Thus, this study implements a detection scheme using effective deep learning algorithms (LSTM and MLP). Also, it tests their ability to detect malware by employing private and public datasets, with accuracy of over than 99%.**

***Keywords*—Smartphone security; machine learning; mobile malware; classification; big data**

I. Introduction

The current century has witnessed various inventions such as smartphone devices. These devices are characterized by advanced features in terms of sophisticated operating systems and gigabytes of memory. They are equipped with advanced sensors such as accelerometer, magnetometer, global positioning system (GPS), and biometric sensors. Due to these developed features, the owner of smartphones can perform various activities, such as sending/ receiving electronic emails, performing financial transactions, contacting with others, taking photos and recording videos, etc. [1]. Statistics show the number of smartphone users surpassed 6.4 billion users globally in 2021, and it is expected to reach 7 billion in 2026 [2].

Mobile applications (apps) are implemented to perform one or more tasks. They are available in official markets and third parties. As of 2021, the popular mobile market apps are: Google Play (3,482,452 apps), Apple app store (2,226,823 apps), Windows store (669,000 apps), and Amazon App store (460619 apps) [2]. Smartphones and mobile market apps are targets for attackers in terms of privacy and security. According to [3], the instances of mobile cybercrime surpassed 14.4 million attacks in the second quarter of 2021, 95% of Android devices can be hacked using a simple text message, and 87% are exposed to serious vulnerability. In September of 2020, the Apple store has pulled 40 apps infected by the XcodeGhost botnet attack [4], which indicates that the malware apps might be found in official apps. Therefore, several approaches from academic and industrial fields have been proposed using machine learning algorithms.

Machine learning is an effective method for intelligent detection of malware on smartphones. Malware detection on smartphones is based on feature analysis by static, dynamic, and hybrid methods [2]. The detection and prediction effectiveness of any machine learning algorithm relies on selecting suitable data and understanding malware behavior.

Problems and Motivation

Previous studies have been proposed to detect known and unknown malware samples using public and private datasets. However, most of utilized public datasets are collected between 2010 and 2017, which raise an important question about how they can detect recent implemented malware while the behavior of mobile malware is changeable. Therefore, there is need to collect updated apps . Besides that , understanding the malware patterns and classifying them into families is an effective way to detect unknown malware.

This study proposed a sustainable and cost-effective malware detection scheme with respect to collecting an updated dataset, classifying malware families , and observing malware behavior .

Contributions

This work address the above-mentioned issues related to detecting Android malware. Its contributions are listed below.

1) A dataset is build with 30,000 samples at the present time, plan to expand to be larger and make it publicly available
2) An Android malware detection approach is proposed using machine and deep learning algorithms with respect to sustainability metrics.

The remaining part of this article is organized as follows: Section II summarizes recent studies in this field. Sections III and IV introduce the proposed methodology and describes the dataset used. Finally, the conclusion of this research work is presented in Section V.

II. Prior Research

Detecting Android malware has gained attention last two decades. There are several proposed approaches employed machine and deep learning. Alzubaidi [2] provides a comprehensive survey in terms of static, dynamic and hybrid feature analysis methods using machine learning algorithms, while Qiu et al. [5] review recent deep learning approaches and addressed challenges like the architecture of deep learning. This sections discusses sustainable Android malware detection approaches then summarizes common public datasets.

A. Sustainable Detection Malware Approaches

Onwuzurike et al. [6] introduced a static approach to detect Android malware, which is known as MaMaDroid. This approach is comprised of three phases. The authors first acquired a dataset with size of 43,940 apps (35,493 malware apps from Drebin and 8,447 normal apps from the Google Play store). Second, they extracted the API calls for each feature, then used principal component analysis (PCA) [7] to rank them. Third, they employed random forest (rf) [8], 1-Nearest Neighbor (1-NN), 3-Nearest Neighbor (3-NN) [9], and support vector machine (SVM) [10] to construct their approach. The authors performed two experiments on detecting unknown malware samples and examined the sustainability of their approach. For the first experiment, they achieved accuracy of 0.99. For the second experiment, they examined the sustainability of the samples in terms of one-year and two-year periods and obtained accuracies of 0.87 and 0.75, respectively.

Zhang et al. [11] developed a method to detect Android malware employing sustainability analysis, which they called APIGRAPH. To construct APIGRAPH, a private dataset was collected consisting of 322,594 samples (290,505 normal apps and 32,089 malware apps). The authors extracted API calls, exceptions, and permissions features and employed RF [8], Model Pool, SVM [10], and deep learning neural networks (DNNs) [12] classifiers. They evaluated their developed method using MamaDroid [13], DroidEvolver [14], Drebin-DL [15] and Drebin [16], based on sustainability, and found that the average enhancement for [13], [14], [15] and [16] was 19.2%, 19.6%, 15.6% and 8%, respectively.

Cai and Jenkins [17] investigated how Android app behavior might change over time. For this purpose, the authors used 155 predefined metrics from [18], which are based on general, ICC and security perspectives. They added the extent, frequency, and distribution for the source and sink invocations of sensitive API calls. A dataset was built including 6432 apps (3431 normal apps and 3001 malware apps). In order to evaluate their approach, they constructed two groups of datasets, based on the year, then performed a comparison using the predefined metrics to rank the most informative metrics and found 52 features to be most informative, which were used for further evaluations. They employed RF and obtained an accuracy of 93% while achieving an accuracy of 82% for their sustainability metric.

Cai et al. [19] implemented a scheme called Droidcat to detect Android malware in terms of systemic app-level profiling. The authors created a private dataset consisting of 34343 apps (17,365 normal apps, 16,978 malware apps). Then, they reduced the samples to 271 apps (136 normal apps, 135 malware apps) meeting their requirements. A total of 122 metrics were defined based on structure (method calls, declaring classes, callback), ICC (Intent, carrying data through URI only), and security (distributions of sources, sinks). The authors utilized RF to detect unknown malware samples, and obtained Precision, Recall and F1 of 97.96%, 97.91%, and 97.84%, respectively. Another experiment was carried out to evaluate the sustainability and obtained results with small standard deviations of 1.34-2.38% in terms of F1.

An approach was introduced by Cai [20] called DroidSpan based on behavioral profiling features. The author collected a total of 26382 samples (13,627 normal apps and 12,755 malware apps), then extracted 52 features based on the extent of sensitive access, categorization of sensitive data and operations accessed, and sensitive method-level control flows. Then, the approach employed RF [8], k-NN [9], SVM with both linear and radial basis function kernels [21] , decision trees [22], naïve Bayes [23] with three models (Gaussian, Multinomial, and Bernoulli), AdaBoost [24], Gradient Tree Boosting [25], Extra Trees, and the Bagging classifier, and evaluated DroidSpan in terms of F1-measure, recall and precision. Among all classifiers, RF obtained the best results. Then, another experiment was performed to examine the sustainability based on same-period detection and obtained 92.88%, 92.68% and 92.61% for precision, recall and F1-score.

B. Common Public Datasets

Current approaches rely on two types of data: private and public datasets. This section summarizes common public datasets.

1) Drebin: Arp et al. [16] built a dataset called Drebin between 2010 and 2012 and comprised of 123, 453 samples for normal applications and 5560 abnormal samples from 179 different families. It is available on https://www.sec.cs.tu-bs.de/~danarp/drebin/.

2) AndroZoo: Allix et al. [27] began building a public dataset in the latter part of 2011, called AndroZoo. The authors implemented a crawling tool to examine the application if it had not been downloaded previously. Then, they installed the application, calculated the s SHA256 checksum, and stored the sample. These samples were submitted to VirusTotal, a portal that allows users to analyze potential malware using various antivirus scanners, including several commercial products, such as McAfee, Symantec, and Avast. The total instances of AndroZoo is 10,774,100 samples and available on https://androzoo.uni.lu/.

3) Malgenome: Malgenome was introduced by Yajin and Xuxian [28] with total of 1260 malware samples covering 49 Android malware families between 2010 and 2011. Malgenome is available on http://malgenomeproject.org/.

4) Contagio Mobile Mini-dump: Contagio Mobile Mini-dump was developed by Mila [29]. Data collection involved a blog published in 2008 that allowed researchers to upload and download malware. As of April 2020, 370 malware samples had been collected. We tested this dataset and found some samples cannot be installed. Contagio can be found on http://contagiodump.blogspot.com/?m=1.

5) PRAGuard: PRAGuard is a publicly available dataset introduced by Maiorca et al. [30]. The dataset contains 10479 malware samples from on 50 malware families. PRAGuard is available on http://pralab.diee.unica.it/en/AndroidPRAGuardDataset.

6) Android Malware Dataset (AMD): The AMD was compiled by Wei et al. [31], and consisted of 24,650 samples collected between 2010 and 2016. The dataset includes various types of malware, such as Trojan, backdoor, and ransomware.

TABLE I. SUMMARY OF UTILIZED PUBLIC DATASET

Dataset	# of samples	Years of collection
Drebin [16]	5560	2011 - 2012
CICAndMal2017 [26]	10854	2015 - 20017
AndroZoo [27]	16,487,972	Late 2011- 2016

The authors employed antivirus scan results as well as automation methods to classify these samples based on behavioral semantics (135 varieties), and 71 malware families. It is available on http://amd.arguslab.org/ .

III. PROPOSED SCHEME

The proposed scheme categorizes into two parts: first part relies on examining how feature engineering might affect on obtained results. The machine learning classifier extracts static and dynamic features, finds most informative features and employs state of the art machine classifiers, while the deep learning scheme studies how deep learning might enhance the gained performance. We performed several scenarios to present the effectiveness of developed scheme.

A. Machine Learning Classifiers

The Machine Learning classifiers scheme consisted of four parts. First, we build a dataset comprised of public and private datasets. Then, two types of features are extracted: permission-based and network traffic features. Once the features are extracted, the most informative features are ranked for further analysis in the third phase. Finally, the developed scheme is evaluated among four scenarios: detecting malware in terms of binary classification, classifying the samples based on their packages and families, and finally considering sustainability metrics in our scheme. Fig. 1 depicts the structure of the proposed scheme.

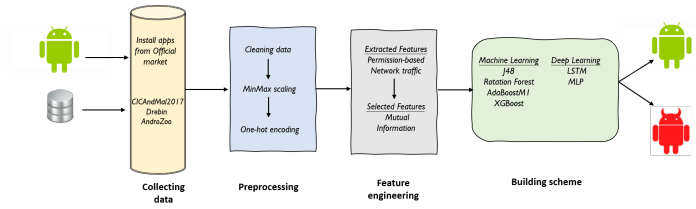

Fig. 1. Architecture of the Implemented Scheme.

1) Acquiring Data: Since the commonly used public datasets were collected between 2008 and 2017 [2], we started to construct a public dataset to be available in the near future for academic research purposes. For this purpose, we picked three public datasets: Drebin [16], CICAndMal2017 [26] and AndroZoo [27], which are summarized in Table I. We, then, targeted the official market for Android devices: the Google Play store to download up to 500 popular apps for 30 different categories using 15 different smartphone devices. Finally, we analyzed and stored downloaded apps in a local database. The process of collecting apps started November 2020 and we are continuing to build our dataset.

2) Pre-processing the Data: Once the data was collected, we performed a pre-processing step for the data which aims to observe any possible noise, remove duplicate apps and keep the updated version of the app, as well as to find missing, redundant and invalid data. For normalization, we used Min-Max scaling [32], and One Hot Encoding [33] to transform non-numeric data into numeric values. After performing this process, we had a total dataset of 15,000 malware apps, and 16500 normal apps. For balance, we constructed an updated dataset with a total of 30,000 samples (15,000 normal apps and 15,000 malware apps) and continued working to make the dataset larger.

3) Extracting Features: To extract the features from our dataset, we developed a tool called AndroAPKF Analyzer, installed it on our devices, ran each app, then extracted permission-based and network traffic features. We extracted two types of features: permission-based (280 extracted features) and network traffic (80 extracted features).

Once the features are extracted, they are stored temporarily in the phone, then sent to the local database for further process. The final stored data had a CSV file extension.

4) Selecting Features: Finding the most informative features is a subsequent phase during construction of the scheme. Using selected features will help to reduce evaluation time and over-fitting and improve the obtained results. There are several approaches that can be used to find most informative features such as mutual information (MI), which is a method to measure the mutual independence of the amount of information that a variable contains about the occurrence of another variable [34].

$$MI(X;Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) log_p \frac{x,y}{p(x)p(y)} \quad (1)$$

where X, Y are discrete or continuous random variables, $p(x)p(y)$ is the product of marginal distributions. For our approach, we adopted the implemented ranking scheme proposed by Alzubaidi et. al [35] to find the most informative features. We ranked permission features and network traffic individually in terms of the top 10, top 50, and all features. Tables II and III present highest informative extracted permission and network traffic features, respectively.

TABLE II. TOP 10 PERMISSION-BASED FEATURE RANKING USING [35]

Feature set	Rank
Android.permission.INTERNET	0.937
Android.permission.READ_PHONE_STATE	0.925
Android.permission.ACCESS_NETWORK_STATE	0.909
Android.permission.SEND_SMS	0.886
Android.permission.WRITE_EXTERNAL_STORAGE	0.881
Android.permission.RECEIVE_BOOT_COMPLETED	0.867
Android.permission.ACCESS_WIFI_STATE	0.852
com.Android.launcher.permission.INSTALL_SHORTCUT	0.834
Android.permission.INSTALL_PACKAGES	0.785
com.android.alarm.permission.SET_ALARM	0.748

B. Experimental Setup

We ran experiments using Microsoft Windows 10 on a 2.11 GHz Intel Core i7 178 processor with 16 GB RAM dGPU

TABLE III. TOP 10 NETWORK TRAFFIC FEATURE RANKING USING [35]

Feature set	Rank
Source IP address	0.913
Source Port	0.906
Destination IP address	0.883
Destination Port	0.876
Flow Duration	0.833
Total Foward Packets	0.795
Total Length of Forward Packets	0.792
Total Backward Packets	0.748
Total Length of Bakward Packets	0.741
Maximum Forward Packet	0.705

device. We also used a virtual machine p2.xlarge EC2 instances to perform further experiments and a local database for storing the data.

C. Performance Evaluation Metrices

Common evaluation metrics that can be used to examine the performance of the implemented methods for detecting malware such as true positive, false positive, accuracy, recall, precision and sustainability are summarized in Table IV. In our implemented scheme, the data are evaluated using accuracy, precision, recall and f1-score.

TABLE IV. COMMON EVALUATION METRICS

Metric	Definition
True positive	A sample is a true positive if the sample is labelled positive as well as prediction being positive
False positive	A sample is a false positive if the sample is labelled negative while it is predicted as a positive
False negative	A sample is called false negative when the sample is labelled positive and is predicted as a negative class
Precision	It is the proportion of correctly classified instances to the instances predicted as positive
Recall	It is the proportion of instances predicted to be positive to the total positive instances
Sustainability	Indicates how the model is adopted for new samples with retraining (re-usability) and without retraining (stability) over times

D. Experimental Evaluation

We initially utilized four machine learning classifiers to examine the effectiveness of the ranking features: J48 [36], rotation forest [37], AdaboostM1 [38], and XGBoost [39]. Table V outlines these classifiers with respect to their definition and platform.

We evaluated the machine learning classifiers using ranking features with the top 10, 50 and all features for the detection of unknown malware (binary classification) using permission and network traffic features.

Using Permission based Feature: We compare the utilized classifiers using top 10,50 and all features in terms of accuracy, precision, recall and f1-score. Among all three feature sets, rotation forest achieved better results with accuracies of 94.4%,98.6% and 96.5%. The rest results are illustrated in Tables VI, VII, and VIII.

TABLE V. CHARACTERISTICS OF THE CLASSIFIERS EMPLOYED

Classifier	Definition	Availability
J48 [36]	Implemented classifier of C4.5 decision tree	Weka 3.8.5
Rotation Forest [37]	Ensemble classifier, categorizes features into subsets, then run Principal Component Analysis on each subset	Weka 3.8.5
AdaBoostM1 [38]	An ensemble classifier used for boosting a nominal-class	Weka 3.8.5
XGBoost [39]	A boosted decision tree, which associates objective functions based on the gradient of the loss optimized function	R 4.0.0

TABLE VI. MALWARE DETECTION BASED ON PERMISSION FEATURE SET (TOP10 FEATURES)

Classifier	Accuracy	Precision	Recall	F1-score
J48	91.8	91.8	91.7	91.7
Rotation Forest	**94.4**	**94.5**	**94.3**	**94.5**
AdaBoostM1	89.6	88.6	89.7	89.5
XGBoost	93.7	93.6	93.5	93.7

1) Using Network Traffic Feature: We also evaluated the proposed approach using network traffic with respect to top 10,50 and all features. Among employed aforementioned classifiers,XGBoost is able to gain the best results with accuracies of 89.8%, 94.2% and 91.6%. The rest results are illustrated in Tables IX, X, and XI.

IV. DEEP LEARNING SCHEME

Deep learning algorithms are a subset of machine learning algorithms based on artificial neural networks, which can be applied to various fields such as computer vision, speech recognition, natural language processing, and lately malware detection. We employed two deep learning classifiers: long short-term memory (LSTM) [40] and multilayer perceptions (MLPs) [41] to enhance the obtained results.

A. Long-Short Term Memory

A brief definition Long short-term memory (LSTM) considers recurrent structures having the ability to learn a sequence of data. Generally, the sequence of features for the Android apps are $x = (x_1, x_2, x_3, ...x_t)$, which represent the input values to the LSTM. Then, the calculated output will be denoted as $ot = (o_{t1}, o_{t2}, o_{t3}, ...o_{tt})$ with respect to continuous calculation of the forget state vector, input/update gate activation vector, output vector, and cell state vector for time $T = 1, 2, 3, ...t$. The calculation of the hidden layer can be calculated as listed below.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

TABLE VII. MALWARE DETECTION BASED ON PERMISSION FEATURE SET (TOP50 FEATURES)

Classifier	Accuracy	Precision	Recall	F1-score
J48	97.3	97.2	97.3	97.3
Rotation Forest	**98.6**	**98.5**	**98.4**	**98.5**
AdaBoostM1	95.4	95.5	95.4	95.4
XGBoost	98.5	98.5	98.6	98.5

TABLE VIII. MALWARE DETECTION BASED ON PERMISSION FEATURE SET (ALL 280 FEATURES)

Classifier	Accuracy	Precision	Recall	F1-score
J48	95.4	95.3	95.2	95.4
Rotation Forest	**96.5**	**96.6**	**96.4**	**96.5**
AdaBoostM1	91.2	91.1	91.2	91.3
XGBoost	96.4	96.3	96.3	96.4

$$\tilde{c_t} = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t o c_{t-1} + i_t o \tilde{c_t}$$

$$h_t = o_t o \sigma_h(c_t)$$

where ft, it, ot,ct, ht, and xt denote the forget state vector, input/update gate activation vector, output vector, cell state vector, hidden state vector, and input vector, respectively.

B. Experimental Result

We evaluated the deep learning scheme in three scenarios, as listed below.

1) Detecting unknown Malware using Binary Class Classification: We initially performed a comparison among: LSTM, MLP and XGBoost in terms of detecting unknown malware using accuracy, precision, recall and f1-score using permission-based features. The obtained results show the superiority of LSTM compared to MLP and XGBoost, which achieved accuracy, precision, recall and f1-score higher than 99%. Fig. 2 depicts the results.

2) Detecting Categories of Malware : Another comparison was performed to examine how these classifiers are able to detect malware based on malware categories. Our malware samples belong to four categories: Adware, Ransomware, Scareware and SMS Malware. We performed four multi-class classifications to examine to which malware family each malware belong to. Comparing LSTM, MLP and XGBoost, LSTM is achieved the better results with accuracy of 97.5%. Fig. 3 outlines the comparison among these classifiers in terms of accuracy, precision, recall and F1-score.

3) Detecting Malware Families: A third comparison was performed to detect which package the malware belongs to. In our evaluation, we defined 45 malware families; therefore, we performed 45 class- classification procedures to examine the samples. We compared LSTM, MLP, and XGBoost and found LSTM achieved best accuracy of 99.5%. Fig. 4 illustrates the obtained results.

TABLE IX. MALWARE DETECTION BASED ON NETWORK TRAFFIC FEATURE SET (TOP10 FEATURES)

Classifier	Accuracy	Precision	Recall	F1-score
J48	87.7	87.6	87.6	87.7
Rotation Forest	88.6	88.5	88.6	88.5
AdaBoostM1	83.3	83.2	83.2	83.3
XGBoost	**89.8**	**89.6**	**89.7**	**89.6**

TABLE X. MALWARE DETECTION BASED ON NETWORK TRAFFIC FEATURE SET (TOP50 FEATURES)

Classifier	Accuracy	Precision	Recall	F1-score
J48	90.4	90.4	90.3	90.3
Rotation Forest	92.6	92.5	92.4	92.5
AdaBoostM1	86.5	86.5	86.4	86.5
XGBoost	**94.2**	**94.1**	**94.2**	**94.1**

4) Examining the Sustainability Performance: Third evaluated of our scheme is examined how our scheme is sustained. We divided the datasets into normal and malware apps based on implemented year with a span of nine years (2010 to 2019) . Then, we trained and tested apps that developed in the same year. The average obtained accuracy is 92.21%. Table XII presents the obtained results in terms of accuracy, precision and recall metrics.

C. Discussion

Although previous studies have obtained promising results in terms of sustainable detecting malware, there are some limitations, which are:

1) Choosing an updated dataset and selecting informative features are vital methods to detect unknown malware and improve obtained results. For example, Onwuzurike et al. [6] utilized a dataset collected between 2010 and 2017, neglected feature selection during building their scheme, and obtained accuracy between 75- 87% .
2) Studying mobile malware in terms of detecting unknown malware families, and categories using multi-class classifications has not covered from Onwuzurike et al. [6], Zhang et al. [11], Cai and Jenkins [17], Cai et al. [19], and Cai [20]

Table XIII presents a comparison between the proposed approach and prior studies. The introduced approach achieved accuracies of 99.2%,99.5%, 97.5%, and 92.2% for detecting unknown malware, detecting malware families, detecting malware categories, and sustainable malware detection, respectively.

V. CONCLUSION

Since the amount of mobile malware increases annually, it is important to implement malware detection that is able to detect possible threats to smartphone devices. This study leveraged machine and deep learning to detect unknown malware apps with accuracy over 99%. We also employed multi-class classification to detect the packages and families of mobile malware, and finally we examined the sustainability of our implemented scheme.

TABLE XI. MALWARE DETECTION BASED ON NETWORK TRAFFIC FEATURE SET (ALL 80 FEATURES)

Classifier	Accuracy	Precision	Recall	F1-score
J48	88.7	88.6	88.5	88.5
Rotation Forest	90.5	90.4	90.5	90.3
AdaBoostM1	82.3	82.2	82.3	82.4
XGBoost	**91.6**	**91.4**	**91.5**	**91.5**

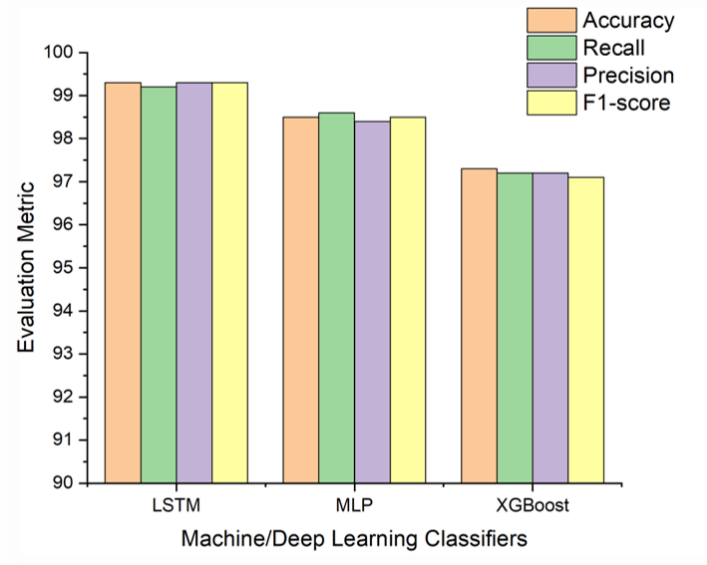

Fig. 2. Performance Evaluation Metrics based on Detecting unknown Malware.

TABLE XII. DATASET EVALUATION BASED ON SAME PERIOD OF DEVELOPED YEAR

Dataset	Accuracy	Recall	Precision
DS1	92.2	92.1	92.2
DS2	93.5	92.4	92.5
DS3	91.6	91.6	91.6
DS4	89.7	89.6	89.5
DS5	92.3	92.4	92.2
DS6	90.3	90.2	90.4
DS7	94.8	94.7	94.8
DS8	91.6	91.6	91.6
DS9	93.9	93.8	93.9
Average	**92.21**	**92.04**	**92.08**

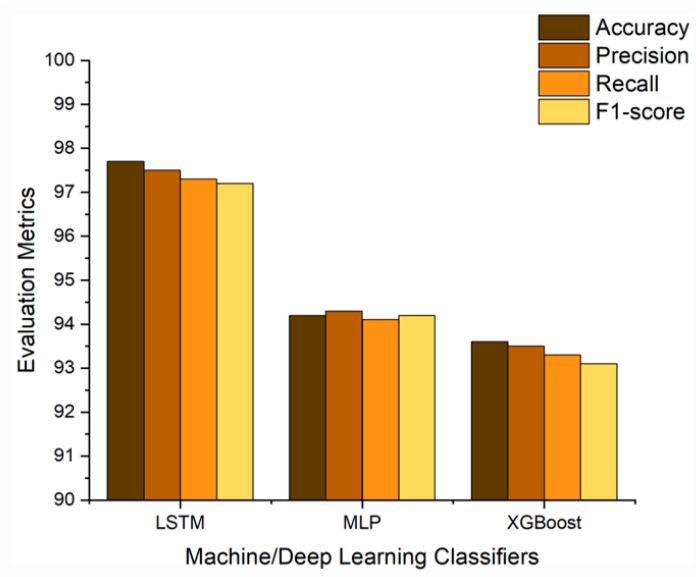

Fig. 3. Performance Evaluation Metrics based on Classifying Malware in Terms of Categories.

For future work, we aim to extend our dataset to download more apps, then make it available for academic use. We also plan to use various deep learning classifiers with different dynamic and static features to detect unknown malware, as well as assess datasets over a period of years for sustainability purposes.

ACKNOWLEDGMENT

The author would like to thank the Deanship of Scientific Research at Umm Al-Qura University for supporting this work by Grant Code: 18-COM-1-01-0007.

REFERENCES

[1] Alzubaidi, A., & Kalita, J. (2016). Authentication of smartphone users using behavioral biometrics. IEEE Communications Surveys & Tutorials, 18(3), 1998-2026.

[2] Alzubaidi, A., 2021. Recent Advances in Android Mobile Malware Detection: A Systematic Literature Review. IEEE Access.

[3] IT threat evolution in Q2 2021. Mobile statistics . Available online: https://securelist.com/it-threat-evolution-q2-2021-mobilestatistics/103636/ (accessed on: 12 - 10 - 2021)

[4] Mobile Malware . Available online: https://usa.kaspersky.com/resource-center/threats/mobile-malware (accessed on: 12 - 10 -2021)

[5] Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., & Xiang, Y. (2020). A survey of Android malware detection with deep neural models. ACM Computing Surveys (CSUR), 53(6), 1-36.

[6] Onwuzurike, L., Almeida, M., Mariconti, E., Blackburn, J., Stringhini, G., & De Cristofaro, E. (2018). A family of droids: Analyzing behavioral model based Android malware detection via static and dynamic analysis. arXiv preprint arXiv:1803.03448.

[7] Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3), 37-52.

[8] Pal, M., 2005. Random forest classifier for remote sensing classification. International journal of remote sensing, 26(1), pp.217-222.

[9] Laaksonen, J., & Oja, E. (1996, June). Classification with learning k-nearest neighbors. In Proceedings of International Conference on Neural Networks (ICNN'96) (Vol. 3, pp. 1480-1483). IEEE.

[10] Suykens, J.A. and Vandewalle, J., 1999. Least squares support vector machine classifiers. Neural processing letters, 9(3), pp.293-300.

[11] Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M. and Yang, M., 2020, October. Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (pp. 757-770).

[12] Hinton, G.E. and Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. science, 313(5786), pp.504-507.

[13] Mariconti, E., Onwuzurike, L., Andriotis, P., De Cristofaro, E., Ross, G. and Stringhini, G., 2016. Mamadroid: Detecting android malware by building markov chains of behavioral models. arXiv preprint

TABLE XIII. COMPARISON OF OUR PROPOSED WITH OTHER SUSTAINABLE DETECTION SCHEME. NOTE: NP REFERS TO NOT PROVIDED

Approach	Extracted features	# of samples	Detecting unknown malware	Detecting malware family	Detecting malware category	Sustainable malware detection
Onwuzurike et al. [6]	API calls	43,940	99%	NP	NP	87%
Cai and Jenkins [17]	extent, frequency, distribution for the source, API calls	6432	NP	NP	NP	82%-93%
Cai [20]	behavioral profiling features	26382	NP	NP	NP	92.88
Our Scheme	permission based, network traffic	30,000	99.2%	99.5%	97.5%	92.21%

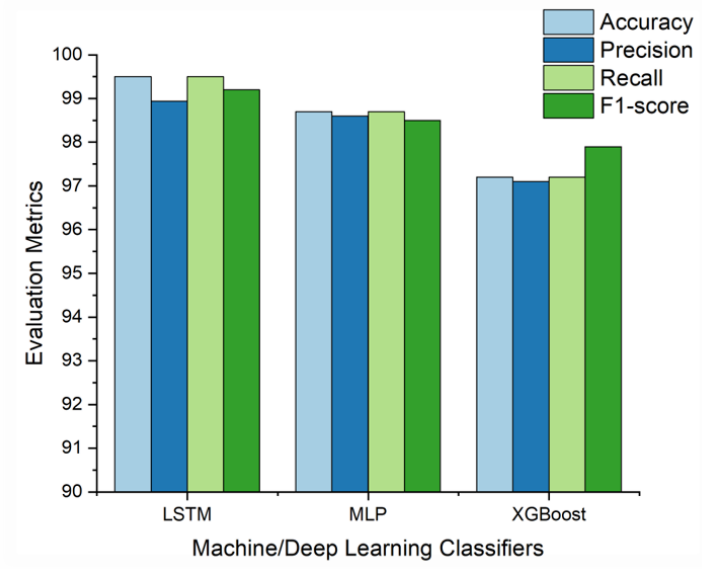

Fig. 4. Performance Evaluation Metrics based on Classifying Malware in Terms of Family.

arXiv:1612.04433.

[14] Xu, K., Li, Y., Deng, R., Chen, K. and Xu, J., 2019, June. DroidEvolver: Self-evolving Android malware detection system. In 2019 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 47-62). IEEE.

[15] Grosse, K., Papernot, N., Manoharan, P., Backes, M. and McDaniel, P., 2017, September. Adversarial examples for malware detection. In European symposium on research in computer security (pp. 62-79). Springer, Cham.

[16] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. Drebin: Effective and explainable detection of android malware in your pocket. In Ndss (Vol. 14, pp. 23-26).

[17] Cai, H. and Jenkins, J., 2018, May. Towards sustainable android malware detection. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (pp. 350-351).

[18] Cai, H. and Ryder, B.G., 2017, September. Understanding Android application programming and security: A dynamic study. In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 364-375). IEEE.

[19] Cai, H., Meng, N., Ryder, B. and Yao, D., 2018. Droidcat: Effective android malware detection and categorization via app-level profiling. IEEE Transactions on Information Forensics and Security, 14(6), pp.1455-1470.

[20] Cai, H., 2020. Assessing and improving malware detection sustainability through app evolution studies. ACM Transactions on Software Engineering and Methodology (TOSEM), 29(2), pp.1-28.

[21] Chang, C.C. and Lin, C.J., 2011. LIBSVM: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3), pp.1-27.

[22] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

[23] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

[24] Schapire, R.E., 2003. The boosting approach to machine learning: An overview. Nonlinear estimation and classification, pp.149-171.

[25] Schapire, R.E., 2003. The boosting approach to machine learning: An overview. Nonlinear estimation and classification, pp.149-171.

[26] Lashkari, A.H., Kadir, A.F.A., Taheri, L. and Ghorbani, A.A., 2018, October. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In 2018 International Carnahan Conference on Security Technology (ICCST) (pp. 1-7). IEEE.

[27] Allix, K., Bissyandé, T.F., Klein, J. and Le Traon, Y., 2016, May. Androzoo: Collecting millions of android apps for the research community. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR) (pp. 468-471). IEEE.

[28] Zhou, Y., & Jiang, X. (2012, May). Dissecting android malware: Characterization and evolution. In 2012 IEEE symposium on security and privacy (pp. 95-109). IEEE.

[29] Parkour, M, Mobile Signatures, Available online: http://contagiodump.blogspot.com, 2008, (Accessed on: 11/11/2021)

[30] Maiorca, D., Ariu, D., Corona, I., Aresu, M., & Giacinto, G. (2015). Stealth attacks: An extended insight into the obfuscation effects on android malware. Computers & Security, 51, 16-31.

[31] Wei, F., Li, Y., Roy, S., Ou, X., & Zhou, W. (2017, July). Deep ground truth analysis of current android malware. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 252-276). Springer, Cham.

[32] Patro, S., & Sahu, K. K. (2015). Normalization: A preprocessing stage. arXiv preprint arXiv:1503.06462.

[33] Rodríguez, P., Bautista, M. A., Gonzalez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. Image and Vision Computing, 75, 21-31.

[34] Anthony, M., Bartlett, P.L. and Bartlett, P.L., 1999. Neural network learning: Theoretical foundations (Vol. 9). Cambridge: cambridge university press.

[35] Alzubaidi, A., Roy, S. and Kalita, J., 2017, May. Ranking most informative apps for effective identification of legitimate smartphone owners. In 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 790-795). IEEE.

[36] Mathuria, M., 2013. Decision tree analysis on j48 algorithm for data mining. Intrenational Journal of Advanced Research in Computer Science and Software Engineering, 3(6).

[37] Rodriguez, J.J., Kuncheva, L.I. and Alonso, C.J., 2006. Rotation forest: A new classifier ensemble method. IEEE transactions on pattern analysis and machine intelligence, 28(10), pp.1619-1630.

[38] Cortes, E. A., Martinez, M. G., & Rubio, N. G. (2007). Multiclass corporate failure prediction by Adaboost. M1. International Advances in Economic Research, 13(3), 301-312.

[39] Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree

boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[40] Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

[41] Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons—from backpropagation to adaptive learning algorithms. Computer Standards & Interfaces, 16(3), 265-278.