Malware Detection in Android OS using Machine Learning Techniques

Maad M. Mijwil[1]
[1]*Computer Techniques Engineering Department, Baghdad College of Economic Sciences University, Baghdad, Iraq*

***Abstract*— Malware is a software that is created to distort or obstruct computer or mobile applications, gather sensitive information or execute malicious actions. These malicious activities include increasing access through personal information, stealing this valuable information from the system, spying on a user's activity, and displaying unwanted ads. Nowadays, mobile devices have become an essential part of our times, therefore we always need active algorithms for malware detection. In this paper, supervised machine learning techniques (SMLTs): Random Forest (RF), support vector machine (SVM), *Naïve Bayes (NB)* and decision tree (ID3) are applied in the detection of malware on Android OS and their performances have been compared. These techniques rely on Java APIs as well as the permissions required by employment as features to generalize their behavior and differentiate whether it is benign or malicious. The experimentation of results proves that *RF has the highest* performance with an accuracy rate of 96.2%.**

***Keywords*—Machine learning techniques, Malware Detection, Android OS, Java applications.**

I. INTRODUCTION

In recent years, the use of smartphones has been exploded [1]. These devices are everywhere in our daily lives [2]. This can be described by the various options these devices offer, such as third-party applications, mobile Internet connections, and the presence of cameras [3] [4]. According to the study conducted by the International Association of Mobile Operators GSMA [5], the number of mobile phone users worldwide will surpass 8 billion at the end of 2020, compared to the 5,600, million with which it closed 2016.

As the market for smartphone consumption has grown exorbitantly, the vulnerability of its operating systems against computer attacks has also increased. Hence, the safety of these devices must be a priority both in companies and in personal use. There are many mobile OSs on the market: Windows Phone, iOS, and Android OS [6] [7]. But Android is the most used in the world, according to the statistics for the year 2019, the number of users using Android OS reached around 2.5 billion

In 2018, Kaspersky Lab's products for the protection of mobile devices [8] recorded a notable increase in the number of malware package installations, which exceeded 9 million, almost triple that of 2016. Today, Mobile malware [9] [10] is on the rise and becoming more complex. Besides, more than 130,000, mobile banking trojans and more than 255,000 trojan were detected [11]. By comparison, more than 10 million malware installation packages were detected between 2006 and 2016, and around 3 million malwares were detected in 2015. Figure 1 shows various types of malware.

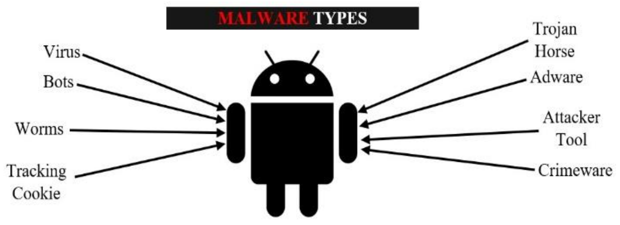


Fig. 1. Malware types

Machine learning (ML) is the branch of Artificial Intelligence that is applied to the investigation of agents/programs that learn or evolve based on their experience [12] [13]. The use of the various ML classifiers to detect malware apps has potential advantages over increased accuracy. it is possible for the ML to be applied efficiently in the extraction of information from large data sets, and consequently, in the detection of malware on Android [14]. The reason for the high number of malware applications (MSA- Malicious software application) on Play Store is that Android OS is an open source and the applications installed on the market are not sufficiently passed through security scans. Many solutions have been suggested for this, using different techniques such as Neural Networks, SVM, ID3, and NB [15] [16].

This investigation presents a malware detection system in Android with an approach based on the permissions required by Java applications and APIs, from which more specific information is obtained about the actions that the application tries to execute [17] [18]. The SVM, ID3, NB and RF supervised learning methods are used to classify samples.

The remainder of this article is as follow: Section 2 provides a description of the proposal. Results and discussion are described in Section 3. Lastly, the conclusions in section 4.

II. DESCRIPTION OF STUDY

A. *Samples*

The choice of samples depends on the collection of ZoneAlarm applications (for security apps). Today, this company has an anti-virus application and it is considered one of the ten most important applications in the year 2020 for anti-virus, and trusted by nearly 100 million users worldwide. Originally, a collection of 200 good apps and 200 other apps with malicious behavior were made trying to cover a certain randomness. The collection of kind

Manuscript received July 24, 2020; accepted December 22, 2020.
*Corresponding author: mr.maad.alnaimiy@baghdadcollege.edu.iq

applications was chosen to try to be diverse and reflect the different types of applications present in the play store app; and also, that they were proportional to the number of samples that exist in each type of application. The following aspects were taken into account when taking the set of malicious samples:

i. Various classifications according to the behavior of the ones that have the greatest impact: SMS Trojans, downloaders and banks, root editors (acquire administrator privileges and take control of the device), extortionists and criminals (blocking & encryption ransomware), adware (advertising software) and malicious tools (malware tool).
ii. Different variants within the same family of malicious programs, and more than one family (names according to antivirus) within a classification according to behavior.
iii. Applications of similar and different sizes within the same name and variants of the malicious program.

For analysis, a selection of 4245 samples from the same repository is made, of which 2325 are malicious and 1920 are benign. In addition, for the set of samples with malicious behavior it is discovered that there is at least one of each variant for each family already in the repository.

B. *Feature Extraction*

Many solutions have been proposed to protect Android users from serious malware threats. In many of these features' extraction is based on the permissions asked by the application. Nevertheless, mechanisms that are based only on permits fail for various reasons [19][20]:

i. The existence of certain permission in the AndroidManifest.xml of the application does not inevitably mean that it is applied within the code.
ii. A large number of requested permits, particularly experts, are not used within the code of the application itself, but are claimed by Ads packages.
iii. Malware can have malicious behavior without requiring any permission.

The purpose of this paper is to beat the deficiencies of a mechanism-based entirely on permissions and create a classifier for Android apps that can be applied in malware detection. To do this, at the feature extraction stage, it was decided to combine permissions and API calls.

C. *Permissions*

In the Android OS, the permissions requested by the app play an important part in managing access rights [21] [22]. Besides, all apps do not have any permission to access user data and change system settings. Android application package (APK) files generally consist of AndroidManifest.xml, classes.dex, maintenance and asset files. In the APK preprocessing stage, application samples are converted to source code and attributes are taken from it. Throughout the installation, the user must allow the app to access all the resources requested by it. Programmers should state the requested permissions for support in the AndroidMani-fest.xml. AndroidManifest.xml contains the name of the APK, the permissions, and some information on the version required by the application. The structure to declare permission in the license file is presented in Figure 2.

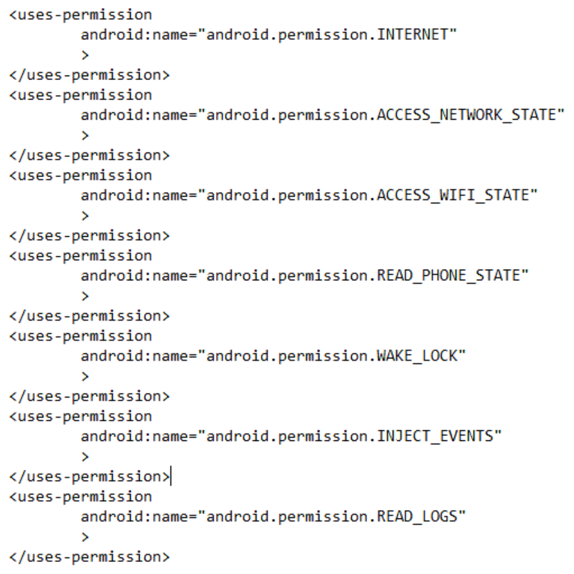
```
<uses-permission android:name="string" />
```

Fig. 1. The declaration of permissions in the An-droidManifest.xml.

To apply them as input in the MLTs, the An-droidManifest.xml is processed, uses-permission tags are searched and the string that represents the type of permission is obtained. A binary vector is then generated that indicates whether the permit is present or not in the analyzed app as shown in Figure 3 and figure 4 show XML file of FakeNetflix malware.

```
...
<uses-permission android:name=" an-
droid.permission.SEND_SMS " />
<uses-permission android:name=" an-
droid.permission.INTERNET " />
<uses-permission android:name=" an-
droid.permission.READ_CONTACTS" />
```

Fig. 3. Permission statement



```
<uses-permission
        android:name="android.permission.INTERNET"
        >
</uses-permission>
<uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE"
        >
</uses-permission>
<uses-permission
        android:name="android.permission.ACCESS_WIFI_STATE"
        >
</uses-permission>
<uses-permission
        android:name="android.permission.READ_PHONE_STATE"
        >
</uses-permission>
<uses-permission
        android:name="android.permission.WAKE_LOCK"
        >
</uses-permission>
<uses-permission
        android:name="android.permission.INJECT_EVENTS"
        >
</uses-permission>
<uses-permission
        android:name="android.permission.READ_LOGS"
        >
</uses-permission>
```

Fig. 4. FakeNetflix malware.xml

D. *API Calls*

In addition to permissions, Android applications have several malicious features, such as code [23]. Source code is not always available, but can anyone get a lot of information by decomposing jar type files. The methods called by the Android and Java APIs are one of the characteristics that can be obtained from these files. To obtain the disassembled code, the following steps were followed: Unzip all android file (.apk file), Convert the classes.dex file to .jar using the d2j-dex2jar.sh tool, and write the disassembled code to a file using a combination of the jar and java commands.

Next, the code is processed and the API calls are extracted and added to the binary vector generated when the permissions are extracted. The main challenge in obtaining this information is that many developers use obfuscation techniques to protect their code or avoid this type of analysis. The obfuscation techniques commonly used work by having classes and methods with a single letter ('a', 'b', etc.) and calling the desired methods indirectly through them.

Another challenge is that if you include the classes of the Android APIs (e.g. android.app.Activity), classification accuracy is lost. This is due to the large size of the API (several thousand classes) compared to the code that calls it, which leads to data noise. For this reason, this research only includes the Java API classes. These provide more specific information about the actions the applications try to perform.

E. Feature Selection

Feature selection is a key part of machine learning. This step leads to the process of reducing the inputs for analysis and processing or getting the most important entries and very necessary for various reasons. One of these reasons is that the selection of characteristics implies a certain degree of cardinality reduction to impose a cut in the number of attributes that will be taken into account when creating a model. Too many attributes can lead to overtraining as the model becomes more complex to the number of available data. Also, if the data set is large, most data mining algorithms will need a much larger learning data set. This step not only enhances the quality of the model but also gives the modeling process more effective. If the user uses unnecessary columns when creating the model, more RAM and CPU will be required during the training process, and the finished model will need more storage space [24].

Another reason for feature selection is that redundant or insignificant data makes it more difficult to detect important patterns. In the process of extracting features, a result of 2355 licenses and 2758 API calls were obtained, which made the attribute vector large and presented the aforementioned drawbacks. To solve this problem, it was determined to assign a value to each attribute. This is calculated by the difference between the percentage of malicious and benign applications that have a specific characteristic with respect to the total applications. In this way, values with a positive sign correspond to the attributes that occur most often in applications with malicious behavior and those that have a negative sign correspond to the most recurrent ones in healthy ones. In this paper it was decided to select the characteristics where the absolute value of the values is greater than a certain threshold. In this way, the attributes with the highest rate of benignity and malice are accepted. Tables 1 to 4 show a summary of the values granted to the attributes.

TABLE I. TOP 10 MOST FREQUENT API CALLS IN BENIGN APPLICATIONS.

API calls	Value
java/util_f/GregorianCalendar_file	-28
java/security_doc/SecureRandom_file	-25
java/util_f/concurrent/ConcurrentLinkedQueue_file	-25
java/util_f/Scanner_file	-23
java/lang_doc/IndexOutOfBoundsException_file	-21
java/lang_doc/Number_file	-21
java/IO_f/FilterOutputStream_file	-20
java/util_f/concurrent/CopyOnWriteArrayList_file	-20
java/util_f/logging/Logger_file	-19
java/nIO_f/channels/FileChannel_file	-19

TABLE II. TOP 10 MOST FREQUENT API CALLS İN MALİCİOUS APPLİCATİONS.

API calls	Value
java/util_f/zip_rar/Deflater_file	40
java/lang_f/Process_file	36
java/util_f/zip_rar/Inflater_file	35
java/lang_f/InstantiationException_file	31
java/lang_f/ProcessBuilder_file	31
java/lang_f/NoSuchMethodException_file	31
java/lang_f/ClassNotFoundException_file	30
java/net_f/Proxy_file	29
java/IO_f/FileReader_file	29
java/security_f/spec/PKCS8EncodedKeySpec_file	27

TABLE III. TOP 10 MOST FREQUENT PERMİTS İN BENİGN APPLİCATİONS.

Permissions	Value
Com_f.android.vending.billing_file	-33
Com_f.android.vending.check_license_file	-22
Com_f.google_site.c2dm.permission_app. receive-file	-17
android.permission_app.collection_photo_file	-7
android.permission_app.apply_credentials_file	-6
android.permission_app.modify_audio_ file	-4
android.permission_app.bluetooth_file	-4
android.permission_app.write_sync_ file	-4
android.permission_app.read_sync_ file	-4
android.permission_app.record_audio_file	-3

TABLE IV. TOP 10 MOST FREQUENT PERMİSSİONS İN MALİCİOUS APPLİCATİONS.

Permissions	Value
android.permission_app.sysm_alert_ file	94
android.permission_app.rece_boot_ file	94
android.permission_app.get_jobs_file	91
Com_f.android.statrt.permission_app.install_ shortcut_file	91
android.permission_app.send_message_file	90
android.permission_app.read_tphone_state_file	74
android.permission_app.receive_message_file	72
android.permission_app.read_message_file	72
android.permission_app.change_wifi_file	71
android.permission_app.change_network_ file	70

III. RESULTS AND DISCUSSION

This section shows an analysis of the results obtained with each of the SMLTs mentioned above (in abstract) using permissions and API calls**.** The dataset generated for Android MSA detection was evaluated on open-source WEKA software. WEKA [25] is a software that includes SMLTs used for data mining tasks. WEKA includes many tools for data preprocessing, clustering, visualization, classification, association rules, and, relationship search. In this context, the success of SMLTs was evaluated according to accuracy (*Accr*): *TNR* (True Negative Rate), *TPR* (True Positive Rate), *f-measure* tests.

$$Accr = \frac{TN + TP}{TN + TP + FP + FN} \quad (1)$$

$$TPR = \frac{TP}{FN + TP} \quad (2)$$

$$TNR = \frac{TN}{FN + TN} \quad (3)$$

$$f-measure = \frac{TP}{TP + FP + FN} \quad (4)$$

Hither, True Positive (*TP*) is the number of times that Android apps that are malicious are detected; False Negative (*FN*) is the number of times that malicious applications are detected normally; False Positive (*FP*) is the number of times that normal applications are detected as malicious; True Negative (TN) gives the number of times that normal applications are normally detected. The accuracy test determines the rate of the number of perfectly classified samples to the total number of samples. *TPR* shows the possibility of labeling malicious applications as malicious, while *TNR* indicates the possibility of labeling normal applications as normal. The harmonic average of *TPR* and *TNR* values, *f-measure*, provides the evaluation of *TPR* and *TNR* together.

In addition, the dataset including of 325 samples applied in this study was mixed randomly and 80% was determined as the learning set and the remaining 20% was determined as the test set. On the other hand, the learning set includes of 183 samples and also the test set includes of 46 samples.

In this paper, classification was made by using a dataset consisting of normal and malicious apps and SMLTs. The method that performs the classification process most successfully with the SMLTs used is determined according to the Accuracy, TPR, TNR and *f-measure* tests and the results are presented in Table 5. From this table, the RF classifier performance is higher than the others in terms of determined metrics. As can be noticed from the confusion matrix in Table 6 for Android MSA detection of the *RF* classifier, 20 malicious applications are classified as malicious. In other words, the *TP* value of the confusion matrix is 20. No malicious practice is classified as normal. So, the *FN* value in the confusion matrix is 0.2 of the normal applications are determined to be malicious. This means that the *FP* value in the confusion matrix is 2.24 applications that are normal are classified as normal. This indicates that the *TN* value in the confusion matrix is 24.

TABLE V. CLASSİFİCATİON OF RESULTS.

Criterion	*ACCR*	*TPR*	*TNR*	*f-measure*
SVM	84.5%	86%	85.8%	81.9%
ID3	80.7%	80.2%	82.1%	79.9%
NB	91.4%	95%	90.5%	88.7%
RF	96.2%	100%	94.7%	94.9%

TABLE VI. THE CONFUSİON MATRİX OF RANDOM FOREST (RF) ALGORİTHM

	Estimate	
Real	Malicious	Benign
Malicious	20	0
Benign	2	24

When the RF matrix confusion matrix is evaluated in terms of cost, it can be presented as an absolute success that the system can detect 20 out of 20 MSAs perfectly. However, contrary to the fact that 24 of 26 normal applications are determined to be normal, 2 of them are classified as malicious and this indicates that the system should be improved since it will create dissatisfaction for users. In brief, although the *FN* value is lower than the *FP* value in machine learning, it is necessary to reduce both classification errors in terms of cost.

IV. CONCLUSIONS

Mobile malware is a constant threat to Android users. As these devices take center stage in our daily lives, their safety and protection become more important. Therefore, the development of effective new techniques for malware detection should be a priority.

In this paper, MSA was determined with a different SMLTs on a dataset including of 325 samples in total. In comparison made on the performance of these algorithms, the RF classifier was found to be more successful. From the results obtained, it is understood that SMLTs are an effective solution for Android MSA detection and that more successful results can be obtained by developing. In this paper, significant results were achieved, 96.2% accuracy in the test stage at best. However, every day the evasion techniques adopted by the attackers continue to be refined, which means that work must continue in this direction.

ACKNOWLEDGMENT

I thank the anonymous editors and reviewers for their useful suggestions to develop this manuscript. Part of this study was presented orally in the Third International Conference on Data Science and Applications 2020 (ICONDATA'20).

REFERENCES

[1] M. Sarwa, and T. R. Soomro, "Impact of Smartphone's on Society", *European J. of Scientific Research,* vol.98, no.2, pp.216-226, 2013.
[2] S. Ali, S. Khusro, A. Rauf, and S. Mahfooz, "Sensors and Mobile Phones: Evolution and State-of-the-Art", *Pakistan J. of Scie.,* vol. 66, no.4, pp.386-400, 2013.
[3] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing", *IEEE Comm. Magazine.,* Vol.48, no.9, pp.140-150, 2010.
[4] M. Kedziora, P. Gawin, M. Szczepanik, and I. Jozwiak, "Android Malware Detection Using Machine Learning and Reverse Engineering", *Signal, Image Processing and Pattern Recognition Conf.,* Sydney, Australia, 22-23 December 2018.
[5] GSMA Report: "The Mobile Economy 2019.pdf", 2019.
[6] T. Grønli , J. Hansen, G. Ghinea, and M. Younas, "Mobile application platform heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS". *IEEE 2014 Advanced Information Networking and Applications (AINA) Conf.,* Victoria, Canada,13-16 May 2014.
[7] B. Padhya, P. Desai, and D. Pawade, "Comparison of Mobile Operating Systems". *Inter. J. of Innovative Research in Computer and Communication Engineering, vol.*4, no. 8, pp.15281-15286,2016.
[8] P. Unuchek, "SecurityList," Kaspersky Lab, Available online :https://securelist.com/pocketcryptofarms/85137/, 2018.
[9] B. Ganesh, A. Chakrabarti, and D. Midhunchakkaravarthy, "A Survey on Various Mobile Malware Attacks and Security Characteristics". *Inter. J. of Latest Trends in Engineering and Technology*, vol.8, no.2, pp. 448-454, 2017.
[10] J. Sahs, and L. Khan, "A Machine Learning Approach to Android Malware Detection". *European Intelligence and Security Informatics Conf.,* Odense, Denmark, 22-24 Augusts 2012.
[11] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer , "DL-Droid: Deep learning based android malware detection using real devices". *Computers & Security J.,* vol. 89 , no.1, pp.1-6, 2020.
[12] B. Zohuri, and F. M. Rahmani, "Artificial Intelligence Driven Resiliency with Machine Learning and Deep Learning Components". *Inter. J. of Nanotechnology & Nanomedicine*, vol. 4, no.2, pp. 1-8, 2019.
[13] T. Tiwari, T. Tiwari, and S. Tiwari, "How Artificial Intelligence, "Machine Learning and Deep Learning are Radically Different?". *Inter. J. of Advanced Research in Computer Science and Software Engineering*, vol.8, no.2, pp. 01-09, 2018.
[14] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, "Android Malware Characterization Using Metadata and Machine Learning

Techniques". *Security and Communication Networks-Hindawi*, Article ID 5749481, pp.1-12, 2018.

[15] S. Rana, C. Gudla, and A. H. Sung , "Evaluating Machine Learning Models for Android Malware Detection – A Comparison Study". *Network Communication & Computing (ICNCC) Conf.*, Taipei, Taiwan, 14-16 December 2018.

[16] R. Riasat, M. sakeena, C. Wang, A H. Sadiq, Y. J. Wang, "A Survey on Android Malware Detection Techniques". *Wireless Communication and Network Engineering (WCNE 2016) Conf., Beijing, China,* 20-21Novmber 2016.

[17] K. Bakour, H. M. Ünver, and R. Ghanem, "The Android malware detection systems between hope and reality". *SN Applied Sciences*, 1120, August 2019.

[18] W. D. Jie, M. C. Hao, W. T. En, L. H. Ming, and W. K. Ping, "DroidMat: Android malware detection through manifest and API calls tracing". *Information Security (Asia JCIS) Conf.*, Tokyo, Japan, 9-10 August 2012.

[19] M. Al Ali, D. Svetinovic, Z. Aung, and S. Lukman, "Malware detection in Android mobile platform using machine learning algorithms" *IEEE Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS) Conf.*, Dubai, United Arab Emirates, 18-20 December 2017.

[20] S. Arshad, A. Khan, M. A. Shah, and M. Ahmed, "Android Malware Detection & Protection: A Survey, *Inter. J. of Advanced Computer Science and Applications,*" vol.7, no.2, pp.463- 475, 2016.

[21] Z. R.Alkindi, M. Sarrab, and N. Alzidi, "Android Application Permission Model: Issues and Privacy Violation," *Free And Open Source Software (Fossc'2019) Conf.*, February 2019.

[22] X. Liu, and J. Liu, "A Two-layered Permission-based Android Malware Detection Scheme". *Mobile Cloud Computing, Services, and Engineering Conf., Oxford, UK*, 8-11 April 2014,

[23] X. Jiang, B. Mao, J. Guan, and X. Huang, "Android Malware Detection Using Fine-Grained Features," *Scientific Programming-Hindawi*, vol. 2020, Article ID 5190138,

[24] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroidmAlware detection using STaticanalySIs of Applications". *IFIP New Technologies, Mobility & Security (NTMS) Conf.*, Larnaca, Cyprus, 21-23 Novmber 2016.

[25] M. Hall, E. Frank, G. Holmes, and B. Pfahringer, "The WEKA Data Mining Software: An Update". *SIGKDD Explorations*, Vol. 11, no.1, pp.10-18, 2009.