International Journal of Information Security and Privacy
Volume 17 • Issue 1

A New Feature Selection Method Based on Dragonfly Algorithm for Android Malware Detection Using Machine Learning Techniques

Mohamed Guendouz, GeCoDe Laboratory, Dr. Moulay Tahar University of Saïda, Saïda, Algeria*
https://orcid.org/0000-0002-9230-8787

Abdelmalek Amine, GeCoDe Laboratory, Dr. Moulay Tahar University of Saïda, Saïda, Algeria
https://orcid.org/0000-0001-9327-7903

ABSTRACT

Android is the most popular mobile OS; it has the highest market share worldwide on mobile devices. Due to its popularity and large availability among smartphone users from all around the world, it becomes the first target for cyber criminals who take advantage of its open-source nature to distribute malware through applications in order to steal sensitive data. To cope with this serious problem, many researchers have proposed different methods to detect malicious applications. Machine learning techniques are widely being used for malware detection. In this paper, the authors proposed a new method of feature selection based on the dragonfly algorithm, named BDA-FS, to improve the performance of Android malware detection. Different feature subsets selected by the application of this proposed method in combination with machine learning were used to build the classification model. Experimental results show that incorporating dragonfly algorithm into Android malware detection performed better classification accuracy with few features compared to machine learning without feature selection.

KEYWORDS

Android Malware Detection, Dragonfly Algorithm, Feature Selection, Machine Learning

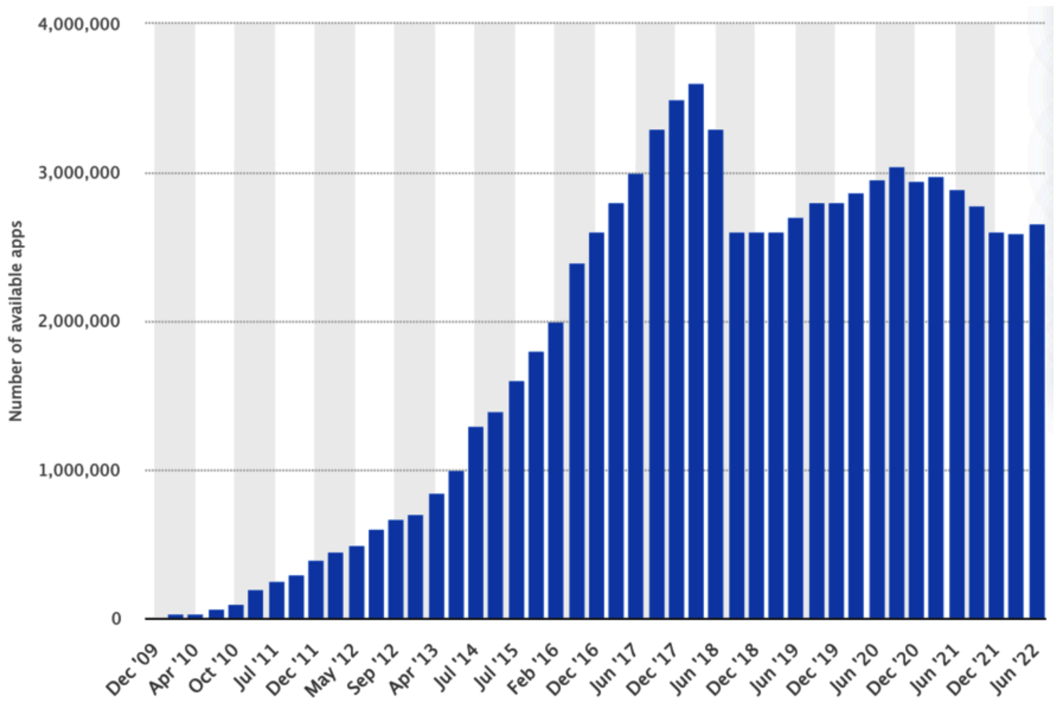
1. INTRODUCTION

Android, the Linux-based open-source mobile operating system is the largest used mobile OS in the world, it dominates the smartphone OS market with 73% share which makes it the most popular OS in the world, with over 2.5 billion active users. That success is due to the open-source nature of Android itself and for the large availability of smartphones that run it on the one hand, and on the other hand, the large number of apps and games freely available and easily accessible for users. Figure 1 shows the number of available applications in Google Play Store from December 2009 to March 2022.

DOI: 10.4018/IJISP.319018

*Corresponding Author

Figure 1. Number of Available Applications in the Google Play Store from December 2009 to March 2022



Android applications are mainly available for download on the Google Play Store which is the official Google app store, and other manufacturer-specific app stores such as: Samsung, Huawei, Xiaomi. Android applications are also available on many unofficial and unsecure third-party websites in a form of APK files. Applications downloaded from these third-party websites could be very dangerous and might contain malware codes since they are not verified by Google or any other device manufacturer, thus, it is necessary to detect malware applications in order to protect user personal data and device integrity.

The primary goal of mobile device malware is to gain access to user data stored locally on the device or on cloud as well as user information used in sensitive financial transactions in mobile banking apps. Mobile malware can be distributed in a variety of ways, including infected file attachments, shared files via Bluetooth and SMS phishing attacks. However, the primary malware distribution channel on mobile devices is currently app stores. According to a recent G DATA's Mobile Security Report (G DATA, 2022), the company's security experts counted more than 2.5 million malware apps for Android devices in 2021. As a result of these factors, Android malware is becoming increasingly problematic for both enterprise and individual users.

In order to deal with those dangerous attacks, researchers have proposed various methods and techniques to effectively detect malware apps on Android. Many of these methods use machine learning algorithms to classify Android apps into benign or harmful using popular classification algorithms. One of the most used techniques in literature is to use Android permissions as features to train and build one or multiple classification models, this type of techniques are known as permission-based methods.

In permission-based malware detection methods, generally the complete set of features is used as input for training classification algorithms without prior feature selection, because of the large number of Android permissions, which can exceed 150 permissions (XU, Zhang & Zhu, 2013), using the whole set of features makes training more difficult and can decrease detection accuracy. Feature selection is an essential stage in all machine learning-based techniques. Obtaining an appropriate feature set will not only help in enhancing classification accuracy, but will also help in decreasing the curse of dimensionality associated with most machine learning-based techniques.

In this paper, a novel permission-based machine learning method for Android malware detection with feature selection using dragonfly optimization algorithm is presented. The main contributions of this paper are summarized as follows:

- 5,000 malicious applications from different malware families and 5,000 benign Android applications from multiple categories were used to generate the dataset.
- Android permissions were extracted from each application in the dataset and used to generate the feature vector.
- A new feature selection method based on Dragonfly algorithm was proposed to select the most relevant permissions for Android malware detection using five machine learning algorithms.
- The performance of our proposed system is demonstrated through experiments using various evaluation metrics.

The rest of this paper is organized as follows. Section 2 explains briefly the related works. Section 3 describes the topics related to our study. In Section 4, the architecture of our proposed system for Android malware detection is presented. Section 5 presents the experimental settings and Section 6 presents and discusses the experimental results. Finally, the conclusion is presented in Section 7.

2. RELATED WORKS

Recently, many techniques and methods have been proposed to detect Android malware applications using machine learning techniques. Traditional Android malware analysis approaches can be classified into three main categories, static, dynamic and hybrid analysis. In this section, we describe briefly the most relevant proposed approaches according to the analysis method they use.

In static analysis, features are extracted from the application without actually running it, such as permissions, strings, API calls and opcode sequences. These features are mainly extracted from .apk files using various reverse engineering tools like Apktool. In (Li et al., 2018), authors proposed SIGPID, an Android malware detection system based on analyzing permission usage, they proposed a method to mine the permission data to find the most important permissions that might be useful in differentiating between benign and malicious apps. Then, they used machine learning techniques to classify various samples of malware and benign apps, their proposed system achieved a classification accuracy of 93.62% in detecting Android malwares. (Sanz et al., 2013) also used extracted permissions to train multiple machine learning classifiers like decision trees, random forest, naïve bayes and SVM on a dataset that consists of 357 benign applications and 249 malware sample. Random forest achieved the best results with 92.00% of classification accuracy.

In (Aafer, Du & Yin, 2013), authors generated relevant features by analyzing API-Level call traces to classify benign and malware apps. Then, they a number of machine learning techniques to classify benign and malware apps. As a result, their system achieved a classification accuracy of 99% using K-NN classifier.

In (Arp, Spreitzenbarth et al., 2014), authors proposed DREBIN, which is an on-device Android malware detection system. The authors extracted various static features from manifest file such as hardware components, permissions and intents. And also, other features from the disassembled code like restricted and suspicious API calls and network addresses. Authors have used SVM classifier for the classification process, results showed that SVM performed a classification accuracy of 94%. Static analysis approaches are not limited only to permissions analysis but also to other features of Android applications like API call (Shen et al., 2018; Mariconti et al., 2016), opcode sequence (Chen et al., 2018; McLaughlin et al., 2017), function call graphs (Gascon et al., 2013; Gao et al., 2018) and the combination of that features (Li, Wang & Xue, 2018; Wang et al., 2018).

In dynamic analysis, features are generated from runtime behaviors of Android applications by running them on real devices or emulators. Such dynamic features are system calls, API call, network traffic and CPU usage. For example, authors in (Canfora et al., 2015) used sequence of system calls and machine learning to automatically detect malware applications, their proposed achieved good detection results with 97% of classification accuracy. In (Wu & Hung, 2014), authors extracted sensitive API call traces from Android applications and applied n-gram model to represent the features. Then, they implemented an SVM classifier to build their malware detection model. Evaluation results showed that their proposed method reached a classification accuracy of 86.1%. Some researchers proposed the combination of multiple dynamic features to build the malware classifier such as combing network traffic, CPU usage and System call (Shabtai et al., 2012; Alam & Vuong, 2013; Afonso et al., 2015; Cai et al., 2018).

On the other hand, hybrid analysis consists of combining both static and dynamic analysis to detect Android malware applications. For example, authors in (Yuan et al., 2014) combined permissions and API calls with network data and dynamic behavior of applications to detect malwares using machine learning algorithms. In (Martinelli, Mercaldo & Saracino, 2017), authors used permissions, opcode sequence and app store information in combination with System call and SMS usage.

3. BACKGROUND

This section presents a detailed overview of some concepts from literature related to our study including: Android permissions, feature extraction, the Dragonfly algorithm and finally some machine learning algorithms used in our study.

3.1 Android Permission System

Android applications are installed as a compressed archive file called APK, the Android application package (APK) file of a third-party application is very similar to a compressed ZIP file. It stores all the contents of the application, including the compiled source code in a DEX format, app's resources like strings, images and colors, and the manifest file named AndroidManifest.xml. The manifest file defines the user permissions required by the application.

Permissions are the pillar of security mechanisms in Android, they control access to critical APIs that perform sensitive operations on hardware components and user's personal data, such operations include: camera access, device location, voice calls, emails and SMS. Every Android application must declare the permissions it needs in advance, the user is informed about the declared permissions during installation time or at runtime so he can grant or deny the application from a certain permission. Permissions are classified into two protection levels, normal and dangerous.

Normal permissions represent relatively minimal danger to the privacy of the user, they are automatically granted at install time by the permissions system. Such permissions are ACCESS_NETWORK_STATE and INTERNET. On the other hand, dangerous permissions have the ability to significantly impact the user's stored data, operation's workflow of other installed applications or the device itself. Dangerous permissions are requested as they are needed while the app is running and the application that need them must verify whether or not it has those permissions every time it runs, because the user can withdraw the permissions at any moment. For instance, CAMERA, READ_SMS and WRITE_EXTERNAL_STORAGE are examples of dangerous permissions.

3.2 Feature Selection

In machine learning systems, the large number of features, which are often unrelated or redundant, creates several issues, such as confounding the learning algorithm, over-fitting models, and decreasing the classification accuracy. The primary goal of feature selection is to choose the most significant features or attributes from a large number of possible features in order to achieve equivalent or even greater classification accuracy than if all the original features were used.

In general, feature selection methods are classified into three main types: filter, wrapper and embedded methods (Liu & Motoda, 1998). In filter-based approaches, the algorithm evaluates the quality of each feature in the dataset separately outside of the training phase. Features are then ordered downward according to their individual score and then some certain top features are selected. In wrapper-based methods the quality of features relies on the performance of a learning algorithm, in this type of methods, the algorithm uses a search strategy to look for a subset of features from the original dataset, then it trains a learning algorithm using only those selected features, based on the outcomes of the learning algorithm, the most highly evaluated features are selected. Embedded methods do both feature selection and algorithm training in the same phase.

In this paper, a wrapper-based feature selection method using a binary version of the dragonfly optimization algorithm was proposed for Android malware detection.

3.3 Dragonfly Algorithm (DA)

Dragonfly algorithm (DA) is one of the most recent and interesting nature-inspired swarm intelligence meta-heuristic optimization algorithm proposed by Mirjalili (Mirjalili, 2016). The DA, as its name suggests, it is inspired from the static (hunting) and dynamic (migration) swarming behaviors of dragonflies, these two proprieties constitute the base of the exploration and exploitation phases of DA, respectively, which are two essential operations in meta-heuristic optimization. The exploration phase is designed by the static swarming behavior of dragonflies, in this phase, dragonflies make small groups and fly over different areas looking for food. The exploration phase, however, is inspired from the dynamic swarming behavior, in this phase, dragonflies fly in larger groups over long distances towards one area or direction in order to migrate.

In order to model the swarm behaviors of dragonflies, five basic individual behaviors are used as follows:

- Separation represents how individuals are separated from other individuals in the neighborhood to avoid collision. This behavior is mathematically modeled as follows:

$$S_i = -\sum_{j=1}^{N} X - X_j \tag{1}$$

where X is the position of the current individual and X_j is the position of the j-th neighbor of X. N is the total number of neighboring individuals.

- Alignment indicates how an individual matches its velocity with the velocity of other neighboring individuals, and it is expressed as follows:

$$A_i = \frac{\sum_{j=1}^{N} V_j}{N} \tag{2}$$

where V_j is the velocity of the j-th individual.

- Cohesion represents the attraction of individuals towards the center of the swarm. It is defined as follows:

$$C_i = \frac{\sum_{j=1}^{N} Xj}{N} - X \tag{3}$$

- Attraction refers to the attraction of individuals towards food source. The attraction between individual i and the food source is calculated using the following equation:

$$F_i = F_{loc} - X \tag{4}$$

where F_{loc} represents the location of the food source.

- Distraction refers to the avoidance of enemies by an individual dragonfly and it is calculated as follows:

$$E_i = E_{loc} + X \tag{5}$$

where E_{loc} denotes the position of the enemy.

In order to simulate the movements of dragonflies and update their position in a search space, the DA uses two vectors: the step vector (ΔX) and the position vector (X). The step vector is similar to the velocity vector used in the Particle Swarm Optimization (PSO) algorithm () and it is updated as follows:

$$\Delta X_{t+1} = \left(sS_i + aA_i + cC_i + fF_i + eE_i\right) - w\Delta X_t \tag{6}$$

where s, a, c, f, e represents the weights of the separation, alignment, cohesion, attraction and distraction of the i-th individual, respectively, w denotes the inertia weight, and t represents iteration number. Using these weights, the DA can simulate different exploration exploitation behaviors during the optimization process.

The new position of the i-th individual is calculated as follows:

$$X_{t+1} = X_t + \Delta X_{t+1} \tag{7}$$

where t is the current iteration.

Algorithm 1 represents the pseudo code of the original dragonfly algorithm.

Algorithm 1. The pseudo code of the Dragonfly Algorithm

```
Initialize the population of the artificial dragonflies $X_i$ (i=1, 2, 3, …,n)
Initialize step vectors $\Delta X_i$ (i=1, 2, 3, …, n)
While (termination criteria is not met) do
    Evaluate all individuals by calculating their fitness values
    Update F and E
    Update the weights: s, a, c, f, e and w
    Calculate S, A, C, F and E
    Update step vectors ($\Delta X_{t+1}$)
    Update position vectors ($X_{t+1}$)
End While.
```

The dragonfly algorithm was originally designed to handle continues optimization problems. In this type of problems, the search space contains continues numerical values and the new position of individuals is updated by adding the velocity vector to the current position of the individual. This mechanism cannot be used to deal with binary optimization problems like feature selection.

In binary optimization problems, the position of an individual is updated by replacing one or more bits of the individual's position vector with a value of 0 or 1. The original version of the DA is converted to a binary version without making any modifications to its structure by using a transfer function (). A transfer function calculates the probability of changing one bit of the individual to 0 or 1 based on the step vector of that individual. In our proposed method, we calculate the probability of changing the individual's position by using the following transfer function:

$$T\left(\Delta X\right) = \left|\frac{\Delta X}{1+\sqrt{\Delta X^2}}\right| \tag{8}$$

The value of T(ΔX) is then used to update the position of the current individual as follows:

$$X_{t+1} = \begin{cases} \neg X_t & r < T\left(\Delta X_{t+1}\right) \\ X_t & r \geq T\left(\Delta X_{t+1}\right) \end{cases} \tag{9}$$

3.4 Machine Learning Algorithms

In order to assess the performance of our proposed method, five different machine learning based classification models were used in this study. These algorithms were used in combination with the dragonfly algorithm for selecting the best features to increase the classification accuracy of the Android malware detection system. The following section describes the used classification models:

- **Decision Trees (DT):** Decision Tree (DT) is a popular supervised machine learning technique used generally for both classification and regression. The DT algorithm builds a classification model to predict the correct class of a data instance by creating a tree-like structure using basic decision rules inferred from data features. This algorithm has grown popularity because of its simplicity and its achievement of good accuracy results in classification problems. C4.5, also called J48, is the most popular implementation of the DT algorithm (Quinlan, 1996) and it's the one used in this study.
- **Random Forest (RF):** RF is a supervised classification method based on the decision trees algorithm (Liaw & Wiener, 2002). It consists of a number of uncorrelated decision tree classifiers trained on various subsets of the dataset and uses average voting to improve classification accuracy and reduce overfitting phenomenon.
- **K-Nearest neighbor classifier (KNN):** The K-NN algorithm (Larose & Larose, 2005) works by finding a predefined number of samples, called neighbors, that are closest in distance to the current sample, and predict the class of that sample as the most aggregated class of its neighbors.
- **Naïve Bayes (NB):** The naïve bayes algorithm is a probabilistic supervised machine learning algorithm based on the Bayes Theorem of conditional probability (Murphy, 2006). The algorithm estimates the probability of each class label for a specific data instance, and the class with the highest probability is considered to be more likely the class label of that instance.
- **Support Vector Machine (SVM):** SVM, also known as Support Vector Network (SVN), is a supervised machine learning model for binary classification. The idea behind SVM is to create a hyperplane that can separate instances of two data classes as far as possible (Cortes & Vapnik, 1995).

4. METHODOLOGY

This section presents a detailed overview of our proposed classification system. The general methodology of the proposed Android malware detection system is described in Figure 2. The proposed approach is divided into three stages: (1) feature extraction, (2) feature selection and (3) machine learning classification of benign/malware apps. In the first step, permissions declared by both benign and malware Android apps are extracted directly from APK files and converted to a feature vector.

In the second stage, pertinent features are selected from the original feature vector of the extracted permissions using the dragonfly algorithm in combination with various machine learning classifiers, features are selected based on their performance in classification accuracy. In the last step, various machine learning classifiers are trained using the selected features in the past stage, and then evaluated and compared to each other.

4.1 Android Malware Dataset Creation

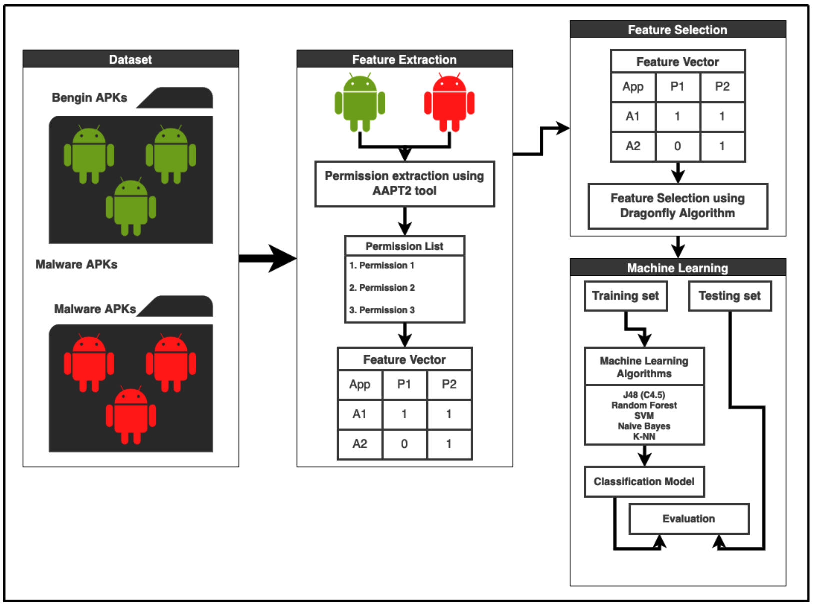
The dataset used in this paper consists of 5,000 malicious and 5000 benign Android applications. Android malware applications were taken from the DREBIN Android Malware Dataset (Arp et al., 2014). DREBIN dataset has been widely used by so many researchers in recent years, it consists of 5,560 Android applications from 179 different malware families.

The benign Android applications were downloaded from the APKCombo website (APKCombo, 2018), which is the largest APK store with more than 8 million Android applications and games. In this study, top Android applications from various categories were downloaded and taken as benign samples.

4.2 Feature Extraction

According to recent researches, permissions, among other static and hybrid features extracted from APKs, are considered to be the most relevant features in Android malware detection. In this study, we use permissions extracted from each benign and malware app in the dataset as features or data attributes in the classification.

Figure 2. The Architecture of the Proposed Methodology



In order to extract permissions from APKs, we used the AAPT2 (Android Asset Packaging Tool) from Google (Google, 2022). AAPT2 is a command-line build tool primarily used to compile and generate the APK file from a project's source code, but it is also used to extract some information from production-ready APKs like: package name, strings and most importantly permissions. The feature extraction phase works as follows: first, the requested permissions are extracted from each APK file in the dataset using AAPT2 dump command, then, a pair application/list of permissions is created for each application. At the end, the feature vector is generated for each application which consists of the pair: application/list of all permissions, this feature vector indicates the use of each permission by the application with 1 or 0, a value of 1 means that the permission is declared inside the APK file, and 0 means otherwise. Figure 3 shows an example of the output result of that command on a malicious sample.

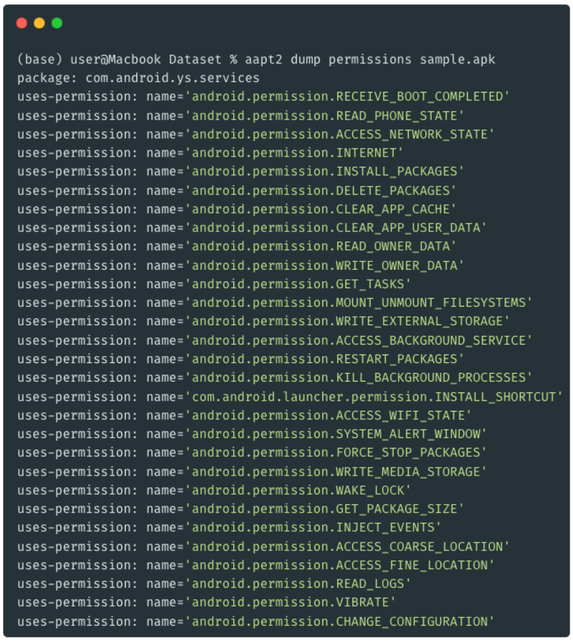
4.3 Feature Selection With the Binary Dragonfly Algorithm

Feature selection plays an important role in machine learning classification. A good feature selection can improve significantly the classification accuracy and detection rate of Android malware by keeping the most relevant features and removes the unnecessary features that are not useful.

In this paper, we propose a new wrapper-based feature selection method based on a binary Dragonfly Algorithm. The feature selection problem is considered as a binary search space optimization problem, which require encoding individual solutions as a binary vector, each position in the vector can take the value of 0 or 1. In our proposed method, a value of 0 means that the corresponding permission is not used for classification, and a value of 1 means the opposite, the permission will be used in classification. The length of the binary vector representing possible solutions is equal to the total number of features in the dataset.

In order to apply the dragonfly algorithm on the feature selection problem, we have used a binary version of the original algorithm by incorporating a transfer function to change the position of individuals from a continuous search space to a binary search space as discussed in Section 3.3.

Figure 3. Output Result of AAPT2 Dump Permissions Command



```
(base) user@Macbook Dataset % aapt2 dump permissions sample.apk
package: com.android.ys.services
uses-permission: name='android.permission.RECEIVE_BOOT_COMPLETED'
uses-permission: name='android.permission.READ_PHONE_STATE'
uses-permission: name='android.permission.ACCESS_NETWORK_STATE'
uses-permission: name='android.permission.INTERNET'
uses-permission: name='android.permission.INSTALL_PACKAGES'
uses-permission: name='android.permission.DELETE_PACKAGES'
uses-permission: name='android.permission.CLEAR_APP_CACHE'
uses-permission: name='android.permission.CLEAR_APP_USER_DATA'
uses-permission: name='android.permission.READ_OWNER_DATA'
uses-permission: name='android.permission.WRITE_OWNER_DATA'
uses-permission: name='android.permission.GET_TASKS'
uses-permission: name='android.permission.MOUNT_UNMOUNT_FILESYSTEMS'
uses-permission: name='android.permission.WRITE_EXTERNAL_STORAGE'
uses-permission: name='android.permission.ACCESS_BACKGROUND_SERVICE'
uses-permission: name='android.permission.RESTART_PACKAGES'
uses-permission: name='android.permission.KILL_BACKGROUND_PROCESSES'
uses-permission: name='com.android.launcher.permission.INSTALL_SHORTCUT'
uses-permission: name='android.permission.ACCESS_WIFI_STATE'
uses-permission: name='android.permission.SYSTEM_ALERT_WINDOW'
uses-permission: name='android.permission.FORCE_STOP_PACKAGES'
uses-permission: name='android.permission.WRITE_MEDIA_STORAGE'
uses-permission: name='android.permission.WAKE_LOCK'
uses-permission: name='android.permission.GET_PACKAGE_SIZE'
uses-permission: name='android.permission.INJECT_EVENTS'
uses-permission: name='android.permission.ACCESS_COARSE_LOCATION'
uses-permission: name='android.permission.ACCESS_FINE_LOCATION'
uses-permission: name='android.permission.READ_LOGS'
uses-permission: name='android.permission.VIBRATE'
uses-permission: name='android.permission.CHANGE_CONFIGURATION'
```

In our proposed method, in order to find the optimal features subset, features (permissions) are evaluated according to the following fitness function:

$$Fitness\left(X_i\right) = \alpha \times ErrorRate\left(C, S_i\right) + \beta \times \left|\frac{n}{N}\right| \tag{10}$$

where X_i is the i-th individual, S_i is the corresponding feature subset of X_i and C is a machine learning classifier. ErrorRate(C, S_i) represents the classification error rate obtained by classifier C using the feature subset S_i. n is the number of selected features and N is the total number of features in the dataset. α and β are two tuning parameters corresponding to the importance of classification performance and the number of features. α is fixed between 0 and 1 and β=1- α. In this paper we fixed α as 0.9 because the classification performance is slightly more important than the number of features.

Algorithm 2 shows the pseudo code of our proposed algorithm for feature selection with dragonfly algorithm.

4.4 Malware Classification Using Machine Learning Classifiers

We used five supervised machine learning algorithms to perform the feature selection using binary dragonfly algorithm. We also used the same five algorithms to conduct experiments on the dataset without using feature selection and compared the results with our proposed approach. The used algorithms are Decision Tree, Random Forest, Naïve Bayes, K-NN and SVM.

5. EXPERIMENTAL STUDY

This section describes briefly the technique used for evaluating our proposed method, the parameters used for each algorithm and the evaluation metrics used to evaluate the performance of the proposed Android malware detection system.

5.1 Experiment Settings

In this study, the K-fold cross-validation technique was used to prepare training and testing sets (Kohavi, 1995). The dataset was randomly sliced into 10 sub-datasets of equal size, and our proposed method was trained 10 times. Each time, one subset is used for testing, and the remaining 9 sub-datasets are used for training.

Our proposed feature selection algorithm was executed 30 times with random seed on an Apple MacBook Pro machine with 2.7 GHz Intel Core i5 processor and 8GB of RAM. We used the Python

Algorithm 2. The pseudo code of the proposed feature selection algorithm (BDA-FS)

```
Input: D, the dataset consisting of N permission features
Output: V, Optimal subset of features of D
    (1) Initialize the population of the artificial dragonflies $X_i$ (i=1, 2, 3, …,n)
    (2) Initialize step vectors $\Delta X_i$ (i=1, 2, 3, …, n)
While (termination criteria is not met) do
        Evaluate all individuals by calculating their fitness values using (10)
        Update F and E using equations (4) and (5)
Update the weights: s, a, c, f, e and w
Calculate S, A, C, F and E using equations (1) to (5)
Update step vectors ($\Delta X_{t+1}$) using equation (6)
Calculate $T(\Delta x)$ using equation (8)
Update position vectors $X_{t+1}$ using equation (9)
End While.
```

Table 1. Parameter settings

Algorithm	Parameter	Value
Binary Dragonfly Algorithm (BDA)	Population size	50
	Number of iterations	100
	Dimension of individual	Total number of features (permissions)
	α in fitness function	0.99
	β in fitness function	0.01
Decision Tree (DT)	criterion	gini
Random Forest (RF)	n_estimators: The number of trees in the forest.	100
Support Vector Machine (SVM)	kernel	RBF

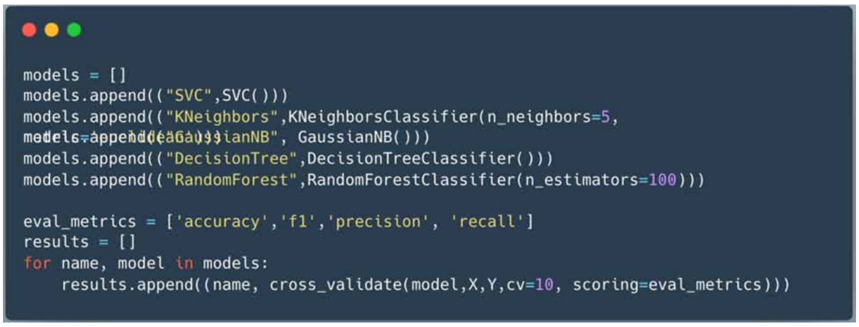
programming language and scikit-learn library for our implementation. The parameter settings of dragonfly algorithm and other machine learning algorithms are shown in Table 1.

Figure 4 shows a portion of the Python script used to train and evaluate machine learning classifiers.

5.2 Evaluation Metrics

In machine learning, binary classification results can be represented by a confusion matrix (Costa et al., 2007). A confusion matrix contains different type of errors made by a classification model. Table 2 shows the general format of a confusion matrix and its content.

Figure 4. Python Script for Classification



```
models = []
models.append(("SVC",SVC()))
models.append(("KNeighbors",KNeighborsClassifier(n_neighbors=5,
models.append(("GaussianNB", GaussianNB()))
models.append(("DecisionTree",DecisionTreeClassifier()))
models.append(("RandomForest",RandomForestClassifier(n_estimators=100)))

eval_metrics = ['accuracy','f1','precision', 'recall']
results = []
for name, model in models:
    results.append((name, cross_validate(model,X,Y,cv=10, scoring=eval_metrics)))
```

Table 2. Confusion Matrix

		Actual class	
		Positive (Malware)	Negative (Benign)
Predicted class	Positive (Malware)	True Positive (TP)	False Positive (FP)
	Negative (Benign)	False Negative (FN)	True Negative (TN)

True Positive (TP): the number of malicious applications correctly classified as Android malware.
False Positive (FP): the number of benign applications incorrectly classified as malicious applications.
False Negative (FN): the number of Android malware applications predicted as benign.
True Negative (TN): the number of benign applications correctly detected as benign.

The following evaluation metrics are derived from the confusion matrix:

- **Accuracy:** Represents the percentage of total correctly classified applications. It is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- **Precision:** represents the percentage of correct positive predictions relative to total positive predictions. It is defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall:** represents the percentage of correct positive predictions relative to total actual positives. It is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

- **F1 Score:** a weighted harmonic mean of precision and recall. The closer to 1, the better the model. F1 score is defined by the following equation:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

6. RESULTS AND DISCUSSION

This section shows and discusses the evaluation results obtained by our proposed method for Android malware detection.

6.1 Accuracy, Number of Selected Features and Training Time

In order to validate the effectiveness of our proposed feature selection method on the classification results, we firstly evaluated each machine learning classifier in term of accuracy using a 10-fold cross-validation. The average accuracy and the training time obtained by each classifier without feature selection and with feature selection using our proposed method in addition to training time taken by each algorithm to build the classification model are summarized in Table 3.

As presented in Table 3, accuracy values indicate that Random Forest (RF) algorithm had the best performance among all algorithms on classifying benign and malicious Android applications without using feature selection. The average accuracy achieved by RF algorithm is 96.52%. Decision Tree (DT) and SVM algorithms have achieved almost the same classification accuracy. On the other hand, Naïve Bayes and K-NN performed badly on detecting malwares. They have achieved the lowest classification accuracies among all other algorithms.

Regarding the average time taken by each algorithm to build the classification model, Table 3 shows that the five algorithms have achieved different training time values. We noticed that SVM

Table 3. Performance Comparison of Each Classifier in Term of Accuracy, Training Time and the Number of Selected Features Before and After Feature Selection

Algorithm	Without feature selection		With proposed BDA-FS		
	Accuracy (%)	Training Time	Accuracy (%)	Training Time	Number of selected features
Decision Tree (DT)	95.48	0.44s	95.29	0.02s	17
Random Forest (RF)	96.52	2s	96.33	0.70s	30
SVM	95.19	8.46s	94.31	2.80s	41
Naïve Bayes	58.97	0.47s	84.11	0.05s	52
K-NN	88.20	0.39s	94.38	0.22s	34

algorithm took longer time to build the classification model among all other algorithms, this is due to its linear and the large number of computational operations it needs to accomplish the work.

Tables 3 clearly shows the impact of our proposed feature selection method on improving both classification accuracy and training time and also in reducing the number of features. For instance, the classification accuracy values of Naïve Bayes and K-NN algorithms were significantly improved after integrating our proposed feature selection method to both of them, their accuracy values were increased by 25.14% and 6.18% respectively. The accuracy values of Decision Tree and Random Forest algorithms were also enhanced a little bit. However, their training time has been reduced significantly, the two algorithms took exactly 0.02s and 0.70s respectively to build their classification models using small feature subsets selected by our proposed method (17 for DT and 30 for RF). Alternatively, the classification performance of SVM has been impacted in a negative way, SVM's accuracy value was reduced by 0.88% but its training time was significantly enhanced. SVM took exactly 2.80s in place of 8.46s to build its classification model using only 41 features from a total of 114 features in the original dataset. Figure 5 demonstrates the impact of our feature selection method on the classification accuracy of all algorithms.

Talking about the performance of feature selection itself, Table 3 shows that the Decision Tree algorithm had the most significant result with 17 selected permissions.

6.2 Precision, Recall and F1-Score

To further investigate the overall performance of machine learning algorithms used in this study, we evaluated them in terms of precision, recall and F-Score. Table 4 presents obtained results by each machine learning classifier before and after integrating our proposed BDA-FS method. Results shown in Table 4 aggregate the previous results and corroborate the positive impact of our proposed method on enhancing Android malware detection.

Based on the data shown in Table 4, we made charts to compare the F1-Score obtained by each classifier before and after feature selection. Figure 6 shows a graph of the F1-Score.

7. CONCLUSION

In this paper, we have proposed a new feature selection method based on a binary Dragonfly Algorithm, called BDA-FS, for Android malware detection using machine learning techniques for classification. In this study, permissions extracted from Android applications using the AAPT2 tool were used as feature vectors for five supervised machine learning algorithms (Decision Tree, Random Forest, Support Vector Machine, Naïve Bayes and K Nearest Neighbors). In order to evaluate the performance of our proposed method, we have compared

Figure 5. Average Accuracy Obtained by Each Classifier Before and After Feature Selection

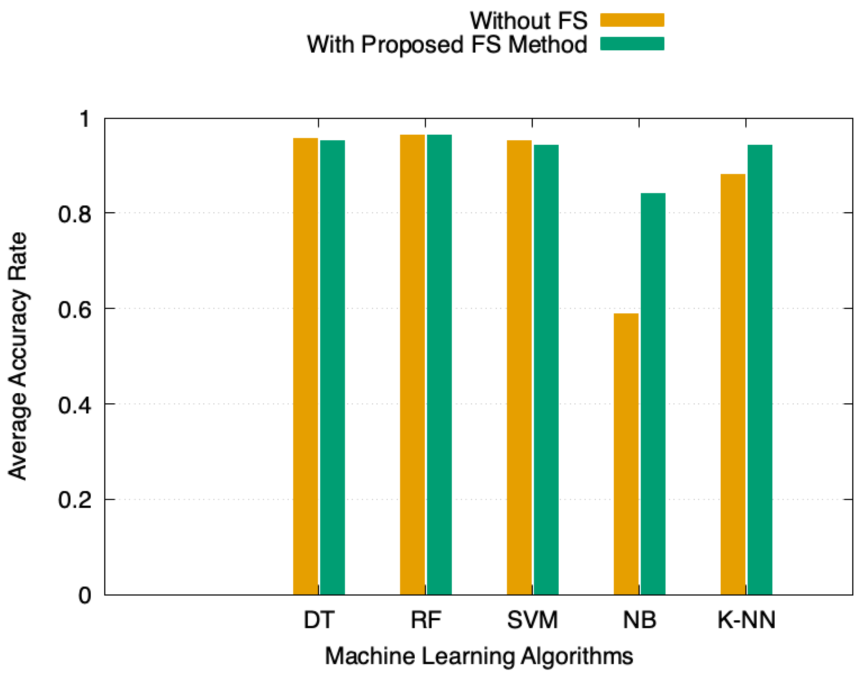


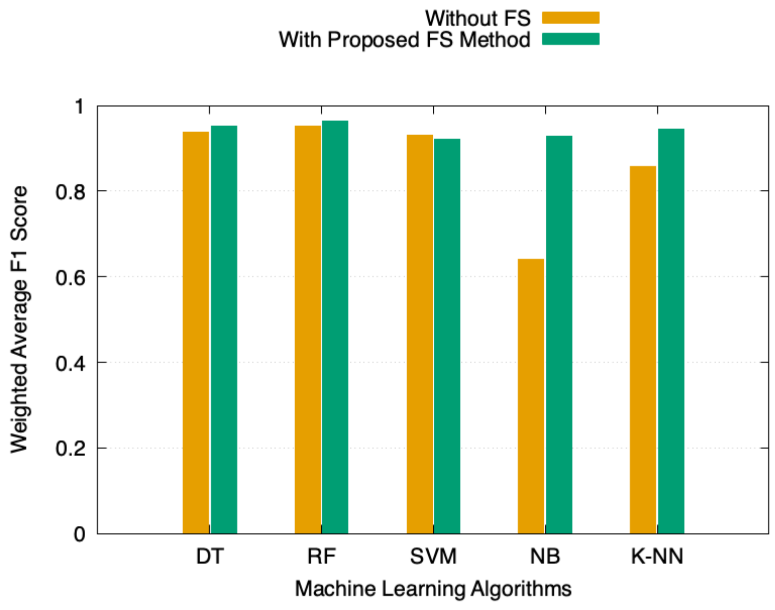
Table 4. Precision, Recall and F1-Score results obtained by each classifier with and without feature selection

Algorithm	Without feature selection			With proposed BDA-FS		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Decision Tree (DT)	0.948	0.928	0.938	0.953	0.953	0.953
Random Forest (RF)	0.970	0.934	0.952	0.963	0.963	0.963
SVM	0.974	0.893	0.932	0.9497	0.8935	0.9207
Naïve Bayes	0.473	0.988	0.640	0.725	0.918	0.929
K-NN	0.776	0.957	0.857	0,944	0,944	0.944

classification performance results of each classifier without using feature selection at all and with using DBA-FS method. Experimental results show that the combination of Random Forest with our proposed Dragonfly algorithm for feature selection achieved the best performance so far with an accuracy of 96.33% using only 30 features from a total of 114 features in the original dataset. Results also demonstrate that our proposed method has improved the performance of other algorithms in term of accuracy, training time and selected features.

This study confirmed that our proposed BDA-FS method can significantly improve the performance of machine learning techniques for Android malware detection by increasing the classification accuracy, reducing the feature space and improving training time.

Figure 6. F1-Score Results of Each Classifier Before and After Feature Selection



CONFLICT OF INTEREST

The authors of this publication declare there is no conflict of interest.

FUNDING AGENCY

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

Aafer, Y., Du, W., & Yin, H. (2013, September). Droidapiminer: Mining api-level features for robust malware detection in android. In *International conference on security and privacy in communication systems* (pp. 86-103). Springer, Cham.

Afonso, V. M., de Amorim, M. F., Grégio, A. R. A., Junquera, G. B., & de Geus, P. L. (2015). Identifying Android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques*, *11*(1), 9–17. doi:10.1007/s11416-014-0226-7

Alam, M. S., & Vuong, S. T. (2013, August). Random forest classification for detecting android malware. In *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing* (pp. 663-669). IEEE.

APKCombo. (2018). APKCombo. APK Combo. https://apkcombo.com/

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & 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).

Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011, October). Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices* (pp. 15-26). ACM. doi:10.1145/2046614.2046619

Cai, H., Meng, N., Ryder, B., & Yao, D. (2018). Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Transactions on Information Forensics and Security*, *14*(6), 1455–1470. doi:10.1109/TIFS.2018.2879302

Canfora, G., Medvet, E., Mercaldo, F., & Visaggio, C. A. (2015, August). Detecting android malware using sequences of system calls. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile* (pp. 13-20). IEEE. doi:10.1145/2804345.2804349

Chan, P. P., & Song, W. K. (2014, July). Static detection of Android malware by using permissions and API calls. In *2014 International Conference on Machine Learning and Cybernetics* (Vol. 1, pp. 82-87). IEEE. doi:10.1109/ICMLC.2014.7009096

Chen, T., Mao, Q., Yang, Y., Lv, M., & Zhu, J. (2018). Tinydroid: a lightweight and efficient model for android malware detection and classification. *Mobile information systems, 2018.*

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. doi:10.1007/BF00994018

Costa, E., Lorena, A., Carvalho, A. C. P. L. F., & Freitas, A. (2007, July). A review of performance evaluation measures for hierarchical classifiers. In *Evaluation methods for machine learning II: Papers from the AAAI-2007 workshop* (pp. 1-6).

G DATA. (2022, June 2). *G DATA Mobile Security Report*. G Data. https://www.gdatasoftware.com/news/2022/02/37321-g-data-mobile-security-report-more-than-25-million-new-malware-apps-for-android-devices

Gao, T., Peng, W., Sisodia, D., Saha, T. K., Li, F., & Al Hasan, M. (2018). Android malware detection via graphlet sampling. *IEEE Transactions on Mobile Computing*, *18*(12), 2754–2767. doi:10.1109/TMC.2018.2880731

Gascon, H., Yamaguchi, F., Arp, D., & Rieck, K. (2013, November). Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security* (pp. 45-54). IEEE. doi:10.1145/2517312.2517315

Google. (n.d.). *AAPT2 (Android Asset Packaging Tool).* Google. https://developer.android.com/studio/command-line/aapt2

Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Ijcai, 14(2), 1137-1145).

Larose, D. T., & Larose, C. D. (2005). K-nearest neighbor algorithm. *Discovering knowledge in data: An introduction to data mining, 90*, 106.

Li, D., Wang, Z., & Xue, Y. (2018, May). Fine-grained android malware detection based on deep learning. In *2018 IEEE Conference on Communications and Network Security (CNS)* (pp. 1-2). IEEE. doi:10.1109/CNS.2018.8433204

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, *14*(7), 3216–3225. doi:10.1109/TII.2017.2789219

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R News*, *2*(3), 18–22.

Liu, H., & Motoda, H. (Eds.). (1998). *Feature extraction, construction and selection: A data mining perspective* (Vol. 453). Springer Science & Business Media. doi:10.1007/978-1-4615-5725-8

Mariconti, E., Onwuzurike, L., Andriotis, P., De Cristofaro, E., Ross, G., & Stringhini, G. (2016). Mamadroid: Detecting android malware by building markov chains of behavioral models. arXiv:1612.04433.

Martinelli, F., Mercaldo, F., & Saracino, A. (2017, April). Bridemaid: An hybrid tool for accurate detection of android malware. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security* (pp. 899-901). ACM. doi:10.1145/3052973.3055156

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., & Joon Ahn, G. (2017, March). Deep android malware detection. In *Proceedings of the seventh ACM on conference on data and application security and privacy* (pp. 301-308). ACM. doi:10.1145/3029806.3029823

Mirjalili, S. (2016). Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing & Applications*, *27*(4), 1053–1073. doi:10.1007/s00521-015-1920-1

Murphy, K. P. (2006). Naive bayes classifiers. *University of British Columbia, 18*(60), 1-8.

Quinlan, J. R. (1996). Learning decision tree classifiers. [CSUR]. *ACM Computing Surveys*, *28*(1), 71–72. doi:10.1145/234313.234346

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Álvarez, G. (2013). Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions* (pp. 289-298). Springer, Berlin, Heidelberg.

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. doi:10.1007/s10844-010-0148-x

Shen, F., Del Vecchio, J., Mohaisen, A., Ko, S. Y., & Ziarek, L. (2018). Android malware detection using complex-flows. *IEEE Transactions on Mobile Computing*, *18*(6), 1231–1245. doi:10.1109/TMC.2018.2861405

Wang, W., Gao, Z., Zhao, M., Li, Y., Liu, J., & Zhang, X. (2018). DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features. *IEEE Access: Practical Innovations, Open Solutions*, *6*, 31798–31807. doi:10.1109/ACCESS.2018.2835654

Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., & Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, *9*(11), 1869–1882. doi:10.1109/TIFS.2014.2353996

Wu, W. C., & Hung, S. H. (2014, October). DroidDolphin: a dynamic Android malware detection framework using big data and machine learning. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems* (pp. 247-252). ACM. doi:10.1145/2663761.2664223

Xu, W., Zhang, F., & Zhu, S. (2013, November). Permlyzer: Analyzing permission usage in android applications. In 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE) (pp. 400-410). IEEE.

Yuan, Z., Lu, Y., Wang, Z., & Xue, Y. (2014, August). Droid-sec: deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM* (pp. 371-372). ACM. doi:10.1145/2619239.2631434

Mohamed Guendouz received his Bachelor's degree in computer science from the Dr.Tahat Moulay University of Saïda, Algeria in 2012, he received his Master's degree from the same university. Now, Mohamed Guendouz is a PhD student at Dr. Tahar Moulay University of Saïda and a researcher at the GeCoDe Research Laboratory, he works on Big Data and Social Networks Analysis, he participated in several international conferences in Algeria as an Author.

Abdelmalek Amine received an engineering degree in Computer Science, a Magister diploma in Computational Science and PhD from Djillali Liabes University in collaboration with Joseph Fourier University of Grenoble. His research interests include big data, IoT, data mining, text mining, ontology, classification, clustering, neural networks, and biomimetic optimization methods. He participates in the program committees of several international conferences and on the editorial boards of international journals. Prof. Amine is the head of GeCoDe-knowledge management and complex data-laboratory at UTM University of Saida, Algeria; he also collaborates with the "knowledge base and database" team of TIMC laboratory at Joseph Fourier University of Grenoble.