computers

MDPI

Article

An Improved Binary Owl Feature Selection in the Context of Android Malware Detection

Hadeel Alazzam [1,*], Aryaf Al-Adwan [2,*], Orieb Abualghanam [3,*], Esra'a Alhenawi [4] and Abdulsalam Alsmady [5]

[1] Department of Intelligent Systems, Al-Balqa Applied University, Al-Salt 19117, Jordan
[2] Department of Autonomous Systems, Al-Balqa Applied University, Al-Salt 19117, Jordan
[3] Department of Computer Science, University of Jordan, Amman 11942, Jordan
[4] Department of Software Engineering, Al-Ahliyya Amman University, Amman 19328, Jordan
[5] Department of Computer Engineering, Jordan University of Science and Technology, Irbid 22110, Jordan
* Correspondence: hadeel.alazzam@bau.edu.jo (H.A.); aryaf_aladwan@bau.edu.jo (A.A.-A.); o.abualganam@ju.edu.jo (O.A.)

Abstract: Recently, the proliferation of smartphones, tablets, and smartwatches has raised security concerns from researchers. Android-based mobile devices are considered a dominant operating system. The open-source nature of this platform makes it a good target for malware attacks that result in both data exfiltration and property loss. To handle the security issues of mobile malware attacks, researchers proposed novel algorithms and detection approaches. However, there is no standard dataset used by researchers to make a fair evaluation. Most of the research datasets were collected from the Play Store or collected randomly from public datasets such as the DREBIN dataset. In this paper, a wrapper-based approach for Android malware detection has been proposed. The proposed wrapper consists of a newly modified binary Owl optimizer and a random forest classifier. The proposed approach was evaluated using standard data splits given by the DREBIN dataset in terms of accuracy, precision, recall, false-positive rate, and F1-score. The proposed approach reaches 98.84% and 86.34% for accuracy and F-score, respectively. Furthermore, it outperforms several related approaches from the literature in terms of accuracy, precision, and recall.

Keywords: Android malware detection; binary owl optimizer; DREBIN dataset

Citation: Alazzam, H.; Al-Adwan, A.; Abualghanam, O.; Alhenawi, E.; Alsmady, A. An Improved Binary Owl Feature Selection in the Context of Android Malware Detection. *Computers* **2022**, *11*, 173. https://doi.org/10.3390/computers11120173

Academic Editors: Phivos Mylonas, Katia Lida Kermanidis and Manolis Maragoudakis

Received: 7 October 2022
Accepted: 22 November 2022
Published: 30 November 2022

1. Introduction

Malware is the abbreviation of "Malicious Software", and refers to unwanted types of software regardless of its type, intent, or distribution method (e.g., virus, Trojan horse, worms, spyware, etc.) [1]. The Malware infects systems with the intent to gain access to sensitive information. Malware detection is the process of detecting malware on a host device, or the process of determining whether a program is malicious or benign. Malware continues to be a problematic security issue, especially in the software and cyber-security fields. Until the advent of smartphones, malware was only significantly found in computers. However, recent technological advancements have seen malware become conspicuous in smartphones and mobile devices that run on the Android platform. The fact that Android is the most popular and widely used platform for smartphones and mobile devices renders it an ideal target for malware attacks. Nonetheless, there have been significant efforts from the corresponding stakeholders aimed at establishing concrete measures for detecting malware in Android platforms.

Notably, the fact that the majority of applications utilized in an Android platform can be accessed from a common source provides leeway for publishers of malware content. As such, there is always the likelihood that a user with malicious intent can publish an application that is intended to act as malware. In the majority of cases, a potential malware application usually mimics a typical app though it is designed to achieve root control

functions. While some applications might ask for access privileges to features such as contacts and text messages, a malware app only requires to be installed in the system to achieve the same purpose [2].

Worse yet, when a malware application is authorized on an Android platform, the results are nothing less than catastrophic. There is the possibility of the malicious app gaining access to confidential and private information [3]. Yet, such an app can send secure information to unauthorized parties primarily for personal reasons, including revenge and blackmail [4]. Then again, a malicious application residing in an Android-based smartphone can be designed in such a way that it culminates in the deactivation of core functions.

With regard to deactivating or stopping core functions and processes, the problem is exacerbated by the fact that it is imperatively difficult for an unsuspecting user to detect it. In the majority of cases, Android-based malware takes the form of friendly or free applications aimed at boosting system performance [5]. A good case in point is an application that offers the premise to block unnecessary adverts. In such a scenario, a user is more inclined to install and allow the application to have root access to the system without the realization that it is actually a malware app.

The problem with having a malware-infected system is that it is imperatively difficult to control the level of functionality executed by the malicious app. A good example is a malware app that overtakes the mobile phone browser's interface and, in effect, displays unnecessary adverts. The adverts might not only be frustrating, but also have the capacity to utilize much-needed phone resources, including memory and storage [6]. The worst thing is that malware disguised as adverts are strategically designed to utilize smartphone resources with the objective of generating resources from illegitimate adverts.

With the continued evolution of malware applications designed for the Android platform, it is noteworthy that several concerted efforts have been put in place for a corresponding mitigation process. In particular, the majority of Original Equipment Manufacturers (OEMs) implement requirements aimed at enhancing the security of devices running on an Android platform [7]. Such an arrangement incorporates the utilization of specific applications designed to detect, prevent, and remove a potential malware application [8]. While some applications used to detect malware are installed by default, others require the specific intervention of the device's user [9]. However, most importantly, the objective is ensure that a typical Android system is equipped with the capacity to detect the presence of any malware content.

As per the assertions made by [10], the changing technological dynamics make it difficult to have in place a universal platform aimed at enhancing the security of an Android platform against malware. Notably, the traditional security systems based on the distributed computing models are increasingly being replaced by state-of-the-art cloud-based systems [1]. The effect is that an application published on one day can be mimicked and distributed the next day under the same identity, albeit integrated with malicious content. In particular, it does not guarantee that an application published for an Android-based device can avoid being infected with malware.

Understandably, the presence of malware in an Android-based mobile device will always remain a possibility. Even as OEMs and Android developers continue to implement robust security measures, so do the sophistication and advancement methods employed by malicious users. A common theme that emerges is that there is not a single mechanism or security approach that can be used to detect and remove malware on an Android platform [11]. As such, the subject of discussion is aimed at exploring innovative ways in which malware content can be detected on a device running on an Android platform. In this paper, a novel approach for detecting malware on an Android platform has been proposed. The proposed approach relies on using an ensemble learning classifier beside an Owl optimizer for choosing the most informative set of features that identify the behavior of a malicious application from a benign one.

The contributions of this paper are summarized as follows:

1. A comprehensive review of the latest research for detecting malicious applications on the Android platform is presented.
2. A detailed analysis of DREBIN dataset is provided.
3. A modified feature selection based on a binary owl optimizer is proposed.
4. A lightweight machine learning approach that is based on ensemble learning and binary Owl optimizer is proposed.

The rest of the paper is organized as follows; Section 2 reviews the latest up-to-date related works, Section 3 presents the proposed feature selection based on binary Owl algorithm. Section 4 discusses the DREBIN dataset, while Section 5 illustrates the used methodology for designing malware detection approach for Android platform. Section 6 discusses the achieved results. Section 8 presents the conclusion.

2. Related Works

Recently, Android malware attacks have received special attention from researchers. In this section, we present the latest research related to Android malware attacks, especially the one used in the DREBIN dataset for evaluation.

Some studies used ensemble learning for identifying Android malware applications such as in the work [12], where the authors proposed a framework called (PIndroid) that used ensemble learning to identify the Android malware application. The proposed framework was based on permission and intent. The achieved results show that the proposed framework provides an accuracy of 99.8% after applying it to 1745 real-world applications.

Furthermore, in [13], the authors proposed a scalable approach to detect malware using ensemble learning and Apache Spark. The ensemble learning used is based on two methods for calculating ranks and weights of the base classifiers. The optimal subset of features was selected based on weights for majority voting. The proposed ensemble method was evaluated using 198,350 files and compared with classical ensemble methods. Their proposed method produced a higher accuracy than the examined ensemble methods.

Other studies used machine learning techniques in general for developing an Android malware detection and attribution framework. For Android IoT devices, Kumar et al. in [14] designed a malware detection system that combines the benefits of machine learning methods and blockchain technology. The process begins with the extraction of malware data using clustering and classification techniques. Clustering methods iteratively remove non-informative features to differentiate between malware and benign applications, while the naive Bayes classifier is used to implement the classification algorithm.

Finally, the blockchain is used to store this data. The proposed framework provides better malware detection accuracy.

In [15], the proposed model was developed for Android malware detection. Their model involved the deployment of a Factorization Machine architecture and the extraction of characteristics from Android applications. They employed clean applications that were gathered from internet app stores, as well as the DREBIN and AMD datasets, for evaluation. According to the results, the suggested technique scored 100% for precision on the DREBIN dataset and 99.22% on the AMD dataset.

The study by [16] serves as an illustration of works that are concerned with deep learning for the development of Android malware detection frameworks. The authors employed deep learning techniques and unprocessed API method call sequences to create MalDozer, an automatic Android malware detection and attribution framework. This framework can be installed on servers, mobile, and Internet of Things (IoT) devices. The Malgenome, DREBIN, merged, and MalDozer datasets, which contain between 1 K and 33 K malware apps and 38 K benign apps, were used by the authors for evaluation. The suggested framework has an F-score ranging from 96 to 99 percent and can identify Android malware and their actual families.

Furthermore, in [17] the researchers offered a number of deep learning models that each focus on learning a particular data distribution for a certain class of malware in order to learn the complex data distribution of the malware dataset. They used Apache Spark's

parallel method to accelerate the model-building process. The authors created a dataset of 3.4 million benignware samples from multiple reliable sources and 2.2 million malware samples taken from six primary sources for evaluation. Results indicate that the suggested model performs better than more conventional methods such as Support Vector Machine, decision trees, and a single deep learning model, and is better equipped to handle the complex malware data distribution.

Later in 2021, Millar et al. developed a multi-view deep learning Android malware detection model [18]. They evaluated the suggested model using metrics from the DREBIN and AMD benchmarks, including weighted average detection rates, false-positive rates, and F1 score. Comparing the suggested model to state-of-the-art works, the weighted average detection rates show an improvement of up to 57%. For the DREBIN dataset and the AMD benchmark dataset, it provides weighted average detection rates of 91% and 81%, respectively.

Another Android apps framework developed by authors in [19] is based on both signature- and heuristic-based analyses. The proposed framework was tested on M0DROID Dataset and achieved 99.81%, 0.38 and 100% for accuracy, FPR and TPR, respectively.

Many research papers have surveyed malware detection techniques used in the literature such as the work by Odusami et al. in [20], where the authors conducted a survey of state-of-the-art malware detection techniques in order to detect gaps in previous methods for guiding the upcoming researchers to develop more effective metrics for the unknown malware in the proposed methods. The results of this survey indicated that machine learning presents the most promising direction based on the detection accuracy. They recommended that researchers focus on deep learning approaches using large datasets for achieving better detection accuracy.

By scanning studies from the past seven years, from 2014 to 2021, the authors of [21] conducted a machine learning techniques survey for Android malware detection. They summarized each study from five different perspectives: analysis type, dataset, feature extraction method, performance metrics and ML classification techniques. Based on their observations, they illustrated the current trends in research and future directions. Results reveal that most methodologies adopt a diverse set of fundamental factors, such as dataset metadata, analytical methods, and assessment criteria.

Rana et al. evaluated the performance of various machine learning models for Android malware detection. They used the DREBIN dataset for evaluation where results demonstrated that using machine learning classifiers for Android malware detection provides an accuracy of over 94% [22].

Recently, the authors in [8] focused on studying the effectiveness of supervised machine learning algorithms for malware detection in Android OS using six well-known classification techniques including (Bernoulli Naive Bayes, Random Forest, L1 and L2 regularization, Shallow neural network, and SVM) under different configurations, and recording the most significant set of features that each classifier depends on for building its decision. They used DREBIN dataset with 123,453 goodware versus 5560 malware applications associated to the 10 feature sets for evaluation. The results showed that employing approximately 1000 features from the original DREBIN feature set is sufficient for all the models to reach the highest classification accuracy.

In 2021, researchers in [23] proposed two attack models based on the data poisoning attack and the evasion attack for investigating several types of adversarial example attacks. They used Support Vector Machine (SVM) with different types of kernel functions to build the malware detector for the Android system. They used some real Android application datasets which contain "malware", and "benign" applications for evaluating the proposed detection approach capability to detect adversarial example attacks. Results show that evasion attacks are more difficult to detect via the malware detection system than data poisoning attacks.

In the same year, in [24], authors introduced a code deobfuscation technique with an Android malware detection system. In order to eliminate interference due to the size of

the application, they proposed interaction terms based on identifying feature interactions. For evaluation they used the DREBIN dataset, and results show that the proposed Android malware detection model achieves an accuracy of 99.55% and a 94.61% F-score.

That same year, the authors published [24] which combined an Android malware detection system with a code deobfuscation technique. They suggested interaction terms based on recognizing feature interactions to remove the interference brought on by the scale of the application. They employed the DREBIN dataset for evaluation, and the results indicate that the suggested Android malware detection model achieves a 99.55% accuracy rate and a 94.61% F-score.

Arif et al. focused on risk assessment using the fuzzy analytic hierarchy process (AHP) method in [25]. They proposed a mobile malware detection system. For evaluation, they used 10,000 samples from the "DREBIN", and "AndroZoo" datasets with permission-based features. This method raised mobile users' awareness concerning the importance of accepting any permission application request. The proposed detector has a high accuracy rate of 90.54%, and a risk assessment more than 80%.

Selvaganapathy et al. investigated the effects of evasion attacks on an anti-malware program that relies on the feed-forward deep neural network model developed by [26] by deploying a deep learning network in order to detect malware attacks. They used the DREBIN dataset for evaluation, with recall, specificity, false-positive rate, false-negative rate, accuracy, and F1 score as evaluation metrics. The findings demonstrated the importance of developing adaptive defenses to foster a secure learning model [27].

Table 1 presents a summary of the latest works in the literature on detecting malicious applications on the Android platform with the dataset, evaluation metrics, and the model used for evaluation. All the listed related works used a random subset of the Drebin dataset combined with applications from other sources, which makes it challenging to verify their results owing to the lack of dataset availability. Our proposed approach has been evaluated using the standard splits provided by the Drebin website. Moreover, the main issue of the Drebin dataset is the complexity of its structure. In this paper, a simplified version of the Drebin dataset is introduced. Furthermore, the proposed binary version of the Owl optimizer accelerates the convergence curve significantly, which affects the whole system's performance.

Table 1. Summary of related works.

Reference	Datasets	Evaluation Metrics	Model
[16]	Malgenome, DREBIN, merged, and MalDozer dataset.	F-score, false alarms, precision, recall, and runtime.	Neural networks.
[8]	DREBIN dataset.	F-score, precision, recall, true negative rate, accuracy, false positive rate, and Area Under Curve (AUC).	Naive Bayes, L1 and L2 regularization, Random Forest, Support Vector Machine (SVM), and Shallow neural network.
[23]	Original DREBIN, malware dataset, and a modified DREBIN dataset, The benign applications are obtained from Google play.	Precision, recall, and F-score.	SVM.
[17]	Constructed dataset with 2.2 million malware samples, and 3.4 million benignware samples.	Area under curve, true positive rate, and false positive rate.	Deep learning model.

Table 1. *Cont.*

Reference	Datasets	Evaluation Metrics	Model
[14]	Combined data set from the Chinese App Store and Google Play Store, containing 5560 malware samples and 6192 benign ones.	True positive rate, F-measure, false alarms, and classification accuracy.	Multi feature Naive Bayes algorithm.
[15]	DREBIN and Malware Dataset (AMD) datasets, in addition to clean applications which collected from online app stores.	Precision, recall, accuracy, F1, and False-Positive Rate.	New Factorization Machine (FM) model.
[12]	Dataset contains 1745 real world applications.	True positive rate, false alarms, accuracy, F-score, model build-up time, and area under curve.	Naive Bayes, Decision Table, Random Forest, Sequential Minimal Optimization, Decision tree, and Multi Lateral Perceptron (MLP).
[24]	DREBIN dataset and AndroZoo dataset.	Accuracy, precision, F-score, recall, and area under curve.	CatBoost, LightGBM, RandomForest, and LineraSVM.
[22]	DREBIN dataset.	Accuracy, precision, F-score, recall, area under curve, and false positive.	Decision Tree, Gradient Boosted , Random Forest, Extremely Randomized Tree, Neural Networks, k Nearest Neighbors, Discriminant Analysis, NB , Logistic Regression, Bagging, K Means, and SVM.
[27]	DREBIN dataset.	Recall, specificity, false-positive rate, false-negative rate, accuracy, and f1_score.	Feed-forward deep neural network model.

3. Owl Optimization Algorithm

The Owl algorithm is a recent nature-inspired optimization algorithm developed by [28]. The Owl optimizer mimics the actions of owls that hunt at night and use their hearing rather than their sight to locate prey. The Owl optimizer is a population-based algorithm that exploits the unique features of owls, such as their feathers, binocular vision, and binaural hearing, which are optimized for silent flight [29]. Owls bear a unique auditory system that causes the sound to reach one ear before the other. As a result, they have evolved a remarkable system for sound localization that is based on the intensity at the surface of a sphere surrounding the prey, as illustrated in Figure 1.

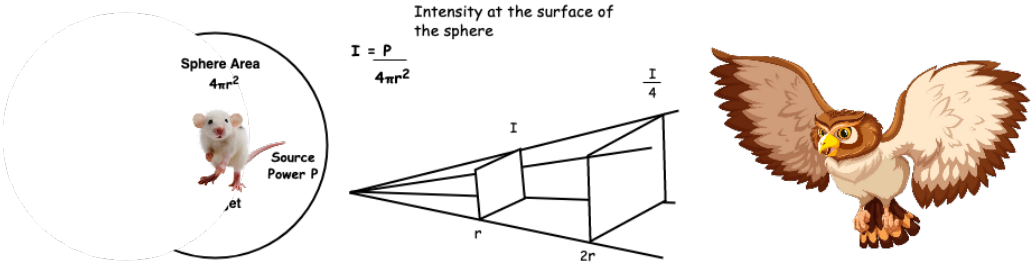

Figure 1. Owl Hunting mechanism.

The Owl search algorithm contains the following phases [30]:

- Initial Population: The forest's population of owls is represented by the initial set of random solutions. Each Owl is represented by a vector of length equal to the number of features in the search problem.
- Owl Evaluation: all Owls in the population are evaluated using a target evaluation or fitness function. The evaluation value indicates how well the solution fits the intensity of the information detected by the owl's ear. The best Owl is the one that receives the maximum intensity in the case of the maximization problem, while the worst Owl is the one that receives the minimum intensity. The intensity information for Owl O_i can be normalized using Equation (1).

$$Intensity(O_i) = \frac{f(O_i) - O_{worst}}{O_{best} - O_{worst}} \tag{1}$$

  where $f(O_i)$ is the fitness value for the Owl O_i. O_{best} and O_{worst} are the best and the worst solutions in terms of their fitness value in the current population.
- Update Owl Location: each Owl updates its position toward the prey. Here, the fittest owl with the highest fitness value is the prey. All the Owls update their locations according to the distance toward the prey as in Equations (2) and (3).

$$R_i = \|O_i, V\|_2 \tag{2}$$

  where V is the location of the prey achieved by the fittest Owl, and R_i is the distance between the target prey and Owl O_i.

$$IC_i = \frac{Intensity(O_i)}{{R_i}^2} + \alpha \tag{3}$$

  where α is a random value between [0, 0.5] that represents the noise and IC_i is the intensity changed toward the prey.

Based on the Intensity changed IC, the Owls will update their positions accordingly by Equation (4).

$${O_i}^{t+1} = \begin{cases} {O_i}^t + \beta * IC_i * \|\alpha V - {O_i}^t\| P_{vm} & < 0.5 \\ {O_i}^t - \beta * IC_i * \|\alpha V - {O_i}^t\| P_{vm} & \geq 0.5 \end{cases} \tag{4}$$

where β decrements linearly from 1.9 to 0 [29]. β first introduces significant changes and encourages the investigation of the search area, and P_{vm} is the likelihood of Owl movement.

Modified Binary Owl Optimizer

In order to hasten the search step, the Owl optimizer has been discretized into a binary Owl version. Each Owl O_i represents a solution, the O_i is a binary vector where its length is equal to the number of feature vectors. The feature vector is binary where 0 indicates the absence of the corresponding feature and 1 indicates the presence of the feature.

The first two phases of the modified binary Owl optimizer are the same as the original version. Whereas the updated Owl location phase is based on the similarity between the Owl O_i and O_{best} using Equation (5) since here we deal with binary vectors.

$$Distance(R_i) = \frac{\#\,of\ similarity\ features}{length\,of\ feature\ vector} \tag{5}$$

For instance, the number of similar features for "11001" and "11010" is 3, and the Distance $R_i = \frac{3}{5} = 0.6$. The Intensity change IC_i is calculated according to Equation (6).

$$IC_i = \frac{Intensity(O_i)}{R_i} + \alpha \tag{6}$$

where α is a random number between [0, 0.5].

Finally, all Owls' locations will be updated according to Equation (7).

$$O_i^{t+1} = \begin{cases} O_i^t & O_{best} < r \\ O_i^t & , O_i \geq r \end{cases} \quad (7)$$

where r is a uniform random number.

Note that the Owl population is stored in a set, if one or more Owls are the same, then a new randomly generated solution will join the set instead of the replicated one. In this way, a new random solution will always be provided the chance to exit the local optima.

4. Data Description

DREBIN dataset has been used to build the proposed malware detection approach for the Android platform. The DREBIN dataset consists of 123,453 benign applications from the Play Store and 5560 Malware applications' samples. The collected applications have been represented in a lightweight format by extracting a set of features from different sources [31]. The dataset set preparation process from the Android applications includes extracting features from the Android manifest and Dex code. Every Android application must have an Android manifest.xml file that describes the essential information about the app to the Android build tool and platform (e.g., permissions, intents and activity) to support application installation and later execution [32]. The Dex code file is a compiled code written for Android and Google Linux-based mobile phone platform. The disassembled code gives information about the API calls and strings contained in the application. The feature sets selected from the "manifest and the Dex code" file are listed in Table 2 [33].

Table 2. Extracted features from manifest and Dex code files.

Manifest File	
Feature Set	**Description**
Hardware Components	Contains the requested hardware components such as request access to the mobile camera.
Requested permissions	Permission granted by the user at the installation time, such as SEND_SMS Permission
App Components	There are four types of application components (services, activities, broadcast receivers, and content providers).
Filtered Intents	On Android, intra-process and inter-process communications are performed through intents. This is a passive data structure that works as a synchronous message, allowing sharing of information between different components and applications.
Dex Code	
Restricted API Calls	API calls are restricted by permissions, inside Dex code, this set searches for API calls that do not have an associated permission.
Used permission	The set of permissions that are requested in manifest and actually used based on the disassembled code. Sometimes they refer to it as real permission.
Suspicious API calls	API calls that are frequently used in malware applications and requested access to sensitive data or resources. There are four types of these API calls; API calls to access sensitive data, API calls for communicating over the internet, API calls for sending and receiving SMS messages and finally API calls used frequently for obfuscation.
Network Addresses	This includes any IP address, URL, or host-name found in the disassembled code.

DREBIN Dataset Structure

The DREBIN dataset contains 5560 Android applications from 179 different malware families in addition to 123,453 benign Android applications. The data were collected in the time period between August 2010 and October 2012 [31]. The main challenge with DREBIN is the scattered record's structure. Multiple files should be mapped to obtain the whole

record with the label class. Thus, extensive pre-processing should be performed before building a model. On the official site, the malware applications were grouped into five separate links, where each link contained 1000 applications [32].

The dataset is structured as follows:

- Feature vector folder: this folder contains the feature vector for the applications, where each application's feature vector is saved in a separate file. Each file has been titled by the application signature in (SHA256) format. Moreover, in the feature vector folder, there are 129,013 files for all benign and malware applications. The feature vector files contain all features selected from the application ("android manifest and Dex code") including the requested permissions, the "used permission", URLs, API calls, etc.
- Family Labels file: this file lists all the signatures (SHA256 hash) of all applications with the corresponding family label (benign, malware family).
- Dataset Splits folder: this folder contains 10 sub-folders, each sub-folder contains training, testing, and validating files. These files only contain a list of signatures for the applications (SHA256 hash).

Figure 2 illustrates the dataset structure.

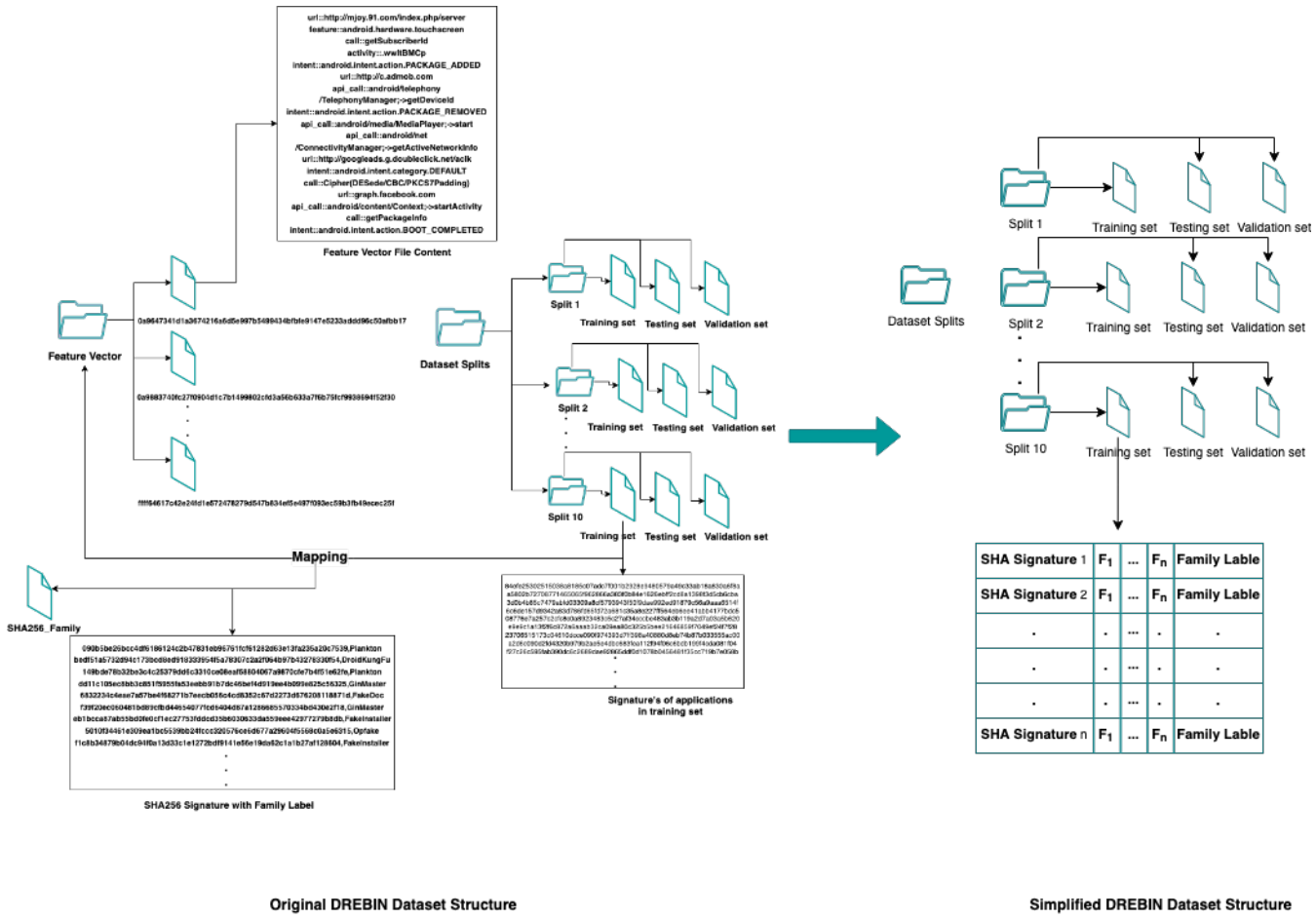

Figure 2. DREBIN Dataset Structure.

It is noted that the extracted feature vector files have 545,356 sparse features which contain numerous typos and irrelevant features (i.e., requested permission that was never used, URLs for images, etc.). Moreover, the dataset requires an extensive mapping to concatenate the application feature vectors with their corresponded signature, and family label. Furthermore, the same work must be performed for all data splits. In this paper, an enhanced simplified version of the DREBIN dataset has been introduced. The enhanced version will help researchers to use the DREBIN dataset for evaluation purposes.

5. Malware Detection Model Development

5.1. An Enhanced Simplified Version of the DREBIN Dataset

As mentioned earlier, DREBIN dataset has many challenges with respect to its structure. In this section, a new enhanced and simplified version of DREBIN dataset is introduced. The first challenge presented by the DREBIN dataset is the variety of feature vector files. There are no standard features and you can also find the same permission request with different syntax including typos. Moreover, the number of features is huge since the feature vector includes many irrelevant features including the following:

- All URLs are removed from the feature vector, since every application has a unique URL referring to images or an external link.
- All features in the requested permission set that are never used and do not affect the functionality of the application.
- The requested permissions with typos. After removing the irrelevant features mentioned above, only distinct features from all files are combined to form the standard feature vector.

The second challenge presented by the DREBIN dataset is the scattered files. The required information for each application should be collected from three locations (feature vector file, the name of the feature vector, and family label file). A mapping process should be conducted to correlate the required information. Moreover, training and testing files only contain the signature of the applications. The simplified version of the DREBIN dataset prepared is to include all information in a new single structure. In the simplified version, the application signatures and families from the "SHA family" file are mapped and concatenated with the content of feature vector files, where each row has the "SHA256" signature, standard feature vector and the family as shown in Figure 3.

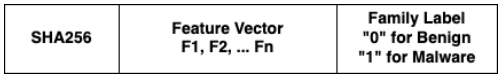

Figure 3. The combined file structure.

Each row content is filled by "0" or "1" indicating whether the feature is used in the application or not. Moreover, the class label "malware families and benign" is transferred to binary classes 0 and 1, where the 0 indicates a benign application, and 1 indicates a malware application. This process has been performed for the 10 standard data splits listed on the DREBIN official site. This will make it easier for researchers to use the DREBIN dataset and conduct a fair comparison with other approaches.

5.2. Feature Selection and Model Development

The DREBIN dataset contains 545,356 sparse features. After removing the requested permission feature set and discarding the URL features, the remaining number of features was reduced to 476 distinct features [33].

For the enhanced version of the DREBIN dataset, feature selection was performed using the modified binary Owl optimizer. Only 476 distinct features have been used as input for the modified binary Owl optimizer. The first population has been generated randomly, where each solution "Owl" O_i is a binary vector with length 476. The "0" indicates the feature's absence in the solution, where "1" indicates presence of the feature in the solution. All the Owls will be evaluated in terms of the fitness function presented in Equation (8) after building the model [34].

$$Fitness\ Function\ = W_a * tpr + W_b * \frac{1}{fpr} + W_c * \frac{F_N}{F_S} \tag{8}$$

where W_a, W_b and W_c are the weights for the True-Positive Rate (TPR), False-Positive Rate (FPR) and the ratio of selected features, respectively. F_N refers to the feature vector

length, and F_S represents the number of selected features in the solution (count "1's" in the feature vector).

The model development includes training the set of features using Random Forest Classifier. The model will then be evaluated using the testing set in terms of the selected performance metrics.

An ensemble learning technique called Random Forest is applied to classification or regression tasks. Several decision trees are built throughout the training process of the random forest. The classification will then be determined by majority vote of all created trees during the test phase [35]. The proposed malware approach is presented in Algorithm 1.

Algorithm 1 Malware Detection Approach Pseudocode.

Input: Population_Size P_s, Number of Iterations N_i, Fitness Function Weights W_a, W_b, W_c.
Output: Global Solution O_{best}

Initialize O_i for each Owl randomly.
Evaluate_Owls $(O_1, O_2, \ldots, O_{P_s})$ by their fitness values.
$O_{best} = Fittest\ Owl\ (minimum\ fitness)$
while $(n_i >= 1)$ **do**
  Update Intensity Change for each Owl by Equation (6)
  Find the Owl distance towards the prey by Equation (5)
  Update Owl location toward the best Owl by Equation (7)
  Train the model for each O_i
  Evaluate Owls $(O_1, O_2, \ldots, O_{P_s})$ by their fitness values using Equation (8).
  Update O_{best}
end while

Evaluate_Owls$(O_1, O_2, \ldots, O_{P_s})$ **:**
forEach O_i *in Owls*
  forEach x in O_i
   Select= [ ]
   if $x_i > 0.5$
   then Select.append(x_i)
  end forEach
  prediction=RandomForest.fit(train_set[:,Select], target_train).predict(test_set)
  Calculate Fitness Value using Equation (8)
  end forEach

6. Experiments and Results

6.1. Experimental Setup

All experiments in this section were conducted using Windows 10, a 64-bit operating system, an Intel Core i7, and 16 GB of RAM. The RF-Owl technique has also been implemented using the Anaconda Python framework version 5.1. Note that an average of 30 runs was used to obtain the final findings (Table 3).

6.2. Performance Metrics

There are several performance metrics used by researchers to evaluate the proposed approaches [36–38]. In this paper, we will evaluate the proposed approach based on "accuracy, F-score, precision, recall, and false-positive rate". Furthermore, the convergence of the modified binary Owl optimizer will be evaluated.

The following are the definitions and the Equations of the selected performance metrics:

- Accuracy: Measures the number of correctly classified applications to the total number of classifications.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

- F-score: F-measure is a harmony measure that take into consideration both the recall and precision.

$$F - score = \frac{2 * TP}{2 * TP + FP + FN} \tag{10}$$

- Precision: Measures the number of correctly predicted applications as malware to the all applications predicted as a malware.

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

- Recall: Measures the number of applications that are correctly predicted as malware to the number of actual malware applications.

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

- False-Positive Rate: Measures the rate of benign applications erroneously classified as malware.

$$False\ Positive\ Rate = \frac{FP}{FP + TN} \tag{13}$$

Table 3. The experimental setup.

Owl Parameters	
Parameter	**Value**
α	A random number between $[0, 0.5]$
β	A linear decreasing number from 1.9 to 0
r	Uniform random number
Number of Iterations	300
Population size (Np)	100
Fitness Function	
α	0.48
β	0.48
γ	0.04

All the above metrics are calculated according to the following confusion matrix parameters [39]:

- **True Positive (TP)**: The quantity of malware application instances that were accurately classified.
- **True Negative (TN)**: The quantity of benign applications that were accurately classified.
- **False Positive (FP)**: The quantity of benign applications that were erroneously classified as malware.
- **False Negative (FN)**: The quantity of malware application instances that were incorrectly classified as benign.

6.3. Results

The evaluation of the proposed approach is divided into two phases; the first phase evaluates the proposed approach using the 10 standard splits of the DREBIN dataset. Table 4 presents the results of all sample splits from the DREBIN dataset. The results of each data sample split are examined based on "precision, recall, FPR, accuracy, F-score, and number of features".

The sample split #2 achieved the best precision and FPR among all other splits, while split #9 achieved the best recall results. Regarding the accuracy and F-score, sample split #1 score the best results. Furthermore, the sample splits were compared in terms of the number of selected features. Sample split #10 has the least number of selected feature,

namely 205. Note that all the sample splits have a number of selected features between 205 and 248.

The proposed method is evaluated versus the examined selected approaches from the literature using an average of all samples split's results.

The performance of the proposed modified binary Owl optimizer is evaluated in terms of the convergence curve compared with the standard continuous version. Figure 4 illustrates the convergence curve of the proposed binary Owl optimizer for feature selection and the standard continuous Owl optimizer. As the figure shows, the modified binary OWL optimizer accelerates the convergence of the algorithm significantly. The fitness value of the binary version improved with each iteration and reached the maximum value at approximately 1000 iterations.

Table 4. All DREBIN Samples Split Results.

Split Sample	Precision	Recall	FPR	Accuracy	F-Score	# of Features
Split #1	0.9930	0.9964	0.1524	0.9897	0.8780	205
Split #2	0.9933	0.9948	0.1484	0.9884	0.8638	221
Split #3	0.9918	0.9951	0.1709	0.9874	0.8504	208
Split #4	0.9924	0.9963	0.1719	0.9890	0.8642	221
Split #5	0.9925	0.9948	0.1683	0.9878	0.8542	225
Split #6	0.9922	0.9963	0.1654	0.9891	0.8730	224
Split #7	0.9927	0.9960	0.1576	0.9890	0.8707	228
Split #8	0.9921	0.9958	0.1729	0.9882	0.8604	216
Split #9	0.9927	0.9965	0.1626	0.9896	0.8743	231
Split #10	0.9928	0.9941	0.1615	0.9873	0.8500	201
Average	0.9924	0.9956	0.1601	0.9884	0.8634	-

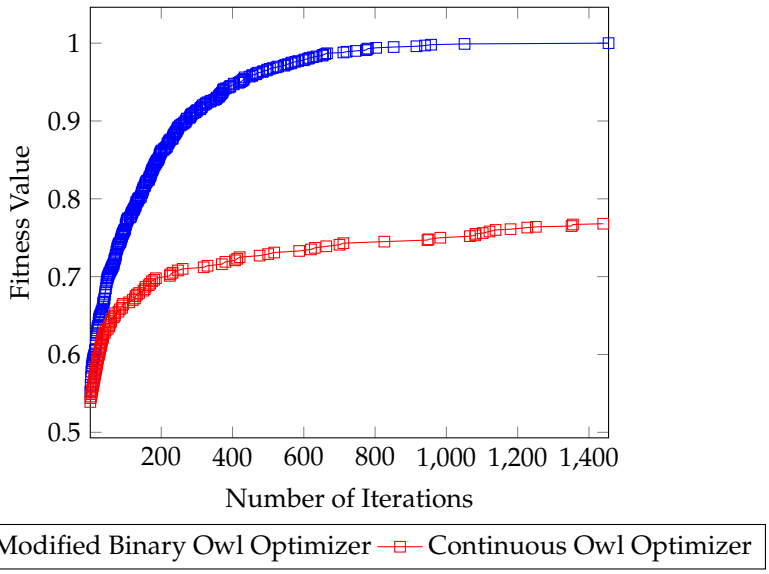

Figure 4. Convergence curve of the modified binary Owl optimizer vs. the continuous Owl optimizer.

Figures 5–9 show comparisons of the evaluation findings for the proposed RF-Owl approach to the approaches introduced by [8,23,24,27,40] in terms of precision, recall, false-positive rate, accuracy and F-score, respectively.

Figure 5 illustrates the precision results for the proposed RF-Owl against six examined approaches from the literature. The findings indicate that, when compared to the others,

the proposed RF-Owl approach exhibited the highest precision. Meanwhile, the BNB approach proposed by [8] exhibited the worst precision score with only 0.469.

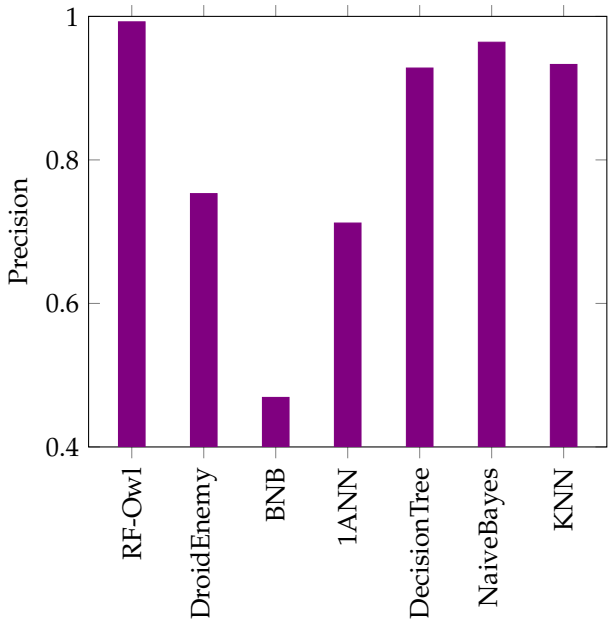

Figure 5. Precision results for the 7 examined algorithms using the DREBIN dataset.

As observed from Figure 6, the RF-Owl has the best recall results compared to the six examined approaches. The KNN and Decision_Tree Approaches came in second place, while the DriodEnemy produced the worst results in terms of recall.

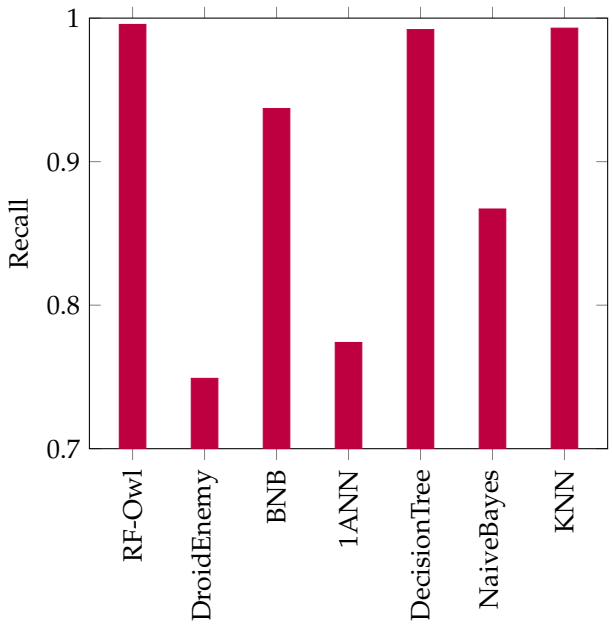

Figure 6. Recall results for the 7 examined algorithms using the DREBIN dataset.

Figure 7 illustrates the accuracy results for the four examined approaches. Regarding accuracy, the RF-Owl, Deep Neural Network (DNN) and the approach that uses the code deobfuscation have approximately the same score. Meanwhile, the DroidEnemy bears the worst accuracy score with only 0.747.

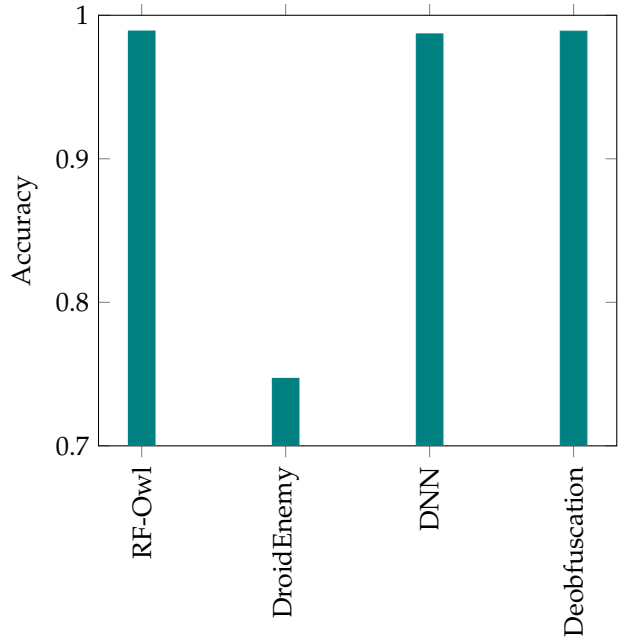

Figure 7. Accuracy results for the 4 examined algorithms using the DREBIN dataset.

Concerning the FPR, the results of FPR for five examined approaches are illustrated in Figure 8. The FPR should be reduced. The proposed RF-Owl approach has the lowest FPR with only 0.1601. While the approach that used the DNN has the worst FPR score with 0.28. The Decision_Tree, Naive_Bayes and KNN approaches have approximately the same FPR results.

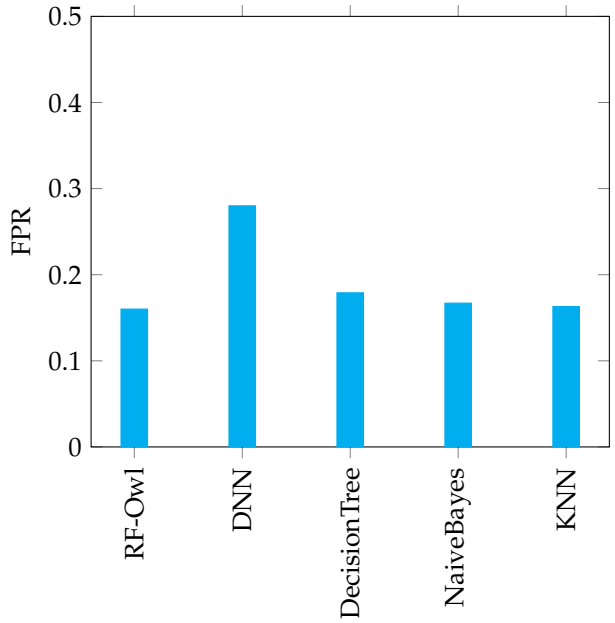

Figure 8. FPR results for the 5 examined algorithms using the DREBIN dataset.

Finally, the F-score has been used to evaluate the examined approaches. Figure 9 presents the results of the F-score for five examined approaches. The RF-Owl approach achieved the best F-score result with 0.8634. The approach of code Deobfuscation came in second place with 0.747, while the BNB approach had the worst F-score result with 0.625.

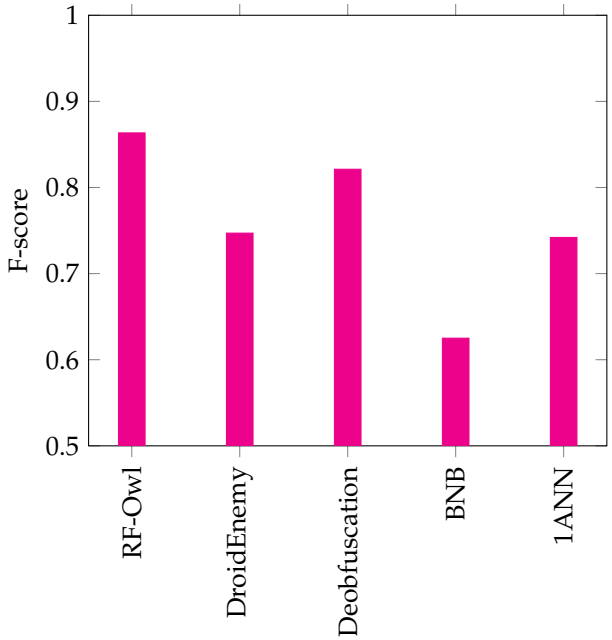

Figure 9. F-score results for the 5 examined algorithms using the DREBIN dataset.

7. Discussion

Table 5 summarizes the precision, recall, accuracy, FPR, and F-score results that have been generated from comparing the proposed approach with some state-of-the-art works. The "-" sign indicate that evaluation measure has not been reported in the corresponding reference.

Table 5. Results summary for all examined approaches compared with the proposed approach.

Reference	Approach/Technique	Precision	Recall	FPR	Accuracy	F-Score
[23]	DroidEnemy	0.749	0.9964	-	0.747	0.752
[24]	Deobfuscation	-	-	-	0.9889	0.8212
[8]	BNB	0.469	0.937	-	-	0.625
[8]	1ANN	0.712	0.774	-	-	0.742
[40]	DecisionTree	0.928	0.992	0.179	-	-
[40]	NaiveBayes	0.9920	0.867	0.167	-	-
[40]	KNN	0.933	0.993	0.163	-	-
[27]	DNN	-	-	0.28	0.987	-
Proposed Approach	RF-Owl	0.9924	0.9956	0.1673	0.9881	0.8634

Table 6 illustrates the comparison between our proposed approach (RF-OWL) and other related approaches that use different metaheuristic algorithms; Particle Swarm Optimization (PSO) in [41], Sophisticated Extrinsic Random-based Ensemble (ERBE) in [42], and Intelligent Water Drop (IWD) in [43]. It can be noticed that RF-OWL outperforms the listed proposals in terms of accuracy, precision and recall. Thus, our approach acheived better performance in terms of precision and recall compared with the other methods. The approach in [43] achieves higher accuracy compared with our approach due to the fact that, but still the proposed approach demonstrates better performance in terms of precision and recall.

Table 6. Results summary for related approaches that are based on different metaheuristic approaches compared with the proposed approach.

Reference	Featuer Selection/Classifier	Precision	Recall	Accuracy
[41]	PDL-PSO	0.988	0.98	0.977
[42]	ERBE	0.936	0.940	0.938
[43]	IWD	0.9535	0.9668	0.9912
Proposed Approach	RF-Owl	0.9924	0.9956	0.9881

Table 7 illustrates a comparison between the proposed approach and different ensemble approaches such as Decision Tree (DT), Random Forest (RF), Extremely Randomized Tree (ERT), Support Vector Machine (SVM), Logistic Regression (LR), and Gradient Boosting (GB). It can be noticed that in [44], a bagging (RF) has been used as an ensemble approach and is the same ensemble approach used in this paper. Our proposed approach achieved better performance in terms of precision, recall and accuracy compared with other approaches, since the whole approach depends on both the feature selection used and the classifier.

Table 7. Results summary for related approaches that are based on different ensemble approaches compared with the proposed approach.

Reference	Ensemble Approach	Precision	Recall	Accuracy
[44]	Stacking (DT, SVM, LR)	0.92	0.91	0.9158
[44]	Bagging (RF)	0.91	0.92	0.9173
[42]	ERBE	0.936	0.940	0.938
[44]	Bagging (ERT)	0.92	0.91	0.9129
[44]	Boosting (GB)	0.86	0.86	0.8611
Proposed Approach	Bagging (RF)	0.9924	0.9956	0.9881

8. Conclusions

With more than four-million Android applications available online, they have become a popular target for cybercriminals. Several malware detection approaches based on machine learning techniques have been proposed to ensure the safety of Android devices. Many research works address the problem of feature selection; however, the number and the type of irrelevant features also affect the performance and accuracy of the system. In this paper, an ensemble learning method with a "modified binary Owl optimizer" is presented for malware detection. The proposed malware detection approach uses a new "modified Owl optimizer" to manage feature selection. The DREBIN dataset was used for evaluation purposes. Many irrelevant features were initially discarded prior to the feature selection process. The features were reduced from 545,356 to 467. The prepared data were then inserted into the proposed model that employs a Random Forest classifier with the modified binary Owl optimizer. The proposed approach outperforms the examined approaches from the literature in terms of accuracy, precision, and recall.

Author Contributions: H.A.: Conceptualization, Methodology, Software, Validation, Investigation, Writing—original draft, Writing—review and editing. A.A.-A.: Writing—original draft, Writing—review and editing. O.A.: Writing—review and editing. E.A.: Writing—review and editing. A.A.: Software, Validation, Investigation. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Almin, S.B.; Chatterjee, M. A novel approach to detect android malware. *Procedia Comput. Sci.* **2015**, *45*, 407–417. [CrossRef]
2. Talal, M.; Zaidan, A.; Zaidan, B.; Albahri, O.S.; Alsalem, M.; Albahri, A.S.; Alamoodi, A.; Kiah, M.L.M.; Jumaah, F.; Alaa, M. Comprehensive review and analysis of anti-malware apps for smartphones. *Telecommun. Syst.* **2019**, *72*, 285–337. [CrossRef]
3. Xu, K. Advanced Malware Detection for Android Platform. Ph.D. Thesis, Singapore Management University, Singapore, 2018.
4. Li, W.; Ge, J.; Dai, G. Detecting malware for android platform: An svm-based approach. In Proceedings of the 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, New York, NY, USA, 3–5 November 2015; pp. 464–469.
5. Amro, B. Malware detection techniques for mobile devices. *Int. J. Mob. Netw. Commun. Telemat. (IJMNCT)* **2017**, *7*. [CrossRef]
6. Truong, H.T.T.; Lagerspetz, E.; Nurmi, P.; Oliner, A.J.; Tarkoma, S.; Asokan, N.; Bhattacharya, S. The company you keep: Mobile malware infection rates and inexpensive risk indicators. In Proceedings of the 23rd International Conference on World Wide Web, Seoul, Republic of Korea, 7–1 April 2014; pp. 39–50.
7. Shabtai, A. Malware detection on mobile devices. In Proceedings of the 2010 Eleventh International Conference on Mobile Data Management, Kansas City, MO, USA, 23–26 May 2010; pp. 289–290.
8. Syrris, V.; Geneiatakis, D. On machine learning effectiveness for malware detection in Android OS using static analysis data. *J. Inf. Secur. Appl.* **2021**, *59*, 102794. [CrossRef]
9. Feizollah, A.; Anuar, N.B.; Salleh, R.; Wahab, A.W.A. A review on feature selection in mobile malware detection. *Digit. Investig.* **2015**, *13*, 22–37. [CrossRef]
10. Vishnoi, A.; Mishra, P.; Negi, C.; Peddoju, S.K. Android Malware Detection Techniques in Traditional and Cloud Computing Platforms: A State-of-the-Art Survey. *Int. J. Cloud Appl. Comput. (IJCAC)* **2021**, *11*, 113–135. [CrossRef]
11. Kouliaridis, V.; Barmpatsalou, K.; Kambourakis, G.; Chen, S. A survey on mobile malware detection techniques. *IEICE Trans. Inf. Syst.* **2020**, *103*, 204–211. [CrossRef]
12. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [CrossRef]
13. Gupta, D.; Rani, R. Improving malware detection using big data and ensemble learning. *Comput. Electr. Eng.* **2020**, *86*, 106729. [CrossRef]
14. Kumar, R.; Zhang, X.; Wang, W.; Khan, R.U.; Kumar, J.; Sharif, A. A multimodal malware detection technique for Android IoT devices using various features. *IEEE Access* **2019**, *7*, 64411–64430. [CrossRef]
15. Li, C.; Mills, K.; Niu, D.; Zhu, R.; Zhang, H.; Kinawi, H. Android malware detection based on factorization machine. *IEEE Access* **2019**, *7*, 184008–184019. [CrossRef]
16. Karbab, E.B.; Debbabi, M.; Derhab, A.; Mouheb, D. MalDozer: Automatic framework for android malware detection using deep learning. *Digit. Investig.* **2018**, *24*, S48–S59. [CrossRef]
17. Zhong, W.; Gu, F. A multi-level deep learning system for malware detection. *Expert Syst. Appl.* **2019**, *133*, 151–162. [CrossRef]
18. Millar, S.; McLaughlin, N.; del Rincon, J.M.; Miller, P. Multi-view deep learning for zero-day Android malware detection. *J. Inf. Secur. Appl.* **2021**, *58*, 102718. [CrossRef]
19. Rehman, Z.U.; Khan, S.N.; Muhammad, K.; Lee, J.W.; Lv, Z.; Baik, S.W.; Shah, P.A.; Awan, K.; Mehmood, I. Machine learning-assisted signature and heuristic-based detection of malwares in Android devices. *Comput. Electr. Eng.* **2018**, *69*, 828–841. [CrossRef]
20. Odusami, M.; Abayomi-Alli, O.; Misra, S.; Shobayo, O.; Damasevicius, R.; Maskeliunas, R. Android malware detection: A survey. In *Communications in Computer and Information Science, Proceedings of the International Conference on Applied Informatics, Bogotá, Colombia, 1–3 November 2018*; Springer: Cham, Switzerland, 2018; pp. 255–266.
21. Kouliaridis, V.; Kambourakis, G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. *Information* **2021**, *12*, 185. [CrossRef]
22. Rana, M.S.; Gudla, C.; Sung, A.H. Evaluating machine learning models for Android malware detection: A comparison study. In Proceedings of the 2018 VII International Conference on Network, Communication and Computing, Taipei City, Taiwan, 14–16 December 2018; pp. 17–21.
23. Bala, N.; Ahmar, A.; Li, W.; Tovar, F.; Battu, A.; Bambarkar, P. DroidEnemy: Battling adversarial example attacks for Android malware detection. *Digit. Commun. Netw.* 2021, *in press*. [CrossRef]
24. Chen, Y.C.; Chen, H.Y.; Takahashi, T.; Sun, B.; Lin, T.N. Impact of Code Deobfuscation and Feature Interaction in Android Malware Detection. *IEEE Access* **2021**, *9*, 123208–123219. [CrossRef]
25. Arif, J.M.; Ab Razak, M.F.; Mat, S.R.T.; Awang, S.; Ismail, N.S.N.; Firdaus, A. Android mobile malware detection using fuzzy AHP. *J. Inf. Secur. Appl.* **2021**, *61*, 102929.

26. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbruecken, Germany, 21–24 March 2016; pp. 372–387.
27. Selvaganapathy, S.; Sadasivam, S. Anti-malware engines under adversarial attacks. *Int. J. Comput. Appl.* **2021**, *44*, 1–14. [CrossRef]
28. Jain, M.; Maurya, S.; Rani, A.; Singh, V. Owl search algorithm: A novel nature-inspired heuristic paradigm for global optimization. *J. Intell. Fuzzy Syst.* **2018**, *34*, 1573–1582. [CrossRef]
29. Lai, G.; Li, L.; Zeng, Q.; Yousefi, N. Developed owl search algorithm for parameter estimation of PEMFCs. *Int. J. Ambient. Energy* **2020**, *43*, 1–10. . [CrossRef]
30. El-Ashmawi, W.H.; Abd Elminaam, D.S.; Nabil, A.M.; Eldesouky, E. A chaotic owl search algorithm based bilateral negotiation model. *Ain Shams Eng. J.* **2020**, *11*, 1163–1178. [CrossRef]
31. Daniel, A.; Michael, S.; Hugo, G.; Konrad, R. Drebin: Efficient and explainable detection of android malware in your pocket. In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 23–26 February 2014.
32. Michael, S.; Florian, E.; Thomas, S.; Felix, C.F.; Hoffmann, J. Mobilesandbox: Looking deeper into android applications. In Proceedings of the 28th International ACM Symposium on Applied Computing (SAC), Coimbra, Portugal, 18–22 March 2013.
33. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. *Ndss* **2014**, *14*, 23–26.
34. Alazzam, H.; Sharieh, A.; Sabri, K.E. A feature selection algorithm for intrusion detection system based on pigeon inspired optimizer. *Expert Syst. Appl.* **2020**, *148*, 113249. [CrossRef]
35. Alazzam, H.; Alsmady, A.; Shorman, A.A. Supervised detection of IoT botnet attacks. In Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems, Dubai, United Arab Emirates, 2–5 December 2019; pp. 1–6.
36. Stiborek, J.; Pevnỳ, T.; Rehák, M. Multiple instance learning for malware classification. *Expert Syst. Appl.* **2018**, *93*, 346–357. [CrossRef]
37. Surendran, R.; Thomas, T.; Emmanuel, S. Gsdroid: Graph signal based compact feature representation for android malware detection. *Expert Syst. Appl.* **2020**, *159*, 113581. [CrossRef]
38. Fan, Y.; Ye, Y.; Chen, L. Malicious sequential pattern mining for automatic malware detection. *Expert Syst. Appl.* **2016**, *52*, 16–25. [CrossRef]
39. Chandak, T.; Shukla, S.; Wadhvani, R. An analysis of "A feature reduced intrusion detection system using ANN classifier" by Akashdeep et al. expert systems with applications (2017). *Expert Syst. Appl.* **2019**, *130*, 79–83. [CrossRef]
40. Yusof, M.; Saudi, M.M.; Ridzuan, F. A new mobile botnet classification based on permission and API calls. In Proceedings of the 2017 Seventh International Conference on Emerging Security Technologies (EST), Canterbury, UK, 6–8 September 2017; pp. 122–127.
41. Al-Andoli, M.N.; Tan, S.C.; Sim, K.S.; Lim, C.P.; Goh, P.Y. Parallel Deep Learning with a hybrid BP-PSO framework for feature extraction and malware classification. *Appl. Soft Comput.* **2022**, *131*, 109756. [CrossRef]
42. Potha, N.; Kouliaridis, V.; Kambourakis, G. An extrinsic random-based ensemble approach for android malware detection. *Connect. Sci.* **2021**, *33*, 1077–1093. [CrossRef]
43. Sharma, R.M.; Agrawal, C.P. MH-DLdroid: A Meta-Heuristic and Deep Learning-Based Hybrid Approach for Android Malware Detection. *Int. J. Intell. Eng. Syst.* **2022**, *15*, 425–435.
44. Rana, M.S.; Sung, A.H. Evaluation of advanced ensemble learning techniques for Android malware detection. *Vietnam J. Comput. Sci.* **2020**, *7*, 145–159. [CrossRef]