MalEfficient10%: A Novel Feature Reduction Approach for Android Malware Detection

Hemant Rathore(✉), Ajay Kharat, Rashmi T, Adithya Manickavasakam, Sanjay K. Sahay, and Mohit Sewak

Department of CS and IS, BITS Pilani, K K Birla Goa Campus, Goa, India
{hemantr,h20190011,h20210020,
f20190181,ssahay,p20150023}@goa.bits-pilani.ac.in

Abstract. The Android OS has recently gained immense popularity among smartphone users. It has also attracted many malware developers, leading to countless malicious applications in the ecosystem. Many recent reports suggest that the conventional signature-based malware detection technique fails to protect android smartphones from new and sophisticated malware attacks. Therefore, researchers are exploring machine learning-based malware detection systems that can successfully discriminate between malware and benign applications: *effectively* and *efficiently*. Existing literature suggests that many machine learning-based models use large feature sets for malware detection. However, classification models based on a large number of features are computationally expensive, time-consuming, and have poor generalizability. Therefore, this paper proposes a reliable feature reduction approach to select the most prominent features for effective and efficient malware detection. The proposed approach is tested on two different datasets, three distinct features, and twenty-six unique classifiers. The twenty-six baseline malware detection models based on 724 features and thirteen classification algorithms achieved an average accuracy and average AUC of 94.73% and 94.49%, respectively. Later we performed feature reduction that works with mutually exclusive and merged feature spaces of android permissions, intents, and opcodes. The proposed feature reduction approach reduced the number of features from 724 to 72 (10% of the original features). We also list the reduced set of 72 features comprising android permissions, intent, and opcode used for malware detection. The reduced features based twenty-six malware detection models achieved an average accuracy and average AUC of 93.12% and 92.97%, respectively. The feature reduction leads to less than 2% reduction in average accuracy and AUC. However, it leads to 85.25% and 91.45% reduction in average test and average training time for twenty-six android malware detection models. Therefore, the feature reduction leads to a minute reduction in the effectiveness but results in massively efficient (w.r.t time) malware detection models.

Keywords: Android · Feature Selection · Machine Learning · Malware Detection · Static Analysis

W. Wang and J. Wu (Eds.): BROADNETS 2023, LNICST 511, pp. 72–92, 2023.
https://doi.org/10.1007/978-3-031-40467-2_5

1 Introduction

Smartphones have been an inseparable part of our everyday life and are currently used by more than 50% of the world's population. A recent report by International Data Corporation (IDC) suggests that android holds 72.2% market share of the global mobile phone operating system, and it is expected to reach 80% by the end of 2023 [3]. These smartphones store a large amount of user personal and business data, which is an attractive target for malware developers. Therefore, android smartphones have seen an immense upsurge of malicious applications in the ecosystem. According to AV-Test, more than 33 million malware has been identified in the android ecosystem as of October 2022 [2]. A recent McAfee Mobile Report (2020) reveals that at least 40% of the world's mobile phones are vulnerable to malware attacks, and a new malware is launched every 10 seconds [1]. Therefore, there is an urgent requirement for proactively detecting malware on the android ecosystem.

Currently, conventional malware detection systems are based on signature and heuristic based malware detection techniques [15,26]. A test application is checked by comparing its signature with known malware signatures in the anti-malware database. To deal with the detection of unknown malware, researchers came up with two broad strategies: (a) static analysis and (b) dynamic analysis. Static analysis dissects and examines applications without executing them. It checks the internal working of the Android Application Package (APK) by reverse engineering. Static features such as Permission (P), Intent (I), and Opcode (O) carry a wealth of information to detect malicious behavior. *Android Permissions* are needed to access critical and sensitive resources [7]. On the other hand, *Android Intents* helps in communication between multiple android components and *Android Opcodes* detail the internal bytecode-level operations of the android OS. Unlike static analysis, dynamic analysis involves running the applications in a sandbox which is a resource-intensive and time-consuming process.

Malware detection based on machine learning consists of two phases: feature extraction and classification [11,26]. In the *feature extraction* phase, features like Permissions, Intents, and Opcodes are extracted from the android applications. In the *classification* phase, machine learning algorithms are used to construct malware detection models. Arp et al. extracted features like permissions, intents, hardware components, network addresses, app components, and API calls from android applications [4]. They constructed a android malware detection model using support vector machine with a detection rate of 94%. However, they used 545,000 features and thus suffered from the curse of dimensionality. Li et al. proposed a novel Factorization Machine-based malware detection system using permission, intent, application components, and hardware features. Their approach used 93,324 features with a resultant accuracy of 99.73% [6].

If the number of features to be analysed is high and the training data is not increased proportionally, the malware detection models tend to overfit. This leads to less-than-expected performance in the real world while also increasing the training and test time. Hence, feature reduction is crucial for static malware

detection. This paper proposes a malware detection system that uses limited android permissions, intents, and opcodes (only 10% of the original feature sets) while maintaining high performance. We further combine the original feature sets and find the best 10% features in the merged set. The reduced feature set is evaluated using thirteen malware classifiers from four different categories, and it achieves promising results. We observed that training and testing time reduced drastically after feature reduction. In summary, we made the following contributions with this work:

1. We proposed a novel feature reduction approach for android malware detection, which reduced each feature set to 10% of its original size while maintaining high performance. The number of Permission features were reduced from 195 to 19, Intent features from 273 to 27, Opcode features from 256 to 25, and Total features from 724 to 72.
2. The twenty-six baseline malware detection models without any feature reduction (724 features) achieved an average accuracy of 94.73% and an average AUC of 94.49%. On the other hand, twenty-six baseline malware detection models with feature reduction (72 features) accomplish an average accuracy of 93.12% and an average AUC of 92.97%. The feature reduction from 724 to 72 resulted in a minute average accuracy and AUC reduction of 1.69% and 1.60%, respectively, for twenty-six malware detection models.
3. The feature reduction (from 724 to 72) drastically improved the average training time by 85.26% and the average testing time by 91.45% for twenty-six android malware detection models.

The rest of this paper is structured as follows. Section 2 explains the proposed framework and implementation details of the feature reduction strategy. Section 3 presents the experimental setup and evaluation parameters, while Sect. 4 discusses experimental results. Section 5 highlights related work in the domain of android malware detection, and finally, Sect. 6 concludes the work.

2 Overview and Proposed Framework

This section explains the problem definition and the proposed framework for effective and efficient android malware detection. Later we explain the feature reduction algorithm.

2.1 Problem Definition

Consider an android dataset 'D' containing 'm' android applications.

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m), \forall x \in X, \forall y \in Y\} \tag{1}$$

The malicious applications M are labeled as 1 and the Benign applications B are labeled as 0. X represents the features of android applications and $Y = \{0, 1\}^m$ represents the labels such that

$$y_i = \begin{cases} 1 & \text{if } x_i \in M \\ 0 & \text{if } x_i \in B \end{cases} \quad \text{for i = 1,2 ... m, where} |B| \approx |M| \tag{2}$$

Each android application can be represented by features such as permissions (P), intents (I) and opcodes (O). Considering android permissions, intents and opcode features, feature set $X \subseteq \{P, I, O, ...\}$ can be constructed by static analysis of the android applications to train malware detection models. Malware detection models aim to find a hypothesis $f(x_i)$ that reduces the number of instances for which $f(x_i) \neq y_i$. The performance of these models can be evaluated using parameters like detection accuracy, area under curve, false positives, training and testing time. As the number of features in X increases, the training and testing time also increases proportionally, even if only a few of the features contribute substantially to the detection performance. Feature selection aims to find the most discriminating/contributing features to increase the classification models' efficiency and overall performance. The goal is to increase the efficiency of malware detection models using feature reduction, tumbling down the train and test time while maintaining high classification performance.

2.2 Proposed Framework

Figure 1 shows the proposed framework to construct an efficient and effective android malware system. Android applications are collected from Drebin and AMD benchmark android malware datasets. Each set has approximately the same number of benign and malware samples labeled and verified using VirusTotal. Android permissions, intents, and opcodes are extracted from each android application by reverse engineering and transformed into feature vectors for static analysis. Further, we reduce the features from each of P, I, and O independently and in combination using our proposed feature reduction algorithm. Finally, we evaluate the effect of feature reduction on malware detection performance using machine learning and deep learning algorithms.

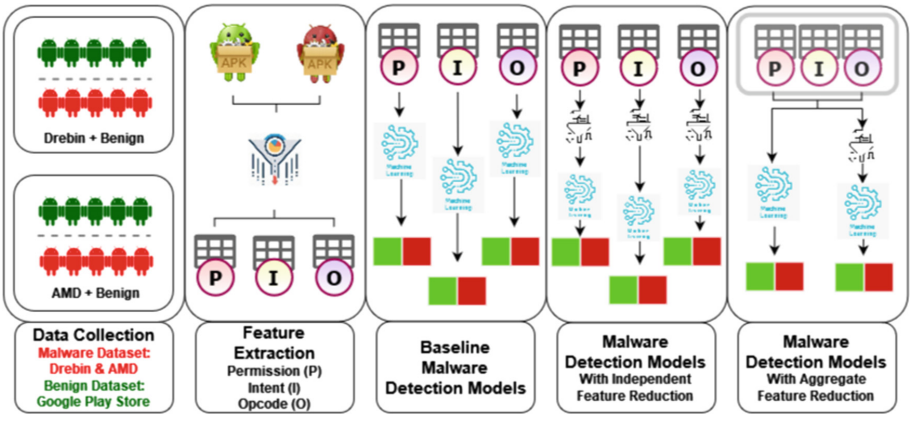

Fig. 1. Proposed framework for effective and efficient android malware detection based on feature reduction.

2.3 Proposed Feature Reduction Approach

Algorithm 1. Proposed Feature Reduction Algorithm

INPUT: Feature Matrix F
OUTPUT: (Reduced set of features, R')
FUNCTION:
getSelection: returns feature importance scores
topFeatures: returns top 10% features
Independent:
1: *Reduced feature set* $R' = \emptyset$
2: $scores \leftarrow getSelection(F)$
3: $sort(scores, descending, inline)$
4: $R' \leftarrow R' \cup topFeatures(scores)$
5: *return* R'
6: **Merging**:
7: **INPUT**: $F'_{m \times n} = \{P_{m \times p}, I_{m \times i}, O_{m \times o}\}$, where $n \leq p + i + o$
8: **OUTPUT**: (Reduced set of features, R'')
9: *Reduced feature set* $R'' = \emptyset$
10: **for** each $feature\_set \in F'$ **do**
11: $R'' \leftarrow R'' \cup Independent(feature\_set)$
12: **end for**

Algorithm 1 illustrates the proposed strategy for feature reduction in malware detection models. It consists of two main functions - Independent and Merging. Independent accepts a feature matrix $F \in \{P, I, O\}$ as input, where P, I, O are permission, intent, and opcode vector space, respectively. The algorithm starts from line number l by initializing the reduced feature set R' to null. In line 2, the current feature space F is passed to a traditional feature selection method using getSelection, which returns a score for each feature. Line 3 sorts F inline in descending order based on the scores. Line 4 finds the best 10% of the features using the calculated scores and stores it in R', finally returning R' in line 5. Merging takes the feature set F' as input, formed by combining P, I, and O feature sets. Line 7 initializes the reduced merged feature set R" to null. The permissions, intents, and opcodes are taken one by one in lines 10 to 12. The Independent function is applied to each feature set and then combined to get the reduced feature set R" in line 11.

Table 1 shows the resultant list of features after applying the feature reduction algorithm on P, I, and O feature sets. The algorithm has determined the most contributing 19 android permissions, 27 intents, and 25 opcodes to be used for further analysis.

Table 1. Reduced feature list containing Permissions (19), Intents (27) and Opcodes (25).

ANDROID PERMISSION (19)	ANDROID INTENT (27)	ANDROID OPCODE (25)
android.permission. ACCESS_COARSE_LOCATION	android.intent.action. ACTION_SHUTDOWN	OP_CHECK_CAST
android.permission. ACCESS_FINE_LOCATION	android.intent.action. BOOT_COMPLETED	OP_CONST_STRING
android.permission. ACCESS_NETWORK_STATE	android.intent.action. CREATE_SHORTCUT	OP_CONST_16
android.permission. ACCESS_WIFI_STATE	android.intent.action. EDIT	OP_CONST_4
android.permission. CHANGE_WIFI_STATE	android.intent.action. MAIN	OP_GOTO
android.permission. GET_ACCOUNTS	android.intent.action. MEDIA_BAD_REMOVAL	OP_IF_EQZ
android.permission. INTERNET	android.intent.action. MEDIA_NOFS	OP_IF_NEZ
android.permission. READ_EXTERNAL_STORAGE	android.intent.action. MY_PACKAGE_REPLACED	OP_IGET
android.permission. READ_PHONE_STATE	android.intent.action. NEW_OUTGOING_CALL	OP_IGET_OBJECT
android.permission. RECEIVE_BOOT_COMPLETED	android.intent.action. PACKAGE_ADDED	OP_INVOKE_DIRECT
android.permission. RECEIVE_SMS	android.intent.action. PACKAGE_REMOVED	OP_INVOKE_INTERFACE
android.permission. WAKE_LOCK	android.intent.action. PICK	OP_INVOKE_STATIC
android.permission. WRITE_EXTERNAL_STORAGE	android.intent.action. REBOOT	OP_INVOKE_VIRTUAL
android.permission. SYSTEM_ALERT_WINDOW	android.intent.action. SCREEN_ON	OP_IPUT_OBJECT
android.permission. SEND_SMS	android.intent.action. SEND	OP_MOVE_RESULT
android.permission. CALL_PHONE	android.intent.action. TIME_SET	OP_MOVE_ RESULT_OBJECT
android.permission. GET_TASKS	android.intent.action. USER_PRESENT	OP_NEW_INSTANCE
android.permission. READ_SMS	android.intent.action. VIEW	OP_NOP
android.permission. CAMERA	android.intent.category. BROWSABLE	OP_RETURN
	android.intent.category. DEFAULT	OP_RETURN_OBJECT
	android.intent.category. HOME	OP_RETURN_VOID
	android.intent.category. INFO	OP_SGET_BYTE
	android.intent.category. LAUNCHER	OP_SGET_SHORT
	android.intent.category. LEANBACK_LAUNCHER	OP_SPUT_BYTE
	android.intent.category. PREFERENCE	OP_SPUT_SHORT
	android.intent.category. SAMPLE_CODE	
	android.intent.extra. REFERRER_NAME	

3 Experimental Setup

This section first explains the datasets used for experiments, followed by data preprocessing and feature extraction performed in this work. Later we discuss the classification algorithms and performance metrics used in this work.

3.1 Malware and Benign Dataset

The datasets contain malware and benign android applications gathered from various authentic sources. We have conducted all the experiments on two different datasets to validate the generalizability of the proposed approach.

1. **Dataset-1:**
   Malware Applications (Drebin): We have used the Drebin dataset developed by Arp et al., which contains 5,560 malicious android applications from over 179 malware families [4]. The malicious applications were collected from the Google Play Store, various Chinese marketplaces, and alternative Russian marketplaces. It also contains 1,260 malicious applications from Android Malware Genome Project. The Drebin dataset is one of the most studied and cited malware datasets available to the research community.
   Benign Applications: We downloaded 8000 android applications from the Google Play store. The downloaded applications were first validated for benignness (i.e., free of malware) using VirusTotal. VirusTotal is owned by Google and aggregates more than 50 antivirus products and online scan engines. Each downloaded application was uploaded to VirusTotal and is labeled benign if all the VirusTotal antiviruses declare it non-malicious. The rest of the non-benign android samples were discarded. The final benign dataset contained 5,721 android applications.
2. **Dataset-2:**
   Malware Applications (AMD): The second malware dataset used for experiments is the Android Malware Dataset (AMD) proposed by Wei et al. [25]. It contains 24,650 malicious android applications from 71 malware families that are categorized into 135 varieties. The malicious samples were collected from 2010 to 2016. The AMD dataset is one of the largest android malware datasets available to the community.
   Benign Applications: We afresh downloaded over 25,000 android applications from the Google Play store between 2015 and 2016. We again used the services of VirusTotal to identify and label benign applications. A downloaded android application was labeled benign if all the applications from VirusTotal declared it as benign. Again, the rest of the non-benign applications were discarded. The final dataset contained 24,408 benign android applications.

3.2 Feature Extraction

We have performed static analysis of android applications and extracted the key features, namely **Permission (P)**, **Intent (I)**, and **Opcode (O)**, for constructing malware detection models.

The android permission system drives security in the android ecosystem by controlling access to data (such as system data, user data, or system state) and controlling actions (such as accessing the internet, recording audio, or connecting to a paired device). Malicious applications try to fool users into accepting unrelated permissions that can together provide easy access to critical and personal data. They can also perform unauthorized actions on the victim's android devices. This led us to use android permissions as a feature set for constructing malware detection models. We first disassembled benign and malicious android applications in the dataset using Apktool. Apktool is a third-party reverse engineering tool used to assemble, disassemble, and reassemble android applications. The disassembled application contains many files, including the *AndroidManifest.xml* file that contains the application's permission usage. We developed a parser that scanned through *AndroidManifest.xml* of each disassembled application in the dataset and generated a permission feature vector. There are 197 unique system-defined permissions in the android ecosystem. The final permission feature vector is binary, where a row represents an android application, and a column represents the usage of the specific android permission in that application.

Similarly, the android intent system is used by applications to signal to the operating system or other applications that a specific event will take place. It can be used as a messaging object to request specific actions to be performed by other applications. Its functionalities allow communication between components which can result in easy access to highly sensitive services in the android system. The *AndroidManifest.xml* file of a disassembled application also contains the list of android intents used by that application. Our parser scanned the *AndroidManifest.xml* of all disassembled applications in the dataset and generated the intent feature vector. There are 273 unique system-defined intents in the android ecosystem. The final intent feature vector is also binary, where a row represents an android application, and a column represents whether the particular intent is used in the application.

An opcode is the first byte of an instruction that indicates the actual operation performed by the processor. Opcodes can detect malicious attacks at the lowest level of the android platform architecture and are part of the instructions used to communicate with the hardware. Each reverse-engineered android application has Dalvik bytecode instructions containing the opcode and operands of that instruction. There are 256 Dalvik opcodes ranging from 00 to FF in the opcode list provided by android. After reverse-engineering each application from the dataset using Apktool, a parser scans it to generate the frequency of each opcode used by the application.

3.3 Classification Algorithms

We validated our proposed feature reduction technique using thirteen malware classification algorithms in four different categories:

1. **Classical Machine Learning Classifiers**
   Classical machine learning (CML) algorithms are the stepping stones to con-

struct more complex ensemble and neural network-based models. They are intuitive to understand and visualize, lightweight with low resource consumption, and highly interpretable. In our work, we have used the following CML algorithms:
(a) Decision Tree Classifier (DT)
(b) Logistic Regression (LR)
(c) K-Nearest Neighbours (kNN)

2. **Bagging based Classifiers**
Bootstrap Aggregation (Bagging) based classifiers are ensembles of individual CML models. Each model is trained on a random sample taken from the dataset with replacement, thus effectively combating high variance issues. The results are then aggregated to make the final decision by majority voting. The Bagging based classifiers used in our work are as follows:
(a) Random Forest (RF)
(b) Extra Trees Classifier (ET)
(c) Bagged Support Vector Machine (Bag SVM)
3. **Boosting based Classifiers**
Boosting-based classifiers iterate over weak learners, identifying and correcting mistakes while reducing bias until a strong learner is built. Results are generally aggregated by assigning weights to each model based on their classification performance. Boosting algorithms used in our work are as follows:
(a) Gradient Boosting (GB)
(b) Adaptive Boosting (AB)
(c) eXtreme Gradient Boosting (XGB)
(d) Light Gradient Boosting Machine (LGBM)
4. **Deep Neural Network-based Classifiers**
Neural networks are complex mathematical models based on the working of the human brain that identifies patterns and extracts key features in large data samples. It consists of an input layer of simple models/neurons, optional hidden layers that learn more abstract features, and a final output layer. Neurons' output is fed into activation functions that decide the neuron's importance while increasing the level of non-linearity in the network. Deep neural networks have multiple hidden layers to learn complex functions that map the input data to the target output. In our work, we have explored neural networks with different number of hidden layers, as given below:
(a) DNN with 1 hidden layer (DNN1L)
(b) DNN with 3 hidden layer (DNN3L)
(c) DNN with 5 hidden layer (DNN5L)

3.4 Evaluation Parameters

In this study, we evaluated the malware detection models using the following performance metrics:

– **True positive (TP)** denotes the number of malicious applications correctly classified by the model.

- **True negative (TN)** denotes the number of benign applications correctly classified by the model.
- **False positive (FP)** denotes the number of benign applications wrongly classified by the model as malware.
- **False negative (FN)** denotes the number of malware applications wrongly classified by the model as benign.
- **Accuracy** is the ratio of the number of correct predictions made over the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- **Area under Curve (AUC)** shows the performance of a classification model across varying classification thresholds.
- **Training time** is the time taken by the classification model to learn from the training samples.
- **Testing time** is time taken by the model to classify the unknown test samples as malware or benign.

4 Experimental Results

In this section, we begin with a discussion on the baseline performance of thirteen different malware detection models. Detailed performance analysis of the proposed feature reduction algorithms on independent and merged feature sets is presented in the following sections. We also discuss the effect of feature reduction on training and testing time. Finally, we compare the different parameters of our proposed approach with other malware detection systems.

4.1 Performance of P, I and O Based Malware Detection Models

In the proposed framework, we built different malware detection models using machine learning and deep learning classifiers. During the feature extraction phase, we generated three feature sets namely, Android Permissions (P), Intents (I), and Opcodes (O) for the Drebin and AMD datasets. The baseline models are built for P, I and O separately before feature reduction. We trained thirteen different malware detection models based on Classical Machine Learning (DT, LR, kNN), Bagging (RF, ET, Bagged SVC), Boosting (GB, AB, XGB, LGBM), and Deep Neural Networks (DNN-1L, DNN-3L, DNN-5L). The models were trained on 70% and tested on 30% of each dataset.

Table 2. Experimental results based on original P, I, and O (Drebin).

	Drebin					
	Permission (P) (195)		Intent (I) (273)		Opcode (O) (256)	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
DT	93.58%	93.30%	79.38%	78.80%	93.27%	93.59%
LR	91.52%	91.29%	77.37%	76.93%	88.01%	87.86%
kNN	93.18%	93.08%	67.33%	62.48%	90.63%	90.49%
RF	95.15%	94.99%	79.83%	79.45%	96.01%	96.22%
ET	95.23%	95.18%	80.05%	79.72%	96.04%	96.15%
Bag SVM	94.09%	93.81%	77.50%	76.97%	80.62%	81.94%
ADB	91.37%	91.06%	77.44%	77.29%	92.93%	93.05%
GDB	91.94%	91.63%	78.46%	78.09%	93.88%	94.03%
XGB	94.73%	94.37%	79.80%	79.35%	96.37%	96.76%
LGBM	94.79%	94.61%	79.61%	79.18%	96.23%	96.39%
DNN1L	93.47%	93.43%	78.98%	78.51%	88.02%	88.29%
DNN3L	94.56%	94.51%	79.52%	79.01%	87.07%	87.42%
DNN5L	94.35%	94.30%	80.02%	79.50%	87.01%	87.45%

Table 2 compares the accuracy and AUC of each model for each feature set of the Drebin dataset. Extra Trees Classifiers (ET) performed the best for the 195 permissions and the 273 intents with accuracies of 95.23% and 80.05% respectively, and AUC of 95.18% and 79.72% respectively. For the 256 Opcodes, XGB performed the best with an accuracy and AUC of 96.37% and 96.76% respectively.

Table 3. Experimental results based on original P, I, and O (AMD).

	AMD					
	Permission (P) (195)		Intent (I) (273)		Opcode (O) (256)	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
DT	94.88%	94.94%	85.59%	85.28%	95.83%	96.08%
LR	93.41%	93.24%	82.63%	82.28%	86.23%	86.39%
kNN	94.90%	94.88%	82.41%	84.54%	94.83%	94.62%
RF	95.97%	95.97%	86.00%	85.70%	97.76%	97.65%
ET	95.95%	95.93%	86.01%	85.74%	97.47%	97.31%
Bag SVM	95.05%	94.69%	83.25%	83.27%	86.86%	87.06%
ADB	92.98%	92.92%	83.34%	82.88%	94.70%	94.62%
GDB	93.50%	93.46%	83.95%	83.69%	95.53%	95.53%
XGB	95.53%	95.57%	85.59%	85.35%	98.33%	98.32%
LGBM	95.22%	95.00%	85.52%	85.13%	97.97%	98.04%
DNN1L	95.18%	95.18%	84.97%	84.84%	92.64%	92.69%
DNN3L	95.34%	95.33%	85.26%	85.13%	89.77%	89.95%
DNN5L	95.48%	95.49%	85.22%	85.08%	87.87%	88.09%

Table 3 compares accuracy and AUC for the models trained on AMD dataset. It is observed that Random Forest (RF) and ET are top performers for permissions and intents, while XGB leads the opcode models. RF performed slightly better than ET for permissions with 95.97% accuracy and 95.97% AUC. ET edges over RF for intents with an accuracy of 86.01% and AUC of 85.74%. XGB performed the best for AMD opcodes, similar to our observations for Drebin, with an accuracy and AUC of 98.33% and 98.32% respectively.

4.2 Performance of (10% of P), (10% of I) and (10% of O) Based Malware Detection Models

A novel feature reduction method was developed over and above classical feature selection approaches to build lightweight and highly accurate malware detectors using the most discriminating/contributing features. Feature scores were estimated for permission, intent, and opcode feature sets in a mutually exclusive manner using local feature selectors. The features were selected until a 10% threshold was reached for the given feature set.

Table 4. Results after feature reduction (10 % of P), (10% of I) & (10% of O)(Drebin).

	Drebin					
	Permission (19)		Intent (27)		Opcode (25)	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
DT	91.34%	88.46%	78.10%	77.53%	92.05%	92.29%
LR	88.49%	86.11%	75.79%	75.79%	83.33%	83.76%
kNN	90.49%	90.31%	71.78%	61.69%	90.17%	90.32%
RF	92.12%	90.25%	78.49%	78.01%	94.39%	94.44%
ET	91.91%	89.14%	78.66%	78.24%	94.64%	94.79%
Bag SVM	91.95%	89.26%	76.26%	76.02%	79.30%	80.41%
ADB	88.31%	86.11%	75.94%	75.75%	88.98%	88.90%
GDB	89.97%	87.65%	77.02%	76.70%	89.75%	89.96%
XGB	92.20%	89.88%	78.19%	77.68%	94.00%	93.98%
LGBM	92.28%	90.25%	78.18%	77.49%	93.46%	93.50%
DNN1L	91.84%	90.31%	77.24%	76.76%	84.50%	84.95%
DNN3L	91.61%	88.95%	78.48%	77.99%	87.54%	87.73%
DNN5L	91.55%	91.05%	78.51%	78.02%	72.84%	73.96%

Table 4 shows the performance of thirteen malware classifiers after reducing the features to the top ten percent of the original Drebin feature sets (P, I, and O). With just 19 permissions, Light Gradient Boosting Machine(LGBM) achieved the highest accuracy of 92.28%, while kNN and DNN1L achieved the highest AUC of 90.31%. For 27 Intents, ET performed the best, with accuracy and AUC of 78.66% and 78.24%, respectively. The 25 Opcode features achieved the best classification

performance for ET with accuracy and AUC of 94.64% and 94.74%, respectively. Comparing Tables 2 and 4, we can see that the performance of the reduced malware classifiers is still close to that of the models trained on the complete feature sets while also reducing the number of features to 10% of the original.

Table 5. Results after feature reduction (10 % of P), (10% of I) & (10% of O) (AMD).

	AMD					
	Permission (19)		Intent (27)		Opcode (25)	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
DT	93.63%	93.57%	85.23%	84.92%	94.35%	94.05%
LR	91.09%	91.06%	81.82%	81.27%	82.61%	82.37%
kNN	93.50%	93.62%	81.86%	84.26%	94.37%	94.14%
RF	94.16%	94.15%	85.55%	85.24%	96.35%	96.29%
ET	94.09%	94.07%	85.50%	85.20%	96.37%	96.39%
Bag SVM	93.71%	93.78%	82.72%	82.76%	86.35%	86.67%
ADB	89.61%	90.03%	82.91%	82.74%	87.71%	88.11%
GDB	92.09%	92.19%	84.11%	83.65%	90.15%	90.18%
XGB	93.97%	94.02%	85.15%	84.82%	95.72%	95.68%
LGBM	93.45%	93.33%	84.97%	84.58%	94.56%	94.30%
DNN1L	93.43%	93.42%	84.69%	84.56%	88.30%	88.46%
DNN3L	93.87%	93.88%	85.01%	84.87%	88.96%	88.96%
DNN5L	93.93%	93.92%	84.83%	84.71%	85.42%	84.04%

Table 5 illustrates the same comparison on the AMD dataset. For the AMD dataset, it was noticed that RF outplayed other algorithms with the highest accuracy and AUC of 94.16% and 94.15%, respectively. For 27 Intents, RF again performs the best with accuracy and AUC of 85.55% and 85.24%, respectively. The 25 opcodes proved to be the best-performing feature set, with the ET achieving accuracy and AUC of 96.37% and 96.39%, respectively. Permissions achieved an average accuracy of 93.12% and average AUC of 93.16%. Intents obtained the least average accuracy and AUC of 84.18% and 84.12%. Opcodes achieved an average accuracy of 90.86% and average AUC of 90.74%. Comparing Tables 3 and 5, we observed that even after reducing the features to 10% of the original feature set, we could maintain the accuracy and AUC close to the original.

4.3 Performance of Merged (P+I+O) and Reduced Merged (10% of (P+I+O)) Based Malware Detection Models

In this section, we compare the performance of the merged feature set (724 features) with the reduced feature set (72) on the malware detection models for Drebin and AMD datasets.

Figure 2 shows that almost all classifier algorithms, noticeably RF, ET, and all the boosting algorithms obtained similar accuracy even after feature reduction to 72 on the Drebin dataset. XGB achieved the highest accuracy before (98.03%) and after feature reduction (97.86%). The average accuracy for Drebin is 92% after feature reduction.

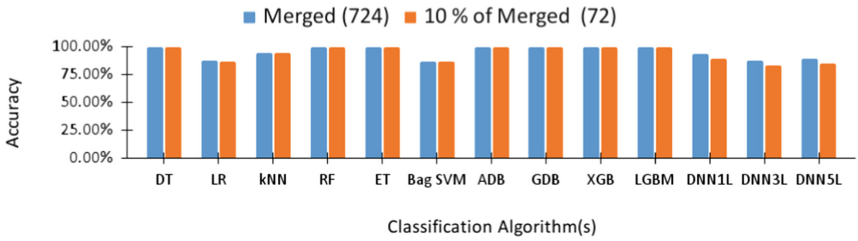

Fig. 2. Accuracy of (P+I+O) and after feature reduction (10% of (P+I+O)) (Drebin).

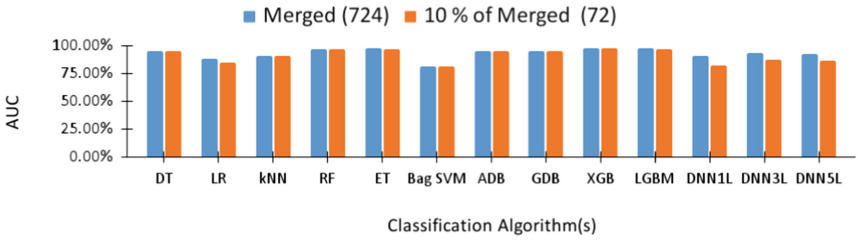

Fig. 3. AUC of (P+I+O) and after feature reduction (10% of (P+I+O)) (Drebin).

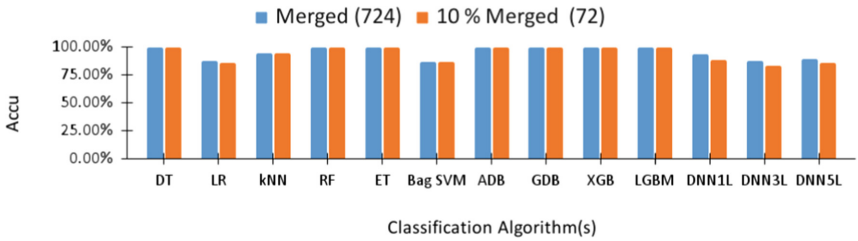

Fig. 4. Accuracy of (P+I+O) and after feature reduction (10% of (P+I+O)) (AMD).

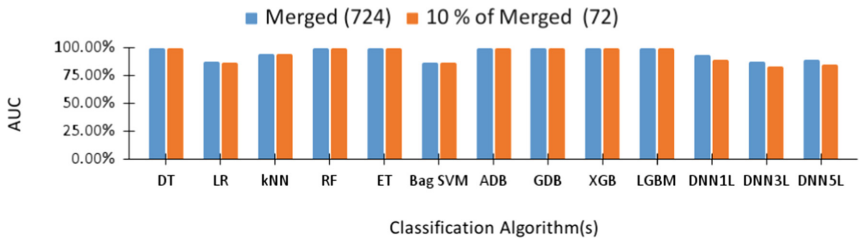

Fig. 5. AUC of (P+I+O) and after feature reduction (10% of (P+I+O)) (AMD).

Figure 3 demonstrates the AUC before and after feature reduction for the Drebin dataset. ET achieved the best AUC of 98.07% before feature reduction, while XGB performed the best with an AUC of 97.65% for the reduced feature sets. All the CML, bagging, and boosting models have achieved approximately equal results before and after feature reduction. The average AUC for Drebin is 91.7% after feature reduction.

Figure 4 demonstrates the near-perfect accuracies for AMD before and after feature reduction. DT, RF, ET, and all the boosting-based algorithms have achieved nearly 100% accuracy for the 724 merged and 72 reduced features. The average accuracy for AMD before and after feature reduction is 95.4% and 94.2%, respectively. It is observed from the results that the 10% selected features are performing almost equal to the original feature set.

Figure 5 demonstrates the AUCs for AMD before and after feature reduction. DT, RF, ET, and all the boosting-based algorithms have achieved nearly 100% AUC for the 724 merged AMD feature sets. This performance was also observed for the 72 features after feature reduction, for which DT, RF, ET, and the boosting algorithms have nearly 100% AUC. kNN obtained the second-highest AUC of 94.7% for the merged set and a close 94.6% for the reduced feature set. The average AUC for AMD before and after feature reduction is 95.4% and 94.2%, respectively. The average accuracies and AUCs before and after feature reduction are greater than 94% for the AMD dataset.

We observed that after reducing the features to 10% of the original merged set, there was only a 1.69% accuracy reduction for Drebin and AMD datasets. The AUC reduced by merely 1.6% on average after reducing the feature set from 724 features to 72 features.

Table 6. Training and Testing time taken by malware detection models on the Merged features and Reduced Features in seconds (Drebin).

	Drebin			
	Merged (P+I+O) (724)		10% of Merged (72)	
	Training Time	Testing Time	Training Time	Testing Time
DT	1.40	0.02	0.30	0.00
LR	3.67	0.03	0.57	0.00
kNN	2.45	5.28	0.19	0.53
RF	1.43	0.08	0.84	0.06
ET	3.21	0.12	0.82	0.07
Bag SVM	148.21	81.29	21.34	7.01
ADB	16.64	1.28	3.22	0.12
GDB	30.93	0.05	6.80	0.01
XGB	18.24	0.14	2.38	0.02
LGBM	6.95	0.15	0.56	0.02
DNN1L	229.89	96.43	21.77	4.52
DNN3L	242.52	125.21	23.95	4.50
DNN5L	454.89	352.46	29.30	5.56

Table 7. Training and Testing time taken by malware detection models on the Merged features and Reduced Features in seconds (AMD).

	AMD			
	Merged (P+I+O) (724)		10% of Merged (72)	
	Training Time	Testing Time	Training Time	Testing Time
DT	0.86	0.06	0.41	0.01
LR	14.76	0.22	7.96	0.01
kNN	14.07	46.42	1.10	6.89
RF	7.25	0.20	2.45	0.08
ET	8.93	0.48	1.65	0.15
Bag SVM	2,862.31	1,014.27	435.40	109.50
ADB	60.58	3.60	15.73	0.58
GDB	106.57	0.23	31.66	0.05
XGB	28.22	0.30	4.70	0.04
LGBM	5.92	0.27	1.12	0.05
DNN1L	528.33	196.22	73.06	20.00
DNN3L	519.96	198.17	82.31	19.20
DNN5L	562.08	196.05	97.33	19.27

4.4 Effect of Feature Reduction on Training Time and Testing Time

With an increase in dimensionality, there is a proportional increase in space, time, and resource consumption, especially when we merge all the feature spaces. However, some features contribute more to the model's discrimination capability than others. In our study, we observed that if only the most contributing features are used to train the models, their performance is similar to the original feature set and their training and testing time reduces drastically. Drebin's highest reduction in training time was 93% with DNN5L, and the lowest was 41% with RF. The highest reduction in testing time was nearly 100% for DT and LR, followed by DNN5L with 98%, and the lowest was 25% with RF.

Similarly, for AMD, kNN achieved the highest reduction in training time with 92.18% and the least was for LR with 46%. LR, DNN1L, and Bag-SVM reduced their testing time by 95%, 89.8%, and 89.2% for AMD, while the least test time reduction was 60% by RF. It is worth noting that the feature reduction creates a tumbling effect on training and testing time. After the proposed feature reduction, the testing time improved drastically by 91.45% on average for the Drebin and AMD datasets. Likewise, the training time improved by 85.25% on average after the feature reduction. The experimental results prove that if the top contributing features are selected for malware detection, there is almost no loss in detection performance, while the training and testing time is reduced drastically. Our proposed algorithm reduced features to the top 10% of the feature sets to build efficient and effective malware detection models.

Table 8. Comparison with existing literature in android malware detection.

Authors	Original Number of Features	Feature Selection	Reduced Number of Features	Feature List	Number and Classifier Used	Detection Accuracy
Arp et al. [4]	545,000	No	-	Yes	1: Linear SVM	94%
C.Li et al. [6]	294,019	No	-	No	1: Factorization Machine	99%
Tangil et al. [22]	20,000	Yes	1000	Yes	1: ET	99.82%
Latha et al. [10]	4115	Yes	518	No	5:RF, SVM, kNN,DT, NC	93.73%
McLaughlin et al. [9]	218	No	-	No	CNN	98%
Wang et al. [23]	135	Yes	40	Yes	3:SVM,DT, RF	94.62%
Chen et al. [5]	230	Yes	107	No	4: RF, SVM,kNN, NB	N/A
Wang et al. [24]	112	Yes	70	Yes	4:NB, DT,NN, kNN	94 - 98%
Our Proposed System	724	Yes	72	Yes	13:CML, Bagging,Boosting,DNN	99.99%

4.5 Comparative Analysis with Existing Literature

Table 8 compares our experimental results with existing literature in the domain of malware detection. Feature reduction is a critical step to avoid the curse of dimensionality, especially in static analysis-based malware detection. Arp et al. used a massive set of 545,000 features and achieved a detection accuracy of

94% [4]. Similarly, Li et al. have also used an entire set of 294,019 permissions and intents to achieve an accuracy of 99.73% for Drebin and 99.05% for AMD [5]. These approaches have failed to find a balance between detection effectiveness and efficiency. [10,22] and [5] have performed feature selection and reduced their feature sets to 1000, 518, and 107 features. They evaluated their approaches on only 1, 5, and 4 machine learning models, respectively, while we have systematically evaluated our proposed system on 13 classifiers from 4 different categories. Despite using fewer features, our proposed system achieved higher detection performance. Wang et al. analyzed android permissions and selected 40 risky permissions [23]. Using permission-based features alone can cause the model to miss significant malicious behavior that could have been noticed if opcodes and intents were also considered. Wang et al. performed dynamic analysis and extracted 112 kernel features, which were further reduced to 70 [24]. However, their approach is highly resource-intensive, with large memory consumption, data collection, and training costs. In contrast, we improved our training time by 93.55% and testing time by 98.42%.

5 Related Work

Android malware has been a serious concern in recent years [12,14,19]. The anti-malware research community and the security industry are actively working on improving malware detection systems while balancing their effectiveness, computational complexity, resource usage, and training and testing time [18,20,21]. Most studies use a single feature space over the same dataset to build malware prediction models using machine learning [13,16,17]. According to a study conducted by Liu et al. between 2014 and 2020, 68 out of 132 papers on android malware defenses were based on permissions [8]. Sun et al. extracted 22 significant permissions out of 135 by performing 3-level pruning on the android permissions features set, achieving 90% accuracy [22]. Wen et al. proposed the PCA-RELIEF feature selection algorithm to find discriminative feature subgroups [22]. They obtained an accuracy of 95.2% using SVM trained on Drebin permissions. However, the work lacks validation from a variety of feature spaces and datasets. Chen et al. trained on opcode sequences alone, achieving a detection rate/TPR of 98.8% [5]. They assigned ten symbols to each type of instruction and used the symbol sequences to train their models. Latha et al. proposed the BOAWFS feature selection method based on Bat optimization to reduce the number of permission features in the CICInvesAndMal2019 android malware dataset [10]. They reduced 4115 features to 518 features and achieved a detection accuracy of 93.73%. However, flagging an application as malware or benign based only on knowledge gained from a single feature type can result in missing out on the overall picture. DroidSeive used syntactic and resource-centric features that hinted at malware obfuscation [22]. Around 1000 features were selected out of 20000 and trained using the Extra trees classifier, achieving a detection accuracy of 99.82%. Their work was not validated with a variety of machine-learning algorithms.

Most of the research works have categorized an application as malware or safe based on a single feature space like permissions or opcodes. The trained models would be more ignorant than models trained on a variety of features. The community has focused on improving detection performance at the cost of efficiency by using large feature sets. This is counter-intuitive as large feature sets can lead to overfitting and the curse of dimensionality, thus reducing the models' real-world performance. It also increases resource consumption, training time, and evaluation time. Studies that have performed feature reduction have not validated their approach and feature set using a variety of machine learning-based malware detection models. The selected features were also not published for further validation.

6 Conclusion

Android OS is synonymous with smartphones these days. Its wide usage, global market share, and ease of application development led to android platforms becoming a breeding ground for malware applications. Current signature-based malware detectors identify malware by comparing with known malware signature databases, but they cannot detect unknown malware. This resulted in a shift to machine learning-based malware detectors. Since android applications produce high dimensional data for machine learning, it is crucial to apply feature reduction techniques to reduce computational time and cost. In this paper, we proposed a feature reduction strategy for static analysis-based malware detection systems. To test the viability of our proposed work, we used two different Android malware data sources, namely Drebin and AMD datasets, to build a total of twenty-six models based on thirteen machine learning algorithms in four different categories (Classical ML, Bagging, Boosting, and DNN). Initially, we built baseline models on the individual permissions (195), intents (273), and opcode (256) feature sets, and then merged the feature sets to 724 features. The proposed feature reduction technique reduced each feature set to 10% of its original dimensions (i.e.) 195 permissions were reduced to 19 features, 273 Intents to 27 features, 256 Opcodes to 25 features, and the merged features (724) to just 72 features. We obtained an average accuracy of 94.73% before feature reduction on the merged features and 93.12% after feature reduction, resulting in just 1.69% reduction. Merged features before reduction achieved an AUC of 94.49% while the reduced features achieved 92.97% resulting in a mere 1.61% reduction. We reduced the average training time drastically by 85.26% and the average testing time by 91.45%. The proposed feature reduction approach resulted in huge reduction in training and testing time while achieving high detection performance comparable to the original high dimensional feature set. In the future, we will focus on adversarial attack and defense strategies to make our malware detection systems more robust against adversarial attacks.

References

1. McAfee Mobile Threat Report. https://www.mcafee.com/en-us/consumer-support/2020-mobile-threat-report.html (2020). Accessed Jan 2023
2. AVTEST. https://portal.av-atlas.org/malware/statistics (2022). Accessed Jan 2023
3. IDC Smartphone Market Share. https://www.idc.com/promo/smartphone-market-share (2022). Accessed Jan 2023
4. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: effective and explainable detection of android malware in your pocket. In: Network and Distributed System Security (NDSS) Symposium, vol. 14, pp. 23–26 (2014)
5. Chen, T., Mao, Q., Yang, Y., Zhu, J.: TinyDroid: a lightweight and efficient model for android malware detection & classification. Mobile Inf. Syst. **2018**, 1–9 (2018)
6. Li, C., Mills, K., Niu, D., Zhu, R., Zhang, H., Kinawi, H.: Android malware detection based on factorization machine. IEEE Access **7**, 184008–184019 (2019)
7. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., Ye, H.: Significant permission identification for machine-learning-based android malware detection. IEEE Trans. Industr. Inf. **14**(7), 3216–3225 (2018)
8. Liu, Y., Tantithamthavorn, C., Li, L., Liu, Y.: Deep learning for android malware defenses: a systematic literature review. arXiv preprint arXiv:2103.05292 (2021)
9. McLaughlin, N., et al.: Deep android malware detection. In: ACM Conference On Data and Application Security and PrivacY (CODASPY), pp. 301–308 (2017)
10. Pushpa Latha, D.: Bat optimization algorithm for wrapper-based feature selection and performance improvement of android malware detection (2021)
11. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y.: A survey of android malware detection with deep neural models. ACM Comput. Surv. (CSUR) **53**(6), 1–36 (2020)
12. Rathore, H., Nikam, P., Sahay, S.K., Sewak, M.: Identification of adversarial android intents using reinforcement learning. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2021)
13. Rathore, H., Sahay, S.K., Thukral, S., Sewak, M.: Detection of malicious android applications: classical machine learning vs. deep neural network integrated with clustering. In: Gao, H., J. Durán Barroso, R., Shanchen, P., Li, R. (eds.) BROADNETS 2020. LNICST, vol. 355, pp. 109–128. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68737-3_7
14. Rathore, H., Samavedhi, A., Sahay, S.K., Sewak, M.: Robust malware detection models: learning from adversarial attacks and defenses. Foren. Sci. Int. Digit. Investig. **37**, 301183 (2021)
15. Sewak, M., Sahay, S.K., Rathore, H.: An investigation of a deep learning based malware detection system. In: 13th International Conference on Availability, Reliability and Security (ARES), pp. 1–5 (2018)
16. Sewak, M., Sahay, S.K., Rathore, H.: Assessment of the relative importance of different hyper-parameters of LSTM for an IDS. In: IEEE Region 10 Conference (TENCON), pp. 414–419. IEEE (2020)
17. Sewak, M., Sahay, S.K., Rathore, H.: DeepIntent: implicitintent based android IDS with E2E deep learning architecture. In: International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 1–6. IEEE (2020)
18. Sewak, M., Sahay, S.K., Rathore, H.: Value-approximation based deep reinforcement learning techniques: an overview. In: International Conference on Computing Communication and Automation, pp. 379–384. IEEE (2020)

19. Sewak, M., Sahay, S.K., Rathore, H.: Deep reinforcement learning for cybersecurity threat detection and protection: a review. In: Krishnan, R., Rao, H.R., Sahay, S.K., Samtani, S., Zhao, Z. (eds.) Secure Knowledge Management In The Artificial Intelligence Era. SKM 2021. Communications in Computer and Information Science, vol. 1549, pp. 51–72. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-97532-6_4
20. Sewak, M., Sahay, S.K., Rathore, H.: DRLDO: a novel DRL based de-obfuscation system for defence against metamorphic malware. Def. Sci. J. **71**(1), 55–65 (2021)
21. Sewak, M., Sahay, S.K., Rathore, H.: DRo: a data-scarce mechanism to revolutionize the performance of DL-based security systems. In: IEEE 46th Conference on Local Computer Networks (LCN), pp. 581–588. IEEE (2021)
22. Sun, L., Li, Z., Yan, Q., Srisa-an, W., Pan, Y.: SigPID: significant permission identification for android malware detection. In: 11th International Conference on Malicious and unwanted software (MALWARE), pp. 1–8. IEEE (2016)
23. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**(11), 1869–1882 (2014)
24. Wang, X., Li, C.: Android malware detection through machine learning on kernel task structures. Neurocomputing **435**, 126–150 (2021)
25. Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current android malware. In: Polychronakis, M., Meier, M. (eds.) DIMVA 2017. LNCS, vol. 10327, pp. 252–276. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60876-1_12
26. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. ACM Comput. Surv. (CSUR) **50**(3), 1–40 (2017)