Computer Science • 20(3) 2019 https://doi.org/10.7494/csci.2019.20.3.3285

Önder Çoban
Selma Ayşe Özel

ADAPTING TEXT CATEGORIZATION FOR MANIFEST BASED ANDROID MALWARE DETECTION

Abstract *Malware is a shorthand of malicious software that are created with the intent of damaging hardware systems, stealing data, and causing a mess to make money, protest something, or even make war between governments. Malware is often spread by downloading some applications for your hardware from some download platforms. It is highly probable to face with a malware while you try to load some applications for your smart phones nowadays. Therefore it is very important that some tools are needed to detect malware before loading them to the hardware systems. There are mainly three different approaches to detect malware: i) static, ii) dynamic, and iii) hybrid. Static approach analyzes the suspicious program without executing it. Dynamic approach, on the other hand, executes the program in a controlled environment and obtains information from operating system during runtime. Hybrid approach, as its name implies, is the combination of these two approaches. Although static approach may seem to have some disadvantages, it is highly preferred because of its lower cost. In this paper, our aim is to develop a static malware detection system by using text categorization techniques. To reach our goal, we apply text mining techniques like feature extraction by using bag-of-words, n-grams, etc. from* `manifest content` *of suspicious programs, then apply text classification methods to detect malware. Our experimental results revealed that our approach is capable of detecting malicious applications with an accuracy between 94.0% and 99.3%.*

Keywords Android, malware detection, text categorization, machine learning.

Citation Computer Science 20(3) 2019: 383–405

1. Introduction

As the use of smart mobile devices increases, the number of applications available for these devices are gradually increasing as well. However, these applications may be malicious software (i.e., malware) and cause security vulnerabilities such as stealing data, causing a mess to make money, protest something, and even make war between governments [21]. Installing an application on mobile devices was very difficult in the past, as there was not any central download platform. But nowadays, the mobile device users can easily download any application from the central platforms such as App Store, Google Play Store, and Windows Store. Many users and developer communities have focused on these platforms and this new model that is developed for distributing and installing applications has been significantly successful [46].

The smart device platforms provide official application markets by adopting some security approaches which are devoted to block malicious attacks. Despite such precautions, however, malware may be installed on the smart devices and/or get an entry to the application markets [46]. Especially, Android platform allows users to install applications from unauthorized sources (e.g., third party markets) and this enables malware to be distributed easily [4]. This means that it is highly probable to face with a malware while you try to load some applications for your mobile device from one of these platforms. A report[1] claimed that approximately half of the Android malware are multifunctional Trojans which steal personal information. There are also approximately 750,000 new Android malware that were detected in the first quarter of 2017[2]. All of these indicators prove that effective malware detection is required for Android applications.

In order to detect and block Android malwares, various approaches were employed so far. These approaches can be classified into three categories, namely, static, dynamic and hybrid [2, 5, 18, 34]. In static analysis, features are extracted by reverse engineering from the Android application file (i.e., apk), without executing the application [4, 5, 22, 58]. Notice that an `apk` file comprises of application codes and resources and created with the files including *AndroidManifest.xml* and *classes.dex* in DEX (Dalvik Executable) format. Static analysis aims to analyze malware using only the information provided by the program itself. This is generally performed by decompilation process which recovers the static source (API/System calls, permissions, opcodes, and hexadecimal byte sequences etc.,) of program [10, 35, 58]. The static analysis has a disadvantage that it can not detect malware that dynamically download malicious code [2] as Android operating system (OS) allows applications to load additional code (malicious or benign) from external sources at runtime. On the other hand, dynamic analysis uses the execution traces of the application in a controlled environment [9] for malware detection. The analysis is performed by running the application on device and such information as system calls, network

[1] `https://securelist.com/IT-Threat-Evolution-Q2-2017-Statistics/79432/`
[2] G Data Security Blog: 8400 New Android Malware Samples Every Day

access, and file and memory modifications are obtained from operating system during runtime [19, 47, 57, 61]. This approach removes the limitations (e.g., unpacking and obfuscation) of the static analysis [2, 34]. However, dynamic approach generally requires more resources in terms of memory space and running time when compared to static analysis [47]. The main difference between the static and dynamic approaches is the necessity of running application on the device [3, 4, 12, 42]. The hybrid analysis is a combination of static and dynamic methods, and it creates a framework to perform both of the analyses [2, 7, 31, 49]. In hybrid approach, the static analysis is performed at first without executing the application. Then, the dynamic analysis is performed, if necessary, by using the information obtained from static analysis [5]. In all approaches, malicious applications are generally detected by borrowing classifiers from the machine learning field [1, 9, 11, 46, 56].

Although static approach may seem to have some disadvantages, it is highly preferred because of its lower cost with respect to other methods. In this paper, our aim is to develop a malware detection system for Android based mobile devices as they are widely used nowadays. Our system is based on static malware detection approach by using text mining methods. Currently, some of the existing works that apply the static approach also use text mining methods by generally considering static data (e.g., permissions, manifest content, meta-data, and source code) of applications [33, 37, 40, 44, 51, 55]. However, differently from existing works, our framework adopts text categorization for this purpose and extracts different features directly from the manifest content of each application to detect malware. Contributions of our study can be summarized as follows:

- We designed a server-side system to enable users to perform malware analysis.
- Differently from existing works, our system is completely based on the manifest contents of the suspicious applications and it employs text mining techniques for the purpose of the malware detection.
- This is the first time that bow, n-gram, and sstf (see Section 3.2) features are used in manifest based malware detection task.
- We conducted intensive experiments to explore the effect of applying dimension reduction together with different circumstances where feature set and classifier are different.
- We obtained better results than other static analysis based studies, from which some of them are completely focused on the manifest file.

The remainder of this paper is as follows: In the next section, we summarize the previous studies. In Section 3, we explain methods used in this paper. We present our experimental results in Section 4. We then discuss the limitations and give future directions of this study in Section 5 and, finally, we present our conclusions in the last section.

2. Literature Review

Android is an open source and widely used OS based on the Linux kernel. As popularity of the Android increases, the number of malwares that target this platform rises as well. Therefore, there are many studies that use static, dynamic, or hybrid approaches to detect malwares and prevent malicious attacks for Android. In this section, we review the literature for Android malware detection (AMD) field and only present studies that are similar to our study. We also summarize these previous works in Table 1.

Table 1

Some of the research works which propose an AMD framework in three different malware detection approaches

Approach	Research Work	Features	Method	Year
Static	SCANDAL [27]	Bytecode of application	Dalvik Core interpretation	2012
	DroidMat [56]	Permissions, Intent messages passing etc., and API calls	Machine learning	2012
	MAMA [46]	Permissions, *uses-feature* tags	Machine learning	2013
	DroidAPIMiner [1]	API calls	Machine learning	2013
	AndroSimilar [20]	Variable length signatures	Fuzzy hashing	2013
	PUMA [45]	*uses-permission/feature* tags	Machine learning	2013
	DroidAnalytics [60]	3 level signatures	Similarity score	2013
	DREBIN [4]	Manifest, Disassembled code	Machine learning	2014
	PMDS [42]	Permissions	Machine learning	2014
	ANASTASIA [22]	Disassembled code features	Machine learning	2016
	ADROIT [35]	Permissions and meta-information from app store	Machine learning	2016
	DMDAM [8]	Permissions	Machine learning	2017
	Arslan *et al.* [6]	Permissions	Machine learning	2019
	Kim *et al.* [28]	Various Features	Deep learning	2019
	Our study	Manifest based textual features (bow, n-gram, and sstf)	Machine learning	2019
Dynamic	TaintDroid [19]	Data flows	Taint tracking	2010
	CrowDroid [9]	Traces of system calls	k-means clustering	2011
	AntiMalDroid [59]	Signatures based on behaviors	Machine learning	2011
	Andromaly [48]	Device states and events	Machine learning	2012
	DroidScope [57]	Traces of system calls	Virtual machine introspection	2012
	DroidScribe [18]	Runtime behaviors	Machine learning	2016
	DroidCat [10]	Application execution traces	Machine learning	2016
	CSCdroid [58]	Determinate system calls	Markov chain and SVM	2017
	Monet [52]	Runtime behavior signature	Signature matching	2017
	Wang *et al.* [55]	HTTP flows	Machine Learning	2018
Hybrid	Andrubis [31]	Manifest meta-data, bytecode, Dalvik VM actions	Code coverage	2014
	Mobile-Sandbox [49]	Permissions, smali codes etc., and runtime behaviors	Machine learning	2015
	DroidNative [2]	ACFG and SWOD signatures from MAIL patterns	A decision tree based similarity detector	2017
	Martin *et al.* [36]	Static and dynamic features	Machine Learning	2019

2.1. Static Analysis

Kim *et al.* proposed a static analyzer, namely, `ScanDal` that uses Dalvik Core interpretation method and extracts features from bytecode of the application [27]. Wu *et al.* proposed `DroidMat`, a system that extracts the information from each application's manifest file, and regards components as entry points to trace API calls related to permissions [56]. `MAMA` extracts permission and *uses-feature* tags from the application's manifest content and uses machine learning methods to distinguish benign applications from malwares [46]. `Drebin` is a machine learning based method that uses features extracted from manifest and disassembled code of application to perform static analysis [4]. Rovelli *et al.* developed a permission based machine learning model for malware analysis [42]. Aafer *et al.* proposed `DroidAPIMiner`, a machine learning based system that relies on the API, package, and parameter level information [1]. `AndroSimilar` is an approach that generates signature by extracting statistically improbable features to detect malicious Android applications [20]. Sanz *et al.* present `PUMA` for detecting malwares through machine learning techniques by analysing the extracted permissions from the application [45]. Zheng *et al.* used opcode level signatures of applications to perform static malware analysis [60]. Fereidooni *et al.* presented a statics machine learning based system, namely, `ANASTASIA` which uses 560 different features extracted from disassembled code of applications [22]. `ADROIT` uses text categorization approach to detect malicious applications. This method extracts features from the meta-data information of application which is available both in the app store and Android manifest [35]. Bhattacharya and Goswami performed malware classification based on permission features by using different machine learning classifiers [8]. Arslan *et al.* performed permission-based malware detection based on machine learning [6]. They obtain results to be processed by a comparing the observed frequencies of the requested permissions both in malicious and benign applications. Kim *et al.* proposed a framework that uses similarity-based feature (e.g., permission, component, environmental etc.) extraction and employs multimodal deep learning for malware detection [28].

2.2. Dynamic Analysis

Cox *et al.* proposed `TaintDroid`, a dynamic system that tracks the flow of privacy sensitive data through third-party applications and uses the data flows to detect malwares [19]. `CrowDroid` is a server-side framework that uses traces of system calls to perform dynamic analysis of applications [9]. Zhao *et al.* proposed a software behavior signature based framework which uses SVM algorithm to detect malicious applications [59]. `Andromaly` is a client-side application that employs features obtained from the device states and events to perform machine learning based dynamic malware detection [48]. `DroidScope` is an emulation based malware analysis engine that can be used to analyze the java and native components of Android applications [57]. Dash *et al.* present `DroidScribe`, a system that generates features at different levels including pure system calls, decoded binder communication, and abstracted behavioral patterns to perform SVM based Android malware family categorization [18]. `DroidCat` uses a set of behavioral features obtained through systematic dynamic profiling of applications for dynamic detection of Android malwares [10]. Differently from other works, `CSCdroid` uses all system calls to construct feature vectors in order to determine the security of applications [58]. Sun *et al.* proposed `Monet` framework that includes both client and server modules and uses both runtime behavior and static structures to detect malware variants [52]. Wang *et al.* proposed a dynamic method which uses text semantic features of mobile traffic for malware detection [55]. This method considers every HTTP flow as a document and then uses n-grams to generate candidate features for machine learning model.

2.3. Hybrid Analysis

There have been only a few studies that perform hybrid analysis of Android applications. The published works that we aware of and use the hybrid approach as follows: `Andrubis` uses features extracted both during the static analysis and dynamic analysis (e.g., Dalvik VM actions, activities etc.) for the purpose of malware detection. `Mobile-Sandbox` is an easily accessible system through a web interface. It combines both static and dynamic approaches and uses different features (e.g., permissions, smali codes, and runtime behaviours etc.) to perform machine learning based malware detection [49]. `DroidNative` is a hybrid system that is able to detect malwares embedded in bytecode or native code [2]. This system performs static analysis on native code and hybrid analysis on byte code. It also adapts a technique from windows malware detection, namely, MAIL for Android malware detection. Martin *et al.* presented a dataset, namely, `OmniDroid` that includes 22K real-world malicious and benign applications. They also performed malware detection on this dataset by using their approach which is based on the fusion of static and dynamic features [36].

Similar to our paper, text categorization approach is employed in some studies to detect malicious applications especially in static approach. Wang *et al.* used lexical features of HTTP header to discover malicious behaviors [55]. The authors obtained HTTP header by tracing the mobile network traffic and utilize n-gram model to extract features from this header that is a structured plain text. Suarez-Tangil *et al.* proposed `Dendroid`, a framework that utilizes text mining approach for code structures based malware detection [51]. The authors present a novel way to measure similarity to automatically classify malwares into families. Santos *et al.* used n-gram based file signatures to detect malicious Android application [44]. Milosevic *et al.* used bag of words model to extract features from the source code of applications for the purpose of malware detection [40]. Malhotra and Bajaj employed text mining approach to extract instruction sets in their signature based pattern matching technique [33]. Mas'ud *et al.* explored the use of different feature selection methods in the n-grams based malware detection. The authors extracted the n-grams from system call sequences of the applications [37]. Wang *et al.* proposed a method, namely, TextDroid which combines text mining and machine learning to detect Android malwares [54]. This method is based on mobile network traffic and uses the HTTP flow header to extract n-gram features that used in malware classification.

As it can be understood from the literature, our work advocates a different idea than others (see Table 1) and the majority of text mining based Android malware detection studies mentioned above utilize n-gram features and do not focus on manifest content. Differently from existing works, in this paper, we utilize text mining approach to extract different level of semantic features (e.g., bow, n-grams, and stylistic features) from manifest content. We utilize these features both separately and together with each other including permission features to observe the effect on performance.

3. System Design

In this section, we introduce methods utilized in our server-side design which adopts text categorization approach for AMD by using machine learning methods. As depicted in Figure 1, Android users and developers are able to perform analysis by sending application file to the remote server hosting our system that uses the following methods to perform malware detection: (1) *decompilation of application archive files*, (2) *textual feature extraction*, (3) *dimension reduction*, and (4) *classification*. When an application is sent (e.g., via a client module) to the remote server, it decompiles APK file to get *AndroidManifest.xml* file.

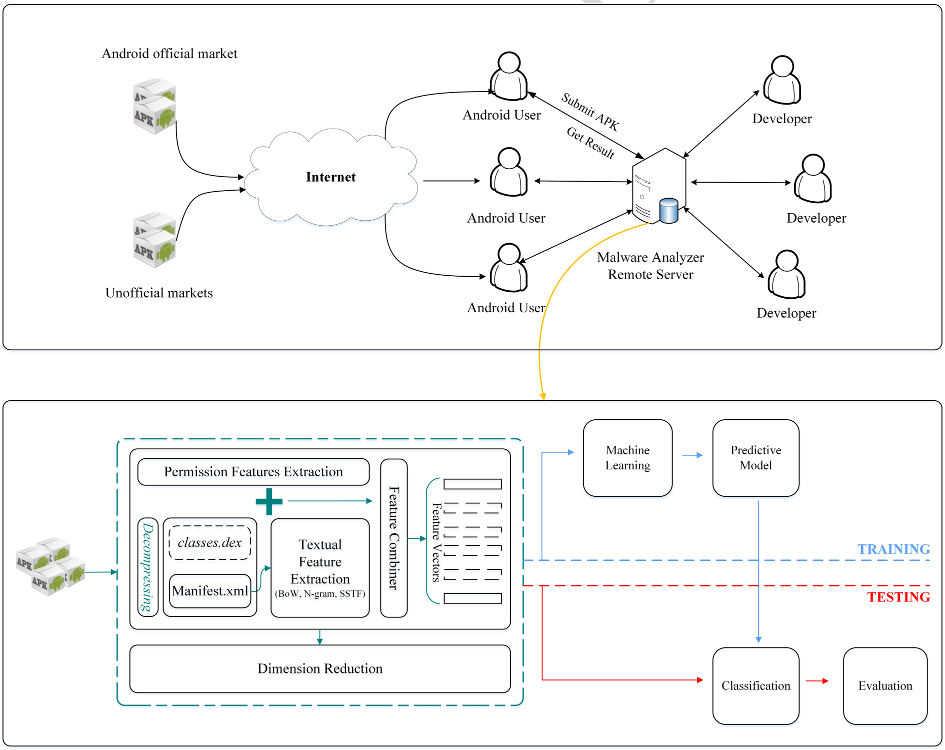

Figure 1. Flowchart of our malware detection process

Then, it extracts features from this manifest content and transforms them into feature vectors. After the feature extraction, dimension reduction is applied to select distinctive features and, finally, the system builds a predictive model on preprocessed data by training a machine learning algorithm. It uses this predictive model to classify previously unseen applications as malicious or benign.

3.1. Text Categorization

Text categorization methods are used in our proposed malware detection system. Text categorization is a subfield of text mining and it is used to automatically assign one of the predefined class labels to a document by using machine learning methods. Let $D = \{d_1, d_2, \ldots, d_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ represent document set and classes respectively. In text categorization, one category c_j is assigned to a document d_i and this process is generally performed by using features extracted from a document set. As the classes are predefined, text categorization is also considered as a supervised machine learning task [16].

3.2. Preprocessing and feature extraction

Each APK file includes the codes and resources (e.g., image and manifest files) of relevant application program. The Extensible Markup Language (XML) files included in these resources are used for installing and/or activating the application. One of the most important XML files is *AndroidManifest.xml* file. In order to run a component by the Android OS, it must be defined in this manifest file. Therefore, the manifest file must be included in application root, and components (e.g., Activities, Broadcast Receivers, Services, and Content Providers) must be defined in this file. This file must also include permission information (e.g., internet access or read access to user's contacts) required by the application, since declaring a permission implies intention to use related API [46].

Our system utilizes text categorization for malware detection and differently from previous studies, it solely focuses on the manifest content that is considered as a document. To do this, the manifest file of each application is obtained by decompressing and it is converted into a file with `txt` extension to extract textual features. Then, the punctuation marks are removed and lowercase conversion is applied over the contents. Next, bag of words (a.k.a bow), n-grams, and stylistic and structural properties (i.e., sstf) of the content are used as features, and performances of these well-known feature extraction methods are compared for the malware detection task. In addition, permission information are directly extracted from each application program file with the help of `ApkReader` tool[3] and each of the application's required permission is also used as a feature. In bow model, documents are tokenized by whitespaces and each token considered as a feature (i.e., word or term) which is used to represented the documents by assigning a weight to each term in the documents [24]. Assigning weight

[3]`https://github.com/hylander0/Iteedee.ApkReader`
[3]32 punctuation marks which are available in ASCII

to each term (i.e., feature) in each document is performed by using the observed frequency of the feature in the document. In n-gram model, on the other hands, features are generally obtained in two different levels: character and word levels. In character level n-gram, each feature is formed by taking the consecutive n-characters from the document, in word-level n-gram, each feature contains n adjacent words. However, we used the character level trigram features, since the character level is more successful than the word level in general [25]. Notice that we chose trigrams as they often akin to perform better than the bigrams [13]. The character level n-gram model is independent from language used to form the document, and it is strong against some cases as misspelling and abbreviation [32]. The sstf model is widely used in text mining based fields (e.g., author identification, text genre detection, music genre classification) to extract features (e.g., structural attributes and statistical information) from structured or semi-structured texts [38,50]. In this paper, we adopted 20 sstf features (see Table 2) for malware detection. We believe that it is the first time that stylistic and structural properties of the manifest content are utilized in AMD field. After the feature extraction, we transformed each manifest file into a numeric feature vector by applying term weighting methods. In term weighting phase, we employed BINARY and Augmented Normalized Term Frequency (ANTF) methods [43] to assign weights for permission-based and textual features respectively.

Table 2

Abbreviated (Abb.) names of the *sstf* features employed in our system.

Abb.	Feature	Abb.	Feature
WPLA	Words Per Line Average	NOEX	Number of Exclamation Mark
LLA	Line Length Average	NOE	Number of Ellipsis Mark
NOQ	Number of Question Mark	NOW	Number of Words
NODQ	Number of Double Quotation Mark	PR	Punctuation Ratio[4]
WLA	Word Length Average	NOH	Number of Hyphen Mark
VR	Vocabulary Richness	CPWA	Character Per Word Average
UWPL	Unique Word Per Line	NOC	Number of Comma Mark
CPWV	Character Per Word Variance	NOL	Number of Lines
NODV	Number of Decimal Value	NOUW	Number of Unique Words
WPLV	Words Per Line Variance	NOCL	Number of Colon Mark

The BINARY and ANTF methods are formulated as follows:

$$W_{BINARY}(p,m) = \begin{cases} 1, \text{if } m \text{ contains permission } p \\ 0, \textit{otherwise} \end{cases} \tag{1}$$

$$W_{ANTF}(c,m) = \frac{1 + TF(c,m)/max_{t \in m} TF(t,m)}{2} \tag{2}$$

In Eq. (1) and Eq. (2), TF represents raw term frequency of term (feature) c in a manifest content m, whereas p corresponds to a permission definition in m. The $max_{t \in m} TF(t,m)$ is also the maximum term frequency in related m.

3.3. Dimension reduction

We employed the Correlation based Feature Selection (CFS) method to select discriminative features and reduce feature space. CFS is a feature selection (FS) filter that ranks feature subsets in accordance with the relationship based on a heuristic function. This heuristic function works on the feature subsets which have high ranked correlation between the classes but does not include any correlation among each others [23]. We preferred to use this feature selector as it does not need to know the number of features to be selected. After selecting the most informative features we use only these selected features to perform classification task.

3.4. Classification

To perform classification, we borrow Naive Bayes (NB), Multinomial Naive Bayes (MNB), Support Vector Machine (SVM), Maximum Entropy (ME), and C4.5 classifiers from the machine learning field to investigate the effect of classifier on the performance of malware detection. We employ these methods as they are well-known and still commonly used classifiers in text categorization field [30].

- NB and MNB are practical methods [39] based on Bayes' probability theory which is utilized in many fields.
- C4.5 algorithm [41] is used to construct decision trees from the data.
- SVM is a kernel based classifier [14] and also robust to data sparsity.
- ME [15] supposes that features are interdependent unlike classifiers that use the Bayes theory.

4. Performance Evaluation

4.1. Datasets

In AMD field, there are many studies which are evaluated on different datasets for different purposes. Some of these datasets consist of only malware samples or various malware families, whereas some of them contain only benign samples. We also have a difficulty in this field that new malware samples arise constantly. In this paper, therefore, we selected the following datasets as evaluation material:

- DREBIN[5] dataset contains 5560 applications from 179 different malware families [4].
- AndroTracker[6] dataset includes 51179 benign and 4554 malware applications from different malware families [26].
- M0DROID (M0) dataset includes 400 samples in benign and malware categories that have equal distribution [17].

[5] `http://user.informatik.uni-goettingen.de/~darp/drebin/`
[6] `http://ocslab.hksecurity.net/andro-tracker`

The DREBIN dataset does not include any sample in benign category and the AndroTracker dataset is too large and unbalanced. The M0 dataset is balanced but its size is quite small when compared with other two datasets. In this study, our aim is to evaluate our system on balanced and different datasets that include both benign and malware samples. Therefore, we created two different datasets using the DREBIN and AndroTracker. In this way, we obtained a subset of AndroTracker (S-AT) by randomly choosing 5311 samples in benign category and whole samples in the malware category. By taking benign samples from the AndroTracker and malware samples from the DREBIN, we also created Drebin-AndroTracker (DRBN-AT) dataset which has 10000 samples equally distributed in two categories. In this process, our aim is to create small and balanced datasets as the size of the datasets is out of the scope of this paper. We use the M0 dataset as it is (without any change). Note that we have removed some application files from which any manifest file or permission information could not be extracted in decompilation process. This is because the reverse engineering tool[7] we used fails in some cases where it may require to find and install mobile OS specific framework files in order to properly decode application program files[8]. The number of samples before and after the decompressing in our selected datasets are given in Table 3.

Table 3
Distribution of the samples among two classes of the selected datasets before and after the decompilation (decompressing)

	# of samples			
Dataset	Before Decompilation		After Decompilation	
	Benign	Malware	Benign	Malware
M0	200	200	128	192
S-AT	5311	4554	3933	4421
DRBN-AT	5000	5000	3933	4637

4.2. Configuration

We utilized the CFS, NB, MNB, SVM, and C4.5 methods through `WEKA`[9] open source machine learning toolkit. At the time of writing, we used `WEKA` 3.6.1, but any version close to this will be sufficiently similar. We also employed the ME algorithm by using `MaxEnt` package[10] within the `OpenNLP`[11] which is a machine learning based toolkit for the processing of natural language text. For all the methods mentioned above, we used the default parameter settings, where kernel of the SVM is `linear`, search method of the CFS is `Best First`, and `confidenceFactor` of the C4.5 is 0.25. We trained

[7] `https://ibotpeaches.github.io/Apktool/`
[8] https://ibotpeaches.github.io/Apktool/documentation/
[9] `http://www.cs.waikato.ac.nz/ml/weka/`
[10] http://maxent.sourceforge.net/about.html
[11] https://opennlp.apache.org/

our ME model with the help of `generative iterative scaling` (GIS) algorithm, assuming 100 `iterations` and no `cutoff`. We validated performance of the classifiers with 10-fold cross validation [29] and evaluated the system for several parameters like accuracy (Acc), precision, recall, and F-measure [53].

4.3. Results

We conducted experiments on three different datasets and investigated the classification results for different classifier and feature set combinations. We also performed experiments for the cases that the feature set is obtained by feature selection and without feature selection. After decompilation of application files, textual features are extracted by using three different (bow, n-gram, sstf) models. Note that we applied preprocessing on manifest contents to remove in-comprehensive information (e.g., punctuation marks) in bow model. In addition, permission information are also extracted to be used as features. In this phase, new feature sets are created by combining these four feature sets with each other and this approach resulted in 10 different feature sets in total. Table 4 gives our four basic and six combined feature sets and their codes used in our experiments.

Table 4
Investigated feature groups and codes in our experiments

Feature Set	Code	Feature Set	Code
Structural and Statistical (SSTF)	SS	SS + BW	SB
BOW	BW	NG + SS	NS
Character level trigram	NG	PF + NG	PN
Permissions	PF	BW + PF	BP
BW + NG	BN	PF + SS	PS

After the feature extraction, the term weighting process is applied so that binary term weighting is used for permission based features, and ANTF weighting is applied for textual features. Afterwards, the datasets are converted into classification-ready structure. This task is also done by feature selection to explore its effect on results. Table 5 presents the number of unique features before and after the feature selection. Even though there are 135 permissions offered by the Android OS[12], developers could define new permissions depending upon the requirements of the application. In addition, the permissions which have the same features can be grouped under different names. Therefore, the number of permission features is greater than 135 in our study.

[12] `http://developer.android.com/reference/android/Manifest.permission.html`

Table 5
The number of unique features for each dataset before and after feature selection

Dataset	FS	Feature Set									
		PF	BW	NG	SS	BN	SB	NS	PN	BP	PS
M0	-	195	4105	3480	20	7585	4125	3500	3675	4300	215
	+	11	28	67	7	61	28	61	59	28	15
S-AT	-	493	35898	12184	20	48082	35918	12204	12677	36391	513
	+	17	54	99	8	71	50	89	54	40	16
DRBN-AT	-	563	38875	12854	20	51729	38895	12874	13417	39438	583
	+	15	51	70	4	67	51	63	54	55	15

As the final step, classification task is performed with the help of different classifiers to observe the effect of selected classifier on the performance of our system. Obtained results for the M0, S-AT, and DRBN-AT datasets are presented in Table 6, Table 7, and Table 8 respectively. The best result for each feature set is denoted in boldface, whereas underlined results represent the best results for the datasets. According to our results, the highest malware detection accuracy rates for the M0, S-AT, and DRBN-AT datasets are obtained as 94.0%, 99.3% and 98.2%, respectively. In Table 9, we also present weighted average values of evaluation measures for the best results on the datasets. The most successful feature model for each of the three datasets is bow model. The feature groups including the bow features (i.e., SB, BP and BN) are more successful than combined feature sets and the SVM generally outperforms other classifiers. It is observed that the feature selection process generally decreases classification success, even though it increases the success in some cases (see Table 6) especially for NB and MNB classifiers.

Our system has the highest success rate of 99.3% on S-AT dataset and this value varies depending on the classifier and feature set. The success rate of the sstf model reaches up to 92.2% (see Table 7) which is promising and quite important in terms of performance, especially when we consider its low feature space.

Table 6
Classification accuracies for the M0 dataset (%).

Classifier	FS	Feature Set									
		PF	BW	NG	SS	BN	SB	NS	PN	BP	PS
SVM	+	78.4	91.2	88.7	75.6	93.4	92.1	88.7	91.8	91.2	88.4
	-	90.0	**92.5**	88.4	**85.3**	92.5	**92.5**	88.7	90.3	**93.4**	**92.8**
ME	+	79.0	90.0	85.9	80.0	90.6	89.6	85.6	88.1	90.0	87.1
	-	**91.5**	85.3	84.0	82.5	84.6	85.9	83.4	85.3	85.9	89.0
C4.5	+	80.9	89.3	85.9	80.6	90.9	89.3	87.5	89.0	89.3	88.4
	-	88.4	87.8	83.1	83.4	86.8	87.8	80.3	85.0	88.7	87.1
NB	+	80.6	91.2	**90.6**	75.9	**<u>94.0</u>**	91.5	88.7	90.0	90.9	86.2
	-	75.6	89.6	86.2	75.9	89.6	89.6	86.5	86.8	89.6	79.6
MNB	+	78.7	91.5	90.0	60.0	**<u>94.0</u>**	91.5	**90.3**	**92.8**	92.8	74.3
	-	79.6	87.8	85.6	60.0	91.5	85.3	85.6	88.7	89.6	79.6

Table 7
Classification accuracies for the S-AT dataset (%).

Classifier	FS	Feature Set									
		PF	BW	NG	SS	BN	SB	NS	PN	BP	PS
SVM	+	88.2	96.3	92.3	83.3	97.5	96.5	94.7	92.8	95.7	90.7
	-	96.2	**99.1**	**95.8**	86.6	**99.3**	**99.2**	**96.3**	**97.7**	**99.3**	**96.7**
ME	+	88.3	96.3	92.6	86.9	96.7	96.2	92.9	92.2	93.5	90.8
	-	93.4	97.4	94.2	88.5	97.1	97.6	94.7	94.8	97.7	94.2
C4.5	+	87.8	97.1	93.3	91.9	97.7	97.0	93.6	92.6	96.6	91.2
	-	**96.8**	97.6	94.0	**92.2**	97.7	96.6	93.1	95.5	97.4	96.6
NB	+	87.8	93.6	87.6	78.7	93.9	93.9	86.5	80.6	91.9	90.0
	-	91.3	94.7	86.5	79.6	92.5	94.9	86.7	86.7	94.9	89.3
MNB	+	53.7	95.0	87.9	52.9	94.8	94.9	87.3	91.0	86.1	88.1
	-	91.9	97.9	89.8	52.9	95.6	98.0	89.8	91.6	97.4	92.0

Table 8
Classification accuracies for the DRBN-AT dataset (%)

Classifier	FS	Feature Set									
		PF	BW	NG	SS	BN	SB	NS	PN	BP	PS
SVM	+	92.3	91.7	85.4	79.2	92.4	91.7	89.0	93.0	93.8	92.3
	-	93.6	**98.0**	**93.6**	80.5	**97.7**	**97.9**	**94.2**	**97.1**	**98.2**	94.0
ME	+	92.4	91.9	88.1	82.0	92.7	91.9	89.2	93.5	93.5	92.1
	-	88.3	93.4	91.2	83.3	93.5	93.8	91.6	92.6	93.8	89.8
C4.5	+	92.1	93.5	88.6	86.6	93.6	93.5	90.4	93.7	95.4	92.4
	-	**94.9**	93.1	91.1	**87.2**	91.7	93.5	90.0	94.6	94.8	**95.0**
NB	+	91.8	86.4	76.5	76.4	88.8	86.4	76.9	72.3	88.1	91.5
	-	86.3	88.0	80.8	74.0	88.0	88.1	81.1	81.2	88.3	83.2
MNB	+	91.1	90.3	83.5	54.1	90.6	90.3	81.7	90.6	92.6	92.0
	-	88.0	94.3	84.5	54.1	90.9	94.3	84.5	88.3	94.2	85.8

Table 9
Evaluation metrics for the best results on evaluated datasets

Dataset	Classifier	Feature Set	Evaluation Metric			
			Precision	Recall	F-Measure	Acc (%)
S-AT	SVM	BN	0.99	0.99	0.99	**99.3**
	SVM	BP	0.99	0.99	0.99	**99.3**
DRBN-AT	SVM	BP	0.98	0.97	0.97	**98.2**
M0	MNB, NB	BN	0.93	0.91	0.91	**94.0**

To prove robustness and efficiency of our text mining based system, we also performed detailed feature analysis to investigate the distribution of feature sets in benign and malicious applications. We select the S-AT dataset for this analysis, as the best accuracies are obtained on this dataset. For our feature analysis, we selected the most observed 45 features (see Figure 2 and Figure 3) from bow and permission feature sets, respectively.

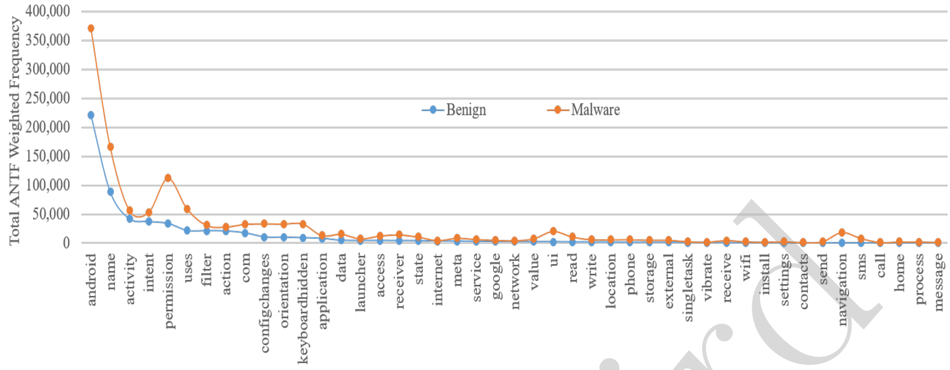

Figure 2. The most frequently observed *bow* features for the S-AT dataset.

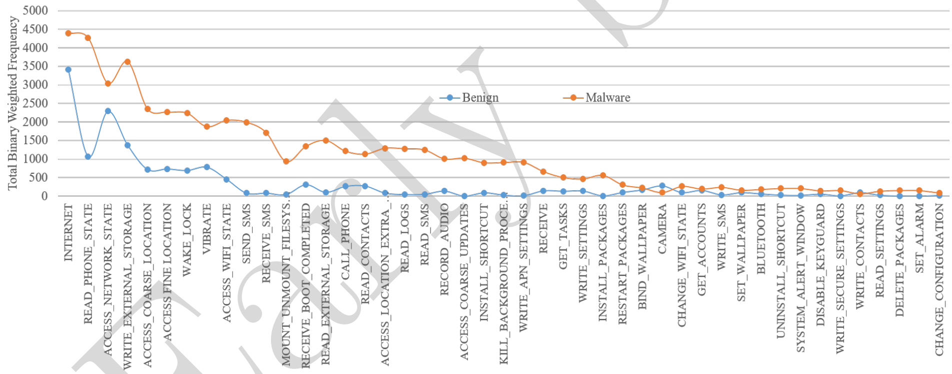

Figure 3. The most frequently observed *permission* features for the S-AT dataset.

As seen from the Figure 2, we found out that activity, intent, permission, and filter are defined more frequently in malware applications. It is also clear that there is a huge difference in frequently used features among benign and malware manifest contents. Figure 3 shows that malware applications mainly use some permissions (e.g., `INTERNET`, `READ_PHONE_STATE`) more frequently. We also observed that bow features mostly consist of tokenized permission names (e.g., `READ_PHONE_STATE` → "read', "phone", "state"). This proves that there is a relevance between bow and permission-based features. In addition, we investigated the total weights of the sstf features between malware and benign applications. In particular, we detected that NOW and CPWV are the most discriminative features (see Figure 4) in sstf model.

All of the sstf features in malicious applications have also higher total weight when compared to benign applications.

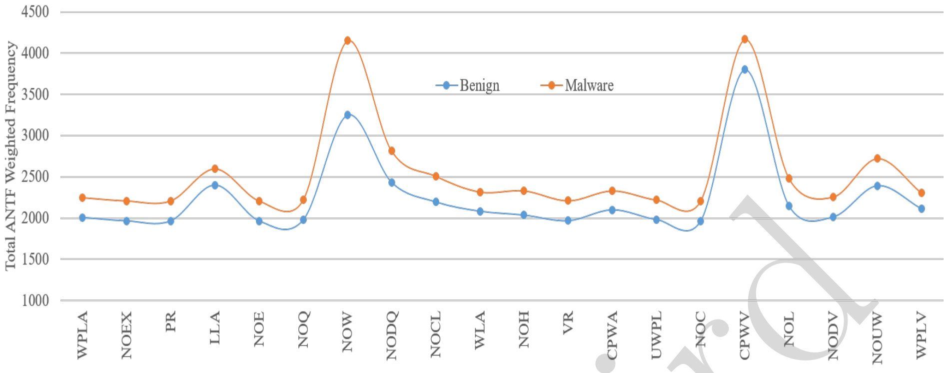

Figure 4. The weighted total frequencies of *sstf* features for the S-AT dataset.

5. Limitations and Future Work

Our system classifies an Android application as malware or benign by only analyzing its manifest content. However, it does not analyze other files that may contain malicious code. In some scenarios, it may not prevent an application by only processing its manifest content. But this application may still be a malware as it can download malicious code at runtime. The reason for this is that the designed system utilizes static analysis approach and lacks dynamic inspection. These issues can be mitigated by combining our system with a complete AMD system that uses hybrid approach. Table 10 supports this idea that our system could be an alternative, when compared to the some of other studies which use static approach.

Table 10
Comparison of the developed system with some of related works that use static approach.

Research Work	Features	# of APK		Acc(%)
		malware	benign	
MAMA [46]	Permissions, *uses-feature* tags	333	333	94.8
DREBIN [4]	Manifest, disassembled code features	5560	123453	93.9
PMDS [12]	Permissions	1450	1500	95.4
DroidMat [42]	Permissions, API calls, intent messages	238	1500	97.8
DroidAPIMiner [56]	API calls	3987	16000	99.0
Ours	Bow, n-grams, sstf, permissions	4421	3933	**99.3**

As there is need to collect and process big data in AMD, it is still not practical to employ most of the existing methods at mobile devices. The reason for this is that the mobile device has limitations on computing resources, processing capability, and memory storage. Therefore, we designed our system as a server-side system. But, we are planning to develop a mobile prototype software which will make it possible for the users to perform Android application analysis by sending the application file to our malware analyzer remote server. We plan to enable the developed system to analyze other source files by using text mining techniques and we will also try to evaluate our improved system on large-scale datasets.

6. Conclusion

In general, malicious application programs are generally given permissions by most Android based mobile device users in permission-based security model of Android, even though the users are warned in risky cases. Therefore, detecting the malwares is crucial before they are installed on the device. For this purpose, in this paper, we designed a system to detect Android malwares by using the static analysis approach based on text mining. Differently from existing works, our system constructs a feature vector for the software to be analyzed, and uses this feature vector to classify the software as Android malware or not by applying text classification methods. The features used in the classification process are mined only from the textual manifest content of the suspicious application. Different feature sets extracted from the manifest content are combined, then feature selection is applied to determine the most important features to be used. Also, several classification algorithms are employed to find out the most performant ones in terms of accuracy. Based on our experiments, we observed that combined feature sets containing bow features improve accuracy, especially when SVM is selected as classifier. Sstf features achieve promising results especially when utilized with C4.5 classifier, even though they generally provide less accuracy. Therefore, we suggest to use the sstf features to analyze not only manifest content but also other source files.

Textual features are generally more successful than permission-based features. On the other hand, manifest files of malware applications have richer content when compared to the benign applications. Our analysis over the S-AT dataset discovered that malware and benign applications have average number of 13 and 6 permissions, respectively. It shows that if a manifest content of an application contains a large permission list it is most likely a malware. Even majority of permissions are innocuous, they may be included into manifest file to lead users to overlook other permissions which can be used in malwares for different purposes. What is more, our experiments prove that bow and n-gram features provide effective identification and detection of malicious applications. Feature selection process decreases the accuracy in general. We think that main reason for this is that bow and n-gram models produce sparse representation of data. Therefore, in some cases, features selected by the CFS algorithm may not highly correlated with classes, especially on large and sparse datasets. Additionally, selected features may not be observed in test samples. The classifier employed with the CFS method also affects the accuracy of malware detection. Consequently, we conclude that our work provides a promising basis for future studies in context of malware detection by using text mining techniques. This also demonstrates that developed system is capable of being used as part of a complete static, dynamic or hybrid AMD system.

References

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

[2] Alam S., Qu Z., Riley R., Chen Y., Rastogi V.: DroidNative: Automating and optimizing detection of Android native code malware variants. In: *computers & security*, vol. 65, pp. 230–246, 2017.

[3] Amamra A., Talhi C., Robert J.M.: Smartphone malware detection: From a survey towards taxonomy. In: *2012 7th International Conference on Malicious and Unwanted Software*, pp. 79–86. IEEE, 2012.

[4] Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K., Siemens C.: Drebin: Effective and explainable detection of android malware in your pocket. In: *Ndss*, vol. 14, pp. 23–26. 2014.

[5] Arshad S., Shah M.A., Khan A., Ahmed M.: Android malware detection & protection: a survey. In: *Int. J. Adv. Comput. Sci. Appl*, vol. 7(2), pp. 463–475, 2016.

[6] Arslan R.S., Doğru İ.A., Barişçi N.: Permission-Based Malware Detection System for Android Using Machine Learning Techniques. In: *International Journal of Software Engineering and Knowledge Engineering*, vol. 29(01), pp. 43–61, 2019.

[7] Barsiya T.K., Gyanchandani M., Wadhwani R.: Android Malware Analysis: A Survey Paper. In: *International Journal of Control, Automation, Communication and Systems (IJCACS)*, vol. 1(1), pp. 35–42, 2016.

[8] Bhattacharya A., Goswami R.T.: DMDAM: data mining based detection of android malware. In: *Proceedings of the First International Conference on Intelligent Computing and Communication*, pp. 187–194. Springer, 2017.

[9] Burguera I., Zurutuza U., Nadjm-Tehrani S.: Crowdroid: behavior-based malware detection system for android. In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 15–26. ACM, 2011.

[10] Cai H., Meng N., Ryder B., Yao D.: Droidcat: Unified dynamic detection of Android malware. Tech. rep., Department of Computer Science, Virginia Polytechnic Institute & State, 2016.

[11] Chakradeo S., Reaves B., Traynor P., Enck W.: Mast: Triage for market-scale mobile malware analysis. In: *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pp. 13–24. ACM, 2013.

[12] Chan P.P., Song W.K.: Static detection of Android malware by using permissions and API calls. In: *2014 International Conference on Machine Learning and Cybernetics*, vol. 1, pp. 82–87. IEEE, 2014.

[13] Çoban Ö., Özyer B., Özyer G.T.: A comparison of similarity metrics for sentiment analysis on Turkish twitter feeds. In: *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 333–338. IEEE, 2015.

[14] Cortes C., Vapnik V.: Support-vector networks. In: *Machine learning*, vol. 20(3), pp. 273–297, 1995.

[15] Cuong N.V., Ha N.T.T.L., Thuy Q., Hieu P.X.: A Maximum Entropy Model for Text Classification. In: *The International Conference on Internet Information Retrieval 2006*, pp. 134–139. 2006.

[16] Dalal M.K., Zaveri M.A.: Automatic text classification: a technical review. In: *International Journal of Computer Applications*, vol. 28(2), pp. 37–40, 2011.

[17] Damshenas M., Dehghantanha A., Choo K.K.R., Mahmud R.: M0droid: An android behavioral-based malware detection model. In: *Journal of Information Privacy and Security*, vol. 11(3), pp. 141–157, 2015.

[18] Dash S.K., Suarez-Tangil G., Khan S., Tam K., Ahmadi M., Kinder J., Cavallaro L.: Droidscribe: Classifying android malware based on runtime behavior. In: *2016 IEEE Security and Privacy Workshops (SPW)*, pp. 252–261. IEEE, 2016.

[19] Enck W., Gilbert P., Han S., Tendulkar V., Chun B.G., Cox L.P., Jung J., McDaniel P., Sheth A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: *ACM Transactions on Computer Systems (TOCS)*, vol. 32(2), p. 5, 2014.

[20] Faruki P., Ganmoor V., Laxmi V., Gaur M.S., Bharmal A.: AndroSimilar: robust statistical feature signature for Android malware detection. In: *Proceedings of the 6th International Conference on Security of Information and Networks*, pp. 152–159. ACM, 2013.

[21] Felt A.P., Finifter M., Chin E., Hanna S., Wagner D.: A survey of mobile malware in the wild. In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 3–14. ACM, 2011.

[22] Fereidooni H., Conti M., Yao D., Sperduti A.: ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In: *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5. IEEE, 2016.

[23] Hall M.A.: *Correlation-based feature selection for machine learning*. Ph.D. thesis, University of Waikato Hamilton, New Zealand, 1999.

[24] Joachims T.: A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Tech. rep., Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[25] Kanaris I., Kanaris K., Houvardas I., Stamatatos E.: Words versus character n-grams for anti-spam filtering. In: *International Journal on Artificial Intelligence Tools*, vol. 16(06), pp. 1047–1067, 2007.

[26] Kang H.J., Jang J.w., Mohaisen A., Kim H.K.: Androtracker: Creator information based android malware classification system. In: *Information Security Applications-15th International Workshop, WISA*, vol. 8909. 2014.

[27] Kim J., Yoon Y., Yi K., Shin J., Center S.: ScanDal: Static analyzer for detecting privacy leaks in android applications. In: *MoST*, vol. 12, p. 110, 2012.

[28] Kim T., Kang B., Rho M., Sezer S., Im E.G.: A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. In: *IEEE Transactions on Information Forensics and Security*, vol. 14(3), pp. 773–788, 2019.

[29] Kohavi R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Ijcai*, vol. 14, pp. 1137–1145. Montreal, Canada, 1995.

[30] Kowsari K., Jafari Meimandi K., Heidarysafa M., Mendu S., Barnes L., Brown D.: Text classification algorithms: A survey. In: *Information*, vol. 10(4), p. 150, 2019.

[31] Lindorfer M., Neugschwandtner M., Weichselbaum L., Fratantonio Y., Van Der Veen V., Platzer C.: Andrubis–1,000,000 apps later: A view on current Android malware behaviors. In: *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pp. 3–17. IEEE, 2014.

[32] Lodhi H., Saunders C., Shawe-Taylor J., Cristianini N., Watkins C.: Text classification using string kernels. In: *Journal of Machine Learning Research*, vol. 2(Feb), pp. 419–444, 2002.

[33] Malhotra A., Bajaj K.: A hybrid pattern based text mining approach for malware detection using DBScan. In: *CSI transactions on ICT*, vol. 4(2-4), pp. 141–149, 2016.

[34] Malhotra A., Bajaj K.: A survey on various malware detection techniques on mobile platform. In: *Int J Comput Appl*, vol. 139(5), pp. 15–20, 2016.

[35] Martín A., Calleja A., Menéndez H.D., Tapiador J., Camacho D.: ADROIT: Android malware detection using meta-information. In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8. IEEE, 2016.

[36] Martín A., Lara-Cabrera R., Camacho D.: Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset. In: *Information Fusion*, vol. 52, pp. 128–142, 2019.

[37] Mas' ud M.Z., Sahib S., Abdollah M.F., Selamat S.R., Huoy C.Y.: A Comparative Study on Feature Selection Method for N-gram Mobile Malware Detection. In: *IJ Network Security*, vol. 19(5), pp. 727–733, 2017.

[38] Mayer R., Neumayer R., Rauber A.: Combination of audio and lyrics features for genre classification in digital audio collections. In: *Proceedings of the 16th ACM international conference on Multimedia*, pp. 159–168. ACM, 2008.

[39] McCallum A., Nigam K., et al.: A comparison of event models for naive bayes text classification. In: *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48. Citeseer, 1998.

[40] Milosevic N., Dehghantanha A., Choo K.K.R.: Machine learning aided Android malware classification. In: *Computers & Electrical Engineering*, vol. 61, pp. 266–274, 2017.

[41] Quinlan J.R.: *C4. 5: programs for machine learning.* Elsevier, 2014.

[42] Rovelli P., Vigfússon Ý.: Pmds: Permission-based malware detection system. In: *International Conference on Information Systems Security*, pp. 338–357. Springer, 2014.

[43] Salton G., Buckley C.: Term-weighting approaches in automatic text retrieval. In: *Information processing & management*, vol. 24(5), pp. 513–523, 1988.

[44] Santos I., Penya Y.K., Devesa J., Bringas P.G.: N-grams-based File Signatures for Malware Detection. In: *ICEIS (2)*, vol. 9, pp. 317–320, 2009.

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

[46] Sanz B., Santos I., Laorden C., Ugarte-Pedrero X., Nieves J., Bringas P.G., Álvarez Marañón G.: MAMA: manifest analysis for malware detection in android. In: *Cybernetics and Systems*, vol. 44(6-7), pp. 469–488, 2013.

[47] Shabtai A., Fledel Y., Elovici Y.: Automated static code analysis for classifying android applications using machine learning. In: *2010 International Conference on Computational Intelligence and Security*, pp. 329–333. IEEE, 2010.

[48] Shabtai A., Kanonov U., Elovici Y., Glezer C., Weiss Y.: Andromaly: a behavioral malware detection framework for android devices. In: *Journal of Intelligent Information Systems*, vol. 38(1), pp. 161–190, 2012.

[49] Spreitzenbarth M., Freiling F., Echtler F., Schreck T., Hoffmann J.: Mobile-sandbox: having a deeper look into android applications. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1808–1815. ACM, 2013.

[50] Stamatatos E., Fakotakis N., Kokkinakis G.: Automatic text categorization in terms of genre and author. In: *Computational linguistics*, vol. 26(4), pp. 471–495, 2000.

[51] Suarez-Tangil G., Tapiador J.E., Peris-Lopez P., Blasco J.: Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. In: *Expert Systems with Applications*, vol. 41(4), pp. 1104–1117, 2014.

[52] Sun M., Li X., Lui J.C., Ma R.T., Liang Z.: Monet: a user-oriented behavior-based malware variants detection system for android. In: *IEEE Transactions on Information Forensics and Security*, vol. 12(5), pp. 1103–1112, 2017.

[53] Varsha M., Vinod P., Dhanya K.: Heterogeneous feature space for Android malware detection. In: *2015 Eighth International Conference on Contemporary Computing (IC3)*, pp. 383–388. IEEE, 2015.

[54] Wang S., Yan Q., Chen Z., Yang B., Zhao C., Conti M.: TextDroid: Semantics-based detection of mobile malware using network flows. In: *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 18–23. IEEE, 2017.

[55] Wang S., Yan Q., Chen Z., Yang B., Zhao C., Conti M.: Detecting android malware leveraging text semantics of network flows. In: *IEEE Transactions on Information Forensics and Security*, vol. 13(5), pp. 1096–1109, 2018.

[56] Wu D.J., Mao C.H., Wei T.E., Lee H.M., Wu K.P.: Droidmat: Android malware detection through manifest and api calls tracing. In: *2012 Seventh Asia Joint Conference on Information Security*, pp. 62–69. IEEE, 2012.

[57] Yan L.K., Yin H.: DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. In: *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pp. 569–584. 2012.

[58] Zhang S., Xiao X.: CSCdroid: Accurately Detect Android Malware via Contribution-Level-Based System Call Categorization. In: *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 193–200. IEEE, 2017.

[59] Zhao M., Ge F., Zhang T., Yuan Z.: Antimaldroid: An efficient svm-based malware detection framework for android. In: *International Conference on Information Computing and Applications*, pp. 158–166. Springer, 2011.

[60] Zheng M., Sun M., Lui J.C.: Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 163–171. IEEE, 2013.

[61] Zhou Y., Wang Z., Zhou W., Jiang X.: Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: *NDSS*, vol. 25, pp. 50–52. 2012.

Affiliations

Önder Çoban
Department of Computer Engineering, Cukurova University, Adana, Turkey,
ocoban@cu.edu.tr, ORCID ID: https://orcid.org/0000-0001-9404-2583

Selma Ayşe Özel
Department of Computer Engineering, Cukurova University, Adana, Turkey,
saozel@cu.edu.tr, ORCID ID: https://orcid.org/0000-0001-9201-6349

Received: 8.05.2019
Revised: 8.05.2019
Accepted: 10.08.2019