Android malware detection based on overlapping of static features

Maryam Nezhadkamali  
Department of Computer Engineering,  
Ferdowsi University of Mashhad,  
Mashhad, Iran  
zehadkamali@mail.um.ac.ir

Somayeh Soltani  
Department of Computer Engineering,  
Ferdowsi University of Mashhad,  
Mashhad, Iran  
somayeh.soltani@mail.um.ac.ir

Seyed Amin Hosseini Seno  
Department of Computer Engineering,  
Ferdowsi University of Mashhad,  
Mashhad, Iran  
hosseini@um.ac.ir

Abstract—Smartphones are increasingly used in everyday life. They execute complex software and store sensitive and private data of users. At the same time, malware targeting mobile devices is growing. There are various Android malware detection methods in the literature, most of which are based on permissions. However, the permission-based methods are usually subverted by some bypass techniques such as over-claim of permissions, permission escalation attack, and zero permission attack. In this paper, an Android malware detection method is proposed which uses API functions and Intents besides permissions. The proposed method modifies the values of some overlapping features. Consequently, the evaluation metrics such as precision, true positive, and false positive and accuracy are improved. The precision of the proposed method increases to 99.7% and the accuracy of this method improved to 98.6%.

Keywords—Android Malware; permission; API function; Intent; Classification; Overlapping Features

I. INTRODUCTION

Android is a mobile operating system which is designed by Google Co. for various devices such as tablets, smartphones, smart watches, and televisions. Nowadays, Google's Android has become the most popular operating system in the world [1]. Therefore, the security of Android is very important, and it is addressed by many researchers.

Malware is a malicious software which tries to damage some parts of system. Malware tries to install itself on the system via different techniques such as spams and social engineering. Then it uses existing vulnerabilities to do its subversive activities. Nowadays, the number of Android malware has grown massively [2, 3]. The most important reasons for this growth are as follows: 1) reduced smartphone prices, 2) the open source kernel of android, 3) easy access to Internet, 4) storing private and sensitive data on smartphones, and 5) the similarity of programming software for PCs and smartphones [4].

In order to detect malware on smartphones, many researchers attempt to propose several techniques. Malware detection approaches are classified into two category: 1) anomaly-based detection, and 2) signature-based detection. Signature-based detection approach creates an individual signature of malware, and stores the signatures of various types of malware in a database. In order to identify an unknown malware, the signature-based approach compares the signature of the unknown malware with the database of known malware. On the other hand, the anomaly-based detection approach prepares a profile of normal behaviors of system. Any deviations from this profile are considered as abnormal behaviors. Both of anomaly-based detection and signature-based approaches use static and/or dynamic methods to extract data. While the static method extracts data from the source code of the malware, the dynamic method extracts data from program execution traces [5-12].

It should be noted that Android itself has security mechanisms. Some of the security mechanisms of Android are as follows: 1) isolated sandboxes for each program, 2) system file encryption, and 3) declaring permission to access to resources [1].

Generally the security model of Android relies on permission-based mechanism. In order to access to protected resources, each program in the permission-based mechanism declares permissions in AndroidManifest.xml. Permissions are requested by programs during the install process. If user accepts all permissions, program will be installed. The shortcoming of the security modes of Android is that the permissions are just shown during the installation process, and Android does not block any program. Unfortunately, as stated by Felt et al. [8], only 17% of users notice these permissions, and 42% of these users are unaware of them.

The capabilities of program such as sending and receiving SMSs, and using Bluetooth and Location are hidden in permissions. Therefore, permissions are valuable resources to identify the behavior of malware. Many researchers [5, 6, 8-10, 12-16] propose methods to detect Android malware using permissions. However, these methods are subverted by some bypass techniques such as permission escalation attack, over-claim of permissions, and zero permission attack.

This paper intends to overcome the shortcomings of the previous works. The proposed method develops a more accurate method using three feature sets including permissions, API functions, and Intents. In this regard, it defines a new concept, named feature pocket. A feature pocket is created when two or more features from separate feature sets overlap with each other, i.e. are defined on the same resource.
II. RELATED WORK

Malware detection approaches are classified into two primary classes: Anomaly-based detection and signature-based detection. Both of these approaches can extract features statically or dynamically. Static features focus on features available in the source code of applications such as AndroidManifest.xml file and Classes.dex file. Dynamic features are extracted during runtime of application. Since the proposed method is a signature-based one, the rest of this section describes some of related work in this domain.

A. Static Signature-based Detection methods

Sharma et al. [7] have extracted permissions and API functions of APK files. They have selected most common used API functions to classify applications. Their proposed method applies the classification algorithms of KNN and Naïve Bayesian. Naïve Bayesian classifier with Information Gain method results in better True Positive Rate.

Liu et al. [5] present a two-layered permission-based detection method. A pair is defined as combination of any two requested permissions. For example, WRITE_SMS and RECEIVE_SMS permissions make a pair. In order to have a trade-off between accuracy and speed, their proposed system works in two layers. In the first layer, a fast detection using the requested permissions is executed. In order to classify the applications which cannot be classified in the first layer, the second layer uses used permissions.

In order to classify Android applications, the method presented in [9] extracts permissions and package names of the application. The permissions and package information of the application are extracted from the manifest and the DEX files. This work uses four classification algorithms including Byes, KNN, Linear Discriminant Function, and RBF Network. It concludes that using both the package information and permissions increases the accuracy of the detection.

Apposcopy [11] is a semantic-based approach for detecting Android malware. Using ICC, Apposcopy describes semantic characteristics of different families of Android malware. It can decide if an application matches the signature of a malware family. Apposcopy achieves the accuracy of 90% over all analyzed malware instances.

Sanz et al. [17] analyze manifest file in order to detect malware. In this regard, multiple feature vectors such as permissions are constructed. Moreover, some other features in the manifest file such as Android:name, Android:Required, and gleVersion are extracted. These feature vectors are binary arrays, in which the value 1 means that the feature is available in the manifest file. Then several supervised machine learning algorithms such as KNN, J48, Random Forest, and Bayesian classification algorithms are used. Experiments show that Random forest classification has the best result.

Moonsamy et al. [18] have used a hierarchical bi-clustering method and a contrast permission pattern mining algorithm to distinguish malware from benign applications.

The method proposed in [12] uses ensemble learning for detecting Android malware. Various features including API calls, commands, and permissions are used in this work. This approach can improve detection rates to 97%.

The method presented in [13] extracts API functions, permissions and dangerous Linux commands. In order to classify applications, it uses the Bayesian classification algorithm. The TPR of this method is 0.9.

DroidMat [6] is a permission-based mechanism which provides a static method for detection of the Android malware. In order to recognize the behavior of Android malware, this work uses the static features including permissions, components, Intent Filters, and API calls. It uses both K-means and KNN to classify the applications as malware or benign. The accuracy of DroidMat reaches 97.87%.

Drebin [16] is a lightweight Android malware detection method which collects many features from an Android application. It extracts various feature sets including hardware components, requested permissions, app components, and intents filters from AndroidManifest.xml. Moreover, it constructs some other feature sets including restricted API calls, used permissions, suspicious API calls, and network addresses. It uses linear Support Vector Machine for classifying applications as benign or malware.

Perhaps the most similar works to the proposed method in this paper are DroidMat [6] and Derbin [16]. However, these methods consider Intent Filters as features, and they do not use Intents. Moreover, these methods are vulnerable to permission-based attacks.

B. Dynamic Signature-based Detection methods

Iker Burguera et al. [15] propose a framework which detects the behavior of malware. The first component of this framework is a client application called Crowdroid. This component monitors Linux Kernel system calls, preprocesses them, and sends them to a centralized server. The second component is a remote server which parses data, and creates a system call vector for each interaction of users. Therefore, for each application, a dataset of behavior data will be created. Finally, the system applies clusters algorithm on each dataset.

The malware detection method in [19] is a rule-based classifier which is based on features of network traffic. However, this method is only capable of detection of those sets of malware which have network connectivity properties. The accuracy of this classifier is reasonably acceptable.

C. Anomaly-based detection methods

The Android malware detection method in [20] is a static analysis on the executables. First, the function calls of the executables are extracted using the readelf command. Next, these function calls are compared with function calls of Linux malware executables.

Andromaly [21] is a behavioral Android malware detection. In order to classify the applications as malware or benign, Andromaly applies various learning algorithms such as Bayesian Networks, and Logistic Regression. Moreover, in order to find the most representative sets of features, it applies different feature selection methods.
Table I compares various Android malware detection techniques presented in this section. Different properties of these techniques including type of detection, method, and feature list are discussed.

### TABLE I. COMPARISON OF VARIOUS ANDROID MALWARE DETECTION TECHNIQUES

<table>
<thead>
<tr>
<th>Article</th>
<th>Approach</th>
<th>Detection Type</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>[5]</td>
<td>Signature</td>
<td>Static</td>
<td>✓</td>
</tr>
<tr>
<td>[6]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[7]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[9]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[12]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[13]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[17]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[18]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[16]</td>
<td>Signature</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[15]</td>
<td>Signature</td>
<td>Dynamic</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[19]</td>
<td>Signature</td>
<td>Dynamic</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[20]</td>
<td>Anomaly</td>
<td>Static</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>[21]</td>
<td>Anomaly</td>
<td>Dynamic</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

III. PRELIMINARIES

A. Basic Definitions

In this section some basic definitions of features of Android which are used in this paper are presented.

According to Android documentation [24], “A permission is a restriction that limits access to a part of the code or to data on the device. The limitation is imposed to protect critical data and code that could be misused to distort or damage the user experience.”

Moreover, according to Android documentation [25], “An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases: starting an activity, starting a service, and delivering a broadcast.”

Implicit and explicit Intents are two primary forms of Intents. While the explicit Intents are used for message passing between components of an application, the implicit Intents allow to interact with components of other applications.

The importance of Intents is that an application which uses them can access to some critical resources without requesting the corresponding permissions. Two primary pieces of information in an Intent are Action and Data. The proposed method uses both of these parts.

B. Permission-Based Attacks

In this section, some permission-based attacks are described [22, 23].

1) Permission escalation attack

Permission escalation allows malware to access to sensitive resources without requesting corresponding permissions directly. In order to get more privilege, malware uses vulnerabilities in unprotected interfaces of privileged applications. As shown in Fig.1, the application A1 is not permitted to use resource R1 which is protected by permission P1. So C1, which is a component of A1, cannot directly access to R1. However, C1 can transitively access to R1 if application A2 has declared permission P1.

![Fig. 1. Permission escalation attack [22]](image)

2) Over-claim of permissions

This type of attack is the most serious threat to Android environment. In order to bypass permission-based detection methods, malware requests permissions more than needed. On the other hand, application with over-claim of permissions is also vulnerable to permission escalation attack. For example, if an independent application requests the unnecessary SEND_SMS permission, the permission can be exploited by malware to use SMS service.

3) Zero permission attack

In this type of attack, malware does not request any permissions. Malware can access to protected resources using applications which have the necessary permissions and accept Intents. For example, the common way to access to Internet is requesting the INTERNET permission. However, malware can bypass this normal process, and access to Internet via a browser application which accepts Intents.

IV. THE PROPOSED MALWARE DETECTION METHOD

Malware detection using just one feature set is not appropriate since it can be bypassed by malware. For example, as stated in the previous section, permission-based detection methods are subverted by permission escalation attack, over-claim of permissions and zero permission attack. Therefore, the proposed method in this paper uses three different feature sets including permissions, API functions, and Intents. It defines a new concept, named feature pocket.

A feature pocket is created when two or more features from separate feature sets overlap with each other, i.e. have the same resource. For example, three features including sendTextMessage API function, Action_Send Intent with sms keyword, and SEND_SMS permission have the same resource, means they have overlap, and thus, a feature pocket including these three features is created. Features in feature pocket take effect from each other and can improve permissions. Fig. 2 illustrates the concept of feature pocket. Table II shows some of feature pockets.
Features in the same feature pocket may change the feature value of each other. For example, in order to bypass permission-based detection mechanism, some malware does not request **SEND_SMS** permission. Therefore, the value of this permission in the feature vector of malware is 0. However, if malware calls **sendTextMessage** API function or builds **Action_Send** Intent with **sms** keyword, the value of **SEND_SMS** permission will be set.

The suggested method works in four phases: 1) dataset creation, 2) feature refinement, 3) feature selection, and 4) benign/malware classification.

The purpose of dataset creation phase is to extract various features including permissions, API functions, and Intents. The feature refinement phase consists of two steps: a) removing extra features, which removes some of rarely used features, and b) modifying feature values, which modifies the values of features within the same feature pocket. The feature selection phase selects a subset of relevant features. The benign/malware classification phase uses various machine learning techniques including SVM, Random Forest, and Decision Tree. The purpose of this phase is to classify various applications as benign or malware. Fig. 3 shows the architecture of the proposed method.

### V. IMPLEMENTATION

In order to implement the proposed detection method, Python 2.7 programming language and Santoku Linux platform [26] are used. In this section the implementation details of four phases of the proposed method are described.

#### A. Dataset Creation

The MalGenome [27] which is an Android malware database is used for dataset creation. It consists of 49 different malware families and 1260 malware applications. Moreover, a collection of 498 benign applications from 27 families is used for dataset creation.

The main purpose of this phase is to extract three feature sets including permissions, API functions, and Intents. The Androguard [28] reverse engineering tool is used for this purpose.

Permissions are extracted from **AndroidManifest.xml**, which is in the root directory of each APK file. According to Android documentation [29], there are 160 permissions in API Level 19.

In order to access to API functions, at first, **Classes.dex** is extracted from APK file. Then **classes.dex** is converted to JAR file which includes multiple .Class files. Finally, API functions are extracted from these .Class files. The process of extraction of API functions is illustrated in Fig. 4. In this work, 130 API functions from the most common packages are selected.
The extraction process of Intents is the same as API functions. In this work, 29 Actions of Intents and 13 Data of Intents are used.

B. Feature Refinement
The feature refinement phase consists of two steps: a) removing extra features, which removes some of rarely used features, and b) modifying feature values, which modifies the values of features within the same feature pocket.

C. Feature Selection
The feature selection phase selects a subset of relevant features. Three feature selection algorithms which are used in the proposed method are as follows:
- L1-based feature selection
- Information gain feature selection
- Gini impurity feature selection

D. Benign/Malware Classification
The classification phase uses various machine learning techniques to classify applications. SVM, Random Forest, and Decision Tree are used in this experiment. In order to find the impact of feature refinement and feature selection, these algorithms are applied two times: before feature refinement and feature selection and after these phases. The results show the improvement of achieved metrics. In the evaluation section, the results are detailed.

VI. EVALUATION
In this section, the evaluation metrics and the achieved results are discussed.

A. Evaluation Metrics
In order to evaluate the detection method, several evaluation metrics are used.
- True Positive (TP): the number of malicious applications which correctly classified as positive.
- False Positive (FP): the number of benign applications which incorrectly classified as positive.
- False Negative (FN): the number of malicious applications which incorrectly classified as negative.
- True Negative (TN): the number of benign applications which correctly classified as negative.

Additional metrics are defined based on TP, FP, TN, and FN.

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (1)
\]

\[
\text{Accuracy} = \frac{TP + TN}{TN + FN + TP + FP} \quad (2)
\]

\[
\text{Recall} = \frac{TP}{FN + TP} \quad (3)
\]

\[
F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)
\]

B. Evaluation Results
In this section the results of the implementation of the proposed method are presented. In order to show the improvement achieved through using Intents and API functions, three feature vectors are constructed. These feature vectors are as follows:
- Feature Vector 1 (FV1): It contains permissions only.
- Feature Vector 2 (FV2): It contains permissions and API functions.
- Feature Vector 3 (FV3): It contains permissions, API functions, and Intents.

The classification algorithms including SVM, Random Forest, and Decision Tree are applied on the constructed dataset. In order to find the impact of feature refinement and feature selection, these classification algorithms are used two times: before feature refinement and selection and after these phases. The results show the improvement of achieved accuracy after feature refinement and feature selection.

Fig. 5 shows the accuracy achieved by running various classification methods before applying feature refinement and feature selection. As shown in this figure, FV1 results in the worst accuracy for all of the classification techniques. The results also show the improvement achieved by adding new features including API functions and Intents. However, before applying feature refinement and feature selection, FV3 acts worse than FV2.

![Fig. 5. Achieved accuracy before applying feature refinement and feature selection](image)

Fig. 6 shows the accuracy achieved by running various classification methods after applying feature refinement and feature selection. As shown in this figure, FV1 results in the worst accuracy for all of the classification techniques. The results also show the improvement achieved by adding new features including API functions and Intents. However, before applying feature refinement and feature selection, FV3 acts worse than FV2.

![Additional features including API functions and Intents](image)
The proposed method using Random Forest as classification algorithm and Information Gain as feature selection algorithm results in 98.6% accuracy. The evaluation metrics achieved by the proposed method using Random Forest and Information Gain are illustrated in Table III.

Table IV compares the proposed method with related works, i.e. DroidMat [6] and the method in [5, 7]. As shown in Table IV, the proposed method results in better evaluation metrics compared to three relate works, and it improves all of evaluation metrics.

![Accuracy vs. Feature Selection Methods](image1.png)

**Fig. 6.** Achieved accuracy after applying feature refinement and feature selection

![Accuracy vs. Feature Selection Methods](image2.png)

**Fig. 7.** Accuracy achieved by running different feature selection and classification algorithms.

Several researchers try to detect Android malware. However, most of the proposed methods are simply subverted by malware. In this paper, an Android malware detection method has been proposed which leverages different features such as permissions, API functions, and Intents. The proposed method modifies the values of some overlapping features. Therefore, the achieved precision increases to 99.7% and the achieved accuracy improved to 98.6%.

### REFERENCES


