A machine learning approach for detecting and categorizing evasion sources in Android malware

Hasan Deeb 1, Hayyan Hasan2,3, Behrouz Tork Ladani2, and Bahman Zamani2
1 Albaath University, Faculty of Informatics Engineering.
2 University of Isfahan, Faculty of Computer Engineering, MDSE Research Group.
3 Albaath University, Faculty of Mechanical and Electrical Engineering.

Evasion techniques are used by some Android malware to hide their malicious behavior and to hinder their execution during the dynamic analysis process. Many tools tackle such evasions by using a manually created list of API functions (as sources of evasions) to detect these evasions. As an important consequence, no matter how good the tool is, it can only guarantee to defeat these evasions and extract the real behavior of the malware if its list of evasion sources is complete. This way, if some evasion sources are missing from the list or when similar API functions are used, the dynamic analysis can be hindered.

In this paper, we propose a machine learning approach to detect and categorize various evasion sources in Android malware. The proposed approach uses a manually collected training dataset to train two classifiers. The first classifier is used to detect the evasion nature of the Android API methods, while the second classifier is used to categorize the detected evasion sources into predefined categories. We applied the proposed approach to a large number of methods extracted from Android API 27. The proposed approach could detect hundreds of evasions with accuracy of 92.8% for the first classifier and 90.5% for the second classifier. The evaluation for 500 real-world samples showed that many of the evasions are detected by our approach, are not considered by the state-of-the-art dynamic analysis frameworks that are indeed used by malware samples.

Index Terms—Evasion techniques, Dynamic analysis, Machine learning.

I. INTRODUCTION

Android OS is a dominant mobile operating system with a market share of 83.8% in 2021 [1]. From the one side, this is a motivation for malware applications to appear and grow. On the other side, there is a significant endeavor to find ways to detect and analyze these malware applications. In general, an analysis process is used to extract the required information and to detect Android malware. Android malware (and Android applications in general) can be analyzed using three approaches. Static analysis that extracts the information from the sample under analysis without running it [2][3]. Dynamic analysis that needs to run the sample in order to extract its real behavior [4][5][6][7], and hybrid analysis that includes using both static and dynamic analysis [8][9].

Dynamic analysis is an approach that runs the sample inside a test environment to extract its real behavior. There are several evasion techniques used by Android malware to hinder the dynamic analysis process. Also, researchers proposed many approaches to defeat these evasions [10] [11]. To this end, these approaches mainly use manually created lists of evasion sources to detect and hence defeat the evasions. As an important consequence, the effectiveness of these approaches is based on the completeness of the evasion source lists. In other words, if the list is not complete and does not contain every possible evasion sources, the dynamic analysis achieved by these approaches can be hindered. Hence, they may not be able to reach the target payload and extract the real behavior of the sample under analysis.

This work focuses on evasion techniques in Android malware. Dynamic analysis approaches always use some fixed lists to detect and hence defeat the evasions in Android malware in order to activate the hidden malicious payloads [10] [11] [12] [13]. The number of defined methods in Android framework is very large, and there are many newly added methods in each update of the Android framework. Hence, there is a big chance to deceit these approaches. That means, if the malware sample uses some methods out of the lists, the analysis process achieved by these approaches can be defeated. Moreover, the large number of the defined methods in different Android framework makes the manual detection and classification of these evasions infeasible. Finally, the newly defined methods in each update in the Android framework impose a heavy load on the analyzer to manually detect and classify the evasions, which can be an error-prone task.

In this paper, we propose a machine learning approach for identifying and categorizing various evasion sources in all Android frameworks. The proposed approach uses the training dataset that was collected in our previous works [6][7]. This dataset includes two types of methods: normal methods and evasion methods (those that can be used by malware to hinder the dynamic analysis process). The evasion methods are categorized into six categories. These categories are the most used categories by malware to evade dynamic analysis. The proposed approach uses two stages of classification. The first stage uses Support Vector Machine (SVM) classifier to detect whether or not the method is evasion. The second stage uses Random Forest (RF) classifier to classify the type of evasions according to the categories defined in the trained dataset. Our decision to use these two classifiers in the proposed approach is based on the nature of the data we deal with, as well as the effectiveness of these classifiers and their high generalization performance.

The proposed approach can use the models obtained from the training process to detect and classify a relatively large number of previously unknown Android API methods. For
example, we applied the resulted models to detect and classify 12759 methods from Android API 27. The proposed approach could detect many new methods that are unknown by the currently available tools that focus on evasions defeating. As a result, the proposed approach can provide the ability to detect the newly used evasions in Android malware and hence provides a comprehensive and more complete list for the dynamic analysis frameworks to deal with these evasions and extract the real behavior of the sample under analysis.

To evaluate the proposed method, we conducted a series of experiments on the proposed classifiers to prove their effectiveness. For the training purpose, we used our collected dataset. The results showed that the proposed approach provides 92.8 accuracy and 92.9 precision for the stage-1 classifier. Also, the accuracy is 90.5, and precision is 90.6 for the stage-2 classifier, which means that the use of the proposed approach to identify evasions can reduce the risk of missing these evasions by the dynamic analysis tools. Consequently, these dynamic analysis tools can defeat these evasions and extract the real behavior of the malware samples. Moreover, to evaluate to which extent the list of evasion methods produced by the proposed approach (after applying it on the methods from Android API 27) are used by the real-world malware samples, we used this list as an input for Droidmon [14]. Then, we utilized 500 samples that are randomly selected from AMD [15] [16] and Contigue Mobile [17] datasets in the evaluation process. The experimental results showed that the real-world samples use some evasions from the Android API methods that are recognized by the proposed approach, but not used by the common dynamic analysis approaches.

The main contributions of this paper are as following.
1) A novel dataset of evasion source methods containing 150 evasion methods obtained from our previous works on real-world samples.
2) A machine learning-based approach to detect and classify various evasions in Android malware even in case of new and previously unseen Android versions and variants.
3) A list of evasion methods that includes 300 new automatically detected evasions obtained by applying the proposed approach on the methods of Android API 27.
4) An implemented version of the proposed approach using Python programming language to evaluate it using a large number of real-world malware samples.

The remainder of this paper is as following. Section II discusses the related work. Section III provides a detailed description of the proposed approach and the used features in each classification stage. Section IV includes the evaluation process that we followed to evaluate the proposed approach. Finally, we conclude the paper in Section V.

II. RELATED WORK

As far as we know, the proposed approach in this paper is the first dedicated approach that focuses on detecting evasion sources in Android framework. However, our approach has been inspired by SUSI [18], which is a machine learning approach to detect and classify sources and sinks used by malware to leak information from Android devices. SUSI (like our approach) uses hand-noted datasets of sources and sinks to predict a larger number of sources and sinks from about 11000 methods in the Android 4.2 framework. Merlin [19] is another approach that uses an incomplete list of sources and sinks and tries to generate a complete one.

Many approaches have been proposed to handle evasion techniques and provide dynamic analysis to the evasive malware. DirectDroid [11] is a tool that uses manually created list of 53 fuzzed APIs to be hooked and defeated during the execution. FuzzdDroid [10] is another tool, that uses hand noted list of 106 fuzzed APIs. These APIs are hooked whenever they are invoked during the execution, and their returned values are set to some other values in order to bypass them and reach the target locations. Ares [12] also, uses a list of fixed APIs number to detect the evasions in the malware samples. However, as we mentioned before, these approaches are effective only if their lists of evasions are complete. In other words, if the list is not complete, these approaches may not be able to detect and hence defeat the used evasions in Android malware.

In general, different Artificial Intelligence (AI) approaches (particularly, machine learning) are used to detect Android malware samples and automatically identify them from various Android markets. These approaches use statically or dynamically extracted features to detect the malicious applications and classify them into different categories. For example, CANDYMAN [20] is a tool that classifies Android malware families by combining dynamic analysis traces and Markov chains. NATICUSdroid [21] is another machine learning approach that uses statically selected native and custom Android permissions as features to detect and classify malware applications from benign applications. IntDroid [22] is another approach that uses sensitive APIs as features to detect malware applications and distinguish them from benign applications.

III. THE PROPOSED APPROACH

This section explains our proposed approach to automatically predict a larger number of evasion methods through learning from hand noted and relatively small number of evasion methods. The proposed system addresses two classification problems. The goal of the first classifier is to detect whether the method is considered as an evasion method or not. The latter will identify under which category the detected evasion method falls. Both classifiers are trained over an extended version of the dataset that we obtained in our previous works [6] [7]. In the following, the architecture of the proposed approach, the features we used in each stage, and the predefined categories are explained.

A. The proposed approach architecture

Figure 1 represents the architecture of the proposed approach. As shown in this figure, the proposed approach takes the API methods we extracted from the Android API 27 as input and provides the categories of all of these methods as output. For the training purposes, our collected dataset is
used. This approach uses two stages of classification. Stage-1 includes using the SVM classifier to detect if the method is considered as evasion or not. In the second stage, all the methods classified as evasion methods will be categorized into predefined categories. A detailed description of each process in the proposed approach is provided in the following.

1) **Data preparation:** The first task of the system is to prepare and manipulate the raw input data to fit the classifiers. This process takes raw Java methods as input and extracts all the features that could be useful to discriminate different input methods. Most of the methods provided by the Android API 27 are extracted and transformed into vectors of features. Various features are used to train each of our following classifiers (see Section III-B). The system is trained over our dataset and then evaluated over the Android API 27 extracted methods.

2) **Stage-1 classification:** This process takes the processed methods (from the first process) as input and detects where it is considered as evasion methods or not. In order to do that, the SVM classifier with Radial basis function (RBF) kernel is used. The SVM classifier is very effective with high dimensional input data and with problems of a number of dimensions larger than the number of samples. Furthermore, it can handle non-separable classification problems efficiently using the kernel trick. Briefly, SVM tries to find the optimal hyperplane that best separates the classes from each other and leads to low generalization error.

3) **Stage-2 classification:** This process takes the Android API methods classified as evasions as inputs and categorizes them into predefined categories. The defined categories are based on our collected dataset (see Section III-B). The used classifier in this stage is the RF classifier. RF is a tree-based classifier that takes the advantages of traditional decision tree along with extra characteristics. RF can achieve better results by using bagging on the samples, majority voting schema, and random subsets of variables. This classifier can scale very well with high-dimensional input data and handle missing values effectively. When the decision tree model grows and becomes very deep, the model may learn irregular patterns that may lead to overfitting. The RF overcomes this drawback by using bagging techniques.

**B. The collected data-set of evasion source methods**

According to the dataset we obtained in our previous studies (Can be accessed via the IEEE dataport). The collected data is a set of methods used to hinder the dynamic analysis of Android malware; an example of these methods is `getDeviceID()` that is used to get the ID of the device where the sample runs. In the case of emulator, this method returns “0000000000000000”. Hence, it can be used to detect the existence of the emulator and hinder the execution accordingly. Another example is `getLongitude()` that is used to request the location of the device where the sample runs and activate the payload based on the location of the device (e.g., the payload can be activated only in China). Therefore, we reused these methods and added other features such as requested permissions, method access modifier, and so on. Moreover, we add the same number of methods (with their features) that are not considered as evasions to provide a balanced training dataset and use it to train the proposed approach.

This dataset contains 300 methods (150 evasion methods and 150 not-evasion methods) extracted from different Android frameworks and are categorized as evasion and non-evasion classes. Not-evasion methods are simply the normal methods that cannot be used as evasions, such as `setDataEnabled()`). The evasion methods are usually used by malware to hinder the dynamic analysis and hide the malicious payload, such as `getDeviceID()`. In our dataset, the evasion methods are categorized into six categories (File access, Integrity check, Location, SMS, Time, Anti-emulation). Among the massive number of features extracted from the malicious applications, we are only interested in the features that could help our classifiers to discriminate between different data samples and achieve high detection accuracy. We use the following extracted features to identify the evasion and not-evasion methods (stage-1 classification process).

- **Package name:** This feature represents the package name for each method. It affects the classifier decision, especially for the popular packages used for evasion.
- **Class modifier:** This feature identifies whether the class is protected or an abstract class. In general, methods from protected classes are not used as evasions.
- **Parts of method name:** A particular part of the method name is taken and used to identify the method names into six cases: get, set, put, is, request, other.
- **Method access modifier:** This feature controls the access to the specified method from other classes or its subclasses. It could be public/private/protected. In general, the evasions methods have public access.
- **Method is returnable:** This feature determines if the method has a return value or not. In general, the evasion methods return values.
- **Parameter is an interface:** This feature indicates whether the method accept a parameter of an interface type or not. In general, this kind of methods belong to not-evasion category since they do not perform direct operations over the data.
- **Parameter type:** This feature indicates the type of the parameter that the method accept. The parameters could be of a concrete type or belong to a specific package. For instance, the methods that accept parameter of package “java.io” are mostly source of evasion.
- **Request permission or not:** This feature identifies whether the method requires specific permission to be executed. Most evasion methods request permissions in order to get system services. On the other hand, the features used to categorize the evasion methods to their corresponding category (stage-2 classifier) are illustrated in the following.

On the other hand, the features used to categorize the
Fig. 1. The architecture of the proposed approach

evasion methods to their corresponding category (stage-2 classifier) are illustrated in the following.

- **Package name:** This feature represents the package name for each method. It affects the classifier decision, especially for the popular packages used for evasion.

- **Class name:** The name of the class can play a vital role in categorizing the evasion methods. For example, methods from `Build` class are categorized as Anti-emulation evasions because they are used to detect the used test environment.

- **Return value type:** This feature identifies the type of the method returned values. For example, the Anti-emulation methods return strings in most cases, while the Location methods return double type in most cases.

- **Parameters number:** This feature represents the number of arguments the method takes as input. For example, the Time methods get 0 or 1 argument, while the File access methods get more than 3 arguments as input.

- **Permission type:** This feature identifies the type of permission the method could request if it exists. Otherwise, it will take “None” value.

It should be noted that all of the features are categorical features except arguments number, which holds numerical data. This fact encouraged us to choose a tree-based classifier to construct the stage-2 classifier because of its well-known performance for this kind of data.

C. The predefined categories

We define six categories based on our empirical study that we achieved on AMD [15] [16] dataset. These categories are described in the following.

- **File access:** This category includes any method used to read special contents from files or databases.

- **Integrity check:** This category includes any method used to check if the malware code is manipulated.

- **Location:** This category includes any method used to detect the location of the test environment.

- **SMS:** This category includes any method used to read special addresses and contents from incoming messages.

- **Time:** This category includes any method used to delay the execution of the sample.

- **Anti-emulation:** This category includes any method used to detect whether the test environment is an emulator or a real device.

IV. EXPERIMENTAL EVALUATION

In this section, we perform a series of experiments on the proposed system to prove its effectiveness. For the training purpose, we use our collected dataset, and for the evaluation purpose, we utilized 500 samples randomly selected from AMD [15] [16] and Contigue Mobile [17] datasets. The experiments were conducted on a laptop running Windows 10 OS that has a core i7 processor with 8 GB RAM. The proposed system is programmed using Python version 3.7.4.

To measure the quality of the classifiers, we use the accuracy, precision, recall, and f1-measure as our classification metrics. To obtain better insight into the generalization performance of our classifiers, 10-fold cross validation was employed for both classifiers. Our experiments and evaluations try to answer the following research questions.

1) Can the proposed approach be used to find evasions in Android malware?

2) Can the proposed approach be used to categorize the detected evasions?

3) Is the obtained list of evasion methods by the proposed approach actually used by real-world malware?

The following section tries to address the aforementioned questions in detail.
A. Effectiveness of the proposed approach in finding evasions

The main goal of the first classifier is to identify whether the method is used as evasion or not. To train this classifier, we used our balanced dataset. In order to obtain the most accurate estimation of this model, we employed 10-fold cross validation. However, we evaluated the performance of the classifier in term of precision (the rate of positive identifications was classified correctly), recall (the rate of correctly classified positive samples), accuracy (the rate of correctly classified samples), F1-score (the harmonic mean of precision and recall). The confusion matrix of the stage-1 classifier is depicted in Figure 2. As shown in this figure, the SVM classifier achieved high detection accuracy through the high value of the true positive (0.92) and true negative (0.94) values. Furthermore, the small values of false positive and false negative indicate that the SVM classifier has a relatively small number of misclassified samples, and there is no bias toward any specific class.

![Image of confusion matrix](image)

**Fig. 2. Confusion matrix of the stage-1 classifier**

The classification metrics that estimate the performance of the classifier are shown in Table I. As can be seen from the table, the stage-1 classifier was able to achieve outstanding performance over our dataset. The average accuracy is 92.8%, with high and close precision and recall. The results indicate that the classifier has no bias towards any class, which proves that the effectiveness of the stage-1 classifier in detecting the evasion methods from the samples under analysis.

<table>
<thead>
<tr>
<th></th>
<th>AUC</th>
<th>Accuracy</th>
<th>F1</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage-1 classifier</td>
<td>0.977</td>
<td>0.928</td>
<td>0.925</td>
<td>0.929</td>
<td>0.921</td>
</tr>
</tbody>
</table>

**TABLE I**

<table>
<thead>
<tr>
<th></th>
<th>AUC</th>
<th>Accuracy</th>
<th>F1</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage-1 classifier</td>
<td>0.977</td>
<td>0.928</td>
<td>0.925</td>
<td>0.929</td>
<td>0.921</td>
</tr>
</tbody>
</table>

B. Effectiveness of the proposed approach in categorizing the detected evasions

The goal of the stage-2 classifier is to identify the category of the detected evasion methods. We extracted all the samples from the training dataset that represent evasion methods and used them to train the stage-2 classifier. To verify the classifier performance, 10-fold cross validation is used while training the stage-2 classifier. The resulted confusion matrix is depicted in Figure 3. As shown in this figure, the stage-2 classifier achieved accuracy equal to 97%, 100%, 98%, 95% in classifying all the methods related to the File access, SMS, Time, and Anti-emulation categories, respectively. On the other hand, the accuracy for the Location and Integrity check categories was not good enough. The reason behind that is the high similarity between the Location and Anti-emulation methods (some methods from the TelephonyManager class can be used to get the location of the test environment, such as getNetworkOperator() method) which prevent the classifier from distinguishing between them. On the other hand, the number of methods belongs to the Integrity check is relatively small. This prevents the classifier from understanding the correct pattern of their features. Another conclusion we can obtain from Figure 3 is that the stage-2 classifier deals with unbalanced data. This kind of data will result in a high bias toward a specific class, which we can see in the results. In our work, we employed 10-fold cross validation to deal with this issue.

![Image of confusion matrix](image)

**Fig. 3. Confusion matrix of the stage-2 classifier**

The classification metrics resulted from training the stage-2 classifier over our dataset are shown in detail in Table II.

<table>
<thead>
<tr>
<th></th>
<th>AUC</th>
<th>Accuracy</th>
<th>F1</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage-2 classifier</td>
<td>0.970</td>
<td>0.905</td>
<td>0.899</td>
<td>0.906</td>
<td>0.892</td>
</tr>
</tbody>
</table>

**TABLE II**

As can be seen from Table II the resulted accuracy is around 90.5%. The general performance of the classifier is good. Since the number of samples is not large enough and the data is not balanced. Some bias may appear in the classifier results, which leads to degrade its performance. However, according to the results that we obtained, we can say that the stage-
2 classifier provides a good performance in categorizing the detected evasion techniques.

C. Potential evasion methods in the Android malware samples

The main question that could be asked at this stage is: “is there any other evasion method used in the malware applications discovered by our system but could not be recognized by the state-of-the-art analyzing tools?” To answer this question, we used 500 samples that are randomly selected from AMD and Contigue Mobile datasets. We used the potential evasion list we obtained from our experiments over Android API 27 as an input to the Droidmon [14]. In general, the Droidmon takes the list obtained from our proposed approach and captures the defined methods in this list whenever they are invoked during the execution. Hence, we ran the selected 500 samples using the framework proposed in [7] and waited for Droidmon to capture the predefined methods.

Interestingly, Droidmon was able to detect many potential new evasions that were not recognized by currently available tools [11] [12] [13]. For example, queryUsageStats() method from UsageStatsManager class. This method is used to get the usage states for all the applications in a specific period. Another example is the getMnc() method from CellIdentityGsm class. This method is used to get the mobile network code. The first method can be used as an Anti-emulation evasion to detect the existence of some applications that are commonly used by normal users, such as Whatsapp. The latter method can be used as a Location evasion to detect the location of the test environment based on the mobile network code.

After analyzing the 500 samples, we found that the number of samples using evasions was 456 samples in overall. Moreover, the number of evasion methods detected by Droidmon was 1301 evasions. Figure 4 represents the number of the detected methods for each predefined category.

![Fig. 4. The number of the detected methods in each category](image)

As can be seen from Figure 4, the most detected category is the Anti-emulation category, which emphasizes the experimental evaluation we have achieved manually. The methods in this category are commonly used by the malware either to detect the type of the test environment or to steal some sensitive information such as the device IMEI. Furthermore, this category includes a wide range of methods that can be used to detect the test environment, unlike the Integrity check category that can include a very limited number of methods.

V. Conclusion

In this paper, we introduced a machine learning approach to detect the evasion methods used by Android malware and to categorize them into six general categories. The proposed approach includes two stages of classification. The stage-1 classification includes using the SVM classifier because of its effectiveness in dealing with linearly non-separable problems, as well as its high generalization performance. While the stage-2 classification includes using the Random Forest classifier because it can scale very well with high-dimensional input data, and it can also handle missing values effectively. We used the results from our previous works to provide a hand noted dataset to train the two classifiers. The stage-1 classifier provides 92.8% accuracy with high and close precision and recall, and the stage-2 classifier provides 90.5% accuracy. We applied the resulted models on the 12759 methods from Android API 27 to find new evasions. To ensure that the real-world malware samples actually use the resulted methods from the prediction process, we used 500 malware samples that are randomly selected from AMD and Contigue Mobile datasets. The results showed that some evasions detected by the proposed approach (and not recognized by state-of-the-art dynamic analysis frameworks) are actually used by real-world samples, which emphasizes our hypothesis that completing the list of evasions increases the effectiveness of the dynamic analysis frameworks.

The proposed approach has some difficulty in identifying the correct category of evasion. The nature of the input data and the samples distribution significantly impact the classification process. Hence, we aim to use other classification features and use more flexible and robust classifiers as our future work.

REFERENCES


