Journal Pre-proof

DroidEnemy: battling adversarial example attacks for Android malware detection

Neha Bala, Aemun Ahmar, Wenjia Li, Fernanda Tovar, Arpit Battu, Prachi Bambarkar

PII: S2352-8648(21)00090-0

DOI: https://doi.org/10.1016/j.dcan.2021.11.001

Reference: DCAN 332

To appear in: *Digital Communications and Networks*

Received Date: 21 September 2020

Revised Date: 27 October 2021

Accepted Date: 1 November 2021

Please cite this article as: N. Bala, A. Ahmar, W. Li, F. Tovar, A. Battu, P. Bambarkar, DroidEnemy: battling adversarial example attacks for Android malware detection, *Digital Communications and Networks*, https://doi.org/10.1016/j.dcan.2021.11.001.

Digital Communications and Networks(DCN)

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/dcan

DroidEnemy: battling adversarial example attacks for Android malware detection

Neha Bala[a], Aemun Ahmar[a], Wenjia Li*[a], Fernanda Tovar[a], Arpit Battu[a], Prachi Bambarkar[a]

[a]**Department of Computer Science, New York Institute of Technology**
New York, NY, 10023, USA

Abstract

In recent years, we have witnessed the proliferation of mobile devices such as smart phones, tablets, smart watches, etc., the majority of which are based on the Android operating system. However, because these Android-based mobile devices are becoming increasingly popular, they are now the primary target of mobile malware, which could cause both privacy leakage and property loss. To address the rapidly deteriorating security issues caused by mobile malware, various research efforts have been made to develop novel and effective detection mechanisms to identify and battle them. Nevertheless, in order to avoid being caught by these malware detection mechanisms, malware authors are inclined to launch adversarial example attacks by tampering with mobile applications. In this paper, several types of adversarial example attacks are investigated and a feasible approach is proposed to fight against them. First, we look at adversarial example attacks on the Android system and prior solutions that have been proposed to address these attacks. Then, we specifically focus on the data poisoning attack and evasion attack models, which may mutate various application features, such as API calls, permissions and the class label, to produce adversarial examples. Then, we propose and design a malware detection approach which is resistant to the adversarial examples. To observe and investigate how the malware detection system is impacted by the adversarial example attacks, we conduct experiments on some real Android application datasets which are composed of both malware and benign applications. Experimental results clearly indicate that the performance of Android malware detection is severely degraded when facing the adversarial example attacks.

KEYWORDS: security, malware detection, adversarial example attack, data poisoning attack, evasion attack, machine learning, Android

1. Introduction

The Internet of Things (IoT) is generally composed of a set of smart devices that are connected with each other via the Internet technologies. IoT devices, together with the mobile operating systems (such as Android and iOS) that run on them, have revolutionized many aspects of our daily lives, such as smart home, autonomous driving, smart healthcare, and so on.

According to a recent study, Android maintains its position as the leading mobile operating system worldwide, with over 70% of mobile devices globally run on Android [1]. While the Android system remains its popularity largely because of the very diversified and powerful mobile applications based on it, it is also becoming more vulnerable to serious security threats such as mobile malware. Generally, mobile malware

*Dr. Wenjia Li (Corresponding author) is an associate professor in computer science at New York Institute of Technology, New York, NY, 10023, USA (email: wli20@nyit.edu).

[1]Neha Bala was an undergraduate student in computer science at New York Institute of Technology, New York, NY, 10023, USA (email: nbala02@nyit.edu).

[2]Aemun Ahmar was an undergraduate student in computer science at New York Institute of Technology, New York, NY, 10023, USA (email: aahmar@nyit.edu).

[3]Fernanda Tovar was an undergraduate student in computer science at New York Institute of Technology, New York, NY, 10023, USA (email: ftovar@nyit.edu).

[4]Arpit Battu was an undergraduate student in computer science at New York Institute of Technology, New York, NY, 10023, USA (email: abattu@nyit.edu).

[5]Prachi Bambarkar was an undergraduate student in computer science at New York Institute of Technology, New York, NY, 10023, USA (email: pbambark@nyit.edu).

aims at generating revenue in an unethical or even illegal manner. To achieve this goal, malware could steal sensitive and valuable user information such as security credentials, current location, contact list, etc., make the user's device send SMSs to premium rate text services, or install adware that forces the user to view web pages or download another malware or unwanted application. Fig. 1 depicts the current status of the Android system, showing its popularity as well as security threats imposed by malware.

To fight against these security threats, researchers have recently developed various approaches to identify malicious applications for the Android system by applying machine learning algorithms, such as random forest, logistic regression, and Support Vector Machine (SVM), etc [2]. However, many malware detection approaches become much less effective when adversarial example attacks are launched by the malware authors whose goal is to ensure that the traditional machine learning based approaches cannot detect the malware any more [3]. To deal with the increasing security threats imposed by the adversarial example attacks, some research efforts have been made recently [3].

In this paper, we first focus on studying three types of popular adversarial example attacks, namely the data poisoning attack, the exploratory attack, and the evasion attack. These attacks are normally launched by providing mutated malware samples to the existing malware detection systems so that malicious applications would falsely be classified as benign ones. Particularly, we develop specific attack models for both the data poisoning attack and the evasion attack to generate adversarial examples. To battle these adversarial example attacks, we develop an attack-aware malware detection approach for the Android system. As shown in Fig. 2, our approach contains the following functionalities: feature extraction, feature reduction, generation of adversarial examples, and detection of malware in the presence of mutated malware.

The major contributions of our work are summarized as follows:

- We develop two attack models based on the data poisoning attack and the evasion attack to generate adversarial examples, which focus on mutating different categories of Android application features, such as API calls, permissions and the class label, to feed to the classifier. With them, we can further improve the robustness of malware detection approach in the presence of adversarial examples.

- We build a malware detector using the Support Vector Machine (SVM) algorithm. To help better cope with the adversarial example attacks, we also look at various types of kernel functions for the SVM algorithm, such as linear and Radial Basis Function (RBF) kernel types.

- To study the effect of adversarial example attacks on the malware detection systems, we perform an experimental study on a real Android application dataset. The experimental results show that the evasion attack causes the classifier to have 100% incorrectly classified instances and 0% correctly classified instances. In addition, the data poisoning attack causes the classifier to have 74.85% correctly classified instances and 25.14% incorrectly classified instances. From the experimental study, we find that the evasion attack could generally cause more performance degradation to the malware detection system than the data poisoning attack.

The remainder of this article is structured as follows: We first review the prior research efforts on Android malware detection in Section 2. In Section 3, we provide a comprehensive overview of the well-known adversarial example attacks against Android malware detection systems. We describe our methodology to fight against the adversarial example attacks in Section 4. To observe the effect of the adversarial example attacks on malware detectors, we perform an experimental study and present its results with some analysis in Section 5. In Section 6, we conclude this research, and point out some possible directions that could be taken in the future to better deal with the adversarial example attacks.

2. Related work

In this section, we survey the literature so as to better understand how the existing malware detection approaches identify malware samples for the Android system. In general, there are three types of malware detection approaches, which are based on static analysis, dynamic analysis, and machine learning algorithms, respectively.

2.1. Static analysis based approaches

In malware detection, static analysis aims at identifying issues and flaws that exist in mobile applications without actually executing them. The traditional static analysis based approaches usually focus on exploring the similarity between the targeted mobile applications and the families of previously known malware, which could be achieved by searching for a particular form of signature [4].

Through static analysis, AndroidLeaks [5] was designed for identifying privacy or leakage of sensitive data for Android applications. In [6], Grace et al. studied RiskRanker, which was an automated framework for analyzing a given mobile application to determine whether or not it demonstrates certain types of malicious actions, such as sending unwanted SMS messages or launching root exploit.

It is worth noting that static analysis has some limitations when used to detect malware, especially when

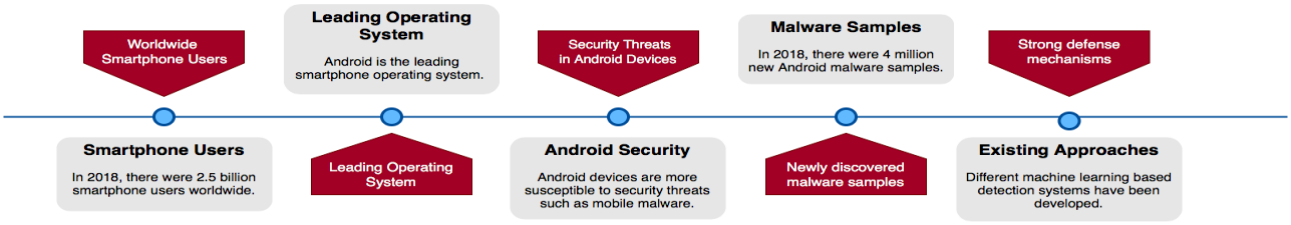

Fig. 1. The latest Android system status in regards to its popularity and security risks

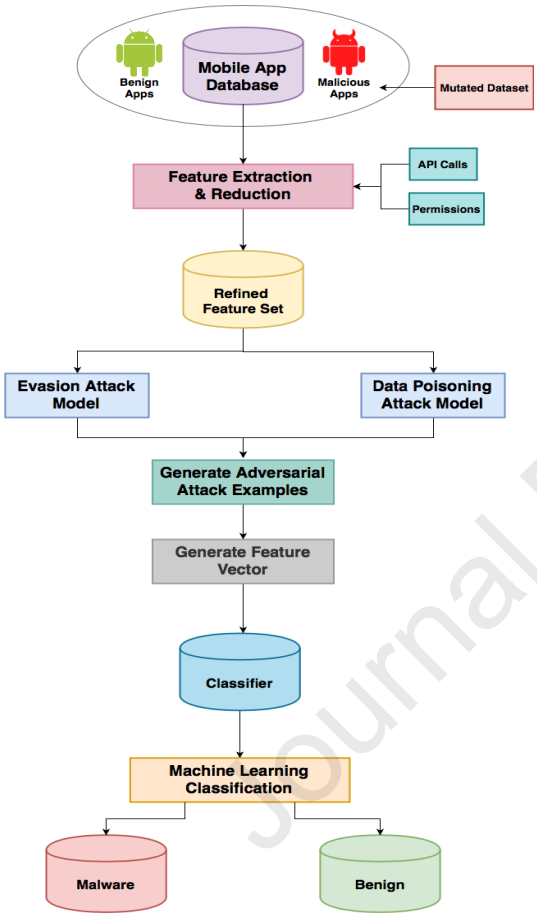

Fig. 2. An overview of the system architecture

used for malware detection in large quantities of mobile applications [7, 8]. In addition, static analysis cannot recognize security susceptibilities of mobile applications at the runtime. Consequently, dynamic analysis could also be performed afterwards in order for more sophisticated malware to be detected.

2.2. *Dynamic analysis based approaches*

Dynamic analysis is normally performed for the purpose of malware detection when the mobile applications are being executed either on a virtual emulator or on a real mobile device to collect information on the runtime behaviors of mobile applications. Dynamic analysis is accomplished by tracking any suspicious activity that utilizes unwanted third party libraries to leak confidential and private data. This approach could generally be implemented by monitoring the invocation of high risk API calls.

DroidScope, proposed in 2012 [9], was an approach to detect malware that can simultaneously and efficiently rebuild both the OS-level and Java-level semantics by the means of extracting three tiered APIs that mirror the three levels of an Android device: hardware, OS and Dalvik Virtual Machine.

In [10], a Multi-Level Anomaly Detector for Android Malware (MADAM) was introduced by Dini et al. MADAM was able to detect system calls at the kernel level and user activities at the user level to precisely depict the behavior of a mobile application.

Since the dynamic analysis based approaches can identify abnormal and malicious behaviors that mobile applications exhibit at the execution time, they are typically considered to be a useful complement to the static analysis based approaches. However, the mobile applications have to be executed to perform dynamic analysis in order to gather run-time app behaviors for a period of time before noticing any suspicious behaviors. Consequently, it typically introduces additional time overhead.

2.3. *Machine learning based approaches*

Both static and dynamic analysis based approaches may not function well for malware detection when handling a great amount of new malware samples, particularly when some of them attempt to evade the detection by applying the automated code obfuscation method. To fight against malware more effectively, researchers have applied different machine learning algorithms in recent years to detect malware in the Android system.

In [11] and [12], the SVM algorithm was deployed to distinguish malicious applications from benign ones in the Android system. In [13], the researchers proposed a Feature Extraction and Selection Tool (FEST) for detecting Android malware, which was based on the frequency of the features found in various malware samples. Based on those selected features, the SVM algorithm was then applied to identify malicious applications.

In [14], Alam and Vuong suggested to use the random forest algorithm to detect malware, aiming at im-

proving the detection effectiveness by tuning multiple parameters in the random forest algorithm.

Yerima et al. [15] applied Bayes's theorem to classify Android applications and identify malicious samples. In this approach, various app-related features, such as permissions, Linux commands, and API calls, are used to train and perform tests on a Bayesian classification model for malware detection.

Wang et al. [16] proposed the DroidDeepLearner approach, which attempted to solve the malware detection problem by using the deep learning algorithm. The primary goal of this approach was to make malware detection more autonomous. In addition, some other research efforts were made recently to better address malware in the Android system, all of which were built upon deep learning [17, 18, 19].

In recent years, the machine learning based malware detectors are susceptible to various adversarial example attacks, such as those introduced in [7, 8, 20, 21]. The overall goal of these adversarial example attacks is to make malware detectors recognize malware as benign apps. Various types of adversarial example attacks will be discussed further in Section 3.

3. Basic principles of adversarial example attacks

The emergence of various detection approaches has been an effective countermeasure against malware. To evade detection by these approaches, malware authors developed and launched adversarial example attacks, in which twisted labeled or fake malware samples could confuse malware detection approaches, thereby weakening their effectiveness [22]. In this section, we have an in-depth discussion on three well-known adversarial example attacks, namely the data poisoning attack, the exploratory attack, and the evasion attack, which are shown in Fig. 3. More specifically, we focus primarily on the evasion attack and the data poisoning attack.

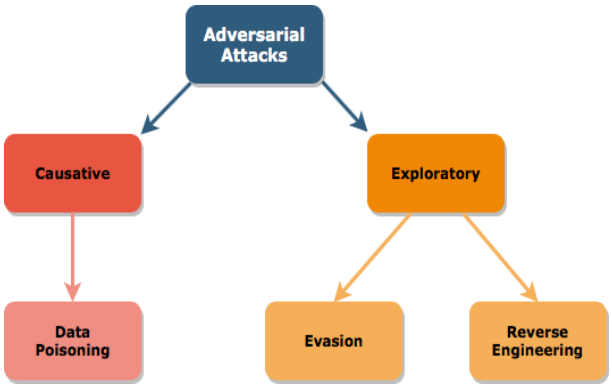

Fig. 3. Categorization of adversarial attacks

3.1. Data poisoning attacks

A recent research study found that the effectiveness of machine learning based malware detectors would be significantly lowered by injecting deliberately falsified data into their training set, which is generally called data poisoning attack [23].

Once the data poisoning attack is launched, the performance of malware detectors is weakened because the adversary obtains access to the learning algorithm's training dataset. Four attack strategies could be adopted when launching the data poisoning attack, which are label modification, data modification, logic corruption, and data injection. These four attack strategies are depicted in Fig. 4.

When the label modification strategy is adopted, some or all of the class labels in the training datasets are modified by the attacker, which would mislead malware detectors into regarding malicious applications as benign ones and vice versa. As for the data injection strategy, although the malware author cannot access the training data or the classifier, the attacker alternatively corrupts the training data by adding new falsified data. When launching the data modification strategy, the attacker simply alters the data before they are used for training, hence poisoning the data. Finally, the attacker directly modifies some portions of the learning algorithm when adopting the logic corruption strategy [24].

In order to combat the data poisoning attack, Chen et al. studied a data poisoning attack that relies on reverse engineering [25]. This attack generates ingenious and well-built samples that infiltrate the training dataset. The authors discussed in details about different types of data poisoning attackers, which include weak, strong, and sophisticated attackers. With reverse engineering, the attackers can discover all details about an Android app, and they implement a customized adversarial crafting algorithm to create these samples using syntax features. These samples are viewed as benign by the classifiers while actually they are malicious [25]. Therefore, these samples can be injected at a large scale into the training data to significantly lower the overall detection accuracy of the malware detectors.

3.2. Exploratory attacks

In the exploratory integrity attack, the adversary passively attempts to go around the machine learning algorithm to exploit blind spots in the learning model, which allows the adversarial attack to remain undetected [26]. The exploratory integrity attack is one of the most frequently studied attacks. During such an attack, the adversary finds ways to evade the classifier, without influencing the classifier itself. Instead, the exploratory attack tries to make an attack look like a normal behavior to the detector, and hide the identifying characteristics of the attack. Some exploratory attacks copy statistical properties to hide the attack.

Fig. 4. Overview of Data Poisoning Attacks

For example, the training data and the classifier are both examined by the attacker who creates the intrusion data.

Exploratory integrity attacks on Intrusion Detection Systems (IDS) and spam filters have been studied extensively. Another common method for avoiding spam filters is by ensuring that a word is not recognizable by the filter, which can be done by changing certain characters or the spelling of the word [26]. A dynamic based approach is used by exploratory attacks in order to evade the user. The exploratory attacks do not modify or mutate the data, instead they are executed when the system is running. Because the exploratory attacks operate by analyzing the classifiers, they are complex and hard to be detected [27].

In general, the exploratory attacks are difficult to defend against as they are usually highly unpredictable. They can be implemented in different ways, but they all follow the similar idea, namely, to be aggressive and attack the classifiers [21].

One of the many ways that the exploratory attacks can be implemented is by trying to learn and model the boundaries of the classifier. More knowledge about the boundaries can also disclose sensitive classifier information and how the defense system works. With proper implementation, this attack can greatly weaken the malware detectors [27].

In reverse engineering, the adversary tries to model the decision boundary of the classifier. An attack of this nature can either be used as a first step prior to a sophisticated exploratory attack, or it can be used on its own. For example, sensitive information can be disclosed by the decision boundary. In the case of linear classifiers, the sign witness test is one reverse engineering technique, in which it is determined by the adversary whether a feature contributes positively or negatively to the final decision of the classifier. In general, a few probe messages can be used to reverse engineer some essential types of classifications.

3.3. Evasion attacks

The evasion attack, which is another type of exploratory attack, is also shown in Fig. 3. The evasion attack is generally viewed as the most widely used attack against the machine learning based malware detectors. By launching an evasion attack, the input data is deliberately modified to evade detection by making the machine learning algorithm classify malware samples as benign applications.

The adversary creates an attack message by manipulating the input that will be detected by the classifier in order to produce incorrect outputs. This approach is called the white box. The modified message can successfully evade the detection system, unlike an ordinary message [27]. In order to make this attack effective, the adversaries are limited on how much their message could be disguised. If the message is also disguised, the classifier may not be able to recognize it [27].

In [28], Tong et al. discussed different approaches to defend against evasion attacks. The three effective approaches are game-theoretic reasoning, robust optimization, and iterative adversarial retraining. However, both the game-theoretic methods and the robust optimization are not general purpose solutions that could work in many different cases.

4. Methodology

In this section, we describe the overall methodology for detecting malware in the Android system when adversarial example attacks are launched.

4.1. Overall design

The overall design of our malware detection system is demonstrated in Fig. 2.

The first step is collecting and pre-processing Android applications. For this research, we collect and maintain an Android application database consisting of benign applications, non-mutated malicious applications, and mutated malicious applications. In order to better identify malware, we rely on a wide variety of mobile application features, such as permissions and API calls.

We then disassemble all the mobile applications to access the manifest file which contains those application features. By gaining access to these files, we then extract features from the applications, and reduce the number of features we have, which allows us to obtain a refined feature set which we could use to generate adversarial attack examples. Furthermore, we launch the adversarial example attacks to generate mutated feature vectors for malware. Finally, the classifier will determine whether the applications are benign or malicious.

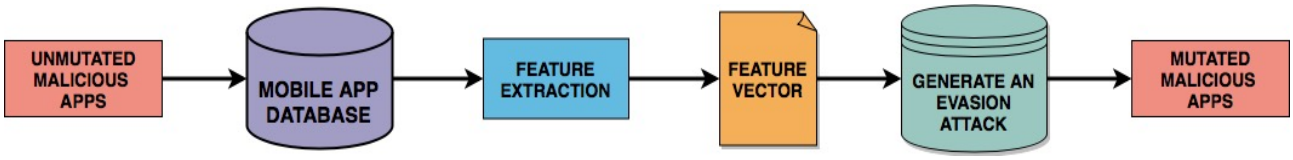

Fig. 5. Workflow of Evasion Attack

4.2. Feature extraction

Using our dataset, we extract features such as permissions and API calls that are located in the AndroidManifest.xml file and in the classes.dex file. We extract all the permissions that are requested by each application. To create feature sets based on feature relevance, we look at the binary N-grams of API calls. Through this analysis, we are able to analyze all the features to identify which specific permissions or API calls are more significant to benign apps and which are more significant to malicious apps. The aim of feature extraction and selection is to generate new features from current ones to decrease the number of features in our dataset. The new reduced feature set will then summarize much of the details contained in the original feature set.

4.3. Feature selection

We use Term Frequency–Inverse Document Frequency (TF-IDF) for the feature selection process, which is a text mining technique normally used for the categorization of documents. The aim of TF-IDF is to measure the relevance of a word in a given text. In this case, TF-IDF will be used for extracting and categorizing features that are more essential to benign applications and those that are more essential to malware applications. Applying TF-IDF for selecting features, will help reduce the feature vector to only the features that are essential to both the benign class and the malware class. For the TF-IDF model the following terms and variables are defined [29].

$$D = d1, d2, \ldots, dn \tag{1}$$

$$V = w1, w2, \ldots, wn \tag{2}$$

In this research, D represents the list of Android applications and d represents a specific application. Furthermore, V represents the list of features while w represents each feature in the list of applications.

There are some features in the application list which are more frequent than others. TF-IDF allows us to create a refined feature vector by giving weights to the features which are more common in each app. In general, TF-IDF is calculated using the specified terms and variables, as follows:

$$tf(w,d) = \frac{fd(w)}{maxf(w)} \tag{3}$$

In this research, the number of times a feature is called in an application is calculated as the TF.

$$idf(w) = log\frac{N}{df(w)} \tag{4}$$

The IDF measures whether the feature in the application list is popular or uncommon. D contains N number of applications while df(w) is the feature frequency or the number of times a feature is called in all of the applications combined.

$$tfidf(w,d) = tf(w,d) \times idf(w) \tag{5}$$

As mentioned before, with TF-IDF we have two feature vectors, one containing features which are most relevant to benign apps and the other containing features which are most relevant to malware apps. These two feature vectors are then used for training the classifier.

4.4. Adversarial example generation

The two feature vectors generated by the TF-IDF technique are fed to the classifier for training against the adversarial attack models used to evade classifiers through malicious samples. In order to strengthen the accuracy of malware detection, the attack models are able to help train the classifier to allow for improved prediction rates in the presence of adversarial examples. For this research, we mainly use two adversarial attack models, namely the evasion attack and the data poisoning attack. Both attack models are applied separately to the same feature vector that is used to train the classifier.

4.4.1. Data poisoning attack

As mentioned previously, in a data poisoning attack, the machine learning algorithms are weakened by inserting malicious data into the training sets. In the data poisoning attack model that we develop, we adopt the label modification (a.k.a. label flipping) strategy, in which the attacker randomly chooses a mobile application and then changes some of its labels in a training dataset. We intend to launch this attack by creating our own label modification attack algorithm based on the idea to simulate a real data poisoning attack.

We generate the attack on the feature vector that is used to train the classifier. This feature vector consists of two labels: 0 and 1. Our label flipping algorithm

chooses a random number of features. For each of the chosen feature, it will flip the label according to whether it is a 0 or a 1.

4.4.2. Evasion attack

As mentioned before, the second type of attack we launch is the evasion attack. In an evasion attack, an attack message is generated by the adversary by manipulating the input that will be detected by the classifier to produce incorrect outputs. An evasion attack occurs when an "adversarial example" is fed to the classifier, i.e., it is fed a malware sample that has been manipulated which the classifier incorrectly categorizes as a benign application.

We work with a dataset consisting of malware applications that have been mutated, as shown in Fig. 5. This dataset is mutated without losing its malicious characteristics and capabilities. We extract features, such as binary N-grams of API calls, from the dataset and create a subset or a representation of features for the malware samples that have been mutated. The feature representation is transformed into a vector space that the attack model uses to produce an attack against the classifier and deceive it into classifying the malware application as a benign one. This will be accomplished before the classifier is trained using the feature vector generated by applying the TF-IDF technique.

4.5. Machine Learning Classifier

For the step of malware detection, we adopt the SVM algorithm, which allows the data to be separated into distinct classes. For our classifier, we have two classes named OriginalMalware and Benign. Depending on the type of dataset we are training or testing the classifier, the classes allow the data to be processed and evaluated. In general, the support vector machine develops an optimal hyperplane that builds a maximum margin between the classes we work with. The data points that are closest to the hyperplane of the support vectors balance the position of the hyperplane.

For our experiment, we used an open source machine learning software called Weka [30]. In order to build an SVM classifier in Weka, we install a widely used library called LibSVM [31]. By using the LibSVM, we build a classifier using different kernel types such as linear and nonlinear kernel functions.

In order to feed a dataset to a classifier, each application is represented in a form of a vector so as to be read and analyzed correctly. For our dataset, we format our feature vectors into arff files which are the recommended file for Weka. An arff file is formatted by first declaring all the attributes or the number of features that we are working with in the dataset. Furthermore, we declare our classes in which the classifier will work with followed by the feature vectors for each application.

The classifier analyzes the vector to predict which class each application belongs to. These predictions are made based on the training set. Once the classifier gives its predictions, it will look at which class each application really belongs to based on the declared class in the arff file. This allows the classifier to give an output of instances that have been correctly classified and instances that have been incorrectly classified.

5. Experimental study

In this research, we conduct an experimental study on two adversarial example attack models, namely the data poisoning attack and the evasion attack. We perform our experiment using both the original Drebin malware dataset [11] and a mutated Drebin dataset [20]. The benign applications are downloaded directly from Google Play.

5.1. Generating a data poisoning attack

In order to implement the data poisoning attack, we use the tools described in [32], namely, RevealDroid and IagoDroid. Both tools are used to implement a data poisoning attack. IagoDroid is a tool that is used to modify classifiers, and RevealDroid generates a model and that model is input into IagoDroid to modify and fool the classifier, therefore implementing the data poisoning attack [32].

Based on the methodology of RevealDroid [33], we developed the label flipping algorithm, which is used to generate a data poisoning attack. The algorithm takes feature vectors in the form of a CSV file as an input to launch the attack, which is shown in Algorithm 1.

```
Data: Non-Mutated Feature Vector
Result: Vector Space
initialization;
for all apks do
    String[] currAPK = split single feature
      string as separate features;
    String label = last comma separated value
      of currAPK;
    for currAPK do
        if label is Malware then
            parse label as Integer;
            add label to vector space;
        else
    end
end
```

Algorithm 1: Feature vector space generation

Once the feature vector is generated, the algorithm runs the attack on the features. As mentioned before, we developed a label flipping attack algorithm, which is shown in Algorithm 2. In this attack, the label of a random number of features is flipped, thus mutating

the APK's features. To run this attack, the algorithm goes through the feature space. For each APK, it selects a random feature and if that feature hasn't been mutated yet, the attack flips the label. If the selected feature has already been mutated, it selects another feature until it finds one that hasn't been mutated yet. After the attack is run on all of the APKs, the resulting vectors are output into a new CSV file.

```
Data: Non-Mutated Feature Vector
Result: Mutated Feature Vector
initialization;
for all apks do
    for random number of apks do
        randCol = choose random feature;
        if randCol not flipped then
            value = get label of random;
            if value == 0 then
                flip value to 1;
            else
                flip value to 0;
            end
            add label to vector space;
        else
            choose another feature;
        end
    end
end
```

Algorithm 2: The label flipping algorithm

5.2. Generating an evasion attack

To generate an evasion attack, we work with non-mutated Drebin malware applications. Using the feature vector of the applications, we run the vectors through the evasion attack model. The evasion attack model will then generate adversarial examples which are represented in the form of feature vectors as well. These mutated feature vectors are then fed into the SVM classifier in order to evade the detection system.

5.3. Performance comparison between linear and non-linear kernels

Using the LibSVM tool, we build our machine learning classifier in Weka. In order to test the performance of the adversarial attack models, we analyze the performance with both linear kernel type and the non-linear kernel type.

Linear kernels are a category of kernel methods. They allow to use linear functions. Linear kernels are the simplest kernel function. The linear kernel uses the dot product between the input and each vector. The equation involves the computation of the inner products of a new input vector with all the support vectors in the data used for training. Non-linear kernels are another category of kernel methods. Non-linear methods transform data to a new representational space first and then apply classification techniques.

5.3.1. Experimental results with non-mutated Drebin dataset

To train the classifier, we used the original Drebin dataset with non-mutated malware samples and the benign samples that are directly obtained from Google Play. In order for the classifier to process the applications, we formatted the malicious and benign apps into feature vectors.

Using the dataset, we find that with the linear kernel type the SVM classifier is able to correctly classify instances by 97.5382% and incorrectly classify instances by 2.4618%. When performing the test with the non-linear kernel type with radial basis function, we find that the SVM classifier is able to correctly classify instances by 87.1345% and incorrectly classify instances by 12.8655%.

5.3.2. Experimental results with the data poisoning attack

After launching the label flipping algorithm, we feed the feature vector that contains the adversarial examples into the SVM classifier. The results demonstrate that the adversarial attack is successful in confusing the classifier.

After feeding the mutated dataset to the classifier, the detection accuracy decreased. It is found that data poisoning by label flipping causes the classifier to have 74.85% correctly classified instances and 25.14% incorrectly classified instances. This is achieved by using the linear kernel type. However, using the non-linear kernel type, the classifier correctly classifies instance by 50.2924% and incorrectly classify instances by 49.7076%. From these results, we can conclude that the linear kernel is more resistant to the data poisoning attack than the non-linear kernel.

5.3.3. Experimental results with the evasion attack

Along with testing the classifier with the data poisoning attack, we have also launched the evasion attack against the linear kernel based SVM classifier to evaluate its performance. We find that the evasion attack is more threatening to the malware detectors than the data poisoning attack.

The testing results show that the classifier is able to correctly detect 0% of the instances and 100% incorrectly classify the instances. This means that by launching the evasion attack, the malware detector could not correctly detect any malware. These results remain the same with the non-linear kernel based SVM classifier.

As shown in the prior research efforts [8, 20], the evasion attack generally poses a severe threat to the existing malware detectors, and it could even drop the detection rate to be almost 0% in different cases. This is consistent with what we find in our experiments.

5.4. Performance study on different adversarial example attacks

In this part, we look at how the malware detector responses to different adversarial attacks in experiments. As shown in Fig. 6, Fig. 7 and Fig. 8, we can clearly find that the the evasion attack seems to be the most effective adversarial attack on the malware detector, because the precision, recall and F-measure will all become 0%. The data poisoning attack has some adversarial impacts on the malware detector, which makes its precision, recall and F-measure drop to around 75%. Compared with these two attacks, the malware detector achieves a higher score of over 97% when no adversarial attack is launched. Therefore, it is clear that the adversarial example attacks do have a severe impact on the malware detection systems for Android.

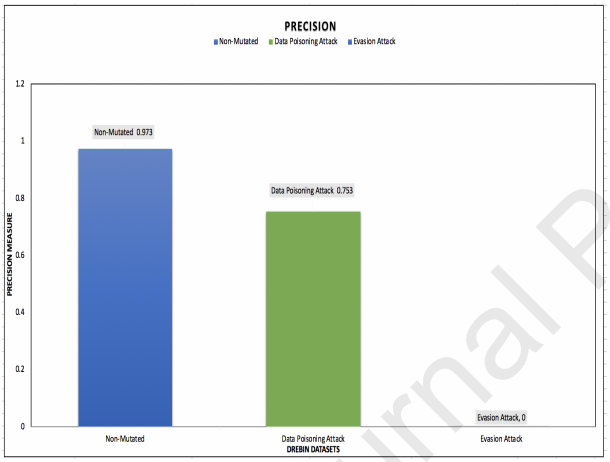

Fig. 6. Precision of no attack V.S. data poisoning attack V.S. evasion attack

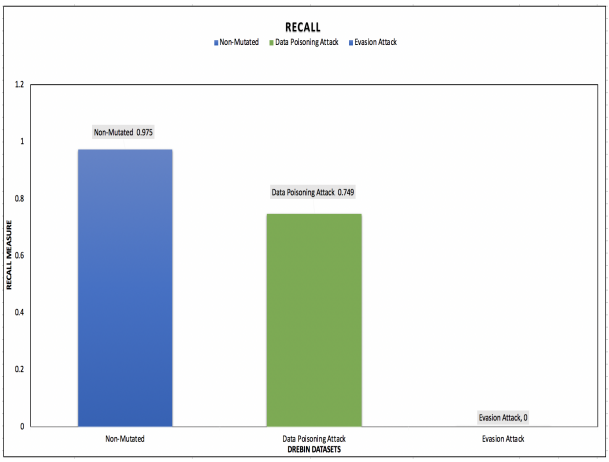

Fig. 7. Recall of no attack V.S. data poisoning attack V.S. evasion attack

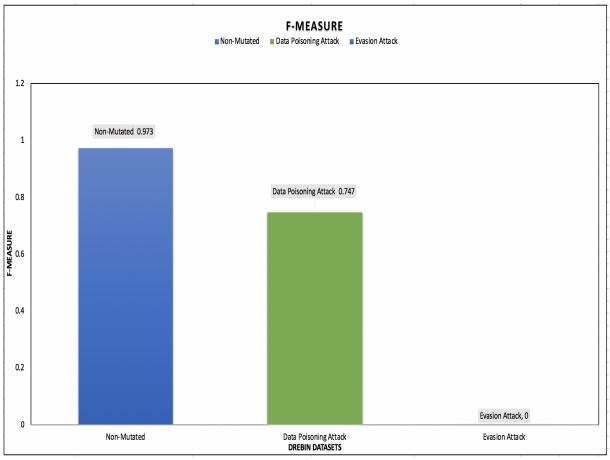

Fig. 8. F-Measure of no attack V.S. data poisoning attack V.S. evasion attack

6. Conclusion

In this research, we study several well-known adversarial example attacks and their impacts on the Android malware detection systems. Within these attacks, we primarily focus on the data poisoning attack and the evasion attack, and implement two attack models so that they could be launched to real Android applications. To observe the effect of the two attacks, some experiments have been conducted. Experimental results clearly show that the two adversarial example attacks can severely degrade the performance of malware detection systems.

As for future directions, we would like to explore more toward how to defend against the adversarial example attacks, especially the evasion attack, and make the malware detection systems more robust. In addition, we want to test the evasion and data poisoning attack models with other machine learning techniques, such as deep learning, which is an emerging machine learning technique that has been widely used in recent years. We feel that deep learning may also help better cope with the adversarial example attacks.

References

[1] S. O'Dea, Market share of mobile operating systems worldwide 2012-2020, https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-200 accessed: 2020-09-10.

[2] M. Ali, D. Svetinovic, Z. Aung, S. Lukman, Malware detection in android mobile platform using machine learning algorithms, 2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS).

[3] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, F. Roli, Yes, machine learning can be more secure! a case study on android malware detection, IEEE Transactions on Dependable and Secure Computing 16 (4) (2019) 711 – 724.

[4] J. Sahs, L. Khan, A machine learning approach to android malware detection, 2012 European Intelligence and Security Informatics Conference.

[5] C. Gibler, J. Crussell, J. Erickson, H. Chen, Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale, in: Proceedings of the 5th International Conference on Trust and Trustworthy Computing, TRUST'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 291–307. doi:10.1007/978-3-642-30921-2_17.
URL http://dx.doi.org/10.1007/978-3-642-30921-2_17

[6] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, Riskranker: Scalable and accurate zero-day android malware detection, in: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, ACM, New York, NY, USA, 2012, pp. 281–294. doi:10.1145/2307636.2307663.
URL http://doi.acm.org/10.1145/2307636.2307663

[7] L. Chen, S. Hou, Y. Ye, Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks, 33rd Annual Computer Security Applications Conference (2017) 362–372.

[8] L. Chen, S. Hou, Y. Ye, S. Xu, Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks, 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (2018) 782–789.

[9] L. K. Yan, H. Yin, Droidscope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis, in: Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 569–584.

[10] G. Dini, F. Martinelli, A. Saracino, D. Sgandurra, Madam: a multi-level anomaly detector for android malware, in: International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, Springer, 2012, pp. 240–253.

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

[12] W. Li, J. Ge, G. Dai, Detecting malware for android platform: An svm-based approach, in: 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, 2015, pp. 464–469. doi:10.1109/CSCloud.2015.50.

[13] K. Zhao, D. Zhang, X. Su, W. Li, Fest: A feature extraction and selection tool for android malware detection, in: 2015 IEEE Symposium on Computers and Communication (ISCC), 2015, pp. 714–720. doi:10.1109/ISCC.2015.7405598.

[14] M. S. Alam, S. T. Vuong, Random forest classification for detecting android malware, in: 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, 2013, pp. 663–669. doi:10.1109/GreenCom-iThings-CPSCom.2013.122.

[15] S. Y. Yerima, S. Sezer, G. McWilliams, I. Muttik, A new android malware detection approach using bayesian classification, in: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 121–128. doi:10.1109/AINA.2013.88.

[16] Z. Wang, J. Cai, S. Cheng, W. Li, Droiddeeplearner: Identifying android malware using deep learning, in: 2016 IEEE 37th Sarnoff Symposium, 2016, pp. 160–165. doi:10.1109/SARNOF.2016.7846747.

[17] X. Su, D. Zhang, W. Li, K. Zhao, A deep learning approach to android malware feature learning and detection, in: 2016 IEEE Trustcom/BigDataSE/ISPA, 2016, pp. 244–251. doi:10.1109/TrustCom.2016.0070.

[18] W. Li, Z. Wang, J. Cai, S. Cheng, An android malware detection approach using weight-adjusted deep learning, in: 2018 International Conference on Computing, Networking and Communications (ICNC), 2018, pp. 437–441. doi:10.1109/ICCNC.2018.8390391.

[19] T. Kim, B. Kang, M. Rho, S. Sezer, E. G. Im, A multimodal deep learning method for android malware detection using various features, IEEE Transactions on Information Forensics and Security 14 (3) (2019) 773–788.

[20] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, K. Ren, Android hiv: A study of repackaging malware for evading machine-learning detection, IEEE Transactions on Information Forensics and Security 15 (2019) 987 – 1001.

[21] W. Yang, D. Kong, T. Xie, C. Gunter, Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps, 33rd Annual Computer Security Applications Conference (2017) 288–302.

[22] W. Li, N. Bala, A. Ahmar, F. Tovar, A. Battu, P. Bambarkar, A robust malware detection approach for android system against adversarial example attacks, in: 2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC), 2019, pp. 360–365.

[23] M. Kermani, S. Kolay, A. Raghunathan, N. Jha, Systematic poisoning attacks on and defenses for machine learning in healthcare, IEEE Journal of Biomedical and Health Informatics 19 (6) (2015) 1893 – 1905.

[24] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Rotaru, B. Li, Manipulating machine learning: Poisoning attacks and countermeasures for regression learning, 2018 IEEE Symposium on Security and Privacy (SP) 1804.00308v1 (2018) 19–35.

[25] S. Chen, M. Xue, L. Fan, L. Ma, Y. Liu, L. Xu, How can we craft large-scale android malware? an automated poisoning attack, IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile).

[26] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, J. D. Tygar, Adversarial machine learning, in: Proceedings of the 4th ACM workshop on Security and artificial intelligence, 2011, pp. 43–58.

[27] I. M. Alabdulmohsin, X. Gao, X. Zhang, Adding robustness to support vector machines against adversarial reverse engineering, in: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, 2014, pp. 231–240.

[28] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, Y. Vorobeychik, Improving robustness of {ML} classifiers against realizable evasion attacks using conserved features, in: 28th {USENIX} Security Symposium ({USENIX} Security 19), 2019, pp. 285–302.

[29] H. C. Wu, R. W. P. Luk, K. F. Wong, K. L. Kwok, Interpreting tf-idf term weights as making relevance decisions, ACM Transactions on Information Systems (TOIS) 26 (3) (2008) 13.

[30] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The weka data mining software: an update, ACM SIGKDD explorations newsletter 11 (1) (2009) 10–18.

[31] C.-C. Chang, C.-J. Lin, Libsvm: A library for support vector machines, ACM transactions on intelligent systems and technology (TIST) 2 (3) (2011) 27.

[32] A. Calleja, A. Martín, H. Menéndez, J. Tapiador, D. Clark, Picking on the family: Disrupting android malware triage by forcing misclassification, Expert Systems With Applications 95 (2017) 113–126.

[33] J. Garcia, M. Hammad, S. Malek, Lightweight, obfuscation-resilient detection and family identification of android malware, ACM Transactions on Software Engineering and Methodology (TOSEM) 26 (3) (2018) 1–29.

Conflict of Interest for Wenjia Li

Anupam Joshi, UMBC, USA

Tim Finin, UMBC, USA

Houbing Song, Embry-Riddle Aeronautical University, USA

Yupeng Hu, Hunan University, China

Feng Zeng, Central South University, China

Yehua Wei, Hunan Normal University, China

Yuansheng Luo, Changsha University of Science and Technology, China

N. Sertac Artan, NYIT, USA

Shenglong Zhang, NYIT, USA

Xin Su, Hunan Police Academy, China

Wei Wei, Xi'an University of Technology, China