An Adversarial Machine Learning Model Against Android Malware Evasion Attacks

Lingwei Chen[1], Shifu Hou[1], Yanfang Ye[1(✉)], and Lifei Chen[2]

[1] Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506, USA
{lgchen,shhou}@mix.wvu.edu, yanfang.ye@mail.wvu.edu
[2] School of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350117, Fujian, China
clfei@fjnu.edu.cn

Abstract. With explosive growth of Android malware and due to its damage to smart phone users, the detection of Android malware is one of the cybersecurity topics that are of great interests. To protect legitimate users from the evolving Android malware attacks, systems using machine learning techniques have been successfully deployed and offer unparalleled flexibility in automatic Android malware detection. Unfortunately, as machine learning based classifiers become more widely deployed, the incentive for defeating them increases. In this paper, we explore the security of machine learning in Android malware detection on the basis of a learning-based classifier with the input of Application Programming Interface (API) calls extracted from the smali files. In particular, we consider different levels of the attackers' capability and present a set of corresponding evasion attacks to thoroughly assess the security of the classifier. To effectively counter these evasion attacks, we then propose a robust secure-learning paradigm and show that it can improve system security against a wide class of evasion attacks. The proposed model can also be readily applied to other security tasks, such as anti-spam and fraud detection.

Keywords: Adversarial machine learning · Android malware detection · Evasion attack

1 Introduction

Due to their mobility and ever expanding capabilities, smart phones have been widely used to perform the tasks, such as banking and automated home control, in people's daily life. In recent years, there has been an exponential growth in the number of smart phone users around the world and it is estimated that 77.7% of all devices connected to the Internet will be smart phones in 2019 [13]. Designed as an open, free, and programmable operation system, Android as one of the most popular smart phone platforms dominates the current market share [1]. However, the openness of Android not only attracts the developers for

© Springer International Publishing AG 2017
S. Song et al. (Eds.): APWeb-WAIM 2017 Workshops, LNCS 10612, pp. 43–55, 2017.
https://doi.org/10.1007/978-3-319-69781-9_5

producing legitimate applications (apps), but also attackers to deliver malware (short for ***ma***licious soft***ware***) onto unsuspecting users to disrupt the mobile operations. Today, a lot of android malware (e.g., Geinimi, DriodKungfu and Hongtoutou) is released on the markets [25], which poses serious threats to smart phone users, such as stealing user information and sending SMS advertisement spams without the user's permission [9]. According to Symantec's recent Internet Security Threat Report [21], one in every five Android apps were actually malware. To protect legitimate users from the attacks of Android malware, intelligent systems using machine learning techniques have been successfully developed in recent years [11,12,23,24,26,29]. Though these machine learning techniques offer exceptional performance in automatic Android malware detection, machine learning itself may open the possibility for an adversary who maliciously "mistrains" a classifier (e.g., by changing data distribution or feature importance) in a detection system. When the learning system is deployed in a real-world environment, it is of a great interest for the attackers to actively manipulate the data to make the classifier producing minimum true positive (i.e., maximumly misclassifying Android malware as benign).

Android malware attackers and anti-malware defenders are actually engaged in a never-ending arms race, where both the attackers and defenders analyze the vulnerabilities of each other, and develop their own optimal strategies to overcome the opponents [5,18]. For example, attackers use repackaging and obfuscation to evade the anti-malware venders' detection. Though, the issues of machine learning security are starting to be leveraged [3,4,6,7,16,19,20,30], most existing researches for adversarial machine learning focus in the area of spam email detection, but rarely for Android malware detection. However, with the popularity of machine learning based detections, such adversaries will sooner or later present. In this paper, with the inputs of Application Programming Interface (API) calls extracted from the smali files (smali is an assemble/dissembler for the Dalvid executable (dex) files and provides readable code in smali language), we explore the security of machine learning in Android malware detection on the basis of a learning-based classifier. The major contributions of our work can be summarized as follows:

- *Thorough exploration of Android malware evasion attacks under different scenarios:* The attacker may have different levels of knowledge of the targeted learning system [19]. We define a set of evasion attacks corresponding to the different scenarios, and implement a thorough security analysis of a learning-based classifier.
- *An adversarial machine learning model against the evasion attacks:* Based on the learning system tainted by the attacks, we present an adversarial learning model against the evasion attacks in Android malware detection, in which we incorporate evasion data manipulation into the learning algorithm and enhance the robustness of the classifier using the security regularization term over the evasion cost.
- *Comprehensive experimental study on a real sample collection from an anti-malware industry company:* We collect the sample set from Comodo Cloud

Security Center, and provide a series of comprehensive experiments to empirically access the performances of our proposed methods.

The rest of the paper is organized as follows. Section 2 defines the problem of machine learning based Android malware detection. Section 3 describes the evasion attacks under different scenarios and their corresponding implementations. Section 4 introduces an adversarial learning model against the evasion attacks. Section 5 systematically evaluates the effectiveness of the proposed methods. Section 6 discusses the related work. Finally, Sect. 7 concludes.

2 Machine Learning Based Android Malware Detection

Based on the collected Android apps, without loss of generality, in this paper, we extract API calls from the smali files as the features, since they are used by the apps in order to access operating system functionality and system resources and thus can be used as representations of the behaviors of an app [12]. To extract the API calls, the Android app is first unzipped to provide the dex file, and then the dex file is further decompiled into smali code (i.e., the interpreted, intermediate code between Java and DalvikVM [8]) using a well-known reverse engineering tool APKTool [2]. The converted smali code can then be parsed for API call extraction. For example, the API calls of "*Lorg/apache/http/HttpRequest;* $\rightarrow$*containsHeader*" and "*Lorg/apache/ http/ HttpRequest;* $\rightarrow$*addHeader*" can be extracted from "winterbird.apk" (MD5: *53cec6444101d19 76af1b253ff5b2226*) which is a theme wallpaper app embedded with malicious code that can steal user's credential. Note that other feature representations, either static or dynamic, are also applicable in our further investigation. Resting on the extracted API calls, we denote our dataset D to be of the form $D = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ of n apps, where $\mathbf{x}_i$ is the set of API calls extracted from app i, and y_i is the class label of app i, where $y_i \in \{+1, -1, 0\}$ ($+1$ denotes malicious, -1 denotes benign, and 0 denotes unknown). Let d be the number of all extracted API calls in the dataset D. Each of the app can be represented by a binary feature vector:

$$\mathbf{x}_i =< x_{i1}, x_{i2}, x_{i3}, ..., x_{id} >, \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$, and $x_{ij} = \{0, 1\}$ (i.e., if app i includes API_j, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$).

The problem of machine learning based Android malware detection can be stated in the form of: $f : \mathcal{X} \rightarrow \mathcal{Y}$ which assigns a label $y \in \mathcal{Y}$ (i.e., -1 or $+1$) to an input app $\mathbf{x} \in \mathcal{X}$ through the learning function f. A general linear classification model for Android malware detection can be thereby denoted as:

$$\mathbf{f} = \text{sign}(f(\mathbf{X})) = \text{sign}(\mathbf{X}^T\mathbf{w} + \mathbf{h}), \tag{2}$$

where $\mathbf{f}$ is a vector, each of whose elements is the label (i.e., malicious or benign) of an app to be predicted, each column of matrix $\mathbf{X}$ is the API feature vector of an app, $\mathbf{w}$ is the coefficients and $\mathbf{h}$ is the biases. More specifically, the machine

learning on the basis of a linear classifier can be formalized as an optimization problem [28]:

$$\underset{\mathbf{f},\mathbf{w},\mathbf{h};\boldsymbol{\xi}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{y}-\mathbf{f}||^2+\frac{1}{2\beta}\mathbf{w}^T\mathbf{w}+\frac{1}{2\gamma}\mathbf{h}^T\mathbf{h}+\boldsymbol{\xi}^T(\mathbf{f}-\mathbf{X}^T\mathbf{w}-\mathbf{h}) \tag{3}$$

subject to Eq. 2, where $\mathbf{y}$ is the labeled information vector, $\boldsymbol{\xi}$ is Lagrange multiplier, β and γ are the regularization parameters. Note that Eq. 3 is a typical linear classification model consisting of specific loss function and regularization terms. Without loss of generality, the equation can be transformed into different linear models depending on the choices of loss function and regularization terms.

3 Implementation of Evasion Attacks

Considering that the attackers may know differently about: (i) the feature space, (ii) the training sample set, and (iii) the learning algorithm [19], we characterize their knowledge in terms of a space Γ that encodes knowledge of the feature space X, the training sample set D, and the classification function f. To this end, we present three well-defined evasion attacks to facilitate our analysis: (1) *Mimicry Attacks*: The attackers are assumed to know the feature space and be able to obtain a collection of apps to imitate the original training dataset, i.e., $\Gamma = (X, \hat{D})$. (2) *Imperfect-knowledge Attacks*: In this case, it's assumed that both the feature space and the original training sample set can be fully controlled by the attackers, i.e., $\Gamma = (X, D)$. (3) *Ideal-knowledge Attacks*: This is the worst case where the learning algorithm is also known to the attackers, i.e., $\Gamma = (X, D, f)$, allowing them to perfectly access to the learning system.

3.1 Evasion Cost

Evasion attacks are universally modeled as an optimization problem: given an original malicious app $\mathbf{x} \in \mathcal{X}^+$, the evasion attacks attempt to manipulate it to be detected as benign (i.e., $\mathbf{x}' \in \mathcal{X}^-$), with the minimal evasion cost. Considering API calls of each app is formatted as a binary vector, the cost of feature manipulation (addition or elimination), can be encoded as the distance between $\mathbf{x}$ and $\mathbf{x}'$:

$$c(\mathbf{x}', \mathbf{x}) = ||\mathbf{a}^T(\mathbf{x}' - \mathbf{x})||_p^p, \tag{4}$$

where p is a real number and $\mathbf{a}$ is a weight vector, each of which denotes the relative cost of changing a feature. The cost function can be considered as ℓ_1-norm or ℓ_2-norm depending on the feature space. For the attackers, to evade the detection, adding or hiding some API calls in a malicious app does not seem difficult. However, some specific API calls may affect the structure for intrusive functionality, which may be more expensive to be modified. Therefore we view the evasion cost c practically significant. Accordingly, there is an upper limit of the maximum manipulations that can be made to the original malicious app $\mathbf{x}$. Therefore, the manipulation function $\mathcal{A}(\mathbf{x})$ can be formulated as

$$\mathcal{A}(\mathbf{x}) = \begin{cases} \mathbf{x}' & \text{sign}(f(\mathbf{x}')) = -1 \; and \; c(\mathbf{x}', \mathbf{x}) \leq \delta_{\max} \\ \mathbf{x} & \text{otherwise} \end{cases}, \tag{5}$$

where the malicious app is manipulated to be misclassified as benign only if the evasion cost is less than or equal to a maximum cost $\delta_{\max}$. Let $\mathbf{f}' = \text{sign}(f(\mathcal{A}(\mathbf{X})))$, then the main idea for an evasion attack is to maximize the total loss of classification (i.e., argmax $\frac{1}{2}||\mathbf{y} - \mathbf{f}'||^2$), which means that the more malicious Android apps are misclassified as benign, the more effective the evasion attack could be. An ideal evasion attack modifies a small but optimal portion of features of the malware with minimal evasion cost, while makes the classifier achieve lowest true positive rate.

3.2 Evasion Attack Method

To implement the evasion attack, it's necessary for the attackers to choose a relevant subset of API calls applied for feature manipulations. To evade the detection with lower evasion cost, the attackers may inject the API calls most relevant to benign apps while remove the ones more relevant to malware. To stimulate the attacks, we rank each API call using Max-Relevance algorithm [17] which has been successfully applied in malware detection [27]. Based on a real sample collection of 2,334 Android apps (1,216 malicious and 1,118 benign) obtained from Comodo Cloud Security Center, 1,022 API calls are extracted. Figure 1 shows the Cumulative Distribution Function (CDF) of the API calls' relevance scores, from which we can see that (1) for different API calls, some are explicitly relevant to malware, while some have high influence on benign apps; and (2) those API calls with extremely low relevance scores (below 0.06 in our case) have limited or no contributions in malware detection, thus we exclude them for feature manipulations. Therefore, we rank each API call and group them into two sets for feature manipulations: $\mathcal{M}$ (those highly relevant to malware) and $\mathcal{B}$ (those highly relevant to benign apps) in the descent order of $I(x, +1)$ and $I(x, -1)$ respectively.

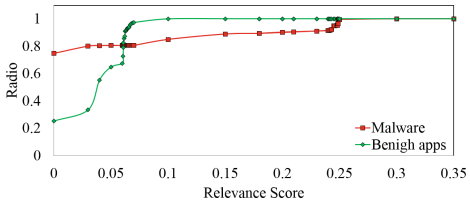

Fig. 1. Relevance score CDF of API calls

To evaluate the capability of an evasion attack, we further define a function $g(\mathcal{A}(\mathbf{X}))$ to represent the accessibility to the learning system by the attack:

$$g(\mathcal{A}(\mathbf{X})) = ||\mathbf{y} - \mathbf{f}'||^2, \tag{6}$$

which implies the number of malicious apps being misclassified as benign. The underlying idea of our evasion attack *EvAttack* is to perform feature manipulations with minimum evasion cost while maximize the total loss of classification in Eq. (6). Specifically, we conduct bi-directional feature selection, that is, forward feature addition performed on $\mathcal{B}$ and backward feature elimination performed on $\mathcal{M}$. At each iteration, an API call will be selected for addition or elimination depending on the fact how it influences the value of $g(\mathcal{A}(\mathbf{X}))$. The evasion attack $\boldsymbol{\theta} = \{\boldsymbol{\theta}^+, \boldsymbol{\theta}^-\}$ will be drawn from the iterations, where $\boldsymbol{\theta}^+, \boldsymbol{\theta}^- \in \{0,1\}^d$ (if API_i is selected for elimination (or addition), then $\theta_i^+(\theta_i^-) = 1$; otherwise, $\theta_i^+(\theta_i^-) = 0$). The iterations will end at the point where the evasion cost reaches to maximum ($\delta_{\max}$), or the features available for addition and elimination are all manipulated. Given $m = \max(|\mathcal{M}|, |\mathcal{B}|)$, *EvAttack* requires $O(n_t m(\mu^+ + \mu^-))$ queries, in which n_t is the number of malicious apps that the attackers want to evade the detection, μ^+ and μ^- are the numbers of selected features for elimination and addition respectively ($\mu^+ \ll d$, $\mu^- \ll d$, $m \ll d$).

4 Adversarial Machine Learning Model Against Evasion Attacks

To be resilient against the evasion attacks, an anti-malware defender may modify features of the training dataset, or analyze the attacks and retrain the classifier accordingly to counter the adversary's strategy [18]. However, these methods typically suffer from assumption of specific attack models or substantial modifications of learning algorithms [15]. In this paper, we provide a systematic model to formalize the impact of the evasion attacks with respect to system security and robustness in Android malware detection. To this end, we perform our security analysis of the learning-based classifier resting on the application setting that the defender draws the well-crafted *EvAttack* from the observed sample space, since the attack is modeled as optimization under generic framework in which the attackers try to (1) maximize the number of malicious apps being classified as benign, and (2) minimizes the evasion cost for optimal attacks over the learning-based classifier [15].

Incorporating the evasion attack $\boldsymbol{\theta}$ into the learning algorithm can be performed through computing a Stackelberg game [20] or adding adversarial samples [15], which enables us to provide a significant connection between training and the adversarial action. An adversarial learning model for Android malware detection is supposed to be more resilient to the evasion attacks. It's recalled that an optimal evasion attack aims to manipulate a subset of the features with minimum evasion cost while maximize the total loss of classification. In contrast, to secure the classifier in Android malware detection, we would maximize the evasion cost for the attacks [30]: from the analysis of the adversary problem [4,14], we can find that the larger the evasion cost, the more manipulations need to be performed, and the more difficult the attack is. Therefore, in our proposed

adversarial learning model, we will not only incorporate the evasion attack $\boldsymbol{\theta}$ into the learning algorithm, but also enhance the classifier by using a security regularization term based on the evasion cost.

We first define the resilience coefficient of a classifier as:

$$s(\mathcal{A}(\mathbf{x}), \mathbf{x}) = 1/c(\mathcal{A}(\mathbf{x}), \mathbf{x}), \tag{7}$$

subject to Eq. 4, which is converse to the evasion cost. We then define a diagonal matrix for the adversary action denoted as $\mathbf{S} \in \mathbb{R}^{n \times n}$, where the diagonal element $S_{ii} = s(\mathcal{A}(\mathbf{x}_i), \mathbf{x}_i)$ and the remaining elements in the matrix are 0. Based on the concept of label smoothness, we can secure the classifier with the constraint as $\mathbf{f'}^T\mathbf{Sy}$. Since the learning based Android malware detection can be formalized as an optimization problem denoted by Eq. 3, we can then bring a security regularization term to enhance its security. This constraint penalizes parameter choices, smooths the effects the attack may cause, and in turn helps to promote the optimal solution for the local minima in the optimization problem. Therefore, to minimize classifier sensitivity to feature manipulation, we can minimize the security regularization term. Based on Eq. 3, we can formulate an adversarial learning model against evasion attacks as:

$$\underset{\mathbf{f'},\mathbf{w},\mathbf{h};\boldsymbol{\xi}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{y} - \mathbf{f'}||^2 + \frac{\alpha}{2}\mathbf{f'}^T\mathbf{Sy} + \frac{1}{2\beta}\mathbf{w}^T\mathbf{w} + \frac{1}{2\gamma}\mathbf{h}^T\mathbf{h} + \boldsymbol{\xi}^T(\mathbf{f'} - \mathbf{X'}^T\mathbf{w} - \mathbf{h}) \tag{8}$$

subject to $\mathbf{f'} = \operatorname{sign}(f(\mathcal{A}(\mathbf{X})))$ and $\mathbf{X'} = \mathcal{A}(\mathbf{X})$, where α is the regularization parameter for the security constraint. To solve the problem in Eq. 8, let

$$\mathcal{L}(\mathbf{f'}, \mathbf{w}, \mathbf{h}; \boldsymbol{\xi}) = \frac{1}{2}||\mathbf{y} - \mathbf{f'}||^2 + \frac{\alpha}{2}\mathbf{f'}^T\mathbf{Sy} + \frac{1}{2\beta}\mathbf{w}^T\mathbf{w} + \frac{1}{2\gamma}\mathbf{h}^T\mathbf{h} + \boldsymbol{\xi}^T(\mathbf{f'} - \mathbf{X'}^T\mathbf{w} - \mathbf{h}). \tag{9}$$

As $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$, $\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = 0$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} = 0$, $\frac{\partial \mathcal{L}}{\partial \mathbf{f'}} = 0$, we have

$$\mathbf{w} = \beta\mathbf{X'}\boldsymbol{\xi}, \tag{10}$$

$$\mathbf{h} = \gamma\boldsymbol{\xi}, \tag{11}$$

$$\mathbf{f'} = \mathbf{X'}^T\mathbf{w} + \mathbf{h}, \tag{12}$$

$$\mathbf{f'} = \mathbf{y} - \frac{\alpha}{2}\mathbf{Sy} - \boldsymbol{\xi}. \tag{13}$$

Based on the derivation from Eqs. 10, 11 and 12, we have

$$\boldsymbol{\xi} = (\beta\mathbf{X'}^T\mathbf{X'} + \gamma\mathbf{I})^{-1}\mathbf{f'}. \tag{14}$$

We substitute Eqs. 14 to 13, then we can get our adversarial model in malware detection (*AdvMD*) as:

$$((\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I}) + \mathbf{I})\mathbf{f}' = (\mathbf{I} - \frac{\alpha}{2}\mathbf{S})(\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I})\mathbf{y}. \tag{15}$$

Since the size of $\mathbf{X}'$ is $d \times n$, the computational complexity for Eq. 15 is $O(n^3)$. If $d < n$, we can follow Woodbury identity [22] to transform Eqs. 15 to 16 and reduce the complexity to $O(d^3)$.

$$(\mathbf{I} + \gamma^{-1}\mathbf{I} - \gamma^{-1}\mathbf{X}'^T(\gamma\beta^{-1}\mathbf{I} + \mathbf{X}'\mathbf{X}'^T)^{-1}\mathbf{X}')\mathbf{f}' = (\mathbf{I} - \frac{\alpha}{2}\mathbf{S})\mathbf{y}. \tag{16}$$

To solve the above secure learning in Eq. 15, conjugate gradient descent method can be applied, which is also applicable to Eq. 16, provided that all the variables are configured as the correct initializations.

5 Experimental Results and Analysis

In this section, three sets of experiments are conducted to evaluate our proposed methods. The real sample collection obtained from Comodo Cloud Security Center contains 2,334 apps (1,216 malicious and 1,118 benign) with 1,022 extracted API calls. In our experiments, we use k-fold cross-validations for performance evaluations. We evaluate the detection performance of different methods using *TP* (true positive), *TN* (true negative), *FP* (false positive), *FN* (false negative), *TPR* (TP rate), *FPR* (FP rate), *FNR* (FN rate), and *ACC* (accuracy).

According to the relevance scores analyzed in Sect. 3.2, those API calls whose relevance scores are lower than the empirical threshold (i.e., 0.06 in our application) will be excluded from consideration. Therefore, $|\mathcal{M}| = 199$, $|\mathcal{B}| = 333$. Considering different weights for API calls, we optimize the weight for each feature ($a \in [0, 1]$). We exploit the average number of API calls that each app possesses (i.e., 127) to define $\delta_{\max}$ based on the CDF drawn from all these numbers. When we set $\delta_{\max}$ as 20% of the average number of API calls that each app possesses (i.e., 25), the average feature manipulation of each app is about 10% of its extracted API calls, which could be considered as a reasonable trade-off to conduct the evasion attacks considering the feature number and manipulation cost. Therefore, we run our evaluation of the proposed evasion attacks with $\delta_{\max} = 25$.

5.1 Evaluation of *EvAttack* Under Different Scenarios

Given $\delta_{\max} = 25$, we first evaluate *EvAttack* under different scenarios: (1) In mimicry (MMC) attack, i.e., $\Gamma = (X, \hat{D})$, we select 200 random apps (100 malware and 100 benign apps) from our collected sample set (excluding those ones for testing) as our mimic dataset and utilize linear Support Vector Machine (SVM) as the surrogate classifier to train these app samples. (2) In imperfect-knowledge (IPK) attack, i.e., $\Gamma = (X, D)$, we implement this attack in a consistent manner as MMC attack where the only difference is that we apply 90%

of the collected apps to train SVM. (3) In Ideal-knowledge (IDK) attack, i.e., $\Gamma = (X, D, f)$, we conduct *EvAttack* based on all the collected apps and the learning model in Eq. 3. Note that, *EvAttack* is applied to all these scenarios resting on the same $\delta_{\max}$, and each attack is performed by 10-fold cross-validation. The experimental results are shown in Table 1, which illustrate that the performance of the attack significantly relies on the available knowledge the attackers have. In ideal-knowledge scenarios, the *FNR* of IDK reaches to 0.7227 (i.e., 72.27% of testing malicious apps are misclassified as benign) that is superior to MMC and IPK.

Table 1. Evaluation of the evasion attacks under different scenarios

Scenarios	TP	FN	ACC	FNR
Before attack	111	8	96.12%	0.0672
MMC attack	85	34	84.91%	0.2857
IPK attack	48	71	68.97%	0.5966
IDK attack	33	86	62.50%	0.7227

5.2 Comparisons of *EvAttack* and Other Attacks

We further compare *EvAttack* with other attack methods including: (1) only injecting API calls from $\mathcal{B}$ (*Method 1*); (2) only eliminating API calls from $\mathcal{M}$ (*Method 2*); (3) injecting $(1/2 \times \delta_{\max})$ API calls from $\mathcal{B}$ and eliminating $(1/2 \times \delta_{\max})$ API calls from $\mathcal{M}$ (*Method 3*); (4) simulating anonymous attack by randomly manipulating API calls for addition and elimination (*Method 4*). The experimental results which average over the 10-fold cross-validations shown in Fig. 2 demonstrate that the performances of the attacks vary when using different feature manipulation methods (*Method 0* is the baseline before attack and *Method 5* denotes the *EvAttack*) with the same evasion cost $\delta_{\max}$: (1) *Method 2* performs worst with the lowest *FNR* which denotes that elimination is not as effective as others; (2) *Method 3* performs better than the methods only applying feature addition or elimination, and the anonymous attack, due to its bi-directional feature manipulation over $\mathcal{B}$ and $\mathcal{M}$; (3) *EvAttack* can greatly improve the *FNR* to 0.7227 and degrade the detection accuracy *ACC* to 62.50%, which outperforms other four feature manipulation methods for its well-crafted attack strategy.

5.3 Evaluation of *AdvMD* Against Evasion Attacks

In this set of experiments, we validate the effectiveness of *AdvMD* to the well-crafted attacks. We use *EvAttack* to taint the malicious apps in the testing set, and access the performances in different ways: (1) the baseline before attack;

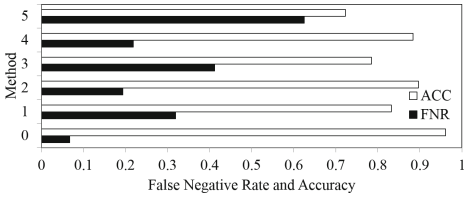

Fig. 2. *FNR*s and accuracies of different attacks.

(2) the classifier under attack; (3) the classifier retrained using the updated training dataset [15,20]; (4) *AdvMD*. We conduct the 10-fold cross-validations, experimental results of different learning models are shown in Fig. 3. Figure 3(a) shows the comparisons of FPR, TPR and ACC for different learning models before/against *EvAttack*, and Fig. 3(b) illustrates the ROC curves of different learning models before/against *EvAttack*. From Fig. 3, we can observe that (i) the retraining techniques can somehow defense the evasion attacks, but the performance still remain unsatisfied; while (ii) *AdvMD* can significantly improve the *TPR* and *ACC*, and bring the malware detection system back up to the desired performance level, the accuracy of which is 91.81%, approaching the detection results before the attack. We also implement the anonymous attack by randomly selecting the features for manipulation. Under the anonymous attack, *AdvMD* has zero knowledge of what the attack is. Even in such case, *AdvMD* can still improve the *TPR* from 0.7815 to 0.8367. Based on these properties, *AdvMD* can be a resilient solution in Andriod malware detection.

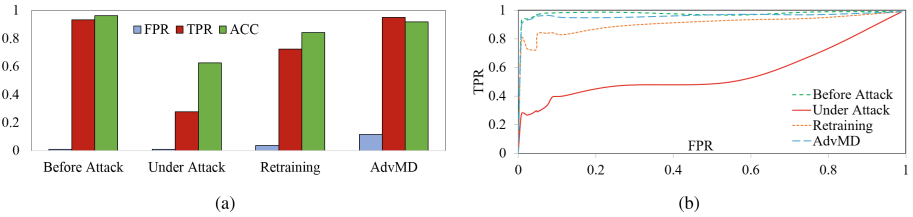

Fig. 3. Comparisons of different learning models before/against *EvAttack*.

6 Related Work

Adversarial machine learning problems are starting to be leveraged from either adversarial or defensive perspectives in some domains, such as anti-spam and intrusion detection. Lowd and Meek [16] introduced an ACRE framework to study how an adversary can learn sufficient information from the features to construct targeted attacks using minimal adversarial cost. Zhang et al. [30],

Li et al. [15], and Biggio et al. [4] took gradient steps to find the closest evasion point $\mathbf{x}'$ to the malicious sample $\mathbf{x}$. Haghtalab et al. [10] proposed to learn the behavioral model of a bounded rational attacker by observing how the attacker responded to three defender strategies. To combat the evasion attacks, ample research efforts have been devoted to the security of machine learning. Wang et al. [20] modeled the adversary action as it controlling a vector $\boldsymbol{\alpha}$ to modify the training dataset $\mathbf{x}$. Debarr et al. [7] explored randomization to generalize learning model to estimate some parameters that fit the data best. Kolcz and Teo [14] investigated a feature reweighting technique to avoid single feature over-weighting. More recently, robust feature selection methods have also been proposed to counter some kinds of evasion data manipulations [20,30]. However, most of these works rarely investigate the security of machine learning in Android malware detection. Different from the existing works, we explore the adversarial machine learning in Android malware detection by providing a set of evasion attacks to access the security of the classifier over different capabilities of the attackers and enhancing the learning algorithm using evasion action and security regularization term.

7 Conclusion and Future Work

In this paper, we take insights into the machine learning based model and its evasion attacks. Considering different knowledge of the attackers, we implement an evasion attack *EvAttack* under three scenarios by manipulating an optimal portion of the features to evade the detection. Accordingly, an adversarial learning model *AdvMD*, enhanced by evasion data manipulation and security regularization term, is presented against these attacks. Three sets of experiments based on the real sample collection from Comodo Cloud Security Center are conducted to empirically validate the proposed approaches. The experimental results demonstrate that *EvAttack* can greatly evade the detection, while *AdvMD* can be a robust and practical solution against the evasion attacks in Android malware detection. In our future work, we will further explore the poisoning attacks in which the attackers alter the training precess through influence over the training data, as well as its resilient detection.

Acknowledgments. The authors would also like to thank the experts of Comodo Security Lab for the data collection and helpful discussions. The work is partially supported by the U.S. National Science Foundation under grant CNS-1618629 and Chinese NSF grant 61672157.

References

1. Android: iOS combine for 91 percent of market. http://www.cnet.com
2. APKTool. http://ibotpeaches.github.io/Apktool/
3. Barreno, M., Nelson, B., Sears, R., Joseph, A.D., Tygar, J.D.: Can machine learning be secure? In: ASIACCS (2006)

4. Biggio, B., Fumera, G., Roli, F.: Evade hard multiple classifier systems. In: Okun, O., Valentini, G. (eds.) Applications of Supervised and Unsupervised Ensemble Methods. Studies in Computational Intelligence, pp. 15–38. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03999-7_2
5. Biggio, B., Fumera, G., Roli, F.: Security evaluation of pattern classifiers under attack. IEEE TKDE **26**(4), 984–996 (2014)
6. Bruckner, M., Kanzow, C., Scheffer, T.: Static prediction games for adversarial learning problems. JMLR **13**, 2617–2654 (2012)
7. Debarr, D., Sun, H., Wechsler, H.: Adversarial spam detection using the randomized hough transform-support vector machine. In: ICMLA 2013, pp. 299–304 (2013)
8. Dex. http://www.openthefile.net/extension/dex
9. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. In: SPSM (2011)
10. Haghtalab, N., Fang, F., Nguyen, T.H., Sinha, A., Procaccia, A.D., Tambe, M.: Three strategies to success: learning adversary models in security games. In: IJCAI (2016)
11. Hou, S., Saas, A., Chen, L., Ye, Y.: Deep4MalDroid: a deep learning framework for android malware detection based on linux kernel system call graphs. In: WIW (2016)
12. Hou, S., Saas, A., Ye, Y., Chen, L.: DroidDelver: an android malware detection system using deep belief network based on API call blocks. In: Song, S., Tong, Y. (eds.) WAIM 2016. LNCS, vol. 9998, pp. 54–66. Springer, Cham (2016). doi:10.1007/978-3-319-47121-1_5
13. IDC. http://www.idc.com/getdoc.jsp?containerId=prUS25500515
14. Kolcz, A., Teo, C.H.: Feature weighting for improved classifier robustness. In: CEAS 2009 (2009)
15. Li, B., Vorobeychik, Y., Chen, X.: A general retraining framework for adversarial classification. In: NIPS 2016 (2016)
16. Lowd, D., Meek, C.: Adversarial learning. In: KDD, pp. 641–647 (2005)
17. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. **27**(8), 1226–1238 (2005)
18. Roli, F., Biggio, B., Fumera, G.: Pattern recognition systems under attack. In: Ruiz-Shulcloper, J., Sanniti di Baja, G. (eds.) CIARP 2013. LNCS, vol. 8258, pp. 1–8. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41822-8_1
19. Šrndic, N., Laskov, P.: Practical evasion of a learning-based classifier: a case study. In: SP (2014)
20. Wang, F., Liu, W., Chawla, S.: On sparse feature attacks in adversarial learning. In: ICDM 2014 (2014)
21. Wood, P.: Internet Security Threat Report 2015. Symantec, California (2015)
22. Woodbury, M.A.: Inverting modified matrices. Statistical Research Group, Princeton University, Princeton, NJ (1950)
23. Wu, D., Mao, C., Wei, T., Lee, H., Wu, K.: DroidMat: android malware detection through manifest and API calls tracing. In: Asia JCIS (2012)
24. Wu, W., Hung, S.: DroidDolphin: a dynamic Android malware detection framework using big data and machine learning. In: RACS (2014)
25. Xu, J., Yu, Y., Chen, Z., Cao, B., Dong, W., Guo, Y., Cao, J.: MobSafe: cloud computing based forensic analysis for massive mobile applications using data mining. Tsinghua Sci. Technol. **18**, 418–427 (2013)

26. Yang, C., Xu, Z., Gu, G., Yegneswaran, V., Porras, P.: DroidMiner: automated mining and characterization of fine-grained malicious behaviors in android applications. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8712, pp. 163–182. Springer, Cham (2014). doi:10.1007/978-3-319-11203-9_10
27. Ye, Y., Li, D., Li, T., Ye, D.: IMDS: intelligent malware detection system. In: KDD 2007 (2007)
28. Ye, Y., Li, T., Zhu, S., Zhuang, W., Tas, E., Gupta, U., Abdulhayoglu, M.: Combining file content and file relations for cloud based malware detection. In: KDD 2011, pp. 222–230 (2011)
29. Yuan, Z., Lu, Y., Wang, Z., Xue, Y.: Droid-Sec: deep learning in android malware detection. In: SIGCOMM (2014)
30. Zhang, F., Chan, P.P.K., Biggio, B., Yeung, D.S., Roli, F.: Adversarial feature selection against evasion attacks. IEEE Trans. Cybern. **46**(3), 766–777 (2015)