

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

HAREL BERGER*, Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Israel

AMIT DVIR, Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Israel

CHEN HAJAJ, Ariel Cyber Innovation Center, Data Science and Artificial Intelligence Research Center, Industrial Engineering and Management Department, Ariel University, Israel

RONY RONEN, Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Israel

Android malware is a spreading disease in the virtual world. Anti-virus and detection systems continuously undergo patches and updates to defend against these threats. Most of the latest approaches in malware detection use Machine Learning (ML). Against the robustifying effort of detection systems, raise the *evasion attacks*, where an adversary changes its targeted samples so that they are misclassified as benign. This paper considers two kinds of evasion attacks: feature-space and problem-space. *Feature-space* attacks consider an adversary who manipulates ML features to evade the correct classification while minimizing or constraining the total manipulations. *Problem-space* attacks refer to evasion attacks that change the actual sample. Specifically, this paper analyzes the gap between these two types in the Android malware domain. The gap between the two types of evasion attacks is examined via the retraining process of classifiers using each one of the evasion attack types. The experiments show that the gap between these two types of retrained classifiers is dramatic and may increase to 96%. Retrained classifiers of feature-space evasion attacks have been found to be either less effective or completely ineffective against problem-space evasion attacks. Additionally, exploration of different problem-space evasion attacks shows that retraining of one problem-space evasion attack may be effective against other problem-space evasion attacks.

ACM Reference Format:

Harel Berger, Amit Dvir, Chen Hajaj, and Rony Ronen. 2022. **Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks**. *ACM Trans. Priv. Sec.* 1, 1 (May 2022), 33 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Malicious files are a nuisance for ordinary users and security experts alike. However, the task of generating suitable and simple defense systems against the spreading disease of malicious files around the world is on the experts' shoulders. The main course of action for malware detection in recent years is Machine Learning [1–6]. Several malware ML-based

*Corresponding author.

Authors' addresses: Harel Berger, harel.berger@mmail.ariel.ac.il, Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Ariel, Israel, Israel; Amit Dvir, amitdv@g.ariel.ac.il, Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Ariel, Israel, Israel; Chen Hajaj, chenha@ariel.ac.il, Ariel Cyber Innovation Center, Data Science and Artificial Intelligence Research Center, Industrial Engineering and Management Department, Ariel University, Ariel, Israel, Israel; Rony Ronen, ronen@mmail.ariel.ac.il, Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Ariel, Israel, Israel.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

Manuscript submitted to ACM

Manuscript submitted to ACM

1

detection systems implement static analysis of the components of the application [2, 7–9]. Other methods explore API sequences or the structure of the commands inside the app [10–13].

Some of the ML-based detection systems were proven by Goodfellow et al. [14] to be vulnerable to manipulations. *Adversarial examples*, the formal terminology of these manipulations, are generated by an adversary that changes samples to achieve misclassification [15–17]. *Evasion attacks* (EA), refer to adversaries that change malicious instances such that they will be classified as benign. EAs can be divided into two types: feature-space attacks and problem-space attacks. Feature-space attacks map the malware sample to a feature vector. Then, add perturbations to the values of the vector. Problem-space attacks change the actual sample. Feature-space evasion attacks utilize ML methods that change feature values to evade correct classification. This approach is general and may suit any domain in which an attacker manipulates a malware sample. Still, feature-space attacks do not consider any realistic constraint embedded in the specific malware domain. For example, deleting a sensor from a malicious Android app, thereby setting the associated feature to 0 (from 1) may damage the functionality of the app and consequently disable the malicious activity of the app. In comparison, problem-space attacks are more concrete and require an expert in the specific domain to generate them.

Most malware is engineered for a specific operating system and may be based on the hardware and functionalities of the victim (as shown in [18]). Therefore, suitable solutions are devised for each platform and domain separately. Among other domains, the Android operating system is very attractive to attackers, as it is still the most popular mobile operating system in the world [19]. As a countermeasure, many works in the security community focus on the correct identification of malware lurking in the Android markets [2, 3, 20–27]. Other studies enhance existing work and make them more robust. For example, retraining detection systems on evasion attacks [9, 28–30]. The current work examines the Android OS domain and explores the efficiency of approaches to robust ML detection systems with retraining on different evasion attacks. Compared to previous works, this work explores retraining on both problem-space and feature-space evasion attacks. The identification of gaps between these two types of retraining in this study proves that retraining on different types of evasion attacks creates different improvements in detection rates. Specifically, sufficient improvement was not demonstrated in feature-space attacks. Moreover, in some cases, the initial detection rate decreased in feature-space attacks.

The contribution of this work is twofold: First, this work demonstrates that retraining a basic classifier on feature-space evasion attacks does not gain sufficient robustness against problem-space evasion attacks. As explained above, feature-space evasion attacks create an abstraction of reality. As a consequence, retraining a basic classifier on feature-space evasion attacks does not strengthen the classifier against problem-space evasion attacks on a specific domain. On the other hand, retraining a basic classifier on problem-space evasion attacks in a specific domain (e.g., the Android domain) is efficient for problem-space attacks in the same domain. Second, this work analyzes the effectiveness of different defense methods based on retraining a basic classifier on one problem-space evasion attack against other problem-space evasion attacks. In this work it is proven that some of these defense methods can be generalized. Three detection systems were examined for these analyses: Drebin [2], an additional version of Drebin using a DNN [9], and MaMaDroid [3] (the first and the latter detection systems are two well-known Android malware detection systems).

The remainder of this paper is as follows. First, the ML fundamentals of malware detection are described in Section 2. Then, the aspects of the retraining process are presented in Section 3. The background on APK structure and on feature sets explored in this study is presented in Section 4. An overview of the experimental design is described in Section 5. The results of the experiments of retrained classifiers in both feature-space and problem-space evasion attacks are presented in Sections 6-7. Related work is discussed in Section 8. The conclusions are presented in Section 9.

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

3

2 FUNDAMENTALS OF MACHINE LEARNING AND MALWARE DETECTION

This section surveys the fundamentals of malware detection and machine learning. First, the formalities of ML are presented in Section 2.1. Then, the definitions of evasion attacks and mitigation techniques are described in Sections 2.2-2.3.

2.1 ML Formalities

The input for a classification problem using supervised machine learning is a training dataset $D = \{(x_i, y_i)\}$, where $x_i \in X \subseteq \mathbb{R}^n$ are feature vectors taken from a feature space X , and $y_i \in L$ are object labels from space L . Each sample in D is assumed to be generated i.i.d. from the same unknown distribution P . The goal of this kind of problem is to identify and train a learning model $h \in H$ (where H is the hypothesis/models' space), which has the minimal expected error regarding new samples drawn from the same distribution P .

In malware detection, one of the supervised machine learning applications, the feature vectors are not given. Instead, the initial data set comprises a set of entities, such as PDF files, with their labels. To create the feature vectors, a preprocessing operation of the entities is required. Feature extractors generate the mapping of entities-to-feature vectors. A simple example is the Android version of an APK file, which can be easily obtained from the OS and then translated to a numerical value. Employing a feature extractor for every entity in the dataset, and then associating the entity with its object label, resulting in a dataset D that fits a standard ML system.

2.2 Evasion Attacks

In an *evasion attack*, a learned ML model $h(x)$ is given. This model returns a label $y = h(x)$ for any random feature vector $x \in X$. In the case of Android malware detection, the label is malicious or benign and the feature vector is extracted from an APK using a feature extractor.

The adversary starts with an initial entity e (such as a malicious APK). From this entity, the attacker extracts a feature vector $x = \phi(e)$. The next steps depend on the type of evasion attack that the attacker runs. If the attacker runs a problem-space evasion attack, it manipulates the actual entity e , thus creating e' . This new e' entity, is associated with the correlative feature vector $x' = \phi(e')$. Alternatively, if the attacker runs a feature-space evasion attack, it transforms the feature vector x to x' , which abstractly describes a transformed entity e' , following $e' = \phi^{-1}(x')$ (ϕ^{-1} is the inverse function of ϕ). The goals of these two kinds of attacks are twofold: First, the label of x' , $h(x')$ should return as inaccurate (the attack succeeds). Second, e' should retain the functionality of e . In the case of feature-space attacks, the functionality of x' is abstracted, because running a feature vector x' is not feasible. A cost function, $c(x, x')$, penalizes great modifications of x' . This cost function accounts for various changes that may harm the functionality of the theoretic entity e' .

Creating problem-space evasion attacks is not easy [31–34], as a clear understanding of the malware samples structure is required. Irresponsible transformation of e may harm the functionality of e' . However, some significant works have demonstrated a method for this (e.g., [27, 31, 35, 36]). On the other hand, it is quite easier to run feature-space attacks, because running them only changes numeric values. However, this kind of attack may suffer from the inverse feature-mapping problem [37–44]). The *inverse feature-mapping problem* emphasizes the fact that feature-space attacks utilize an abstraction of reality, using a mapping function ϕ of the entity e to the feature vector x . Manipulations on x by feature-space attacks may not represent feasible changes in the practical domain of the entity e . Therefore, applying the inverse function ϕ^{-1} to a manipulated feature vector x' can result in a non-functional e' .

2.3 Mitigation Techniques

Evasion attacks are a great threat to malware detection systems. Therefore, multiple research studies have explored the attacks’ mitigation techniques (e.g., [6, 45–54]). Among these approaches, there are three that should be noted: (a) game-theoretic reasoning, (b) robust optimization, and (c) iterative adversarial retraining. Some of these approaches cannot be used on every type of ML detection system, such as robust optimization, because solving these problems requires a special structure, such as a continuous feature space [45–47]. On the other hand, iterative adversarial retraining does not include any assumption about the detection system or the attacker [54, 55]. In light of the fact that this study involves problem-space evasion attacks and different types of feature values (binary and real numbers), iterative adversarial retraining is the only defense mechanism that can be applied to each of the learning models.

3 ASPECTS OF ROBUSTIFICATION ATTEMPTS

This work explores two main aspects:

- (1) *Validation of feature-space retraining*: Evaluation of the robustness gained by feature-space evasion attacks retraining in the presence of problem-space evasion attacks.
- (2) *Generalizability of evasion attacks*: Examining the generalizability of retraining processes against evasion attacks.

These aspects are explored using the abstraction of a defender and an attacker¹. The defender chooses an ML defense, which in this work is a learned model $h(x)$. The attacker reacts to this defense with an attack $O(h; D)$, namely an attack that generates manipulated samples from a given dataset D , and the learned model h . A measurement, u , is formulated, using the formula $u(h; O(h; D))$. This formula depicts the accuracy of the classifier h on the manipulated samples produced by the attack, $O(h; D)$. The goal of the defender is to optimize its defense, using the following optimization problem:

$$\max_h u(h; O(h; D)). \quad (1)$$

Iterative adversarial retraining (in short, iterative retraining) is used to solve the optimization problem (1). In this method, malicious instances are manipulated by $O(h; D)$ and iteratively added to the training data. Then, the classifier is adjusted by retraining with the new training data. Thus, the classifier learns the evasion attack. However, using a large number of iterations may create a tendency toward this attack. In other words, the model may be overfitting to this attack. Consequently, the classification of clean data may be damaged. Therefore, the number of iterations should be chosen in correlation with the effect on the classification of clean data. The iterative adversarial retraining approach was previously proposed to harden classifiers against evasion attacks [34, 55, 57]. The variant of iterative retraining that was used in this work follows the following steps:

- (1) Initial a classifier, h .
- (2) Execute the *evasion attack* on several malicious samples from the training data to generate new feature vectors and test the classifier on them.
- (3) Add all of the new feature vectors to the training data and retrain the classifier.
- (4) Stop after a fixed number of iterations, or when no new samples can be added.

Following the above, the evaluation of the two aspects of the retraining method is now described.

A feature-space evasion attack is used for the evaluation of *validation*, which is termed $\tilde{O}(h; D)$. This attack is tested to check whether it can be a proxy for a problem-space attack, $O(h; D)$. Using the retraining process mentioned above,

¹A similar approach was suggested in previous work, formulated as a Stackelberg game [34, 47, 52, 56].

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

5

a defense against \tilde{O} is constructed; the resulting hardened classifier is termed \tilde{h} . Parallely, a *baseline* h^* is created. This is a robust classifier used against the target problem-space evasion attack O , by means of the same retraining technique, but in this case against the problem-space evasion attack O . Next, the performance of \tilde{h} and h^* against O is compared. If \tilde{h} is found to be ineffective against O , then \tilde{O} is a poor attack proxy. However, if \tilde{h} is found to be robust to the problem-space attack O , \tilde{O} is a sufficient proxy for the target evasion attack. A graphic description of the validation process can be found in Fig. 1 for better clearance. Evaluation of the validation aspect is discussed in Section 6.

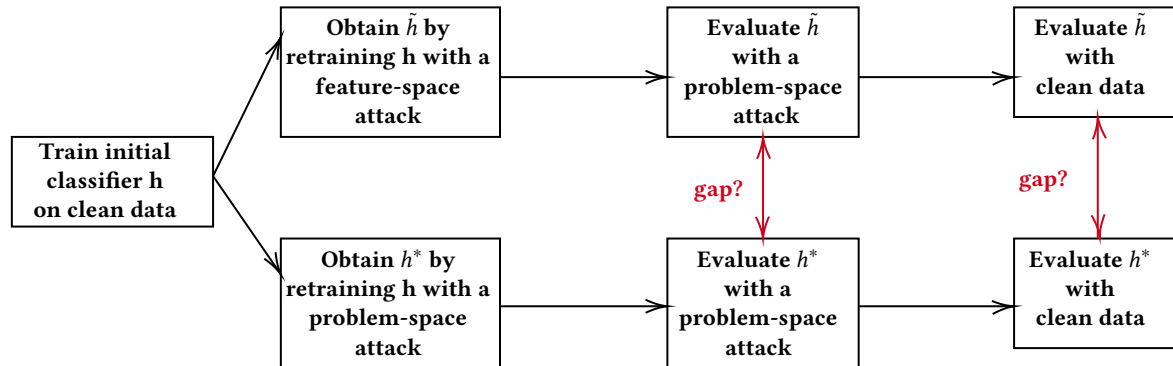


Fig. 1. **Validation** The initial classifier is retrained in two different ways, one by a problem-space evasion attack and another one by a feature-space evasion attack, thus creating two retrained classifiers. The performance of the two classifiers is evaluated, both on problem-space evasion attack and on clean data.

The *generalizability* is evaluated slightly different from the *validation* evaluation. Again, \tilde{O} , a proxy attack, is considered for a problem-space evasion attack. However, in this case, it can be a feature-space or a problem-space attack. A defense \tilde{h} against \tilde{O} is constructed using the same process as before. For this evaluation, instead of utilizing one evasion attack O , a set of target attacks $\{O_i\}$ is tested against \tilde{h} . A proxy attack \tilde{O} is generalizable if \tilde{h} remains robust to at least most of the set of attacks $\{O_i\}$; If not, \tilde{O} is declared non-generalizable. Generalizability is the main topic of Section 7. A graphic description of the evaluation of the generalizability aspect can be found in Fig. 2 for better clearance.

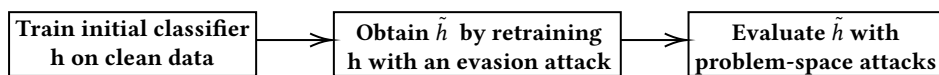


Fig. 2. **Generalizability** The initial classifier is retrained with an evasion attack (problem-space or feature-space), thus creating a retrained classifier. The performance of the retrained classifier is evaluated with a set of problem-space evasion attacks.

4 BACKGROUND

4.1 APK Structure

The Android PacKage (APK) is a popular format used to encapsulate and distribute Android OS's applications. It is in fact a compressed file containing the following main files/directories: *manifest*, *classes.dex*, *layout files*, *res*, and *assets*. The manifest file contains vital information to run the APK, such as the permissions that must be granted to run the application. The binary code is encapsulated in *classes.dex*, which can be translated to various reverse engineering

languages (e.g., Smali). The layout files define how the graphic components are ordered on each page of the application (i.e., activity). The res and assets directories include resources and assets which are not code files, such as pictures and videos. A more detailed explanation can be found in [58].

4.2 Feature Sets and Targeted Classifiers

There are various detection systems for Android malware. Two well-known detection systems, Drebin [2] and MaMaDroid [11]², are mainly distinguished by their feature sets, since their underlining learning models are standard (even though they are different models). The feature sets represent different aspects of Android applications. Drebin analyzes static content from the application and creates a feature vector of the absence/presence of specific components/commands from the app. Therefore, the set of derived features of Drebin is binary. MaMaDroid creates a control flow graph of the app, then analyzes it using Hidden-Markov chains to obtain the transition probability between families/packages inside the app. Consequently, the underlining feature set of MaMaDroid holds real values. The feature sets of these two detection machines were used in this study.

5 EXPERIMENTAL METHODOLOGY

This section describes the experimental methodology of this work³. First, the evasion attacks (both problem-space and feature-space) assessed in this study are discussed (Sections 5.1-5.2). Then, the datasets are presented (Section 5.3). At last, the evaluation metrics used in the experiments are reported (Sections 5.4).

5.1 Problem-Space Evasion Attacks

This section describes the problem-space attacks used in iterative retraining. Most of the evasion attacks in the literature are feature-space attacks. It is more complicated to create a fully functional evasion attack [31–34]. In this study, it is assumed that the attacker has no access to any information except for its malicious samples and the feature set of the targeted system. Other studies have different assumptions. For example, the attacker model proposed by Pierazzi et al. [43] incorporates an attacker with *perfect knowledge*. In other words, the attacker runs a white-box attack and has access to the training data, the feature set, the learning algorithm, and the loss function. In reality, most simple attackers do not have this kind of access. This current study uses *zero-knowledge* attack models, with the addition of the knowledge of the feature set. Consequently, several recent problem-space evasion attacks (e.g. [43]) are not examined in this study, as they define different attacker models. Additional attacks that add non-operational noise to the APK (e.g. [35]) are not considered as well. This study explores attacks that modify the code of the application. Adding no-ops is a different approach, which is more recognizable with static analysis of the application (as argued in [27]). Therefore, this kind of evasion attack was not examined as well.

As the target classifiers of this study analyze different types of features, each target classifier is retrained with other problem-space evasion attacks. First, four evasion attacks against Drebin and Drebin-DNN are described, followed by the evasion attacks against MaMaDroid. Drebin analyzes both the Smali code files and the manifest file (in comparison, MaMaDroid only analyzes the Smali code files). Thus, Drebin and its deep learning version, Drebin-DNN, create an extended environment for attacks.

²These renowned detection systems have played a pivotal role in recent work on Android malware detection [6–9, 27, 31, 35, 59–65].

³An implementation of this work is available at Github [66]. For an access, please contact the authors.

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

7

5.1.1 *DroidChameleon (DC)* [33]. The first problem-space evasion attack in this study is DroidChameleon, a framework for evasion attacks, with various manipulation techniques. The attack that was used in this work is data encryption, which targets classifiers that statically analyze the Smali code files. This attack encrypts only constant strings that are found in the Smali code. This attack is used against the Drebin and Drebin-DNN classifiers.

5.1.2 *Random SB*. Given the feature set of Drebin, this attack enumerates the key features of an application and conceals the appearances of multiple suspicious and restricted API calls, strings, and IP addresses of the application and stores their information in the manifest file. A random subset of items from the list of key features is modified to add randomness to the attack. These items are enumerated based on the feature set of the classifier. In this way, the classifier does not recognize the appearance of these items. For example, the attack replaces an occurrence of the suspicious API call `sendMessage()` with an encoded version of it (`c2VuZFRleHRNZXNzYWdlKCKk=`). It adds a reflection object to the Smali code that decrypts the encrypted content and runs the decrypted API call in runtime. This attack is used against the Drebin and Drebin-DNN classifiers.

5.1.3 *Manifest Based attack (MB)* [27]. In this attack, the attacker blindly changes all of the uses-permission tags in the manifest file to uses-permission-SDK-23 tags. In other words, it hides the permission requests of the app. As proven in [27], the permission requests receive high weights in Drebin and, therefore, are a target for evasion attacks. This attack is used against the Drebin and Drebin-DNN classifiers.

5.1.4 *Random MB*. A random version of the MB attack was used, as well as the original MB attack. In this version, a random subset of the permission requests of each app is concealed in a way that is similar to the original MB attack. This attack is used against the Drebin and Drebin-DNN classifiers.

5.1.5 *Structure Break (STB) Evasion Attacks* [6]. The Structure Break evasion attacks, change the structure of an application. Move parts of the Smali code files from their original places in the application to other places and accordingly change each reference to the moved files. This attack targets Control Flow Graph (CFG) based classifiers, and specifically MaMaDroid. Two variants of the STB attack are considered [6], termed Random and Black-Hole Statistical evasion attacks. The difference between the two variants comprises their operation - randomly changing the flow of the app or statistically based on the samples the attacker obtains. These attacks are used against the MaMaDroid classifier.

5.2 Feature-Space Evasion Attacks

The aim of evasion attacks is to manipulate the obtained malicious samples to appear as benign as possible. This target can be translated into an optimization problem in several ways, thus creating feature-space attacks. In this paper, it was translated one way and solved using Coordinate Greedy (Section 5.2.2). Another way to achieve miss-classification is by adding statistical noise. The Salt & Pepper noise attack (Section 5.2.1) was examined, which operates in this manner. Coordinate Greedy and Salt & Pepper are gradient-free. Therefore, they can be applied to both DNN models and more simple ML models. Consequently, they are excellent candidates for this study, which explores different kinds of models. Both attacks can take many unrealistic approaches, as they do not consider the actual meaning of feature values. For example, flipping the bit that correlates to a GPS sensor, does not eliminate the actual use of it in the code. Therefore, as reported in the results, feature-space attacks do not serve as proxies for real problem-space evasion attacks.

5.2.1 Salt & Pepper (SP) [9, 67–69]. In the Salt & Pepper noise attack, noise is added to the original sample until it is miss-classified. First, a random set of elements of the original sample are selected. Then, the values of these elements are minimized or maximized (e.g., flipping ones to zeros in the binary feature values) until a miss-classification is achieved. This attack was used against all models.

5.2.2 Coordinate Greedy (CG) [34, 55, 70]. A multi-objective optimization problem in the feature space formulates the second feature-space evasion attack. This attack is known as Coordinate Greedy:

$$\underset{x}{\text{minimize}} \quad Q(x') = f(x') + \lambda c(x', x) \quad (2)$$

In this optimization problem, x' is a feature vector obtained by an evasion attack that runs on a malicious seed x . Furthermore, $f(x')$ is the score of x' , given by an actual classifier $h(x') = \text{sgn}(f(x'))$. The cost of changing x' into x is defined as $c(x', x)$. Finally, λ is a parameter that limits the transformation cost. The cost function c is computed by the l_2 norm between x' and x : $c(x', x) = \sum_i |x'_i - x_i|^2$. Since the features of Drebin and MaMaDroid are binary or real, the l_2 norm is well received as a cost function.

Equation (2) presents a non-convex optimization problem. A stochastic local search called *Coordinate Greedy* (alternatively known as an iterative improvement) (CG), designed for combinatorial search domains is used to solve it. In light of the fact that CG computes a local optimum, it is common to execute it multiple times with different starting points. A basic implementation of CG from Tong et al. [34] was used against Drebin, Drebin-DNN and MaMaDroid. This version was implemented for DNN models. To address the underlining models of Drebin (an SVM model) and MaMaDroid (RF/KNN/DT models), CG was altered to fit models that output infinite score values of Drebin and finite values (between 0-1) of MaMaDroid.

5.3 Datasets

The APK file collection was obtained from the AndroZoo dataset [71] (specifically from the Google Play market [72]) and the Drebin data set [2]. The benign APKs were gathered from AndroZoo, and the malicious ones⁴ from Drebin. This dataset includes ~75,000 benign apps, and ~5,700 malicious apps. This ratio is according to a recent well-known work on benign/malicious populations of Android applications [60].

From this dataset, ~4,300 malicious and ~60,000 benign APKs were used as training data, and another ~1,200 malicious and ~15,000 benign files as clean test data, which is another term for non-adversarial data. From the malicious training data, 40 seeds were selected as the retraining seeds and 100 seeds from the test data were chosen as the test seeds. Only 40 malicious seeds were used in the retraining process to make the experiment more realistic. This is because in reality a small amount of malicious data is obtained and adapted by the classifier. This set of 40 samples was sufficient to make the basic models robust against evasion attacks. In the evaluation phase, 100 malicious seeds were used. Each seed in the retraining and evaluation phases was manipulated by a problem-space evasion attack.

The retraining and evaluation experiments were run on a Linux server (Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz, 48 cores, and 128 GB memory, running Ubuntu 18.04).

⁴Recent works [9, 31, 73–75] used a similar approach, choosing the Drebin dataset as their source for malicious APKs and AndroZoo as their source of benign apps.

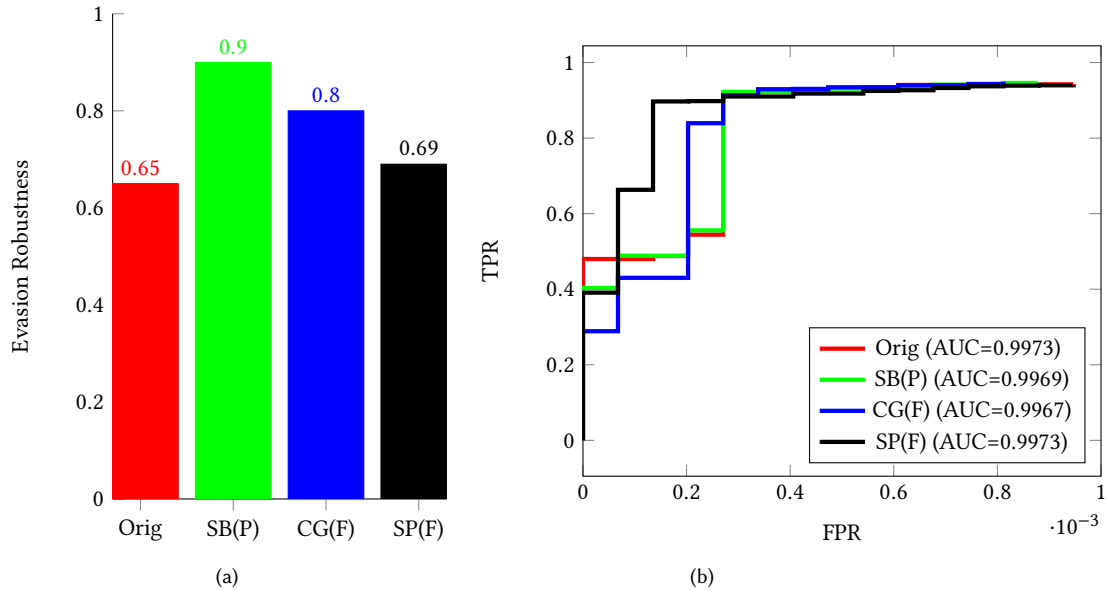


Fig. 3. Evasion robustness of the Random SB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin detection system.

5.4 Evaluation Metrics

Two evaluation metrics were used in this work: evasion robustness and traditional evaluation (as in [34, 76–78]). The evaluation of evasion robustness was performed using 100 malicious APK seeds that were transformed using problem-space evasion attacks. Evasion robustness of 0% means that the detection system did not identify any manipulated sample, while the evasion robustness of 100% means that the classifier detected each manipulated sample. The traditional evaluation metric was used as well, with clean test data, both of malicious APK and benign APK origin. On these data, the ROC (receiver operating characteristic) curve and the corresponding AUC (area under the curve) are computed.

6 VALIDATION OF FEATURE-SPACE RETRAINING

This paper assesses the difference in robustness of detection systems based on three different methodologies: a. simply training the system with the samples from the dataset; b. iteratively retraining the classifier using manipulated instances generated by a problem-space attack; c. iteratively retraining the classifier using manipulated instances generated by a feature-space attack. As the evaluation focuses on problem-space attacks, it is clear that the second methodology will perform best and serve as the upper bound. Thus, it is set as a baseline, to assess the degradation using either of the other two methodologies. In each evaluation, the retrained classifier on an evasion attack is signed by the name of the evasion attack (or an acronym) and the first letter of the attack type. For example, DC(P) stands for the retrained classifier on the DroidChameleon attack, which is a problem-space evasion attack. Similarly, CG(F) refers to the retrained classifier on the CG attack, which is a feature-space evasion attack. . In order to get a clearer view of the differences between the curves, the scale of the ROC curve was changed to 0-0.001. A close examination shows that a wider scale did not depict any difference between the explored classifiers. Three ML detection systems are evaluated in this section as a proof of

concept: Drebin, Drebin-DNN, and MaMaDroid. Each section describes the experiments that were carried out on the specific classifier.

6.1 Drebin

The first set of experiments was carried out on an SVM model, trained on the set of features of the well-known Android malware classifier, Drebin [2]. In these experiments, the C parameter was optimized using CV on clean data to the value of 1. For every experiment, the basic model was the SVM, retrained using different problem-space evasion attacks: Random SB (Section 6.1.1), DroidChameleon (Section 6.1.1), MB (Section 6.1.2), and Random MB (Section 6.1.3). In these experiments, the CG and SP feature space attacks were considered, while setting $\lambda = 0.005$ for the CG attack and $\epsilon = 1000$ for the SP attack (both were the defaults of each implementation [34, 79]). A short discussion (Section 6.1.5) concludes this section.

6.1.1 Random SB Experiment. First, the Random SB attack was run against the original Drebin classifier without any retraining process, and the classifier demonstrated an evasion robustness of 65%. Next, to create the baseline, Drebin was iteratively retrained with the Random SB evasion attack. As can be seen in Fig. 3a, the retrained classifier SB(P) achieved an evasion robustness of 90%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 80% and 69%, respectively. This shows that defenses that rely on acknowledging feature-space evasion attacks do not sufficiently identify the Random SB attack. Also in this experiment, the CG(F) retrained classifier outperforms the SP(F) retrained classifier by 11%. Furthermore, the three retrained classifiers were found to be basically as accurate as the original Drebin classifier on clean data, with an Area under the ROC Curve (AUC) of more than 99% (Fig. 3b). Therefore, it is proven that a highly robust classifier for the Random SB attack (the SB(P) classifier) can be achieved without significantly harming its effectiveness on non-adversarial data.

6.1.2 DroidChameleon Experiment. First, the DroidChameleon attack was run on the original Drebin classifier without any retraining process, and the classifier demonstrated an evasion robustness of 80%. Next, Drebin was iteratively retrained with the DroidChameleon evasion attack to create the baseline. As depicted in Fig. 4a, the DC(P) retrained classifier achieved an evasion robustness of 92%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 78% and 85%, respectively. The retraining process with the CG attack decreased the evasion robustness of the initial classifier by 2%. This shows that a defense that relies on recognizing feature-space evasion attacks will not correctly identify the DroidChameleon attack. Also in this experiment, the SP(F) retrained classifier outperformed the CG(F) retrained classifier by 8%. Additionally, the three retrained classifiers were basically as accurate as the original Drebin classifier on clean data, with an AUC of more than 99% (Fig. 4b). Therefore, it is proven that a highly robust classifier to the DroidChameleon attack (the DC(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

6.1.3 MB Experiment. As a first step, the MB attack was run on the original Drebin classifier without any retraining process, and the classifier demonstrated an evasion robustness of 42%. Then, to create the baseline, Drebin was iteratively retrained with the MB evasion attack. As illustrated in Fig. 5a, the MB(P) retrained classifier achieved an evasion robustness of 88%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 53% and 55%, respectively. This time, the retraining processes are more distinguishable, with a wide gap of 33% in evasion robustness, between the CG(F) and SP(F) on the one hand, and MB(P) on the other hand. This shows that a defense that relies on recognizing feature-space evasion attacks will not identify the MB attack. In addition, the CG(F) and SP(F)

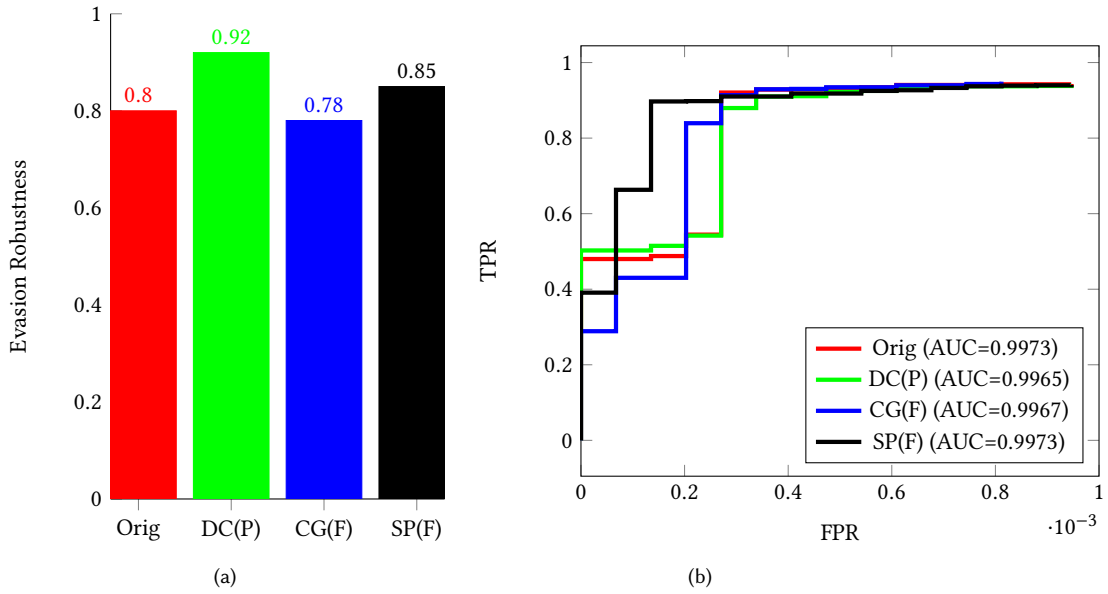


Fig. 4. Evasion robustness of the DroidChameleon attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin detection system.

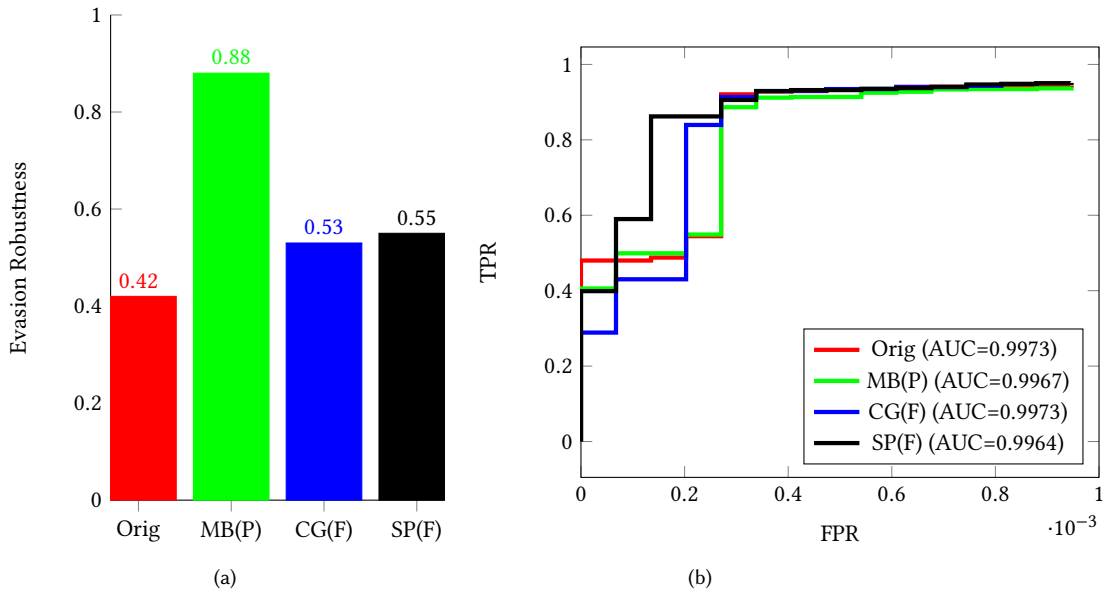


Fig. 5. Evasion robustness of the MB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin detection system.

classifiers achieve a similar evasion robustness. Lastly, the three retrained classifiers are basically as accurate as the

original Drebin classifier on clean data, with an AUC of more than 99% (Fig. 5b). Therefore, it is proven that a highly robust classifier for the MB attack (the MB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

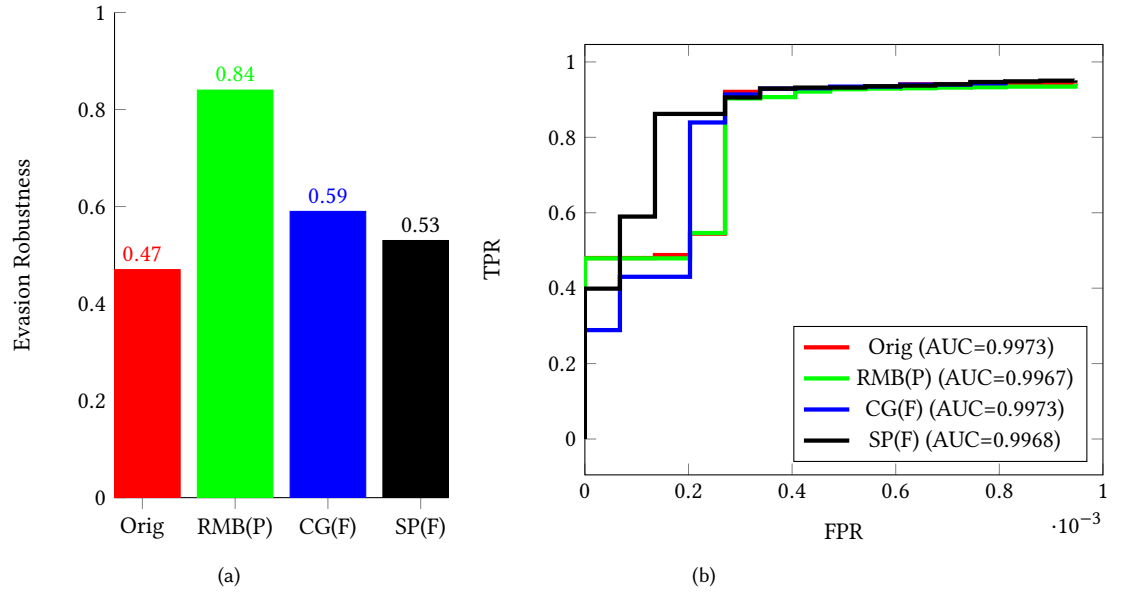


Fig. 6. Evasion robustness of the Random MB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin detection system.

6.1.4 Random MB Experiment. In the beginning, the Random MB attack was run on the original Drebin classifier without any retraining process, and the classifier resulted in an evasion robustness of 47%. Then, to create the baseline, Drebin was iteratively retrained with the Random MB evasion attack. As can be seen in Fig. 6a, the RMB(P) retrained classifier achieved an evasion robustness of 84%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 59% and 53%, respectively. As in the MB experiment, the retraining processes' are distinguishable, with a wide gap of 25% in the evasion robustness, between the CG(F) and SP(F) on the one hand, and the RMB(P) on the other hand. This shows that a defense that relies on recognizing feature-space evasion attacks will not identify the Random MB attack. Also in this experiment, the CG(F) retrained classifier outperformed the SP(F) retrained classifier by 6%. Lastly, the three retrained classifiers are basically as accurate as the original Drebin classifier on clean data, with an AUC of more than 99% (Fig. 6b). Therefore, it has been proven that a highly robust classifier to the Random MB attack (the RMB(P) classifier) can be achieved without significant damage to its effectiveness on clean data.

6.1.5 Discussion. The results presented in this section show that problem-space evasion attacks that target the Smali code are more identifiable by the Drebin classifier and its retrained classifiers than attacks that target the manifest file. The results with reference to the Random SB and DroidChameleon experiments, show that the retrained classifiers on feature-space evasion attacks trailed the retrained classifier on the problem-space evasion attack by 7%-11%. However, the experiments of the MB and Random MB attacks proved that problem-space evasion attacks that target permission requests are less predictable by the retrained classifiers on feature-space evasion attacks. In these experiments, the

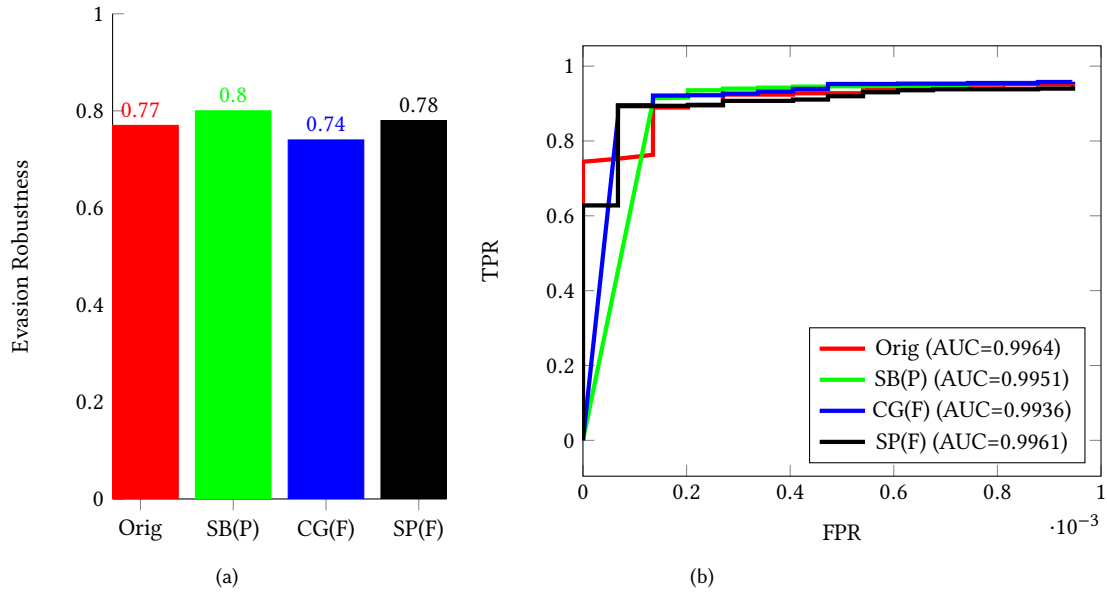


Fig. 7. Evasion robustness of the Random SB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin-DNN detection system.

CG(F) and SP(F) classifiers trailed the MB(P) and RMB(P) classifiers by more than 30%. This also supports the results of previous work [27, 31]. Nonetheless, these results ascertained that the same phenomena occur for classifiers that are iteratively retrained and that the difference between retraining processes of feature-space and problem-space evasion attacks is distinguishable. The next section explores the DNN version of Drebin.

6.2 Drebin-DNN

In this set of experiments, a DNN model, termed Drebin-DNN, was trained on the feature set of Drebin (as in [9]). Drebin-DNN consisted of two fully-connected hidden layers, where each comprised 160 neurons. The activation function was ReLU. An Adam optimizer was used for optimization, with 150 epochs, and a mini-batch of size 128. The learning rate was 0.001.

For each experiment, the basic DNN model was retrained with different problem-space evasion attacks: Random SB (Section 6.2.1), DroidChameleon (Section 6.2.2), MB (Section 6.2.3), and Random MB (Section 6.2.4). The CG(F) and SP(F) feature space attacks were considered in these experiments, using the same configurations as with Drebin. A short discussion (Section 6.2.5) is provided at the end of this section.

6.2.1 Random SB Experiment. First, the Random SB attack was run against the original Drebin-DNN classifier without any retraining process, and the classifier demonstrated an evasion robustness of 77%. Next, to establish the baseline classifier, Drebin-DNN was iteratively retrained with the Random SB evasion attack. As demonstrated in Fig. 7a, the SB(P) retrained classifier achieved an evasion robustness of 80%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 74% and 78%, respectively. These results show that defense that relies on recognizing feature-space evasion attacks will not correctly identify the Random SB attack. Furthermore, the retraining process

with the CG attack decreased the evasion robustness of the basic classifier by 3%. Furthermore, the three retrained classifiers were found to be basically as accurate as the original Drebin-DNN classifier on clean data, with an AUC of more than 99% (Fig. 7b). This experiment shows that sometimes the retraining process in problem-space attacks does not result in a classifier that is significantly more robust.

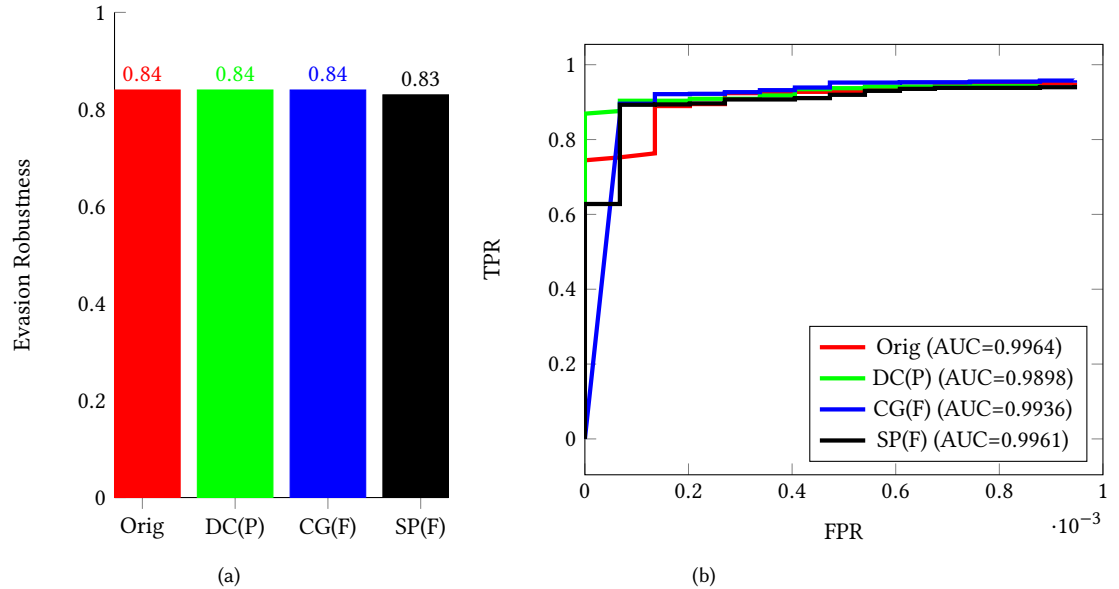


Fig. 8. Evasion robustness of the DroidChameleon attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin-DNN detection system.

6.2.2 DroidChameleon Experiment. First, the DroidChameleon attack was run on the original Drebin-DNN classifier without any retraining process, and the classifier resulted in an evasion robustness of 84%. Next, to establish the baseline classifier, Drebin-DNN was iteratively retrained with the DroidChameleon evasion attack. As depicted in Fig. 8a, the DC(P) retrained classifier achieved an evasion robustness of 84%, showing that the retraining process in this experiment did not produce any gain in evasion robustness. The CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 84% and 83%, respectively. This experiment also shows that sometimes the retraining process in problem-space attacks does not result in a classifier that is significantly more robust. In addition, the three retrained classifiers are basically as accurate as the original Drebin classifier on clean data, with an AUC of $\sim 99\%$ (Fig. 4b).

6.2.3 MB Experiment. Initially, the MB attack was run with the original Drebin-DNN classifier without any retraining process, and the classifier demonstrated an evasion robustness of 61%. Then, to create the baseline, Drebin-DNN was iteratively retrained with the MB evasion attack. As illustrated in Fig. 9a, the MB(P) retrained classifier achieved an evasion robustness of 88%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 50% and 62%, respectively. This time, the retraining processes were more distinguishable, with a wide gap of over 26% in the evasion robustness, between the CG(F) and SP(F), on the one hand, and the MB(P) on the other. Also, the retrained CG(F) classifier decreased the evasion robustness of the original classifier by 11%. This shows that a defense that relies on recognizing feature-space evasion attacks will not identify the MB attack. Also, the retrained classifiers

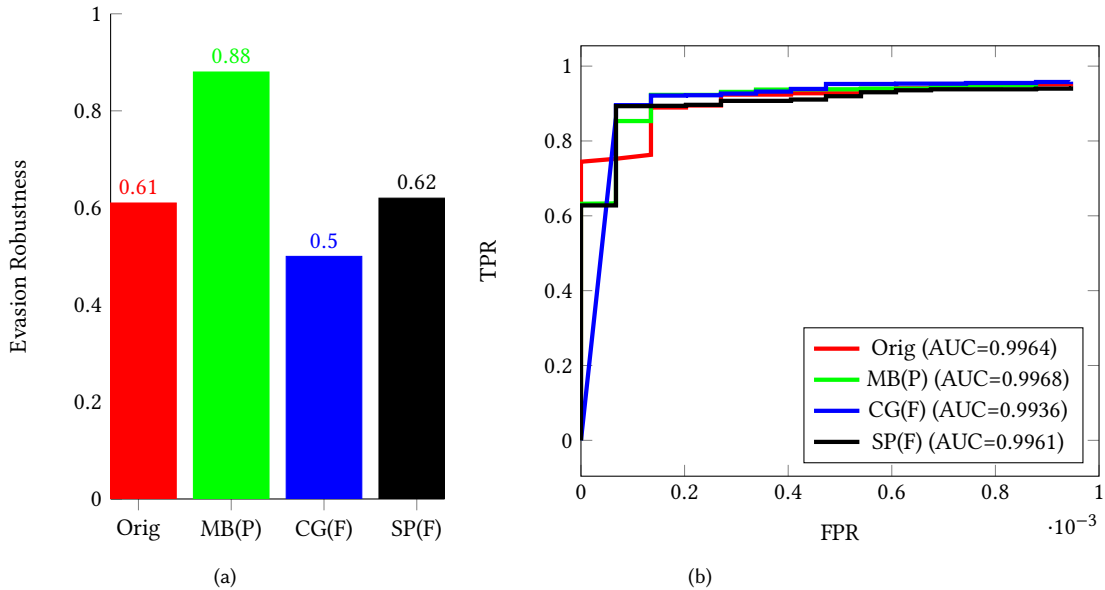


Fig. 9. Evasion robustness of the MB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin-DNN detection system.

in feature-space attacks performed slightly different than in the previous experiments. In this experiment, the SP(F) retrained classifier outperformed the CG(F) retrained classifier by 12%. Lastly, the three retrained classifiers are basically as accurate as the original Drebin-DNN classifier on clean data, with an AUC of more than 99% (Fig. 9b). Therefore, the results demonstrate a highly robust classifier for the MB attack (the MB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

6.2.4 Random MB Experiment. First, the Random MB attack was run on the original Drebin-DNN classifier without any retraining process, and the classifier resulted in an evasion robustness of 59%. Then, to create the baseline, Drebin was iteratively retrained with the Random MB evasion attack. As illustrated in Fig. 10a, the RMB(P) retrained classifier achieved an evasion robustness of 81%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 51% and 63%, respectively. As in the MB experiment, the retraining processes' were distinguishable, with a gap of 17% in evasion robustness between the CG(F) and SP(F) on the one hand, and the RMB(P) classifier on the other hand. Also, the retrained CG(F) classifier decreased the evasion robustness of the original classifier by 8%. These results show that a defense that relies on recognizing feature-space evasion attacks will not identify the Random MB attack. Also, the SP(F) retrained classifier outperformed the CG(F) retrained classifier by 12%. Lastly, the three retrained classifiers were basically as accurate as the original Drebin classifier on clean data, with an AUC of more than 99% (Fig. 10b). Therefore, the results prove that a highly robust classifier for the Random MB attack (the RMB(P) classifier) can be achieved without significant damage to its effectiveness on clean data.

6.2.5 Discussion. In conclusion, problem-space evasion attacks that target the Smali code are more identifiable by the Drebin-DNN classifier and its retrained classifiers, as depicted in the results of the Random SB attack experiment. Also, the DroidChameleon experiment demonstrated that retraining does not always add robustness. However, the

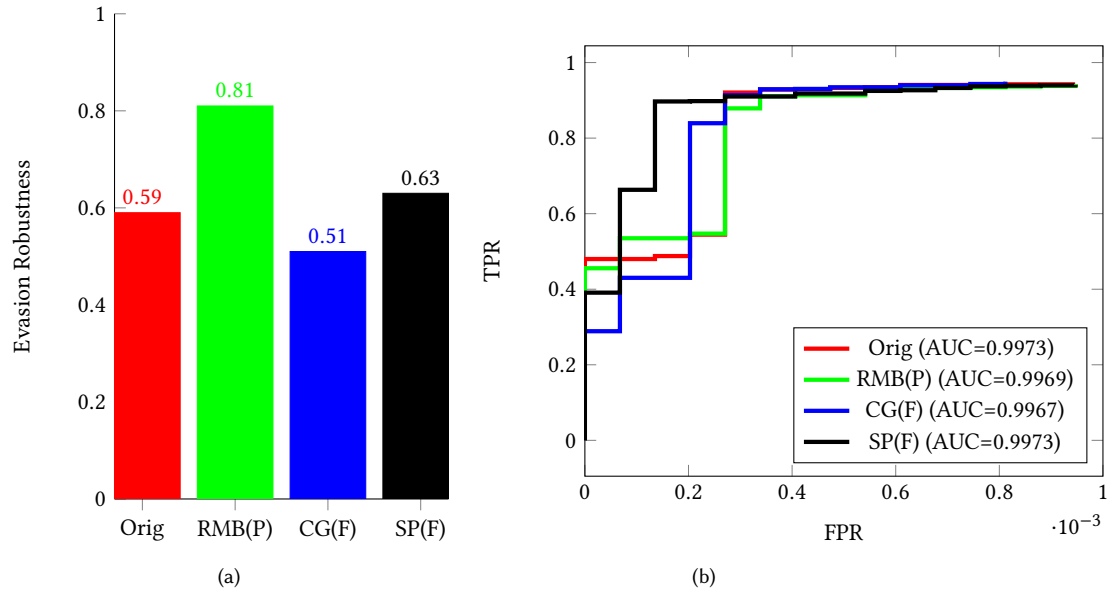


Fig. 10. Evasion robustness of the Random MB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the Drebin-DNN detection system.

experiments with the MB and Random MB attacks proved that problem-space evasion attacks that target permission requests are less predictable by the retrained classifiers in feature-space evasion attacks. In these experiments, the CG(F) and SP(F) classifiers trailed the MB(P) and RMB(P) classifiers by at least 12%. As with the Drebin experiments these results support previous work [27, 31]. However, these results added the understanding that the same phenomena occur for a DNN model that is iteratively retrained and that the difference between retraining processes of feature-space and problem-space evasion attacks is distinguishable as well.

Moreover, sometimes the feature-space evasion attack decreases the evasion robustness compared to the original classifier. For example, the retrained CG(F) classifier decreased the evasion robustness of the original classifier by 11% when tested with the MB attack, and by 8% when tested with the Random MB attack. This decrease may seem small. Nonetheless, every malicious sample that is not recognized as such damages the credibility of a malware detection system. Consequently, it is an important anecdote. Also, it is important to acknowledge that retraining with feature-space attacks does not always make the classifier more robust. This robustification process may create a bias toward unrealistic data and, as a result, affect the identification of real problem-space evasion attacks.

6.3 MaMaDroid

The third set of experiments utilizes the feature set of MaMaDroid [3], a well-known Android malware classifier that analyzes the control flow graph of the application. This set of experiments is different from previous experiments, as this set includes multiple models - 1NN, 3NN, and RF (as in the original MaMaDroid paper [3]). An additional DT model was examined as well, as it improves the overall evasion robustness of MaMaDroid, as proven by Berger et al. [6]. As the four models showed similar results, three out of the four models' results are presented in Appendix A. The results presented relate to the RF model, which was chosen by [3] as the most promising model. This RF model is termed

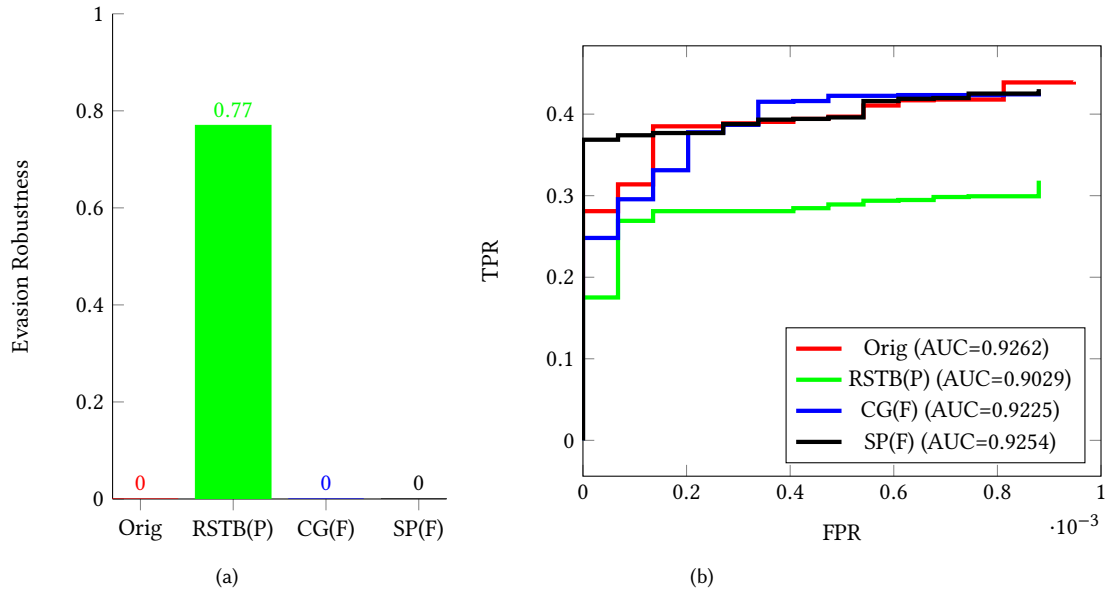


Fig. 11. Evasion robustness of the Random STB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the MaMaDroid detection system.

MaMaDroid until the end of this study. MaMaDroid was assessed using two experiments, where each one utilized a different variant of the STB attack [6]: Random (Section 6.3.1) and Black-Hole Statistical STB (Section 6.3.2) attacks. The CG and SP were the feature-space attacks that were examined. A slight modification was made to the CG, since the original CG targets binary features, and MaMaDroid analyzes real-value features. Therefore, instead of flipping from 0 to 1 and vice versa, a constant value was added in a cyclic way. The value chosen was 0.01. The results of the Random STB experiment are described in Section 6.3.1. This section also concludes with a short discussion 6.3.3.

6.3.1 Random STB Experiment. First, the Random STB attack was run on the original MaMaDroid classifier without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Random STB evasion attack. As demonstrated in Fig. 11a, the RSTB(P) retrained classifier achieved an evasion robustness of 77%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 0%. This shows that a defense that relies on recognizing feature-space evasion attacks will not identify the Random STB attack. Lastly, the three retrained classifiers were basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 90% (Fig. 11b). Therefore, a highly robust classifier to the Random STB attack (the RSTB(P) classifier) can be achieved without significant damage to its effectiveness on clean data.

6.3.2 Black-Hole Statistical STB Experiment. First, the Black-Hole Statistical STB attack was carried out on the original MaMaDroid classifier without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Black-Hole Statistical STB evasion attack. As depicted in Fig. 12a, the BSTB(P) retrained classifier achieved an evasion robustness of 96%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 0%. This shows that a defense that relies on recognizing

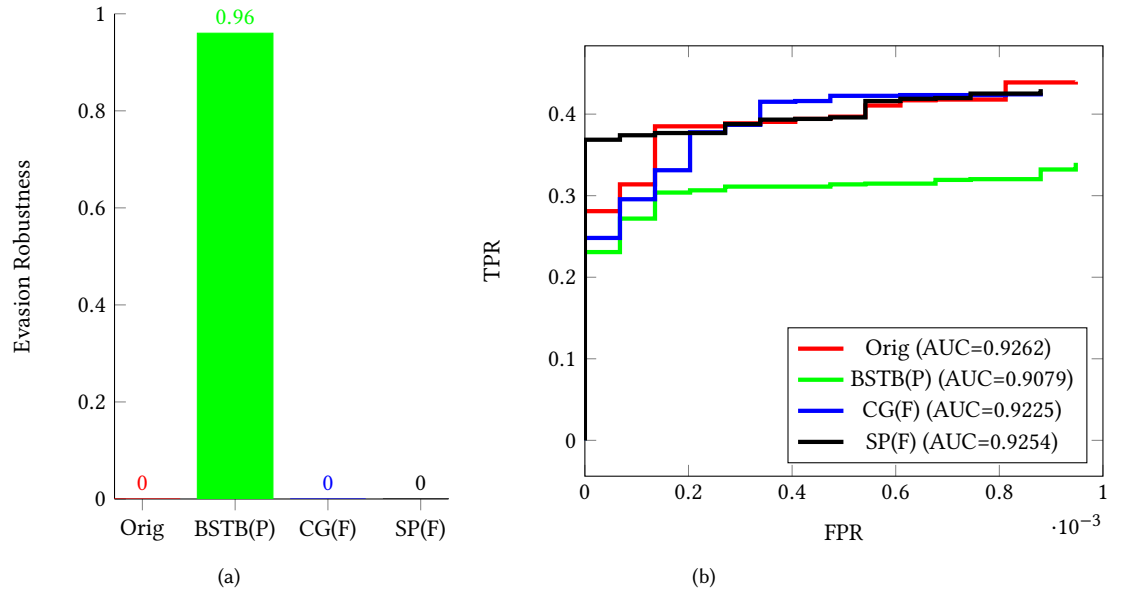


Fig. 12. Evasion robustness of the Black-Hole Statistical STB attack (a) and performance on clean data (b) of different retrained classifiers (and original classifier) for the MaMaDroid detection system.

feature-space evasion attacks will not identify the Black-Hole Statistical STB attack. Lastly, the three retrained classifiers were basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 90% (Fig. 12b). Therefore, a highly robust classifier for the Black-Hole Statistical STB attack (the BSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

6.3.3 Discussion. As was proven in both experiments, the MaMaDroid classifier is vulnerable to STB attacks. Thus, attacks that target the structure of a sample evade classification as malicious by detection machines that analyze the control flow graph. In addition to these findings, which were proven in [6], this was proven in the above experiments to be true for retrained classifiers as well. The retraining process on feature-space attacks does not add defense against this kind of attack. The examined feature-space attacks, CG and SP, randomly choose an index in the feature vector and change their values. The examined problem-space attacks, STB, move partial values from one feature to another. For example, the Black-Hole Statistical STB attack moves feature values to specific parts of the feature value that mainly has values of 0 or close to it. The exact amount depends on the specific application, its structure, and its complexity. Therefore, it cannot be mimicked by feature-space evasion attacks, specifically by CG or SP. It is more complex. As proven in the previous section, retraining classifiers on feature-space evasion attacks does not cover real-life evasion attacks.

7 PROBLEM-SPACE EVASION ATTACKS GENERALIZABILITY

Hitherto, retrained classifiers on feature-space attacks were tested to evaluate their robustness against problem-space evasion attacks. Retraining with feature-space attacks was proven in the last section to be ineffective against most of the problem-space evasion attacks. Therefore, the feature-space attacks are eliminated from this discussion. As each

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

AT \ DEF	Basic	RSB	DC	MB	RMB
RSB	65%	+25%	+21%	+15%	+29%
DC	80%	+4%	+10%	+9%	+11%
MB	42%	+33%	+9%	+46%	+45%
RMB	47%	+24%	0%	+42%	+41%

Table 1. Generalizability of problem-space evasion attacks against Drebin. The basic classifier is Drebin. The problem-space evasion attacks (AT) are Random SB (RSB), Droichemeleon (DC), MB, and Random MB (RMB). Each retrained classifier (DEF) is described by its delta from the basic classifier (Basic).

problem-space evasion attack differs, in means of the weak spots it targets, it is natural to wonder whether classifiers that are robust to one problem-space evasion attack remain robust to other problem-space evasion attacks. In this section, each retrained classifier on a problem-space attack was tested with the other problem-space evasion attacks against its basic classifier. For example, the retrained classifier for the DroidChameleon attack was tested with the Random SB, DroidChameleon, MB, and Random MB attacks. In other words, these experiments attempted to check whether or not problem-space evasion attacks are generalizable. In this section, the retrained classifiers are denoted only by their initials, as all of them were retrained on problem-space evasion attacks. The metric used was evasion robustness, as it is the most suitable way to measure the differences in the identification of evasion attacks. However, instead of using the actual evasion robustness of each attack, the delta from the basic classifier was used to draw the effects of each retraining process. Due to the differences between the basic classifiers, each basic classifier is described in a separate section, as before. First, Drebin’s results are described in Section 7.1. Then, Drebin-DNN’s results are described in Section 7.2. Finally, the results of MaMaDroid are shown in Section 7.3.

7.1 Drebin

This section describes the results of running the evasion attacks against Drebin with each of the retrained classifiers using problem-space evasion attacks. The idea behind these experiments was to assess the generalizability of the attacks. The results are described in Table 1, which show that in most cases, the retrained classifiers using a specific problem-space attack remained robust against other problem-space evasion attacks. Specifically, retrained classifiers in the Random SB and Random MB attacks were found to be the most robust. Because both of the evasion attacks have a random element, each iteration created a somewhat different manipulated version of the original sample. Therefore, the retraining process was shown to be the most efficient.

Additionally, the nature of similar problem-space evasion attacks was revealed. In other words, in each pair of evasion attacks that target the same part of the sample, one evasion attack was found to be a great candidate for retraining a classifier against the other attack. For example, the Random SB and Droichemeleon both target the Smali code. The SB classifier was found to be more robust to Droichemeleon than the basic classifier by 4%. On the other hand, the DC classifier was found to be more robust to the SB attack than the basic classifier by 21%. Four percent might seem negligent. However, because each manipulated sample that evades the classifier is a threat to the public, it still matters. In addition, a similar phenomenon occurs with the MB and Random MB as they share a common target - the manifest file. The MB classifier proves to be more robust than the basic classifier by 46% against the Random MB attack. The RMB classifier outperforms the basic classifier by 45% when tested with the MB attack. These two retrained classifiers of the four problem-space retrained classifiers performed the best.

AT \ DEF	Basic	RSB	DC	MB	RMB
RSB	76%	+6%	+6%	+10%	+10%
DC	85%	0%	0%	+2%	+2%
MB	58%	-7%	-7%	+19%	+19%
RMB	57%	-5%	-5%	+22%	+22%

Table 2. Generalizability of problem-space evasion attacks against Drebin-DNN. The basic classifier is Drebin-DNN. The problem-space evasion attacks (AT) inspected are Random SB (RSB), Droichemeleon (DC), MB, and Random MB (RMB). Each retrained classifier (DEF) is described by its delta from the basic classifier (Basic).

Another interesting result shows that not every retraining process on a problem-space attack creates a robust classifier against any other problem-space attack. As detailed in Table 1, the DC classifier did not increase evasion robustness in comparison to the basic classifier when the tested evasion attack was RMB.

To summarize, the results of the retrained classifiers on problem-space evasion attacks show that each one of these evasion attacks was found to be generalizable. However, the results when using the MB or the Random MB evasion attacks to retrain a classifier compared to those using the other two evasion attacks were better. The findings also show that some problem-space evasion attacks do not increase the evasion robustness of the classifier against other problem-space evasion attacks. This is understandable because retraining a classifier with new data sometimes creates a tendency in the classification process toward these new data. However, when retraining and evaluating with similar problem-space evasion attacks, this phenomenon does not appear.

7.2 Drebin-DNN

This section describes the results of running the evasion attacks against Drebin-DNN, with each one of the retrained classifiers using problem-space evasion attacks. The results, described in Table 2, demonstrate that in most cases, the retrained classifiers using a specific problem-space attack stayed robust against other problem-space evasion attacks. Specifically, the retrained classifiers on the MB and Random MB attacks were found to be the most robust, with a delta of at least 2% in evasion robustness (compared to the basic classifier) when tested on other attacks.

Additionally, the nature of the similar problem-space evasion attacks was once again confirmed. In each pair of evasion attacks that target the same part of the sample, one evasion attack was found to be a reasonable candidate for retraining a classifier against the other attack. For example, the Random SB and Droichemeleon target both the Smali code. The DC classifier was found to be more robust to the SB attack than the basic classifier by 6%. The SB classifier was found to be as robust as the basic classifier to the Droichemeleon attack. In addition, a similar phenomenon was revealed for the MB and Random MB because they share a common target, i.e., the manifest file. The MB classifier was shown to be more robust than the basic classifier by 22% against the Random MB attack. The RMB classifier outperformed the basic classifier by 19% when tested on the MB attack.

Another interesting set of results shows that not every retrained classifier on a problem-space attack correctly classifies each type of problem-space evasion attack. The results presented in Table 2 show that the DC classifier and the SB classifier were evaded by an additional 7% in comparison to the basic classifier when the evasion attack was MB and an additional 5% when the evasion attack was Random MB.

To summarize this section, the results of the retrained classifiers on problem-space evasion attacks showed that using the MB or the Random MB evasion attacks to retrain a classifier is better than using the other two evasion attacks.

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

AT \ DEF	Basic	RSTB	BSTB
RSTB	0%	+79%	+79%
BSTB	0%	+96%	+96%

Table 3. Generalizability of problem-space evasion attacks against MaMaDroid. The basic classifier is MaMaDroid. The problem-space evasion attacks (AT) inspected are Random STB (RSTB) and Black-Hole Statistical STB (BSTB). Each retrained classifier (DEF) is described by its delta from the basic classifier (Basic).

Each of these evasion attacks was found to be generalizable. Some problem-space evasion attacks damaged the evasion robustness of the classifier against other problem-space evasion attacks. This understandable because retraining a classifier with new data sometimes creates a tendency in the classification process toward these new data. However, when retraining and evaluating with similar problem-space evasion attacks, this phenomenon does not occur.

Two insights merged from the generalizability assessments of Drebin and Drebin-DNN. First, the attacks that are generalizable in both detection systems is the MB and RMB attack, which targets the permission requests, a component that is of a high value for both Drebin and Drebin-DNN (as was proven in [27]). On the other hand, a less efficient retraining process was achieved when retraining on the DroidChameleon attack. This attack achieved the lowest evasion robustness deltas among the other retrained classifiers.

7.3 MaMaDroid

This section describes the results of running the evasion attacks against MaMaDroid, with each one of the retrained classifiers using the STB evasion attacks. The idea behind these experiments was to explore the generalizability of these evasion attacks. As in Section 6.3, the results of the RF model are described in this section. The results of the other three models are provided in the Appendix B. The results of the generalizability of the evasion attacks against MaMaDroid, presented in Table 3, demonstrate that the retrained classifiers had the same results for each one of the evasion attacks. In other words, the evasion robustness gained from retraining with each of the STB attacks resulted in the same evasion robustness against each one of the other evasion attacks. These attacks are also generalizable. The evasion robustness of the Random STB attack was found to be 79% more than the basic classifier. The retrained classifiers gained an evasion robustness of 96% more than the basic classifier against the Black-Hole STB attack.

The STB attacks share a common approach, as stated in the paper in which they originated [6]. Therefore, it is not surprising that the classifiers that were retrained on them would achieve identical results when confronting with the same attack. Still, it is important to address this similarity after the retraining process. As the results show, for any specific retrained classifier, the difference in evasion robustness remained the same, namely - the random STB attack was more evasive than the Black-Hole Statistical STB attack.

To summarize, the results of the comparison between the retrained classifiers on problem-space evasion attacks showed that similar problem-space evasion attacks create similar effective retrained classifiers (i.e., the cases of the STB attacks). Also, a retrained classifier on one type of problem-space evasion attack can be sufficiently robust against another type of evasion attack (e.g., the MB classifier and the DC attack). However, it is not guaranteed, as exemplified by the DC classifier (of the Drebin-DNN system) which decreased the initial evasion robustness against the MB and RMB attacks. Also, at times, retraining on one problem-space evasion attack will degrade the evasion robustness against other problem-space evasion attacks. This is understandable, as retraining on one attack may create a tendency in the classification process toward this attack. Consequently, the retrained classifier is less efficient against other attacks.

8 RELATED WORK

This section describes some of the related work on evasion attacks and defenses in ML-based malware detection; A general insightful survey can be found in Vorobeychik and Kantarcioglu [52].

8.1 Evasion attacks and Adversarial Examples

One of the well-celebrated problem-space evasion attacks on ML was suggested by Fogla et al. [80, 81]. This attack targeted an anomaly-based IDS. More recent work used several methods such as obfuscation, reflection, and adding noise to the original sample. Demontis et al. [7] used the obfuscation of suspicious constant values in the APK such as package names and API calls. DaDidroid [82] explored a similar approach to that of Demontis et al. [7] using obfuscation as well. Rastogi et al. [83] presented an evasion attack, which combined the approach of Demontis et al. [7] with the addition of the reflection approach (reflection loads additional code at runtime).

Another form of problem-space evasion attacks adds noises to the application to achieve miss-classification. A stub function, one of the famous types of noise, is a non-operational function, that does not change any functionality of the app. However, it slightly modifies the original order of API calls in the application. A well-known example of a stub function addition is Android HIV [35], where the authors added suspicious functions that are not invoked to evade the Drebin classifier, and stub functions to evade the MaMaDroid classifier. A more recent example of adding noise to the app is the work of Pierazzi et al. [43], where the authors implanted functions from benign apps in malicious apps to be miss-classified by the Drebin and Sec-SVM [7] classifiers. Rosenberg et al. [84] generated evasion attacks using three approaches, namely: the addition of non-operational functions, obfuscation of strings, and encoding of API calls. Cara et al. [85] added non-invoked classes to the end of functions to achieve miss-classification.

In addition to problem-space evasion attacks, a vast amount of feature-space was explored [38, 45, 55–57, 57, 86–94]. The most common domain of feature-space evasion attacks is image classification and, more specifically, the art of evading deep neural networks [88, 95–99]. Several approaches attempted to generate a new version of adversarial examples in the real world. An example of one is the use of stickers on a stop sign to achieve misclassification. These attacks are considered analogous to problem-space evasion attacks [97, 100].

8.2 Robustification Methods Against Evasion Attacks

The first approach to robustification of detection systems was devised by Dalvi et al. [89]. The authors formulated robust classification as a game between the classifier and the attacker and produced an optimal classifier strategy. Other approaches defined robust classification as a Min-Max loss problem. In this approach, the adversary’s goal is the maximization of the loss of the defender by means of small modifications of the feature values [51, 53, 101–103]. Additional approaches to design robust classifiers are the use of a non-zero-sum game [46, 47, 55–57], or a Stackelberg game, where the learner is the leader, while the attacker is the follower [47, 56, 57, 104]. Finally, iterative retraining defense mechanisms have been proposed, both for general evasion attacks [55, 57], and specifically for deep learning detection systems of computer vision [88, 95, 103]. This list of the robustification efforts shares one important characteristic: the underlining attacks that are used for the robustification of basic classifiers are feature-space attacks, which, as this paper proves, hardly constitute a proxy for practical problem-space evasion attacks.

9 DISCUSSION AND CONCLUSION

This paper presents the examination of the robustification efforts of ML-based malware detection systems, using problem-space and feature-space evasion attacks. Several observations can be drawn from this work. First, a defense that is based on learning feature-space evasion attacks apparently fails to achieve high robustness against problem-space evasion attacks. This observation raises certain doubts about the commonplace focus on such attacks as an upgrade to ML defense and suggests that the practical usefulness of such techniques cannot be overlooked. However, there is a difference between the types of problem evasion attacks. The attacks that change the content of the Smali code itself are more predictable and usually do not greatly change feature values. Consequently, retrained feature-space classifiers can be used to detect such attacks. In comparison, attacks that target the manifest file change feature values that are less commonly changed. Therefore, the retrained feature-space classifiers fail to identify this kind of attack. For example, the DroidChameleon attack changes constant string values that might not take part in the feature extraction of Drebin. On the contrary, the MB attacks change permission requests, which are correlated to features that Drebin analyzes and are highly weighted by it. Therefore, the MB attacks are harder to identify.

Second, the complexity of problem-space evasion attacks is seldomly hard to map to the feature space. Some of the changes that occur during a problem-space evasion attack, as mentioned with regards to DroidChameleon, are not translated to a change in the feature vector extracted by the ML. Consequently, this evasion attack has high evasion robustness in reference to each of the retrained classifiers, including the retrained classifiers in feature-space attacks. However, some changes, such as the case of STB evasion attacks, move feature values inside the feature vector, that cannot be forecast and mathematically analyzed in a general way. The art of devising these attacks is not very complicated to implement. Also, these attacks do not alter the functionality of the sample. But determining which part of the feature vector moves to other parts of it is very difficult, and can take infinite time for mathematical-based approaches like retraining a classifier on feature-space attacks.

Third, some problem-space retraining can increase robustness against other problem-space evasion attacks. Two problem-space evasion attacks that target the same part of the APK sample seem to have common elements. As was proven, it is true that problem-space retraining with an evasion attack that targets the Smali code was efficient against another evasion attack that targets the Smali code. The same idea is true for attacks that target the manifest file. However, not every classifier retrained with one type of problem-space evasion attack, is robust against another type of problem-space evasion attack. These phenomena show that the robustness gained from retraining may harm the basic detection of malicious entities, as it creates a bias towards a specific kind of evasion attack, which does not reflect reality. In the daily apps markets, various kinds of malicious applications can be found, in which some encapsulate different kinds of evasion attacks. Therefore, this kind of bias harms the credibility of retraining on problem-space evasion attacks.

The gaps between retraining processes of different types of evasion attacks were drawn. However, these findings are just the tip of the iceberg. Retraining with unrealistic feature-space evasion attacks, and counting on this process as a robustness technique is not realistic.

This work used a particular class of feature-space attacks, with the l_2 norm to measure the cost of feature modifications, and stochastic local search to compute evasion attempts. Future evasion attack algorithms may be developed which might be better than the evasion attacks that were used in this study. Testing new feature-space attacks is left for future work.

Other directions will also be explored in future work. For example, the problem-space evasion attacks that were tested targeted a specific part of the application. More complex evasion attacks that target both the Smali code files and the manifest file may create a greater challenge for Android malware detection systems. Their effects should be evaluated, and also the effect of the retraining process of basic classifiers. In addition, two types of feature vectors were tested: Drebin’s and MaMaDroid’s. Each feature vector depicts other aspects of the sample. Different types of feature vectors, such as features extracted from the dynamic analysis of the application, are missing from the analysis provided in this study. These feature vectors are a base for the future extension of this work.

ACKNOWLEDGMENTS

This work was supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber Directorate of the Prime Minister’s Office.

REFERENCES

- [1] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *International Conference on Security and Privacy in Communication Systems*, pages 86–103. Springer, 2013.
- [2] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.
- [3] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *Annual Symposium on Network and Distributed System Security*, 2017.
- [4] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report*, 14(1):16–29, 2009.
- [5] Jing Lin, Laurent L Njilla, and Kaiqi Xiong. Secure machine learning against adversarial samples at test time. *EURASIP Journal on Information Security*, 2022(1):1–15, 2022.
- [6] Harel Berger, Chen Hajaj, Enrico Mariconti, and Amit Dvir. Mamadroid2.0 – the holes of control flow graphs, 2022.
- [7] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Iginio Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 2017.
- [8] Chenglin Li, Keith Mills, Di Niu, Rui Zhu, Hongwen Zhang, and Husam Kinawi. Android malware detection based on factorization machine. *IEEE Access*, 7:184008–184019, 2019.
- [9] Deqiang Li and Qianmu Li. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. *IEEE Transactions on Information Forensics and Security*, 15:3886–3900, 2020.
- [10] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 356–367. ACM, 2016.
- [11] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security*, 22(2):14, 2019.
- [12] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. Cdgdroid: Android malware detection based on deep learning using cfg and dfg. In *International Conference on Formal Engineering Methods*, pages 177–193. Springer, 2018.
- [13] XU Zhiwu, Kerong Ren, and Fu Song. Android malware family classification and characterization using cfg and dfg. In *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 49–56. IEEE, 2019.
- [14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [15] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer, 2017.
- [16] Aditya Kuppa, Slawomir Grzonkowski, Muhammad Rizwan Asghar, and Nhien-An Le-Khac. Black box attacks on deep anomaly detectors. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, page 21. ACM, 2019.
- [17] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.
- [18] Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, and Mark Schloesser. Facts & analyses on the threat scenario: The av-test security report 2019/2020, 2020. <https://www.av-test.org/en/news/facts-analyses-on-the-threat-scenario-the-av-test-security-report-2019-2020/>.
- [19] Statcounter GlobalStats. Mobile operating system market share worldwide ,feb 2021 - feb 2022, 2022. <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- [20] Haipeng Cai and John Jenkins. Towards sustainable android malware detection. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, pages 350–351. ACM, 2018.

- [21] Sen Chen, Minhui Xue, Zhushou Tang, Lihua Xu, and Haojin Zhu. Stormdroid: A streaming-based machine learning system for detecting android malware. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 377–388. ACM, 2016.
- [22] Gianluca Dini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra. Madam: a multi-level anomaly detector for android malware. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 240–253. Springer, 2012.
- [23] Ngoc Anh Huynh, Wee Keong Ng, and Kanishka Ariyapala. A new adaptive learning algorithm and its application to online malware detection. In *International Conference on Discovery Science*, pages 18–32. Springer, 2017.
- [24] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. “andromaly”: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.
- [25] Asaf Shabtai, Lena Tenenboim-Chekina, Dudu Mimran, Lior Rokach, Bracha Shapira, and Yuval Elovici. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43:1–18, 2014.
- [26] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia Joint Conference on Information Security*, pages 62–69. IEEE, 2012.
- [27] Harel Berger, Chen Hajaj, Enrico Mariconti, and Amit Dvir. Crystal ball: From innovative attacks to attack effectiveness classifier. *IEEE Access*, 2021.
- [28] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *2015 IEEE 39th annual computer software and applications conference*, volume 2, pages 422–433. IEEE, 2015.
- [29] Wei Yuan, Yuan Jiang, Heng Li, and Minghui Cai. A lightweight on-device detection method for android malware. *IEEE transactions on systems, man, and cybernetics: systems*, 2019.
- [30] Kazuya Nomura, Daiki Chiba, Mitsuaki Akiyama, and Masato Uchida. Auto-creation of android malware family tree. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.
- [31] Harel Berger, Chen Hajaj, and Amit Dvir. Evasion is not enough: A case study of android malware. In *International Symposium on Cyber Security Cryptography and Machine Learning*, pages 167–174. Springer, 2020.
- [32] N. Šrđić and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *IEEE Symposium on Security and Privacy*, pages 197–211, 2014.
- [33] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers: A case study on PDF malware classifiers. In *Network and Distributed System Security Symposium*, 2016.
- [34] Liang Tong, Bo Li, Chen Hajaj, Chaowei Xiao, Ning Zhang, and Yevgeniy Vorobeychik. Improving robustness of {ML} classifiers against realizable evasion attacks using conserved features. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 285–302, 2019.
- [35] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 2019.
- [36] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1332–1349. IEEE, 2020.
- [37] Battista Biggio, Iginio Corona, Davide Maiorca, Blaine Nelson, Nedim Šrđić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.
- [38] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, 2014.
- [39] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, pages 43–58, 2011.
- [40] Davide Maiorca, Iginio Corona, and Giorgio Giacinto. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 119–130. ACM, 2013.
- [41] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. Towards robust detection of adversarial infection vectors: Lessons learned in pdf malware. *arXiv preprint arXiv:1811.00830*, 2018.
- [42] Erwin Quiring, Alwin Maier, and Konrad Rieck. Misleading authorship attribution of source code using adversarial learning. In *28th USENIX Security Symposium*, pages 479–496, 2019.
- [43] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy*, pages 1308–1325. IEEE Computer Society, 2020.
- [44] Hamid Bostani and Veelasha Moonsamy. Evadroid: A practical evasion attack on machine learning for black-box android malware detection. *arXiv preprint arXiv:2110.03301*, 2021.
- [45] Battista Biggio, Iginio Corona, Davide Maiorca, Blaine Nelson, Nedim Šrđić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 387–402, 2013.
- [46] M. Brückner and T. Scheffer. Static prediction games for adversarial learning problems. *Journal of Machine Learning Research*, (Sep):2617–2654, 2012.
- [47] M. Brückner and T. Scheffer. Stackelberg games for adversarial prediction problems. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 547–555, 2011.
- [48] H. Xu, C. Caramanis, and S. Mannor. Robustness and regularization of support vector machines. *Journal of Machine Learning Research*, 10:1485–1510, 2009.

- [49] Nicolas Papernot, Patrick McDaniel, Xi Wu, and Somesh Jha. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy*, 2016.
- [50] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. In *IEEE European Symposium on Security and Privacy*, 2018.
- [51] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.
- [52] Yevgeniy Vorobeychik and Murat Kantarcioglu. *Adversarial Machine Learning*. Morgan and Claypool, 2018.
- [53] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, 2018.
- [54] Liang Tong, Bo Li, Chen Hajaj, Chaowei Xiao, Ning Zhang, and Yevgeniy Vorobeychik. Improving robustness of ℓ_1 classifiers against realizable evasion attacks using conserved features. In *28th USENIX Security Symposium*, pages 285–302, 2019.
- [55] Bo Li and Yevgeniy Vorobeychik. Evasion-robust classification on binary domains. *ACM Transactions on Knowledge Discovery from Data*, 2018.
- [56] Bo Li and Yevgeniy Vorobeychik. Feature cross-substitution in adversarial classification. In *Neural Information Processing Systems*, pages 2087–2095, 2014.
- [57] Alex Kantchelian, J. D. Tygar, and Anthony D. Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pages 2387–2396, 2016.
- [58] Android Developers. Understand the apk structure, 2022. <https://developer.android.com/topic/performance/reduce-apk-size#apk-structure>.
- [59] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. *arXiv preprint arXiv:1911.02142*, 2019.
- [60] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. ℓ_1 : Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium*, pages 729–746, 2019.
- [61] Nadia Daoudi, Kevin Allix, Tegawendé François D Assise Bissyande, and Jacques Klein. A deep dive inside drebin: An explorative analysis beyond android malware detection scores. *ACM Transactions on Privacy and Security*.
- [62] Jinsung Kim, Younghoon Ban, Eunbyeol Ko, Haehyun Cho, and Jeong Hyun Yi. Mapas: a practical deep learning-based android malware detection system. *International Journal of Information Security*, pages 1–14, 2022.
- [63] Yonghao Gu and Liangxun Li. Graph-evolvedroid: Mitigate model degradation in the scenario of android ecosystem evolution. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3588–3591, 2021.
- [64] Ahmed Abusnaina, Mohammed Abuhamad, Hisham Alasmay, Afsah Anwar, Rhongho Jang, Saeed Salem, Daehun Nyang, and David Mohaisen. Dl-flmc: Deep learning-based fine-grained hierarchical learning approach for robust malware classification. *IEEE Transactions on Dependable and Secure Computing*, 2021.
- [65] Yérom-David Bromberg and Louison Gitzinger. Droidautoml: A microservice architecture to automate the evaluation of android machine learning detection systems. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 148–165. Springer, 2020.
- [66] Harel Berger, Amit Dvir, and Chen Hajaj. Framework implementation. Github, 2022. <https://github.com/ArielCyber/framework2>.
- [67] Guangquan Xu, GuoHua Xin, Litao Jiao, Jian Liu, Shaoying Liu, Meiqi Feng, and Xi Zheng. Ofei: A semi-black-box android adversarial sample attack framework against dlaas. *arXiv preprint arXiv:2105.11593*, 2021.
- [68] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. A framework for enhancing deep neural networks against adversarial malware. *IEEE Transactions on Network Science and Engineering*, 8(1):736–750, 2021.
- [69] Raadhesh Kannan, Chin Ji Jian, and XiaoNing Guo. Adversarial evasion noise attacks against tensorflow object detection api. In *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–4. IEEE, 2020.
- [70] Holger H. Hoos and Thomas Stutzle. *Stochastic Local Search : Foundations & Applications*. Morgan Kaufmann, 2004.
- [71] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories*, pages 468–471. IEEE, 2016.
- [72] Google. GooglePlay app market. GooglePlay website, 2008. <https://play.google.com/store/apps/>.
- [73] Chenyue Wang, Linlin Zhang, Kai Zhao, Xuhui Ding, and Xusheng Wang. Advandmal: Adversarial training for android malware detection and family classification. *Symmetry*, 13(6):1081, 2021.
- [74] Raphael Labaca-Castro, Luis Muñoz-González, Feargus Pendlebury, Gabi Dreó Rodosek, Fabio Pierazzi, and Lorenzo Cavallaro. Universal adversarial perturbations for malware. *arXiv preprint arXiv:2102.06747*, 2021.
- [75] Satheesh Kumar Sasidharan and Ciza Thomas. Prodroid—an android malware detection framework based on profile hidden markov model. *Pervasive and Mobile Computing*, 72:101336, 2021.
- [76] Shun Tobiyama, Yukiko Yamaguchi, Hajime Shimada, Tomonori Ikuse, and Takeshi Yagi. Malware detection with deep neural network using process behavior. In *2016 IEEE 40th annual computer software and applications conference (COMPSAC)*, volume 2, pages 577–582. IEEE, 2016.
- [77] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th international conference on malicious and unwanted software (MALWARE)*, pages 11–20. IEEE, 2015.
- [78] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. Androdialysis: Analysis of android intent effectiveness in malware detection. *computers & security*, 65:121–134, 2017.

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

27

- [79] Marco Melis, Ambra Demontis, Maura Pintor, Angelo Sotgiu, and Battista Biggio. secml: A python library for secure and explainable machine learning. *arXiv preprint arXiv:1912.10013*, 2019.
- [80] Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *USENIX Security Symposium*, 2006.
- [81] Prahlad Fogla and Wenke Lee. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *ACM Conference on Computer and Communications Security*, pages 59–68, 2006.
- [82] Muhammad Ikram, Pierrick Beaume, and Mohamed Ali Kaafar. Dadidroid: An obfuscation resilient tool for detecting android malware via weighted directed call graph modelling. *arXiv preprint arXiv:1905.09136*, 2019.
- [83] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. Droidchameleon: evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 329–334. ACM, 2013.
- [84] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 490–510. Springer, 2018.
- [85] Fabrizio Cara, Michele Scalas, Giorgio Giacinto, and Davide Maiorca. On the feasibility of adversarial sample creation using the android system api. *Information*, 11(9):433, 2020.
- [86] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, pages 274–283, 2018.
- [87] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57, 2017.
- [88] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.
- [89] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 99–108, 2004.
- [90] Daniel Lowd and Christopher Meek. Adversarial learning. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 641–647, 2005.
- [91] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ACM Asia Conference Computer and Communications Security*, pages 16–25, 2006.
- [92] F. Zhang, P.P.K. Chan, B Biggio, D.S. Yeung, and F. Roli. Adversarial feature selection against evasion attacks. *IEEE Transactions on Cybernetics*, 2015.
- [93] Blaine Nelson, Benjamin I.P. Rubinstein, Ling Huang, Anthony D. Joseph, Steven J. Lee, Satish Rao, and J.D. Tygar. Query strategies for evading convex-inducing classifiers. *Journal of Machine Learning Research*, pages 1293–1332, 2012.
- [94] Yevgeniy Vorobeychik and Bo Li. Optimal randomized classification in adversarial settings. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 485–492, 2014.
- [95] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. In *International Conference on Learning Representations*, 2016.
- [96] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016. arxiv preprint.
- [97] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016.
- [98] Qiuling Xu, Guanhong Tao, Siyuan Cheng, and Xiangyu Zhang. Towards feature space adversarial attack by style perturbation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10523–10531, 2021.
- [99] Siyuan Cheng, Yingqi Liu, Shiqing Ma, and Xiangyu Zhang. Deep feature space trojan attack of neural networks by controlled detoxification. *arXiv preprint arXiv:2012.11212*, 2020.
- [100] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Computer Vision and Pattern Recognition*, 2018.
- [101] Choon Hai Teo, Amir Globerson, Sam Roweis, and Alexander J. Smola. Convex learning with invariances. In *Neural Information Processing Systems*, 2007.
- [102] Yan Zhou, Murat Kantarcioglu, Bhavani M. Thuraisingham, and Bowei Xi. Adversarial support vector machine learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1059–1067, 2012.
- [103] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.
- [104] Paolo Russu, Ambra Demontis, Battista Biggio, Giorgio Fumera, and Fabio Roli. Secure kernel machines against evasion attacks. In *ACM Workshop on Artificial Intelligence and Security*, pages 59–69, 2016.

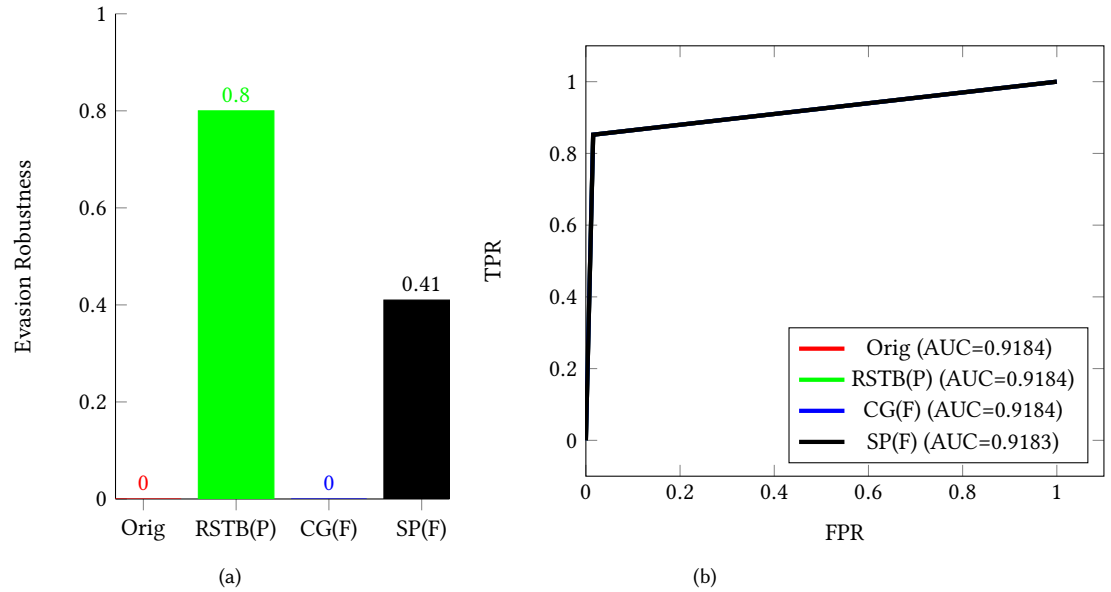


Fig. 13. Evasion robustness of the Random STB attack (a) and performance on clean data (b) for the MaMaDroid detection system. The underlining model is 1NN.

APPENDIX

A MAMADROID MODELS - FEATURE SPACE EVALUATION

This appendix provides the results of the evaluation of retraining based on feature-space attacks of the three models that were inspected in MaMaDroid, other than RF: 1NN, 3NN, DT. Due to the fact that some of the models had probabilities of 100% and 0% between the classes, fewer points in the roc evaluation, the roc curves were enlarged to the default range of 0-1. The results of the 1NN model are presented in Section A.1. The results of the 3NN model are discussed in Section A.2. The results of the DT model are presented in Section A.3.

A.1 1NN

This section describes the evaluation of the retrained classifiers of the 1NN model. First, the Random STB experiment is presented in Section A.1.1, and then, the Black-Hole Statistical STB experiment is described in Section A.1.2.

A.1.1 Random STB Experiment. First, the Random STB attack was run on the original MaMaDroid classifier without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Random STB attack. As depicted in Fig. 13a, the RSTB(P) retrained classifier achieved an evasion robustness of 80%. In comparison, the CG(F) retrained classifier resulted in an evasion robustness of 0%, and the SP(F) retrained classifier obtained evasion robustness of 41%. The retraining processes are fully distinguishable, with a wide gap of 39% in evasion robustness between the RSTB(P) and the CG(F) and SP(F) classifiers. This shows that defense that relies on recognizing feature-space evasion attacks will not fully identify the Random STB attack. Lastly, the three retrained classifiers were basically as accurate as the original MaMaDroid classifier on clean data,

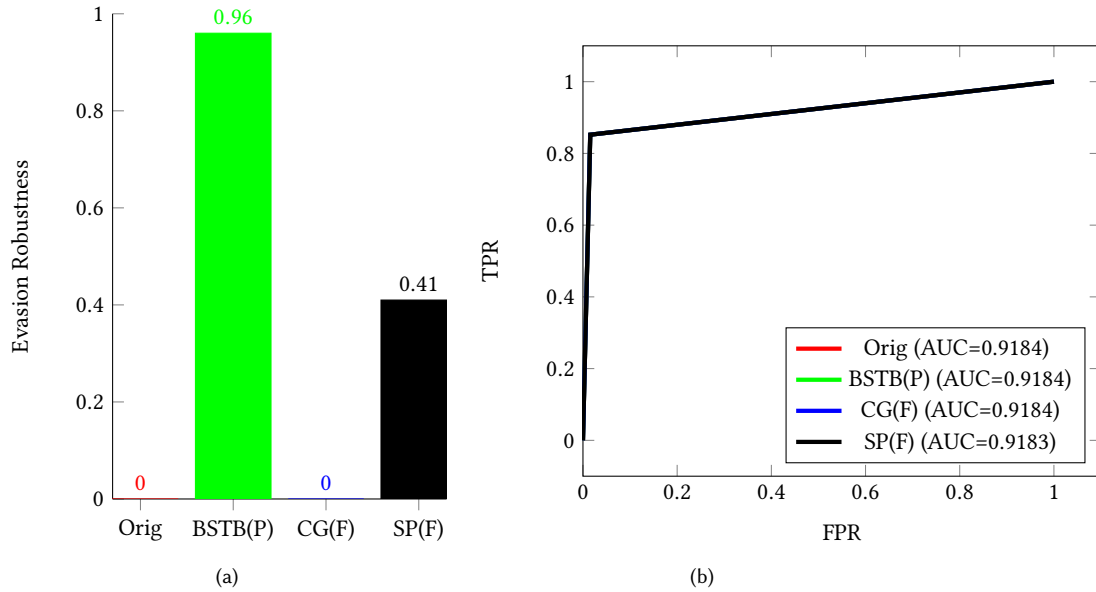


Fig. 14. Evasion robustness of the Black-Hole Statistical STB attack (a) and performance on clean data (b) for the MaMaDroid detection system. The underlining model is 1NN.

with an AUC of more than 91% (Fig. 13b). Therefore, a highly robust classifier for the Random STB attack (the RSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

A.1.2 Black-Hole Statistical STB Experiment. At first, the Black-Hole statistical STB attack was run on the original MaMaDroid classifier Without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Black-Hole Statistical STB attack. As illustrated in Fig. 14a, the BSTB(P) retrained classifier achieved an evasion robustness of 96%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 0% and 41%, respectively. The retraining processes are fully distinguishable, with a wide gap of 55% in the evasion robustness. This shows that a defense that relies on recognizing feature-space evasion attacks will not identify the Black-Hole Statistical STB attack. Lastly, the three retrained classifiers are basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 91% (Fig. 14b). Therefore, a highly robust classifier for the Black-Hole Statistical STB attack (the BSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

A.2 3NN

This section describes the evaluation of the retrained classifiers of the 3NN model. First, the Random STB experiment is detailed in Section A.2.1. Then, the Black-Hole Statistical STB experiment is described in Section A.2.2.

A.2.1 Random STB Experiment. First, the Random STB attack was run on the original MaMaDroid classifier without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Random STB attack. As illustrated in Fig. 15a, the RSTB(P) retrained classifier achieved an evasion robustness of 75%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness

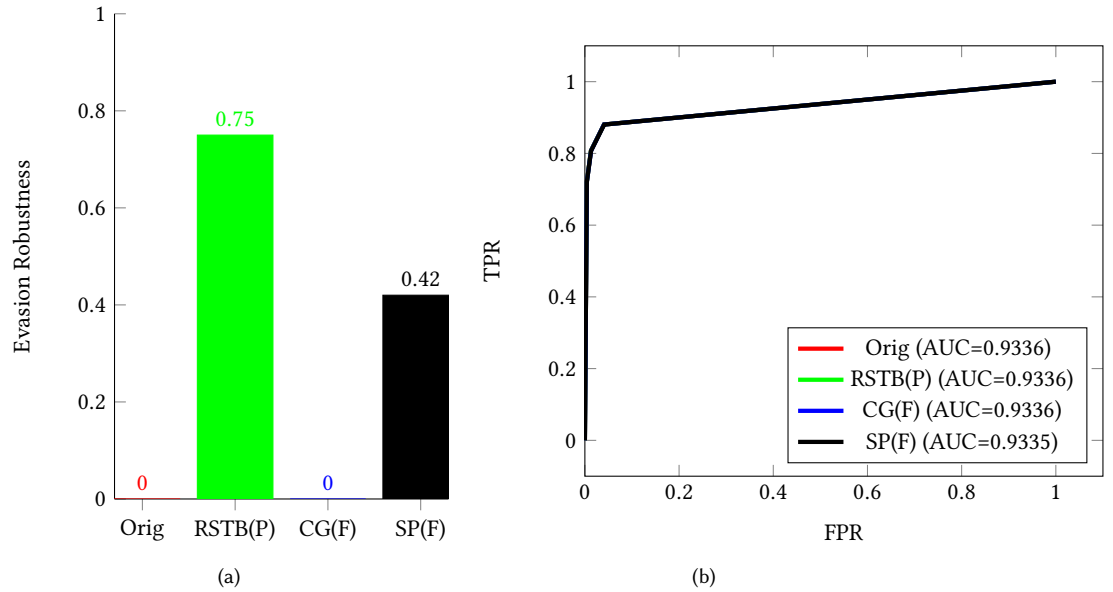


Fig. 15. Evasion robustness of the Random STB attack (a) and performance on clean data (b) for the MaMaDroid detection system. The underlining model is 3NN.

of 0% and 42%, respectively. The retraining processes are fully distinguishable, with a wide gap of 33% in evasion robustness. This shows that defense that relies on recognizing feature-space evasion attacks will not identify the Random STB attack. Lastly, the three retrained classifiers are basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 93% (Fig. 15b). Therefore, a highly robust classifier for the Random STB attack (the RSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

A.2.2 Black-Hole Statistical STB Experiment. At first, the Black-Hole statistical STB attack was run on the original MaMaDroid classifier Without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Black-Hole Statistical STB attack. As demonstrated in Fig. 16a, the BSTB(P) retrained classifier achieved an evasion robustness of 98%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 0% and 41%, respectively. The retraining processes are fully distinguishable, with a wide gap of 57% in the evasion robustness. This shows that defense that relies on recognizing feature-space evasion attacks will not identify the Black-Hole Statistical STB attack. Lastly, the two retrained classifiers are basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 93% (Fig. 16b). Therefore, a highly robust classifier for the Black-Hole Statistical STB attack (the BSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

A.3 DT

This section describes the evaluation of retrained classifiers on feature-space attacks of the DT model. The DT model was suggested in [6] as more effective than the 3 models of the original MaMaDroid [3]. Therefore, it was considered as

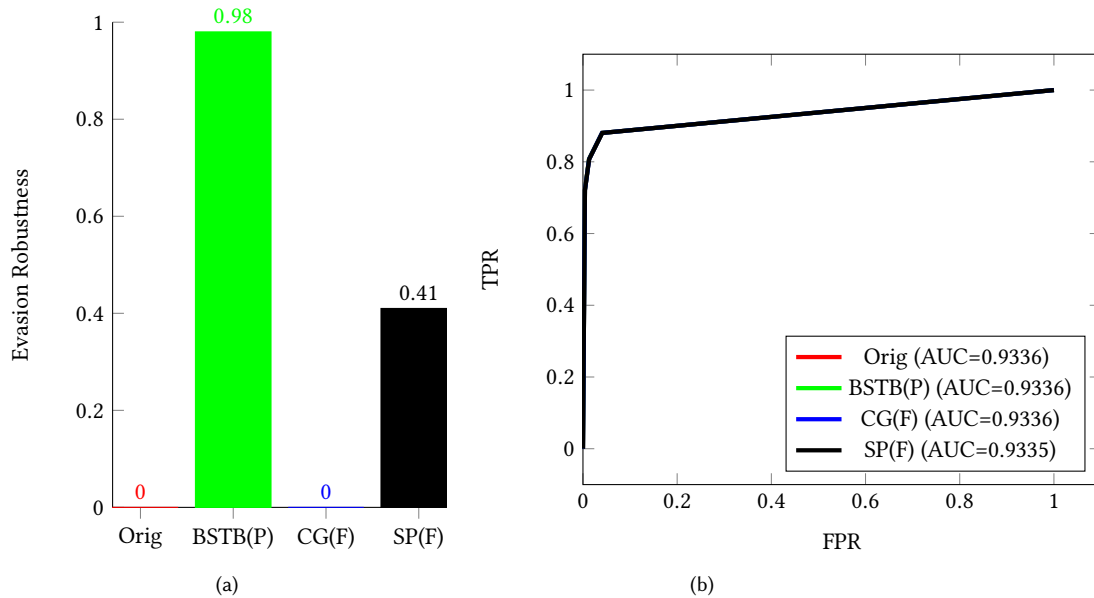


Fig. 16. Evasion robustness of the Black-Hole Statistical STB attack (a) and performance on clean data (b) for the MaMaDroid detection system. The underlying model is 3NN.

well. First, the Random STB experiment is described in Section A.3.1 and then, the Black-Hole Statistical STB experiment is detailed in Section A.3.2.

A.3.1 Random STB Experiment. First, the Random STB attack was run on the original MaMaDroid classifier without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Random STB attack. As depicted in Fig. 17a, the RSTB(P) retrained classifier achieved an evasion robustness of 73%. In comparison, the CG(F) and SP(F) retrained classifier resulted in an evasion robustness of 0%. The retraining processes are fully distinguishable, with a wide gap of 73% in evasion robustness. This shows that defense that relies on recognizing feature-space evasion attacks will not identify the Random STB attack. Lastly, the two retrained classifiers are basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 89% (Fig. 17b). Therefore, a highly robust classifier for the Random STB attack (the RSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

A.3.2 Black-Hole Statistical STB Experiment. At first, the Black-Hole statistical STB attack was run on the original MaMaDroid classifier Without any retraining process, and the classifier resulted in an evasion robustness of 0%. Then, to create the baseline, MaMaDroid was iteratively retrained with the Black-Hole Statistical STB attack. As illustrated in Fig. 18a, the BSTB(P) retrained classifier achieved an evasion robustness of 94%. In comparison, the CG(F) and SP(F) retrained classifiers resulted in an evasion robustness of 0%. The retraining processes are fully distinguishable, with a wide gap of 94% in evasion robustness. This shows that defense that relies on recognizing feature-space evasion attacks will not identify the Black-Hole Statistical STB attack. Lastly, the two retrained classifiers are basically as accurate as the original MaMaDroid classifier on clean data, with an AUC of more than 89% (Fig. 16b). Therefore, a highly robust

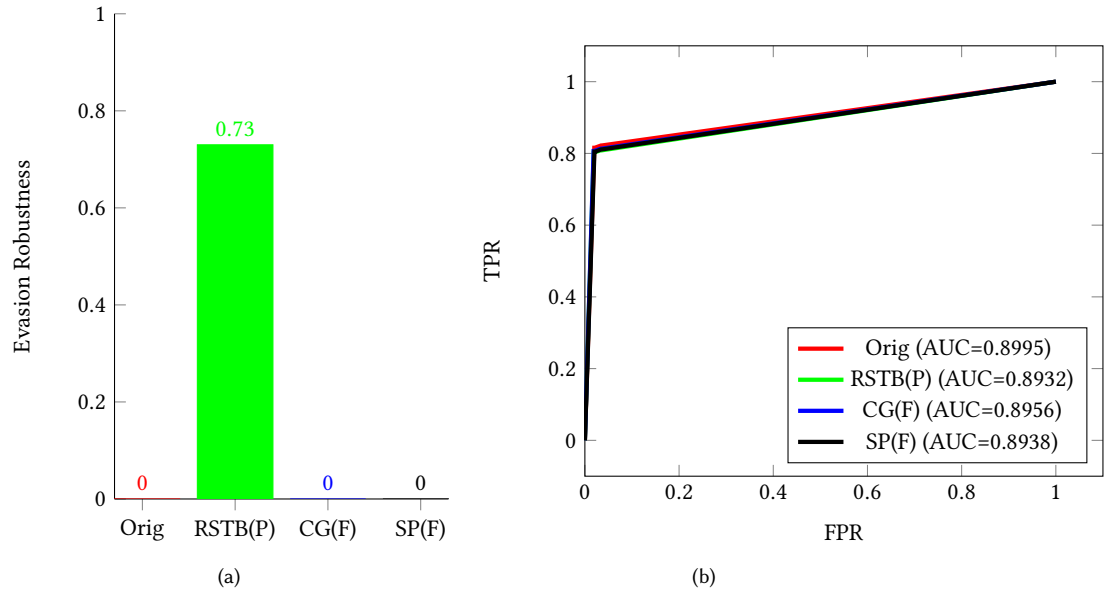


Fig. 17. Evasion robustness of the Random STB attack (a) and performance on clean data (b) for the MaMaDroid detection system. The underlining model is DT.

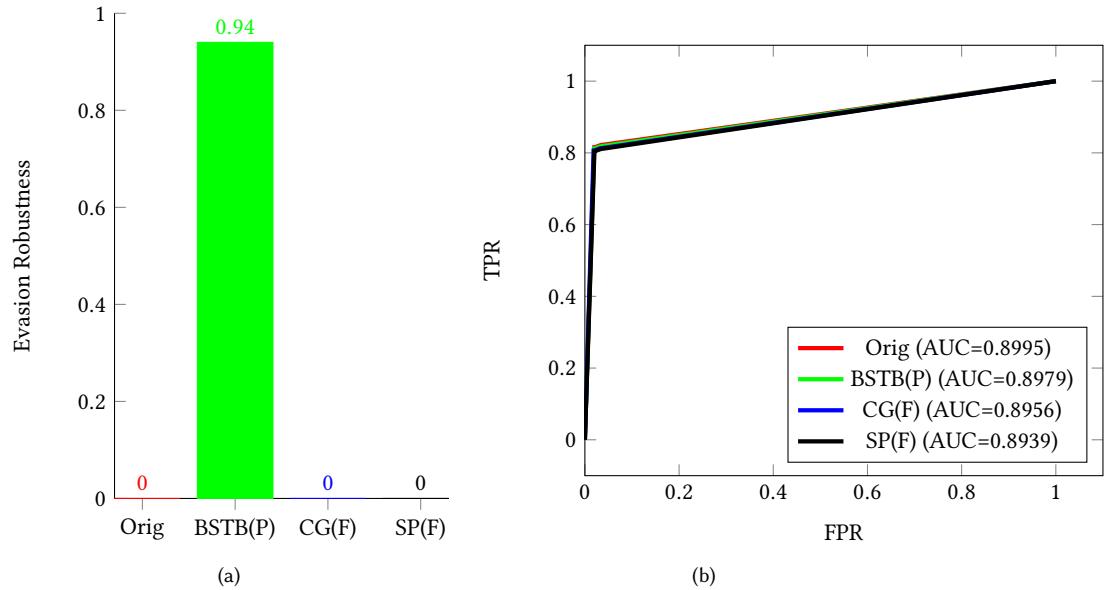


Fig. 18. Evasion robustness of the Black-Hole Statistical STB attack (a) and performance on clean data (b) for the MaMaDroid detection system. The underlining model is DT.

classifier for the Black-Hole Statistical STB attack (the BSTB(P) classifier) can be achieved without significant damage to its effectiveness on non-adversarial data.

Do You Think You Can Hold Me? The Real Challenge of Problem-Space Evasion Attacks

AT \ DEF	Basic	RSTB	BSTB
RSTB	0%	+79%	+79%
BSTB	0%	+96%	+96%

Table 4. Evasion Robustness of problem-space retrained classifiers against other problem-space evasion attacks. The basic classifier is MaMaDroid’s 1NN. The inspected problem-space evasion attacks (AT) are Random STB (RSTB) and Black-Hole Statistical STB (BSTB). Each retrained classifier (DEF) is described by its delta from the basic classifier (Basic).

AT \ DEF	Basic	RSTB	BSTB
RSTB	0%	+79%	+79%
BSTB	0%	+96%	+96%

Table 5. Evasion Robustness of problem-space retrained classifiers against other problem-space evasion attacks. The basic classifier is MaMaDroid’s 3NN. The problem-space evasion attacks (AT) are Random STB (RSTB) and Black-Hole Statistical STB (BSTB). Each retrained classifier (DEF) is described by its delta from the basic classifier (Basic).

AT \ DEF	Basic	RSTB	BSTB
RSTB	0%	+69%	+69%
BSTB	0%	+94%	+94%

Table 6. Evasion Robustness of problem-space retrained classifiers against other problem-space evasion attacks. The basic classifier is MaMaDroid’s DT. The problem-space evasion attacks (AT) are Random STB (RSTB) and Black-Hole Statistical STB (BSTB). Each retrained classifier (DEF) is described by its delta from the basic classifier (Basic).

B MAMADROID MODELS - PROBLEM-SPACE EVALUATION

This section reports the evaluation of 1NN, 3NN and DT retrained classifiers that confront the three STB attacks, to check whether or not the attacks are generalizable. This section is a continuation of Section 7.3, where only the RF model was described. The results show that the 1NN and 3NN retrained classifiers (Tables 4 and 5) show results similar to those of the retrained RF classifiers. The DT retrained classifiers (Table 6) were slightly less efficient. However, the same behavior that was established before can be seen with these results: Each retrained classifier with an STB attack, no matter which of the attacks was used, resulted in the same evasion robustness against the STB attacks. Therefore, as with the RF model, these STB attacks were proven to be generalizable.