Effectiveness of Moving Target Defenses for Adversarial Attacks in ML-based Malware Detection

Aqib Rashid, Jose Such

Abstract—Several moving target defenses (MTDs) to counter adversarial ML attacks have been proposed in recent years. MTDs claim to increase the difficulty for the attacker in conducting attacks by regularly changing certain elements of the defense, such as cycling through configurations. To examine these claims, we study for the first time the effectiveness of several recent MTDs for adversarial ML attacks applied to the malware detection domain. Under different threat models, we show that novel transferability and query attack strategies can increase the evasion rate by 50+% against these defenses across Android and Windows, with 90+% evasion rate in some cases. We also show that fingerprinting and reconnaissance are possible and demonstrate how attackers may obtain critical defense hyperparameters as well as information about how predictions are produced. Based on our findings, we present key recommendations for future work on the development of effective MTDs for adversarial attacks in ML-based malware detection.

Index Terms—Adversarial machine learning, Adversarial examples, Malware detection, Machine learning security, Deep learning

1 Introduction

ML models offer undeniable advantages in several domains [1], [2], including malware detection, where they can classify benign and malicious executables. However, ML models are vulnerable to adversarial attacks [3], [4], [5], which exploit the faults of ML algorithms to exert control over predictions. In particular, *evasion attacks* are performed by generating an adversarial example that crosses the decision boundary to *evade* the classifier [3], [4], where an attacker can have a malware sample predicted as benign [6]. Thus, defenses are needed to counter such attacks. Recent work has proposed several feature-based, gradient-based, and randomization-based defenses (e.g., [3], [6], [7], [8], [9], [10], [11], [12], [13]). However, several been proven ineffective for dealing with adversarial attacks recently [14], [15], including in the malware detection domain [16].

Recently, moving target defenses (MTDs) have been proposed as a remedy [12], [17], [18], [19], [20], [21]. MTDs use several ML models (rather than a single model) in a specific manner to defend against adversarial attacks. They aim to make attackers less effective at reconnaissance and targeted attacks by regularly moving the defense (e.g., constituent models, how predictions are produced) [21]. MTDs belong to the family of ensemble defenses, which aim to outperform single models in terms of robustness, increased complexity for the attacker, variance reduction, prediction accuracy, and generalization [22]. Therefore, several MTDs for adversarial ML have been proposed, using techniques such as increasing the heterogeneity of constituent models, diversity training of constituent models, dynamically regenerating the models, query-limiting, and strategically choosing models [12], [17], [18], [19], [20], [23], [24], [25].

However, prior to our work, the performance of MTDs against adversarial ML attacks under different threat models has not been evaluated, nor compared with each other, nor extensively compared with other defenses. It is important to assess whether MTDs may be a line worth investigating further to remedy adversarial ML attacks, and if so, which strategies and ways to design MTDs seem more effective.

In this paper, we present the first evaluation of several recent MTDs applied to the ML-based malware detection domain. To conduct this evaluation, we use transferability and query attack strategies from prior work as well as novel ones that we propose to maximize the evasion of MTDs. Additionally, we offer methods to conduct fingerprinting and reconnaissance to increase the understanding of how the target MTD works to enhance attacks, with minimal knowledge about it initially. Based on our evaluation, we offer recommendations for developing effective future MTDs. The main contributions of our work are:

1) We conduct the first evaluation of several MTDs to defend against adversarial ML attacks applied to the malware detection domain. Across Android and Windows, we show that the MTDs can be evaded with minimal information.

2) We examine the performance of MTDs using existing transferability and query attack strategies as well as novel improvements to these strategies for maximizing the evasion of MTDs. Our novel strategies increase the evasion rate by up to 50% versus prior attack strategies.

3) We show that it may be possible to fingerprint and recognize MTDs with a set of initial techniques that allow us to discover the predictive nature of the MTDs studied and, in some cases, some of their critical hyperparameters.

4) Informed by the evidence produced in our evaluation, we derive key insights and make crucial recommendations to help design more effective MTDs against adversarial ML.

The paper is organized as follows. Sections 2 and 3 provide the background and threat models considered. Sec-

• *The authors are with the Department of Informatics, King's College London, Strand, London WC2R 2LS, United Kingdom. Jose Such is also with VRAIN, Universitat Politècnica de València, Spain.*
E-mail: {aqib.rashid, jose.such}@kcl.ac.uk

tion 4 details the attack strategies used. Section 5 details the experimental setting. Sections 6-9 report our results. Section 10 synthesizes our findings and offers recommendations for building MTDs based on them. Section 11 discusses related work, and we conclude in Section 12.

2 Background

ML-based Malware Detection offers several advantages over signature-based detection methods, such as rapid assimilation of large datasets and the ability to generalize to unknown threats [5]. Typically, deep neural networks (DNNs) are trained on binary feature vectors representing *benign* (i.e., *goodware*) and *malware* software executables. For developing software representations that can be fed into ML models, *feature extraction* parses an executable into a binary feature vector. The quality of ML models depends on the features used during training [5], [26], [27]. However, adversarial ML has increased the attack surface [3].

Adversarial ML. Our work concerns a type of adversarial ML attack called *evasion attacks*, where an attacker perturbs the features of an input sample to obtain a specific prediction from the model [28], meaning that a malware sample is predicted as benign. An adversarial example is a perturbed version of the feature vector representing the executable. Even if the attacker does not have direct access to the target model, an attack can still be performed. Due to *transferability*, adversarial examples developed for one model may evade other models because of weaknesses shared by classifiers [3]. In a *transferability attack*, the attacker relies on the transferability property of adversarial examples to hold between the target model and a substitute model, which is an estimation of the target model for which a single DNN is commonly used [4], [5], [29]. Meanwhile, *query attacks* do not use substitute models [29], [30], [31], [32], [33], [34], [35], [36] but instead generate adversarial examples by iteratively perturbing a malware sample based on queries to the target model. In other domains, techniques for this include gradient and decision boundary estimation [29], [31], [33]. However, these techniques do not cater to the constraints of the malware detection domain regarding discrete features or functionality preservation. Instead, *software transplantation-based approaches* [32], [35], [37] can be used for query attacks. This involves perturbing a malware sample with benign "donor" features.

Defenses Against Adversarial ML. Numerous defensive approaches for adversarial ML attacks have been proposed. These include gradient-based [7], [8], feature-based [9], [10] and randomization-based [11], [12] approaches, as well as techniques like (ensemble) adversarial training [3], [8]. Most approaches are typically single-model defenses, where one model is made robust [24]. However, several recent surveys [14], [15], [16] have found these defenses to be ineffective.

Moving Target Defenses (MTDs) have been deployed in different security areas [21], [38], [39], [40], [41], [42] and applied in various fields, such as industrial control systems [43], network intrusion detection systems [44], [45], distributed systems [46], web applications [47], and cloud security [48]. MTDs regularly change their configuration and move system components to increase uncertainty and make it more difficult to understand their behavior [21].

MTDs Against Adversarial ML Attacks. MTDs have been proposed to defend against adversarial ML attacks [12], [17], [18], [19], [20], [21], [49], [50], [51] by using several ML models in a specific manner to produce a prediction. In this setting, MTDs claim to boost prediction accuracy, robustness, generalization, and to reduce variance compared with other defenses through techniques that "move" the defense's configuration. Techniques include model regeneration [19], [20], game-theoretic movement strategies [12], [17], [50], using adversarially-trained student models with different model selections [18], [19] and combinatorially-boosted ensembles [51]. Within the context of adversarial ML, MTDs can be categorized as either dynamic or hybrid. For an input sample X, a dynamic MTD may return the prediction y at first, before returning the prediction y' later, such that $y \neq y'$. Meanwhile, a hybrid MTD continues to return the same prediction y until a specific condition is reached, causing it to alter itself (e.g., regenerating models if a query budget is reached). Then, the prediction for X may be y' such that $y \neq y'$. Contrarily, other ensemble defenses (such as voting [52]) are static, where the same prediction for an input is returned. The non-static predictive nature of MTDs introduces yet another layer of complexity for attackers that is absent in other defenses.

Novelty & Contributions. While prior work has shown the ineffectiveness of single-model defenses for adversarial ML [15], [16], [28], [52], [53], [54], MTDs for adversarial ML have not been widely evaluated, nor compared with each other, nor extensively compared with other defenses. It is essential to determine if MTDs are a worthwhile research direction to protect against adversarial ML attacks and, if so, which approaches for designing MTDs appear more effective. Therefore, in this paper, we present the first evaluation of several MTDs using a range of attacks and threat models.

3 Threat Model

In our work, attackers aim to evade a feature-based ML model to have malware samples predicted as benign.

Target Model. The target model is an MTD consisting of several ML models. This MTD system could be considered as part of an AV software on a host, or acting as a defense at a network entry point, which exists in order to detect malware. To train these ML models, software executables are represented as binary feature vectors based on raw extracted features as shown in Figure 1. Following prior work on ML-based malware detection [32], [55], [56], the features we use include the libraries, API calls, permissions, or network addresses used, among others, which are provided by the datasets we use, detailed in Section 5. With the features $1 \ldots M$, a vector X can be constructed for each input sample such that $X \in \{0, 1\}^M$. Here, $X_i = 1$ or $X_i = 0$ indicates the presence or absence of feature i respectively. The feature vectors and their associated class labels are used to train the constituent binary classification models.

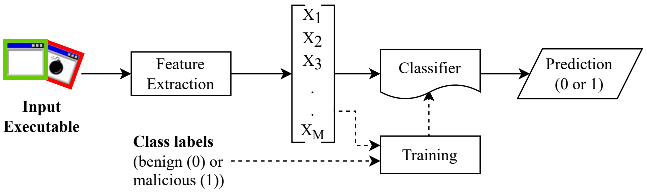


Fig. 1: Overview of a malware detection classifier.

The constituent models are then used in an MTD known as the *oracle* O. A user requests a prediction from O, which uses the constituent models in a specific manner to return a prediction. For example, an MTD may produce a probability distribution to *move* between (i.e., choose), the constituent models [12]. O only returns the prediction to limit the understanding of its behavior.

Attacker's Goal. The attacker's goal is to generate an adversarial example X' to evade O. Suppose $O\colon X \in \{0,1\}^M$ and we have some function $check()$ to check the functionality of an input sample. This goal can then be summarized as:

$$check(X) = check(X'); O(X) = 1; O(X') = 0 \qquad (1)$$

That is, the adversarial example X' must retain original malicious functionality from malware sample X but result in a *benign* prediction from O. For predictions, 1 represents the malware class and 0 the benign class.

Attacker Capabilities & Knowledge. We model two types of attackers with different levels of knowledge, as commonly featured in prior work [28], [33], [57]. Neither attacker knows that the target model is an MTD. The limited-knowledge *gray-box attacker* has access to the same training data as the target model and has knowledge of the representation of the features across the dataset. However, they have no knowledge of the parameters, configurations, or constituent models of the target model. This scenario represents when sensitive model information may have been leaked. Therefore, for transferability attacks (see "Adversarial ML" in Section 2), they train substitute models using the training data and attack them with the aim of generating adversarial examples that transfer to the oracle [3], [4], [57]. To conduct query attacks, the gray-box attacker uses their knowledge of the features to apply suitable perturbations using a software transplantation-based approach in a heuristically-driven manner. In contrast, the *black-box attacker* can only observe the predicted outputs for their queries. They have no knowledge about the target system but have some information pertaining to the kind of feature extraction performed (e.g., the static analysis that a malware detection classifier may consider). Therefore, they conduct a transferability attack which analogous to a chosen-plaintext attack as described in prior literature [6]. This is achieved by querying the oracle sufficiently to train a substitute model, which is an estimation of the oracle, and then attacking it to generate adversarial examples that transfer to the oracle [3], [4], [57] (unlike the gray-box attacker, who can use the training data). For query attacks, the black-box attacker also uses a transplantation-based approach but without any extra information to guide the attack [32], [35].

4 Attack Strategies

In this section, we present the attack strategies used to evaluate MTDs under the threat model described previously.

4.1 Transferability Attacks

In transferability attacks, attackers construct substitute models that are approximations of the oracle. Adversarial examples are then generated for these substitute models, in anticipation that they will transfer to the oracle [3], [4], [5], [29], [53], [54], [58]. To evaluate MTDs against transferability attacks, we use practical strategies from prior work as well as our own. Attack strategies from prior work are included as baselines and to show MTDs' performance with already existing, general strategies that are not tailored to MTDs. These include the Single DNN strategy [4], where a single DNN is used as the substitute model [4], [5], [29], as well as the Ensemble DNN strategy [58], where several DNNs are used as the substitute models.

Additionally, we propose a novel attack strategy that specifically considers that the target model *may be* an MTD. This is akin to an attacker conducting a SQLi attack against a server, just in case the attack succeeds, even if the target application is not using a SQL database. For this, we present two main improvements on previous strategies. First, our transferability attack strategy utilizes an ensemble of diverse substitute models (including different ML families). Second, as a novel technique to increase attack success against MTDs, our attack strategy aims to maximize the transferability of adversarial examples across substitute models prior to evaluating them on the oracle by checking transferability across (part of) the substitute models constructed. We include these two additions because MTDs may change the model used for predictions dynamically, and so ensuring a degree of transferability of adversarial examples between substitute models maximizes success on the oracle, as we confirm later on experimentally.

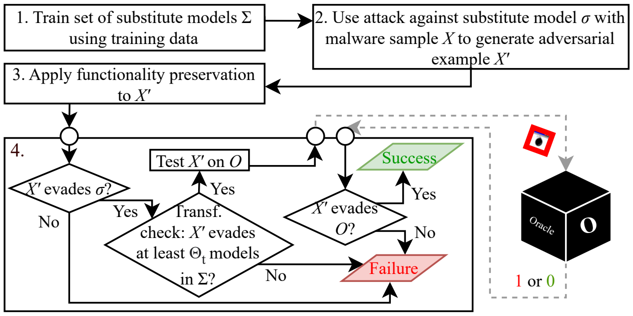

1. Train set of substitute models Σ using training data
2. Use attack against substitute model σ with malware sample X to generate adversarial example X′
3. Apply functionality preservation to X′
4.
X′ evades σ?
No
Yes
Transf. check: X′ evades at least Θt models in Σ?
Yes
No
Test X′ on O
X′ evades O?
Yes
No
Success
Failure
Oracle
O
1 or 0

Fig. 2: Overview of our transferability attack strategy.

Figure 2 summarizes the key steps of our transferability attack strategy, with each step described in detail next:

Step 1) Developing Substitute Models. The aim here is to train substitute models that are an estimation of O (the oracle). It is well-established that a model can be evaded by utilizing different model architectures and a dataset independent of it [3]. As detailed in Section 5, we particularly favor an ensemble of substitute models as diverse as possible to increase the chances of finding highly-transferable adversarial examples that will transfer to the oracle.

For each attacker, the procedure for constructing the substitute models differs due to their capabilities. The *black-box attacker* has no capabilities beyond observing the predictions for their queries. Therefore, under this scenario, we develop a synthetic dataset (Δ) that is an estimation of the input-output relations of the oracle. To develop Δ, we use a set B_{train} (containing benign and malware input samples) and query O with each input sample $X \in B_{train}$, recording the prediction. For example, if $O(X) = 0$, then the input-output relation $X \mapsto 0$ is stored in Δ. Δ is then used to train the ensemble of diverse substitute models (Σ) for our attack strategy. This provides white-box access to models that are an estimation of the oracle. For the other attack strategies that we test (that is, the Single DNN [4] and Ensemble DNN [58] strategies), Δ is used to train the substitute model(s) according to their procedures. Prior work in other domains has suggested that substitute models require a large training dataset [4] and, while unrealistic, an attacker could create a replica of the oracle with infinite queries. However, we show later that a high evasion rate can be achieved with a small set of input samples (i.e., $|B_{train}| \leq 100$). With fewer queries to the oracle, an attacker can remain stealthier, with lower chances of their behavior being detected [30], [59].

Conversely, the *gray-box attacker* has access to the training data of the defense. Therefore, we train an ensemble of diverse substitute models (Σ) using the available training data. The advantage is that no queries to the oracle are necessary, reducing the time cost and the risk of adversarial behavior being detected. However, as we demonstrate later in Section 7, using the training data of the target models may not always lead to the best attack performance. Because they are based on direct queries to each oracle, the substitute models developed by the black-box attacker may reflect their characteristics and behavior better. Also, given that the gray-box attacker has knowledge of the dataset, an alternative technique is to use universal adversarial perturbations (UAPs) [60], [61]. With UAPs, the same sets of perturbations are reused and applied to several malware samples to generate adversarial examples. However, we show later in Section 7 that this attack approach is less successful.

Step 2) Generating an Adversarial Example. As usual in all transferability attacks, once the substitute models have been constructed, we have full access to them. Then, we can use an attack against a substitute model with existing white-box attacks from the literature to generate an adversarial example X' from a malware sample X (e.g., FGSM [54], JSMA [53]— see Section 5 for the full list of white-box attacks used in the evaluation). The idea is that this adversarial example is likely to *transfer* to the target model (the oracle).

Step 3) Applying Functionality Preservation. We then validate the perturbations that have been used to generate X'. This is done in order to preserve the original functionality of X within the feature space as a lower bound, which is essential in this domain. Otherwise, one may have an adversarial example that evades the oracle but has lost its functionality. In this process, any invalid perturbations found are reverted to their original values. Note that the perturbations that are valid or invalid and the way in which functionality preservation works in practice depend on the particular target platform for the malware. We detail valid and invalid perturbations for each of the cases we explore in the evaluation (Android and Windows) later in Section 5.

Step 4) Transferability Across Substitutes & Evaluation. We then ensure that the adversarial example X' still evades the substitute model used to build it, as the process for validating the perturbations may have reversed some perturbations used to cross the decision boundary. If so, after this, we include a crucial new step where we look at how successful X' is in evading the other substitute models (i.e., a local transferability check is performed before actually testing X' on the oracle O). As the attack success relies on the transferability property holding between Σ and O, our hypothesis is that adversarial examples that transfer better across all σ in the ensemble of substitute models Σ are more likely to evade O. For this, we check whether X' transfers across a proportion of substitute models (Θ_t) before testing it on O. For example, $\Theta_t = 0.75$ means that X' must evade 75% of substitute models. If X' adequately transfers across the substitute models and then evades O, it is counted as a success, whereas a failure is when O is not evaded by X'.

4.2 Query Attacks

Query attacks generate adversarial examples by iteratively perturbing an input sample [29], [30], [31], [32], [33], [34], [36], rather than using substitute models. Most query attacks, however, are designed for the image recognition domain and therefore perturb features continuously, meaning they are less effective in our domain due to its constraints, such as discrete features and functionality preservation. For example, as explained in [32], a feature for an API call (e.g., $WriteFile()$) cannot be perturbed continuously (e.g., $WriteFile() + 0.001$). For this, an entirely new feature is required, offering the same functionality.

To overcome these problems, we can use software transplantation-based approaches. This means that features from benign samples are used to perturb a malware sample (e.g., a feature is added to a malware sample), which can be conducted in a scenario with less [32], [37] or more [35] information about the target model. Overall, this allows malware samples to cross the decision boundary of the oracle while catering to the constraints of this domain. When conducting this attack, limiting the number of queries to the oracle is critical, as adversarial behavior can be detected when analyzing queries for abnormalities [30]. Moreover, some MTDs use query budgets, which may hinder the construction of adversarial examples. Hence, we use the parameter n_{max} to govern the maximum number of allowed queries. We offer two approaches for performing query attacks in black-box and gray-box scenarios.

Black-box Query Attack. Under the black-box scenario, the attacker has no knowledge of the target model besides the predicted output. This means that the attack is conducted in a non-heuristic manner, where randomly-chosen features are perturbed accordingly. Our black-box query attack strategy is inspired by Rosenberg et al.'s [32]. However, our threat model considers *less information* about the target model under the black-box condition. We assume no access to prediction confidence scores or usage of sliding windows, so we use a modified version of their decision-based attack without these assumptions. Furthermore, if allowed by the dataset (see Section 5), our black-box attack strategy makes use of both feature addition *and* feature removal to increase the possible avenues for evasion. In our black-box query attack, a malware sample X is perturbed using features from a set of benign donor samples ($\mathcal{B}$) to generate an adversarial example X' using a transplantation-based approach. However, this is performed in a non-heuristic manner because of this attacker's weaker capabilities. This means that the feature to perturb during each iteration of the attack is chosen *randomly*, which can be added to (0 to 1) or removed from (1 to 0) the feature vector (as permitted by the dataset). Feature removal can be performed when features absent in a benign sample (but perhaps functionally insignificant in X) are removed to cross the decision boundary. The attack process takes constraints regarding functionality preservation into consideration by only making perturbations that are allowed (see Section 5 later). The transplantation of the features continues until O is evaded, n_{max} is reached, or the possible features of the benign sample are exhausted (in which case another benign sample may be used).

Gray-box Query Attack. The gray-box attacker has increased knowledge about the target model, allowing them to conduct potentially more effective attacks. That is, since this attacker has access to the training data and knowledge of the statistical representation of the features across the dataset, features can be added to a malware sample in a heuristically-driven manner (as opposed to randomly per the black-box query attack strategy). The gray-box attack strategy perturbs those features in a malware sample X, which appear more frequently in benign samples. This promotes traversal of the decision boundary in a superior manner to the random approach adopted by the black-box attacker. We show later that this significantly increases attack success and reduces queries to the oracle.

Hence, as per Figure 3, from the data available to the attacker, features from benign samples are first sorted by their frequency into a vector $\vec{s}$. This vector can be reused whenever the attack is conducted. Then, using a malware sample X, the next feature from $\vec{s}$ that preserves the original functionality of X is added to generate X', (rather than a randomly-chosen feature per the black-box strategy). Recall from before that only certain perturbations for each target platform will preserve the functionality of the malware sample (platform-specific details of valid perturbations in Section 5). As before, perturbations are validated for functionality preservation before being tested on O. The transplantation of features continues until the generated adversarial example X' evades O, n_{max} is reached, or the possible features are exhausted.

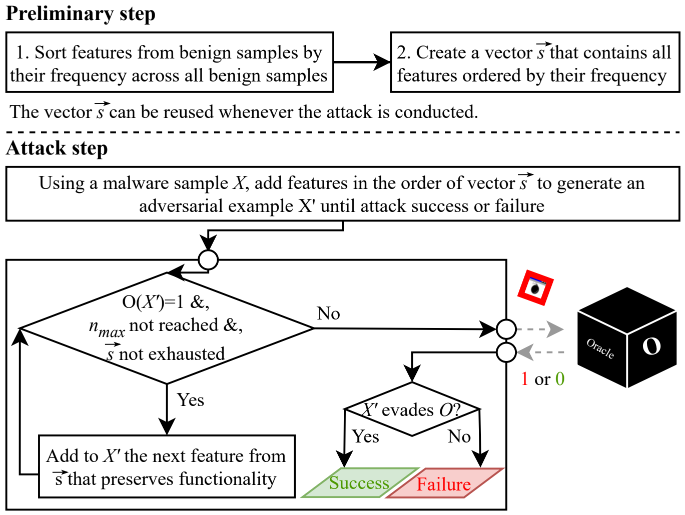

Preliminary step
1. Sort features from benign samples by their frequency across all benign samples
2. Create a vector $\vec{s}$ that contains all features ordered by their frequency
The vector $\vec{s}$ can be reused whenever the attack is conducted.
Attack step
Using a malware sample X, add features in the order of vector $\vec{s}$ to generate an adversarial example X' until attack success or failure
O(X')=1 &,
n_{max} not reached &,
$\vec{s}$ not exhausted
No
Yes
Add to X' the next feature from $\vec{s}$ that preserves functionality
X' evades O?
Yes
No
Success
Failure
1 or 0
Oracle
O

Fig. 3: Overview of our gray-box query attack strategy.

5 Experimental Setup

Datasets. In many security applications, sampling from the true distribution is challenging and sometimes impossible [62], [63]. Particularly, the number of publicly-available, up-to-date datasets in our domain is a well-known problem, which limits the remits and conclusions of academic work in this domain [62], [63]. To partially mitigate this, we use three datasets that cover different platforms and collection methods that have been used previously (e.g., [35], [55], [61], [64], [65], [66], [67], [68]): DREBIN for Android malware [69], SLEIPNIR for Windows malware [27], and AndroZoo for Android malware [70].

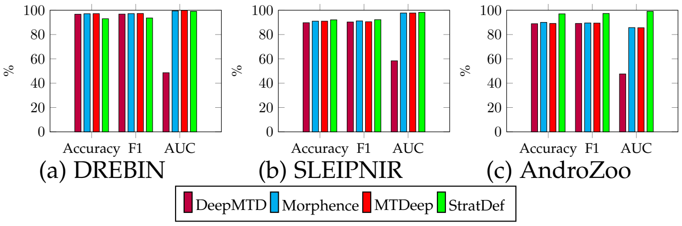
DREBIN has 123,453 benign samples and 5,560 malware samples, based on extracted static features. Hence, we use 5,560 samples from each class (benign and malware). SLEIPNIR consists of 19,696 benign samples and 34,994 malware samples. The features of this dataset are derived from Windows API calls in PE files parsed by the LIEF library [71] into a vector representation. SLEIPNIR is used to represent Windows out of simplicity, with a convenient binary feature-space, enabling a clearer comparison between the Android and Windows datasets as our work is in the feature-space. To keep the dataset balanced, we use 19,696 samples from each class. AndroZoo [70] is a large-scale dataset with timestamped Android apps from between January 2017 and December 2018. This dataset offers apps from different stores and markets, with VirusTotal summary reports for each. Similar to prior work [35], [68], [72], we consider an app malicious if it has 4 or more VirusTotal detections, and benign if it has 0 VirusTotal detections (with apps that have 1-3 detections discarded). Thus, the final dataset contains $\approx$ 150K recent applications, with 135,859 benign apps and 15,778 malicious apps. We use 15,778 samples from each class (benign and malware). As in recent publications [55], [73], and for completeness, we use a large number of features for each dataset, i.e., 58,975 for DREBIN, 22,761 for SLEIPNIR, and 10,000 for AndroZoo. Note, however, that we provide an experiment in Appendix D with far fewer features for some datasets (500 features), suggesting that the number of features does not play a role in the results.

The datasets are split randomly for each class according to the Pareto principle, with an 80:20 ratio for training and test sets. Subsequently, using the same ratio, this training set is further partitioned into the final training and validation sets for constructing the models and defenses we evaluate. Effectively, this produces the 64:16:20 split that has been commonly used before (e.g., [74], [75]). The validation set is used to tune the models during development. The remaining test set is used in the attacks and is further split into training, validation, and the final test set. The training set here is used in the black-box transferability attack and corresponds to B_{train}, which is used to obtain the input-output relations of O for developing Δ. For DREBIN, $|B_{train}| = 1423$. As SLEIPNIR and AndroZoo are larger, we perform additional splits to reduce $|B_{train}|$ to 1513 and 1494 respectively. Meanwhile, the malware samples in the final test set are the input samples for transferability and query attacks. For DREBIN, SLEIPNIR, and AndroZoo, there are 229, 230 and 234 such samples respectively.

We consider established guidelines for conducting malware experiments [76]. For example, as the models in our evaluation decide whether an input sample is benign or malicious, retaining benign samples in the datasets is necessary. This also means that we do not need to strictly balance datasets over malware families. Instead, we balance datasets across the positive and negative classes, and randomly select unique samples from each class to appear in the training and test sets (without any chance of repetition) [5], [16], [27].

Moving Target Defenses. We evaluate four MTDs that are configured to provide maximal robustness using the parameters suggested in their *original papers*. As we show in the next subsection, these MTDs offer sound performance in non-adversarial conditions for malware detection, so they are adequate for use in this domain. That is, these MTDs are domain-agnostic. We present the configuration for each defense in Appendix A. ***DeepMTD*** is a hybrid MTD [20]. n student models are generated by perturbing weights of a base DNN, with w controlling the amount of perturbations. To make a prediction, if more than T x 100% of the outputs from the student models are the same, the input sample is considered non-adversarial (and the majority class is the final prediction). Otherwise, it is classified as adversarial. To keep the system "moving", new models are generated when the system is idle. ***Morphence*** is a hybrid MTD [19]. It generates n student models from a base model by shuffling weights randomly. Then, p student models are adversarially-trained (where $p \leq n$). For a user's query, the prediction of the most confident student model is returned. Morphence uses a query budget; student models are regenerated if the number of queries exceeds Q_{max}. ***MTDeep*** [12] is a dynamic MTD. This defense models the interactions between attackers and defenders as a Bayesian Stackelberg game to produce a strategy vector to choose models at prediction-time based on attack intensity. The resulting strategy can be pure (where only a single model is chosen) or mixed (where one of several models can be chosen). ***StratDef*** [17] is a dynamic MTD that provides a framework to systematically construct, select, and strategize a set of models. It gives particular consideration to the heterogeneity of its constituent models to minimize transferability, and it uses different optimizers to choose the best strategy for selecting a model at prediction time.

Performance of MTDs in Non-adversarial Conditions. Some MTDs in this paper have not been applied to the malware detection domain and instead have been first applied in the image recognition domain. However, similar to defenses such as adversarial training (which was first applied to images and then to malware detection [55]), MTDs can operate effectively across a range of domains. As we show in Figure 4, the MTDs that we choose to evaluate offer sound predictions (90+% accuracy) on test sets in the malware detection domain, thereby showcasing their ability to work well in this domain too. DeepMTD seems to offer moderate performance considering AUC, but we still choose to include it in the evaluation just in case this is a result of the mechanism used against adversarial ML attacks, so we can compare it with other MTDs.



(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 4: Malware detection performance of evaluated MTDs.

Other Evaluated Defenses. As MTDs are a type of *ensemble* defense — that utilize several models — we compare their performance with other types of defenses that use ensembles (in some manner) as baselines. ***Voting*** is a static ensemble defense that has previously been applied to malware detection [52], [77]. Multiple models are involved in making the prediction, which is determined by *majority* or *veto* voting. In majority voting, each constituent model makes a prediction, and the class with the most predictions is returned. In veto voting, if any constituent model returns a "malware" prediction, the entire system returns this prediction. ***Ensemble adversarial training*** is a defense that trains a single model by using adversarial examples for other models [8]. The adversarially-trained model is then used to produce predictions. We use an adversarially-trained neural network (NN-AT) that is constructed using the training data described previously. Then, it is boosted by training with a quantity of adversarial examples that is 25% of the size of the training data, which is a sizeable amount to promote high adversarial robustness. Using a set of vanilla models (see Appendix B), we generate adversarial examples by applying a range of attacks (listed in the following section) and conducting the functionality preservation procedure (described later). A proportion of these adversarial examples are then used for adversarially-training the neural network model (NN-AT).

Substitute Models (Transferability Attack). For our transferability attack (Section 4.1), we use an ensemble of diverse substitute models to maximize evasion. With minimal tuning of hyperparameters to demonstrate the simplicity of our attack strategy, we construct four substitute models: a decision tree (DT), a neural network (NN), a random forest (RF) and a support vector machine (SVM) — see Appendix B for architectures. We use the scikit-learn [78], Keras [79] and Tensorflow [80] libraries for training. We use $\Theta_t = 0.75$ to ensure that an adversarial example evades the majority of substitute models before testing it on the oracle. We also test other values in Sections 6 and 7.

Generating Adversarial Examples. For the *transferability attack* (Section 4.1), we generate adversarial examples for substitute models using a variety of attacks: the Basic Iterative Method [81], Decision Tree attack [59], Fast Gradient Sign Method [54], Jacobian Saliency Map Approach [53] and SVM attack [59]. These attacks produce continuous feature vectors and do not consider functionality preservation. Therefore, after applying these attacks, we round the values in the generated continuous feature vectors to produce discrete feature vectors, representing the presence or absence of a feature (e.g., usage of a particular library). For example, if an attack changes the value of a particular feature to < 0.5, it is set to 0 in the feature vector; meanwhile, if the value is ≥ 0.5, it is set to 1 in the feature vector. We then check for invalid perturbations to preserve functionality within the feature-space. Only after invalid perturbations are reverted does an adversarial example proceed further in the attack pipeline according to the attack strategy.

For the *query attack* (Section 4.2), we apply the attack strategies under the black-box and gray-box scenarios. In both scenarios, a malware sample is perturbed by transplanting features from benign samples [32], [35], [37]. For example, if a particular feature is enabled in benign samples (i.e., its value is 1 in the feature vectors), it is added to the malware sample (changed from 0 to 1 in the feature vector for the malware sample) in order to move closer to crossing the decision boundary. The difference between the black-box and gray-box attack strategies lies in the choice of which features to perturb first. The gray-box attacker perturbs features based on their frequency in benign samples based on their knowledge of the dataset. Meanwhile, the black-box attacker chooses which features to perturb randomly, as in [32], as no further information is available.

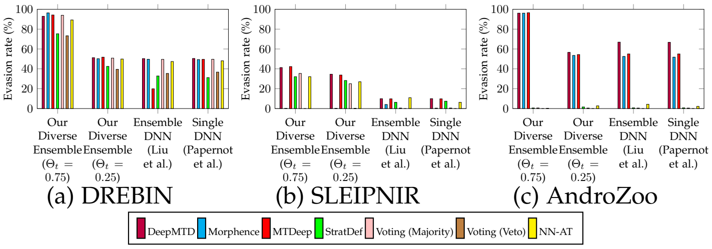
In both transferability and query attacks, the *permitted (valid) perturbations* (either feature addition or removal) for each dataset are determined by consulting with industry documentation, previous work [27], [35], [56], [61], and the feature representation for each dataset. DREBIN and AndroZoo allow for both feature addition and removal [56], [82] — see Appendix C for a summary of the allowed perturbations. In contrast, due to the encapsulation by LIEF's feature extraction process when developing the dataset, we can only perform feature addition on SLEIPNIR. This results in a more *constrained attack surface* for SLEIPNIR, as there is a greater restriction on the perturbations that can be applied, leading to, as we will see later on, less effective attacks. The procedure we use offers a lower bound of functionality preservation within the feature-space, similar to prior work [55], [56], [83]. While we remain in the feature-space, the perturbations we perform could be translated to the problem-space as well. For example, feature addition could be achieved by adding dead-code or by using opaque predicates [84]. Feature removal — which is more complex but still achievable — could be performed by rewriting the dexcode, encrypting API calls and network addresses (e.g., removing the features but retaining functionality).

Evaluation Metrics. We use several metrics in our work, similar to previous work [4]. The *evasion rate* is defined as the number of adversarial examples that evade the oracle over the number of adversarial examples that evade the substitute models. Furthermore, as MTDs may employ different models at prediction-time, an adversarial example may not always evade the oracle. Therefore, we include the *repeat evasion rate (RER)* as an additional metric for evaluating the oracle. This measures the number of times an adversarial example evades the oracle out of 100 attempts. We also use standard ML metrics such as accuracy, F1, AUC, and false positive rate (FPR) to evaluate the models and defenses.

6 Black-box Results

6.1 Transferability Attack

Evasion rate. In general, we observe that attacks against the majority of MTDs are effective regardless of the attack strategy used. Figure 5 shows that MTDs seem easily evaded, with the average evasion rate across all attack strategies sitting at 56% for DREBIN, 12.9% for SLEIPNIR, and 50.3% for AndroZoo. However, the best attack performance is achieved by our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$), with peak evasion rates of 96.3% for DREBIN, 42.3% for SLEIPNIR, and 96.1% for AndroZoo. This demonstrates that the majority of MTDs possess insufficient movement mechanisms and do not adequately prevent attackers from developing sufficient representations of them. This is especially true in less constrained environments, such as with DREBIN and AndroZoo. For these datasets, the attack surface is greater due to the ability to perform more perturbations (i.e., both feature addition and removal), which leads to greater evasion.



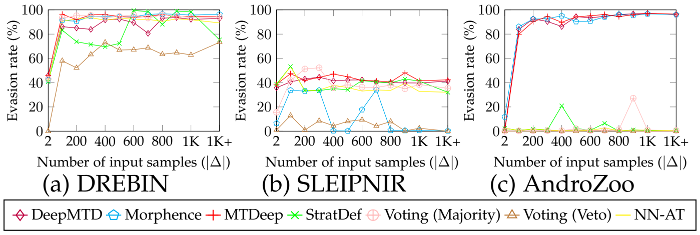
(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 5: Evasion rate of attack strategies.

For SLEIPNIR, there are greater restrictions on the perturbations that can be applied by attacks. Hence, this represents a more constrained environment for the attacker, where MTDs have it easier to defend themselves. The overall lower average evasion rate implies that MTDs can work well in these conditions. In particular, Morphence and veto voting are evaded the least, with an average evasion rate across all strategies sitting at $\approx 1\%$ for Morphence and 0% for veto voting. However, although veto voting offers good protection, it exhibits a high FPR (see Section 8 later). Interestingly, StratDef seems least evaded across all the datasets — for AndroZoo, the evasion rate of the attack sits at below 1%. Also, although transferability attacks seem to be less effective for SLEIPNIR, we show later that query attacks are more effective and manage to extensively evade all defenses even with this dataset.

Comparing attack strategies. In terms of how the different strategies perform, Figure 5 shows that our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$) achieves the highest evasion rate. For DREBIN and AndroZoo, the evasion rate sits at 65+% against most defenses (including single-model defenses), which is considerably greater than the $\approx 50\%$ evasion rate of the baseline attack strategies (Ensemble DNN and Single DNN). This is because, unlike other attack strategies, our attack strategy maximizes the transferability of adversarial examples locally across an ensemble of diverse substitute models before evaluating them on the oracle, thereby reducing attack failures. Even in a more constrained environment (with SLEIPNIR), our attack strategies surpass the Single DNN and Ensemble DNN attack strategies with a peak evasion rate of 40+% (versus $\approx 10\%$). As our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$) offers the highest evasion rate across all datasets, we use this approach for the rest of the black-box evaluation.

Varying the number of input samples ($|\Delta|$). We now vary the input samples to construct substitute models. This allows us to explore whether an effective transferability attack can be performed against MTDs with fewer samples to construct substitute models. Previously, the maximum number of input samples was used to develop Δ (i.e., $|B_{train}|$). We therefore cap $|\Delta|$ and query each defense with different numbers of input samples from each class, up to the maximum available (1K+, i.e., $|B_{train}|$). As $|\Delta|$ increases, it should produce a better representation of the oracle (up to a point), while fewer input samples should be less detectable. Recall that we use our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$) for this experiment. Figure 6 shows that for DREBIN and AndroZoo, as $|\Delta|$ increases past two (where the evasion rate sits at $\approx 40\%$), the evasion rate increases and reaches up to 90+% after which it stabilizes. Generally, veto voting and StratDef are more robust defenses with less evasion for DREBIN these datasets. Fluctuations in the performance of some defenses are attributable to several reasons, such as model regeneration (Morphence) and dynamic model selection (StratDef). Moreover, Δ can become noisier due to inaccurate data, resulting in substitute models with a poorer approximation of the oracle's behavior at some intervals, explaining the dips in evasion rates for some defenses (DeepMTD and veto voting). Interestingly, most defenses are no better than the single-model defense, NN-AT.

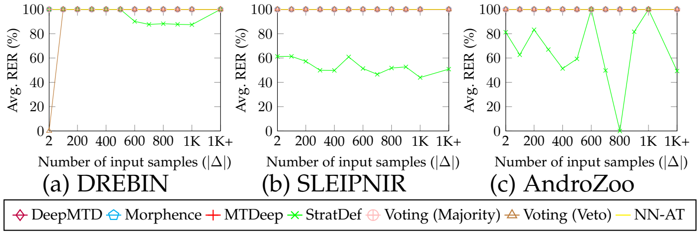


(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 6: Evasion rate vs. number of input samples.

Meanwhile, in the more constrained environment with SLEIPNIR, attack performance varies more, with the evasion rate peaking at 53%. However, there is significantly less evasion compared with DREBIN and AndroZoo due to a smaller attack surface being available to exploit. Morphence's model regeneration procedure causes it to exhibit greater fluctuations in its performance under this more constrained dataset. Despite this, evasion is achievable even if its query budget is exceeded. For AndroZoo, we observe that nearly all MTDs are evaded in the same manner as DREBIN. As $|\Delta|$ increases past 100, the evasion rate against these defenses sits at 80+%. Interestingly, though, StratDef and the non-MTDs exhibit significant robustness; in some cases, the attack fails to achieve significant evasion.

Repeat Evasion Rate. Figure 7 shows the average repeat evasion rate (RER) versus the number of input samples. This measures how many times (out of 100) an adversarial example evades the oracle, averaged across the attack. For StratDef, the average RER is 95% for DREBIN (minimum of 87%), 53% for SLEIPNIR (minimum of 44%), and 65.4% for AndroZoo (minimum of 0%). Meanwhile, for DeepMTD, Morphence, MTDeep, and voting, we achieve 100% RER, which means that adversarial examples have a higher chance of repeatedly evading the same oracle. This is due to the predictive nature of each defense, the characteristics of their *movement* mechanisms, or the lack of diversity among their constituent models. As single-model defenses such as NN-AT are completely static, they experience 100% RER.

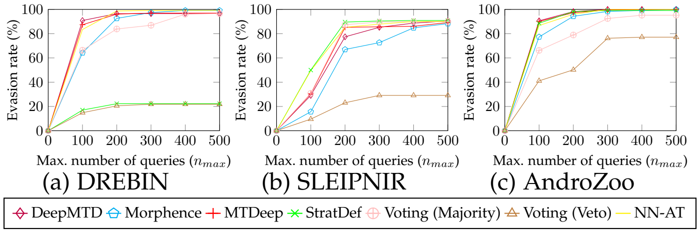


(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 7: Average RER vs. number of input samples. Lines appear merged as most defenses exhibit 100% RER.

6.2 Query Attack

Unlike the transferability attack, the black-box query attack does not use substitute models. Instead, a malware sample is perturbed using features from benign samples until it evades the oracle or resources are exhausted (e.g., number of queries, available features to perturb).

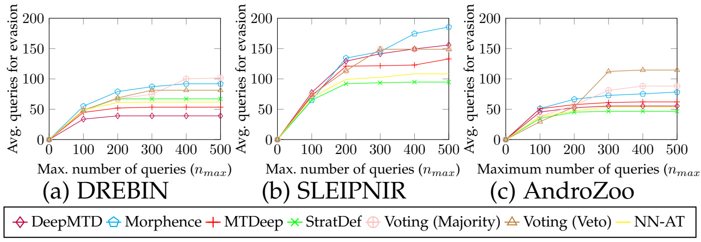


(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 8: Evasion rate vs. maximum number of queries.

Figure 8 shows the results of the black-box query attack. This attack clearly outperforms the transferability attack, as models that were hardly evaded by the transferability attack (especially for SLEIPNIR and AndroZoo) are evaded by the query attack with fewer queries. The average evasion rate of the query attack across all defenses sits at 72% for DREBIN, 69% for SLEIPNIR, and 90.3% for AndroZoo (with peak evasion rates of 99%, 91%, and 100% respectively). For DREBIN, veto voting and StratDef exhibit the greatest robustness, with the attack only achieving $\approx 20\%$ evasion rate. For SLEIPNIR, although the environment is still more constrained in terms of the attack surface (i.e., fewer possible perturbations), veto voting and Morphence can be evaded still, unlike in the transferability attack. As before, the evasion rate is relatively lower for SLEIPNIR because there are fewer perturbations that can be applied to the malware samples, limiting the evasion opportunities. However, as the perturbations used by the attack are tailored to this domain, we observe that the attack yields better attack performance. This is evident in the results of the attack against StratDef for AndroZoo, where the black-box transferability attack yielded insignificant evasion. Veto voting remains robust across the datasets because a single model influences the prediction and the perturbations selected by the attack are insufficient to evade it. However, it exhibits a higher FPR (see Section 8 later). Defenses such as DeepMTD, MTDeep, and majority voting — as well as the single-model defense, NN-AT — offer minimal protection against this attack.

There is a strong correlation between n_{max} (the maximum number of queries allowed) and the evasion rate. In fact, we also examine the number of queries in Figure 9, which shows the average number of queries required to evade each defense versus n_{max}. Most defenses can be evaded with less than 100 queries for DREBIN, 150 for SLEIPNIR, and 100 for AndroZoo — see Appendix E for extended results. For other domains that use continuous features (such as image recognition), query attacks may require substantially more queries [31] to achieve attack success compared with domains that use discrete features. This is because perturbations in a discrete feature-space (e.g., 0 to 1) have a greater effect on the final prediction, meaning that fewer of them may be required to achieve evasion, compared with smaller perturbations made per query (e.g., +0.01) in a continuous feature-space. For all datasets, attacks against static defenses (or those behaving statically) need fewer queries for evasion.



(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 9: Avg. queries for evasion vs. max. queries.

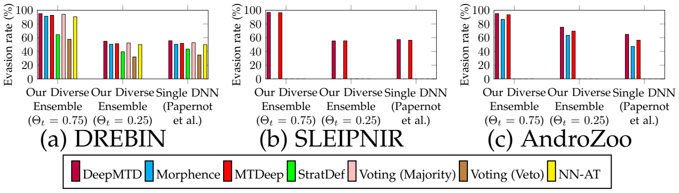
Furthermore, experiments for the transferability attack show that all defenses, besides StratDef, exhibit 100% average RER. For the query attack, the same is true for DREBIN — even for StratDef — where adversarial examples achieve 100% average RER against the defenses. For SLEIPNIR, the average RER for StratDef is 53.9% as the adversarial examples are not as effective as the transferability attack, while all other defenses exhibit 100% average RER. For AndroZoo, the average RER is 100% for all defenses for both attacks, except for StratDef, whose average RER sits at 53.5%. We demonstrate that our attack strategy remains effective despite the black-box scenario.

7 Gray-box Results

Under this attack scenario, we have access to the training data of defenses, which we use to develop substitute models for the transferability attack. Moreover, we have knowledge of the statistical representation of the features, which can be used to enhance the query attack. With greater information about the dataset, we also evaluate each defense against Universal Adversarial Perturbations (UAPs) [60], [61].

7.1 Transferability Attack

We evaluate each defense against our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$), having shown its superior performance. Our attack strategy, using the Diverse Ensemble but with $\Theta_t = 0.25$ is also included as an alternative, as is the Single DNN strategy [4] as a baseline for comparison. The Ensemble DNN is not included as it has been proven less effective than our attack strategy, and its performance is on-par with the Single DNN strategy. Recall that, under the gray-box scenario, the substitute models for each oracle are equivalent for each attack strategy as they are trained on the same training data as the defenses, rather than a synthetic dataset that is an estimation of O's input-output relations.



(a) DREBIN (b) SLEIPNIR (c) AndroZoo

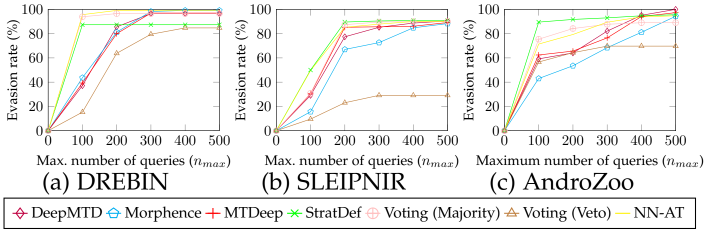
Fig. 10: Evasion rate of attack strategies.

Figure 10 shows the evasion rate for the attack strategies under this threat model. In the less constrained environments with DREBIN and AndroZoo, our attack strategy can achieve a 55+% evasion rate across the MTDs for DREBIN and mostly 80+% for AndroZoo. Interestingly, when compared with the black-box transferability attack, the trends and the performance of each defense are similar. However, under a smaller attack surface and hence a more constrained environment with SLEIPNIR, there is greater difficulty in evading the MTDs in general. For the defenses that can be evaded, similar results are observable as in the black-box attack, with a 95+% evasion rate against the weaker defenses. In contrast, the vast majority of MTDs and defenses offer effective resilience against the attack, with defenses such as Morphence and veto voting remaining quite resilient to adversarial examples. The attack is also unable to evade StratDef, while the evasion rate for majority voting is non-zero, with less than 10 successful adversarial examples in total (which is why some results seem missing — see Appendix E for full results).

In general, weaker defenses suffer more against the gray-box attack than the black-box attack, while for attacks against less static defenses, having access to the training data (as in the gray-box threat model) is not as useful in conducting attacks as it may seem. This phenomenon can be attributed to several factors. Firstly, substitute models for the black-box attack are more representative of each oracle, as they are based on direct queries to them when the synthetic dataset Δ is constructed. This suggests that, in these circumstances, the substitute models capture the oracle's traits and behavior better, even with a smaller dataset. Consequently, adversarial examples for the black-box substitute models may transfer better to the oracle. Moreover, a larger training set for substitute models — as in the case of the gray-box transferability attack — may increase overfitting in the substitute models, resulting in inferior attack performance (see Appendix G for experiments evaluating attack performance versus training set size).

7.2 Query Attack

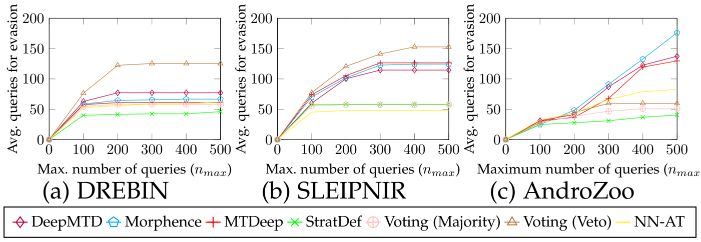
The gray-box query attack uses the frequency of features in benign samples to determine the order of transplantation. We hypothesize that this should reduce the number of queries needed and improve attack performance.



(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 11: Evasion rate vs. max number of queries.

Figure 11 shows the evasion rate versus n_{max}. Similar to the black-box query attack, there is a strong correlation between n_{max} and the evasion rate, which reaches up to 99% for DREBIN, 94% for SLEIPNIR, and 100% for AndroZoo (with an average evasion rate of 84.6%, 91%, and 79.6% respectively, across all defenses and maximum query sizes). However, the average evasion rate is also higher than the black-box query attack (by $\approx 20\%$) as well as the gray-box transferability attack, with nearly all defenses being evaded even in the more constrained environment of SLEIPNIR. As has been a common theme, defenses exhibiting mostly static behavior perform worse, with an evasion rate of 90+%. This shows that a heuristically-driven query attack method specifically designed for this domain can achieve better evasion against the MTDs, even in more constrained environments. Interestingly, veto voting still offers robustness for SLEIPNIR, although we achieve a higher evasion rate with this attack than previously.



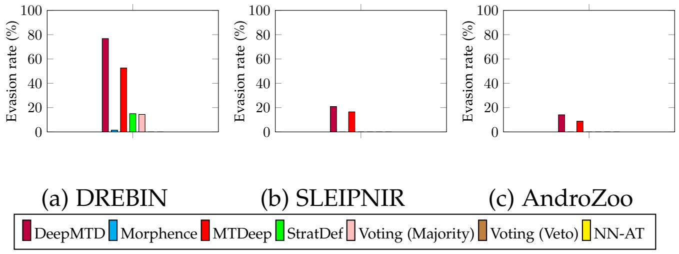
(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 12: Avg. queries for evasion vs. max. queries.

The gray-box query attack also performs better than the black-box query attack with respect to the queries required to achieve this level of evasion. From Figure 12, we observe that fewer than 80 queries for DREBIN are enough to cripple most defenses. For SLEIPNIR, there is a further 44% decrease in the average number of queries compared with the black-box attack. Meanwhile, AndroZoo also presents a reduction in the number of queries required to achieve evasion. Therefore, on the balance of numbers, the gray-box query attack outperforms the black-box attack considering the datasets and the number of queries permitted. Moreover, the average RER for the gray-box query attack for DREBIN is 100% for all defenses but 0.2% higher than the black-box query attack on StratDef for SLEIPNIR. The average RER for AndroZoo similar to its black-box counterpart, with a 0.4% reduction for StratDef, but 100% for other defenses.

7.3 Universal Adversarial Perturbations

Recent research has identified universal adversarial perturbations (UAPs) as a cost-effective technique for producing adversarial examples [60], [61]. With UAPs, a set of perturbations can be applied to multiple malware samples to generate adversarial examples. In other words, sets of perturbations that are known to cause input samples to cross the decision boundary are reused across several malware samples. We evaluate each defense against adversarial examples generated with UAPs to compare with our attack strategies. The UAPs are derived from the dataset that the gray-box attacker has access to. Using the adversarial examples produced by the gray-box transferability attack, we examine if a set of perturbations has been reused exactly to produce adversarial examples from its original samples. In such cases, the adversarial examples are deemed to have been generated with UAPs.



(a) DREBIN (b) SLEIPNIR (c) AndroZoo

Fig. 13: Evasion rate of attack (UAPs).

Figure 13 shows the evasion rate of adversarial examples generated with UAPs. There is significantly less evasion compared with the transferability and query attacks, with an average evasion rate of 20% for DREBIN, 4.7% for SLEIPNIR, and 3.3% for AndroZoo (with peak evasion rates of 77%, 21%, and 14.1%, respectively). DeepMTD and MTDeep are evaded the most, especially for SLEIPNIR and AndroZoo. Similar to the transferability attack for these datasets, most defenses are not evaded by adversarial examples generated through UAPs, with a significantly lower average evasion rate overall.

8 Beyond Adversarial Robustness

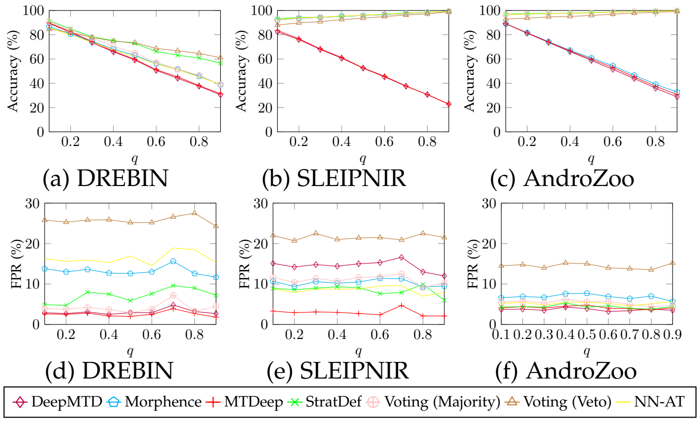
Some defenses consistently perform well against black-box and gray-box attacks. Defenses like Morphence, StratDef, and veto voting appear more resilient than others. Hence, it may seem appealing to deploy these defenses. However, metrics beyond adversarial robustness need to be considered, especially in the malware detection domain where the false positive rate (FPR) must remain low [37], [55], [85], [86]. A high FPR means a less reliable and more frustrating service as analysts are flooded with false alarms and users find their benign queries misclassified as malware. Therefore, we simulate different system conditions with varying degrees of adversity to understand how each defense performs under these scenarios while considering other metrics. To do this, each defense is queried over 1000 times with different proportions of adversarial examples represented by the attack intensity (q). For example, at $q = 0.5$, half of the queries to the defense are adversarial, while the remaining proportion is an equal number of benign and non-adversarial malware input samples. The adversarial examples are those produced previously under the gray-box transferability attack to maintain uniformity, as the black-box attack produces different adversarial examples for each oracle. We evaluate each defense with $0.1 \leq q \leq 0.9$ to avoid the less likely cases of system conditions where $q = 0$ (where there are no adversarial queries) and $q = 1$ (where all queries are adversarial).

Figure 14 shows the accuracy and false positive rate (FPR) of each defense when queried over 1000 times with different proportions of adversarial examples (see Appendix F for F1 and AUC). While veto voting and Morphence offer good accuracy, particularly for SLEIPNIR, the greater FPR associated with these defenses can also be seen across all values of q. The FPR of veto voting is much higher than most other defenses. If such defenses were deployed in a real-world setting, users would often receive incorrect predictions, which is unsuitable for the malware detection domain, as explained before. Conversely, MTDs like StratDef offer a more balanced performance in this regard, while also being more well-rounded against the adversarial threat.

9 Fingerprinting & Reconnaissance

An enhanced understanding of defenses can be gained through fingerprinting and reconnaissance. Recall that reconnaissance can yield useful information for *future* attacks. If an attacker has a greater awareness of how an MTD operates, they could improve future attacks. We introduce two methods to demonstrate how this could be achieved, showing promising results and widening the scope for more elaborated attacks in future work.

Determining Predictive Nature of Defenses. As discussed in Section 2, an MTD is dynamic or hybrid in nature



(a) DREBIN (b) SLEIPNIR (c) AndroZoo

(d) DREBIN (e) SLEIPNIR (f) AndroZoo

Fig. 14: Accuracy and FPR vs. q.

(whereas other defenses may be static). For defenses exhibiting static behavior (as well as hybrid defenses, to some degree), predictions will remain the same. This is ideal for an attacker, as it guarantees that an adversarial example can be reused, thereby ensuring its future success. Conversely, a defense behaving dynamically will exhibit less predictable behavior, leading to reduced repeat evasion (as seen before).

Determining this predictive nature can be accomplished by querying each model with different input samples repeatedly (n times). Subsequently, a consensus mechanism determines whether predictions for the same input sample have changed across the queries. Fluctuations in predictions are an indication of a dynamic defense, while static defenses will always produce the same prediction for a single input sample. For instance, upon applying this technique with $n = 100$, several defenses such as DeepMTD, MTDeep, and voting (majority and veto) mostly produce predictions statically. This is understandable, as voting is static by nature. Meanwhile, the lack of variance in predictions indicates that DeepMTD may not be effectively regenerating student models, while MTDeep may be employing a pure strategy, which occurs when it computes the optimal strategy to be that only a single model from its ensemble serves the predictions. This is also why the evasion rate against these defenses is perhaps considerably higher in our previous experiments. Meanwhile, Morphence also delivers predictions statically until its query budget is exceeded. However, as we show in the following section, if the attacker discovers that the oracle produces different predictions after n queries, they can adapt their attack to use $< n$ queries to ensure the oracle's static behavior. StratDef exhibits dynamic behavior; for example, predictions for the same input samples vary by an average of 5.7% for DREBIN and 2.7% for SLEIPNIR on the test set described in Section 5.

Determining Movement of Hybrid Defenses. Hybrid defenses provide static predictions until a condition is met, at which point predictions for may differ from before. For example, Morphence uses a query budget for this. However, this may be discoverable using the method described previously to determine the oracle's predictive nature. This is an example of stealing hyperparameters in ML [87].

Across the queries, the specific point where the oracle modifies its predictions can indicate whether a query budget exists and allow for an estimation of its value. For example, Morphence's queuing system means that a longer waiting time implies greater system utilization. Therefore, when system utilization is low, we determine the predictive nature of the defense but with a single input sample and a much larger value of n (e.g., 10,000). The intended result is that predictions vary across n to give an indication of the value of any query budget. If predictions do not change, it could be due to the input sample or because n is smaller than the query budget. In an experiment to discover the query budget for Morphence (Q_{max}), Figure 15 shows that the prediction for the same input sample varies between 3-5K and 7-8K queries. This implies that the oracle (which is an example of a Morphence instance) changes after $\approx$ 1000 queries (or $Q_{max} \leq$ 1000), which can be confirmed by using another input sample. With this knowledge, we could develop Δ for a black-box transferability attack without exceeding the query budget to ensure the static behavior of the oracle or perform a query attack with an n_{max} lower than the estimated Q_{max}. Recall that this is an *evaluation instance* of Morphence. In practice, one may find that a different n is required to conduct this reconnaissance exercise.

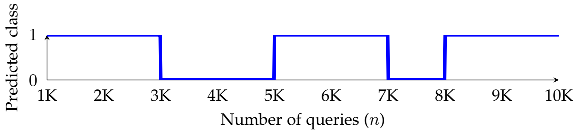


Fig. 15: Prediction for the same input sample changes between 3-5K and 7-8K queries, implying that $Q_{max} \leq 1K$.

10 Discussion & Recommendations

In this section, we discuss the overall findings from our evaluation. Based on these and our experimental results, we make key recommendations for developing effective MTDs against adversarial attacks in ML-based malware detection.

In some cases, the evaluated MTDs are no better than the other ensemble or single-model defenses. Among the evaluated MTDs, DeepMTD and MTDeep perform poorly across the board, regardless of the type of attack strategy. Our results show that both seem to exhibit a consistent lack of variance in predictions, leading to almost static behavior in practice due to inherent weaknesses and flaws in their design. As already observed in previous work [18], DeepMTD's mechanism for regenerating its student models only works when the defense is idle for some time. Regarding MTDeep, we observe that in this domain, its optimizer only generates pure strategies; that is, it only uses a single model in practice to make the predictions. Therefore, these defenses are unable to take advantage of the main characteristic that an MTD should offer, which is dynamic behavior to increase the attacker's uncertainty.

Meanwhile, Morphence and StratDef offer some degree of robustness in different attack scenarios. In particular, our results show that StratDef performs well for transferability attacks under both black-box and gray-box attack scenarios. Meanwhile, Morphence offers slightly better performance than other MTDs against query attacks, especially for lower values of n_{max} where the attacker faces more restrictions on the number of queries for their attack. The improved performance of these defenses can be attributed to more dynamic behavior (i.e., "how to move"). We observe in practice that both Morphence and StratDef vary their behavior dynamically and use higher model diversity (i.e., "what to move") with diverse constituent models beyond DNNs, which makes attacks against these MTDs less successful. Despite this, their performance is far from perfect, and in some threat models and attacks, particularly the gray-box query attack, both Morphence and StratDef can be well-evaded. Also, and particularly in the case of Morphence, we have shown that it can be fingerprinted, leaving key hyperparameters that govern its behavior exposed to attackers. Intriguingly, the number of constituent models of an MTD does not strongly correlate with robustness. In fact, MTDs such as DeepMTD (which use several constituent models) offer less robustness than MTDs, which use fewer models, such as StratDef and Morphence. This highlights the need for better constituent models and movement strategies.

Yet, we believe that there is scope for MTDs to be a promising direction for defending against adversarial ML, especially in the malware detection domain. If designed well, MTDs have the potential to introduce a layer of complexity and uncertainty for the attacker to thwart attacks. Hence, based on our results, we make key and actionable recommendations for developing effective MTDs to motivate further work in this area.

We believe that MTDs designed for and applied to any domain should follow these recommendations in order to promote maximal robustness, as they promote the fundamental aspects of what constitutes an MTD. However, future research work may be required to investigate the nuances of different domains, as the performances is known to vary accordingly as we have shown in this work.

Beyond designing for movement: Ensure dynamic behavior in practice. Our empirical analysis reveals a critical finding: even MTDs explicitly designed for movement, such as MTDeep and DeepMTD, can exhibit static behavior in practice. This highlights the importance of not only designing MTDs with dynamic movement mechanisms but also rigorously testing whether these mechanisms function as intended when deployed. If an MTD behaves statically in practice, it is essential to adapt or redesign its movement mechanism to ensure effective dynamic behavior. Our evidence demonstrates that MTDs which exhibit static or insufficiently dynamic behavior are significantly less effective at mitigating adversarial ML threats compared to MTDs with robust dynamic mechanisms. In fact, we show that defenses with inadequate movement mechanisms can perform no better than static single-model defenses such as adversarial training. For instance, dynamic or hybrid MTDs, such as Morphence or StratDef, offer superior performance across various threat models and attack strategies, as evidenced by improved evasion rates and other evaluation metrics. Furthermore, ensuring consistent and effective movement mechanisms can help reduce high repeat evasion rates by introducing inconsistencies in how adversarial examples behave against the oracle.

Increase the diversity of "what to move". We recommend that the constituent components of MTDs (i.e., the models) must be diverse. This is actionable by considering the use of different model families. Our work has shown that using a dynamic or hybrid MTD may still be ineffective if its constituent models can be evaded through transferability. To minimize transferability between the constituent models, the constituent models need to be as diverse as possible. In fact, we have seen that MTDs that use different families (and not just DNNs) seem to perform better. This means that other model families (e.g., random forests, support vector machines, etc.) should also be included in the constituent models.

Be independent on "how to move". We recommend that the movement mechanisms for MTDs should not be influenced solely by user behavior. Some defenses (e.g., Morphence) regenerate models to "move" their configuration, which is impacted exclusively by user activity (e.g., number of user queries), rather than by an independent mechanism (e.g., time or randomness). A user-influenced mechanism makes the oracle an easy target. Therefore, MTDs must "move" independently and cycle through configurations to prevent attacks. This can be achieved with effective sequencing, strategies or model cycles at prediction-time.

Consider leakage of hyperparameters. We recommend that the design of MTDs must not expose sensitive information about their operation. Hyperparameters are carefully designed and chosen for a specific context and may compromise a defense if leaked. For example, in Section 9, we demonstrate how a query budget can be discovered. This could make it easy for the attacker to consider the available query budget in a future attack, allowing them to take advantage of the defense before it changes its behavior. Therefore, it is critical that the design of an MTD accommodates the potential leakage of hyperparameters. The defender should operate under the assumption that hyperparameters may be leaked. For example, in the scenario described earlier, a method to mitigate hyperparameter leakage could be to cycle through models at prediction-time (randomly or strategically) so that it cannot be reliably ascertained how often predictions are changing.

Design or couple MTDs with greater system awareness is vital. We recommend that MTDs need more information about the world they operate within, which can be achieved by coupling them with other types of defenses. While an MTD has several advantages, it cannot identify when an attack is taking place. For instance, in our query attacks, this means that after several queries, most MTDs can be evaded as single-model defenses (like adversarially-trained models). Therefore, greater system awareness is vital, especially for protecting against query attacks given that particularly dynamic MTDs could vary their strategy at run-time. Prior work [30], [88] has shown that stateful detection methods could help detect attacks against single-model defenses in some cases. Therefore, an actionable research avenue could be to explore whether such *stateful* detection methods as well as cyber-threat intelligence could be combined with an MTD to offer increased awareness and robustness. This would lead to an adaptive MTD system for detecting and reacting to potential *attacks in progress* (e.g., by changing strategies or deciding to make a different move).

11 Related Work

Szegedy et al. [3] introduced adversarial examples, demonstrating that they transfer across different models. Other work explored transferability attacks within other domains (e.g., image recognition) [4], [89], [90]. Papernot et al. [4] explored methods for attacking remotely-hosted single-model classifiers based on a black-box threat model introduced in

[57], which was further explored in the malware detection domain [5], [29], [33], [91]. Liu et al. [58] then demonstrated that adversarial examples generated using ensembles of DNNs can lead to the evasion of a black-box image classifier. Our transferability attack improves on these approaches, as we use an ensemble of diverse substitute models and validate the transferability of adversarial examples across the substitute models before testing on the oracle. Prior work also developed query attacks (primarily for images) [29], [30], [31], [32], [33], [34], [36] to generate adversarial examples using techniques such as gradient and decision boundary estimation. These attacks cannot deal with discrete features and functionality preservation [32]. Compared to prior work [32], our query attack does not assume access to prediction scores or use sliding windows, so we consider a threat model with even less information. Additionally, our gray-box query attack chooses which features to perturb in a heuristic manner rather than randomly.

Other work has explored model stealing, which is a different adversarial aim from ours [87], [92], [93], [94]. For instance, [92] shows how to obtain parameters from remote classifiers with partial knowledge of models. Several studies on adversarial ML have been conducted in other domains, such as network systems and website fingerprinting [95], [96], [97], [98], [99]. Nonetheless, we show that the hyperparameters of MTDs and details about their behavior can be stolen in a black-box setting.

A challenge within ML-based malware detection is model sustainability. As malware evolves, it becomes difficult to generalize models to detect unseen behavior, making classifiers unsustainable [66], [100], [101]. Due to *concept drift*, models may make poor decisions when facing the latest threats (but not always [65]). Prior work has suggested periodically retraining models [102], [103], though this may reduce their learning ability. Thus, the constant evolution of malware makes it a moving target in its own right. To deal with this, MTDs could use a detection system for concept drift [66] and then retire vulnerable constituent models.

Our work is part of a series of evaluations of defenses against adversarial ML attacks. Prior work has demonstrated that single-model defenses are ineffective at dealing with adversarial examples [15], [16], [28], [52], [53], [54]. He et al. [104] showed that weak ensembles are insufficient against adversarial ML using image datasets. Work conducted in our domain has provided a grim view of the capabilities of available defenses [16], [52]. We are the first to conduct a comparative evaluation of MTDs. Our work proposes novel attack strategies for increasing evasion against different models, especially MTDs. Moreover, past research on MTDs [12], [17], [18], [19], [20], [21], [49], [50], [51], [105], [106] has not evaluated MTDs under different threat models, nor compared MTDs with each other, nor offered practical recommendations for developing effective MTD systems based on an evaluation of recent MTDs. Although [106] makes some suggestions, they are different from ours, as we do not recommend randomization or using only DNNs.

12 Conclusion

We studied the effectiveness of several MTDs against evasion attacks in ML-based malware detection. For this, we used existing transferability and query attack strategies as well as novel strategies specifically tailored to MTDs. Under different scenarios, we demonstrated that our attack strategies achieve high evasion with minimal queries to the target model. Moreover, we demonstrated that it may be possible to understand how a target model operates through fingerprinting and reconnaissance methods.

Based on our findings, we provided recommendations for developing effective MTDs. MTDs that behave statically should be avoided with dynamic behavior validated, while MTDs that utilize diverse constituent models with effective movement should be favored. That is, an MTD should use models from different families (beyond just DNNs or just different DNN architectures) to maximize diversity for limiting the effectiveness of transferability attacks. Furthermore, these models should be used dynamically and moved strategically to maximize uncertainty and complexity for the attacker. In addition, we suggested that MTDs be coupled with greater system awareness to detect and react to *attacks in progress*. This could pave the way for a promising way forward in dealing with attacks. This may be accomplished by developing a response that is both automated and stateful [30], and/or based on cyber-threat intelligence [107], [108]. Moreover, a potential limitation of our study is that there is room for greater investigation into the efficiency of MTDs, and the resources that are required to develop them both securely and efficiently. Future work should aim to consider and recommend methods for developing secure and practical MTDs.

Acknowledgements

This research was partially funded by the INCIBE's strategic SPRINT (Seguridad y Privacidad en Sistemas con Inteligencia Artificial) project with funds from the EU-NextGenerationEU through the Spanish government's Plan de Recuperación, Transformación y Resiliencia.

References

[1] Kaiming et al., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[2] C. Chio and D. Freeman, *Machine learning and security: Protecting systems with data and algorithms.* " O'Reilly Media, Inc.", 2018.

[3] Szegedy et al., "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2014.

[4] Papernot et al., "Practical black-box attacks against machine learning." *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519, 2018.

[5] Rosenberg et al., "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*. Springer, 2018, pp. 490–510.

[6] Chakraborty et al., "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[7] Papernot et al., "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[8] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[9] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[10] Wang et al., "Random feature nullification for adversary resistant deep architecture," *arXiv preprint arXiv:1610.01239*, 2016.

[11] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," *arXiv preprint arXiv:1711.01991*, 2017.

[12] Sengupta et al., "Mtdeep: boosting the security of deep neural nets against adversarial attacks with moving target defense," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] Rosenberg et al., "Adversarial machine learning attacks and defense methods in the cyber security domain," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

[14] Carlini et al., "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.

[15] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *International conference on machine learning*. PMLR, 2018, pp. 274–283.

[16] R. Podschwadt and H. Takabi, "On Effectiveness of Adversarial Examples and Defenses for Malware Classification." *International Conference on Security and Privacy in Communication Systems*, pp. 380–393, 2019.

[17] A. Rashid and J. Such, "Stratdef: Strategic defense against adversarial attacks in ml-based malware detection," *Computers & Security*, vol. 134, p. 103459, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823003693

[18] Qian et al., "Ei-mtd: Moving target defense for edge intelligence against adversarial attacks," *ACM Transactions on Privacy and Security (TOPS)*, vol. 25, no. 3, may 2022. [Online]. Available: https://doi.org/10.1145/3517806

[19] A. Amich and B. Eshete, "Morphence: Moving target defense against adversarial examples," in *Annual Computer Security Applications Conference*, ser. ACSAC. New York, NY, USA: Association for Computing Machinery, 2021, p. 61–75. [Online]. Available: https://doi.org/10.1145/3485832.3485899

[20] Q. Song, Z. Yan, and R. Tan, "Deepmtd: Moving target defense for deep visual sensing against adversarial examples," *ACM Trans. Sen. Netw.*, vol. 18, no. 1, oct 2021. [Online]. Available: https://doi.org/10.1145/3469032

[21] Cho et al., "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[23] Yang et al., "Dverge: diversifying vulnerabilities for enhanced robust generation of ensembles," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5505–5515, 2020.

[24] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu, "Improving adversarial robustness via promoting ensemble diversity," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4970–4979.

[25] S. Kariyappa and M. K. Qureshi, "Improving adversarial robustness of ensembles with diversity training," *arXiv preprint arXiv:1901.09981*, 2019.

[26] Grosse et al., "Adversarial perturbations against deep neural networks for malware classification." *arXiv preprint arXiv:1606.04435*, 2016.

[27] Al-Dujaili et al., "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 76–82.

[28] Papernot et al., "Sok: Security and privacy in machine learning," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 399–414.

[29] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.

[30] S. Chen, N. Carlini, and D. Wagner, "Stateful detection of black-box adversarial attacks," in *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, 2020, pp. 30–39.

[31] Chen et al., "Hopskipjumpattack: A query-efficient decision-based attack," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1277–1294.

[32] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Query-efficient black-box attack against sequence-based malware classifiers," in *Annual Computer Security Applications Conference*, 2020, pp. 611–626.

[33] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2137–2146.

[34] Li et al., "Qeba: Query-efficient boundary-based blackbox attack," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1221–1230.

[35] Pierazzi et al., "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2020, pp. 1308–1325. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00073

[36] A. N. Bhagoji, W. He, B. Li, and D. Song, "Practical black-box attacks on deep neural networks using efficient query mechanisms," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[37] Yang et al., "Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 288–302.

[38] Paruchuri et al., "Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 895–902.

[39] M. Tambe, *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011.

[40] B. C. Ward, S. R. Gomez, R. W. Skowyra, D. Bigelow, J. Martin, J. Landry, and H. Okhravi, "Survey of cyber moving targets second edition," 2018.

[41] S. Jajodia, A. K. Ghosh, V. S. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense II - Application of Game Theory and Adversarial Modeling*, ser. Advances in Information Security. Springer, 2013, vol. 100. [Online]. Available: https://doi.org/10.1007/978-1-4614-5416-8

[42] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding focus in the blur of moving-target techniques," *IEEE Security & Privacy*, vol. 12, no. 2, pp. 16–26, 2014.

[43] A. C. Pappa, A. Ashok, and M. Govindarasu, "Moving target defense for securing smart grid communications: Architecture, implementation & evaluation," in *2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2017, pp. 1–5.

[44] H. Wang, F. Li, and S. Chen, "Towards cost-effective moving target defense against ddos and covert channel attacks," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, ser. MTD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 15–25. [Online]. Available: https://doi.org/10.1145/2995272.2995281

[45] R. Zhuang, S. Zhang, A. Bardas, S. A. DeLoach, X. Ou, and A. Singhal, "Investigating the application of moving target defenses to network security," in *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, 2013, pp. 162–169.

[46] N. O. Ahmed and B. Bhargava, "Mayflies: A moving target defense framework for distributed systems," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, ser. MTD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 59–64. [Online]. Available: https://doi.org/10.1145/2995272.2995283

[47] S. Sengupta, S. G. Vadlamudi, S. Kambhampati, A. Doupé, Z. Zhao, M. Taguinod, and G.-J. Ahn, "A game theoretic approach to strategy generation for moving target defense in web applications," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '17. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2017, p. 178–186.

[48] A. G. Bardas, S. C. Sundaramurthy, X. Ou, and S. A. DeLoach, "Mtd cbits: Moving target defense for cloud-based it systems," in *ESORICS*, 2017.

[49] P. Dasgupta and J. Collins, "A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks," *AI Magazine*, vol. 40, no. 2, pp. 31–43, 2019.

[50] Roy et al., "A moving target defense against adversarial machine learning," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, ser. SEC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 383–388. [Online]. Available: https://doi.org/10.1145/3318216.3363338

[51] Izmailov et al., “Combinatorial boosting of classifiers for moving target defense against adversarial evasion attacks,” in *Proceedings of the 8th ACM Workshop on Moving Target Defense*. New York, NY, USA: Association for Computing Machinery, 2021, p. 13–21. [Online]. Available: https://doi.org/10.1145/3474370.3485661

[52] R. K. Shahzad and N. Lavesson, “Comparative analysis of voting schemes for ensemble-based malware detection,” *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 1, pp. 98–117, 2013.

[53] Papernot et al., “The limitations of deep learning in adversarial settings,” in *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[54] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[55] Grosse et al., “Adversarial examples for malware detection,” in *European symposium on research in computer security*. Springer, 2017, pp. 62–79.

[56] D. Li, Q. Li, Y. Ye, and S. Xu, “A framework for enhancing deep neural networks against adversarial malware,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 736–750, 2021.

[57] P. Laskov *et al.*, “Practical evasion of a learning-based classifier: A case study,” in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 197–211.

[58] Y. Liu, X. Chen, C. Liu, and D. Song, “Delving into transferable adversarial examples and black-box attacks,” *arXiv preprint arXiv:1611.02770*, 2016.

[59] Grosse et al., “On the (statistical) detection of adversarial examples,” *arXiv preprint arXiv:1702.06280*, 2017.

[60] Moosavi-Dezfooli et al., “Universal adversarial perturbations,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[61] Labaca-Castro et al., “Universal adversarial perturbations for malware,” *arXiv preprint arXiv:2102.06747*, 2021.

[62] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, “Dos and don’ts of machine learning in computer security,” in *Proc. of the USENIX Security Symposium*, 2022.

[63] O. A. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[64] Y. Wang, M. Alhanahnah, X. Meng, K. Wang, M. Christodorescu, and S. Jha, “Robust learning against relational adversaries,” in *Advances in Neural Information Processing Systems*.

[65] Dauodi et al., “A deep dive inside drebin: An explorative analysis beyond android malware detection scores,” *ACM Trans. Priv. Secur.*, vol. 25, no. 2, may 2022. [Online]. Available: https://doi.org/10.1145/3503463

[66] Jordaney et al., “Transcend: Detecting concept drift in malware classification models,” in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 625–642. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney

[67] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, “Transcending transcend: Revisiting malware classification in the presence of concept drift,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 805–823.

[68] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, “Tesseract: Eliminating experimental bias in malware classification across space and time,” in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC’19. USA: USENIX Association, 2019, p. 729–746.

[69] Arp et al., “Drebin: Effective and explainable detection of android malware in your pocket.” in *Ndss*, vol. 14, 2014, pp. 23–26.

[70] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, “Androzoo: Collecting millions of android apps for the research community,” in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR ’16. New York, NY, USA: ACM, 2016, pp. 468–471. [Online]. Available: http://doi.acm.org/10.1145/2901739.2903508

[71] R. Thomas, “Lief - library to instrument executable formats,” https://lief.quarkslab.com/, April 2017.

[72] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu *et al.*, “Reviewer integration and performance measurement for malware detection,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 122–141.

[73] Demontis et al., “Yes, machine learning can be more secure! a case study on android malware detection,” *IEEE Transactions on Dependable and Secure Computing*, 2017.

[74] A. Li, T. Luo, T. Xiang, W. Huang, and L. Wang, “Few-shot learning with global class representations,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9715–9724.

[75] H. Yao, Y. Wei, J. Huang, and Z. Li, “Hierarchically structured meta-learning,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 7045–7054.

[76] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. v. Steen, “Prudent practices for designing malware experiments: Status quo and outlook,” in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 65–79.

[77] S. Y. Yerima and S. Sezer, “Droidfusion: A novel multilevel classifier fusion approach for android malware detection,” *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018.

[78] Pedregosa et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[79] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[80] Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. [Online]. Available: https://www.tensorflow.org/

[81] A. Kurakin, I. Goodfellow, S. Bengio *et al.*, “Adversarial examples in the physical world,” 2016.

[82] Z. Abaid, M. A. Kaafar, and S. Jha, “Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers,” in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, 2017, pp. 1–10.

[83] G. Severi, J. Meyer, S. Coull, and A. Oprea, “Explanation-guided backdoor poisoning attacks against malware classifiers,” in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[84] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Twenty-third annual computer security applications conference (ACSAC 2007)*. IEEE, 2007, pp. 421–430.

[85] Stokes et al., “Attack and defense of dynamic analysis-based, adversarial neural malware classification models,” *arXiv preprint arXiv:1712.05919*, 2017.

[86] D. Li, Q. Li, Y. F. Ye, and S. Xu, “Arms race in adversarial malware detection: A survey,” *ACM Comput. Surv.*, vol. 55, no. 1, nov 2021. [Online]. Available: https://doi.org/10.1145/3484491

[87] B. Wang and N. Z. Gong, “Stealing hyperparameters in machine learning,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 36–52.

[88] H. Li, S. Shan, E. Wenger, J. Zhang, H. Zheng, and B. Y. Zhao, “Blacklight: Scalable defense for neural networks against Query-Based Black-Box attacks,” in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2117–2134. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/li-huiying

[89] Demontis et al., “Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks,” in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 321–338.

[90] N. Papernot, P. McDaniel, and I. Goodfellow, “Transferability in machine learning: from phenomena to black-box attacks using adversarial samples,” *arXiv preprint arXiv:1605.07277*, 2016.

[91] Demetrio et al., “Functionality-preserving black-box optimization of adversarial windows malware,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3469–3478, 2021.

[92] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction apis,” in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 601–618.

[93] Biggio et al., “Evasion attacks against machine learning at test time,” in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[94] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[95] H. A. Alatwi and C. Morisset, “Adversarial machine learning in network intrusion detection domain: A systematic review,” *arXiv preprint arXiv:2112.03315*, 2021.

[96] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, “Patch-based defenses against web fingerprinting attacks,” in

Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, ser. AISec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 97–109. [Online]. Available: https://doi.org/10.1145/3474369.3486875

[97] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '19. IEEE Press, 2019, p. 810–822. [Online]. Available: https://doi.org/10.1109/ASE.2019.00080

[98] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating DNN-Based traffic analysis systems in Real-Time with blind adversarial perturbations," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2705–2722. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/nasr

[99] L. Schönherr, K. Kohls, S. Zeiler, T. Holz, and D. Kolossa, "Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding," *arXiv preprint arXiv:1808.05665*, 2018.

[100] H. Cai and J. Jenkins, "Towards sustainable android malware detection," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 350–351. [Online]. Available: https://doi.org/10.1145/3183440.3195004

[101] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for android," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 272–273.

[102] F. Maggi, W. Robertson, C. Kruegel, and G. Vigna, "Protecting a moving target: Addressing web application concept drift," in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 21–40. [Online]. Available: https://doi.org/10.1007/978-3-642-04342-0_2

[103] Imam et al., "A survey of attacks against twitter spam detectors in an adversarial environment," *Robotics*, vol. 8, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2218-6581/8/3/50

[104] He at al., "Adversarial example defense: Ensembles of weak defenses are not strong," in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[105] F. Ahmed, P. Vaishnavi, K. Eykholt, and A. Rahmati, "Ares: A system-oriented wargame framework for adversarial ml," in *2022 IEEE Security and Privacy Workshops (SPW)*, 2022, pp. 73–79.

[106] P. Martin, J. Fan, T. Kim, K. Vesey, and L. Greenwald, "Toward effective moving target defense against adversarial ai," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 2021, pp. 993–998.

[107] Shu et al., "Threat intelligence computing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1883–1898.

[108] Z. Zhu and T. Dumitras, "Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 458–472.

[109] Li et al., "Enhancing deep neural networks against adversarial malware examples," *arXiv preprint arXiv:2004.07919*, 2020.

Aqib Rashid Dr Aqib Rashid received his Ph.D. degree in Computer Science from King's College London. He has investigated methods to defend against adversarial attacks on ML-based malware detection. He is exploring this problem through his various research interests at the intersection of systems security and machine learning.

Jose Such Prof Jose Such is Professor in the Department of Informatics at King's College London and Director of the KCL Cybersecurity Centre. His research interests are at the intersection of artificial intelligence, human-computer interaction and cybersecurity, with a strong focus on human-centred AI security, ethics, and privacy. He has been Principal Investigator for the Discovering and Attesting Digital Discrimination, and Secure AI Assistants EPSRC projects.

Appendix A
Evaluated Defenses

Each defense has been evaluated using configuration and parameters as close as possible to its original paper.

Defense	Configuration/parameters/setup
DeepMTD [20]	$w = 0.3$, $n = 20$, $T = 0.6$
Morphence [19]	$n = 4$, $p = 3$, $Q_{max} = 1000$
MTDeep [12]	5 DNNs as constituent models. Assumed $\alpha = 1$
StratDef [17]	Variety-GT using same models as paper. Assumed strong attacker and $\alpha = 1$
Majority & Veto voting	Using same models as StratDef-Variety-GT
NN-AT	4 fully-connected layers (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax)). Adversarially-trained with up 25% size of training data

Appendix B
Architectures of Substitute Models (& Vanilla Models)

These model architectures are used for the substitute models in our transferability attack strategies. Additionally, we construct a set of vanilla models with these architectures that are used to generate a set of adversarial examples for Ensemble Adversarial Training (see Section 5).

Model	Parameters
Decision Tree	max_depth=5, min_samples_leaf=1
Neural Network	3 fully-connected layers (100 (Relu), 50 (Relu), 2 (Softmax))
Random Forest	max_depth=100
Support Vector Machine	LinearSVC with probability enabled

Appendix C
Permitted Perturbations for DREBIN and AndroZoo

DREBIN [69] and AndroZoo [70] are Android datasets, both of which can be divided into eight feature families comprised of extracted static features such as permissions, API calls, hardware requests, and URL requests. According to industry literature and prior work (e.g., [17], [27], [35], [56], [61], [82], [109]), features may be added or removed during attacks to traverse the decision boundary, based on the feature family.

However, malicious functionality must be preserved as a core constraint in this domain. As we operate in the feature-space, we offer a lower bound of functionality preservation. For example, attacks cannot remove features from the manifest file nor intent filter, and component names must be consistently named. Therefore, the table below enumerates the perturbations for each feature family that are allowed. For example, if a feature belonging to the S2 family is removed by an attack, then its original value is restored as it is not permitted to be removed (see Section 5).

	Feature families	Addition	Removal
manifest	S1 Hardware	✓	✗
	S2 Requested permissions	✓	✗
	S3 Application components	✓	✓
	S4 Intents	✓	✗
dexcode	S5 Restricted API Calls	✓	✓
	S6 Used permission	✗	✗
	S7 Suspicious API calls	✓	✓
	S8 Network addresses	✓	✓

TABLE 1: Permitted perturbations for Android datasets. These are determined by consulting industry documentation and prior work [17], [27], [35], [56], [61], [82], [109].

Appendix D
Effectiveness of Attacks Against Models With Fewer Features

We evaluate the performance of our gray-box query attack strategy against two vanilla neural network models to demonstrate the effectiveness of attacks when the feature-space is *drastically* reduced. That is, for each dataset, we train two vanilla neural network models: one with the full set of features (described in Section 5); and another with only 500 features. For the neural network with reduced features, the features are selected by using the `SelectKBest` function of the scikit-learn library (with the chi2 scoring function that computes the chi-squared stats between each non-negative feature and class). The table below shows that the attack performs just as well when the feature-space is significantly decreased for DREBIN and SLEIPNIR.

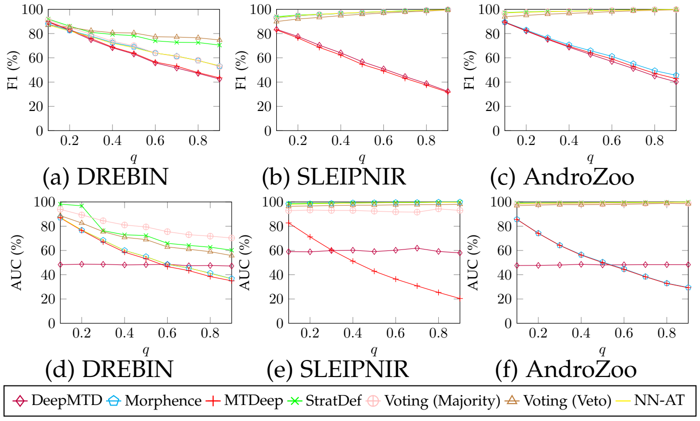
	500 features	**Full features**
DREBIN	100%	98.6%
SLEIPNIR	100%	99.5%

TABLE 2: Evasion rate achieved by gray-box query attack against vanilla neural networks with different numbers of features.

Appendix E
Extended Results

The extended results are located in the following anonymous repository: https://osf.io/nym5a/?view_only=4ba9b399086c4f7cadc65a6a4e8da83e

Appendix F
F1 and AUC (Section 8)



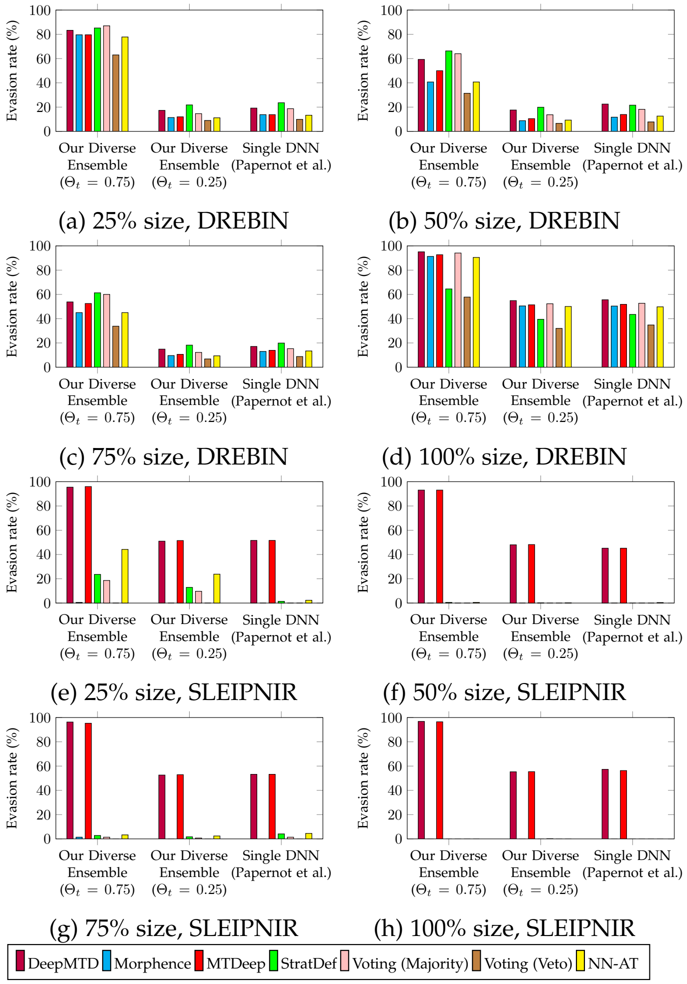
(a) DREBIN (b) SLEIPNIR (c) AndroZoo

(d) DREBIN (e) SLEIPNIR (f) AndroZoo

Fig. 16: F1 & AUC vs. q.

Appendix G
Varying Training Data Size in Gray-box Transferability Attack (Section 7.1)

We evaluate how the attack performs when the substitute models are trained with different sizes of training data (% of the original size). For DREBIN, the average evasion rate decreases between 25% to 75% sizes, after which it increases once 100% of the training data is used to construct substitute models. Meanwhile, for SLEIPNIR, we observe peak attack performance at 25% training data, with this decreasing as the training set size increases. The evasion rate against DeepMTD and MTDeep remains consistent, with only the performance against other models decreasing dramatically. This largely supports the idea that overfitting may be occurring. The disparity between the trends in DREBIN and SLEIPNIR can be attributed to the differences and nuances of each dataset relating to the different-sized attack surfaces of each. The black-box transferability attack still performs better. We primarily attribute this phenomenon to the suggestions made earlier in the paper: the substitute models capture the oracle's traits and behavior better, which means that adversarial examples for the black-box substitute models transfer better to the oracle.



(a) 25% size, DREBIN

(b) 50% size, DREBIN

(c) 75% size, DREBIN

(d) 100% size, DREBIN

(e) 25% size, SLEIPNIR

(f) 50% size, SLEIPNIR

(g) 75% size, SLEIPNIR

(h) 100% size, SLEIPNIR

Fig. 17: Evasion rate vs. sizes of training data.