Explaining and Measuring Functionalities of Malware Detectors

Wei Wang  
University of Adelaide  
Australia

Ruoxi Sun  
University of Adelaide  
Australia

Tian Dong  
Shanghai Jiao Tong University  
China

Shaofeng Li  
Shanghai Jiao Tong University  
China

Minhui Xue  
University of Adelaide  
Australia

Gareth Tyson  
Hong Kong University of Science and Technology  
Hong Kong, China

Haojin Zhu  
Shanghai Jiao Tong University  
China

ABSTRACT

Numerous open-source and commercial malware detectors are available. However, their efficacy is threatened by new adversarial attacks, whereby malware attempts to evade detection, e.g., by performing feature-space manipulation. In this work, we propose an explainability-guided and model-agnostic framework for measuring the ability of malware to evade detection. The framework introduces the concept of Accrued Malicious Magnitude (AMM) to identify which malware features should be manipulated to maximize the likelihood of evading detection. We then use this framework to test several state-of-the-art malware detectors’ ability to detect manipulated malware. We find that (i) commercial antivirus engines are vulnerable to AMM-guided manipulated samples; (ii) the ability of a manipulated malware generated using one detector to evade detection by another detector (i.e., transferability) depends on the overlap of features with large AMM values between the different detectors; and (iii) AMM values effectively measure the importance of features and explain the ability to evade detection. Our findings shed light on the weaknesses of current malware detectors, as well as how they can be improved.

1 INTRODUCTION

Malware continues to be one of the most pressing security issues that users face today. Recent research has shown that the total number of malware infections has been rising for the last decade (2009 to 2018) [6]. In 2018, the number of malware infections was 812.6 million, while during the first nine months of 2019, at least 7.2 billion malware attacks and 151.9 million ransomware attacks were reported. Thomas et al. [64] presents the risks of stolen credentials raised by malware, suggesting that 7–25% of exposed passwords match a victim’s Google account. Furthermore, the attack rate has hit a new high during the COVID-19 pandemic [13]. These figures suggest that traditional signature-based methods cannot keep up with the rampant growth of novel malware. Hence, commercial antivirus companies have started using machine learning [1, 63] to enable detection without the need for signatures. However, research has demonstrated that attackers can evade machine learning-based detectors by manipulating the malware features that such detectors use [31, 32, 50, 52, 68, 69]. Because of this, commercial antivirus systems are susceptible to adversarial attacks [58]. Although there has been several works [24, 34, 35] looking at adversarial attacks in computer vision (where adversaries change specific pixels), adversarial attacks on malware are far less understood.

For the purposes of this paper, we divide such attacks into two broad categories. The first group relies on problem-space obfuscation. Here we consider the problem space as a domain containing real-world objects (e.g., malware code, images, audio). Obfuscations in the problem-space change the semantic meanings of code snippets and further obfuscate the malicious signatures or patterns, thereby fooling rule-based malware detectors. Researchers have proposed a variety of such obfuscation techniques to generate adversarial malware that can evade detection by manipulating this domain [9, 11, 14, 16, 20, 41]. These include approaches such as hiding the control flow, inserting dummy code, and manipulating variable names.

The second group of adversarial attack relies on feature-space manipulation. This is performed on feature vectors that a detector induces from the problem-space. For example, a malware detector may induce a feature vector representing the control flow of malware code. This, however, means an attacker must know exactly how to change the problem-space (e.g., code) to result in a specific change to the feature space. Such attacks are becoming more prominent because machine learning-based detectors have reduced the efficacy of problem-space attacks. This occurs when a problem-space modification does not influence the projected feature space, thereby negating its impact on the malware detector.

Despite this, feature-space manipulation is more difficult than arbitrary code modification. This is because, after manipulating the feature space, it is necessary to map the modification onto the malware’s code. However, a single byte change can break the program or damage the malware’s original purpose. As a result, adversaries usually cannot directly modify the raw bytes of the program file. Instead, feature-space manipulation requires finding the correct action(s) on the problem-space that will influence the feature values (but without changing run-time functionality). These actions could be, for example, adding a redundant section (e.g., adding a new code section without linking its address in the section table) or injecting dead code that is unreachable (e.g., adding a file I/O request under an always-false condition, so that the dummy code will never be executed). Note, we are not the first to explore this topic. Similar techniques have been implemented by Demetrio et al. [30] in a
black-box optimization of adversarial Windows malware. However, they focus on the problem-space, instead of feature-space manipulation.

With the above challenge in mind, we focus on exploring how to guide feature-space manipulations and how to invert them back to the problem space. We do this with the explicit goal of evaluating functionalities of state-of-the-art malware detectors to identify malware with specific manipulations. Since most commercial malware detectors are not open-source, this must be done in a detector-agnostic manner (i.e., decoupling the attack strategy from the specifics of the detector). With this in mind, we design a detector-agnostic evasion attack which conducts feature-space manipulation and converting back to problem space to generate new adversarial sample binaries. We then evaluate it against various state-of-the-art malware detectors. In contrast to prior research, we further propose a novel method to explain the root cause of an attack’s ability to work across different detectors (i.e., its “transferability”). Our research helps security researchers to better understand evasion attacks and provides insights on how to improve malware defence strategies. The main contributions of this paper are three-fold:

- We propose an explainability-guided and model-agnostic malware detector measurement framework (§4). Our framework generates adversarial malware while preserving the malicious functionalities of the malware. We exploit SHapley Additive exPlanations (SHAP) and introduce the concept of Accrued Malicious Magnitude (AMM) to guide the feature selection approach for feature-space manipulation. We further project the manipulated feature back to problem-space with a binary builder that generates adversarial samples.

- We use AMM to measure the performance of state-of-the-art malware detectors protecting against adversarial attacks (§6.1). We show that commercial antivirus engines are vulnerable to AMM-based adversarial samples, while a detector with multiple different feature extraction functions reduces the impact of the adversarial attacks in a certain degree. Experimental results indicate that our approach has significant evasion capability, which decreases the detection rates of seven malware detectors by 56.47%, and bypasses an average of 25 out of the 60 antivirus engines in VirusTotal (VT). We also present the generalizability of our AMM approach by applying it on WinPE malware detectors (§6.3).

- We explain how manipulations trained on one detector can work on another detector (i.e., transferability) through our explainability-guided approach (§6.2). Our explainability-guided approach shows that the transferability relies on the overlaps of features with large AMM values between different machine learning models.

- We further explore the effectiveness of our proposed attack on improved machine learning-based detectors that exclude important features while training (§6.4). Results show that AMM values can effectively measure the importance of features and the capability of flipping classification results. We suggest that machine learning-based AV products should consider using the AMM values to improve their performance.

To the best of our knowledge, this is the first paper to systematically evaluate the weaknesses of malware detectors in a way that combines feature-space and problem-space with semantic explainability.

2 RELATED WORK & MOTIVATION

In this section, we introduce state-of-the-art research in malware detection and machine learning model explanation domains, followed by an motivating example.

2.1 Related Work and Background

Malware detectors. Many modern antivirus engines utilize rule-based analysis, such as signature matching, static unpacking, heuristics matching, and emulation techniques [7, 44]. However, rule-based antivirus engines rely heavily on expert knowledge. With the advantage of feature extraction derived from machine learning techniques, there is a flurry of work that integrates machine learning models into malware detectors [21, 43, 44, 66, 67]. We focus our evaluation on detectors that use static features due to their prevalence in providing pre-execution detection and prevention for many commercial endpoint protection solutions, such as Kaspersky [8], Avast [5], and ESET [3].

A few studies have explored the effect of obfuscations on anti-malware products, utilizing off-the-shelf tools. Maioreca et al. [50] and Pomilia [53] evaluated several anti-malware products using code obfuscation by a single tool. Hammad et al. [38] conducted a large-scale empirical study that evaluates the effectiveness of the top anti-malware products, including 7 open-source, academic, and commercial obfuscation tools. Several studies [25, 56, 61] have evaluated machine learning-based malware classifier models with the adversarial samples generated by generative adversarial networks (GANs) or automated poisoning attacks. Chen et al. [26, 27] studied machine learning classifiers with global robustness properties. Barbero et al. [22] proposed a method to cope with concept drift which may lead to performance degradation of malware detectors. Li et al. [47] conducted an empirical study to detect dataset shift and adversarial examples in Android malware detectors. Compared to the evaluation conducted in our paper, the scope of these studies only covers either the rule-based products or the machine learning-based models in isolation (rather than both).

Evasion attacks against malware detectors. The goal of the evasion attacks is to generate a small perturbation for a given malware sample that results in it being misclassified. This type of attack has been extensively explored in computer vision, and previous research efforts have also investigated the applicability of such techniques to malware classification. Xu et al. [68] proposed a genetic programming-based approach to perform a directed search for evasive variants for PDF malware. Demetrio et al. [29] demonstrated that genetic programming based adversarial attacks are applicable to portable executable (PE) malware classifier. Two recent works [18, 60] also applied deep reinforcement learning to generate adversarial samples for Windows PE malware to bypass machine learning models.

SHapley Additive exPlanations (SHAP). Research into explainable machine learning has proposed multiple systems to interpret the predictions of complex models. In this paper, we rely on SHAP [48] (based on the coalitional game theory concept of Shapley values). Hence, we briefly describe its operation. The SHAP
framework subsumes several earlier model explanation techniques together, including LIME [54] and Integrated Gradients [62]. SHAP has the objective of explaining the final value of a prediction by attributing a value to each feature based on its contribution to the final result. To accomplish this task, the SHAP frameworks train a surrogate linear explanation model $g$ of the form:

$$f(x) = g(x'),$$

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j,$$

(1)

where $f$ is the original model, $x$ is the input sample to be attributed, $x'$ is the coalition vector of $x$. For each entry of $x'$, its value is 1 if the corresponding feature is "present" and 0 if "absent". $\phi_0 = \mathbb{E}_X (f(X))$ is the average prediction of the original model on sampled dataset $X$. The Shapley value $\phi_j \in \mathbb{R}$ is the feature attribution for the $j^{th}$ feature $x'_j$ to the model’s decision. Summing the effects of all feature attributions approximates the difference of prediction for $x$ and the average of the original model. Further, the SHAP framework connects LIME and Shapley values to fit Equation 1 and explains any machine learning-based model without internal knowledge.

LIME uses a linear explanation model $g(x')$ to locally approximate the original model, where locality is measured in the simplified binary input space, i.e., $x' \in \{0, 1\}^M$. To find $\phi$, LIME minimizes the following objective function:

$$\xi = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g),$$

(2)

where $L$ is the squared loss over a set of samples in the simplified input space weighted by the kernel function $\pi_x$, and $\Omega$ penalizes the complexity of $g \in G$ where $G$ is hypothesis space. Therefore, based on the input feature vectors and the output predictions of the model, we can use the model’s coefficients to approximate the importance of each feature.

2.2 Motivating Example

To motivate our adversarial attacks, we analyze a sequence of source code decoded from an Android malware, which is tagged as malicious by 39/67 detectors from VirusTotal (VT) [15]. From the source, we find a snippet of malicious code shown in the top part of Figure 1. As shown in lines 3 and 4, the malware executes a native scripts via root permissions by `su -c ./script1` command.

In order to bypass machine learning-based detectors, we must perturb its API-call-based feature space towards 'benign'. Hence, we insert several function calls with always-false condition closure (e.g., `time<0`) to ensure they are unreachable during run-time, preserving the original (malicious) functionality. These function calls are randomly selected from benign features, which involve a list of function calls extracted from benign apps, provided by an Android dataset, Drebin [21]. The inserted code is marked as blue in the middle part of Figure 1. After rebuilding the source code, the modified binary is identified by 33 scanners ~ 6 fewer than the originally, and it bypasses the machine-learning detector provided by Drebin. The remainder of this paper develops an explainability-guided feature-space manipulation framework and problem-space rebuilding tool.

3 THREAT MODEL & PROBLEM DEFINITION

In this section, we define the threat model and take a deep dive into our research problem.

3.1 Threat Model

We follow the methodology by Carlini et al. [23] and describe the threat model of evasion attacks against malware detectors from three aspects: the adversary’s goals, capabilities, and knowledge.

Adversary goal. The adversary’s goal is to manipulate malware samples to evade the detection of malware detectors, including white-box, grey-box and black-box detectors. The type of malware we consider in this study is Android APKs. We will also discuss Windows Portable Executive (WinPE) malware. In the evaluation, we only use binary detectors which determine if the software under test is benign or malicious. The goal of attackers in this work is to cause the malicious samples to be misclassified as benign.

Adversary capability. We assume that the adversary does not have access to the training phase or the model of the machine learning-based detectors. For instance, the adversary cannot inject poisoned data in the training dataset or manipulate any code or output of detectors. However, they will still have some basic knowledge about machine learning-based detectors, e.g., the access to open-source datasets, features extraction methods [55], or off-the-shelf machine-learning detectors. In addition, we introduce black-box detectors whose feature extraction functions and internal architecture are kept unknown to the adversary.

Adversary knowledge. In this work, we assume that an attacker has full knowledge of one machine learning-based malware detector, including its feature extraction functions, architecture, and training dataset. Such a white-box model will be used as the source of adversarial sample generator. For the target detectors, the adversary has no knowledge about the detectors’ training dataset, inner structure, or detection mechanism. We will evaluate the target detectors in two scenarios, i.e., the attacker knows (grey-box) or does not know the feature extraction method (black-box).
3.2 Problem Definition

Our goal is to evaluate the efficacy of evasion attacks against malware detectors using generated adversarial samples. Considering a malware detector mapping a software sample \( x \in X \) to a classification label \( l \in \{0, 1\} \) (where 0 represents benign and 1 represents malicious), the goal of evasion attack can be summarized as:

\[
F(x) = 1, \quad x_a = \text{Gen}(x), \quad F(x_a) = 0, \quad (3)
\]

where \( F \) could be either a trained machine learning model or an antivirus engine. \( x \) is the original malware sample, and \( \text{Gen} \) is the sample generator that is able to generate adversarial sample \( x_a \) while keeping its malware functionality the same as \( x \).

To ensure the reproducibility and coverage of our evaluation, we have several criteria on the selection of evasion attack and adversarial sample generation strategies:

- **Easy-to-obtain.** We only utilize open-source and off-the-shelf tools, instead of proposing any new attack technique ourselves.
- **Compatible.** We combine multiple evasion attacks, which we believe will put greater stress on the detectors and make the evaluation as comprehensive as possible.
- **Explainable.** An explainable approach is preferred as it will help us to analyze the evaluation results and find out potential weaknesses in malware detectors.

To establish such an evaluation strategy, we consider generating adversarial samples through perturbation the feature space (e.g., manipulating values in feature vectors) and converting the manipulation back to the problem space (e.g., modifying malware source code and rebuild the binary). To achieve our goal, the problem can be split into two sub-problems: (i) generating adversarial samples and (ii) evaluating them against malware detectors, both of which are detailed in §4.

3.3 Ethical Considerations

Our research is concentrated on the defence scope that explains the adversarial evasion attacks and determines the potential weaknesses of current malware detection methodologies. Hence, we declare: (i) the motivating example we presented is only a code snippet without actual functionality; (ii) all tools and datasets involved in our experiment are publicly available, and we also anonymize the antivirus engines with a simple serial number label; (iii) considering the potential security issues, we will not release the source code of the proposed attack and any adversarial samples, as well as the information of commercial antivirus involved in our evaluation, except for academic uses approved by our ethical committee.

4 METHODOLOGY

Our research methodology consists of three key components (see Figure 2): (i) explainability-guided feature selection, to select the feature manipulation; (ii) an adversarial sample generator, to generate the evasive samples; and (iii) an evasion attack evaluation, to evaluate the proposed evasion attacks against four different machine-learning detectors, three antivirus engines and VirusTotal [15], to explain their transferability and the impact of important features on the accuracy. Note, transferability refers to the performance of evasion attacks when generating adversarial samples for one machine-learning model and then applying them to different detectors; important features refer to a set of features that a machine-learning model depends on the most.

4.1 Step 1: Feature Selection

In the first step of our methodology, we utilize SHapley Additive exPlanations (SHAP) to create an explainability-guided adversarial example. SHAP calculates how much one feature contributes to an individual prediction. In this step we generate SHAP values of the input dataset. The workflow of the explainability-guided feature selection is illustrated in Algorithm 1. For a set of seed malware, \( S \), we aim to generate a corresponding adversarial sample set, \( A \), such that they evade the target model only by modifying features.
Pre-processing. We extract features from the training samples $X$ of a trained machine learning model $m$ (line 2). Then the vectorized samples, $X'$, and the model are input to $\text{shap}()$ to calculate the SHAP value matrix $M$ (line 3). The matrix is then used to select the most evasive features and the most benign-oriented values.

Feature selection. To select the feature that has the largest malicious magnitude, we propose the concept of Accrued Malicious Magnitude (AMM). The AMM is defined as the product of the magnitude of SHAP values in each feature and the number of samples that have malicious-oriented values in the corresponding feature. By calculating AMM values, we select the feature that has the largest modifiable capability and has the most samples to be modified as the adversarial examples. Specifically, starting from the getRange($M$) line 5, we first calculate the range of SHAP values in each feature and store the results in a one-dimension vector $D$. $D$ indicates the potential magnitude we can modify on each feature. i.e., each $d_i \in D$ presents the difference between the maximum SHAP value and the minimum SHAP value of feature $f_i$. Next, for each feature, we count how many samples have SHAP value larger than the mean SHAP value of that feature (the countLarge($M$) in line 6). Note that in our experiments, we labeled malicious as 1 and benign as 0. Therefore, a larger $c_i \in C$ means that, for feature $f_i$, there are more samples that have a SHAP value towards malicious, such that more samples can be manipulated towards benign. Therefore, we select the most evasive feature according to the AMM values, denoting the dot product of the range of SHAP values ($D$) and the number of SHAP values greater than mean ($C$) (line 7).

Value selection. Once we have identified the feature $f$ to compromise, the next step is to choose the value for the selected feature to guide the manipulation. Then we select the most benign-oriented value, $v$, in the problem space. This corresponds to the most negative value in $M[f]$, the SHAP values of feature $f$ (line 9).

Update feature patch. After obtaining a pair of $(f, v)$, if the selected feature $f$ is manipulable, we add the pair into map $P$ as the Feature Patch to be used in the feature-space manipulation (line 11). Although the SHAP framework can find features that impact the decision boundary, some of them cannot be manipulated directly. For example, consider the feature that counts the size of a binary, when we modify the value of another feature, the former will be modified indirectly. Therefore, the features and values we select to be manipulated follow two principles employed by the previous literature [36, 37, 55]. These principles are: (i) features are manipulable in the original problem space; and (ii) selected features have no dependencies or cannot be affected by other features.

Greedy strategy. After obtaining feature-value pairs, we conduct a greedy strategy, removing samples that have the same value, $v$, for feature $f$ from the dataset (lines 12 to 16). We do this to make sure that the same feature-value pair will not be selected again. The procedure repeats until we find $N$ feature-value pairs. These $N$ pairs are then used in the next stage to generate the adversarial malware samples.

### Algorithm 1: AMM-based Feature-Space Selection

| Line | Description |
|------|-------------|
| 1    | $P = \text{map}([\text{Feature}, \text{Value}])$; |
| 2    | $X' \leftarrow \text{vectorize}(X)$; |
| 3    | $M \leftarrow \text{shap}(X', m)$; |
| 4    | **while** $|P| < N$ **do** |
| 5    | $D \leftarrow \text{getRange}(M)$; |
| 6    | $C \leftarrow \text{countLarge}(M)$; |
| 7    | $\text{AMM} \leftarrow D \cdot C$; |
| 8    | $f \leftarrow \text{arg max}(\text{AMM})$; |
| 9    | $v \leftarrow \text{arg min}(M[f])$; |
| 10   | **if** $\text{isManipulatable}(f)$ **then** |
| 11   | $P \leftarrow P \cup (f, v)$; |
| 12   | **for** each $x' \in X'$ **do** |
| 13   | **if** $x'[f] \neq v$ **then** |
| 14   | $idx \leftarrow \text{getIndex}(X', x')$; |
| 15   | $M \leftarrow M \setminus M[idx]$; |
| 16   | $X' \leftarrow X' \setminus x'$; |
| 17   | **return** $P$; |

### 4.2 Step 2: Adversarial Sample Generator

In the next step, the adversarial sample generator applies features manipulation from previous steps and generate adversarial samples. Explainability-guided feature-space manipulation involves changing features selected by Algorithm 1 to mislead the detector. Feature-space manipulation. To train a machine learning model, the first step is to convert input data into vectors of features (i.e., the feature extraction process). In an evasion attack, we manipulate features to induce misclassifications. However, not every feature has equal influence on the result of the detector, so the question becomes: how can we gain insight into a model’s decision in a generic, model-agnostic way? Thus, we rely on SHAP to understand which features drive the model towards a benign classification. Guided by SHAP, we can manipulate the malware sample and cause a misclassification. Importantly, we must ensure the malware functionality is preserved. Equation 4 summarizes the feature-space manipulation:

$$ x' = \text{vectorize}(x), $$

$$ a' = \text{manipulateFeature}(x', P), $$

$$ \text{Gen}'(x) = \text{buildSample}(a', x), $$

where $x'$ is the result of applying feature extraction on sample $x$ using $\text{vectorize}()$. The SHAP value matrix, $M$, is obtained through $\text{shap}()$, the SHAP algorithm. $m$ represents the machine learning model. $\text{manipulateFeature}()$ manipulates the sample in feature-space guided by SHAP. Note that, the sample generator $\text{Gen}'()$ will take the manipulated feature-space sample $a'$ and the original sample $x$ as input, and implement the changes in feature-space back to problem-space to generate the adversarial sample, while keeping its malware functionality.

Note that, due to the strong semantic restrictions of the binaries, we cannot simply choose any arbitrary pairs of feature and values.
for our evasion attack. Instead, we restrict the feature-space manipulation to only features and values that are independent and can be modified with original functionalities preserved. Therefore, we design a binary builder to implement the inverting of features, and mapping the manipulation back to the problem space.

**Binary builder.** In order to evaluate adversarial samples on malware detectors and AV engines, feature-space manipulation needs to be applied to problem-space binaries. To ensure that no loss of functionality is inadvertently introduced as a side effect of feature manipulation, we only apply these changes to unreachable area of binaries so that these changes will never be executed during run-time. Then, we apply these changes on seed binaries with the help of open-source binary builders.

We take an Android APK as an example. Since features are a vector of boolean values representing the existence of a feature, proposed by Drebin [21], the feature value could only be modified from 0 (absence) to 1 (presence) to preserve original functionalities. We first leverage Apktool [4] to decompose an APK file into smali [12] code, a structured assembly language. API calls and network URLs are transformed to smali instruction code, which is wrapped by an unreachable disclosure, e.g., an always-false condition closure. The smali code is then inserted into the small file of the main activity. Features representing Android manifest components are inserted into AndroidManifest.xml file directly. Finally, we utilize Apktool to assemble all decompiled and manipulated files into an adversarial APK sample. If a feature manipulation cannot be implemented in this way, we skip it and continue with the next most important feature.

### 4.3 Step 3: Malware Detector Evaluation

After the adversarial samples are generated, we conduct a series of evaluations aiming to explain and measure the functionalities of malware detectors.

**Evaluation of detector performance.** The method of detector performance is straightforward. We first input the seed malware into each detector under test and collect their detection rate of malware as baselines. Next, the adversarial samples generated from the white-box model will be input to the detectors (including the white-box model itself). We then compare difference on the detection rate of seed and adversarial samples to measure whether the detector is vulnerable to adversarial samples, i.e., whether the manipulation on AMM features leads to the flipping of detector results.

**Transferability analysis.** Considering that machine learning-based detectors may use a same or similar feature extraction method, it is possible that multiple detectors focus on the same features; further, it is also likely that, when different feature extraction methods are used or even when the detectors only use problem-space information, different detectors may still rely on features overlapped with each other. Therefore, we assume that such overlapping may exist among detectors and let the evasion attacks transfer from the generation model to other models, i.e., transferability.

Specially, in machine learning, the adversarial examples generated from one machine learning model are very likely to remain effective on other models that are trained on the same data distribution due to the similarity of decision boundaries. More powerful transferability an adversarial sample has, more effective the adversarial samples are in other different detectors. To evaluate the transferability of adversarial samples generated by our AMM-based approach, we will generate samples from different models and apply them onto different models. If such transferability exists, we will further investigate the feature-space overlaps among models to identify the root cause of transferability.

**Evaluation of improved detectors.** Inspired by recent research [40], a detection model can be improved by removing important features from the training phase, where the important features refer to the ones that are sensitive to the model prediction accuracy. Shapley Additive Global importance [28] (SAGE) is a framework that measures how much a feature contributes to the prediction accuracy of a model (see more details in Appendix D). We utilize SAGE to improve the detection models and evaluate their performance against adversarial samples. Specifically, we first calculate SAGE values of each feature. Since the most important features are the ones with largest SAGE values, we sort the features by SAGE values in a descending order and select the top features. Then we remove the top features from samples and generate a new training set. Finally, an improved model $m'$ is trained with the new training set. To establish a thorough comparison, we train another model that excludes top AMM-based features to compare and explain the performance of improved models.

## 5 EXPERIMENT SETUP

In this section, we describe the setup of our experiment including the experiment environment, the datasets and model training, how we extract features, how we implement the binary builder, and how we obfuscate the sample.

### 5.1 Experimental Environment

Our experimental environment is a PC workstation with 64GB RAM, AMD Ryzen 3750X 8-core CPU and Linux Mint 20.1 Cinnamon installed with 256GB swap partition. The script running environment is Python 3.9.9.

### 5.2 Target Detectors

During the evaluation, we perform the evaluation of malware detectors using the proposed AMM approach, including 4 machine learning-based detectors, 3 open-sourced commercial antivirus engines, and 60 antivirus engines available at VirusTotal [15]. The detectors are summarized in Table 1.

To follow the convention of the prior studies [21, 55], we select 4 off-the-shelf machine learning-based malware detectors.

- **LightGBM (LGBM)** [42] is a free and open source distributed gradient boosting framework, based on the decision tree algorithm, originally developed by Microsoft.
- **Support Vector Machine (SVM)** is a set of supervised learning methods used for classification, regression and outliers detection.
- **Random Forests (RF)** is an ensemble learning method that combines a multitude of decision trees to provide classification.
- **Deep Neural Network (DNN)**. In addition, we introduce a simple-structured DNN with one input layer and three fully-connected hidden layers that followed by ReLU activation function (the last one ends with a Softmax function).
whether a feature exists in an application or not. Features are categorized to 8 logical subsets representing hardware components, requested permissions, app components, filtered intents, restricted API calls, permissions used in the source codes, suspicious API calls, and network addresses, respectively. Since the ratio of malicious and benign apps is biased, we randomly select 5,560 benign samples but a minor amount among malicious samples; and (ii) set as 1 in a significant amount in malicious samples but a minority of benign samples. Features in the former represent benign-oriented features while those in the latter represent malicious-oriented features. The detailed algorithm can be found in Appendix §C.

Basic Iterative Method [45] (BIM) and C&W [24] are two white-box setting adversarial attacks that require to access the gradients of machine learning models. They have outstanding performance of adversarial attacks in computer vision. However, they are not capable in the malware detection domain, because adversarial samples cannot be generated by perturbing each feature arbitrarily with noise, and thus the objective function of those attacks may not successfully converge. In addition, our evaluation aims to generate runnable adversarial malware samples that are effective on both machine learning-based detectors and antivirus engines. However, for antivirus engines, the attacker can only query the models without accessing the gradients of models. Therefore, we exclude BIM and C&W attacks from the evaluation.

### 6 EVALUATION & RESULTS

We next employ our pipeline to evaluate the efficacy of our evasion techniques, as well as test the susceptibility of the detectors to adversarial attacks. Specifically, we compare the detection rates between original samples and the adversarial samples generated by our proposed strategy.

According to the experiment in Appendix A, the number of AMM-based features to manipulate is 75. However, the distribution of each feature set is not balanced, i.e., \( S_3 \) has the largest amount of patching features as shown in Table 2. Therefore, to compare the performance of proposed evasion attacks, we introduce two different feature selection functions and generate three types of adversarial samples: (i) patching balanced features from the manifest file and the rest four from the Dalvik Executable (dex) file, and then these features are used to train and evaluate the machine learning based detectors mentioned above.

### 5.4 Benchmark Methods

In order to evaluate the performance of evasion attacks guided by AMM values, we introduce an intuitive statistics-based feature selection strategy according to the APK dataset in §5.3. In this feature selection strategy, we aim to find features (i) set as 1 in a major amount among benign samples but a minor amount among malicious samples; and (ii) set as 1 in a significant amount in malicious samples but a minority of benign samples. Features in the former represent benign-oriented features while those in the latter represent malicious-oriented features. The detailed algorithm can be found in Appendix §C.

| Name  | Type         | Description                                                                 |
|-------|--------------|-----------------------------------------------------------------------------|
| LGBM  | White-box    | LightGBM, a tree-based classifier.                                           |
| SVM   | Grey-box     | A linear support vector machine classifier.                                 |
| RF    | Grey-box     | A random forest classifier.                                                 |
| DNN   | Black-box    | A feed-forward neural network with 3 hidden layers.                         |
| AV1   | Black-box    | An open-source antivirus engine.                                            |
| AV2   | Black-box    | A commercial antivirus engine.                                              |
| AV3   | Black-box    | A commercial antivirus engine.                                              |
| VT    | Black-box    | VirusTotal, a free online service that integrates over 70 antivirus detectors. |

* Only 7 \( S_1 \) features are found from top 1000 AMM features.

### Table 1: Target detectors

| Name   | Type         | Description                                                                 |
|--------|--------------|-----------------------------------------------------------------------------|
| LGBM   | White-box    | LightGBM, a tree-based classifier.                                           |
| SVM    | Grey-box     | A linear support vector machine classifier.                                 |
| RF     | Grey-box     | A random forest classifier.                                                 |
| DNN    | Black-box    | A feed-forward neural network with 3 hidden layers.                         |
| AV1    | Black-box    | An open-source antivirus engine.                                            |
| AV2    | Black-box    | A commercial antivirus engine.                                              |
| AV3    | Black-box    | A commercial antivirus engine.                                              |
| VT     | Black-box    | VirusTotal, a free online service that integrates over 70 antivirus detectors. |

### Table 2: Selected feature numbers of three strategies.

| Sets Descriptions | AMM (Default) | AMM (Balanced) | Statistics |
|-------------------|---------------|----------------|-------------|
| \( S_1 \) Hardware Components | 1 | 7* | 1 |
| \( S_2 \) Requested Permissions | 15 | 10 | 3 |
| \( S_3 \) App Components | 5 | 10 | 94 |
| \( S_4 \) Filtered Intents | 10 | 10 | 11 |
| \( S_5 \) Restricted API calls | 7 | 10 | 5 |
| \( S_6 \) Used Permissions | 4 | 10 | 2 |
| \( S_7 \) Suspicious API calls | 10 | 10 | 0 |
| \( S_8 \) Network Addresses | 23 | 10 | 183 |

* Only 7 \( S_1 \) features are found from top 1000 AMM features.

In our threat model, the adversary has full knowledge to one machine learning-based detector. Here we set LGBM as this white-box detector. Note that any machine learning-based detector could serve this role as our approach is model-agnostic.

We utilize the feature extraction function from Drebin [21] to train LGBM, SVM and RF, and we refer to the feature extraction function from recent work [51] to train DNN. As the adversary knows the feature extraction function of LGBM, the SVM and RF match the grey-box scenario and the DNN model fits the black-box scenario.

For antivirus engines, AV1 is an open-source antivirus engine while AV2 and AV3 are commercial antivirus products. VirusTotal is an online service that provides over 70 antivirus scanners to detect malicious files and URLs. Our experiment found that 60 scanners are always available in malware detection while others are not stable, i.e., sometimes available and sometimes not. Therefore, we leverage these 60 scanners as a benchmark. All antivirus engines fall into the black-box scenario as the attacker has no specific knowledge about them.

### 5.3 Datasets and Model Training

In our experiments, we conduct evaluations on Android dataset and machine learning-based detectors. The Android Application Package (APK) is the package file format used by the Android operating system for distribution of mobile apps. We use the well-studied Drebin [21] dataset which contains features extracted from 5,560 malicious and 123,453 benign samples. These features are represented by over 545,000 dimension Boolean vectors indicating whether a feature exists in an application or not. Features are categorized to 8 logical subsets representing hardware components, requested permissions, app components, filtered intents, restricted API calls, permissions used in the source codes, suspicious API calls, and network addresses, respectively. Since the ratio of malicious and benign apps is biased, we randomly select 5,560 benign samples, making up 11,120 samples with 76,889 feature dimension, to balance the dataset. Further, we create a random 50:20:30 split of samples for training, validation, and testing set respectively to train a LightGBM model. We will discuss different models with the same dataset in §6.2. In our experiment, we evaluate adversarial samples on four machine learning models, shown in Table 1.

To train ML models mentioned above, we employ Androguard [2] to extract raw features from APK samples. Androguard is a python tool to analyze and manipulate Android files. It disassembles an APK file and converts its byte code and resource files into a readable and structured format. We further extract the first four types of features from the manifest file and the rest four from the Dalvik Executable (dex) file, and then these features are used to train and evaluate the machine learning based detectors mentioned above.
We start by evaluating our proposed evasion attack with three methods: (i) patching 300 statistics-based features (i.e., Statistics-based strategy); (ii) patching 300 AMM-based features (i.e., AMM-based strategy); and (iii) patching 300 statistics-based features (i.e., Statistics-based strategy).

Note that three strategies have different time consumption of feature selection. In our experiment, both default and balanced AMM-based strategies consumed around 6 hours in generating a SHAP value matrix from 11,200 samples with 76,889 features. After generating the SHAP matrix, both strategies took about 30 seconds to select features. Meanwhile, the statistics-based strategy only consumed around 10 seconds to select features. This is because the AMM-based strategy requires complex matrix computation and iteration while the statistics-based strategy simply sums up the feature matrix and iterates the value list once.

6.1 Evaluation of Detectors

We start by evaluating our proposed evasion attack with three strategies against four machine learning-based detectors and VirusTotal (VT). As our performance metric, we focus on the difference in detection rates between the original samples and the adversarial samples. Figure 3 illustrates the detection rates of each strategy against each detector. The y-axis refers to the detection rates of samples, whereas each detector is presented on the x-axis. In our evaluation, we generate APK adversarial samples from the LGBM model with three strategies, listed in Table 2, and evaluate adversarial samples on four machine learning-based detectors, three antivirus engines and VT. All four machine learning-based detectors have reasonable accuracy (above 90%). Note that the detection rate of VT in Figure 3 represents the rate of how many detectors on average mark one sample as malicious out of 60 detectors in VT.

Performance against AMM-based Strategy. This strategy selects 75 features purely based on AMM values. Comparing with the detection rate of seed malware, adversarial samples have a significant evasion performance on LGBM, SVM and RF, with detection rates of 14.36%, 0.00% and 45.67%. However, DNN, as a black-box detector with an absolutely different feature extraction function, detected 90.12% adversarial samples. Comparing with seed malware detection rate, the DNN detector is not remarkably impacted by the adversarial attack with different feature extraction function.

Performance against Balanced AMM-based Strategy. The default AMM-based strategy selects an unbalanced amount of features in eight feature sets. Table 2 shows that very few features in $S_1$, $S_3$ and $S_9$ are selected. Therefore, we choose the top 10 features of each set from the top 1000 AMM features to compare the performance of AMM features with balanced number in different sets. However, only 7 features were found from $S_1$, making up 73 features in total.

Note that we evaluated samples with simply reordering the items in their AndroidManifest.xml files, and found that their scores decreased by 6 by average. It means that many antivirus detectors do not exactly focus on malicious behaviours of input samples.

Summary. From the experimental results above, both white-box (LGBM) and grey-box (SVM and RF) detectors can be sufficiently evaded by adversarial samples from all three strategies. In contrast, the black-box (DNN) detector is not impacted by feature manipulation due to different feature extraction functions. This indicates...
6.2 Transferability Analysis

Transferability is the ability for an attack to be effective against multiple learning-based detectors. To study this, we next generate AMM-based adversarial samples from SVM and RF and evaluate them on LGBM, SVM, and RF, which have the same feature extraction functions. Here, we seek to understand how a manipulation guided by one detector performs against the other detectors. To unveil what causes the transferability, we evaluate LGBM, SVM, and RF, since they share the same feature extraction functions. We observe the earlier results in Figure 3 and find that the AMM-guided approaches have a notable performance of evading detection across LGBM, SVM, and RF. Recall that, according to Algorithm 1, we select the features with highest AMM values as the most evasive features to conduct feature-space manipulation. In this section, we analyze how the attack transfer to other detectors in two aspects: feature overlaps and detection rates.

Feature overlap. To explore the reason why our proposed attack can transfer across detectors, we present the AMM values of the top features across each dataset in a heatmap, shown in Figure 4. As the number of features differs across each dataset, we only display the top 1024 features that have the highest AMM values in LGBM, the generation model. In each subplot, we present the 1024 features as 32 rows by 32 columns of dots (normalized to $[0, 1]$), where the darker dots indicate higher values of AMM (which indicates a greater possibility to be selected a feature to be manipulated). Further, we sort the features in the original dataset according to the AMM values in descending order. Therefore, darker dots scattered in the upper zone of the nine subplots indicate that there are more features having been selected across detectors.

From the heatmap we can observe that (i) large AMM values of LGBM (dark dots) overlap with most of the counterparts of SVM; (ii) many large AMM values of RF are out of the scope of the counterparts of LGBM. The overlaps resonates with the main study results — the transferability from LGBM to SVM outperforms the transferability from LGBM to RF. Thus, the overlaps explains why the evasion attack can transfer across learning-based detectors. Simply put, if we manipulate enough features across different learning-based models (i.e., feature overlaps), the evasion attack can be transferred.

Detection rate. We can also evaluate transferability of adversarial samples generated by inspecting the detection rates. In this experiment, we generated default AMM-based adversarial samples generated from LGBM, SVM, and RF models. Figure 5 shows the detection rates of adversarial samples across machine learning detectors. The y-axis represents each dataset with three generating models, and the x-axis shows the target detectors. In the figure, a darker color indicates a higher detection rate (representing lower transferability).

Figure 5 shows that adversarial samples generated from LGBM and RF can effectively transfer the evasion attacks across three detectors. Adversarial samples from SVM have poor performance on the RF detector. Overall the SVM detector has the worst detection performance on adversarial samples (i.e., 0.00% of adversarial samples are detected), while the RF detector performs the best.

Takeaway 2: The evasion attack transferability depends on the overlaps of features with large Accrued Malicious Magnitude (AMM) values between different learning-based models.

6.3 Generalizability Analysis

To examine if our evasion attack can generalize to other operating systems, we next test how effective our evasion attack is on Windows Portable Executable (WinPE).

WinPE. The Portable Executable (PE) format is the standard file format for executables, object code, and Dynamic Link Libraries (DLLs) used in 32- and 64-bit versions of the Windows operating systems. We use SOREL-20M [39] as the WinPE dataset in our experiment. SOREL-20M is a representative public dataset of malicious and benign WinPE samples used for malware classification, consisting of 2,381-dimensional feature vectors extracted from 9,470,626 benign and 9,919,251 malicious samples, as well as corresponding malicious binaries. It leverages the feature extraction function from Ember [19] and provides larger dataset. We randomly choose 10,000 benign and 10,000 malicious samples to train LGBM, SVM, RF and DNN with the same feature extraction function.

Feature manipulation. Previous work [55] shows that only 17 features can be modified directly and indirectly to preserve the
We next seek to build on the lessons learnt above, to expand our important features on the original model and two updated model guided by AMM values. Therefore, we seek to improve the performance of adversarial samples detected by models excluding different amount of important features.

Figure 6: Detection rates of seed malware and AMM-based adversarial samples detected by models excluding different amount of important features.

![Figure 6: Detection rates of seed malware and AMM-based adversarial samples detected by models excluding different amount of important features.](image)

Figure 7: Precisions and detection rates of seed malware $S_m$, adversarial samples $S_a$ (AMM-based features) and adversarial samples $S_i$ (important features) on the original model and two updated model $m_1$ and $m_2$.

Results. In the experiment, we only evaluate the default AMM-based strategy. This is because the WinPE has fewer features to manipulate, and its feature structures are different from APK features in Drebin. The results are shown in Figure 13. Our proposed strategy has a remarkable evasion performance on LGBM, SVM and RF detectors. Since most WinPE features correlate with each other, the method of parsing and generating WinPE binaries (i.e., directly modifying values and adding empty sections) may negatively affect the performance of our proposed attack, illustrated in DNN. In a nutshell, the test result shows that our proposed evasion attack is also effective on Windows.

6.4 Revisiting AMM with Improved Detectors

We next seek to build on the lessons learnt above, to expand our attack. We first show how we can improve detector performance before, in turn, exploring how adversarial attacks could be improved. In addition, we compare the capability of improving detectors between AMM and SAGE.

Improving detectors. Our evaluation shows that machine learning-based detectors are vulnerable to adversarial attacks guided by AMM values. Therefore, we seek to improve the performance of existing detector so that adversarial samples can be identified. We follow the methodology in §4.3 and generate an improved LGBM detector $m_1$ by excluding SAGE-based important features in the training phase. The improved model $m_1$ aims to evaluate AMM-based adversarial samples $S_a$. In order to measure the effectiveness of AMM-based and SAGE-based features, we also generate another improved detector $m_2$ by excluding AMM-based features and generate adversarial samples $S_i$ by applying SAGE-based important features.

To understand how many important features can noticeably improve the detection ability against adversarial samples, we generated new machine learning detectors with different number of important features excluded from the training set. The result is shown in Figure 6. The y-axis refers to the model accuracy and detection rates of seed malware and AMM-based samples, whereas the numbers of features excluded from models are presented on the x-axis. From the result, we found that excluding 220 important features allows the new detector to perform similar detection rates of original malware (80.56%) and adversarial samples (80.56%). Therefore, we employ 220 important features in the following experiment.

Measuring detectors. Considering the ability for detectors to improve their accuracy by removing features, we next experiment with two new evasion attacks based on removing features using AMM: (i) manipulating the top AMM-based features to generate adversarial samples $S_a$, and generating an improved LGBM detector $m_1$ by excluding important features; (ii) manipulating the top important features to generate adversarial sample $S_i$, and generating an improved LGBM detector $m_2$ by excluding AMM-based features. To evaluate AMM and SAGE in the same conditions, we use the same parameters: the top 75 features for adversarial samples and the top 220 features for improved detectors. Meanwhile we introduce the seed samples $S_m$ and the original LGBM detector as a benchmark.

In the experiment, we leverage 5,459 seed malwares to generate adversarial samples where we randomly select 300 samples for 20-round tests on each detector.

Figure 7 presents box plots of the detection rate and prediction rate. Recall, the prediction value defines the confidence that a sample is malware. We present results for three malware datasets: seed malware $S_m$, adversarial sample $S_a$ and $S_i$. We define a sample as malicious when the prediction value is greater than or equals to 0.5. On the original detector, less than 15% of samples in both $S_a$ and $S_i$ are classified as malicious. On $m_1$, the average detection rate of the $S_a$ samples increase to 67.3% while their prediction values range from 0.23 to 0.82. In contrast, the detection rates of $S_i$ on $m_2$ are around 78.2%, while their prediction value range from 0.69 to 0.91, which means that $m_2$ can detect more adversarial samples than $m_1$.

This result indicates that AMM-based features have better evasion capability than SAGE-based important features. Meanwhile, improved detectors guided by AMM values has better performance on adversarial detection.

Comparison of AMM and SAGE. Next, we compare how AMM-based and important features impact the detection. Figure 8 illustrates the feature distribution of AMM and SAGE values and the amount of evaded samples manipulating the corresponding features. The X-axis indicates normalized AMM values; the Y-axis is normalized SAGE values. As shown in the figure, AMM-based features are more centrally located on low SAGE-value area while
important features are more likely to have small AMM values. They have a big portion of common features, most of which have small SAGE values but with large AMM values. This result indicates that evaded samples are more likely to have modified features with large AMM values. In addition, these features have small SAGE values so that they are less likely to downgrade the performance of detectors but more to flip prediction labels.

Summary. The results show that our proposed AMM framework can guide us to better select effective features, generating adversarial samples and improving machine learning detectors than SAGE. This is because AMM values explain the capability of features that can flip classification results, and guide the way to improve the detection performance. In contrast, SAGE values reflect the importance of features, explaining how much the feature contributes to accuracy of a machine learning detector.

Takeaway 3:
- AMM values measure the importance of features and the capability of flipping classification results, while SAGE values measure how much a feature contributes to the prediction accuracy.
- Machine learning-based AV products should consider using the AMM values to improve their detectors.

7 CASE STUDIES

In this section, we conduct two case studies to understand: (i) why some adversarial samples cannot evade detection; and (ii) why the SVM implementation seldom detects an adversarial sample.

7.1 A Case Study on Evasion Capability

Our prior results have shown that not all adversarial samples can evade the detection. The reason could be either that the number of manipulated features is not enough to invert the prediction, or that the manipulated features have a limited impact on the prediction. To explore the reason, we choose two seed malicious APK examples, Sample 1 and Sample 2, to generate their adversarial samples. We then test their evasion capability. The adversarial sample of Sample 1 inverts its prediction as benign, and that of Sample 2 remains malicious.

First we use Sample 1 and Sample 2 to manipulate different numbers of features (guided by Algorithm 1) and compare their prediction values. We consider a sample malicious when its prediction value is larger than 0.5 (benign otherwise). Initially, the prediction values of Sample 1 and Sample 2 are 0.99975447 and 0.9997253, and raw scores given by LGBM are 8.31183171 and 8.19956212. After manipulating 75 selected features, the prediction value of Sample 1 turns to 0.28416445 (i.e., benign) with a raw score of -0.92389732. In contrast, the prediction value of Sample 2 remains positive (i.e., malicious) at 0.91948969 with raw score 2.4354351.

We further generate SHAP values of the original and adversarial samples with (N = 75 features selected) of Sample 1 and Sample 2 to analyze the impact of the manipulated features. The results are shown in Figure 9. The subfigures labeled (A) and (B) indicate the original and adversarial samples of Sample 1, respectively. Further, (C) and (D) show results for for Sample 2. The X-axis indicates the prediction value; the Y-axis is the feature ID. $f(x)$ and $f(x_a)$ are raw scores of the original and adversarial samples given by the LGBM detector. We use red bars to indicate positive SHAP values, and blue bars to highlight negative SHAP values of each feature ($\phi_j$ in Equation 1). Feature IDs shown in the figure are parts of manipulated features that had the greatest impact on Sample 1.

Figure 9(b) shows that the SHAP values of manipulated features change significantly towards negative (blue bars), thereby pushing the output, $f(x_a)$, towards negative. In contrast, the SHAP values of the features of Sample 2, shown in Figure 9(d), change far less. Specifically, only features 46227, 40271, 39950 and 73533 are manipulated towards negative while 40960 is not. This means that we cannot manipulate enough features to force the decision making towards benign for Sample 2. This result indicates that the manipulated features have limited impact on Sample 2 to invert the result from malicious to benign. If we increase the number of selecting features to 290, Sample 2 is then identified as benign. Therefore, the detection result depends on how many features have malicious-oriented values that we can manipulate in the sample. However, infinitely increasing the number of selecting features would also lead to heavily computational load and decrease the efficiency of attack.

Takeaway 4: The evasion capability of a manipulated sample depends on the number of its features with malicious-oriented values that can be manipulated.

7.2 A Case Study on Detection Efficiency

In previous experiments, the SVM detector could not detect adversarial samples from all three generation strategies. The previous case study has shown that Sample 2 is still unable to evade detection from LGBM. Therefore, we utilize Sample 2 to conduct a case study to unveil the detection efficiency of SVM.

First we generate adversarial samples by manipulating from 1 to 80 features. We then calculate their decision function values from the SVM. The decision function value represents whether a predicted sample by the classifier occupies the side of and the distance to the Hyperplane. We still use features selected from LGBM and validate them on SVM. Results are shown in Figure 10, where the Y-axis represents the decision function values of samples guided by AMM-based, Balanced AMM-based and Statistics-based strategies. The X-axis shows the number of features manipulated.
Weights of Features

0.5

0.0

-0.5

Weights of Features

Default AMM

Balanced AMM

Statistic

Figure 11: Distribution of weights of features in SVM models trained by samples generated from different strategies.

Positive decision function values represent malicious results while negative ones represent benign results.

The result shows that adversarial samples with AMM-based and balanced AMM-based strategies invert their prediction results while manipulating from 13 to 16 features. The statistics-based adversarial sample is predicted as benign when manipulating 62 features. This result indicates that a small amount (i.e., from 13 to 16) of features selected by top AMM values can invert the detection result of Sample 2 by SVM.

From the result, we speculate that AMM-guided features occupy large weights of SVM classification. To verify this, we export the weights to each feature in the SVM detector and compare weight values of selected features. Figure 11 illustrates weight values of selected features, where the y-axis represents the weight of each feature, and the x-axis shows the indexes of selected features. We can see that most weight values of features selected by the statistics-based strategy are close to 0, while most weight values of features selected by other two strategies have relatively large negative values, making the prediction decision values to be negative (i.e., benign). We also compared selected features from LGBM with SVM features from top AMM values and found that most of LGBM selected features have large AMM values in SVM, which matches the result in Figure 4.

This case study confirms that features selected from LGBM occupy large negative weights in the SVM, making the prediction result of the adversarial sample benign.

Takeaway 5: The transferability of adversarial attacks arises because AMM-guided features selected from one model have determinate weights in another.

8 DISCUSSION & LIMITATIONS
This section discusses challenges to our work, as well as limitations and how they might be addressed in the future.

Learning models. Our experiments are conducted on three existing models followed by the prior research [21, 55], including SVM, LightGBM and a feed-forward neural network. These models, however, are trained with the default configuration and parameters from each dataset. This does not preclude the possibility that alternative models and configurations may gain superior performance.

Adaptive defenses. Existing defenses against adversarial generation, for example, adversarial training [49] and differential privacy (DP) [17, 33, 46], may not be effective against our proposed evasion attack. For adversarial training, it is because that our attack is based on the transferability of important features among different learning-based detectors, in lieu of optimization tricks, e.g., FGSM [35]. On the other hand, DP-based robust machine learning techniques cannot defend against our attacks, because unbounded random perturbations may break the generated samples’ functionality. One possible adaptive defense is to analyze feature distribution of queries before we carry out de facto malware detection. For example, Slack et al. [59] propose an adversarial classification approach to fool explanation methods, e.g., SHAP and LIME. They offer a different classifier, e.g., an extremely biased one, along with the original one to the input according to the perturbation distribution. However, this classification approach can be an attack target and cannot always boost the detection accuracy.

Dynamic detection. Dynamic feature detection can be a practical defence against our evasion attack since we only insert static unreachable instructions into the malware. Feature-space manipulation and problem-space obfuscation rely on static syntactic and structural modification. These modifications can bypass static machine learning-based detectors and rule-based antivirus engines. However, the malicious behaviours will still be exposed during runtime and identified by the detectors that adopt dynamic analysis. That said, dynamic feature detection consuming more resources to monitor this approach may be impractical on a large scale.
9 CONCLUSION

This paper has proposed an explainability-guided malware detector measurement framework, Accrued Malicious Magnitude (AMM), to guide the feature selection approach for adversarial attacks and model improvement. We also developed a binary builder to apply feature-space manipulation on problem-space binaries. We use AMM to measure the performance of state-of-the-art malware detectors protecting against adversarial attacks. Our research includes the following key findings: (i) commercial antivirus engines are vulnerable to AMM-based adversarial samples, while a detector with multiple feature extraction functions reduces the impact in a certain degree; (ii) the transferability of adversarial attack relies on the overlaps of features with large AMM values between different machine learning models; (iii) AMM values can effectively measure the importance of features and explain the capability of flipping classification results. According to our findings, we suggest that machine learning-based AV products should consider using the AMM values to improve their performance. Exploring the latter constitutes our key line of future work, as we believe this could prompt a new approach to defending against evasion attacks.

REFERENCES

[1] [n.d.]. AI and machine learning. https://www.avast.com/en-us/technology/ai-and-machine-learning
[2] [n.d.]. Androguard documentation. https://androguard.readthedocs.io/en/latest/
[3] [n.d.]. Antivirus and Internet security | ESET. https://www.eset.com/
[4] [n.d.]. Apktool - A tool for reverse engineering Android APK files. https://androguard.readthedocs.io/en/latest/
[5] [n.d.]. Avast. https://www.avast.com/
[6] [n.d.]. Cyber security statistics. https://purplesec.us/resources/cyber-security-statistics/
[7] [n.d.]. How anti-virus software works. https://cs.stanford.edu/people/eroberts/cc201/projects/viruses/anti-virus.html
[8] [n.d.]. Kaspersky cyber security solutions for home & business. https://www.kaspersky.com.au/
[9] [n.d.]. Kiteshield. https://github.com/GunshipPenguin/kiteshield
[10] [n.d.]. pefile. https://github.com/erocarrera/pefile
[11] [n.d.]. RelocBonus. https://github.com/nickcano/RelocBonus
[12] [n.d.]. Smali. https://github.com/JesusFreke/smali
[13] [n.d.]. Sonicwall research malware attacks 2019. https://www.nmsapertut.com/cyberecurity-research/sonicwall-research-malware-attacks-2019/
[14] [n.d.]. Virbox protector. https://www.sense.com.cn/VirboxProtector.html
[15] [n.d.]. VirusTotal. https://www.virustotal.com
[16] [n.d.]. VMPProject software protection. https://vmpsoft.com/
[17] [n.d.]. vmProject. https://vmproject.com/
[18] [n.d.]. WP Projekt software protection. https://wpsoft.pl/
[19] [n.d.]. Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, and Kunal Talwar. 2018. Learning to evade static PE machine learning malware models via reinforcement learning. arXiv preprint arXiv:1801.08917 (2018).
[20] [n.d.]. Hyrum S Anderson, Anant Khurkar, Bobby Filas, David Evans, and Phil Roth. 2018. Learning to evade static PE machine learning malware models via reinforcement learning. arXiv preprint arXiv:1804.08637 (2018).
[21] [n.d.]. Simone Aonzo, Gabriele Claudiu Georgiu, Luca Verderame, and Alessio Merlo. 2020. Obfuscator: An open-source black-box obfuscation tool for Android apps. SoftwareX (2020).
[22] [n.d.]. Daniel Arg. Michael Spreitzenbarth, Malte Huhner, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Effective and explainable detection of Android malware in your pocket. In Network and Distributed System Security Symposium (NDSS).
[23] [n.d.]. Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending transcend: Revisiting malware classification with conformal evaluation. In IEEE Symposium on Security and Privacy (SP).
[24] [n.d.]. Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, and Aleksander Madry. 2019. On evaluating adversarial robustness. arXiv preprint arXiv:1902.06705 (2019).
[25] [n.d.]. Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In IEEE Symposium on Security and Privacy (SP).
[26] [n.d.]. Sen Chen, Minhui Xue, Linglan Fan, Lei Ma, Yang Liu, and Linhua Xu. 2019. How can we craft large-scale Android malware? An automated poisoning attack. In International Workshop on Artificial Intelligence for Mobile (AIMobile).
[27] [n.d.]. Yizheng Chen, Shiqi Wang, Dashingong She, and Suman Jana. 2020. On training robust (PDF) malware classifiers. In 29th [USENIX] Security Symposium (USENIX Security Conference 2020) 2343–2360.
[28] [n.d.]. Ian Covert, Scott Lundberg, and Su-Lee Lee. 2020. Understanding Global Feature Contributions With Additive Importance Measures. arXiv:2004.06668 [cs.LG].
[29] [n.d.]. Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2020. Efficient black-box optimization of adversarial windows malware with constrained manipulations. arXiv preprint arXiv:2003.15250 (2020).
[30] [n.d.]. Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. IEEE Transactions on Information Forensics and Security (2021).
[31] [n.d.]. Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Ifigenia Corona, Giorgio Giacinto, and Fabio Roli. 2017. Yes, machine learning can be more secure! A case study on Android malware detection. IEEE Transactions on Dependable and Secure Computing (2017).
[32] [n.d.]. Ambra Demontis, Marco Melis, Maura Finton, Matthew Jajegielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2017. Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks. In USENIX Security Symposium (USENIX Security 2017).
[33] [n.d.]. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. Calibrating noise to sensitivity in private data analysis. In Theory of Cryptography, Third Theory of Cryptography Conference (TCC).
[34] [n.d.]. Kevin Eykholt, Ivan Evtimov, Earleish Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In IEEE Conference on Computer Vision and Pattern Recognition.
[35] [n.d.]. Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. International Conference on Learning Representations (2015).
[36] [n.d.]. Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435 (2016).
[37] [n.d.]. Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In European Symposium on Research in Computer Security.
[38] [n.d.]. Mahmoud Hammad, Joshua Garcia, and Sam Malek. 2018. A large-scale empirical study on the effects of code obfuscations on Android apps and anti-malware products. In International Conference on Software Engineering. 421–431.
[39] [n.d.]. Richard Harang and Ethan M. Rudd. 2020. SOREL-2M: A large scale benchmark dataset for malicious PE detection. arXiv preprint arXiv:2012.07634 (2020).
[40] [n.d.]. Andrew Byas, Shubani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksandr Madry. 2019. Adversarial Examples Are Not Bugs, They Are Features. arXiv:1905.02175 [stat.ML].
[41] [n.d.]. Jinho Jung, Chanil Jeon, Max Wolosoky, Insu Yun, and Taesoo Kim. 2017. AVPASS: Leaking and bypassing advanced detection model automatically. In Black Hat USA Briefings (Black Hat USA).
[42] [n.d.]. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qirui Ye, and Tie-Yan Liu. 2017. LightGBM: A high efficient gradient boosting algorithm on Decision Tree. Advances in Neural Information Processing Systems.
[43] [n.d.]. TaeGuen Kim, Bojoong Kang, Min Rho, Sakir Sezer, and Eul Gyu Im. 2018. A multimodal deep learning method for Android malware detection using various features. IEEE Transactions on Information Forensics and Security (2018).
[44] [n.d.]. Dhillon Kirat and Giovanni Vigna. 2015. MalGene: Automatic extraction of malware. In ACM SIGSAC Conference on Computer and Communications Security.
[45] [n.d.]. Alexey Kuzrin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. arXiv:1607.02533 [cs.CV].
[46] [n.d.]. Mathias Lécyvier, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples. In IEEE Symposium on Security and Privacy (SP).
[47] [n.d.]. Deqing Li, Tian Qiu, Shuo Chen, Qianmu Li, and Shouhuai Xu. 2021. Can we leverage predictive uncertainty to detect dataset shift and adversarial examples in Android malware detection? Annual Computer Security Applications Conference (ACSAC) (2021).
[48] [n.d.]. Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In Conference on Neural Information Processing Systems.
[49] [n.d.]. Aleksandr Madry, Aleksandr Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations.
APPENDICES

Table 3: APK feature IDs and their description

| Feature ID | Description |
|------------|-------------|
| 46227      | receiver=com.google.android.apps.analytics.AnalyticsReceiver |
| 40271      | call=Landroid/media/MediaPlayer->reset/() |
| 39956      | api=Landroid/app/Service->getSystemService(java.lang.String);iljava.lang/Object;int=android.intent.action.VIEW |
| 49559      | url=maps.google.com |

A Hyperparameters

This section introduces how we choose the most effective obfuscation strategy and the amount of features manipulated in explainability-guided sample generation for APK binaries.

B Structures of WinPE Binaries

Figure 14 illustrates the structure of WinPE binary. Specifically, a WinPE binary involves five parts:

- **DOS Header** is a legacy header from DOS era to maintain compatibility with legacy Windows systems.
- **PE Header** involves general information including the target architecture, number of sections and symbols, timestamp, and the header size.
- **Optional Header** contains detailed information required by the system to load, such as the entry-point address, d11 characteristics, size of file, and version information.
- **Section Table** is a list of section headers storing the section name, address, relocations, and other general information.
Algorithm 2: Statistics-based Feature-Space Selection

Input: Dataset $X$, labels $Y$ and the number of benign- and malicious-oriented features to be selected $N$.

Output: Benign-oriented features $B$ and malicious-oriented features $M$.

1. $B = \{\}$;
2. $M = \{\}$;
3. $X' \leftarrow \text{vectorize}(X)$;
4. $M', B' \leftarrow \text{divideDataset}(X', Y)$;
5. $M_{\text{sum}} \leftarrow \text{sum}(M', \text{axis}=0)$;
6. $B_{\text{sum}} \leftarrow \text{sum}(B', \text{axis}=0)$;
7. $M_{\text{sort}} \leftarrow \text{sortDescend}(M_{\text{sum}}, \text{value} > 0)$;
8. $B_{\text{sort}} \leftarrow \text{sortDescend}(B_{\text{sum}}, \text{value} > 0)$;
9. $m_{\text{top}} = M_{\text{sort}}[\text{len}(M') \times 0.1]$;
10. $b_{\text{top}} = M_{\text{sort}}[\text{len}(B') \times 0.1]$;
11. $m_{\text{bottom}} = M_{\text{sort}}[\text{len}(M') \times 0.9]$;
12. $b_{\text{bottom}} = M_{\text{sort}}[\text{len}(B') \times 0.9]$;
13. $d = X'.c\text{olumns}$;
14. for $i$ in $[0...d]$ do
15.   if $B_{\text{sum}}[i] \geq b_{\text{top}}$ & $M_{\text{sum}}[i] \leq m_{\text{bottom}}$ & size($B$) < $N$ then
16.     $B \leftarrow B \cup \{i\}$;
17.   if $M_{\text{sum}}[i] \geq m_{\text{top}}$ & $B_{\text{sum}}[i] \leq b_{\text{bottom}}$ & size($M$) < $N$ then
18.     $M \leftarrow M \cup \{i\}$;
19. return $M, B$;

- Sections contains contiguous chunks of bytes hosting the real content of the executable. For example, .text stores the code, .data stores global variables, and .rdata stores read-only constants and counting.

C  Statistics-based Feature Selection

Algorithm 2 illustrates the process of feature selection. First, after vectorizing the dataset, $X'$ is divided into malicious set $M'$ and benign set $B'$. Then we summarize all feature values across the samples by feature, representing the number of existence of each feature $M_{\text{sum}}$ and $B_{\text{sum}}$. Then we choose top and bottom 10% summary value as the threshold of majority and minority feature in benign and malicious dataset. Finally, we traverse $M'$ and $B'$ to find benign and malicious-oriented features $B$ and $N$. In feature manipulating phase, benign-oriented features will be set as 1 in a sample while malicious ones will be set as 0.

D  Shapley Additive Global importance (SAGE)

SHAP calculates the contribution of each feature to an individual prediction (local interpretability). Shapley Additive Global importance (SAGE) [28] summarizes each feature’s importance based on the predictive power it contributes across whole dataset (global interpretability). The features that are most critical for the model to make good predictions will have large values, while unimportant features will have small values. To accomplish this task, SAGE defines the restricted model $f_S$ that only a part of the entire features set are chosen as:

$$f_S(x_S) = \mathbb{E}[f(X) \mid X_S = x_S],$$

where $X_S \equiv \{X_i \mid i \in S\}$ and $S \subseteq D$ is a subset of the full features set $D$. Given a loss function $\ell$, it defines the prediction power $\nu_f(S)$ given a subset of features $S$:

$$\nu_f(S) = \mathbb{E}[\ell(f_S(x_S), Y)] - \mathbb{E}[\ell(f_S(x_S), Y)].$$

Mean prediction Using features $X_S$

where $\nu_f(S)$ represents the performance of $f$ given features $X_S$. As well known in game theory, Shapley values are the unique credit distribution that satisfies the above axioms. The Shapley value $\phi_i$ of feature $X_i$ is defined as:

$$\phi_i(\nu_f) = \frac{1}{d} \sum_{S \subseteq \{D \setminus \{i\}} |S|^{-1} \left( \nu_f(S \cup \{i\}) - \nu_f(S) \right).$$

where each Shapley value $\phi_i(\nu_f)$ is a weighted average of the incremental changes from adding $i$ to subsets $S \subseteq D \setminus \{i\}$.