Arms Race in Adversarial Malware Detection: A Survey

DEQIANG LI and QIANMU LI, Nanjing University of Science and Technology
YANFANG (FANNY) YE, Case Western Reserve University
SHOUHUAI XU, University of Colorado Colorado Springs

Malicious software (malware) is a major cyber threat that has to be tackled with Machine Learning (ML) techniques because millions of new malware examples are injected into cyberspace on a daily basis. However, ML is vulnerable to attacks known as adversarial examples. In this article, we survey and systematize the field of Adversarial Malware Detection (AMD) through the lens of a unified conceptual framework of assumptions, attacks, defenses, and security properties. This not only leads us to map attacks and defenses to partial order structures, but also allows us to clearly describe the attack-defense arms race in the AMD context. We draw a number of insights, including: knowing the defender's feature set is critical to the success of transfer attacks; the effectiveness of practical evasion attacks largely depends on the attacker's freedom in conducting manipulations in the problem space; knowing the attacker's manipulation set is critical to the defender's success; and the effectiveness of adversarial training depends on the defender's capability in identifying the most powerful attack. We also discuss a number of future research directions.

CCS Concepts: • **Security and privacy → Malware and its mitigation**; • **Theory of computation → Adversarial learning;**

Additional Key Words and Phrases: Malware detection, adversarial machine learning, evasion attacks, poisoning attacks

ACM Reference format:
Deqiang Li, Qianmu Li, Yanfang (Fanny) Ye, and Shouhuai Xu. 2021. Arms Race in Adversarial Malware Detection: A Survey. *ACM Comput. Surv.* 55, 1, Article 15 (November 2021), 35 pages.
https://doi.org/10.1145/3484491

Work was partly done when Shouhuai Xu was affiliated with University of Texas at San Antonio, One UTSA Circle, San Antonio, Texas 78249.
Q. Li is supported in part by the National Key R&D Program of China under Grant 2020YFB1805503, the Jiangsu Province Modern Education Technology Research Project under Grant 84365, the National Vocational Education Teacher Enterprise Practice Base "Integration of Industry and Education" Special Project (Study on Evaluation Standard of Artificial Intelligence Vocational Skilled Level). Y. Ye and S. Xu are supported in part by NSF Grant #2122631 (#1814825). S. Xu is also supported in part by ARO Grant #W911NF-17-1-0566, NSF Grant #2115134, and Colorado State Bill 18-086.
Authors' addresses: D. Li and Q. Li, Nanjing University of Science and Technology, Xiaolingwei Street, Nanjing, Jiangsu, China, 210094; emails: lideqiang@njust.edu.cn, qianmu@njust.edu.cn; Y. (Fanny) Ye, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556, USA; email: yye7@nd.edu; S. Xu, University of Colorado Colorado Springs, 1420 Austin Bluffs Pkwy, Colorado Springs, Colorado, 80918, USA; email: sxu@uccs.edu.

0360-0300/2021/11-ART15 $15.00
https://doi.org/10.1145/3484491

1 INTRODUCTION

Malicious software (**malware**) is a big cyber threat and has received a due amount of attention. For instance, Kaspersky reports that 21,643,946 unique malicious files were detected in the year 2018, 24,610,126 in 2019, and 33,412,568 in 2020 [47, 72]. A popular defense against malware is to use *signature-based* detectors [44], where a signature is often extracted by malware analysts from known malware examples. This approach has two drawbacks: signatures are tedious to extract and can be evaded [34] by a range of techniques (e.g., encryption, repacking, and polymorphism [7, 21, 63, 86, 106, 111, 139]). This incompetence has motivated the use of **Machine Learning** (**ML**) based malware detectors, which can be automated to some degree and can possibly detect *new* malware examples (via model generalization or knowledge adaptation [8, 11, 43, 45, 56, 57, 64, 66, 84, 121, 137, 138, 143]). More recently, **Deep Learning** (**DL**) has been used for malware detection (see, e.g., [99, 103, 125]).

While promising, ML-based malware detectors are vulnerable to attacks known as *adversarial examples* [16, 60, 114]. There are two kinds of attacks. One is *evasion attack*, where the attacker perturbs test examples to *adversarial examples* to evade malware detectors [2, 16, 39, 53, 57, 91, 120, 127]. The other is *poisoning attack*, where the attacker manipulates the training dataset for learning malware detectors [30, 40, 118]. These attacks usher in the new field of **Adversarial Malware Detection** (**AMD**) [16, 27, 36, 39, 40, 53, 114, 118, 127, 133, 134].

The state-of-the-art in AMD is that there are some specific results scattered in the literature but there is no systematic understanding. This is true despite that there have been attempts at systematizing the related field of **Adversarial Machine Learning** (**AML**) [9, 26, 60, 140], which however cannot be automatically translated to AMD. This is so because malware detection has three *unique* characteristics which are not exhibited by the other application domains (e.g., image or audio processing). (i) There are no common, standard feature definitions because both attackers and defenders can define their own features to represent computer files. As a consequence, attackers can leverage this "freedom" in feature definition to craft adversarial examples. (ii) Malware features are often discrete rather than continuous and program files are often highly structured with multiple modalities. This means that arbitrarily perturbing malware files or their feature representations might make the perturbed files no more executable. This also means that the discrete domain makes perturbation a *non-differentiable* and *non-convex* task. (iii) Any meaningful perturbation to a malware example or its feature representation must preserve its malicious functionality. For example, the **Android Package Kit** (**APK**) requires that the used permissions are publicized in the AndroidManifest.xml, meaning that removing permissions in this manifest file would incur a runtime error. The preceding (ii) and (iii) make both the attacker's and defender's tasks more challenging than their counterparts where small perturbations are not noticeable (e.g., images).

Our Contributions. We propose a conceptual framework for systematizing the AMD field through the lens of *assumptions*, *attacks*, *defenses*, and *security properties*. In specifying these, we seek rigorous definitions whenever possible, while noting that these definitions have been scattered in the literature. Rigorous definitions are important because they can serve as a common reference for future studies. The framework allows us to map the known attacks and defenses into some partial order structures and systematize the AMD attack-defense arms race.

We make a number of observations, including: (i) the indiscriminate attack that treats malicious examples as equally important has been extensively investigated, but targeted and availability attacks are much less investigated; (ii) the evasion attack is much more extensively studied than the poisoning attack; (iii) there is no silver-bullet defense against evasion and poisoning attacks; (iv) **sanitizing examples** (**SE**) is effective against black-box and gray-box attacks, but not white-box attacks; (v) AMD security properties have been evaluated empirically rather than rigorously;

(vi) there is no theoretical evidence to support that the effectiveness of defense techniques on the training set can generalize to other adversarial examples.

We draw a number of insights, including: (i) knowing defender's feature set is critical to the success of transfer attacks, highlighting the importance of keeping defender's feature set secret (e.g., by randomizing defender's feature set); (ii) the effectiveness of practical evasion attacks largely depends on the attacker's degree of freedom in conducting manipulations in the problem space (i.e., a small degree of freedom means harder to succeed); (iii) effective defenses often require the defender to know the attacker's manipulation set, explaining from one perspective why it is hard to design effective defenses; (iv) effectiveness of **adversarial training** (**AT**) depends on the defender's capability in identifying the most powerful attack.

Finally, we discuss a number of future research directions, which hopefully will inspire and encourage many researchers to explore them.

Related Work. The closely related prior work is Maiorca et al. [81], which surveys previous studies in adversarial malicious PDF document detection. In contrast, we consider the broader context of AMD and propose novel partial orders to accommodate AMD assumptions, attacks, defenses, and properties. There are loosely-related prior studies, which survey prior AML studies (but not focusing on AMD), including [9, 10, 17, 26, 79, 94, 105, 140]. For example, Yuan et al. [140] survey attack methods for generating adversarial examples, while briefly discussing evasion attacks in the AMD context; Barreno et al. [9, 10] propose a taxonomy of AML attacks (causative vs. exploratory attacks, integrity vs. availability attacks, and targeted vs. indiscriminate attacks); Biggio et al. [17] propose a defense framework for protecting **Support Vector Machines** (**SVMs**) from evasion attacks, poisoning attacks, and privacy violations; Papernot et al. [94] systematize AML security and privacy with emphasis on demonstrating the tradeoff between detection accuracy and robustness.

Paper Outline. Section 2 describes our survey and systematization methodology and framework. Section 3 applies our framework to systematize the literature AMD studies. Section 4 discusses future research directions. Section 5 concludes the article.

2 SURVEY AND SYSTEMATIZATION METHODOLOGY

Terminology, Scope, and Notations. In the AMD context, a defender $\mathcal{I}$ aims at using ML to detect or classify computer files as benign or malicious; i.e., we focus on *binary classification*. An attacker $\mathcal{A}$ attempts to make malicious files evade $\mathcal{I}$'s detection by leveraging *adversarial files* (interchangeably, *adversarial examples*). Adversarial malware examples are often generated by perturbing or manipulating malware examples, explaining why we will use the two terms, perturbation and manipulation, interchangeably. Adversarial attacks can be waged in the training phase of a ML model (a.k.a., poisoning attack) or in the test phase (a.k.a., evasion attack). It is worth mentioning that the *privacy violation* attack [60] is waged in addition to the preceding two attacks because $\mathcal{A}$ can always probe $\mathcal{I}$'s detectors. A file, benign and malicious alike, is adversarial if it is intentionally crafted to (help malicious files) evade $\mathcal{I}$'s detection, and non-adversarial otherwise. We focus on $\mathcal{I}$ using supervised learning to detect malicious files, which may be adversarial or non-adversarial because they co-exist in the real world with no self-identification. This means that, we do not consider the large body of malware detection literature that does not cope with AMD, which has been addressed elsewhere (e.g., [101]). Table 1 summarizes the main notations used in the article.

2.1 Brief Review on ML-based Malware Detection

Let $\mathcal{Z}$ be the *example space* of benign/malicious adversarial/non-adversarial files. Let $\mathcal{Y} = \{+, -\}$ or $\mathcal{Y} = \{1, 0\}$ be the *label space* of binary classification, where $+/1$ $(-/0)$ means a file is

Table 1. Main Notations Used in the Article

Notation	Meaning
$\mathbb{R}$ ($\mathbb{R}_+$)	the set of (positive) real numbers
$\mathcal{A}, \mathcal{I}$	attacker and defender (treated as algorithms)
$\mathbb{P}$	the probability function
$z, z' \in \mathcal{Z}$	$\mathcal{Z}$ is example space; z' is obtained by perturbing z
S	defender $\mathcal{I}$'s feature set for representing files
$\mathbf{x}, \mathbf{x}' \in \mathcal{X}$	$\mathcal{X} = \mathbb{R}^d$ is d-dimensional feature space; $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ are, respectively, feature representations of $z, z' \in \mathcal{Z}$
$\mathcal{Y}, y$	$\mathcal{Y}$ is the label space of binary classification, $\mathcal{Y} = \{+/1, -/0\}$; $y \in \mathcal{Y}$
$\mathcal{D} = \mathcal{Z} \times \mathcal{Y}$	the file-label (i.e., example-label) space
$D_{train} \subset \mathcal{D}, n$	the training set in file-label space; $n = \lvert D_{train} \rvert$
D_{test}	the test set in file-label space
D_{poison}, D'_{poison}	D'_{poison} is set of adversarial file-label pairs obtained by perturbing non-adversarial files in $D_{poison} \subset \mathcal{D}$
$O(z, z')$	$O(z, z') : \mathcal{Z} \times \mathcal{Z} \to \{\texttt{true}, \texttt{false}\}$ is an oracle telling if two files have the same functionality or not
δ	a manipulation for perturbing files *with* preserving their functionalities
$\mathcal{M}$, $\mathcal{Z}_{\mathcal{M}} \subseteq \mathcal{Z}$	$\mathcal{M}$ is manipulation set in the problem space; $\mathcal{Z}_{\mathcal{M}}$ is set of adversarial files generated using $\mathcal{M}$
$\mathbf{M}$, $\mathcal{X}_{\mathbf{M}} \subseteq \mathcal{X}$	$\mathbf{M}$ is feature manipulation set; $\mathcal{X}_{\mathbf{M}}$ is set of adversarial feature vectors generated using $\mathbf{M}$
$\Gamma(z, z')$	$\Gamma(z, z') : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}_+$ measures the degree of manipulation for perturbing $z \in \mathcal{Z}$ into $z' \in \mathcal{Z}$
$C(\mathbf{x}, \mathbf{x}')$	$C(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is the function measuring the cost incurred by changing feature vector $\mathbf{x}$ to $\mathbf{x}'$
$\delta_z \in \mathcal{M}$	δ_z is a set of manipulations of z w.r.t. z'
$\delta_{\mathbf{x}} \in \mathbf{M}$	$\delta_{\mathbf{x}} = \mathbf{x}' - \mathbf{x}$ is a perturbation vector of $\mathbf{x}$ w.r.t. $\mathbf{x}'$
$\phi : \mathcal{Z} \to \mathcal{X}$	feature extraction function; $\mathbf{x} \leftarrow \phi(z)$, $\mathbf{x}' \leftarrow \phi(z')$
φ, f	$\varphi : \mathcal{X} \to \mathbb{R}$ is classification function; $f : \mathcal{Z} \to \mathbb{R}$ is classifier $f = \varphi(\phi(\cdot))$; by abusing notation a little bit, we also use "$+ \leftarrow f(z)$" to mean that f predicts z as malicious when $f(z) \geq \tau$ for a threshold τ
$F_\theta : \mathcal{X} \to \mathbb{R}$	ML algorithm with parameters θ
$L : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$	loss function measuring prediction error of F_θ
EL, WR, AT	defense techniques: Ensemble Learning, Weight Regularization, AT
VL, RF, IT, CD, SE	defense techniques: Verifiable Learning, Robust Feature, Input Transformation, Classifier ranDomization, SE
BE, OE, BP, OP	attack tactics: basic and optimal evasion; basic and optimal poisoning
GO,SF,MI	attack techniques: Gradient-based Optimization, Sensitive Features, MImicry
TR,HS,GM,MS	attack techniques: TRansferability, Heuristic Search, Generative Model, Mixture Strategy
$A_1, \ldots, A_5$	the 5 attributes under $\mathcal{I}$'s control; they are known to $\mathcal{A}$ at respective degrees $a_1, \ldots, a_5$
$A_6, \ldots, A_9$	the 4 attributes under $\mathcal{A}$'s control; they are known to $\mathcal{I}$ at respective degree $a_6, \ldots, a_9$
RR, CR, DR, TR	security properties: Representation Robustness, Classification Robustness, Detection Robustness, Training Robustness

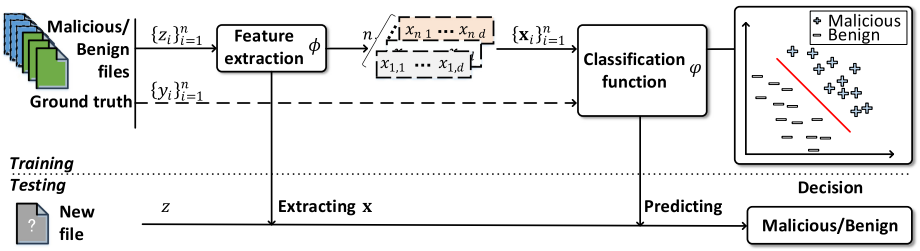

Fig. 1. Illustration of ML-based malware detector.

malicious (benign). Let $\mathcal{D} = \mathcal{Z} \times \mathcal{Y}$ be the file-label (example-label) space. For training and evaluating a classifier in the absence of adversarial files, $\mathcal{I}$ is given a set $D \subset \mathcal{D}$ of non-adversarial benign/malicious files as well as their *ground-truth* labels. $\mathcal{I}$ splits D into three disjoint sets: a training set $D_{train} = \{(z_i, y_i)\}_{i=1}^n$, a validation set for model selection, and a test set for evaluation. Each file $z_i \in \mathcal{Z}$ is characterized by a set S of features and represented by a numerical vector $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$ in the d-dimensional *feature space* $\mathcal{X} = \mathbb{R}^d$, which accommodates both continuous and discrete feature representations [1, 55, 56, 69, 113, 121]. The process for obtaining feature representation $\mathbf{x}_i$ of $z_i \in \mathcal{Z}$ is called *feature extraction*, denoted by a function $\phi : \mathcal{Z} \to \mathcal{X}$ with $\mathbf{x}_i \leftarrow \phi(S, z_i)$. Because ϕ can be hand-crafted (denoted by ϕ_c), automatically learned (denoted by ϕ_a), or a hybrid of both [13], we unify them into ϕ such that $\phi(S, z) = \phi_a(\phi_c(S, z))$; when only manual (automatic) feature extraction is involved, we can set ϕ_a (ϕ_c) as the identity map. There are two kinds of features: *static features* are extracted via static analysis (e.g., strings, API calls [4, 129]); *dynamic features* are extracted via dynamic analysis (e.g., instructions, registry activities [42, 64]).

As highlighted in Figure 1, $\mathcal{I}$ uses $\{(z_i, y_i)\}_{i=1}^n$ to learn a malware detector or classifier $f : \mathcal{Z} \to [0, 1]$, where $f(z) = \varphi(\phi(S, z))$ is composed of feature extraction function $\phi : \mathcal{Z} \to \mathcal{X}$ and classification function $\varphi : \mathcal{X} \to [0, 1]$. Note that $f(z) \in [0, 1]$, namely $\varphi(\mathbf{x}) \in [0, 1]$ with $\mathbf{x} \leftarrow \phi(S, z)$, can be interpreted as the probability that z is malicious (while noting that calibration may be needed [90]). For a given threshold $\tau \in [0, 1]$, we further say (by slightly abusing notations) z is labeled by f as $+$, or $+ \leftarrow f(z)$, if $f(z) \geq \tau$, and labeled as $-$ or $- \leftarrow f(z)$ otherwise. In practice, f is often specified by a learning algorithm F with learnable parameter θ (e.g., weights) and a hand-crafted feature extraction ϕ_c; then, θ is tuned to minimize the empirical risk associated with a loss function $L : [0, 1] \times \mathcal{Y} \to \mathbb{R}$ measuring the prediction error of F_θ [124] (e.g., cross-entropy [49]), namely

$$\min_\theta \mathcal{L}(\theta, D_{train}) = \min_\theta \frac{1}{n} \sum_{(z_i, y_i) \in D_{train}} (L(F_\theta(\phi_c(S, z_i)), y_i)). \quad (1)$$

Example 1 (The Drebin Malware Aetector). Drebin is an Android malware detector trained from static features [8]. Table 2 summarizes Drebin's feature set, which includes four subsets of features S_1, S_2, S_3, S_4 extracted from the AndroidManifest.xml and another four subsets of features S_5, S_6, S_7, S_8 extracted from the disassembled DEX code files (recalling that DEX code is compiled from a program written in some language and can be understood by the Android Runtime). Specifically, S_1 contains features related to the access of an APK to smartphone hardware (e.g., camera, touchscreen, or GPS module); S_2 contains features related to APK's requested permissions listed in the manifest prior to installation; S_3 contains features related to application components (e.g., *activities*, *service*, *receivers*, and *providers*.); S_4 contains features related to APK's communications with the other APKs; S_5 contains features related to critical system API calls, which cannot run without appropriate permissions or the *root* privilege; S_6 contains features corresponding to the used

Table 2. Drebin Features

Feature set	
Manifest	S_1 Hardware components
	S_2 Requested permissions
	S_3 App components
	S_4 Filtered intents
Dexcode	S_5 Restricted API calls
	S_6 Used permissions
	S_7 Suspicious API calls
	S_8 Network addresses

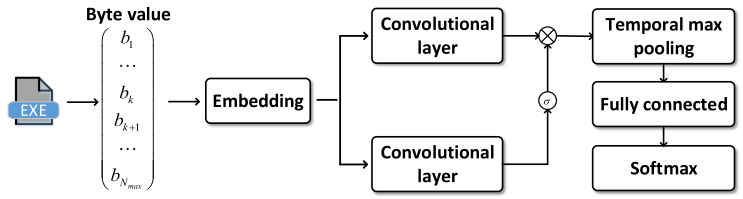

Fig. 2. MalConv architecture [99].

permissions; S_7 contains features related to API calls that can access sensitive data or resources in a smartphone; and S_8 contains features related to IP addresses, hostnames and URLs found in the disassembled codes. The feature representation is binary, meaning $\phi = \phi_c : \mathcal{Z} \mapsto \{0,1\}^d$ with $|S| = d$ and $\mathbf{x} = (x_1, \ldots, x_d)$, where $x_i = 1$ if the corresponding feature is present in the APK z and $x_i = 0$ otherwise. A file z in the feature space looks like the following:

$$\mathbf{x} = \phi(z) \rightarrow \begin{pmatrix} \cdots & \cdots \\ 0 & \texttt{permission::SEND\_SMS} \\ 1 & \texttt{permission::READ\_CONTACTS} \\ \cdots & \cdots \\ 1 & \texttt{api\_call::getDeviceID} \\ 0 & \texttt{api\_call::setWifiEnabled} \\ \cdots & \cdots \end{pmatrix} \begin{matrix} \\ \} S_2 \\ \\ \\ \} S_5 \\ \\ \end{matrix}$$

Drebin uses a linear SVM to learn classifiers.

Example 2 (The MalConv Malware Detector). MalConv [99] is **Convolutional Neural Network** (**CNN**)-based Windows **Portable Executable** (**PE**) malware detector learned from raw binary programs (i.e., end-to-end detection) [67]. Figure 2 depicts its architecture. The sequence of binary code is transformed into byte values (between 0 to 255) with the maximum length bounded by N_{max} (e.g., $N_{max} = 2^{21}$ bytes or 2MB). Each byte is further mapped into a real-valued vector using the embedding [35]. The CNN layer and pooling layer learn abstract representations. The embedding, CNN and pooling layers belong to feature extraction ϕ_a, and the fully-connected and softmax layers belong to the classification operation φ.

2.2 Framework

We systematize AMD studies through the lens of four aspects: (i) the assumptions that are made; (ii) the attack or threat model in terms of attacker $\mathcal{A}$'s *objective* and $\mathcal{A}$'s *input*, with the latter including $\mathcal{A}$'s information about the defender $\mathcal{I}$ and $\mathcal{A}$'s own; (iii) the defense in terms of $\mathcal{I}$'s

objective and $\mathcal{I}$'s *input*, with the latter including $\mathcal{I}$'s information about $\mathcal{A}$ and $\mathcal{I}$'s own; (iv) the security properties that are at stake. These four aspects are, respectively, elaborated below.

2.2.1 Systematizing Assumptions. Five assumptions have been made in the AMD literature. Assumption 1 below says that the data samples in D are **Independent and Identically Distributed** (**IID**), which is a strong assumption and researchers have started to weaken it [52, 110].

Assumption 1 (IID Assumption; See, e.g., [107]). *Computer files in training data and testing data are independently drawn from the same distribution.*

Assumption 2 below is adapted from AML context, where humans can serve as an oracle O for determining whether two images are the same [126]. In the AMD context, O can be instantiated as (or approximated by) malware analysts [2, 27, 65, 114] or automated tools (e.g., Sandbox [36, 134]), with the latter often using heuristic rules produced by malware analysts (e.g., YARA [3]).

Assumption 2 (Oracle Assumption; Adapted from [126]). *There is an oracle* $O : \mathcal{Z} \times \mathcal{Z} \to \{\texttt{true}, \texttt{false}\}$ *that tells if two files* $z, z' \in \mathcal{Z}$ *have the same functionality or not;* $\texttt{true} \leftarrow O(z, z')$ *if and only if* z *and* z' *have the same functionality.*

Assumption 3 below says that there is a way to measure the degree of manipulations by which one file is transformed to another.

Assumption 3 (Measurability Assumption [36, 68]). *There is a function* $\Gamma(z, z') : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}_+$ *that measures the degree of manipulations according to which a file* $z' \in \mathcal{Z}$ *can be derived from the file* $z \in \mathcal{Z}$.

Since Assumption 3 is often difficult to validate, $\Gamma(z, z')$ may be replaced by a function that quantifies the degree of manipulation that can turn feature representation $\mathbf{x}$ into $\mathbf{x}'$, where $\mathbf{x} = \phi(S, z)$ and $\mathbf{x}' = \phi(S, z')$. This leads to:

Assumption 4 (Smoothness Assumption [13]). *There is a function* $C(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ *such that* $C(\phi(S, z), \phi(S, z')) \approx 0$ *when* $(\Gamma(z, z') \approx 0) \wedge (\texttt{true} \leftarrow O(z, z'))$.

Assumption 5 below says that the inverse of feature extraction, ϕ^{-1}, is solvable so that a perturbed representation $\mathbf{x}'$ can be mapped back to a legitimate file.

Assumption 5 (Invertibility Assumption [76]). *Feature extraction* ϕ *is invertible, meaning that given* $\mathbf{x}'$*, the function* $\phi^{-1} : \mathcal{X} \to \mathcal{Z}$ *produces* $z' = \phi^{-1}(\mathbf{x}')$.

Recall that the feature extraction function ϕ may be composed of a hand-crafted ϕ_c and an automated ϕ_a, where ϕ_c may be neither differentiable nor invertible [16, 97]. This means $\mathbf{x}'$ may not be mapped to a legitimate file. Researchers tend to relax the assumption by overlooking the interdependent features [76, 114], while suffering from the side-effect $\mathbf{x}' \neq \phi(\phi^{-1}(\mathbf{x}'))$ [97, 114].

2.2.2 Systematizing Attacks. We systematize attacks from two perspectives: *attacker's objective* (i.e., what the attacker attempts to accomplish) and *attacker's input* (i.e., what leverages the attacker can use). Whenever possible, we seek rigorous definitions to specify the attacker's input, while noting that these definitions have been scattered in the literature. We believe this specification is important because it can serve as a common reference for future studies. To demonstrate this, we discuss how to apply it to formulate a partial-order structure for comparing attacks.

Attacker's Objective. There are three kinds of objectives: (i) Indiscriminate, meaning $\mathcal{A}$ attempts to cause as many false-negatives as possible [6, 28, 53, 100, 127, 136]; (ii) Targeted, meaning $\mathcal{A}$ attempts to cause specific false-negatives (i.e., making certain malicious files evade the detection [40, 118]); (iii) Availability, meaning $\mathcal{A}$ attempts to frustrate defender $\mathcal{I}$ by rendering $\mathcal{I}$'s classifier f unusable (e.g., causing substantially high false-positives [20, 30, 40, 87, 98]).

Table 3. Attributes for Specifying $\mathcal{A}$'s and $\mathcal{I}$'s Input

Attributes	Attacker $\mathcal{A}$'s input	Defender $\mathcal{I}$'s input
Attributes under $\mathcal{I}$'s control but may be known to $\mathcal{A}$ to some extent		
A_1: Training set D_{train}	$a_1 \in [0,1]$	1
A_2: Defense technique	$a_2 \in \{0,1\}$	1
A_3: Feature set S	$a_3 \in [0,1]$	1
A_4: Learning algorithm F_θ	$a_4 \in [0,1]$	1
A_5: Response	$a_5 \in \{0,1\}$	1
Attributes under $\mathcal{A}$'s control but may be known to $\mathcal{I}$ to some extent		
A_6: Manipulation set $\mathcal{M}$	1	$a_6 \in [0,1]$
A_7: Attack tactic	1	$a_7 \in \{0,1\}$
A_8: Attack technique	1	$a_8 \in \{0,1\}$
A_9: Adversarial examples	1	$a_9 \in [0,1]$

Attacker's Input. Table 3 highlights the attributes we define to describe $\mathcal{A}$'s input, including: five attributes $A_1, \ldots, A_5$ that are under $\mathcal{I}$'s control (indicated by 1) but may be known to $\mathcal{A}$ at some extent $a_1, \ldots, a_5$, respectively; and four attributes $A_6, \ldots, A_9$ that are under $\mathcal{A}$'s control (indicated by 1). These attributes are elaborated below.

(i) A_1: it describes $\mathcal{I}$'s training set D_{train} for learning classifier f. We use $a_1 \in [0,1]$ to represent the extent at which D_{train} is known to $\mathcal{A}$. Let $\hat{D}_{train}$ be the training files that are known to $\mathcal{A}$. Then, $a_1 = |\hat{D}_{train} \cap D_{train}|/|D_{train}|$.

(ii) A_2: it describes $\mathcal{I}$'s techniques, which can be **Ensemble Learning** (**EL**), **Weight Regularization** (**WR**), AT, **Verifiable Learning** (**VL**), **Robust Feature** (**RF**), **Input Transformation** (**IT**), **Classifier ranDomization** (**CD**), and SE. Let $A_2 \in \{\mathsf{EL}, \mathsf{WR}, \mathsf{AT}, \mathsf{VL}, \mathsf{RF}, \mathsf{IT}, \mathsf{CD}, \mathsf{SE}\}$ and $a_2 \in \{0,1\}$ such that $a_2 = 0$ means $\mathcal{A}$ does not know $\mathcal{I}$'s techniques and $a_2 = 1$ means $\mathcal{A}$ knows $\mathcal{I}$'s technique. The techniques are defined as follows. Definition 1 says that $\mathcal{I}$ constructs multiple classifiers and uses them collectively in malware detection.

Definition 1 (Ensemble Learning or EL *[142]).* Let $\mathcal{H}$ be $\mathcal{I}$'s classifier space. Given K classifiers $\{f_i\}_{i=1}^K$ where $f_i \in \mathcal{H}$ and $f_i : \mathcal{Z} \to [0,1]$, let f_i be assigned with weight ω_i with $\sum_{i=1}^K \omega_i = 1$ and $\omega_i \geq 0$. Then, $f = \sum_{i=1}^K \omega_i f_i$.

Definition 2 says that $\mathcal{I}$ uses regularization (e.g., ℓ_2 regularization [89] or *dropout* [112]) to decrease model's sensitivity to adversarial examples.

Definition 2 (Weight Regularization or WR *[49]).* Given a regularization item Ω (e.g., constraints imposed on the learnable parameters), the empirical risk is $\min_\theta [\mathcal{L}(\theta, D_{train}) + \Omega(\theta)]$, where $\mathcal{L}$ is defined in Equation (1).

Definition 3 says that $\mathcal{I}$ proactively makes its classifier f perceive some information about adversarial files. That is, $\mathcal{I}$ augments the training set by incorporating adversarial examples that may be produced by $\mathcal{I}$, $\mathcal{A}$, or both.

Definition 3 (Adversarial Training or AT *[53]).* Let D' denote a set of adversarial file-label pairs. Then, $\mathcal{I}$ tunes model parameters by minimizing the empirical risk: $\min_\theta [\mathcal{L}(\theta, D_{train}) + \beta \mathcal{L}(\theta, D')]$, where $\beta \geq 0$ denotes a balance factor.

Definition 4 says that $\mathcal{I}$ intentionally over-estimates the error incurred by $\mathcal{A}$'s manipulations and then minimizes it.

Definition 4 (Verifiable Learning or VL *[130]).* Given $(z, y) \in D_{train}$ and a manipulation set $\hat{\mathcal{M}}$ known by $\mathcal{I}$, let $z(\hat{\mathcal{M}})$ denote the upper and lower boundaries on $\hat{\mathcal{M}}$. Then, this defense technique minimizes the following loss function derived from Equation (1): $L(F_\theta(\phi_c(S, z)), y) + \beta L(F_\theta(\phi_c(S, z(\hat{\mathcal{M}}))), y)$.

Definition 5 says that $\mathcal{I}$ uses a set of features $S^* \subseteq S$ that can lead to higher detection capability against adversarial example attacks.

Definition 5 (Robust Feature or RF*; Adapted from [141]).* Given a training set $D_{train} \cup D'$ that contains (adversarial) file-label pairs, the set of RF set S^* is

$$S^* = \arg\min_{\tilde{S} \subset S} \sum_{(z,y) \in D_{train} \cup D'} L(\widetilde{F}_\theta(\phi_c(\tilde{S}, z)), y),$$

where $\widetilde{F}_\theta$ is F_θ or a simplified learning algorithm that is computationally faster than F_θ [141].

Definition 6 says that $\mathcal{I}$ aims at using non-learning methods (e.g., de-obfuscation as shown in Proguard [14]) to offset $\mathcal{A}$'s manipulations.

Definition 6 (Input Transformation or IT*, Adapted from [28]).* Let $\texttt{IT} : \mathcal{Z} \to \mathcal{Z}$ denote an IT in the file space. Given file z and transformation IT, the classifier is $f = \varphi(\phi(\texttt{IT}(z)))$.

Definition 7 says that $\mathcal{I}$ randomly chooses m classifiers and uses their results for prediction. That is, $\mathcal{I}$ aims at randomizing the feature representation used by f, the learning algorithm, and/or response to $\mathcal{A}$'s queries (to prevent $\mathcal{A}$ from inferring information about f).

Definition 7 (Classifier Randomization or CD*; Adapted from [65]).* Given $\mathcal{I}$'s classifier space $\mathcal{H}$ and an input file z, $\mathcal{I}$ randomly selects m classifiers from $\mathcal{H}$ with replacement, say $\{f_i\}_{i=1}^m$. Then, $f = \frac{1}{m}\sum_{i=1}^m f_i(z)$.

Instead of enhancing malware detectors, Definition 8 provides an alternative that detects the adversarial examples for further analysis.

Definition 8 (Sanitizing Examples or SE*; Adapted from [23, 30]).* $\mathcal{I}$ aims at detecting adversarial files by using function $\text{flag} : \mathcal{Z} \to \{\texttt{yes}, \texttt{no}\}$ to flag a file as adversarial (yes) or not (no).

(iii) A_3: it describes $\mathcal{I}$'s feature set S. We use $a_3 \in [0, 1]$ to represent the extent at which $\mathcal{A}$ knows about S. Let $\hat{S}$ denote the features that are known to $\mathcal{A}$. Then, $a_3 = |\hat{S} \cap S|/|S|$.

(iv) A_4: it describes $\mathcal{I}$'s learning algorithm F_θ, the set of trainable parameters θ, and hyperparameters (which are set manually, e.g., β in Definition 3) [40, 88]. We use $a_4 \in [0, 1]$ to represent that $\mathcal{A}$ knows an a_4 degree about A_4, where $a_4 = 0$ means $\mathcal{A}$ knows nothing and $a_4 = 1$ means $\mathcal{A}$ knows everything.

(v) A_5: it describes $\mathcal{I}$'s response to $\mathcal{A}$'s query to f (if applicable), which is relevant because $\mathcal{A}$ can learn useful information about f by observing f's responses [119]. We define $a_5 \in \{0, 1\}$ such that $a_5 = 0$ means there is a limit on the response that can be made by $\mathcal{A}$ to f (referred as LQ) and $a_5 = 1$ means there is no limit (referred as FQ).

(vi) A_6: it describes $\mathcal{A}$'s manipulation set in the problem space, which describes perturbations for generating adversarial files (adapted from *perturbation set* in the AML literature [123]):

$$\mathcal{M} = \{\delta : (z' \leftarrow \mathcal{A}(z, \delta)) \wedge (\texttt{true} \leftarrow \mathcal{O}(z, z')) \wedge (z \in \mathcal{Z}) \wedge (z' \neq z)\}.$$

$\mathcal{M}$ is application-specific. For instance, an APK permits adding codes or renaming class names [31, 39, 53, 76], a Windows PE permits adding codes or changing PE *section* names [5, 37, 38, 68], and a **Portable Document Format** (**PDF**) file permits appending dead-code at its end [114] or add new instructions [25, 134]. This means that a perturbation $\delta \in \mathcal{M}$ can be a tuple specifying

an operator (e.g., addition or removal), an object (e.g., a feature used by $\mathcal{I}$), and other kinds of information (e.g., perturbation location in a file).

Since it is often infeasible to enumerate the entire manipulation set, $\mathcal{A}$ may leverage an empirical one $\widetilde{\mathcal{M}}$ [31, 39, 53, 76, 97, 114], which can be defined in the problem or feature space. Manipulations in the problem space must not violate the relevant constraints (e.g., adding APIs in an APK should not cause the request of unauthorized permissions). Manipulations in the feature space facilitate efficient computing via gradient-based methods as long as the inverse feature mapping ϕ^{-1} is available. Furthermore, we can use the manipulation set $\mathcal{M}$ to define a *feature manipulation set* $\mathbf{M}$:

$$\mathbf{M} = \{\delta_{\mathbf{x}} = \mathbf{x}' - \mathbf{x} : (\mathbf{x} = \phi(z)) \wedge (\mathbf{x}' = \phi(z')) \wedge (z' \leftarrow \mathcal{A}(z, \delta)) \wedge (\delta \in \mathcal{M}) \wedge (z \in \mathcal{Z})\}. \tag{2}$$

In order to compute $\mathbf{M}$ efficiently, one strategy is to estimate a feature-space analog of $\widetilde{\mathcal{M}}$, denoted by $\widetilde{\mathbf{M}}$ [109, 114]. This, however, demands resolving the invertibility Assumption 5.

(vii) A_7: it describes $\mathcal{A}$'s attack tactics. We consider two tactics: classifier *evasion* and classifier *poisoning*. For the evasion attack, we consider three variants: **basic evasion** (**BE**), **optimal evasion 1** (**OE1**), and **optimal evasion 2** (**OE2**). For the poisoning attack, we consider two variants: **basic poisoning** (**BP**) and **optimal poisoning** (**OP**). Correspondingly, we have $A_7 \in \{\mathsf{BE}, \mathsf{OE1}, \mathsf{OE2}, \mathsf{BP}, \mathsf{OP}\}$. These tactics are elaborated below, while noting that they do not explicitly call oracle O because definitions of manipulation sets $\mathcal{M}$ already assure that manipulations preserve functionalities of non-adversarial files.

As shown in Definition 9, the BE attack is that $\mathcal{A}$ uses a set of perturbations $\delta_z \subseteq \mathcal{M}$ to manipulate a malicious file z, which is classified by $\mathcal{I}$'s classifier f as $+ \leftarrow f(z)$, to an adversarial file z' such that $- \leftarrow f(z')$.

Definition 9 (Basic Evasion or BE *[53]).* $\mathcal{A}$ looks for $\delta_z \subseteq \mathcal{M}$ to achieve the following for $z \in \mathcal{Z}$ with $+ \leftarrow f(z)$:

$$- \leftarrow f(z') \text{ where } (z' \leftarrow \mathcal{A}(z, \delta_z)) \wedge (\delta_z \subseteq \mathcal{M}).$$

As shown in Definition 10, the attacker attempts to minimize the degree of perturbations. In other words, this attack tactic is the same as BE, except that $\mathcal{A}$ attempts to minimize the manipulation when perturbing a non-adversarial file $z \in \mathcal{Z}$ into an adversarial file $z' \in \mathcal{Z}$.

Definition 10 (Optimal Evasion 1 or OE1*; Adapted from [24]).* $\mathcal{A}$ attempts to achieve the following for $z \in \mathcal{Z}$ with $+ \leftarrow f(z)$:

$$\min_{z'} \Gamma(z', z) \text{ s.t. } (z' \leftarrow \mathcal{A}(z, \delta_z)) \wedge (\delta_z \subseteq \mathcal{M}) \wedge (- \leftarrow f(z')).$$

As shown in Definition 11, the attacker attempts to maximize $\mathcal{I}$'s loss for waging high-confidence evasion attacks, while noting the small perturbations may be incorporated.

Definition 11 (Optimal Evasion 2 or OE2*; Adapted from [16]).* $\mathcal{A}$ attempts to achieve the following for $z \in \mathcal{Z}$ with $+ \leftarrow f(z)$:

$$\max_{z'} L(F_\theta(\phi_c(S, z')), +) \text{ s.t. } (z' \leftarrow \mathcal{A}(z, \delta_z)) \wedge (\delta_z \subseteq \mathcal{M}) \wedge (- \leftarrow f(z')).$$

Let $D'_{poison} \subset \mathcal{D}$ be a set of adversarial file-label pairs obtained by manipulating non-adversarial files in D_{poison}. Let $D'_{train} \leftarrow D_{train} \cup D'_{poison}$ be the contaminated training data for learning a classifier f' with parameters θ'. As shown in Definition 12, the BP attack is that the attacker aims at making f' mis-classify the files in a dataset D_{target}, while accommodating the attacks that $\mathcal{A}$ manipulates labels of the files in D_{poison} [95].

Definition 12 (Basic Poisoning or BP *[9]).* Given a set D_{target} of files where $+ \leftarrow f(\dot{z})$ for $\dot{z} \in D_{target}$ and a set D_{poison} of non-adversarial files, $\mathcal{A}$ attempts to perturb files in D_{poison} to adversarial ones $D'_{poison} = \{(\mathcal{A}(z, \delta_z), \mathcal{A}(y)) : ((z, y) \in D_{poison}) \wedge (\delta_z \subseteq \mathcal{M}) \wedge (\mathcal{A}(y) \in \{+, -\})\}$ such that classifier f' learned from $D'_{train} \leftarrow D_{train} \cup D'_{poison}$ mis-classifies the files in D_{target}. Formally, the attacker intents to achieve the following for $\forall\, \dot{z} \in D_{target}$: $- \leftarrow f'(\dot{z})$ where f' is learned from $D'_{train} \leftarrow D_{train} \cup D'_{poison}$.

As shown in Definition 13, the OP attack is the same as BF, except that $\mathcal{A}$ attempts to maximize the loss when using classifier f' with parameter θ' to classify files in D_{target}. Definition 13 can have multiple variants by considering bounds on $|D'_{poison}|$ [116] or bounds on the degree of perturbations $|\delta_z|$ [118].

Definition 13 (Optimal Poisoning or OP *[87]).* Given D_{poison}, $\mathcal{A}$ perturbs D_{poison} into D'_{poison} for achieving:

$$\max_{D'_{poison}} \mathcal{L}(\theta', D_{target}) \text{ where } \theta' \leftarrow \arg\min_{\theta} \mathcal{L}(\theta, D_{train} \cup D'_{poison}).$$

(viii) A_8: it describes $\mathcal{A}$'s attack techniques, such as **Gradient-based Optimization** (**GO**), **Sensitive Features** (**SF**), **MImicry** (**MI**), **TRansferability** (**TR**), **Heuristic Search** (**HS**), **Generative Model** (**GM**), and **Mixture Strategy** (**MS**). We denote this by $A_8 \in \{\text{GO, SF, MI, TR, HS, GM, MS}\}$. Let $\mathcal{A}$ have a classifier $\hat{f}$, which consists of a hand-crafted feature extraction $\hat{\phi}_c$ and a parameterized model $\hat{F}_{\hat{\theta}}$. Let $\mathcal{A}$ also have an objective function $L_{\mathcal{A}} : [0, 1] \times \mathcal{Y} \rightarrow \mathcal{R}$, which measures $\hat{f}$'s error or $\mathcal{A}$' failure in evasion [16, 24]. Note that $\hat{f}$ and $L_{\mathcal{A}}$ can be the same as, or can mimic (by leveraging $\mathcal{A}$'s knowledge about $\mathcal{I}$'s attributes $A_1, \ldots, A_5$), $\mathcal{I}$'s classifier f and loss function L, respectively.

The attack technique specified by Definition 14 is that $\mathcal{A}$ solves the feature-space optimization problems described in Definitions 10, 11, and 13 by using some GO method and then leverages the invertibility Assumption 5 to generate adversarial malware examples.

Definition 14 (Gradient-based Optimization or GO*, Adapted from [24, 87]).* Let $\mathbf{x} \leftarrow \hat{\phi}_c(\hat{S}, z)$ and $\mathbf{x}' \leftarrow \mathbf{x} + \delta_{\mathbf{x}}$. The feature-space optimization problem in Definition 10 can be written as

$$\min_{\delta_{\mathbf{x}}} C(\mathbf{x}, \mathbf{x} + \delta_{\mathbf{x}}) \text{ s.t. } (\delta_{\mathbf{x}} \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]) \wedge (\hat{F}_{\hat{\theta}}(\mathbf{x}') < \tau), \tag{3}$$

where $\underline{\mathbf{u}}$ and $\overline{\mathbf{u}}$ are, respectively, the lower and upper bounds on $\mathbf{M}$ (e.g., $\delta_{\mathbf{x}} \in [-\mathbf{x}, 1 - \mathbf{x}]$ for binary representation $\mathbf{x}$). The feature-space optimization problem in Definition 11 can be written as

$$\max_{\delta_{\mathbf{x}}} L_{\mathcal{A}}(\hat{F}_{\hat{\theta}}(\mathbf{x} + \delta_{\mathbf{x}}), +) \text{ s.t. } (\delta_{\mathbf{x}} \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]). \tag{4}$$

The feature-space optimization problem specified in Definition 13 can be written as

$$\max_{\delta_{\mathbf{x}} \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]} \mathbb{E}_{(\dot{z}, \dot{y}) \in D_{target}} L_{\mathcal{A}}(\hat{F}_{\hat{\theta}'}(\hat{\phi}_c(\hat{S}, \dot{z}), \dot{y})),\ \forall (z, y) \in D_{poison} \tag{5}$$

$$\text{where } \hat{\theta}' \leftarrow \arg\min_{\hat{\theta}} \mathbb{E}_{(z_t, y_t) \in \hat{D}_{train} \cup \{(\hat{\phi}_c^{-1}(\hat{\phi}_c(z) + \delta_{\mathbf{x}}), y')\}} L_{\mathcal{A}} \left(\hat{F}_{\hat{\theta}}(\hat{\phi}_c(\hat{S}, z_t), y_t)\right).$$

In order to calculate the gradients of loss function $L_{\mathcal{A}}$ with respect to $\delta_{\mathbf{x}}$ in Equations(3) and (4), inequality constraints can be handled by appending penalty items to the loss function in question and box-constraints can be coped with by using gradient projection [24, 78]. Since $\mathbf{x} + \delta_{\mathbf{x}}$ is continuous, the GO attack technique needs to map $\delta_{\mathbf{x}}$ to a discrete perturbation vector in $\mathbf{M}$, for instance by using the nearest neighbor search [78]. The gradients of loss function $L_{\mathcal{A}}$ with respect to $\delta_{\mathbf{x}}$ in Equation (5) are delicate to deal with. One issue is the indirect relation between $L_{\mathcal{A}}$ and

$\delta_{\mathbf{x}}$, which can be handled by the chain rule [73]. Another issue is the difficulty that is encountered when computing the partial derivatives $\partial\hat{\theta}'/\partial\delta_{\mathbf{x}}$ [87]. For dealing with this, researchers often relax the underlying constraints (e.g., by supposing that $\hat{F}_{\hat{\theta}}$ is a linear model).

The attack technique specified by Definition 15 is that $\mathcal{A}$ perturbs malware examples by injecting or removing a small number of features to decrease the classification error measured by the loss function $L_{\mathcal{A}}$ as much as possible.

Definition 15 (Sensitive Features or SF*, Adapted from [74]).* For evasion attacks, $\mathcal{A}$ aims at maximizing the following with respect to a given malware example-label pair $(z, +)$:

$$\max_{z'} L_{\mathcal{A}}\left(\hat{F}_{\hat{\theta}}(\hat{\phi}(\hat{S}, z')), +\right) \text{ s.t. } (z' \leftarrow \mathcal{A}(z, \delta_z)) \wedge (\delta_z \subseteq \mathcal{M}) \wedge (|\delta_z| \leq m),$$

where m is the maximum degree of manipulations.

For poisoning attacks, $\mathcal{A}$ aims at maximizing the following with respect to the given D_{poison} and D_{target},

$$\max_{D'_{poison}} \mathbb{E}_{(\dot{z},\dot{y})\in D_{target}} L_{\mathcal{A}}(\hat{F}_{\hat{\theta}'}(\hat{\phi}_c(\hat{S}, \dot{z}), \dot{y})), \tag{6}$$

where $\hat{\theta}'$ is learned from $\hat{D}_{train} \cup D'_{poison}$ such that $\forall z' \in D'_{poison}$ is obtained via the perturbation δ_z with obeying $(z' \leftarrow \mathcal{A}(z, \delta_z)) \wedge (\delta_z \subseteq \mathcal{M}) \wedge (z \in D_{poison}) \wedge (|\delta_z| \leq m)$.

The attack technique specified by Definition 16 is that $\mathcal{A}$ perturbs malware example z by mimicking a benign example, while noting that this attack technique can be algorithm-agnostic.

Definition 16 (MImicry or MI*, Adapted from [114]).* Given a set of benign examples D_{ben} and a malware example z, $\mathcal{A}$ aims at achieving the following minimization:

$$\min_{\delta_z \in \mathcal{M}} \Gamma(\mathcal{A}(z, \delta_z), z_{ben}) \text{ s.t. } (\exists\, z_{ben} \in D_{ben}). \tag{7}$$

The attack technique specified by Definition 16 can be extended to accommodate the similarity between representations in the feature space [16, 114]. The attack technique specified by Definition 17 is that $\mathcal{A}$ generates adversarial examples against a surrogate model $\hat{f}$.

Definition 17 (TRansferability or TR*, Adapted from [91]).* $\mathcal{A}$ learns a surrogate model $\hat{f}$ of f from $\hat{D}_{train}$, $\hat{S}$, $\hat{\phi}_c$ and $\hat{F}$. For evasion attacks, $\mathcal{A}$ achieves $- \leftarrow \hat{f}(z')$ by perturbing malware example z to z' and then attacks f with z'. For poisoning attacks, $\mathcal{A}$ contaminates $\hat{f}$ to $\hat{f}'$ such that $- \leftarrow \hat{f}'(\dot{z})$ for $\forall(\dot{z}, \dot{y}) \in D_{target}$, by poisoning the training set $\hat{D}_{train}$ with D'_{poison} and the attacks f with D'_{poison}.

The attack technique specified by Definition 18 is that $\mathcal{A}$ searches perturbations in $\mathcal{M}$ via some heuristics, while leveraging oracle O's responses to $\mathcal{A}$'s queries and f's responses to $\mathcal{A}$'s queries. Since $\mathcal{M}$ is defined with respect to the problem space, this attack technique does not need the invertibility Assumption 5.

Definition 18 (Heuristic Search or HS*).* Let h be a function taking O's response and f's response as input. Given a malware example z, $\mathcal{A}$ looks for an m-length manipulation path

$$\langle z_{(0)}, z_{(1)}, \ldots, z_{(m)}\rangle \text{ s.t. } z_{(i+1)} = \mathcal{A}(z_{(i)}, \delta_{z,(i)}) \wedge (\delta_{z,(i)} \in \mathcal{M}) \wedge (h(O, f, z, z_{(i)}) \leq h(O, f, z, z_{(i+1)}))$$

where $z_{(0)} = z$.

The attack technique specified by Definition 19 is that $\mathcal{A}$ uses a generative model G with parameters θ_g to perturb malware representation vectors and then leverages the invertibility Assumption 5 to turn the perturbed vector into an adversarial malware example.

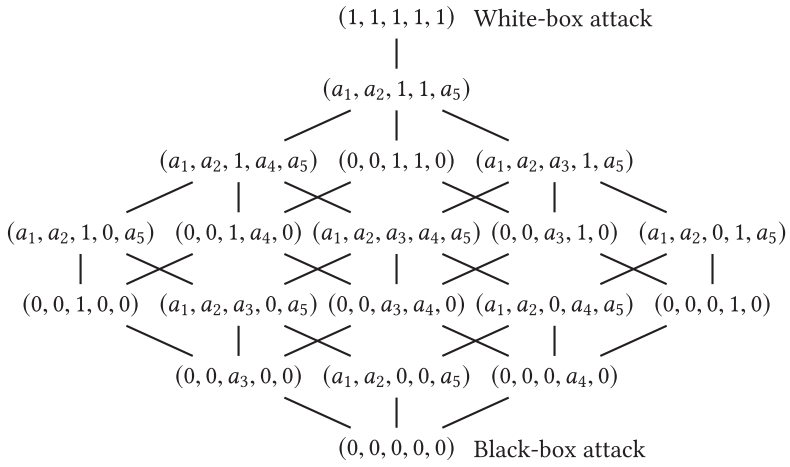

Fig. 3. A portion of the partial order defined over $(a_1, \ldots, a_5)$.

Definition 19 (Generative Model or GM*).* Given a malware representation vector $\mathbf{x} = \hat{\phi}_c(\hat{S}, z)$, $\mathcal{A}$ achieves

$$\max_{\theta_g} L_{\mathcal{A}} \left(\hat{F}_{\hat{\theta}}(G_{\theta_g}(\mathbf{x})), + \right) \text{ s.t. } G_{\theta_g}(\mathbf{x}) \in [\underline{\mathbf{u}} - \mathbf{x}, \overline{\mathbf{u}} - \mathbf{x}]$$

and leverages the invertibility Assumption 5 to obtain an adversarial example $z' = \hat{\phi}_c^{-1}(G_{\theta_g}(\mathbf{x}))$.

The attack technique specified by Definition 20 is that $\mathcal{A}$ combines multiple perturbation methods to perturb an example.

Definition 20 (Mixture Strategy or MS *[76]).* Let $\mathcal{H}_A$ denote the space of generative methods and $\mathcal{W}_a = \{\mathbf{w}_a : \mathbf{w}_a = (w_{a,1}, \ldots, w_{a,K}), w_{a,i} \geq 0\}$ with $i = 1, \ldots, K$ denote the weights space. Given a malware example z, $\mathcal{A}$ aims at achieving

$$\max_{\mathbf{w}_a} L_{\mathcal{A}}(\hat{F}_{\hat{\theta}}(\hat{\phi}(z')), +) \text{ s.t. } \left(z' = \sum_{i=1}^{K} w_{a,i} g_i(z) \right) \wedge (O(z, z') = \text{true})) \wedge (g_i \in \mathcal{H}_A) \wedge (\mathbf{w}_a \in \mathcal{W}_a).$$

(ix) A_9: it corresponds to $\mathcal{A}$'s adversarial files. Given file manipulation set $\mathcal{M}$, the corresponding set of adversarial files is defined as $\mathcal{Z}_{\mathcal{M}} = \{\mathcal{A}(z, \delta_z) : (z \in \mathcal{Z}) \wedge (\delta_z \subseteq \mathcal{M})\}$. Given feature manipulation set $\mathbf{M}$, the set of adversarial feature vectors is: $\mathcal{X}_{\mathbf{M}} = \{\mathbf{x}' : (\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}) \wedge (\delta_{\mathbf{x}} \in \mathbf{M})\}$.

On the Usefulness of the Preceding Specification. The preceding specification can be applied to formulate a partial order in the attribute space, which allows to compare attacks unambiguously. Figure 3 depicts how vector $(a_1, \ldots, a_5)$ formulates a partial order between the widely-used informal notions of *black-box* attack, namely $(a_1, a_2, a_3, a_4, a_5) = (0, 0, 0, 0, 0)$, and *white-box* attack, namely $(a_1, a_2, a_3, a_4, a_5) = (1, 1, 1, 1, 1)$; there are many kinds of gray-box attacks in between.

2.2.3 Systematizing Defenses. Similarly, we systematize defenses from two perspectives: *defender's objective* (i.e., what the defender aims at achieving) and *defender's input* (i.e., what leverages the defender can use). We also discuss how to apply the specification to formulate a partial-order structure for comparing defenses.

Defender's Objectives. $\mathcal{I}$ aims at detecting ideally all of the malicious files, adversarial and non-adversarial alike, while suffering from small side-effects (e.g., increasing false-positives).

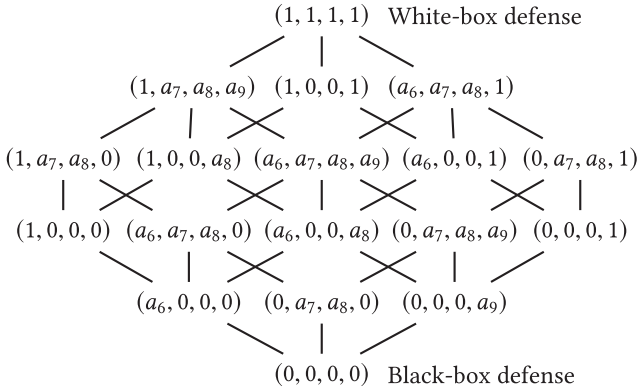

Fig. 4. A portion of the partial order defined over $(a_6, \ldots, a_9)$.

Defender's Input. As highlighted in Table 3, $\mathcal{I}$'s input includes attributes $A_1, \ldots, A_5$, which are under $\mathcal{I}$'s control, and the extent $a_6, \ldots, a_9$ at which $\mathcal{I}$, respectively, knows about attributes $A_6, \ldots, A_9$, which are under $\mathcal{A}$'s control. Note that $A_1, \ldots, A_9$ have been defined above.

- We define $a_6 \in [0, 1]$ to represent the extent at which $\mathcal{I}$ knows $\mathcal{A}$'s manipulation set $\mathcal{M}$. Let $\hat{\mathcal{M}} \subseteq \mathcal{M}$ denote the subset of $\mathcal{A}$'s manipulation set known to $\mathcal{I}$. Then, we set $a_6 = |\hat{\mathcal{M}}|/|\mathcal{M}|$.
- We define $a_7 \in \{0, 1\}$ such that $a_7 = 0$ means $\mathcal{I}$ does not know $\mathcal{A}$'s attack tactic $A_7 \in \{\text{BE}, \text{OE1}, \text{OE2}, \text{BP}, \text{OP}\}$ and $a_7 = 1$ means $\mathcal{I}$ knows $\mathcal{A}$'s tactic.
- We define $a_8 \in \{0, 1\}$ such that $a_8 = 1$ ($a_8 = 0$) means the defender does (not) know $\mathcal{I}$'s attack technique $A_8 \in \{\text{GO}, \text{SF}, \text{TR}, \text{MI}, \text{HS}, \text{GM}, \text{MS}\}$.
- We use $a_9 = |\hat{\mathcal{Z}}_{\mathcal{M}}|/|\mathcal{Z}_{\mathcal{M}}|$ to represent the extent at which $\mathcal{I}$ knows about $\mathcal{A}$'s adversarial files, where $a_9 \in [0, 1]$ and $\mathcal{Z}_{\mathcal{M}}$ is $\mathcal{A}$'s adversarial files and $\hat{\mathcal{Z}}_{\mathcal{M}} \subseteq \mathcal{Z}_{\mathcal{M}}$ is known to $\mathcal{I}$.

On the Usefulness of the Preceding Specification. Similarly, the defense specification can be used to formulate a partial order in the attribute space, paving the way for comparing defenses unambiguously. Figure 4 depicts how vector $(a_6, \ldots, a_9)$ formulates a partial order between the widely-used informal notions of black-box defense $(a_6, a_7, a_8, a_9) = (0, 0, 0, 0)$ and white-box defense $(a_6, a_7, a_8, a_9) = (1, 1, 1, 1)$; there are many kinds of gray-box defenses in between.

2.3 Systematizing Security Properties

Since $f = \varphi(\phi(\cdot))$, we decompose f's security properties into φ's and ϕ's. We consider: **Representation Robustness** (**RR**), meaning that two similar files have similar feature representations; **Classification Robustness** (**CR**), meaning that two similar feature representations lead to the same label; **Detection Robustness** (**DR**), meaning that feature extraction function ϕ returns similar representations for two files with the same functionality; **Training Robustness** (**TR**), meaning that small changes to the training set does not cause any significant change to the learned classifier. With respect to small perturbations, Definitions 21 and 22 below collectively say that when two files z and z' are similar, they would be classified as the same label with a high probability. Since the classification function φ is linear, we can obtain a ϵ-robust φ analytically, where ϵ is a small scalar that bounds the perturbations applied to feature vectors [80]. This means that the main challenge is to achieve RF extraction.

*Definition 21 (*RR *or* (ϵ, η)*-robust Feature Extraction; Adapted from [126]).* Given constants $\epsilon, \eta \in [0, 1]$, and files $z, z' \in \mathcal{Z}$ such that $(\Gamma(z, z') \approx 0) \wedge (\texttt{true} \leftarrow O(z, z'))$, we say feature extraction

function ϕ is (ϵ, η)-robust if

$$\mathbb{P}(C(\mathbf{x}, \mathbf{x}') \leq \epsilon) = \mathbb{P}(C(\phi(z), \phi(z')) \leq \epsilon) > 1 - \eta.$$

*Definition 22 (*CR *or ϵ-robust Classification [12]).* Given constant $\epsilon \in [0, 1]$ as in Definition 21 and any feature vectors $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, we say classification function φ is ϵ-robust if

$$(C(\mathbf{x}, \mathbf{x}') \leq \epsilon) \rightarrow ((\varphi(\mathbf{x}) > \tau) \wedge (\varphi(\mathbf{x}') > \tau)).$$

Definition 23 specifies DR, which says that feature extraction function ϕ returns similar representations for two different files as long as they have the same functionality. Note that Definitions 22 and 23 collectively produce a malware detector with DR.

*Definition 23 (*DR *or (O, η)-robust Feature Extraction; Adapted from [2]).* Given constant $\eta \in [0, 1]$ and two files $z, z' \in \mathcal{Z}$ such that $(\Gamma(z, z') >> 0) \wedge (\texttt{true} \leftarrow O(z, z'))$, we say feature extraction ϕ is (O, η)-robust if $\mathbb{P}(C(\phi(z), \phi(z')) \leq \epsilon) > 1 - \eta$.

Suppose, we impose a restriction on the adversarial files set D'_{poison} such that $|D'_{poison}| \leq \gamma |D_{train}|$ for some constant $\gamma \in [0, 1]$. Let classifier f' be learned from $D_{train} \cup D'_{poison}$. Definition 24 says that a classifier f' learned from poisoned training set can predict as accurately as f learned from D_{train} with a high probability.

*Definition 24 (*TR *or (γ, ζ)-robust Training; Adapted from [116]).* Given classifiers f learned from D_{train} and f' learned from $D_{train} \cup D'_{poison}$ where $|D'_{poison}| \leq \gamma |D_{train}|$, and small constants $\zeta \in [0, 1]$, we say f' is (γ, ζ)-robust if $\forall z \in \mathcal{Z}$

$$\Big((f(z) > \tau) \wedge (|D'_{poison}| \leq \gamma |D_{train}|)\Big) \rightarrow (\mathbb{P}(f'(z) > \tau) > 1 - \zeta).$$

3 SYSTEMATIZING AMD ARMS RACE

We systematize attacks according to $\mathcal{A}$'s objective, input, assumptions, the security properties that are broken, and the types of victim malware detectors (e.g., Windows vs. Android). Similarly, we systematize defenses according to $\mathcal{I}$'s objective, input, assumptions, the security properties that are achieved, and the types of enhanced malware detectors (e.g., Windows vs. Android). We group attacks (defenses) according to the attacker's (defender's) techniques and then summarize them in a table according to the publication date in chronological order. For convenience, we will use wildcard $*$ to indicate any value in a domain (e.g., $[0, 1]$); we will use $\vee$ to describe $\mathcal{A}$'s and $\mathcal{I}$'s "broader" input (if applicable). For example, $(0, 1, 0, 1, 0 | A_6, \ldots, A_9) \vee (1, 0, 1, 1, 1 | A_6, \ldots, A_9)$ means that $\mathcal{A}$ has either $(a_1, a_2, a_3, a_4, a_5) = (0, 1, 0, 1, 0)$ or $(a_1, a_2, a_3, a_4, a_5) = (1, 0, 1, 1, 1)$. Finally, we will present the attack-defense escalation.

3.1 Systematizing Attack Literature

3.1.1 Attacks Using GO. Biggio et al. [16] propose solving the problem of optimal evasion attacks by leveraging GO techniques. They focus on high-confidence evasion attacks with small perturbations (cf. Definition 11). Given a malware representation-label pair $(\mathbf{x}, y = +)$, the optimization problem specified in Equation(4) with respect to GO is instantiated as

$$\max_{\delta_{\mathbf{x}}} L_{\mathcal{A}}(\hat{F}_{\hat{\theta}}(\mathbf{x} + \delta_{\mathbf{x}}), y = +) = \min_{\delta_{\mathbf{x}}} (L(F_{\theta}(\mathbf{x} + \delta_{\mathbf{x}}, y = -)) - \beta_a \mathcal{K}(\mathbf{x} + \delta_{\mathbf{x}}))$$
$$\text{s.t. } (\delta_{\mathbf{x}} \in [\mathbf{0}, \overline{\mathbf{u}}]) \wedge (C(\mathbf{x}, \mathbf{x} + \delta_{\mathbf{x}}) \leq m),$$

where $\beta_a \geq 0$ is a balance factor and $\mathcal{K}$ is a density estimation function for lifting $\mathbf{x} + \delta_{\mathbf{x}}$ to the populated region of benign examples. Since $\delta_{\mathbf{x}} \geq \mathbf{0}$, the manipulation only permits object injections to meet the requirement of preserving malicious functionalities. The attack is validated

Table 4. Summary of AMD attacks (✓ means applicable, ● means 0, ○ means 1, and ◉ means a value in $[0, 1]$)

Attack (in chronological order)	Attack Objective			Attack Input										Assumptions					Broken Properties				Malware detector		
	Indiscriminate	Targeted	Availability	A_1: Training set D_{train}	A_2: Defense technique	A_3: Feature set	A_4: Learning algorithm	A_5: Response	A_6: Manipulation set	A_7: Attack tactic	A_8: Attack technique	A_9: Adversarial example set	IID assumption	Oracle assumption	Measurability assumption	Smoothness assumption	Invertibility assumption	RR: Representation Robustness	CR: Classification Robustness	DR: Detection Robustness	TR: Training Robustness	Windows Program	Android Package	PDF
Smutz and Stavrou [109]	✓			○	○	○	○	○	M	OE2	MI	$\mathcal{X}_M$			✓				✓					✓
Biggio et al. [16]	✓			○	○	○	○	○	M	OE2	GO	$\mathcal{X}_M$		✓	✓		✓		✓					✓
				●	●	○	◉	●																	
Maiorca et al. [82]	✓			●	●	●	●	●	$\mathcal{M}$	BE	MI	$\mathcal{Z}_M$		✓						✓				✓
Šrndić and Laskov [114]	✓			●	●	◉	●	●	M	BE	TR	$\mathcal{X}_M$												
				●	●	◉	◉	●	$\mathcal{M}$	BE	TR	$\mathcal{Z}_M$		✓			✓			✓				✓
				○	●	◉	●	●	M	OE2	TR	$\mathcal{X}_M$												
				○	●	◉	◉	●	$\mathcal{M}$	OE2	TR	$\mathcal{Z}_M$												
Xu et al. [134]	✓			●	●	●	●	○	$\mathcal{M}$	BE	HS	$\mathcal{Z}_M$								✓				✓
Carmony et al. [25]	✓			●	●	●	●	●	$\mathcal{M}$	BE	MI	$\mathcal{Z}_M$		✓						✓				✓
Hu and Tan [58]	✓			●	●	○	●	○	M	BE	GM	$\mathcal{X}_M$	✓	✓			✓			✓		✓		
Hu and Tan [59]	✓			●	●	○	●	○	M	BE	GM	$\mathcal{X}_M$	✓	✓			✓			✓		✓		
Demontis et al. [39]	✓			○	○	○	○	○	M	OE2	SF	$\mathcal{X}_M$		✓	✓		✓		✓				✓	
				●	●	○	◉	●																	
Grosse et al. [53]	✓			●	○	○	○	○	M	OE2	SF	$\mathcal{X}_M$		✓	✓		✓	✓	✓				✓	
Chen et al. [29]	✓			○	○	○	○	○	M	OE2	SF	$\mathcal{X}_M$		✓	✓		✓		✓			✓		
Khasawneh et al. [65]	✓			●	●	◉	●	○	M	BE	TR	$\mathcal{X}_M$												
									$\mathcal{M}$	BE	TR	$\mathcal{Z}_M$		✓			✓		✓			✓		
Dang et al. [36]	✓			●	●	●	●	●	$\mathcal{M}$	BE	HS	$\mathcal{Z}_M$								✓				✓
Muñoz-González et al. [87]			✓	○	○	○	○	○	M	OP	GO	$\mathcal{X}_M$		✓			✓				✓	✓		
				○	●	○	●	●																	
Yang et al. [136]	✓			●	●	●	●	○	$\mathcal{M}$	BE	HS	$\mathcal{Z}_M$		✓	✓					✓			✓	
Rosenberg et al. [100]	✓			●	●	○	●	○	M	BE	TR	$\mathcal{X}_M$												
									$\mathcal{M}$	BE	TR	$\mathcal{Z}_M$		✓			✓	✓	✓			✓		
Anderson et al. [5]	✓			●	●	●	●	○	$\mathcal{M}$	OE2	GM	$\mathcal{Z}_M$	✓	✓	✓			✓	✓			✓		
Kreuk et al. [70]	✓			●	○	○	○	○	M	OE2	GO	$\mathcal{X}_M$		✓			✓	✓	✓			✓		
Chen et al. [30]	✓			○	○	○	○	○	M	BP	SF	$\mathcal{X}_M$		✓	✓		✓				✓		✓	
				●	●	●	○	○																	
				○	●	◉	○	○																	
Al-Dujaili et al. [2]	✓			●	○	○	○	○	M	OE2	GO	$\mathcal{X}_M$		✓			✓			✓		✓		
Suciu et al. [118]		✓		◉	●	○	◉	●	M	BP	SF	$\mathcal{X}_M$												
				○	●	◉	◉	●						✓	✓		✓				✓		✓	
				○	●	○	●	●	$\mathcal{M}$	BP	SF	$\mathcal{Z}_M$												
				○	○	○	○	○																	
Kolosnjaji et al. [68]	✓			●	○	○	○	○	M	OE2	GO	$\mathcal{X}_M$		✓			✓	✓	✓			✓		
Suciu et al. [117]	✓			●	○	○	○	○	M	OE2	GO	$\mathcal{X}_M$		✓			✓	✓	✓			✓		
Chen et al. [31]	✓			●	●	○	●	●	M	OE1	GO	$\mathcal{X}_M$												
				●	●	○	●	○						✓	✓		✓		✓				✓	
				○	●	○	●	●	M	OE2	SF	$\mathcal{X}_M$												
				○	●	○	●	○																	
Pierazzi et al. [97]	✓			○	○	○	○	○	$\mathcal{M}$	OE2	MI	$\mathcal{Z}_M$		✓						✓			✓	
Li and Li [76]	✓			●	○	○	○	○	M	OE2	MS	$\mathcal{X}_M$												
									$\mathcal{M}$	OE2	MS	$\mathcal{Z}_M$		✓			✓			✓			✓	

by using the PDF malware detector and the feature representation is the number of appearances of hand-selected keywords (e.g., JavaScript). Because the perturbation is continuous, the authors suggest searching a discrete point close to the continuous one and aligning the point with $\nabla L_{\mathcal{A}}(\mathbf{x}+\delta_{\mathbf{x}}, y = +)$. This attack makes the invertibility Assumption 5 because it operates in the feature space. Experimental results show that when $\mathcal{I}$ employs no countermeasures, knowing $\mathcal{I}$'s feature set S and learning algorithm F are sufficient for $\mathcal{A}$ to evade $\mathcal{I}$'s detector. This attack and its variants have been shown to evade PDF malware detectors [17, 102, 114, 141], PE malware detectors [68], Android malware detectors [76], and Flash malware detectors [83]. The kernel density estimation item makes the perturbed representation $\mathbf{x} + \delta_{\mathbf{x}}$ similar to the representations of benign examples,

explaining the successful evasion. In summary, the attack works under the Oracle, Measurability, and Invertibility assumptions. $\mathcal{A}$'s input is, or $\mathcal{A}$ can be characterized as, $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, *, 0|\mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_\mathbf{M})$ and $\mathcal{A}$ breaks the CR property.

Al-Dujaili et al. [2] propose evasion attacks against DNN-based malware detectors in the feature space. In this attack, $\mathcal{A}$ generates adversarial examples with possibly large perturbations in the feature space. More precisely, given a representation-label pair $(\mathbf{x}, y = +)$, the optimization problem of Equation (4) with respect to GO is instantiated as: $\max_{\delta_\mathbf{x}} L(F_\theta(\mathbf{x} + \delta_\mathbf{x}), y = +)$ s.t. $\delta_\mathbf{x} \in [\mathbf{0}, \mathbf{1} - \mathbf{x}]$. The attack has four variants, with each perturbing the representation in a different direction (e.g., normalized gradient of the loss function using the ℓ_∞ norm). A "random" rounding operation is used to map continuous perturbations into a discrete domain. When compared with the basic rounding (which returns 0 if the input is smaller than 0.5, and returns 1 otherwise), the "random" rounding means that the threshold of rounding is sampled from the interval $[0, 1]$ uniformly. For binary feature representation, the manipulation set $\mathbf{M_x} = [\mathbf{0}, \mathbf{1} - \mathbf{x}]$ assures the flipping of 0–1. The effectiveness of the attack is validated using Windows malware detector in the feature space. In summary, the attack works under the Oracle and Invertibility assumptions with $\mathcal{A}$ input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_\mathbf{M})$ and breaks the DR property.

Kreuk et al. [70] propose an evasion attack in the feature space against MalConv, which is an end-to-end Windows PE malware detector (as reviewed in Section 2.1) [99]. Given a malware embedding code $\mathbf{x}$, the optimization problem of Equation (4) with respect to GO is instantiated as: $\min_{\delta_\mathbf{x}} L(F_\theta([\mathbf{x}|\delta_\mathbf{x}]), y = -)$ s.t. $\|\delta_\mathbf{x}\|_p \leq \epsilon$, where $|$ means concatenation and $\|\cdot\|_p$ is the p norm where $p \geq 1$. Because MalConv is learned from sequential data, perturbation means appending some content to the end of a PE file. Perturbations are generated in a single step by following the direction of the ℓ_∞ or ℓ_2 normalized gradients of the loss function [51, 71]. For instance, the attack based on the ℓ_∞ norm is $\tilde{\mathbf{x}}' = \mathbf{x} - \epsilon \cdot \text{sign}(\nabla_\mathbf{x}(L(F_\theta(\mathbf{x}), -))$, where $\text{sign}(x) = +1$ (-1) if $x \geq 0$ $(x < 0)$. Since the embedding operation uses a look-up table to map discrete values (0, 1, ..., 255) to the learned real-value vectors, the attack uses a nearest neighbor search to look for the learned embedding code close to $\tilde{\mathbf{x}}'$. In summary, the attack works under the Oracle and Invertibility assumptions with input $(a_1, \ldots, a_5|\ A_6, \ldots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_\mathbf{M})$ and breaks the RR and CR properties.

Kolosnjaji et al. [68] and Suciu et al. [117] independently propose gradient-based attacks in the feature space to evade MalConv [99]. Both studies also use the loss function exploited by Kreuk et al. [70]. Kolosnjaji et al. [68] use the manipulation set $\mathcal{M}$ corresponding to appending instructions at the end of a file. This attack proceeds iteratively and starts with randomly initialized perturbations. In each iteration, continuous perturbations are updated in the direction of the ℓ_2 normalized gradient of the loss function with respect to the input, and then a nearest neighbor search is applied to obtain discrete perturbations. Suciu et al. [117] perturb embedding codes in the direction of the ℓ_∞ normalized gradient of the loss function, while adding instructions in the mid of a PE file (e.g., between PE *sections*) while noting that appended content could be truncated by MalConv. Both attacks work under the Oracle and Invertibility assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_\mathbf{M})$ and break the RR and CR properties.

Muñoz-González et al. [87] propose the OP attack in the feature space (Definition 13), which is NP-hard. In this case, the optimization problem of Equation (5) is relaxed by supposing that the classifier is linear to render the optimization problem tractable [20, 87, 132]. The attack is waged against Windows PE malware detectors. Feature set includes API calls, actions, and modifications in the file system; each file is represented by a binary vector. The attack has two variants: one uses white-box input, where $\mathcal{A}$ derives D'_{poison} from $\mathcal{I}$'s detector f; the other uses gray-box input, where $\mathcal{A}$ knows $\mathcal{I}$'s training set as well as feature set and trains a surrogate detector. The

attack works under the Oracle and Invertibility assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \text{OP}, \text{GO}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0|\mathbf{M}, \text{OP}, \text{GO}, \mathcal{X}_\mathbf{M})$ and breaks TR.

3.1.2 Attacks Using SF. Demontis et al. [39] propose the OE2 in the feature space to perturb important features in terms of their weights in the linear function $\varphi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, where $\mathbf{w} = (w_1, w_2, \ldots, w_d)$ is a weight vector, b is the bias, and d is the dimension of feature space. The attack is waged against Drebin malware detector (which is reviewed in Section 2.1). $\mathcal{A}$ manipulates the x_i's with largest $|w_i|$'s as follows: flip $x_i = 1$ to $x_i = 0$ if $w_i > 0$, flip $x_i = 0$ to $x_i = 1$ if $w_i < 0$, and do nothing otherwise, while obeying the manipulation set $\mathbf{M}$ corresponding to the injection or removal of features. The attack works under the Oracle, Measurability, and Invertiblity assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, *, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$ and breaks the CR property.

Grosse et al. [53] propose a variant of the **Jacobian-based Saliency Map Attack** (**JSMA**) [92] in the feature space against the Drebin malware detector (which is reviewed in Section 2.1). Instead of using SVM, **Deep Neural Network** (**DNN**) is used to build a detector. Important features are identified by leveraging the gradients of the softmax output of a malware example with respect to the input. A large gradient value indicates a high important feature. $\mathcal{A}$ only injects manifest features to manipulate Android Packages and generates adversarial files from $\mathcal{I}$'s detector f. The attack works under the Oracle, Measurability, and Invertiblity assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$ and breaks the RR and CR properties.

Chen et al. [29] propose an evasion attack in the feature space by perturbing the important features derived from a wrapper-based feature selection algorithm [27, 29, 96, 141]. The attacker's loss function $L_\mathcal{A}$ has two parts: (i) the classification error in the mean squared loss and (ii) the manipulation cost $C(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} c_i |x_i - x_i'|$, where $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{x}' = (x_1', \ldots, x_d')$, and c_i is the hardness of perturbing the ith feature while preserving malware's functionality. The attack is waged against Windows PE malware detector that uses hand-crafted Windows API calls as features and the binary feature representation. However, there are no details about the composition of manipulation set. This attack works under the Oracle, Measurability, and Invertibility assumptions with input $(a_1, \ldots, a_5|A_7, \ldots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}))$ and breaks CR.

Chen et al. [31] propose evasion attacks in the feature space against two Android malware detectors, MaMaDroid [84], and Drebin [8]. The manipulation set $\mathbf{M}$ corresponds to the injection of manifest features (e.g., *activities*) and API calls. $\mathcal{A}$ evades MaMaDroid by using the OE1 (Definition 10) and OE2 (Definition 11), and evades Drebin by using OE2. The optimization problem of OE1 Equation (3) is solved using an advanced gradient-based method known as C&W [24]. OE2 is solved using JSMA [92]. Because JSMA perturbs SF, we categorize this attack into the SF group. The OE2 attack works under the Oracle, Measurability, and Invertibility assumptions, with four kinds of input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 1, 0, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, 0, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$, and breaks the CR property. The OE1 attack works under the same assumptions with the same input except using attack technique GO, and breaks the CR property.

Chen et al. [30] propose a BP attack in the feature space against Android malware detectors. The feature set contains syntax features (e.g., permission, hardware, and API) and semantic features (e.g., sequence of pre-determined program behaviors such as `getDevicedID`→ URL→`openConnection`). The ML algorithm used is SVM, random forest, or K-Nearest Neighbor (KNN) [22, 107]. The malware representations are perturbed using a JSMA variant [92] against the SVM-based classifier (while noting JSMA is applicable neither to random forests nor to KNN because they are gradient-free). Feature manipulation set $\mathbf{M}$ corresponds to the injection of syntax features. $\mathcal{A}$ poisons $\mathcal{I}$'s training set by injecting perturbed perturbations

with label −. The attack works under the Oracle, Measurability, and Invertibility assumptions with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 0, 1, 1 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, *, 1, 1 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M})$ and breaks TR.

Suciu et al. [118] propose a BP attack in both feature and problem spaces. The authors obtain D'_{poison} by applying small manipulations to non-adversarial benign files and then obtain their labels as given by VirusTotal service [108]. $\mathcal{A}$'s objective is to make $\mathcal{I}$'s classifier f mis-classify a targeted malware file z_{mal} as benign. $\mathcal{A}$ proceeds as follow: (i) obtain an initial benign file z_{ben}, where $z_{ben} \approx z_{mal}$ in the feature space with respect to the ℓ_1 norm; (ii) use the JSMA method [92] to manipulate z_{ben} to z'_{ben} by making a small perturbation so that they have similar feature representations; (iii) add z'_{ben} and its label obtained from VirusTotal to D'_{poison} and use $D_{train} \cup D'_{poison}$ to train classifier f' (Definition 15); (iv) undo the addition if z'_{ben} lowers the classification accuracy significantly, and accept it otherwise. The attack is waged against the Drebin malware detector and the manipulation set corresponds to the feature injection of permission, API, and strings. This attack works under the Oracle, Measurability, and Inversiability assumptions with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (*, 0, 1, *, 0 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 1, 1, 1, 1 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, *, *, 0 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_\mathbf{M})$ and breaks the TR property. The study generates adversarial malware examples, but does not test their malicious functionalities.

3.1.3 Attacks Using MI. Smutz and Stavrou [109] propose a mimicry attack in the feature space to modify features of a malicious file to mimic benign ones, where $\mathcal{A}$ knows $\mathcal{I}$'s classifier f. Discriminative features are identified by observing their impact on classification accuracy. The attack perturbs features of malware examples by replacing their value with the mean of the benign examples. The attack is leveraged to estimate the robustness of PDF malware detectors without considering the preservation of malware functionality. The attack works under the Measurability assumption with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \text{OE2}, \text{MI}, \mathcal{X})$ and breaks the CR property.

Maiorca et al. [82] propose a reverse mimicry attack against PDF malware detectors in the problem space. Instead of modifying malicious files to mimic benign ones, $\mathcal{A}$ embeds malicious payload (e.g., JavaScript code) into a benign file. The attack can be enhanced by using parser confusion strategies, which make the injected objects being neglected by feature extractors when rendered by PDF readers [25]. The attack works under the Oracle assumption with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (0, 0, 0, 0, 0 | \mathcal{M}, \text{BE}, \text{MI}, \mathcal{Z}_\mathcal{M})$ and breaks the DR property.

Pierazzi et al. [97] propose a white-box evasion attack against the Drebin malware detector and then an enhanced version of the detector in the problem space [8, 39]. They intend to bridge the gap between the attacks in the problem space and the attacks in the feature space. In addition, four realistic constraints are imposed on the manipulation set $\mathcal{M}$, including available transformation, preserved semantics, robustness to preprocessing, and plausibility. In order to cope with the side-effect features when incorporating gradient information of $\mathcal{I}$'s classifier, the attacker first harvests a set of manipulations from benign files; Manipulations in the problem space are used to query $\mathcal{I}$'s feature extraction for obtaining perturbations in the feature space; an adversarial malware example is obtained by using the manipulations corresponding to the perturbations that have a high impact on the classification accuracy. This attack works under the Oracle assumption with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 1, 1, 1, 1 | \mathcal{M}, \text{OE2}, \text{MI}, \mathcal{Z}_\mathcal{M})$ and breaks the DR property.

3.1.4 Attacks Using TR. Šrndić and Laskov [114] investigate the mimicry attack and the aforementioned gradient descent and kernel density estimation attack against the PDFrate service, where $\mathcal{A}$ knows some features used by $\mathcal{I}$. $\mathcal{A}$ makes the representation of an adversarial malware example similar to a benign one. Manipulation set $\mathcal{M}$ corresponds to adding objects into PDF files.

Both attacks perturb feature vectors against a surrogate model, and then map the perturbed feature representations to the problem space by injecting manipulations between the body and the trailer of a PDF file. For the mimicry attack, $\mathcal{A}$ uses $N_{ben} > 0$ benign examples to guide manipulations, resulting in N_{ben} perturbed examples. The example incurring the highest classification error is used as an adversarial example. The attack works under the Oracle and Invertibility assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, *, 0, 0|\mathbf{M}, \text{BE}, \text{TR}, \mathcal{X}_\mathbf{M}) \vee (0, 0, *, *, 0|\mathbf{M}, \text{BE}, \text{TR}, \mathcal{X}_\mathbf{M}) \vee (1, 0, *, 0, 0|\mathbf{M}, \text{BE}, \text{TR}, \mathcal{X}_\mathbf{M}) \vee (1, 0, *, *, 0|\mathbf{M}, \text{BE}, \text{TR}, \mathcal{X}_\mathbf{M}) \vee (0, 0, *, 0, 0|\mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_\mathcal{M}) \vee (0, 0, *, *, 0|\mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_\mathcal{M}) \vee (1, 0, *, 0, 0|\mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_\mathcal{M}) \vee (1, 0, *, *, 0|\mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_\mathcal{M})$ and breaks DR. The gradient-based attack neglects the constraint of small perturbations and works under the same assumptions with the same input except using the attack technique OE2.

Khasawneh et al. [65] propose an evasion attack in both the feature and problem spaces against malware detectors learned from dynamic hardware features (e.g., instruction frequency), where $\mathcal{A}$ knows some features used by $\mathcal{I}$. The attack proceeds as follows. $\mathcal{A}$ first queries $\mathcal{I}$'s classifier to obtain a surrogate model and then generates adversarial files against the surrogate model. Manipulation set $\mathcal{M}$ corresponds to the injection of some features because the others (e.g., memory access) are uncontrollable. Perturbations are conducted to the important features that are identified by large weights in the model. The attack works under the Oracle and Invertibility assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, *, 0, 1|\mathbf{M}, \text{BE}, \text{TR}, \mathcal{X}_\mathbf{M}) \vee (0, 0, *, 0, 1|\mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_\mathcal{M})$ and breaks the CR property.

Rosenberg et al. [100] propose an evasion attack in both the feature and problem spaces against a **Recurrent Neural Network** (**RNN**) based surrogate model, which is learned from API call sequences. In this attack, $\mathcal{A}$'s training data is different from $\mathcal{I}$'s, but the labels are obtained by querying $\mathcal{I}$'s detector. In order to reduce the number of queries to $\mathcal{I}$'s detector, $\mathcal{A}$ augments its training data using the Jacobian-based augmentation technique [91] and modifies the API sequence of an example in the direction of the ℓ_∞ normalized gradient of the loss function. Manipulation set $\mathbf{M}$ corresponds to inserting no-op API calls. Experimental results show that adversarial examples generated from a surrogate RNN model can evade SVM, DNN, and RNN detectors. The attack works under the Oracle and Invertibility assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 1, 0, 1|\mathbf{M}, \text{BE}, \text{TR}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, 0, 1|\mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_\mathcal{M})$ and breaks the RR and CR properties.

3.1.5 Attacks Using HS. Xu et al. [134] propose black-box evasion attacks in the problem space against two PDF malware detectors known as PDFrate [109] and Hidost [115], respectively. Given a malicious file z, $\mathcal{A}$ uses a genetic algorithm to iteratively generate z' from z as follows: (i) $\mathcal{A}$ manipulates a set of candidates (or z in the initial iteration) via object deletion, insertion, or replacement. (ii) $\mathcal{A}$ queries these variants to O and f. (iii) $\mathcal{A}$ succeeds when obtaining a successful adversarial example z', namely $(\texttt{True} \leftarrow O(z, z')) \wedge (- \leftarrow f(z'))$; otherwise, $\mathcal{A}$ uses a score function to select candidates for the next iteration or aborts after reaching a threshold number of iterations. The score function h varies with classifiers; for PDFrate, $h(O, f, z, z') = 0.5 - f(z')$ if $O(z, z') = \text{true}$, and returns -0.5 if $O(z, z') = \text{false}$. This attack models an Oracle and works with the input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 0, 0, 1|\mathcal{M}, \text{BE}, \text{HS}, \mathcal{Z}_\mathcal{M})$ and breaks the DR property.

Yang et al. [136] propose evasion attacks against Android malware detectors in the problem space. In this attack, $\mathcal{A}$ also uses a genetic algorithm to perturb a malware example z iteratively. In each iteration, $\mathcal{A}$ extracts some features and calculates similarity scores between the malicious APKs in the feature space; the features that have high impact on the similarity scores are selected; the manipulations are to perturb the selected features. The attack works under the Oracle and Measurability assumptions with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 0, 0, 1|\mathcal{M}, \mathcal{Z}, \text{BE}, \text{HS}, \mathcal{Z}_\mathcal{M})$ and breaks the DR property.

Dang et al. [36] propose a black-box evasion attack against malware detectors (e.g., PDFrate) in the problem space. Given a malicious file z, $\mathcal{A}$ uses the hill-climbing algorithm to iteratively generate adversarial file z' from z. In each iteration, $\mathcal{A}$ generates a path of variants sequentially, each of which is perturbed from its predecessor using manipulations corresponding to object deletion, insertion, or replacement. A score function h is leveraged to select candidates, such as $h(O, f, z, z') = \text{mal}_{z'} - \text{clf}_{z'}$ or $h(O, f, z, z') = \text{mal}_{z'}/\text{clf}_{z'}$, where $\text{mal}_{z'}$ denotes the length of the first example turned from malicious to benign (obtaining by using an oracle) on the manipulation path (cf. Definition 18) and $\text{clf}_{z'}$ denotes the length of the first malware example that has successfully misled the classifier f. Both examples of interest are obtained by a binary search, effectively reducing the number of queries to oracle O and f. The attack models an Oracle and works with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (0, 0, 0, 0, 0 | \mathcal{M}, \text{BE}, \text{HS}, \mathcal{Z}_{\mathcal{M}})$ and breaks the DR property.

3.1.6 Attacks Using GM. Hu and Tan [58] propose an evasion attack against Windows malware detectors in the feature space, by using **Generative Adversarial Networks** (**GAN**) [50]. In this attack, $\mathcal{A}$ modifies the binary representation of Windows API calls made by malicious files, namely flipping some feature values from 0 to 1. $\mathcal{A}$ learns a generator G_{θ_g} and a discriminator from $\mathcal{A}$'s training dataset. The discriminator is a surrogate detector learned from feature vectors corresponding to $\mathcal{A}$'s benign files and those produced by G_{θ_g}, along with labels obtained by querying $\mathcal{I}$'s detector f. An adversarial example feature vector is generated by using $\mathbf{x}' = \max(\mathbf{x}, \text{round}(G_{\theta_g}(\mathbf{x}, \mathbf{a})))$, where $\mathbf{a}$ is a vector of noises, round is the round function, and max means element-wise maximum. Hu and Tan [59] also propose another evasion attack using the Seq2Seq model [33]. Both attacks work under the IID, Oracle, and Invertibility assumptions with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (0, 0, 1, 0, 1 | \mathbf{M}, \text{BE}, \text{GM}, \mathcal{X}_{\mathbf{M}})$ and break the DR property.

Anderson et al. [5] propose a **Reinforcement Learning** (**RL**)-based evasion attack against Windows PE malware detectors in the problem space. Manipulation set $\mathcal{M}$ is the RL action space, which includes some bytecode injections (e.g., API insertion) and some bytecode deletion. Attacker $\mathcal{A}$ learns an RL agent on $\mathcal{A}$'s data, with labels obtained by querying defender $\mathcal{I}$'s detector f. The learned agent predicts manipulations sequentially for a given malware example. Moreover, $\mathcal{A}$ is restricted by only applying a small number of manipulations to a malicious PE file. Experimental results show that the attack is not as effective as others (e.g., gradient-based methods). The attack works under the Oracle and Measurability assumptions with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (0, 0, 0, 0, 1 | \mathcal{M}, \text{OE2}, \text{GM}, \mathcal{Z}_{\mathcal{M}})$ and breaks the RR and CR properties.

3.1.7 Attacks Using MS. Li and Li [76] propose evasion attacks against DNN-based Android malware detectors in both feature and problem spaces. Given four gradient-based attack methods, the attack looks for the best one to perturb malware representations. $\mathcal{A}$ can iteratively perform this strategy to modify the example obtained in the previous iteration. Experimental results show that the mixture of attacks can evade malware detectors effectively. The attack works under the IID, Oracle, and Invertibility assumptions with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \text{BE}, \text{MS}, \mathcal{X}_{\mathbf{M}}) \vee (1, 1, 1, 1, 1 | \mathcal{M}, \text{BE}, \text{MS}, \mathcal{Z}_{\mathcal{M}})$ and breaks the DR property.

3.1.8 Drawing Observations and Insights. We summarize the preceding reviews with the following observations. (i) Indiscriminate attacks have been much more extensively investigated than targeted attacks and availability attacks. (ii) Evasion attacks have been much more extensively studied than poisoning attacks. (iii) The Oracle assumption has been widely made. In addition, we draw the following insights.

Insight 1. *(i) Knowing the defender's feature set is critical to the success of transfer attacks, highlighting the importance of keeping the defender's feature set secret (e.g., randomizing the defender's feature set). (ii) The effectiveness of evasion attacks largely depends on the attacker's degree of*

Table 5. Summary of AMD Defenses (✓ Means Applicable, ● Means 0, ○ Means 1, and ◉ Means a Value in [0, 1])

Defense (in chronological order)	Defense Objective	Defense Input									Assumptions					Achieved Properties				Malware Detector		
	malware detection	A_1: Training set D_{train}	A_2: Defense technique	A_3: Feature set	A_4: Learning algorithm	A_5: Response	A_6: Manipulation set	A_7: Attack tactic	A_8: Attack technique	A_9: Adversarial example set	IID assumption	Oracle assumption	Measurability assumption	Smoothness assumption	Invertibility assumption	RR: Representation Robustness	CR: Classification Robustness	DR: Detection Robustness	TR: Training Robustness	Windows Program	Android Package	PDF
Biggio et al. [15]	✓	D_{train}	EL	S	F_θ	FQ	●	○	●	●	✓						✓					✓
Smutz and Stavrou [110]	✓	D_{train}	SE	S	F_θ	FQ	●	○	●	●	✓							✓				✓
Zhang et al. [141]	✓	D_{train}	RF	S	F_θ	FQ	○	○	●	●	✓		✓	✓		✓	✓					✓
Demontis et al. [39]	✓	D_{train}	WR	S	F_θ	FQ	○	○	●	●	✓		✓				✓				✓	
Wang et al. [127]	✓	D_{train}	IT	S	F_θ	FQ	●	○	●	●	✓						✓			✓		
Grosse et al. [53]	✓	D_{train}	WR	S	F_θ	FQ	●	○	●	●	✓					✓	✓				✓	
Grosse et al. [53]	✓	D_{train}	AT	S	F_θ	FQ	○	○	○	●	✓		✓			✓	✓				✓	
Chen et al. [29]	✓	D_{train}	AT	S	F_θ	FQ	○	○	○	●	✓		✓				✓			✓		
Khasawneh et al. [65]	✓	D_{train}	CD	S	F_θ	FQ	●	○	○	●	✓						✓			✓		
Dang et al. [36]	✓	D_{train}	SE	S	F_θ	LQ	●	○	●	●	✓							✓				✓
Yang et al. [136]	✓	D^*_{train}	AT	S	F_θ	FQ	●	○	●	◉	✓							✓			✓	
Yang et al. [136]	✓	D_{train}	SE	S	F_θ	FQ	○	○	○	●	✓							✓			✓	
Chen et al. [27]	✓	D_{train}	RF	S	F_θ	FQ	○	○	●	●	✓		✓	✓		✓	✓				✓	
Incer et al. [61]	✓	D_{train}	VL	S	F_θ	FQ	○	○	●	●	✓					✓	✓	✓		✓		
Chen et al. [30]	✓	D_{train}	SE	S	F_θ	FQ	●	○	●	●			✓						✓		✓	
Al-Dujaili et al. [2]	✓	D_{train}	AT	S	F_θ	FQ	○	○	●	●	✓							✓		✓		
Chen et al. [28]	✓	D_{train}	IT	S	F_θ	FQ	●	○	●	●	✓						✓				✓	
Jordan et al. [62]	✓	D_{train}	RF	S	F_θ	FQ	○	○	●	●						✓	✓	✓				✓
Li et al. [77]	✓	D_{train}	AT	S	F_θ	FQ	●	○	●	●	✓		✓	✓		✓	✓			✓		
Tong et al. [122]	✓	D_{train}	RF	S	F_θ	FQ	○	○	●	●	✓					✓	✓	✓				✓
Li and Li [76]	✓	D_{train}	AT	S	F_θ	FQ	○	○	●	●	✓							✓			✓	
Chen et al. [32]	✓	D_{train}	VL	S	F_θ	FQ	○	○	●	●	✓		✓	✓		✓	✓					✓
Li et al. [78]	✓	D_{train}	EL+AT+RF	S	F_θ	FQ	○	○	●	●	✓		✓					✓			✓	

D^*_{train} contains D_{train} and a portion of $\mathcal{A}$'s adversarial examples.

freedom in conducting manipulations in the problem space (i.e., a smaller degree of freedom means it is harder for the attack to succeed).

3.2 Systematizing Defense Literature

3.2.1 Defenses Using EL. Biggio et al. [15] propose a one-and-a-half-class SVM classifier against evasion attacks, by leveraging an interesting observation (i.e., decision boundaries of one-class SVM classifiers are tighter than that of two-class SVM classifiers) to facilitate outlier detection. Specifically, the authors propose an ensemble of a two-class classifier and two one-class classifiers, and then combine them using another one-class classifier. The defense can enhance PDF malware detectors against gradient-based attacks [16], which can be characterized as $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \text{OP2}, \text{GO}, \mathcal{X}_\mathbf{M})$. However, the defense cannot thwart attacks incurring large perturbations. Independent of this study, other researchers propose using the random subspace and bagging techniques to enhance SVM-based malware detectors, dubbed **Multiple Classifier System SVM** (**MCS-SVM**), which leads to evenly distributed weights [18, 39]. These defenses work under the IID assumption with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{EL}, S, F_\theta, \text{FQ} | 0, 1, 0, 0)$ and achieves the CR property.

3.2.2 Defenses Using WR. Demontis et al. [39] propose enhancing the Drebin malware detector $\varphi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ by using box-constraint weights. The inspiration is that the classifier's sensitivity to

perturbations based on the ℓ_1 norm is bounded by the ℓ_∞ norm of the weights. This defense hardens the Drebin detector against a mimicry attack with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 1, 0, 0|\mathbf{M}, \text{BE}, \text{MI}, \mathcal{X}_\mathbf{M})$, obfuscation attack [41] with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 0, 0, 0|\mathcal{M}, \text{BE}, -, \mathcal{Z}_\mathcal{M})$, and the attack that modifies important features [39] with input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$, here "$-$" means inapplicable. Experimental results show this defense outperforms MSC-SVM [18]. The defense works under the IID, Oracle, and Measurability assumptions with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \text{WR}, S, F_\theta, \text{FQ}|1, 1, 0, 0)$ and achieves CR.

Grosse et al. [53] investigate how to apply two defense techniques known as distillation [93] and retraining [120] to enhance the DNN-based Drebin malware detector. The distillation technique can decrease a model's generalization error by leveraging a teacher to relabel the training data represented by real-value vectors (rather than one-hot encoding). It uses retraining to tune a learned model with respect to an augmented training set with adversarial examples. Both defenses are estimated against a variant of JSMA and can be characterized by their input as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (0, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$. Experimental results show the two defenses achieve limited success. The defense based on the distillation technique works under the IID assumption with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \text{WR}, S, F_\theta, \text{FQ}|0, 1, 0, 0)$ and achieves the RR and CR properties. The defense based on the retraining technique works under the IID and Measurability assumptions with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \text{AT}, S, F_\theta, \text{FQ}|1, 1, 1, 0)$ and achieves the RR and CR properties.

3.2.3 Defenses Using AT. Chen et al. [29] adapt a generic retraining framework proposed in the AML context [75] to enhance linear malware detectors. The defense uses a label smoothness regularization technique to mitigate the side-effect of AT [135]. The defense is evaluated using Windows malware detectors against "feature selection"-based evasion attacks, which can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (1, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$. The defense works under the IID and Measurability assumptions and can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \text{AT}, S, F_\theta, \text{FQ}|1, 1, 1, 0)$, while assuring the CR property.

Yang et al. [136] propose a defense against genetic algorithm-based evasion attacks that can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (0, 0, 0, 0, 1|\mathcal{M}, \text{BE}, \text{HS}, \mathcal{Z}_\mathcal{M})$. The defense uses three techniques: AT, SE, and WR [39]. The AT uses one half of $\mathcal{A}$'s adversarial examples. The defense works under the IID assumption and can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D'_{train}, \text{AT}, S, F_\theta, \text{FQ}|0, 1, 0, *)$, where D'_{train} is the union of D_{train} and a portion (e.g., one half) of $\mathcal{A}$'s adversarial examples. The defense of SE is learned from manipulations used by the attacker and works under the IID assumption with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \text{SE}, S, F_\theta, \text{FQ}|1, 1, 1, 0)$. Both defenses achieve the DR property. The defense of wight regularization is reviewed in Section 3.2.2.

Al-Dujaili et al. [2] adapt the idea of minmax AT (proposed in the AML context) to enhance DNN-based malware detectors. In this defense, the inner-layer optimization generates adversarial files by maximizing the classifier's loss function; the outer-layer optimization searches for the parameters θ (of DNN F_θ) that minimize the classifier's loss with respect to the adversarial files. The defense enhances Windows malware detectors against attacks with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (0, 1, 1, 1, 1|\mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_\mathbf{M})$. Experimental results show that malware detectors that are hardened to resist one attack may not be able to defend against other attacks. By observing this phenomenon, researchers propose using a mixture of attacks to harden DNN-based malware detectors [76]. The defense works under the IID assumption and can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \text{AT}, S, F_\theta, \text{FQ}|1, 1, 0, 0)$. The defense assures the DR property.

Li et al. [77] propose a DNN-based attack-agnostic framework to enhance adversarial malware detectors. The key idea is dubbed adversarial regularization, which enhances malware detectors via the (approximately) optimally small perturbation. The framework wins the AICS'2019 adversarial malware classification challenge organized by MIT Lincoln Lab researcher [131], without knowing anything about the attack. The defense works under the IID, Measurability, and Smoothness assumptions with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{AT}, S, F_\theta, \mathsf{FQ}|0, 1, 0, 0)$ and assures the RR and CR properties. In the extended study [78], the authors further enhance the framework with six defense principles, including EL, AT, and RR learning. The enhanced defense is validated with 20 attacks (including 11 gray-box attacks and 9 white-box attacks) against Android malware detectors. The enhanced defense works under the IID and Measurability assumptions with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{AT} + \mathsf{EL} + \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$ and assures the DR property.

*3.2.4 Defenses Using Verifiable Learning (*VL*).* Incer et al. [61] propose using monotonic malware classifiers to defend against evasion attacks, where *monotonic* means $\varphi(\mathbf{x}) \leq \varphi(\mathbf{x}')$ when $\mathbf{x} \leq \mathbf{x}'$ [54]. Technically, this can be achieved by using (i) RFs that can only be removed or added but not both and (ii) monotonic classification function (e.g., linear models with non-negative weights). The resulting classifier can thwart any attack that perturbs feature values monotonically. The defense works under the IID assumption with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{VL}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$ and assures the RR, CR, and DR properties.

Chen et al. [32] propose a defense to enhance PDF malware detectors against evasion attacks, by leveraging the observation that manipulations on PDF files are subtree additions and/or removals. They also propose new metrics for quantifying such structural perturbations. This allows to adapt the *symbolic interval analysis* technique proposed in the AML context [128] to enhance the PDF malware detectors. The defense can cope with attacks leveraging small perturbations in the training phase. This defense works under the IID, Measurability, and Smoothness assumptions with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{VL}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$ and achieves the RR and CR properties.

3.2.5 Defenses Using RF*.* Zhang et al. [141] propose leveraging optimal adversarial attacks for feature selection. The defense enhances PDF malware detectors against gradient-based attacks, which can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathsf{OE2}, \mathsf{GO}, \mathcal{X}_\mathbf{M})$. The defense works under the IID, Measurability, and Smoothness assumptions and can be characterized as $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$. The defense assures the RR and CR properties.

Tong et al. [122] propose refining features into invariant ones to defend against genetic algorithm-based evasion attacks with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (0, 0, 0, 0, 1|\mathcal{M}, \mathsf{BE}, \mathsf{HS}, \mathcal{Z}_\mathcal{M})$. Experimental results show that AT can be further leveraged to enhance the robustness of the defense. The defense works under IID assumption with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$, and achieves RR, CR, and DR.

Chen et al. [27] propose mitigating evasive attacks by filtering features according to their importance $|w_i|/c_i$ with respect to the linear function $\varphi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, where x_i, w_i, and c_i denote, respectively, the ith component of $\mathbf{x}$, $\mathbf{w}$, and the constraint on manipulation cost $\mathbf{c}$. The defense enhances Android malware detectors against three attacks: a random attack with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (0, 0, 1, 0, 0|\mathbf{M}, \mathsf{BE}, -, \mathcal{X}_\mathbf{M})$, a variant of the mimicry attack with input $(0, 0, 1, 0, 0|\mathbf{M}, \mathsf{BE}, \mathsf{MI}, \mathcal{X}_\mathbf{M})$, and the attack that modifies important features with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathsf{BE}, \mathsf{SF}, \mathcal{X}_\mathbf{M})$, where "–" means inapplicable. The defense works under the IID, Measurability, and Smoothness assumptions with input $(A_1, \ldots, A_5|a_6, \ldots, a_9) = (D_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$ and achieves RR and CR.

Jordan et al. [62] propose a robust PDF malware detector against evasion attacks by interpreting JavaScript behaviors using static analysis. A PDF file is classified as malicious when it calls a vulnerable API method or when it exhibits potentially malicious or unknown behaviors. The defense is validated against the *reverse mimicry* attack [82] with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (0, 0, 0, 0, 0 | \mathcal{M}, \text{BE}, \text{MI}, \mathcal{Z}_{\mathcal{M}})$. The defense has input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{RF}, S, F_\theta, \text{FQ} | 1, 1, 0, 0)$ and achieves RR, CR, and DR.

3.2.6 Defenses Using IT. Wang et al. [127] propose the *random feature nullification* to enhance DNN-based malware detectors against the attack of **Fast Gradient Sign Method (FGSM)** [51] by nullifying (or dropping) features randomly in both training and testing phases. This offers a probabilistic assurance in preventing a white-box attacker from deriving adversarial files by using gradients of the loss function with respect to the input. The defense enhances Windows malware detectors against the FGSM attack with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (0, 1, 1, 1, 1 | \mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_{\mathbf{M}})$. The defense works under IID assumption with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{IT}, S, F_\theta, \text{FQ} | 0, 1, 0, 0)$ and achieves CR.

DroidEye [28] defends Android malware detectors against evasion attacks by quantizing binary representations, namely transforming binary representations into real values and then using compression to reduce the effect of adversarial manipulations. The defense enhances linear malware detectors against a "feature selection"-based attack with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_{\mathbf{M}})$ [29] and the FGSM attack with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (0, 1, 1, 1, \ 1 | \mathbf{M}, \text{OE2}, \text{GO}, \mathcal{X}_{\mathbf{M}})$ [51]. The defense works under IID assumption with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{IT}, S, F_\theta, \text{FQ} | 0, 1, 0, 0)$ and achieves CR.

3.2.7 Defenses Using CD. Khasawneh et al. [65] propose randomizing classifiers (i.e., using one randomly chosen from a pool of classifiers that use heterogeneous features) to defend against transfer attacks. The defense is validated against an attack which perturbs important features with input $(a_1, \ldots, a_5 | a_6, \ldots, a_9) = (0, 0, *, 0, 1 | \mathcal{M}, \text{BE}, \text{SF}, \mathcal{Z}_{\mathcal{M}})$. The defense works under the IID assumption with input $(a_1, \ldots, a_5 | a_6, \ldots, a_9) = (D_{train}, \text{CD}, S, F_\theta, \text{FQ} | 0, 1, 1, 0)$ and achieves the CR property.

3.2.8 Defenses Using SE. Smutz and Stavrou [110] propose an ensemble classifier to defend against gray-box evasion attacks by returning classification results as benign, uncertain and malicious according to the voting result (e.g., [0%, 25%] classifiers saying malicious can be treated as benign, [25%, 75%] saying malicious can be treated as uncertain, and [75%, 100%] saying malicious can be treated as malicious). The defense enhances a PDF malware detector against three types of evasion attacks: gradient-based attack [114] with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 0, *, *, 0 | \mathcal{M}, \text{OE2}, \text{TR}, \mathcal{Z}_{\mathbf{M}})$, mimicry attack with input $(1, 0, *, *, 0 | \mathcal{M}, \text{BE}, \text{TR}, \mathcal{Z}_{\mathbf{M}})$, and reverse mimicry attack with input $(0, 0, 0, 0, 0 | \mathcal{M}, \text{BE}, \text{MI}, \mathcal{Z}_{\mathbf{M}})$ [82]. The defense works under the IID assumption with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{SE}, S, F_\theta, \text{FQ} | 0, 1, 0, 0)$ and achieves DR.

Dang et al. [36] propose enhancing PDF malware detectors by lowering the classification threshold τ and restricting the maximum query times, rendering genetic algorithm-based evasion attacks harder to succeed. This defense works under the IID assumption with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{SE}, S, F_\theta, \text{LQ} | 0, 1, 0, 0)$ and achieves DR.

Chen et al. [30] investigate defending Android malware detectors against poisoning attacks with input $(a_1, \ldots, a_5 | A_6, \ldots, A_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \text{BP}, \text{SF}, \mathcal{X}_{\mathbf{M}})$. The idea is to filter adversarial files that are distant from non-adversarial ones, where distance is measured by the Jaccard index, Jaccard-weight similarity, and cosine similarity. The defense works under the Measurability assumption with input $(A_1, \ldots, A_5 | a_6, \ldots, a_9) = (D_{train}, \text{SE}, S, F_\theta, \text{FQ} | 0, 1, 0, 0)$ and achieves TR.

3.2.9 Drawing Observations and Insights. Summarizing the preceding discussions, we draw the following observations. (i) Most studies focus on black-box defenses (i.e., the defender knows little about the attacker), which is against the principle of "knowing yourself and knowing your enemy". (ii) Most studies focus on defenses against evasion attacks rather than poisoning attacks. (iii) There is no silver bullet defense against evasion attacks or poisoning attacks, at least for now. (iv) Sanitizing adversarial files as outliers is effective against black-box and gray-box attacks, but not white-box attacks. (v) The security properties achieved by defenses have been evaluated empirically rather than rigorously proven (despite that provable security is emerging on the small degree of perturbations; see for example [32, 48]). (vi) There is no theoretical evidence to support that the effectiveness of defense tactics on the training set (e.g., AT and verifiable learning) can generalize to other adversarial examples. In addition, we draw the following insights:

Insight 2. *(i) Effective defenses often require the defender to know the attacker's manipulation set. In the real world, it is hard to achieve this, explaining from one perspective why it is hard to design effective defenses. (ii) The effectiveness of adversarial training depends on the defender's capability in identifying the most powerful attack.*

3.3 Systematizing AMD Arms Race

Figure 5 displays AMD attack-defense arms race surrounding three malware detectors: PDFrate, Drebin, and DNN-based detector. For a better visual effect, we group papers that proposed defense methods in terms of a common input $(a_6, \ldots, a_9)$. For example, we group [136],[53], and [29] together because the defenders in both papers have input $(a_6, \ldots, a_9) = (1, 1, 1, 0)$, while noting that their input on $(A_1, \ldots, A_5)$ may or may not be different. We also simplify attack and defense inputs while preserving the *critical information* when an attack (defense) works for multiple inputs. For example, $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 1, 0, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$ is the *critical information* for attack input $(a_1, \ldots, a_5|A_6, \ldots, A_9) = (0, 0, 1, 0, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, 0, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 1|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$ because it is the weakest attack input in the partial order formulated by these $(a_1, \ldots, a_5)$'s. This suggests us to focus on attack input $(0, 0, 1, 0, 0|\mathbf{M}, \text{OE2}, \text{SF}, \mathcal{X}_\mathbf{M})$ because it is already able to break some defense and automatically implies that a stronger input can achieve the same (while noting some special cases, see discussion in Section 3.2.1). Multiple defense inputs are simplified in the same manner.

Arms race in PDF malware detection: We summarize two sequences of escalations caused by PDFrate [109]. In one sequence, PDFrate is defeated by transfer attacks, which are realized by gradient-based and mimicry methods against surrogate models [114]. These attacks trigger the defense escalation to an ensemble detector built on top of some diversified classifiers [110]. This defense [110] triggers attack escalation to *reverse mimicry attacks* [82], which trigger the defense escalation of using robust hand-crafted features [62]. This defense represents the state-of-the-art PDF malware detector, but still incurs a high false-positive rate. In the other sequence of arms race, PDFrate is defeated by genetic algorithm-based attacks [134]. These attacks trigger the defense escalation to [36] and [122]. The former defense [36] restricts the responses to attacker queries, but can be defeated by the escalated attack that leverages the hill-climbing algorithm (also shown in [36]). The latter defense [122] uses invariant features to thwart the attacks and represents another state-of-the-art PDF malware detectors.

Arms race in android malware detection: Drebin is defeated by the attack that modifies a limited number of important features [39], which also proposes the new defense to defeat the escalated attack. This defense [39] triggers the attack escalation to, and is defeated by, the genetic algorithm-based attack [136] and the mimicry-alike attack [97]. The former attack [136] triggers the escalated defense (also presented in [136]) that leverages attack mutations to detect adversarial examples

[136]. The latter attack [97] injects objects in APKs and (in principle) can be defeated by the monotonic classifier [61, 97]. These escalated defenses represent the state-of-the-art Android malware detectors, but still incur a high false-positive rate.

Arms race in DNN-based malware detection: The DNN-based detector [53] triggers four gradient-based evasion attacks presented in [2], which also hardens the DNN malware detector by using an *minmax adversarial training* instantiation to incorporates the ℓ_∞ normalized gradient-based attack. This escalated defense [2] triggers the mixture of attacks presented in [76]. The defense of *minmax adversarial training* incorporating a mixture of attacks can defeat a broad range of attacks, but still suffers from the mimicry attack and other mixtures of attacks [76]. As a consequence, there are no effective defenses can thwart all kinds of attacks.

Independent arms race: There are studies that have yet to trigger cascading arms races, including: (i) Studies [29, 30, 118, 127] propose independent attacks and then show how to defeat these attacks. (ii) Studies [31, 58, 59, 68, 70, 87, 100, 117] propose attacks to defeat naive malware detectors. (iii) Studies propose defenses to counter some attacks [15, 27, 28, 32, 77].

4 FUTURE RESEARCH DIRECTIONS (FDRS)

FRD 1: *Pinning down the root cause(s) of adversarial malware examples.* Speculations on root cause(s) include: (i) invalidity of the IID assumption because of distribution drifting, namely that testing files and training files are drawn from different distributions [16, 19, 52]); (ii) incompetent feature extraction [39, 134]; (iii) high dimensionality of malware representations [48]; (iv) insufficient scale of training data [104]; (v) low-probability "pockets" in data manifolds [120]; (vi) linearity of DNNs [51]; and (vii) large curvature of decision boundaries [46, 85].Although these speculations may be true, more studies are needed in order to (in)validate them.

FRD 2: *Characterizing the relationship between transferability and vulnerability*. In the AMD context, an attacker may use a surrogate model to generate adversarial examples and a defender may use a surrogate model for AT. Transferability is related to the extent at which knowledge gained by a surrogate model may be the same as, or similar to, what is accommodated by a target model. The wide use of surrogate models in the AMD context suggests that there may be a fundamental connection between knowledge transferability and model vulnerability.

FRD 3: *Investigating adversarial malware examples in the wild.* In the AMD context, it is challenging to generate practical adversarial malware examples to correspond to perturbations conducted in the feature space, owing to realistic constraints. On the other hand, an attacker can directly search for manipulations in the problem space. This may cause large perturbations, putting the value of studies on small perturbations in question. This represents a fundamental open problem that distinguishes the field of AMD from its counterparts in other application settings. This issue is largely unaddressed by assuming that there is an oracle for telling whether manipulated or perturbed features indeed correspond to a malware sample or not.

FRD 4: *Quantifying the robustness and resilience of malware detectors*. Robustness and resilience of malware detectors against adversarial examples need to be quantified, ideally with a provable guarantee. For this purpose, one may adapt the reduction-based paradigm underlying the provable security of cryptographic primitives and protocols.

FRD 5: *Designing malware detectors with provable robustness and resilience guarantees.* Having understood the root cause(s) of adversarial examples, characterized the effect of transferability, investigated the effectiveness of practical attacks, and designed metrics for quantifying the robustness and resilience of malware detectors, it is imperative to investigate robust malware detectors with provable robustness, ideally as rigorous as what has been achieved in the field of cryptography. In this regard, RF extraction, AL, and VL are promising candidates for making breakthroughs.

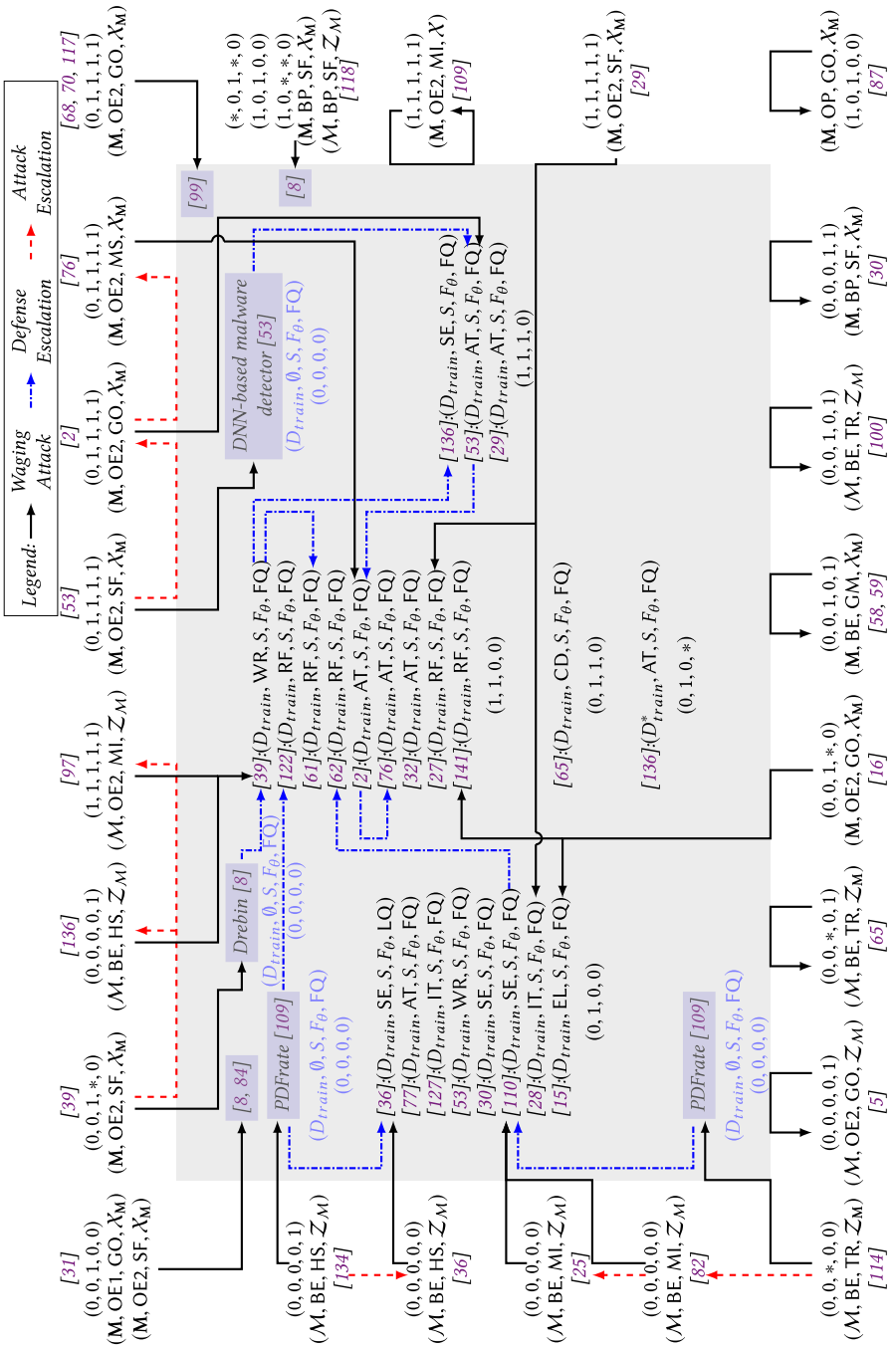

Fig. 5. Arms race in AMD attack and defense escalations.

FRD 6: *Forecasting the arms race in malware detection.* Arms race is a fundamental phenomenon inherent to the cybersecurity domain. In order to effectively defend against adversarial malware, one approach is to deploy proactive defense, which requires the capability to forecast the arms race between malware writers and defenders. For instance, it is important to predict how

attacks will evolve and what kinds of information would be necessary in order to defeat such attacks.

5 CONCLUSION

We have presented a framework for systematizing the field of AMD through the lens of assumptions, attacks, defenses, and security properties. This paves the way for precisely relating attacks and defenses. We have also shown how to apply the framework to systematize the AMD literature, including the arms race between AMD attacks and defenses. We have reported a number of insights.

The study leads to a set of future research directions. In addition to the ones described in Section 4, we mention the following two, which are discussed here, because there are rarely studies on these aspects. (i) To what extent explainability (or interpretability) of ML models can be leveraged to cope with adversarial malware examples? It is intuitive that explainability could be leveraged to recognize adversarial examples because they may not be explainable [37]. (ii) To what extent uncertainty quantification can be leveraged to cope with adversarial malware examples? If the uncertainty associated with detectors' predictions on adversarial malware examples are inherently and substantially higher than the uncertainty associated with non-adversarial malware examples, this fact can be leveraged to recognize adversarial malware examples. Finally, we reiterate that the research community should seek to establish a solid foundation for AMD. Although this foundation can leverage ideas and techniques from AML, the unique characteristics of AMD warrant the need of a unique foundation.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments that guided us in improving the article.

REFERENCES

[1] Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. 2004. N-gram-based detection of new malicious code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference*. Vol. 2. IEEE, 41–42.

[2] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O.Reilly. 2018. Adversarial deep learning for robust detection of binary encoded malware. In *Proceedings of the 2018 IEEE Security and Privacy Workshops*. IEEE, 76–82.

[3] Victor M. Alvarez. 2019. Yara. (May 2019). Retrieved May 2, 2019 from http://virustotal.github.io/yara/.

[4] Blake Anderson, Curtis Storlie, and Terran Lane. 2012. Improving malware classification: Bridging the static/dynamic gap. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. ACM, 3–14.

[5] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to evade static PE machine learning malware models via reinforcement learning. arXiv:1801.08917. Retrieved from https://arxiv.org/abs/1801.08917.

[6] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. 2017. Evading machine learning malware detection. *Presented at Black Hat USA*. Informa PLC, 1–6.

[7] Axelle Aprville and Ange Albertini. 2014. Hide android applications in images. (October 2014). Retrieved October, 2014 from https://www.blackhat.com/.

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. DREBIN: Effective and explainable detection of android malware in your pocket. In *Proceedings of the Network and Distributed System Security Symposium*. Vol. 14. The Internet Society, 23–26.

[9] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. Doug Tygar. 2010. The security of machine learning. *Machine Learning* 81, 2 (2010), 121–148.

[10] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. Doug Tygar. 2006. Can machine learning be secure?. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*. ACM, 16–25.

[11] Karel Bartos, Michal Sofka, and Vojtech Franc. 2016. Optimized invariant representation of network traffic for detecting unseen malware variants. In *Proceedings of the 25th {USENIX} Security Symposium ({USENIX} Security 16)*. USENIX Association, 807–822.

[12] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, 2613–2621.

[13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (2013), 1798–1828.

[14] Benjamin Bichsel, Veselin Raychev, Petar Tsankov, and Martin Vechev. 2016. Statistical deobfuscation of android applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 343–355.

[15] Battista Biggio, Igino Corona, Zhi-Min He, Patrick PK Chan, Giorgio Giacinto, Daniel S. Yeung, and Fabio Roli. 2015. One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time. In *Proceedings of the International Workshop on Multiple Classifier Systems*. Springer, 168–180.

[16] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim .rndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference*. Springer, 387–402.

[17] Battista Biggio, Igino Corona, Blaine Nelson, Benjamin IP Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. 2014. Security evaluation of support vector machines in adversarial environments. In *Proceedings of the Support Vector Machines Applications*. Springer, 105–153.

[18] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2010. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics* 1, 1-4 (2010), 27–41.

[19] B. Biggio, G. Fumera, and F. Roli. 2014. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering* 26, 4 (Apr. 2014), 984–996. DOI: https://doi.org/10.1109/TKDE.2013.57

[20] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning*. Omnipress, 105–153.

[21] Leyla Bilge and Tudor Dumitras. 2012. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. ACM, 833–844.

[22] Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (2001), 5–32.

[23] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 3–14.

[24] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*. IEEE, 39–57.

[25] Curtis Carmony, Xunchao Hu, Heng Yin, Abhishek Vasisht Bhaskar, and Mu Zhang. 2016. Extract me if you can: Abusing PDF parsers in malware detectors. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*. The Internet Society.

[26] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2021. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology* 6, 1 (2021), 25–45.

[27] Lingwei Chen, Shifu Hou, and Yanfang Ye. 2017. Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 362–372.

[28] Lingwei Chen, Shifu Hou, Yanfang Ye, and Shouhuai Xu. 2018. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks. In *Proceedings of the IEEE/ACM 2018 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 782–789.

[29] Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. 2017. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *Proceedings of the European Intelligence and Security Informatics Conference*. IEEE, 99–106.

[30] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Computers & Security* 73 (2018), 326–344.

[31] X. Chen, C. Li, D. Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. 2019. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security* 15 (2019), 987–1001.

[32] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. 2020. On training robust PDF malware classifiers. In *Proceedings of the 29th USENIX Security Symposium*. USENIX Association, 2343–2360.

[33] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. ACL, 1724–1734.

[34] CISCO. 2018. CISIO reporter. (December 2018). Retrieved December 2, 2018 from https://www.cisco.com.

[35] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12, Aug (2011), 2493–2537.
[36] Hung Dang, Yue Huang, and Ee-Chien Chang. 2017. Evading classifiers by morphing in the dark. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 119–133.
[37] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2019. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *Proceedings of the 3rd Italian Conference on Cyber Security*. Vol. 2315. CEUR-WS.org.
[38] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection. *ACM Transactions on Privacy and Security* 24, 4 (2021), 1–31.
[39] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino 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* 16, 4 (2017), 711–724.
[40] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, 321–338.
[41] Dexguard. 2018. Dexguard @ONLINE. (December 2018). Retrieved December, 2018 from https://www.guardsquare.com/en/products/dexguard.
[42] Hermann Dornhackl, Konstantin Kadletz, Robert Luh, and Paul Tavolato. 2014. Malicious behavior patterns. In *Proceedings of the 2014 IEEE 8th International Symposium on Service Oriented System Engineering*. IEEE, 384–389.
[43] Pang Du, Zheyuan Sun, Huashan Chen, Jin-Hee Cho, and Shouhuai Xu. 2018. Statistical estimation of malware detection metrics in the absence of ground truth. *IEEE Transactions on Information Forensics and Security* 13, 12 (2018), 2965–2980.
[44] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys* 44, 2 (2012), 6.
[45] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. 2018. Gotcha—sly malware!: Scorpion a Metagraph2vec based malware detection system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 253–262.
[46] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. 2016. Robustness of classifiers: From adversarial to random noise. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, 1632–1640.
[47] M. Garnaeva, F. Sinitsyn, Y. Namestnikov, Denis Makrushin, and Alexander Liskin. 2016. Kaspersky Security Bulletin Overall Statistics for 2016. (December 2006). Retrieved December 2006 from https://media.kasperskycontenthub.com.
[48] Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian Goodfellow. 2018. Adversarial spheres. In *Proceedings of the 6th International Conference on Learning Representations*. OpenReview.net.
[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. Vol. 1. MIT press, Cambridge, MA.
[50] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, 2672–2680.
[51] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations*. OpenReview.net.
[52] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (Statistical) Detection of Adversarial Examples. arXiv:1702.06280. Retrieved from https://arxiv.org/abs/1702.06280.
[53] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *Proceedings of the European Symposium on Research in Computer Security*. Springer, 62–79.
[54] Maya Gupta, Andrew Cotter, Jan Pfeifer, Konstantin Voevodski, Kevin Canini, Alexander Mangylov, Wojciech Moczydlowski, and Alexander Van Esbroeck. 2016. Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research* 17, 1 (2016), 3790–3836.
[55] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. 2016. DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Mining*. 61.
[56] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD*. ACM, 1507–1515.
[57] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2018. Make evasion harder: An intelligent android malware detection system. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. ijcai.org, 5279–5283.

[58] Weiwei Hu and Ying Tan. 2017. Generating adversarial malware examples for black-box attacks based on GAN. arXiv:1702.05983. Retrieved from https://arxiv.org/abs/1702.05983.
[59] Weiwei Hu and Ying Tan. 2018. Black-box attacks against RNN based malware detection algorithms. In *Proceedings of the Workshops of the 32nd AAAI Conference on Artificial Intelligence*. AAAI Press, 245–251.
[60] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. ACM, 43–58.
[61] Inigo Incer, Michael Theodorides, Sadia Afroz, and David A. Wagner. 2018. Adversarially robust malware detection using monotonic classification. In *Proceedings of the 4th ACM International Workshop on Security and Privacy Analytics*. ACM, 54–63.
[62] Alexander Jordan, François Gauthier, Behnaz Hassanshahi, and David Zhao. 2019. Unacceptable behavior: Robust pdf malware detection using abstract interpretation. In *Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security*. ACM, 19–30.
[63] Jinho Jung, Chanil Jeon, Max Wolotsky, Insu Yun, and Taesoo Kim. 2017. AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically. (July 2017). Retrieved July 2017 from https://www.blackhat.com/.
[64] Kris Kendall and Chad McMillan. 2007. Practical malware analysis. In *Proceedings of the Black Hat Conference*. (2007). Online; access at December 2019.
[65] Khaled N. Khasawneh, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Lei Yu. 2017. RHMD: Evasion-resilient hardware malware detectors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 315–327.
[66] Jin-Young Kim, Seok-Jun Bu, and Sung-Bae Cho. 2018. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. *Information Sciences* 460-461 (2018), 83–102.
[67] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.), ACL, 1746–1751.
[68] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *Proceedings of the 26th European Signal Processing Conference*. IEEE, 533–537.
[69] Bojan Kolosnjaji, Ghadir Eraisha, George Webster, Apostolis Zarras, and Claudia Eckert. 2017. Empowering convolutional networks for malware classification and analysis. In *Proceedings of the 2017 International Joint Conference on Neural Networks*. IEEE, 3838–3845.
[70] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. 2018. Deceiving end-to-end deep learning malware detectors using adversarial examples. In *Proceedings of NeurIPS 2018 Workshop on Security in Machine Learning*. Retrieved from https://secml2018.github.io/malware.pdf.
[71] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net.
[72] Kaspersky Lab. 2018. Kaspersky . (May 2018). Retrieved May, 2018 from https://www.kaspersky.com/.
[73] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.
[74] Qi Lei, Lingfei Wu, Pin-Yu Chen, Alexandros Dimakis, Inderjit Dhillon, and Michael Witbrock. 2019. Discrete adversarial attacks and submodular optimization with applications to text classification. In *Proceedings of Machine Learning and Systems*. mlsys.org.
[75] Bo Li, Yevgeniy Vorobeychik, and Xinyun Chen. 2016. A General Retraining Framework for Scalable Adversarial Classification. In *Proceedings of NeurIPS 2016 Workshop on Adversarial Training*. Retrieved from https://sites.google.com/site/nips2016adversarial/WAT16_paper_2.pdf?attredirects=0.
[76] Deqiang Li and Qianmu Li. 2020. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3886–3900.
[77] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2019. Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and application to AICS'2019 challenge. In *Proceedings of the AAAI-19 Workshop on Artificial Intelligence for Cyber Security*. Retrieved from http://arxiv.org/abs/1812.08108.
[78] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2021. A framework for enhancing deep neural networks against adversarial malware. *IEEE Transactions on Network Science and Engineering* 8, 1 (2021), 736–750.
[79] Qiang Liu, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor CM Leung. 2018. A survey on security threats and defensive techniques of machine learning: A data driven view. *IEEE Access* 6 (2018), 12103–12117.
[80] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the 6th International Conference on Learning Representations*. OpenReview.net.
[81] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. 2019. Towards adversarial malware detection: Lessons learned from PDF-based attacks. *ACM Computing Surveys* 52, 4 (Aug. 2019), 1–36. DOI: https://doi.org/10.1145/3332184

[82] Davide Maiorca, Igino Corona, and Giorgio Giacinto. 2013. Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious PDF files detection. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. ACM, 119–130.

[83] Davide Maiorca, Ambra Demontis, Battista Biggio, Fabio Roli, and Giorgio Giacinto. 2020. Adversarial detection of flash malware: Limitations and open issues. *Computers & Security* 96 (2020), 101901.

[84] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium*. The Internet Society.

[85] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, Pascal Frossard, and Stefano Soatto. 2018. Robustness of classifiers to universal perturbations: a geometric perspective. In *Proceedings of the 6th International Conference on Learning Representations*. OpenReview.net.

[86] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference*. IEEE, 421–430.

[87] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 27–38.

[88] Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. 2016. Adaptive and scalable android malware detection through online learning. In *Proceedings of the 2016 International Joint Conference on Neural Networks* . IEEE, 2484–2491.

[89] Andrew Y. Ng. 2004. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the 21st International Conference on Machine Learning*. Association for Computing Machinery, 78.

[90] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*. Vol. 119. ACM, 625–632.

[91] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. Association for Computing Machinery, 506–519.

[92] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy*. IEEE, 372–387.

[93] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy*. IEEE, 582–597.

[94] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. 2018. Sok: Security and privacy in machine learning. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy*. IEEE, 399–414.

[95] Andrea Paudice, Luis Muñoz-González, and Emil C. Lupu. 2018. Label sanitization against label flipping poisoning attacks. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Vol. 11329. Springer, 5–15.

[96] Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 8 (2005), 1226–1238.

[97] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. 2020. Intriguing properties of adversarial ML attacks in the problem space. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 1308–1325.

[98] Niels Provos and Thorsten Holz. 2007. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Pearson Education, London.

[99] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. 2018. Malware detection by eating a whole exe. In *Proceedings of the Workshops at the 32nd AAAI Conference on Artificial Intelligence*. AAAI Press.

[100] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2018. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In *Research in Attacks, Intrusions, and Defenses*, Michael Bailey. Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis (Eds.), Springer, Cham, 490–510.

[101] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. 2012. Prudent practices for designing malware experiments: Status quo and outlook. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 65–79.

[102] Paolo Russu, Ambra Demontis, Battista Biggio, Giorgio Fumera, and Fabio Roli. 2016. Secure kernel machines against evasion attacks. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM, 59–69.

[103] Joshua Saxe and Konstantin Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *Proceedings of the 10th International Conference on Malicious and Unwanted Software*. IEEE, 11–20.

[104] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. 2018. Adversarially robust generalization requires more data. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, 5019–5031.

[105] Alexandru Constantin Serban and Erik Poll. 2018. Adversarial examples - a complete characterisation of the phenomenon. arXiv:1810.01185. Retrieved from https://arxiv.org/abs/1810.01185.

[106] Syed Zainudeen Mohd Shaid and Mohd Aizaini Maarof. 2015. In memory detection of windows API call hooking technique. In *Proceedings of the 2015 International Conference on Computer, Communications, and Control Technology*. IEEE, 294–298.

[107] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York.

[108] Hispasec Sistemas. 2019. VirusTotal. (May 2019). Retrieved May 2, 2019 from https://www.virustotal.com.

[109] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF detection using metadata and structural features. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 239–248.

[110] Charles Smutz and Angelos Stavrou. 2016. When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors. In *Proceedings of the 23rd Network and Distributed System Security Symposium*. The Internet Society.

[111] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. 2008. Bitblaze: A new approach to computer security via binary analysis. In *Proceedings of the International Conference on Information Systems Security*. Springer, 1–25.

[112] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[113] Nedim Šrndić and Pavel Laskov. 2013. Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*. The Internet Society, 1–16.

[114] N. Šrndić and P. Laskov. 2014. Practical evasion of a learning-based classifier: A case study. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. IEEE, 197–211.

[115] Nedim Šrndić and Pavel Laskov. 2016. Hidost: A static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security* 2016, 1 (2016), 22.

[116] Jacob Steinhardt, Pang Wei W. Koh, and Percy S Liang. 2017. Certified defenses for data poisoning attacks. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc., 3517–3529.

[117] O. Suciu, S. E. Coull, and J. Johns. 2019. Exploring adversarial examples in malware detection. In *Proceedings of the 2019 IEEE Security and Privacy Workshops*. IEEE, 8–14.

[118] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. 2018. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18)*. USENIX Association, 1299–1316.

[119] Johan A. K. Suykens and Joos Vandewalle. 1999. Least squares support vector machine classifiers. *Neural Processing Letters* 9, 3 (1999), 293–300.

[120] Christian Szegedy, Wojciech Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*. OpenReview.net.

[121] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: Large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD*. ACM, 1524–1533.

[122] Liang Tong, Bo Li, Chen Hajaj, Chaowei Xiao, Ning Zhang, and Yevgeniy Vorobeychik. 2019. Improving robustness of ML classifiers against realizable evasion attacks using conserved features. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, 285–302.

[123] Florian Tramèr and Dan Boneh. 2019. Adversarial training and robustness for multiple perturbations. In *Proceedings of the Advances in Neural Information Processing Systems 32: NeurIPS 2019*. Curran Associates, Inc., 5858–5868.

[124] Vladimir Vapnik. 1991. Principles of risk minimization for learning theory. In *Proceedings of the Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 831–838.

[125] R. Vinayakumar and K. P. Soman. 2018. Deepmalnet: Evaluating shallow and deep networks for static PE malware detection. *ICT Express* 4, 4 (2018), 255–258.

[126] Beilun Wang, Ji Gao, and Yanjun Qi. 2017. A Theoretical Framework for Robustness of (Deep) Classifiers against Adversarial Samples. In *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net.

[127] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia, Xinyu Xing, Xue Liu, and C. Lee Giles. 2017. Adversary resistant deep neural networks with an application to malware detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1145–1153.

[128] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX Security*. William Enck and Adrienne Porter Felt (Eds.), USENIX Association, 1599–1614.

[129] W. Wang, X. Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. 2014. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security* 9, 11 (Nov 2014), 1869–1882.

[130] Eric Wong and J. Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. PMLR, 5283–5292.

[131] AICS workshop. 2018. AICS 2019 Workshop Challenge Problem. (December 2018). Retrieved December, 2018 from http://www-personal.umich.edu/~arunesh/AICS2019/challenge.html.

[132] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is feature selection secure against training data poisoning?. In *Proceedings of the International Conference on Machine Learning*. JMLR.org, 1689–1698.

[133] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. 2014. An evasion and counter-evasion study in malicious websites detection. In *Proceedings of the 2014 IEEE Conference on Communications and Network Security*. IEEE, 265–273.

[134] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically evading classifiers: A case study on PDF malware classifiers. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*. The Internet Society.

[135] Peng Yang and Peilin Zhao. 2015. A min-max optimization framework for online graph classification. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 643–652.

[136] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. 2017. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 288–302.

[137] Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. 2009. SBMDS: An interpretable string based malware detection system using SVM ensemble with bagging. *Journal in Computer Virology* 5, 4 (2009), 283.

[138] Yanfang Ye, Tao Li, Donald A. Adjeroh, and S. Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Computing Surveys* 50, 3 (2017), 1–40.

[139] I. You and K. Yim. 2010. Malware obfuscation techniques: A brief survey. In *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. IEEE, 297–300.

[140] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems* 30, 9 (2019), 2805–2824.

[141] Fei Zhang, Patrick P. K. Chan, Battista Biggio, Daniel S. Yeung, and Fabio Roli. 2016. Adversarial feature selection against evasion attacks. *IEEE Transactions on Cybernetics* 46, 3 (2016), 766–777.

[142] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, Boca Raton, FL.

[143] Ziyun Zhu and Tudor Dumitras. 2016. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 767–778.

Received June 2020; revised July 2021; accepted August 2021