SoK: Arms Race in Adversarial Malware Detection

Deqiang Li*, Qianmu Li*, Yanfang (Fanny) Ye †, Shouhuai Xu‡
* Nanjing University of Science and Technology, † Case Western Reserve University, ‡ University of Texas at San Antonio

***Abstract*—Malicious software (malware) is a major cyber threat that shall be tackled with Machine Learning (ML) techniques because millions of new malware examples are injected into cyberspace on a daily basis. However, ML is known to be vulnerable to attacks known as adversarial examples. In this SoK paper, we systematize the field of Adversarial Malware Detection (AMD) through the lens of a unified framework of assumptions, attacks, defenses and security properties. This not only guides us to map attacks and defenses into some partial order structures, but also allows us to clearly describe the attack-defense arms race in the AMD context. In addition to manually drawing insights, we also propose using ML to draw insights from the systematized representation of the literature. Examples of the insights are: knowing the defender's feature set is critical to the attacker's success; attack tactic (as a core part of the threat model) largely determines what security property of a malware detector can be broke; there is currently no silver bullet defense against evasion attacks or poisoning attacks; defense tactic largely determines what security properties can be achieved by a malware detector; knowing attacker's manipulation set is critical to defender's success; ML is an effective method for insights learning in SoK studies. These insights shed light on future research directions.**

I. Introduction

Malware (malicious software) is a big cyber threat and has received a due amount of attention. For example, Kaspersky reports that malware attacked 2,871,965 devices in year 2016, 1,126,701 in 2017, and 830,135 in 2018 [1], [2]. A popular defense against malware is to use *signature-based* detectors [3], 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 [4] by a range of techniques (e.g., encryption, repacking, polymorphism [5], [6], [7], [8], [9], [10], [11]). The incompetence of this approach has motivated the use of Machine Learning (ML) based malware detectors, which can be automated at some extent and can possibly detect *new* malware examples (via model generalization or knowledge adaptation [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]). More recently, Deep Learning (DL) has been used for malware detection (see, e.g., [25], [26], [27]).

While promising, ML-based malware detectors are vulnerable to attacks known as *adversarial examples* [28], [29], [30]. There are two kinds of attacks. One is *evasion attack*, where the attacker perturbs test examples to *adversarial examples* to evade malware detectors [31], [32], [28], [33], [17], [34], [35], [36], [37]. The other is *poisoning attack*, where the attacker manipulates the training data for learning malware detectors [38], [39], [40]. These attacks usher in the new field of Adversarial Malware Detection (AMD) [41], [42], [36], [28], [29], [43], [44], [37], [32], [39], [40].

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) [45], [30], [46], [47], which however cannot be automatically translated to AMD because malware detection has three *unique* characteristics that are not possessed by other domains, such as image or audio processing. (i) There is no common, standard feature definitions because both attacker and defender can define their own features to represent computer files. This "freedom" in feature definition can be leveraged by the attacker to craft adversarial examples, which can be hard to detect. (ii) Malware features are often discrete rather than continuous and program files are often highly structured (e.g. Windows PE [48]). This means that arbitrarily perturbing malware files or their feature representations might make the perturbed files no more executable or malicious. This also means that the discrete domain makes perturbations a *non-differentiable* and *non-convex* task. For example, the Android Package file requires that the used permissions be publicized in the *AndroidManifest.xml*, meaning that removing a permission in the manifest file would incur a runtime error. (iii) Any meaningful perturbation to a malware example or its feature representation must preserve its malicious functionality. The preceding (ii) and (iii) make both the attacker's and defender's tasks more challenging than their image and video counterparts for which small perturbations would not be noticeable.

Our Contributions. We propose a framework for systematizing the AMD field through the lens of assumptions, attacks, defenses and security properties. The framework allows us to map known attacks and defenses into some partial order structures and systematize the AMD attack-defense arms race. We *manually* draw a number of insights, including: (i) untargeted attack is much more extensively investigated than targeted and frustration attacks; (ii) evasion attack has been much more extensively studied than poisoning attack; (iii) a certain assumption, which will be defined later, is widely made; (iv) knowing defender's feature set is critical to attacker's success, highlighting the importance of (e.g.) *randomizing defender's features*; (v) defensive studies have mainly focused on black-box defenses and defending against evasion attacks; (vi) there is currently no silver bullet defense against evasion attacks or poisoning attacks; (vii) AMD security properties have been evaluated empirically rather than rigorously; (viii) sanitizing adversarial examples is effective against black-box and grey-box attacks, but not white-box attacks. In addition, we propose *using ML to draw insights* from the systematized representation of AMD attacks and defenses. This leads to new insights that we could not draw manually, such as: (ix) attack tactic (as a core part of the threat model) largely determines what security property of a malware detector can be broke; (x) defense tactic largely determines what security properties can be achieved by a malware detector; (xi) knowing attacker's manipulation set is critical to defender's success, highlighting importance of *knowing your enemy*; (xii) ML is an effective method for insights learning in SoK studies. We also outline a number of future research directions.

Related Work. The closely related prior study is Maiorca et al. [49], which surveys previous studies in adversarial malicious PDF documents detection by inheriting an earlier attack framework introduced in AML context [45], [50], [51]. 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 systematize or survey prior AML studies (but not focusing on AMD), including [47], [45], [50], [52], [53], [54], [46], [55]. For example, Yuan et al. [47] survey attack methods for generating adversarial examples, while briefly discussing evasion attacks in AMD context; Barreno et al. [45], [50] propose a taxonomy of AML attacks (causative vs. exploratory attacks, integrity vs. availability attacks, and targeted vs. indiscriminate attacks); Biggio et al. [53] propose a defense framework for protecting Support Vector Machines (SVMs) from evasion attacks, poisoning attacks and privacy violations; Papernot et al. [54] systematize AML security and privacy with emphasis on demonstrating the trade-off between detection accuracy and robustness.

Paper Outline. Section II describes our framework. Section III uses our framework to systematizing literature AMD studies. Section IV discusses future research directions. Section V concludes the paper.

II. Systematization Framework

Terminology, Scope and Notations. In the context AMD, a defender $\mathcal{D}$ aims to use 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 $\mathcal{D}$'s detection by exploiting *adversarial files* (interchangeably, *adversarial examples*). A file, benign and malicious alike, is adversarial if it is intentionally crafted to (help malicious files) evade $\mathcal{D}$'s detection, and *non-adversarial* otherwise. We focus on $\mathcal{D}$ 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., [56]). Table I summaries the main notations used in the paper.

TABLE I: Main notations used in the paper

Notation	Meaning
$\mathbb{R}$ ($\mathbb{R}_+$)	the set of (positive) real numbers
$\mathcal{A}, \mathcal{D}$	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 $\mathcal{D}$'s feature set for representing files
$\mathbf{x}, \mathbf{x}' \in \mathcal{X}$	$\mathcal{X} = \mathbb{R}^d$ is 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; $y \in \mathcal{Y}$
$\mathcal{I} = \mathcal{Z} \times \mathcal{Y}$	the file-label (i.e., example-label) space
$I_{train} \subset \mathcal{I}, n$	the training set in file-label space; $n = \|I_{train}\|$
I_{test}	the test set in file-label space
I_{poison}, I'_{poison}	I'_{poison} is set of adversarial file-label pairs obtained by perturbing non-adversarial files in $I_{poison} \subset \mathcal{I}$
$\mathcal{O}(z, z')$	$\mathcal{O}(z, z') : \mathcal{Z} \times \mathcal{Z} \to \{\texttt{true}, \texttt{false}\}$ is an oracle telling if two files have the same functionality or not
$\delta \in \Delta$	manipulation vs. set of manipulations for perturbing files *with* or *without* preserving their functionalities
$\mathcal{M} \subseteq \Delta$, $\mathcal{Z}_{\mathcal{M}}$	$\mathcal{M}$ is file manipulation set; $\mathcal{Z}_{\mathcal{M}} \subset \mathcal{Z}$ 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 degree of manipulation for perturbing $z \in Z$ 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 $\mathbf{x}'$
$\delta_{\mathbf{x}} \in \mathbb{R}^d$	$\delta_{\mathbf{x}} = \mathbf{x}' - \mathbf{x}$ 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
$F_\theta : \mathcal{X} \to \mathbb{R}$	machine learning algorithm with parameters θ
$L : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$	loss function measuring prediction error of F_θ
BE, OE, BP, OP	attack tactics: basic and optimal evasion; basic and optimal poisoning
EL, WR, AT	defense tactics: ensemble learning, weight regularization, adversarial training
IT, CD, RF, SE	defense tactics: input transformation, classifier randomization, robust feature, sanitizing examples
$A_1, \ldots, A_5$	the 5 attributes under $\mathcal{D}$'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 $\mathcal{D}$ at respective degree $a_6, \ldots, a_9$
RR, CR, DR, TR	Representation Robustness, Classification Robustness, Detection Robustness, Training Robustness
$*$	the wildcard (taking any value in a domain)

A. Brief Review on ML-based Malware Detection

Let $\mathcal{Z}$ be the *example space* of benign/malicious adversarial/non-adversarial files. Let $\mathcal{Y} = \{+, -\}$ be the *label space* of binary classification, where $+$ ($-$) means a file is malicious (benign). Let $\mathcal{I} = \mathcal{Z} \times \mathcal{Y}$ be the file-label (example-label) space. For training and evaluating a classifier in the absence of adversarial files, $\mathcal{D}$ is given a set $I \subset \mathcal{I}$ of non-adversarial benign/malicious files as well as their *ground-truth* labels. $\mathcal{D}$ splits I into three disjoint sets: a training set $I_{train} = \{(z_i, y_i)\}_{i=1}^n$ where $(z_i, y_i) \in I$, 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 d features and represented by a d-dimension vector $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$ in *feature space* $\mathcal{X} = \mathbb{R}^d$, which accommodates both continuous and discrete feature representations [57], [58], [13], [59], [60], [14]. The process for obtaining feature representation $\mathbf{x}_i$ of $z_i \in \mathcal{Z}$ is called *feature extraction*, denoted by function $\phi : \mathcal{Z} \to \mathcal{X}$ with $\mathbf{x}_i \leftarrow \phi(z_i)$, where ϕ may be hand-crafted, automatically learned, or a hybrid of both [61], [62]. There are two kinds of features: *static features* are extracted via static analysis (e.g., strings, API calls [63], [64], [65]); *dynamic features* are extracted via dynamic analysis (e.g., instructions, registry activities [23], [66]).

As highlighted in Figure 1, $\mathcal{D}$ uses $\{(z_i, y_i)\}_{i=1}^n$ to learn a malware detector or classifier $f : \mathcal{Z} \to [0,1]$, where $f(\cdot) = \varphi(\phi(\cdot))$ 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 $\mathbf{x} \leftarrow \phi(z)$, can be interpreted as the probability that z is malicious. For a given threshold $\tau \in [0,1]$, we further say (*by abusing notation a little bit*) 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 some parameters (e.g., weights) that can be collectively denoted by θ, which is tuned to minimize an empirical risk [67]:

$$\min_{\theta} \ \mathcal{L}(\theta, I_{train}) = \min_{\theta} \ \frac{1}{n} \sum_{(z_i,y_i)\in I_{train}} \left[L(F_\theta(\phi(z_i)), y_i) \right], \tag{1}$$

where $L : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ is an appropriate loss function measuring prediction error of F_θ (e.g., cross-entropy [68]).

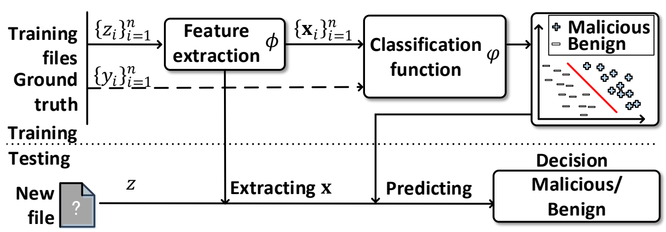

Fig. 1: Illustration of ML-based malware detector/classifier.

B. Systematization Methodology

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

C. Systematizing Assumptions

Four assumptions have been made in the AMD literature. Assumption 1 below is quite strong and researchers have started to weaken it [69], [70].

Assumption 1 (IID assumption; see, e.g., [71])**.** *Computer files in training data and testing data are independently drawn from the same distribution.*

Assumption 2 below is adapted from AML context, where human can serve as an oracle $\mathcal{O}$ for determining whether two images are the same [72]. In AMD context, $\mathcal{O}$ may be instantiated as (or approximated by) malware analysts [29], [44], [33], [73] or automated tools (e.g., Sandbox [42], [41]).

Assumption 2 (Oracle assumption; adapted from [72])**.** *There is an oracle $\mathcal{O} : \mathcal{Z} \times \mathcal{Z} \to \{\texttt{true}, \texttt{false}\}$ that tells if two files $z, z' \in \mathcal{Z}$ have the same functionality or not;* $\texttt{true} \leftarrow \mathcal{O}(z, z')$ *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 manipulation by which one file becomes another.

Assumption 3 (Measurability assumption [74], [75], [42])**.** *There is a function $\Gamma(z, z') : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}_+$ that measures the degree of manipulation according to which a file $z' \in \mathcal{Z}$ can be derived from 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 $\mathbf{x} = \phi(z)$ and $\mathbf{x}' = \phi(z')$. This leads to:

Assumption 4 (Smoothness assumption [61])**.** *There is a function $C(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ such that $C(\phi(z), \phi(z')) \approx 0$ when* $(\Gamma(z, z') \approx 0) \wedge (\texttt{true} \leftarrow \mathcal{O}(z, z'))$.

D. Systematizing Attacks

Attacker's Objective. There are three kinds of objectives: (i) Untargeted, meaning $\mathcal{A}$ attempts to cause as many false-negatives as possible [37], [76], [32], [77], [78], [79]; (ii) Targeted, meaning $\mathcal{A}$ attempts to cause specific false-negatives (i.e., making certain malicious files evade the detection [40], [39]); (iii) Frustration, meaning $\mathcal{A}$ attempts to frustrate $\mathcal{D}$ by rendering $\mathcal{A}$'s classifier f unusable (e.g., causing substantially high false-positives [38], [80], [40], [81], [82]).

Attacker's Input. Table II highlights the attributes we define to describe $\mathcal{A}$'s input, including: five attributes $A_1, \ldots, A_5$ that are under $\mathcal{D}$'s control 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.

TABLE II: Attributes for specifying $\mathcal{A}$'s and $\mathcal{D}$'s input.

Attributes	Attacker $\mathcal{A}$'s input	Defender $\mathcal{D}$'s input
Attributes under $\mathcal{D}$'s control but may be known to $\mathcal{A}$ to some extent		
A_1: Training set I_{train}	$a_1 \in [0,1]$	1
A_2: Defense tactic	$a_2 \in \{0,1\}$	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 $\mathcal{D}$ to some extent		
A_6: Manipulation set $\mathcal{M}$	1	$a_6 \in [0,1]$
A_7: Manipulation space	1	$a_7 \in \{0,1\}$
A_8: Attack tactic	1	$a_8 \in \{0,1\}$
A_9: Adversarial examples	1	$a_9 \in [0,1]$

A_1 corresponds to $\mathcal{D}$'s training set I_{train} for learning classifier f. We use $a_1 \in [0,1]$ to represent the extent at which I_{train} is known to $\mathcal{A}$. Let $\hat{I}_{train} \subseteq I_{train}$ be the subset of $\mathcal{D}$'s training files that are known to $\mathcal{A}$. Then, $a_1 = |\hat{I}_{train}|/|I_{train}|$.

A_2 describes $\mathcal{D}$'s tactic, which can be Ensemble Learning (EL), Weight Regularization (WR), Adversarial Training (AT), Input Transformation (IT), Classifier Randomization (CD), Robust Feature (RF), Sanitizing Examples (SE). Let

$A_2 \in \{\mathsf{EL}, \mathsf{WR}, \mathsf{AT}, \mathsf{IT}, \mathsf{CD}, \mathsf{RF}, \mathsf{SE}\}$ and $a_2 \in \{0,1\}$ such that $a_2 = 0$ means $\mathcal{A}$ does not know $\mathcal{D}$'s tactic and $a_2 = 1$ means $\mathcal{A}$ knows $\mathcal{D}$'s tactic. The tactics are defined below.

Definition 1 (ensemble learning or EL [83])**.** *$\mathcal{D}$ uses an ensemble of classifiers for malware detection. Formally, let $\mathcal{H}$ be the set of possible classifiers. Given m classifiers $\{f_i\}_{i=1}^m$ where $f_i \in \mathcal{H}$ and $f_i : \mathcal{Z} \to \mathbb{R}$, let f_i be assigned with weight v_i with $\sum_{i=1}^m v_i = 1$ and $0 \le v_i \le 1$. Then, $f = \sum_{i=1}^m v_i f_i$.*

Definition 2 (weight regularization or WR [68])**.** *$\mathcal{D}$ uses weight regularization to generalize models when optimizing parameter θ of F_θ. Formally, given regularization item Ω, the empirical risk is defined as $\min_\theta \; [\mathcal{L}(\theta, I_{train}) + \Omega(\theta)]$, where $\mathcal{L}(\theta, I_{train})$ is defined in Eq. (1).*

Definition 3 (adversarial training or AT [32])**.** *$\mathcal{D}$ aims to make its classifier f accommodate some information about adversarial files. Formally, let I' denote a set of adversarial file-label pairs, of which the adversarial files may be produced by $\mathcal{D}$, $\mathcal{A}$ or both. Then, $\mathcal{D}$ aims to tune model parameters θ to minimize the empirical risk: $\min_\theta \; [\mathcal{L}(\theta, I_{train} \cup I')]$, where $\mathcal{L}(\theta, I_{train} \cup I')$ can be defined as in Eq. (1).*

Definition 4 (input transformation or IT, adapted from [79])**.** *$\mathcal{D}$ aims to offset $\mathcal{A}$'s manipulations. Formally, let $\mathtt{IT} : \mathcal{Z} \to \mathcal{Z}$ denote an input transformation in the file space. Given file z and transformation $\mathtt{IT}$, the classifier is $f = \varphi(\phi(\mathtt{IT}(z)))$.*

Note that Definition 4 can be extended to accommodate transformations defined in the feature space.

Definition 5 (classifier randomization or CD; adapted from [73])**.** *$\mathcal{D}$ aims to randomize 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). Formally, given a set $\mathcal{H}$ of possible classifiers and an input file z, $\mathcal{D}$ randomly selects m classifiers from $\mathcal{H}$ with replacement, say $\{f_i\}_{i=1}^m$. Then, $f = \frac{1}{m}\sum_{i=1}^m \; f_i(z)$.*

Definition 6 (robust feature or RF; adapted from [84])**.** *$\mathcal{D}$ aims to use the features that can lead to higher detection capability. Formally, let ACC denote the accuracy of a classification function learned from a feature set $S' \subset S$. Given the file-label space $\mathcal{I}$ that contains (adversarial) file-label pairs, the set of robust feature set S^* is $S^* = \underset{S' \subset S}{\arg\max}(\mathrm{ACC}_{\mathcal{I}}(S'))$.*

Definition 7 (sanitizing examples or SE; adapted from [38], [85])**.** *$\mathcal{D}$ aims to detect adversarial files by inventing a function* $\mathsf{flag} : \mathcal{Z} \to \{\mathtt{yes}, \mathtt{no}\}$ *that indicates whether a file is adversarial* (yes) *or not* (no)*.*

A_3 corresponds to $\mathcal{D}$'s feature set S. We use $a_3 \in [0,1]$ to represent the extent at which $\mathcal{A}$ knows about S. Let $\hat{S} \subset S$ denote the features that are known to $\mathcal{A}$. Then, $a_3 = |\hat{S}|/|S|$.

A_4 is $\mathcal{D}$'s learning algorithm F and parameters and hyperparameters [86], [80], [40], [87]. 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.

A_5 corresponds to $\mathcal{D}$'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 [88]. We define $a_5 \in \{0,1\}$ such that $a_5 = 0$ means there is a limit on the number of queries that can be made by $\mathcal{A}$ to f (referred as LQ) and $a_5 = 1$ means there is no limit (referred as FQ).

A_6 corresponds to $\mathcal{A}$'s manipulation set in the file or feature space, which describes perturbations for generating adversarial files (adapted from *perturbation set* in the AML literature [89]). Let Δ be the set of manipulations for perturbing files, with or without preserving their functionalities. We define *file manipulation set* $\mathcal{M}$ as:

$$\begin{aligned}\mathcal{M} \;\; = \;\; & \{\delta : (z' \leftarrow \mathcal{A}(z,\delta)) \wedge (\mathtt{true} \leftarrow \mathcal{O}(z,z')) \wedge \\ & (z \in \mathcal{Z}) \wedge (\delta \in \Delta) \wedge (z' \neq z)\}.\end{aligned}$$

Note that computer files can be manipulated in many ways (e.g., code injection, string encryption, packing, and object replacement [5], [6], [7], [90], [49], [36], [41], [29], [91], [73]). Let $\blacktriangle$ denote the set of possible manipulations in the feature space. We define *feature manipulation set* $\mathbf{M}$ corresponding to file manipulation set $\mathcal{M}$ as:

$$\begin{aligned}\mathbf{M} \;\; = \;\; & \{\delta_{\mathbf{x}} = \mathbf{x}' - \mathbf{x} : (\mathbf{x} = \phi(z)) \wedge (\mathbf{x}' = \phi(z')) \wedge \\ & (z' \leftarrow \mathcal{A}(z,\delta)) \wedge (\delta \in \mathcal{M}) \wedge (z \in \mathcal{Z})\}.\end{aligned}$$

A_7 indicates $\mathcal{A}$'s manipulation space, meaning manipulations are conducted in file space $\mathcal{Z}$, feature space $\mathcal{X}$, or both (referred as $\mathcal{ZX}$).

A_8 corresponds to $\mathcal{A}$'s tactic. We consider two tactics: *evasion attack* and *poisoning attack*. For the evasion attack, we consider three variants: basic evasion (BE), optimal evasion 1 ($\mathsf{OE1}$) and optimal evasion 2 ($\mathsf{OE2}$). For the poisoning attack, we consider two variants: basic poisoning (BP) and optimal poisoning (OP). Correspondingly, we have $A_8 \in \{\mathsf{BE}, \mathsf{OE1}, \mathsf{OE2}, \mathsf{BP}, \mathsf{OP}\}$. These tactics are elaborated below, while noting that they do not explicitly call oracle $\mathcal{O}$ because definitions of manipulation sets $\mathcal{M}$ and $\mathbf{M}$ already assure that manipulations preserve functionalities of non-adversarial files.

Definition 8 (basic evasion or BE [32])**.** *$\mathcal{A}$ manipulates a malicious file z, which is correctly classified by $\mathcal{D}$'s classifier f as $+ \leftarrow f(z)$, into an adversarial file z' such that $- \leftarrow f(z')$. Formally, $\mathcal{A}$ achieves the following for $z \in \mathcal{Z}$ with $+ \leftarrow f(z)$:*

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

Definition 9 (optimal evasion 1 or $\mathsf{OE1}$; adapted from [92])**.** *It 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 $z' \in \mathcal{Z}$. Formally, $\mathcal{A}$ attempts to achieve the following for $z \in \mathcal{Z}$ with $+ \leftarrow f(z)$:*

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

Let $\mathcal{A}$ have (i) a classifier $g : \mathcal{Z} \to \mathbb{R}$ that can mimic $\mathcal{D}$'s classifier f and (ii) a loss function $L_{\mathcal{A}} : \mathcal{R} \times \mathcal{Y} \to \mathcal{R}$ that measures g's error, which indicates $\mathcal{A}$' failure in evasion.

Definition 10 (optimal evasion 2 or OE2; adapted from [28]). *It is the same as* BE*, except that* $\mathcal{A}$ *minimizes the loss* $L_{\mathcal{A}}$ *when generating adversarial malicious file* $z' \in \mathcal{Z}$*. Formally,* $\mathcal{A}$ *attempts to achieve the following for* $z \in \mathcal{Z}$ *with* $+ \leftarrow f(z)$*:*

$$\min_{z'} L_{\mathcal{A}}(g(z'), -) \text{ s.t. } (z' \leftarrow \mathcal{A}(z, \delta)) \wedge (\delta \in \mathcal{M}) \wedge (\Gamma(z', z) \approx 0).$$

Note that Definition 9 is about minimizing $\mathcal{A}$'s manipulations, whereas Definition 10 is about minimizing $\mathcal{A}$'s loss.

Let $I'_{poison} \subset \mathcal{I}$ be a set of adversarial file-label pairs obtained by manipulating non-adversarial files in I_{poison}. Let $I'_{train} \leftarrow I_{train} \cup I'_{poison}$ be the contaminated training data $\mathcal{D}$ for learning classifier $\widetilde{f}$ with parameters $\widetilde{\theta}$.

Definition 11 (basic poisoning or BP [45]). *Let* f *be the classifier learned from training dataset* I_{train}*. Given a set* I_{target} *of files where* $+ \leftarrow f(\dot{z})$ *for* $\dot{z} \in I_{target}$ *and a set* I_{poison} *of non-adversarial files,* $\mathcal{A}$ *attempts to perturb files in* I_{poison} *to adversarial ones* $I'_{poison} = \{(\mathcal{A}(z, \delta), \mathcal{A}(y)) : ((z, y) \in I_{poison}) \wedge (\delta \in \mathcal{M}) \wedge (\mathcal{A}(y) \in \{+, -\})\}$ *such that classifier* $\widetilde{f}$ *learned from* $I'_{train} \leftarrow I_{train} \cup I'_{poison}$ *misclassify the files in* I_{target}*. Formally, the attacker wants to achieve the following for* $\forall\ \dot{z} \in I_{target}$*:* $- \leftarrow \widetilde{f}(\dot{z})$ *where* $\widetilde{f}$ *is learned from* $I'_{train} \leftarrow I_{train} \cup I'_{poison}$*.*

Note that Definition 11 accommodates the attack tactic that $\mathcal{A}$ directly manipulates labels of the files in I_{poison} [93].

Definition 12 (optimal poisoning or OP [82]). *It is the same as* BP*, except that* $\mathcal{A}$ *attempts to maximize the loss when using classifier* $\widetilde{f}$ *with parameter* $\widetilde{\theta}$ *to classify files in* I_{target}*. Formally,*

$$\max_{I'_{poison}} \mathcal{L}(\widetilde{\theta}, I_{target}) \text{ where } \widetilde{\theta} = \arg\min_{\theta} \mathcal{L}(\theta, I_{train} \cup I'_{poison}).$$

Definition 12 has multiple variants by considering surrogate malware detector [82], bounds on $|I'_{poison}|$ [94], or bounds on perturbations I'_{poison} [39].

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

$$\mathcal{X}_{\mathbf{M}} = \{\mathbf{x}' : (\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}) \wedge (\delta_{\mathbf{x}} \in \mathbf{M})\} = \{\phi(z') : z' \in \mathcal{Z}_{\mathcal{M}}\}.$$

On the Usefulness of the Preceding Definitions. The definitions formulate a partial order in the attribute space, paving the way for comparing attacks. For example, there are many grey-box attacks between *black-box* attack $(a_1, a_2, a_3, a_4, a_5) = (0, 0, 0, 0, 0)$ and *white-box* attack $(a_1, a_2, a_3, a_4, a_5) = (1, 1, 1, 1, 1)$, as illustrated by Figure 5 of Appendix A.

E. Systematizing Defenses

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

Defender's Input. As highlighted in Table II, $\mathcal{D}$'s input includes attributes $A_1, \ldots, A_5$, which are under $\mathcal{D}$'s control, and the extent $a_6, \ldots, a_9$ at which $\mathcal{D}$ 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 $\mathcal{D}$ knows $\mathcal{A}$'s file manipulation set $\mathcal{M}$ or feature manipulation set $\mathbf{M}$, depending on $\mathcal{A}$'s manipulation space $A_7 \in \{\mathcal{Z}, \mathcal{X}, \mathcal{ZX}\}$. Let $\hat{\mathcal{M}} \subseteq \mathcal{M}$ and $\hat{\mathbf{M}} \subseteq \mathbf{M}$ respectively denote the subset of $\mathcal{A}$'s manipulation set known to $\mathcal{D}$. Then, $a_6 = |\hat{\mathcal{M}}|/|\mathbf{M}|$ or $a_6 = |\hat{\mathbf{M}}|/|\mathbf{M}|$ depending on A_7.

We define $a_7 \in \{0, 1\}$ such that $a_7 = 0$ means $\mathcal{D}$ does not know $\mathcal{A}$'s manipulation space for certain and $a_7 = 1$ means that $\mathcal{D}$ knows $\mathcal{A}$'s manipulation space (i.e., $\mathcal{Z}$, $\mathcal{X}$, or $\mathcal{ZX}$).

We define $a_8 \in \{0, 1\}$ such that $a_8 = 0$ means $\mathcal{D}$ does not know $\mathcal{A}$'s tactic $A_8 \in \{\mathsf{BE}, \mathsf{OE1}, \mathsf{OE2}, \mathsf{BP}, \mathsf{OP}\}$ and $a_8 = 1$ means $\mathcal{D}$ knows $\mathcal{A}$'s tactic (and its parameters, if applicable).

We use $a_9 = |\mathcal{Z}^*_{\mathcal{M}}|/|\mathcal{Z}_{\mathcal{M}}|$ to represent the extent at which $\mathcal{D}$ 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 $\mathcal{Z}^*_{\mathcal{M}} \subseteq \mathcal{Z}_{\mathcal{M}}$ is known to $\mathcal{D}$.

On the Usefulness of the Preceding Definitions. The definitions formulate a partial order in the attribute space, paving the way for comparing defenses. For example, there are many grey-box defenses between *black-box* defense $(a_6, a_7, a_8, a_9) = (0, 0, 0, 0)$ and *white-box* defense $(a_6, a_7, a_8, a_9) = (1, 1, 1, 1)$, as illustrated by Figure 6 of Appendix B.

F. 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), Classification Robustness (CR), Detection Robustness (DR) and Training Robustness (TR).

Definition 13 (RR or (ϵ, η)-robust feature extraction; adapted from [72]). *It says that two similar files have similar feature representations. Formally, given constants* $\epsilon, \eta \in [0, 1]$*, and files* $z, z' \in \mathcal{Z}$ *such that* $(\Gamma(z, z') \approx 0) \wedge (\mathtt{true} \leftarrow \mathcal{O}(z, z'))$*, 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 14 (CR or ϵ-robust classification [95]). *It says that two similar feature representations lead to the same label. Formally, given constant* $\epsilon \in [0, 1]$ *as in Definition 13 and any feature vectors* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$*, we say classification function* φ *is* ϵ*-robust if*

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

Definitions 13 and 14 collectively say that when two files z and z' are similar (i.e., considering small manipulations), they would be classified into the same label with a high probability.

Definition 15 (DR or η-robust detection; adapted from [33]). *It says that when large manipulations are relevant, classifier* f *should classify two very different files with the same label as long as they have the same functionality. Formally, given constant* $\eta \in [0, 1]$ *and any files* $z, z' \in \mathcal{Z}$ *such that* $(\Gamma(z, z') >> 0) \wedge (\mathtt{true} \leftarrow \mathcal{O}(z, z'))$*, we say malware detector* f *is* η*-robust if*

$$\mathbb{P}((f(z) > \tau) \wedge (f(z') > \tau)) > 1 - \eta.$$

Let adversarial files set I'_{poison} be restricted to $|I'_{poison}| \leq \gamma|I_{train}|$ for some constant $\gamma \in [0,1]$. Let classifier $\widetilde{f}$ be learned from $I_{train} \cup I'_{poison}$.

Definition 16 (TR or (γ, ζ)-robust training; adapted from [94])**.** *It says classifier $\widetilde{f}$ learned from poisoned training set can predict as well as f with a high probability. Formally, given classifiers f learned from I_{train} and $\widetilde{f}$ learned from $I_{train} \cup I'_{poison}$ where $|I'_{poison}| \leq \gamma|I_{train}|$, and small constants $\zeta \in [0,1]$, we say $\widetilde{f}$ is (γ, ζ)-robust if $\forall z \in \mathcal{Z}$*

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

III. 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 $\mathcal{D}$'s objective, input, assumptions, the security properties that are achieved, and the types of enhanced malware detectors (e.g., Windows vs. Android). We present attacks (defenses) in their chronological order of publication and then summarize them in a succinct table. 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 $\mathcal{D}$'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.

A. Systematizing Attack Literature

Biggio et al. [28] propose optimal evasion attack tactic OE2 in feature space $\mathcal{X}$, dubbed *gradient descent and kernel density estimation*. Feature representation is the number of appearances of hand-selected keywords (e.g., JavaScript); feature manipulation set $\mathbf{M}$ corresponds to feature injections. $\mathcal{A}$'s loss has two parts: one accommodates $\mathcal{D}$'s classifier f misclassifying adversarial feature vector $\mathbf{x}'$ and the other corresponds to lifting $\mathbf{x}'$ into a populated region of benign files. Under the Oracle and Measurability Assumptions, $\Gamma(z,z') \approx 0$ is replaced by $C(\mathbf{x},\mathbf{x}') \leq \epsilon$. The authors use gradient descent to solve OE2 optimization (Definition 10). When $\mathcal{D}$ employs no countermeasures, knowing $\mathcal{D}$'s feature set S and learning algorithm F is sufficient for $\mathcal{A}$ to evade $\mathcal{D}$'s detector. This attack and its variants can evade PDF malware detectors [29], [53], [105], [84], PE malware detector [75] and Flash malware detector [106]. In summary, $\mathcal{A}$'s input is $(a_1,\ldots,a_5|A_6,\cdots,A_9) = (1,1,1,1,1|\mathbf{M},\mathcal{X},\mathsf{OE2},\mathcal{X}_\mathbf{M}) \vee (0,0,1,*,0|\mathbf{M},\mathcal{X},\mathsf{OE2},\mathcal{X}_\mathbf{M})$ and $\mathcal{A}$ breaks the CR property.

Smutz and Stavrou [96] propose a mimicry attack to modify features of a malicious file to mimic benign ones, while assuming that $\mathcal{A}$ knows $\mathcal{D}$'s classifier f. The attack is validated using PDF malware detector. The attack works under the Measurability assumption with input $(a_1,\ldots,a_5|A_6,\cdots,A_9) = (1,1,1,1,1|\Delta,\mathcal{X},\mathsf{OE2},\mathcal{X})$ and breaks the CR property.

Maiorca et al. [97] propose a *reverse mimicry* attack against PDF malware detectors. Instead of modifying malicious files to mimic benign ones, $\mathcal{A}$ embeds malicious payload into benign files. The attack works under the Oracle assumption with input $(a_1,\ldots,a_5|A_6,\cdots,A_9) = (0,0,0,0,0|\mathcal{M},\mathcal{Z},\mathsf{BE},\mathcal{Z}_\mathcal{M})$ and breaks the DR property.

Šrndić and Laskov [29] investigate the mimicry attack [107], [29] and the aforementioned *gradient descent and kernel density estimation* attack against PDF malware detectors (e.g., PDFrate), without being constrained by small manipulations. Manipulation set $\mathcal{M}$ corresponds to adding instructions into PDF files. The attack first derives perturbed feature vectors using a surrogate model, then maps the perturbed feature vectors to the file space, and finally inject manipulations at pre-determined positions of a file to obtain an adversarial file. The attack works under the Oracle assumption with input $(a_1,\ldots,a_5|A_6,\cdots,A_9) = (0,0,*,0,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M}) \vee (0,0,*,*,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M}) \vee (1,0,*,0,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M}) \vee (1,0,*,*,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M})$ and breaks the DR property.

Xu et al. [41] propose an evasion attack against PDF malware detectors [96], [29], [108]. $\mathcal{A}$ can query $\mathcal{D}$'s detector f at will. Manipulation set $\mathcal{M}$ corresponds to deleting, inserting, and modifying PDF components (e.g., `/Root/Kids/1`). $\mathcal{A}$ manipulates a malware file to obtain a candidate and uses $\mathcal{O}$ (e.g., sandbox) and detector f to decide if the candidate is functionality-preserving and adversarial. This attack works with empirical Oracle and input $(a_1,\cdots,a_5|A_6,\cdots,A_9) = (0,0,0,0,1|\mathcal{M},\mathcal{Z},\mathsf{BE},\mathcal{Z}_\mathcal{M})$ and breaks DR.

Yang et al. [78] propose an evasion attack against Android malware detectors. $\mathcal{A}$ uses a surrogate model and manipulation set $\mathcal{M}$ corresponding to Android obfuscations (e.g., repacking). The attack works under the Oracle assumption with input $(a_1,\cdots,a_5|A_6,\cdots,A_9) = (0,0,0,0,1|\mathcal{M},\mathcal{Z},\mathsf{BE},\mathcal{Z}_\mathcal{M})$ and breaks the DR property.

Hu and Tan [98] propose an evasion attack using Generative Adversarial Networks (GAN) [109]. $\mathcal{A}$ modifies the binary representation of malicious files in Windows API calls to mimic benign files, by flipping some feature values from 0 to 1. $\mathcal{A}$ learns a generator, say G_{θ_g}, and a discriminator from $\mathcal{A}$'s training dataset, which is different from $\mathcal{D}$'s. An adversarial feature vector can be generated using $\mathbf{x}' = \max(\mathbf{x}, \mathrm{round}(G_{\theta_g}(\mathbf{x},\mathbf{a})))$, where $\mathbf{a}$ is a vector of noise, round is the round function, and $\max$ means element-wise maximum. 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 predicted by $\mathcal{D}$'s detector f. Hu and Tan [99] propose another evasion attack in [110]. Both attacks work under the Oracle assumption with input $(a_1,\cdots,a_5|A_6,\cdots,A_9) = (0,0,1,0,1|\mathbf{M},\mathcal{X},\mathsf{BE},\mathcal{X}_\mathbf{M})$ and breaks the DR property.

Demontis et al. [36] propose the optimal evasion OE2 tactic to perturb important features in terms of their weights in the linear function $\varphi(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b$ of SVM classifier, where $\mathbf{w} = (w_1,w_2,\cdots,w_d)$ is a weight vector, b is bias, and d is the input dimension. $\mathcal{A}$ manipulates the x_i's with largest $|w_i|$'s as follows: decrease x_i if $w_i > 0$, increase x_i if $w_i < 0$, and do nothing otherwise, while obeying the manipulation set $\mathbf{M}$ that corresponds to feature injection and removal. The

TABLE III: Summary AMD attacks, where ✓indicates applicability, ● means 0, ○ means 1, ⊙ means a value in $[0, 1]$.

Attack (in chronological order)	Attack Objective			Attack Input									Assumptions				Broken Properties				Type of Victim Malware Detector		
	Untargeted objective	Targeted objective	Frustration objective	A_1: Training set I_{train}	A_2: Defense tactic	A_3: Feature set	A_4: Learning algorithm	A_5: Response	A_6: Manipulation set	A_7: Manipulation space	A_8: Attack tactic	A_9: Adversarial example	IID assumption	Oracle assumption	Measurability assumption	Smoothness assumption	RR: Representation Robustness	CR: Classification Robustness	DR: Detection Robustness	TR: Training Robustness	Windows Program	Android Package	PDF
Biggio et al. [28]	✓			○ ●	○ ●	○ ○	○ ⊙	○ ●	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓	✓			✓					✓
Smutz and Stavrou [96]	✓			○	○	○	○	○	▲	$\mathcal{X}$	OE2	$\mathcal{X}$			✓			✓					✓
Maiorca et al. [97]	✓			●	●	●	●	●	$\mathcal{M}$	$\mathcal{Z}$	BE	$\mathcal{Z}_{\mathcal{M}}$		✓					✓				✓
Šrndić and Laskov [29]	✓			● ● ○ ○	● ● ● ●	⊙ ⊙ ⊙ ⊙	● ⊙ ● ⊙	● ● ● ●	$\mathbf{M}$	$\mathcal{Z}\mathcal{X}$	BE	$\mathcal{X}_{\mathbf{M}}$		✓					✓				✓
Xu et al. [41]	✓			●	●	●	●	○	$\mathcal{M}$	$\mathcal{Z}$	BE	$\mathcal{Z}_{\mathcal{M}}$							✓				✓
Yang et al. [78]	✓			●	●	●	●	○	$\mathcal{M}$	$\mathcal{Z}$	BE	$\mathcal{Z}_{\mathcal{M}}$		✓					✓			✓	
Hu and Tan [98]	✓			●	●	○	●	○	$\mathbf{M}$	$\mathcal{X}$	BE	$\mathcal{X}_{\mathbf{M}}$			✓				✓		✓		
Hu and Tan [99]	✓			●	●	○	●	○	$\mathbf{M}$	$\mathcal{X}$	BE	$\mathcal{X}_{\mathbf{M}}$		✓					✓		✓		
Demontis et al. [36]	✓			○ ●	○ ●	○ ○	○ ⊙	○ ●	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓	✓			✓				✓	
Grosse et al. [32]	✓			●	○	○	○	○	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓	✓		✓	✓				✓	
Chen et al. [100]	✓			○	○	○	○	○	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓		✓		✓			✓		
Khasawneh et al. [73]	✓			●	●	⊙	●	○	$\mathbf{M}$	$\mathcal{Z}\mathcal{X}$	BE	$\mathcal{X}_{\mathbf{M}}$		✓				✓			✓		
Rosenberg et al. [101]	✓			●	●	○	●	○	$\mathbf{M}$	$\mathcal{Z}\mathcal{X}$	BE	$\mathcal{X}_{\mathbf{M}}$		✓			✓	✓			✓		
Dang et al. [42]	✓			●	●	●	●	●	$\mathcal{M}$	$\mathcal{Z}$	BE	$\mathcal{Z}_{\mathcal{M}}$							✓				✓
Anderson et al. [91]	✓			●	●	●	●	○	$\mathcal{M}$	$\mathcal{Z}$	OE2	$\mathcal{Z}_{\mathcal{M}}$		✓			✓	✓			✓		
Al-Dujaili et al. [33]	✓			●	○	○	○	○	$\mathbf{M}$	$\mathcal{X}$	BE	$\mathcal{X}_{\mathbf{M}}$		✓					✓		✓		
Kreuk et al. [102]	✓			●	○	○	○	○	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓			✓	✓			✓		
Kolosnjaji et al. [75]	✓			●	○	○	○	○	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓			✓	✓			✓		
Suciu et al. [103]	✓			●	○	○	○	○	$\mathbf{M}$	$\mathcal{X}$	OE2	$\mathcal{X}_{\mathbf{M}}$		✓			✓	✓			✓		
Chen et al. [104]	✓			● ● ○ ○	● ● ● ●	○ ○ ○ ○	● ● ● ●	● ○ ● ○	$\mathbf{M}$ $\mathbf{M}$	$\mathcal{X}$ $\mathcal{X}$	OE1 OE2	$\mathcal{X}_{\mathbf{M}}$ $\mathcal{X}_{\mathbf{M}}$		✓	✓			✓				✓	
Muñoz-González et al. [82]			✓	○ ○	○ ●	○ ○	○ ●	○ ●	$\mathbf{M}$	$\mathcal{X}$	OP	$\mathcal{X}_{\mathbf{M}}$		✓						✓	✓		
Chen et al. [38]	✓			○ ● ○	○ ● ●	○ ● ⊙	○ ○ ○	○ ○ ○	$\mathbf{M}$	$\mathcal{X}$	BP	$\mathcal{X}_{\mathbf{M}}$		✓						✓		✓	
Suciu et al. [39]		✓		⊙ ○ ○ ○	● ● ● ○	○ ⊙ ○ ○	⊙ ⊙ ● ○	● ● ● ○	$\mathbf{M}$	$\mathcal{Z}\mathcal{X}$	BP	$\mathcal{X}_{\mathbf{M}}$		✓						✓		✓	

attack is waged against Android malware detector learned from the Drebin dataset (see Appendix C). The attack works under the Oracle and Measurability assumptions with input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}}) \vee (0, 0, 1, *, 0|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}})$ and breaks the CR property.

Grosse et al. [32] propose a variant of JSMA to evade DNN-based malware detector learned from the Drebin dataset. $\mathcal{A}$ injects manifest features to manipulate Android Packages and generates adversarial files from $\mathcal{D}$'s detector f directly, while $\mathcal{D}$ may employ countermeasures such as adversarial training. The attack works under the Oracle and Measurability assumptions with input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}})$ and breaks RR and CR.

Chen et al. [100] propose considering manipulation cost $C(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} c_i |x_i - x'_i|$, where c_i is the hardness of changing the ith feature while preserving malware's functionality. $\mathcal{A}$ uses a *wrapper-based* feature selection algorithm (e.g., max-relevance or information-gain [100], [44], [84], [111]) to select important features for manipulation. The attack is waged against Windows PE malware detector that uses hand-crafted Windows API calls as features and the binary feature representation. This attack works under the Oracle and Measurability assumptions with input $(a_1, \ldots, a_5|A_7, \cdots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}}))$ and breaks CR.

Khasawneh et al. [73] propose evading hardware malware detector learned from low-level hardware features (e.g., instruction mixes [112]). $\mathcal{A}$ knows some features used by $\mathcal{D}$. $\mathcal{A}$ first performs reverse-engineering to obtain a surrogate model of $\mathcal{D}$'s detector, and then generate adversarial files to evade the surrogate model. Manipulation set $\mathcal{M}$ corresponds to injecting instructions. The attack works under the Oracle assumption with input $(a_1, \ldots, a_5|A_6, \cdots, A_9) = (0, 0, *, 0, 1|\mathbf{M}, \mathcal{Z}\mathcal{X}, \mathsf{BE}, \mathcal{X}_{\mathcal{M}})$ and breaks CR.

Rosenberg et al. [101] use a Recurrent Neural Network

(RNN) surrogate model to evade malware detectors learned from Windows API calls. $\mathcal{A}$'s training data is different from $\mathcal{D}$'s, but their labels are obtained from $\mathcal{A}$'s detector. $\mathcal{A}$ can further augment its training data using the Jacobian-based augmentation technique [113]. $\mathcal{A}$ modifies an API sequence in the direction of the ℓ_∞ normalized gradient of the loss function. Manipulation set $\mathbf{M}$ corresponds to inserting no-op API calls. $\mathcal{A}$ can evade DNN- and RNN-based detectors. The attack works under the Oracle assumption with input $(a_1, \ldots, a_5|A_6, \cdots, A_9) = (0, 0, 1, 0, 1|\mathbf{M}, \mathcal{ZX}, \mathsf{BE}, \mathcal{X}_\mathbf{M})$ and breaks RR and CR.

Dang et al. [42] propose an evasion attack against malware detectors (e.g., PDFrate). Given a malicious file z, $\mathcal{A}$ uses the hill-climbing algorithm [114] to iteratively generate adversarial file z' from z as follows: (i) $\mathcal{A}$ generates a sequence of variants from each candidate (or z in the initial iteration). (ii) $\mathcal{A}$ queries variant to $\mathcal{O}$ and $\mathcal{D}$'s detector f. (iii) $\mathcal{A}$ succeeds when obtaining z' where $(\texttt{True} \leftarrow \mathcal{O}(z, z')) \wedge (- \leftarrow f(z'))$; otherwise, $\mathcal{A}$ uses a score function to select candidates for the next iteration or aborts after reaching a threshold number of iterations. The attack has input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 0, 0, 0, 0|\mathcal{M}, \mathcal{Z}, \mathsf{BE}, \mathcal{Z}_\mathcal{M})$ and breaks DR.

Anderson et al. [77], [91] propose a Reinforcement Learning-based evasion attack against PE malware detectors. $\mathcal{A}$'s input includes labels predicted by $\mathcal{A}$'s detector f. Manipulation set $\mathcal{M}$ corresponds to code injection (e.g., API insertion) and code deletion (e.g., change section names). $\mathcal{A}$ learns to evade f by applying a small number of manipulations to a malicious PE file, meaning $\Gamma(z, z') \leq 10$. Experimental results show that the attack is not as effective as others (e.g., gradient-based methods). The attack works under the Oracle assumption with input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 0, 0, 0, 1|\mathcal{M}, \mathcal{Z}, \mathsf{OE2}, \mathcal{Z}_\mathcal{M})$ and breaks RR and CR.

Al-Dujaili et al. [33] propose evasion attacks against DNN-based malware detectors. $\mathcal{A}$ generates adversarial files in feature space $\mathcal{X}$ by maximizing $\mathcal{D}$'s loss and using possibly large perturbations. Manipulation set $\mathbf{M}$ corresponds to flipping 0 to 1 in the binary feature representation. The attack works under the Oracle assumption with input $(a_1, \ldots, a_5|A_6, \cdots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BE}, \mathcal{X}_\mathbf{M})$ and breaks DR.

Kreuk et al. [102] propose an OE2 attack against MalConv [25], which is an end-to-end Windows PE malware detector (see Appendix D). $\mathcal{A}$ generates adversarial files by manipulating non-adversarial malicious files in the direction of normalized gradients of the loss function [115], [116], [117]. For example, the ℓ_∞-norm based attack in the feature space is $\mathbf{x}' = \mathbf{x} - \epsilon \cdot \text{sign}(\nabla_\mathbf{x}(L(F_\theta(\mathbf{x}), -))$, where $\text{sign}(a) = +1$ (-1) if $a \geq 0$ $(a < 0)$, and F_θ, ϵ, L are respectively a DNN, a perturbation bound, and the DNN's loss function. Feature manipulation set $\mathbf{M}$ corresponds to appending instructions at pre-determined locations of PE files. The attack works under the Oracle assumption with input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$ and breaks RR and CR.

Kolosnjaji et al. [75] and Suciu et al. [103] independently propose gradient-based attacks to evade MalConv [25]. The former uses (i) file manipulation set $\mathcal{M}$ corresponding to appending instructions at the end of a file and (ii) ℓ_2 normalized gradients of the classifier's loss function to modify embedding codes (i.e., features corresponding to modifiable instructions in PE files). The latter perturbs embedding codes in the direction of ℓ_∞ normalized gradients of the classifier's loss function, while aiming to add instructions to PE files. Both attacks work under the Oracle assumption with input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$ and break RR and CR.

Chen et al. [104] propose evasion attacks against two Android malware detectors, MaMaDroid and Drebin [21], [20], in the feature space, using manipulation set $\mathbf{M}$ corresponding to manifest features (e.g., *activities*) and APIs injections. $\mathcal{A}$ evades MaMaDroid by using the optimal evasions OE1 and OE2 tactics and evades Drebin by using OE2, where OE1 is solved using the method proposed in [92] and OE2 is solved using the JSMA method [118]. The OE2 tactic works under the Oracle and Measurability assumptions, with four kinds of input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, 0, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$, and breaks CR. The OE1 attack works under the same assumptions with the same input except using tactic OE1, and breaks CR.

Muñoz-González et al. [82] propose the optimal poisoning OP, which is NP-hard. The optimization problem can be transformed into an easier problem that can be tackled using the *back-gradient* optimization method when the KKT condition holds [80], [119], [82]. The attack is waged against the Windows PE malware detector. Feature set S 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, allowing $\mathcal{A}$ to derive I'_{poison} from $\mathcal{D}$'s detector f; and the other uses grey-box input, allowing $\mathcal{A}$ to know $\mathcal{D}$'s training set as well as feature set and to train a surrogate detector. The attack works under the Oracle assumption with input $(a_1, \ldots, a_5|A_6, \cdots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OP}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{OP}, \mathcal{X}_\mathbf{M})$ and breaks TR.

Chen et al. [38] propose a basic poisoning BP tactic against Android malware detectors. Feature manipulation set $\mathbf{M}$ corresponds to feature injection or removal, but not both. $\mathcal{A}$ poisons $\mathcal{D}$'s training data by injecting adversarial malicious files with label $-$. The attack works under the Oracle assumption with input $(a_1, \ldots, a_5|A_6, \cdots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BP}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 0, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BP}, \mathcal{X}_\mathbf{M}) \vee (1, 0, *, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BP}, \mathcal{X}_\mathbf{M})$ and breaks TR.

Suciu et al. [39] propose a basic poisoning attack to obtain I'_{poison} by applying small manipulations to non-adversarial benign files, whose labels are obtained by querying VirusTotal. $\mathcal{A}$'s objective is to make $\mathcal{D}$'s classifier f misclassify a target malware file z_{mal} as benign. $\mathcal{A}$ proceeds as following: (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 [118] to manipulate z_{ben} to z'_{ben} by applying a small perturbation so that they have similar feature representations; (iii) use the combination of the pristine training set and the crafted training set to train classifier $\widetilde{f}$; and (iv) add z'_{ben} to I'_{poison} if it lowers the classification accuracy slightly, and reject it otherwise. The attack is waged against the Drebin detector [20]. This attack works under the Oracle assumption with input $(a_1,\ldots,a_5|A_6,\cdots,A_9) = (*,0,1,*,0|\mathbf{M},\mathcal{ZX},\mathsf{BP},\mathcal{X}_\mathbf{M}) \vee (1,1,1,1,1|\mathbf{M},\mathcal{ZX},\mathsf{BP},\mathcal{X}_\mathbf{M}) \vee (1,0,*,*,0|\mathbf{M},\mathcal{ZX},\mathsf{BP},\mathcal{X}_\mathbf{M}) \vee (1,0,1,0,0|\mathbf{M},\mathcal{ZX},\mathsf{BP},\mathcal{X}_\mathbf{M})$ and breaks TR.

Summary and Drawing Insights (via machine learning). Table III succinctly summarizes the preceding attacks. From Table III and the preceding discussion we draw:

Insight 1 (Insights manually drawn)**.** *(i) Untargeted attack has been most extensively investigated; targeted and frustration attacks have been little investigated. (ii) Evasion attack has been much more extensively studied than poisoning attack. (iii) The* Oracle *assumption has been widely made. (iv) Knowing defender's feature set is critical to the attacker's success, suggesting the importance of keeping defender's feature set secret or randomizing defender's feature set.*

In order to draw further insights, we propose applying ML to the systematized structured data in Table III. We propose formulating *insights learning* as a multiclass classification problem. We use attributes of attack objective, attack input and assumption as features and treat the broken properties as the labels to learn. We preprocess data as follows: for features corresponding to attack objective and assumption, we use 1 to represent "applicable" and 0 otherwise; for features corresponding to $A_1,\ldots,A_5$, we set $a_i = 0.5$ for $1 \le i \le 5$; for features corresponding to categorical $A_6,\ldots,A_9$, we use integers to represent the elements (e.g., 0,1,2 respectively representing $\blacktriangle, \mathbf{M}, \mathcal{M}$ for A_6). This leads to a dataset of 41 attacks, each of which has 16 dimensions. We use the dataset to train a Random Forest model with default hyper-parameters in scikit-learn [120]. We focus on learning important attributes.

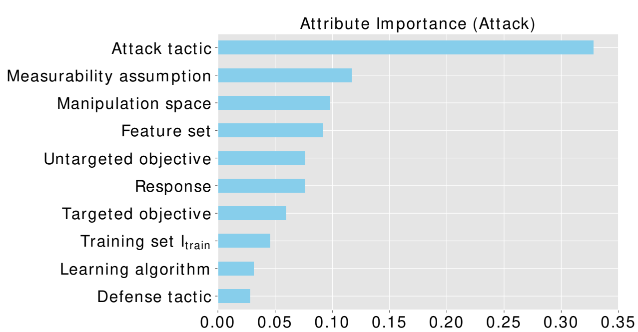

Fig. 2: Insights learning from systematized AMD attacks via attribute importance.

Figure 2 highlights the most important features, from which we make the following observations. (i) Attack tactic is the most important feature in determining the security property that is broken. This can be explained because BE breaks DR, OE2 usually breaks CR and RR, and poisoning attacks (BP and OP) break TR. (ii) The Measurability assumption is the second important feature. This can be explained because RR and CR are defined with respect to feature-space small-manipulation attacks, which need this assumption. (iii) Manipulation space is the third important feature perhaps because small manipulations (corresponding to CR) is usually used in the feature space; whereas, large manipulations are often used in file space or both spaces. (iv) Feature set is the fourth important feature, resonating the manually-drawn insight (iv) mentioned above.

Insight 2 (Insights learned via ML)**.** *Technique-wise, attack tactic (a core part of the threat model) largely determines what security property is broken. Methodology-wise, ML is an effective method for insights learning in SoK studies.*

B. Systematizing Defense Literature

Biggio et al. [121] propose a one-and-a-half-class classifier against evasion attacks, which aims to leverage the insight — the decision boundary of one-class classifiers is often tighter than that of two-class classifiers — to assure that outliers can be correctly classified [121]. The defense enhances the PDF malware detector against the gradient descent-based attack [28] with input $(a_1,\cdots,a_5|A_6,\cdots,A_9) = (1,1,1,1,1|\mathbf{M},\mathcal{X},\mathsf{OP2},\mathcal{X}_\mathbf{M}) \vee (0,0,1,*,0|\mathbf{M},\mathcal{X},\mathsf{OP2},\mathcal{X}_\mathbf{M})$. The defense works under the IID assumption with input $(A_1,\cdots,A_5|a_6,\cdots,a_9) = (I_{train},\mathsf{EL},S,F_\theta,\mathsf{FQ}|0,0,0,0)$ and achieves the CR property.

Smutz and Stavrou [69] propose an ensemble classifier to defend against 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 attacks [29] with input $(a_1,\ldots,a_5|A_6,\cdots,A_9) = (0,0,*,0,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M}) \vee (0,0,*,*,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M}) \vee (1,0,*,0,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M}) \vee (1,0,*,*,0|\mathbf{M},\mathcal{ZX},\mathsf{BE},\mathcal{X}_\mathbf{M})$.

The defense works under the IID assumption with input $(A_1,\cdots,A_5|a_6,\cdots,a_9) = (I_{train},\mathsf{SE},S,F_\theta,\mathsf{FQ}|0,0,0,0)$ and achieves DR.

Zhang et al. [84] investigate how to use adversarial feature selection to defend against evasion attacks. The defense enhances PDF malware detectors against the gradient descent-based attack, which can be succinctly described as $(A_1,\cdots,A_5|a_6,\cdots,a_9) = (1,1,1,1,1|\mathbf{M},\mathcal{X},\mathsf{OE2},\mathcal{X}_\mathbf{M}) \vee (0,0,1,*,0|\mathbf{M},\mathcal{X},\mathsf{OE2},\mathcal{X}_\mathbf{M})$. The defense works under the IID, Oracle Measurability and smoothness assumptions with input $(A_1,\cdots,A_5|a_6,\cdots,a_9) = (I_{train},\mathsf{RF},S,F_\theta,\mathsf{FQ}|1,1,0,0)$, and achieves RR and CR.

Demontis et al. [36] enhance the linear Drebin malware detector $\varphi(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b$ by using box-constraint weights $w_i^{(lb)} \le w_i \le w_i^{(ub)}$ for $1 \le i \le d$ and leveraging that a classifier's sensitivity to ℓ_1-norm based manipulations is bounded by the ℓ_∞-norm of the weights. This defense enhances the Drebin detector against the mimicry attack with input $(a_1,\cdots,a_5|A_6,\cdots,A_9) = (0,0,1,0,0|\mathbf{M},\mathcal{X},\mathsf{BE},\mathcal{X}_\mathbf{M})$, the

TABLE IV: Summary of AMD defenses, where ✓ indicates applicability, ● means 0, ○ means 1, ⦿ means a value in $[0, 1]$.

Defense (in chronological order)	Defense Objective	Defense Input									Assumptions				Achieved Properties				Type of Enhanced Malware Detector		
	malware detection	A_1: Training set I_{train}	A_2: Defense tactic	A_3: Feature set	A_4: Learning algorithm	A_5: Response	A_6: Manipulation set	A_7: Manipulation space	A_8: Attack tactic	A_9: Adversarial example	IID assumption	Oracle assumption	Measurability assumption	Smoothness assumption	RR: Representation Robustness	CR: Classification Robustness	DR: Detection Robustness	TR: Training Robustness	Windows Program	Android Package	PDF
Biggio et al. [121]	✓	I_{train}	EL	S	F_θ	FQ	●	●	●	●	✓					✓					✓
Smutz and Stavrou [69]	✓	I_{train}	SE	S	F_θ	FQ	●	●	●	●	✓						✓				✓
Zhang et al. [84]	✓	I_{train}	RF	S	F_θ	FQ	○	○	●	●	✓	✓	✓	✓	✓	✓					✓
Demontis et al. [36]	✓	I_{train}	WR	S	F_θ	FQ	○	●	●	●	✓	✓	✓			✓				✓	
Grosse et al. [32]	✓	I_{train}	WR	S	F_θ	FQ	●	●	●	●	✓				✓	✓				✓	
Grosse et al. [32]	✓	I_{train}	AT	S	F_θ	FQ	○	○	○	●	✓	✓	✓		✓	✓				✓	
Tong et al. [122]	✓	I_{train}	RF	S	F_θ	FQ	○	●	●	●					✓	✓	✓				✓
Wang et al. [37]	✓	I_{train}	IT	S	F_θ	FQ	●	●	●	●						✓			✓		
Chen et al. [100]	✓	I_{train}	AT	S	F_θ	FQ	○	○	○	●	✓	✓	✓			✓			✓		
Khasawneh et al. [73]	✓	I_{train}	CD	S	F_θ	FQ	●	●	○	●	✓					✓			✓		
Yang et al. [78]	✓	$\mathcal{Z}^*_{\mathcal{M}} \cup I_{train}$	AT	S	F_θ	FQ	●	○	●	⦿	✓						✓			✓	
Yang et al. [78]	✓	I_{train}	SE	S	F_θ	FQ	○	●	○	●	✓						✓			✓	
Dang et al. [42]	✓	I_{train}	SE	S	F_θ	FQ	●	●	●	●	✓						✓				✓
Chen et al. [44]	✓	I_{train}	RF	S	F_θ	FQ	○	○	●	●	✓	✓	✓	✓	✓	✓				✓	
Incer et al. [123]	✓	I_{train}	RF	S	F_θ	FQ	○	●	●	●	✓				✓	✓	✓		✓		
Al-Dujaili et al. [33]	✓	I_{train}	AT	S	F_θ	FQ	○	○	●	●	✓	✓					✓		✓		
Chen et al. [79]	✓	I_{train}	IT	S	F_θ	FQ	●	●	●	●						✓				✓	
Jordan et al. [124]	✓	I_{train}	RF	S	F_θ	FQ	○	●	●	●					✓	✓	✓				✓
Li et al. [125]	✓	I_{train}	AT	S	F_θ	FQ	●	●	●	●	✓		✓	✓	✓	✓			✓		
Chen et al. [38]	✓	I_{train}	SE	S	F_θ	FQ	●	●	●	●			✓					✓		✓	

obfuscation attack [126] with input $(a_1, \cdots, a_5 | A_6, \cdots, A_9) = (0, 0, 0, 0, 0 | \mathcal{M}, \mathcal{Z}, \mathsf{BE}, \mathcal{Z}_\mathcal{M})$, and the attack that modifies important features [36] with input $(a_1, \cdots, a_5 | A_6, \cdots, A_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, *, 0 | \mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$. The defense works under the IID, Oracle and Measurability assumptions with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{WR}, S, F_\theta, A_5 | 1, 0, 0, 0)$ and achieves CR.

Grosse et al. [32] propose two defenses, dubbed *distillation* and *retraining*, to enhance DNN-based malware detectors. Both defenses enhance an Android malware detector against a variant of the JSMA attack with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (0, 1, 1, 1, 1 | \mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$. The distillation defense works under the IID assumption with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{WR}, S, F_\theta, \mathsf{FQ} | 0, 0, 0, 0)$ and achieves RR and CR. The retraining defense works under the IID, Oracle and Measurability assumptions with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{AT}, S, F_\theta, \mathsf{FQ} | 1, 1, 1, 0)$ and achieves RR and CR.

Tong et al. [122], [127] propose refining features into ideally invariant ones to defend against evasion attacks. The defense is validated against the genetic algorithm attack [41] with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (0, 0, 0, 0, 1 | \mathcal{M}, \mathcal{Z}, \mathsf{BE}, \mathcal{Z}_\mathcal{M})$. The defense has input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ} | 1, 0, 0, 0)$, and achieves RR, CR and DR.

Random feature nullification (RFN) [37] enhances DNN-based malware detectors against the FGSM attack [115] 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 input. The defense enhances Windows malware detectors against the FGSM attack with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (0, 1, 1, 1, 1 | \mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, *, 0 | \mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$. The defense has input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{IT}, S, F_\theta, \mathsf{FQ} | 0, 0, 0, 0)$ and achieves CR.

SecDefender [100] enhances linear malware detectors by adopting the generic retraining framework proposed in AML context [128]; the idea is to incorporate a label smoothness regularization to reduce some negative impact of adversarial training [129]. It is validated using Windows malware detectors against "feature selection"-based white-box evasion attacks with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (1, 1, 1, 1, 1 | \mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$. The defense works under the IID, Oracle and Measurability assumptions with input $(A_1, \cdots, A_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{AT}, S, F_\theta, \mathsf{FQ} | 1, 1, 1, 0)$ and achieves CR.

Khasawneh et al. [73] propose randomizing classifiers (i.e., using one randomly chosen from a pool of classifiers that use heterogeneous features) to defend against reverse engineering attacks. The defense is validated against an attack with input $(a_1, \ldots, a_5 | a_6, \cdots, a_9) = (0, 0, *, 0, 1 | \mathcal{M}, \mathcal{Z}, \mathsf{BE}, \mathcal{Z}_\mathcal{M})$. The defense works under the IID assumption with input $(a_1, \ldots, a_5 | a_6, \cdots, a_9) = (I_{train}, \mathsf{CR}, S, F_\theta, \mathsf{FQ} | 0, 0, 1, 0)$ and achieves CR.

Yang et al. [78] leverage adversarial training and variant detector and box-constraint weights to defend against genetic algorithm-based evasion attacks with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (0, 0, 0, 0, 1|\mathcal{M}, \mathcal{Z}, \mathsf{BE}, \mathcal{Z}_{\mathcal{M}})$. Adversarial training works under the IID assumption with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train} \cup \mathcal{Z}^*_{\mathcal{M}}, \mathsf{AT}, S, F_\theta, \mathsf{FQ}|0, 1, 0, *)$ and achieves DR. The variant detector works under the IID assumption with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{SE}, S, F_\theta, \mathsf{FQ}|1, 0, 1, 0)$ and achieves DR.

Dang et al. [42] propose enhancing PDF malware detectors by lowering the classification threshold τ to classify more files as malicious, rendering evasion attacks harder to succeed. This defense works under the IID assumption with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{SE}, S, F_\theta, \mathsf{FQ}|0, 0, 0, 0)$ and achieves DR.

Chen et al. [44] propose mitigating evasive attacks by filtering features according to their importance $|w_i|/c_i$ with respect to linear SVM classification function $\varphi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, where x_i, w_i and c_i are respectively the ith component of $\mathbf{x}$, $\mathbf{w}$ and the constraint on manipulation cost $\mathbf{c}$. The defense enhances Android malware detectors against an attack with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (0, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{BE}, \mathcal{X}_{\mathbf{M}})$, against a variant of the mimicry attack with input $(0, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{BE}, \mathcal{X}_{\mathbf{M}})$, and against the well-crafted attack with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BE}, \mathcal{X}_{\mathbf{M}})$. The defense works under the IID, Oracle, $\mathsf{Measurability}$ and $\mathsf{smoothness}$ assumptions with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$ and achieves RR and CR.

Incer et al. [123] propose learning a monotonic malware classifier against the evasion attack that manipulates malware files by injecting instructions. A monotonic classification function satisfies that $\varphi(\mathbf{x}) \leq \varphi(\mathbf{x}')$ when $\mathbf{x} \leq \mathbf{x}'$ [130]. Monotonic features are the ones that can be modified by insertion or removal operations, but not both. These kinds of features can be extracted from PE files (e.g., dynamic imports) and are used to train monotonic malware detectors, meaning that the attack can perturb these features by adding instructions only. This causes an attack to fail when $\varphi(\mathbf{x}) \leq \varphi(\mathbf{x}')$. The defense works under the IID assumption with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 0, 0, 0)$ and achieves RR, CR and DR.

Al-Dujaili et al. [33] adapt the idea of *minmax* adversarial training (in AML context) to enhance DNN-based malware detectors as follows: the inner-layer optimization generates adversarial files by maximizing the classifier's loss function; the outer-layer optimization searches for parameters θ of DNN F_θ that minimize the classifier's loss upon the generated adversarial files. The defense enhances Windows malware detectors against attacks with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (0, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}})$. The defense works under the IID and Oracle assumptions with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{AT}, S, F_\theta, \mathsf{FQ}|1, 1, 0, 0)$ and achieves DR.

DroidEye [79] 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 $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}})$ [100] and the FGSM attack with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (0, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_{\mathbf{M}})$ [115]. The defense has the input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{IT}, S, F_\theta, \mathsf{FQ}|0, 0, 0, 0)$ and achieves CR.

Jordan et al. [124] propose a robust PDF malware detector against evasion attacks by interpreting JavaScript behaviours using static analysis. A PDF file is classified as malicious when it calls a vulnerable API method or when it exhibits a potentially malicious or unknown behavior. The defense is validated against the *reverse mimicry* attack [97] with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (0, 0, 0, 0, 0|\mathcal{M}, \mathcal{Z}, \mathsf{BE}, \mathcal{Z}_{\mathcal{M}})$. The defense has input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{RF}, S, F_\theta, \mathsf{FQ}|1, 0, 0, 0)$ where $\theta = \emptyset$ and achieves RR, CR and DR.

Li et al. [125] propose a DNN-based attack-agnostic framework for classifying adversarial malware files. The framework wins the AICS'2019 adversarial malware classification challenge organized by MIT Lincoln Lab researcher [131], without knowing about the attack. The defense works under the IID, $\mathsf{Measurability}$ and $\mathsf{Smoothness}$ assumptions with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{AT}, S, F_\theta, \mathsf{FQ}|0, 0, 0, 0)$ and achieves RR and CR.

Chen et al. [38] investigate defending Android malware detectors against poisoning attacks with input $(a_1, \ldots, a_5|A_6, \cdots, A_9) = (1, 1, 1, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BP}, \mathcal{X}_{\mathbf{M}}) \vee (0, 0, 0, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BP}, \mathcal{X}_{\mathbf{M}}) \vee (1, 0, *, 1, 1|\mathbf{M}, \mathcal{X}, \mathsf{BP})$. 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 $\mathsf{Measurability}$ assumption with input $(A_1, \cdots, A_5|a_6, \cdots, a_9) = (I_{train}, \mathsf{SE}, S, F_\theta, \mathsf{FQ}|0, 0, 0, 0)$ and achieves TR.

Summary and Drawing Insights (via ML). Table IV succinctly summarizes the preceding defenses. From Table IV and the preceding discussion we draw:

Insight 3 (Manually manually drawn)**.** *(i) Most studies focus on black-box defenses (i.e., defender knows little about the attacker), which is against the principle of "knowing yourself and knowing your enemy". (ii) Most studies have been focusing 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) AMD security properties have been evaluated empirically rather than rigorously proven (despite that provable security in AML is emerging; see for example [132], [133], [134]). (v) Sanitizing adversarial files as outliers is effective against black-box and grey-box attacks, but not white-box attacks.*

In order to draw further insights, we train another Random Forest model from the systematized structure data presented in

Table IV. We use attributes of defense objective, defense input, and assumptions as features and treat the achieved properties as the labels to learn. The data preprocessing is the similar to insight learning from AMD attacks. This leads to a dataset of 20 defenses, each of which has 14 dimensions.

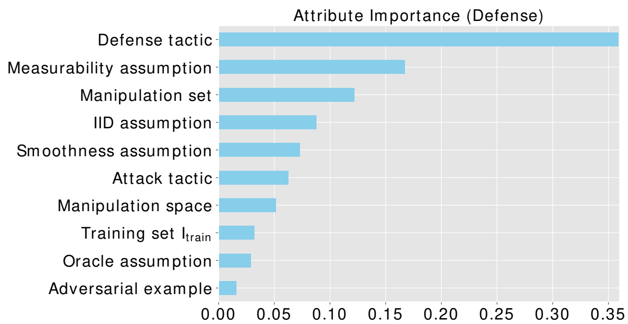

Fig. 3: Insights learning from systematized AMD defenses via attribute importance.

Figure 3 shows the most important features on AMD defenses, from which we make the following observations. (i) Defense tactic is the most important feature in determining the empirically achieved security property. This can be explained because $\mathsf{EL}, \mathsf{IT}, \mathsf{CD}, \mathsf{WR}$ enhance the classifier against small manipulations to achieve the CR property; SE usually achieves the DR property against large manipulations, which could be detected as outliers; AT, RF enhance classifiers against certain attacks specified by the defender. (ii) The $\mathsf{Measurability}$ assumption is the second important feature. (iii) Manipulation set is the third important feature because knowing this attribute enables the defender to design RF and AT tactics, which can enhance classifiers to achieve empirical RR, CR and DR properties.

Insight 4 (Insights automatically learned)**.** *Technique-wise, defense tactic largely determines what security properties can be (empirically) achieved and knowing attacker's manipulation set is critical to defender's success.*

C. Systematizing AMD Arms Race

Figure 4 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 have a common input $(a_6, \ldots, a_9)$. For example, we group [32] and [100] 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, $(a_1, \ldots, a_5) = (0, 0, 1, 0, 0)$ is the *critical information* for attack input $(a_1, \cdots, a_5|A_6, \cdots, A_9) = (0, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (0, 0, 1, 0, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M}) \vee (1, 0, 1, 0, 1|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$ 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 $(0, 0, 1, 0, 0|\mathbf{M}, \mathcal{X}, \mathsf{OE2}, \mathcal{X}_\mathbf{M})$ because it is already able to break some defense and automatically implies that a stronger input can of course achieve the same. Multiple defense inputs are simplified in the same manner.

Arms Race Surrounding PDF Malware Detector. PDFrate caused two sequences of escalations. In one sequence, PDFrate is defeated by [29], which triggers defense escalation to [69]; [69] triggers attack escalation to [97], which can evade [69]; [97] triggers defense escalation to [124], which is a state-of-the-art enhancement of PDFrate. In the other sequence, PDFrate is defeated by [41], which triggers defense escalation to [42] and [122]; the escalated defense in [42] can be defeated by the escalated attack presented in the very same paper [42]; [122] is another state-of-the-art enhancement to PDFrate.

Arms Race Surrounding Android Malware Detector. Drebin is defeated by [36], which also proposes new defense that can defeat the escalated attack; the escalated defense in [36] triggers the attack escalation to [78], which can defeat the escalated defense in [36]; the escalated attack in [78] triggers an escalated defense presented in the very same paper [78], which remains the state-of-the-art enhancement to Drebin.

Arms Race Surrounding DNN-based Malware Detector. The DNN-based detector [32] triggers an escalated attack in [33], which triggers the defense escalation in [136]; [136] is the state-of-the-art monotonic DNN-based detector.

Independent Arms Race. There are studies that have yet to trigger cascading arms races, including: (i) Studies [38], [37], [100], [39] propose independent attacks and then show how to defeat these attacks. (ii) Studies [104], [98], [99], [101], [102], [75], [103], [82] propose attacks to defeat naive malware detectors. (iii) Studies propose defenses to counter some attacks [121], [44], [79], [125].

IV. Future Research Directions (FRDs)

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 [28], [137], [70]); (ii) incompetent feature extraction [41], [36]; (iii) high dimensionality of malware representations [134]; (iv) insufficient training data [138]; (v) low-probability "pockets" in data manifolds [31]; (vi) linearity of DNNs [115]; and (vii) large curvature of decision boundaries [139], [140].

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

FRD 3: *Quantifying robustness and resilience of malware detectors*. Robustness and resilience of malware detectors against adversarial examples needs to be quantified, ideally

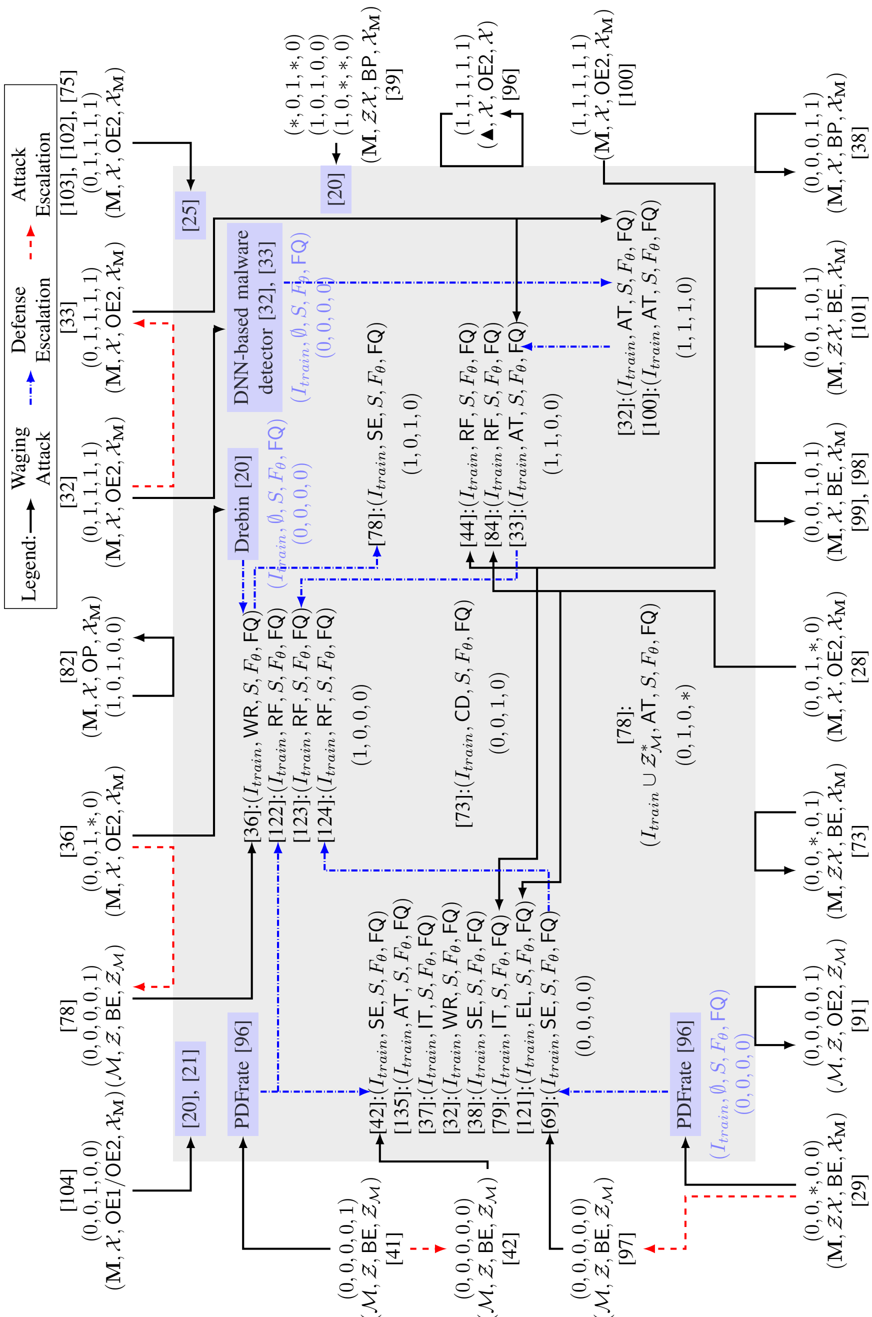

Fig. 4: Arms race in AMD attack and defense escalations.

with a provable guarantee. For this purpose, one may adapt the reduction-based paradigm underlying the provable security of cryptographic primitives and protocols.

FRD 4: *Designing malware detectors with provable robustness and resilience guarantees*. Having understood the root cause(s) of adversarial examples, characterized the effect of transferability, and designed metrics for quantifying the robustness and resilience of malware detectors, practical malware detectors with provable robustness should be designed.

FRD 5: *Forecasting the arms race in malware detection*. Arms race is a fundamental phenomenon inherent to the cybersecurity domain. In order to effectively defend against malware, one approach is to deploy proactive defense, which requires the capability in forecasting the arms race between malware writers and defenders.

V. Conclusion

We have presented a framework for systematizing the field of AMD through the lens of assumptions, attacks, defenses and security properties, which paves the way for precisely relating attacks and defenses. We have showed how to apply the framework to systematize the AMD literature, including the arms race between AMD attacks and defenses. We have reported deeper insights by using a manual effort and by applying machine learning for insights learning. We have discussed a number of future research directions.

References

[1] M. Garnaeva, F. Sinitsyn, and Y. Namestnikov, "Overall statistic for 2016," *Blue Book*, 2016.

[2] K. Lab, "Kaspersky @ONLINE," May 2018. [Online]. Available: https://www.kaspersky.com/

[3] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012.

[4] CISCO, "Cisio reporter," Avaiabled at https://www.cisco.com (2018/12/02).

[5] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007, pp. 421–430.

[6] S. Z. M. Shaid and M. A. Maarof, "In memory detection of windows api call hooking technique," in *Computer, Communications, and Control Technology (I4CT), 2015 International Conference on*. IEEE, 2015, pp. 294–298.

[7] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, Nov 2010, pp. 297–300.

[8] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "Bitblaze: A new approach to computer security via binary analysis," in *International Conference on Information Systems Security*. Springer, 2008, pp. 1–25.

[9] A. Aprville and A. Albertini, "Hide android applications in images," *Black Hat Europe*, 2014.

[10] L. Bilge and T. Dumitras, "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, 2012, pp. 833–844.

[11] J. Jung, C. Jeon, M. Wolotsky, I. Yun, and T. Kim, "AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically," in *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, Jul. 2017.

[12] Y. Ye, T. Li, D. A. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 41:1–41:40, 2017.

[13] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by association: large scale malware detection by mining file-relation graphs," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1524–1533.

[14] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1507–1515.

[15] Z. Zhu and T. Dumitraş, "Featuresmith: Automatically engineering features for malware detection by mining the security literature," in *CCS*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 767–778. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978304

[16] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 807–822.

[17] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Make evasion harder: An intelligent android malware detection system," in *Proceedings of the Twenty-Seventh IJCAI*, 2018, pp. 5279–5283.

[18] Y. Fan, S. Hou, Y. Zhang, Y. Ye, and M. Abdulhayoglu, "Gotcha - sly malware!: Scorpion A metagraph2vec based malware detection system," in *Proceedings of KDD'2018*, 2018, pp. 253–262.

[19] P. Du, Z. Sun, H. Chen, J. Cho, and S. Xu, "Statistical estimation of malware detection metrics in the absence of ground truth," *IEEE Trans. Information Forensics and Security*, vol. 13, no. 12, pp. 2965–2980, 2018.

[20] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[21] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[22] Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao, "Sbmds: an interpretable string based malware detection system using svm ensemble with bagging," *Journal in computer virology*, vol. 5, no. 4, p. 283, 2009.

[23] K. Kendall and C. McMillan, "Practical malware analysis," in *Black Hat Conference, USA*, 2007, p. 10.

[24] "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, vol. 460-461, pp. 83 – 102, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025518303475

[25] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] R. Vinayakumar and K. Soman, "Deepmalnet: Evaluating shallow and deep networks for static pe malware detection," *ICT Express*, 2018.

[27] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE, 2015, pp. 11–20.

[28] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases: European Conference*. Springer, 01 2013, pp. 387–402.

[29] N. Šrndić and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 197–211.

[30] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

[31] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[32] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 62–79.

[33] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. OReilly, "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 76–82.

[34] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *EISIC'2017*, 2017, pp. 99–106.

[35] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519.

[36] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[37] Q. Wang, W. Guo, K. Zhang, and et al., "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd KDD*. ACM, 2017, pp. 1145–1153.

[38] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers & security*, vol. 73, pp. 326–344, 2018.

[39] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1299–1316.

[40] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "On the intriguing connections of regularization, input gradients and transferability of evasion and poisoning attacks," *arXiv preprint arXiv:1809.02861*, 2018.

[41] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers: A case study on pdf malware classifiers," in *NDSS*, January 2016.

[42] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *CCS*. ACM, 2017, pp. 119–133.

[43] L. Xu, Z. Zhan, S. Xu, and K. Ye, "An evasion and counter-evasion study in malicious websites detection," in *CNS, 2014 IEEE Conference on*. IEEE, 2014, pp. 265–273.

[44] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *ACSAC*. USA: ACM, 2017, pp. 362–372.

[45] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[46] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *CoRR*, vol. abs/1810.00069, 2018. [Online]. Available: http://arxiv.org/abs/1810.00069

[47] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.

[48] Microsoft, "Pe format," 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/debug/pe-format

[49] D. Maiorca, B. Biggio, and G. Giacinto, "Towards adversarial malware detection: Lessons learned from pdf-based attacks," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 78:1–78:36, Aug. 2019. [Online]. Available: http://doi.acm.org/10.1145/3332184

[50] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 16–25.

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

[52] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. Leung, "A survey on security threats and defensive techniques of machine learning: a data driven view," *IEEE access*, vol. 6, pp. 12 103–12 117, 2018.

[53] B. Biggio, I. Corona, B. Nelson, B. I. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli, "Security evaluation of support vector machines in adversarial environments," in *Support Vector Machines Applications*. Springer, 2014, pp. 105–153.

[54] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[55] A. C. Serban and E. Poll, "Adversarial examples-a complete characterisation of the phenomenon," *arXiv preprint arXiv:1810.01185*, 2018.

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

[57] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "N-gram-based detection of new malicious code," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, vol. 2. IEEE, 2004, pp. 41–42.

[58] N. Šrndić and P. Laskov, "Detection of malicious pdf files based on hierarchical document structure," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, 2013, pp. 1–16.

[59] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 3838–3845.

[60] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl4md: A deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer , 2016, p. 61.

[61] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE T-PAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[62] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[63] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, Nov 2014.

[64] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804 – 811, 2015.

[65] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012, pp. 3–14.

[66] H. Dornhackl, K. Kadletz, R. Luh, and P. Tavolato, "Malicious behavior patterns," in *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*. IEEE, 2014, pp. 384–389.

[67] V. Vapnik, "Principles of risk minimization for learning theory," in *Advances in neural information processing systems*, 1992, pp. 831–838.

[68] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[69] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors." in *NDSS*, 2016.

[70] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. D. McDaniel, "On the (statistical) detection of adversarial examples," *arxiv preprint*, vol. abs/1702.06280, 2017.

[71] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[72] B. Wang, J. Gao, and Y. Qi, "A theoretical framework for robustness of (deep) classifiers against adversarial examples," *arXiv preprint arXiv:1612.00334*, 2016.

[73] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Rhmd: evasion-resilient hardware malware detectors," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 315–327.

[74] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," in *CoRR*, 2018.

[75] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*, Sep. 2018, pp. 533–537.

[76] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against rnns and other calls based malware classifiers," *arXiv preprint*, 2017.

[77] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," *Black Hat*, 2017.

[78] W. Yang, D. Kong, T. Xie, and C. A. Gunter, "Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps," in *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 288–302.

[79] L. Chen, S. Hou, Y. Ye, and S. Xu, "Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks," in *FOSINT-SI'2018*, 2018, pp. 253–262.

[80] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[81] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

[82] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 27–38.

[83] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

[84] F. Zhang, P. P. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE transactions on cybernetics*, vol. 46, no. 3, pp. 766–777, 2016.

[85] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 3–14.

[86] A. Narayanan, L. Yang, L. Chen, and L. Jinliang, "Adaptive and scalable android malware detection through online learning," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 2484–2491.

[87] J. D. Tygar, "Adversarial machine learning," *IEEE Internet Computing*, vol. 15, no. 5, pp. 4–6, 2011.

[88] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[89] F. Tramèr and D. Boneh, "Adversarial training and robustness for multiple perturbations," *arXiv preprint arXiv:1904.13000*, 2019.

[90] P. OKane, S. Sezer, and K. McLaughlin, "Obfuscation: The hidden malware," *IEEE Security & Privacy*, vol. 9, no. 5, pp. 41–47, 2011.

[91] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, “Learning to evade static pe machine learning malware models via reinforcement learning,” *arXiv preprint arXiv:1801.08917*, 2018.
[92] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” pp. 39–57, 2017.
[93] A. Paudice, L. Muoz-Gonzlez, and E. C. Lupu, “Label sanitization against label flipping poisoning attacks,” 2018.
[94] J. Steinhardt, P. W. W. Koh, and P. S. Liang, “Certified defenses for data poisoning attacks,” in *Advances in Neural Information Processing Systems*, 2017, pp. 3517–3529.
[95] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, “Measuring neural net robustness with constraints,” in *Advances in neural information processing systems*, 2016, pp. 2613–2621.
[96] C. Smutz and A. Stavrou, “Malicious pdf detection using metadata and structural features,” in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC ’12. New York, NY, USA: ACM, 2012, pp. 239–248. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420987
[97] D. Maiorca, I. Corona, and G. Giacinto, “Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious pdf files detection,” in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ser. ASIA CCS ’13. New York, NY, USA: ACM, 2013, pp. 119–130.
[98] W. Hu and Y. Tan, “Generating adversarial malware examples for black-box attacks based on gan,” 02 2017.
[99] ——, “Black-box attacks against rnn based malware detection algorithms,” *arXiv preprint arXiv:1705.08131*, 2017.
[100] L. Chen, Y. Ye, and T. Bourlai, “Adversarial machine learning in malware detection: Arms race between evasion attack and defense,” in *Intelligence and Security Informatics Conference (EISIC), 2017 European*. IEEE, 2017, pp. 99–106.
[101] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, “Generic black-box end-to-end attack against rnns and other api calls based malware classifiers,” *arXiv preprint arXiv:1707.05970*, 2017.
[102] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, “Deceiving end-to-end deep learning malware detectors using adversarial examples,” *arXiv preprint arXiv:1802.04528*, 2018.
[103] O. Suciu, S. E. Coull, and J. Johns, “Exploring adversarial examples in malware detection,” in *2019 IEEE Security and Privacy Workshops (SPW)*, May 2019, pp. 8–14.
[104] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, “Android hiv: A study of repackaging malware for evading machine-learning detection,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 987–1001, 2020.
[105] P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli, “Secure kernel machines against evasion attacks,” in *Proceedings of the 2016 ACM workshop on artificial intelligence and security*. ACM, 2016, pp. 59–69.
[106] D. Maiorca, B. Biggio, M. E. Chiappe, and G. Giacinto, “Adversarial detection of flash malware: Limitations and open issues,” *arXiv preprint arXiv:1710.10225*, 2017.
[107] N. Šrndic and P. Laskov, “Detection of malicious pdf files based on hierarchical document structure,” in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*. Citeseer, 2013, pp. 1–16.
[108] N. Šrndić and P. Laskov, “Hidost: a static machine-learning-based detector of malicious files,” *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 22, 2016.
[109] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
[110] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
[111] H. Peng, F. Long, and C. Ding, “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.
[112] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, “Hardware-based malware detection using low-level architectural features,” *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3332–3344, 2016.
[113] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against deep learning systems using adversarial examples,” *arXiv preprint*, 2016.
[114] I. P. Gent and T. Walsh, “Towards an understanding of hill-climbing procedures for sat,” in *AAAI*, vol. 93. Citeseer, 1993, pp. 28–33.
[115] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
[116] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *arXiv preprint arXiv:1706.06083*, 2017.
[117] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” *arXiv preprint arXiv:1611.01236*, 2016.
[118] N. Papernot, P. McDaniel, S. Jha, and et al., “The limitations of deep learning in adversarial settings,” in *EuroS&P, 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.
[119] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, “Is feature selection secure against training data poisoning?” in *International Conference on Machine Learning*, 2015, pp. 1689–1698.
[120] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[121] B. Biggio, I. Corona, Z.-M. He, P. P. Chan, G. Giacinto, D. S. Yeung, and F. Roli, “One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time,” in *International Workshop on Multiple Classifier Systems*. Springer, 2015, pp. 168–180.
[122] L. Tong, B. Li, C. Hajaj, C. Xiao, and Y. Vorobeychik, “A framework for validating models of evasion attacks on machine learning, with application to pdf malware detection,” *arXiv preprint arXiv:1708.08327*, 2017.
[123] I. Incer, M. Theodorides, S. Afroz, and D. Wagner, “Adversarially robust malware detection using monotonic classification,” in *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*. ACM, 2018, pp. 54–63.
[124] A. Jordan, F. Gauthier, B. Hassanshahi, and D. Zhao, “Safe-pdf: Robust detection of javascript pdf malware using abstract interpretation,” *arXiv preprint arXiv:1810.12490*, 2018.
[125] D. Li, Q. Li, Y. Ye, and S. Xu, “Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and application to aics2019 challenge,” in *The AAAI-19 Workshop on Artificial Intelligence for Cyber Security (AICS), 2019*, 2019.
[126] G. nv., “Dexgurad,” Avaiabled at https://www.guardsquare.com (2018/12/02).
[127] L. Tong, B. Li, C. Hajaj, C. Xiao, and Y. Vorobeychik, “Hardening classifiers against evasion: the good, the bad, and the ugly,” *CoRR, abs/1708.08327*, 2017.
[128] B. Li, Y. Vorobeychik, and X. Chen, “A general retraining framework for scalable adversarial classification,” *arXiv preprint arXiv:1604.02606*, 2016.
[129] P. Yang and P. Zhao, “A min-max optimization framework for online graph classification,” in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 643–652.
[130] M. Gupta, A. Cotter, J. Pfeifer, K. Voevodski, K. Canini, A. Mangylov, W. Moczydlowski, and A. Van Esbroeck, “Monotonic calibrated interpolated look-up tables,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3790–3836, 2016.
[131] A. workshop, “Aics 2019 workshop challenge problem @ONLINE,” 2018. [Online]. Available: http://www-personal.umich.edu/~arunesh/AICS2019/challenge.html
[132] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, “Ai 2: Safety and robustness certification of neural networks with abstract interpretation,” in *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.
[133] A. Sinha, H. Namkoong, and J. Duchi, “Certifying some distributional robustness with principled adversarial training,” 2018.
[134] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, “Adversarial spheres,” *arXiv preprint arXiv:1801.02774*, 2018.

[135] D. Li, Q. Li, Y. Ye, and S. Xu, “Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics’2019 challenge,” *arXiv preprint arXiv:1812.08108*, 2018.
[136] A. . Workshop, “Robust malware detection challenge @ONLINE,” 2019. [Online]. Available: https://sites.google.com/view/advml/advml19-challenge
[137] B. Biggio, G. Fumera, and F. Roli, “Security evaluation of pattern classifiers under attack,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, April 2014.
[138] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry, “Adversarially robust generalization requires more data,” in *Advances in Neural Information Processing Systems*, 2018, pp. 5019–5031.
[139] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, “Robustness of classifiers: from adversarial to random noise,” in *Advances in Neural Information Processing Systems*, 2016, pp. 1632–1640.
[140] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, and S. Soatto, “Robustness of classifiers to universal perturbations: A geometric perspective,” in *6th International Conference on Learning Representations*, 2018.
[141] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[142] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri, “Are loss functions all the same?” *Neural Computation*, vol. 16, no. 5, pp. 1063–1076, 2004.
[143] Y. Kim, “Convolutional neural networks for sentence classification,” *arXiv preprint arXiv:1408.5882*, 2014.
[144] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” *Journal of machine learning research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

Appendix

A. Illustrating the Partial Order Formulated by Black-box vs. Grey-box vs. White-box Attacks

Figure 5 illustrates how vector $(a_1, \cdots, a_5)$ 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 grey-box attacks in between. This structure pays the way for unambiguously comparing attacks.

B. Illustrating the Partial Order Formulated by Black-box vs. Grey-box vs. White-box Defenses

Figure 6 illustrates 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 $(a_6, a_7, a_8, a_9) = (1, 1, 1, 1, 1)$; there are many kinds of grey-box defenses in between. This structure pays the way for unambiguously comparing defenses.

C. The Drebin Android Malware Detector

Drebin is an Android malware detector learned from static features [20]. Table V summarizes the Drebin feature set S, which consists of 8 feature subsets, including 4 subsets S_1, S_2, S_3, S_4 of features extracted from *AndroidManifest.xml* and 4 subsets S_5, S_6, S_7, S_8 of features extracted from the disassembled dexcode. Specifically, S_1 contains features related to the access of an Android package (APK) to smartphone devices (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 etc.); S_4 contains features related to APK’s communications with the operating system; 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 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 code.

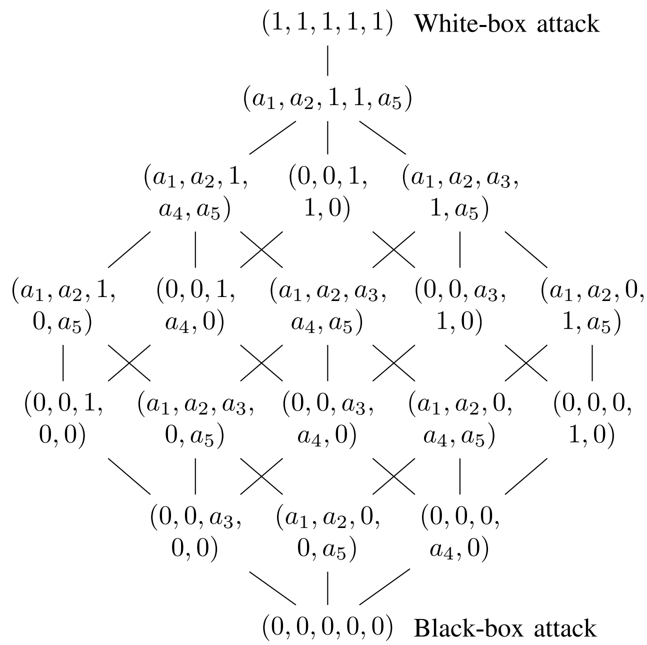

Fig. 5: A portion of the partial order defined over $(a_1, \ldots, a_5)$.

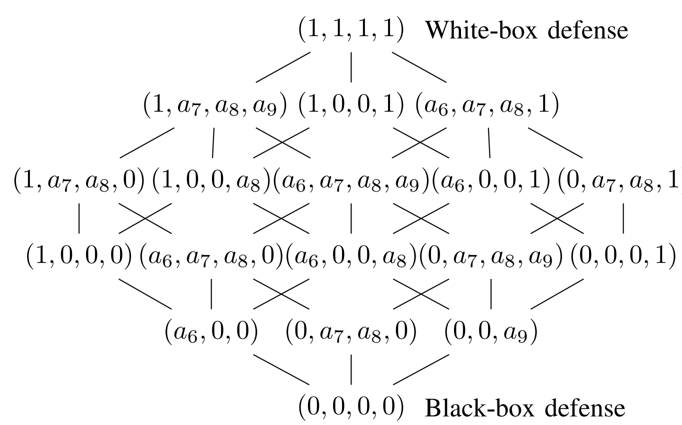

Fig. 6: A portion of the partial order defined over $(a_6, \ldots, a_9)$.

TABLE V: 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

The feature representation is binary, meaning $\phi : \mathcal{Z} \mapsto \{0,1\}^d$ 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:

$$\mathbf{x} = \phi(z) \mapsto \begin{pmatrix} \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \\ 0 \\ \cdots \end{pmatrix} \begin{array}{ll} \cdots & \\ \left.\begin{array}{l} \texttt{permission::SEND\_SMS} \\ \texttt{permission::READ\_CONTACTS} \end{array}\right\} & S_2 \\ \cdots & \\ \left.\begin{array}{l} \texttt{api\_call::getDeviceID} \\ \texttt{api\_call::setWifiEnabled} \end{array}\right\} & S_5 \\ \cdots & \end{array}$$

Drebin uses a linear SVM [141] to learn a classification. The linear discriminant function is generally expressed as $F = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w}$ is the weight vector and b is the bias. It uses the hinge loss function to tune parameters $\mathbf{w}$ and b [142].

D. The MalConv Windows Malware Detector

MalConv [25] is CNN-based Windows malware detector learned from raw binary programs (i.e., end-to-end) [143]. Figure 7 exhibits its architecture. At the input layer, the sequence of binary code is transformed into byte value (between 0 to 255). The length of the input is bounded by m (e.g., $m = 2^{21}$ bytes or 2MB). Each byte is further mapped into a real-valued vector using embedding [144]. The CNN layer and pooling layer aim to learn high-level representations.

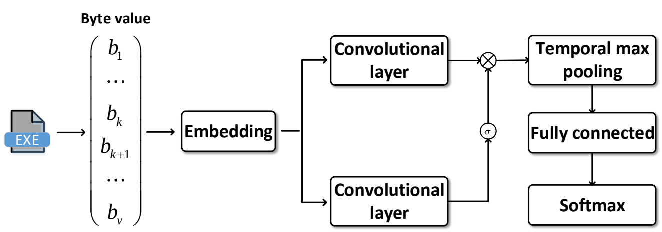

Fig. 7: MalConv architecture [25].