

On the Security Risks of ML-based Malware Detection Systems: A Survey

PING HE, Zhejiang University, China and UCL, UK

YUHAO MAO, ETH Zürich, Switzerland

CHANGJIANG LI, Stony Brook University, USA

LORENZO CAVALLARO, UCL, UK

TING WANG, Stony Brook University, USA

SHOULING JI, Zhejiang University, China

Malware presents a persistent threat to user privacy and data integrity. To combat this, machine learning-based (ML-based) malware detection (MD) systems have been developed. However, these systems have increasingly been attacked in recent years, undermining their effectiveness in practice. While the security risks associated with ML-based MD systems have garnered considerable attention, the majority of prior works is limited to adversarial malware examples, lacking a comprehensive analysis of practical security risks. This paper addresses this gap by utilizing the CIA principles to define the scope of security risks. We then deconstruct ML-based MD systems into distinct operational stages, thus developing a stage-based taxonomy. Utilizing this taxonomy, we summarize the technical progress and discuss the gaps in the attack and defense proposals related to the ML-based MD systems within each stage. Subsequently, we conduct two case studies, using both inter-stage and intra-stage analyses according to the stage-based taxonomy to provide new empirical insights. Based on these analyses and insights, we suggest potential future directions from both inter-stage and intra-stage perspectives.

CCS Concepts: • **Security and privacy** → **Software and application security**; • **Computing methodologies** → **Machine learning**.

ACM Reference Format:

Ping He, Yuhao Mao, Changjiang Li, Lorenzo Cavallaro, Ting Wang, and Shouling Ji. 2025. On the Security Risks of ML-based Malware Detection Systems: A Survey. 1, 1 (May 2025), 35 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 Introduction

Since the emergence of the Creeper Virus in 1971 [188], malware has posed a persistent threat to computational devices. With over 1 billion identified malware samples [14], user privacy and data integrity are at significant risk. In response, machine learning-based (ML-based) malware detection (MD) systems have been proposed and widely deployed in computational devices [85]. These systems have gained popularity due to their effective use of powerful ML algorithms to efficiently detect and classify potential threats [15, 16].

Authors' Contact Information: Ping He, gnip@zju.edu.cn, Zhejiang University, Hangzhou, China and UCL, London, UK; Yuhao Mao, yuhao.mao@inf.ethz.ch, ETH Zürich, Zürich, Switzerland; Changjiang Li, meet.cjli@gmail.com, Stony Brook University, New York, USA; Lorenzo Cavallaro, l.cavallaro@ucl.ac.uk, UCL, London, UK; Ting Wang, inbox.ting@gmail.com, Stony Brook University, New York, USA; Shouling Ji, sji@zju.edu.cn, Zhejiang University, Hangzhou, China.

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

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

Manuscript submitted to ACM

1

In recent years, many attacks (e.g., StingRay [169], HRAT [214], JP [202], BagAmmo [100], and AdvDroidZero [76]) have targeted ML-based MD systems, compromising their ability to detect and protect against malware. These attacks exploit vulnerabilities in program analysis and ML models. To counter these emerging threats, various defenses [19, 49, 102, 113] have been proposed, enabling these systems to withstand security risks posed by dynamic environments and adversarial attackers.

Despite the growing interest in the security of ML-based MD systems, a comprehensive and systematic understanding of this field is still absent, hindering its further development. Previous studies [12, 142, 156] have identified common pitfalls in developing such systems but have overlooked security threats from adversarial attackers, such as dataset poisoning risks. In addition, recent surveys [98, 107, 120, 200] primarily focus on adversarial malware examples during system deployment, neglecting the complexities throughout the lifetime of an ML-based MD system. In practice, it is crucial to recognize that security risks can emerge at any stage of the system’s lifecycle, necessitating a comprehensive analysis framework to effectively assess and mitigate these risks.

To address the security risks in ML-based MD systems, we propose a comprehensive stage-based taxonomy that considers the entire lifecycle of these systems, dividing attack and defense proposals based on the four operational stages of the MD system. Through this taxonomy, we systematically examine the current proposals, identify research gaps, and enable system developers to understand and mitigate security risks at each stage. Our analysis offers a holistic perspective on attacks and defenses in ML-based MD systems, facilitating the development of more robust and secure systems.

We present our findings derived from the analysis of the existing attack and defense proposals in two directions: high-level findings (Section 3.5) and low-level findings for each stage discussed as the open problems (Section 4-7). High-level findings (in terms of CIA) include: 1) although the security risks at each stage focus on different components of the ML-based MD system, they frequently target the integrity of the system, i.e., escaping detection; 2) confidentiality, i.e., stealing/protecting the copyrighted models and data, is systematically underrepresented in the existing proposals. Low-level findings, to name a few, include: 1) strong knowledge assumptions and limited adaptability hinder the practicality of attacks in the malware domain, especially for backdoor attacks; 2) deobfuscation, as a popular defense, has limited practicality due to the necessary software or hardware requirements; 3) certified defense, which guarantees non-existence of certain attacks, is emerging but still exploratory in malware detection; 4) current adversarial query-based attacks merely insert dummy benign features into the malware and keep the malicious features intact, while the latter could be designed to ease the attack; 5) recent adaptive attacks have broken robust function call features which are the underlying assumption of many state-of-the-art (SOTA) defense proposals.

Further, we conduct two case studies focusing on inter-stage and intra-stage analyses to provide empirical insights (Section 8). Specifically, we find that poisoned models in the malware domain are not necessarily more vulnerable to adversarial attacks, whereas the opposite is true in the image domain [139]. This discrepancy is likely due to problem space constraints and the discrete nature of feature spaces in malware detection. Additionally, concept drift detection methods in the malware domain exhibit a high true positive rate in identifying adversarial malware, likely due to the distribution shifts caused by adversarial perturbations. We leave further exploration of these directions and findings to future work. Finally, based on our taxonomy and analysis, we propose promising directions for future inter-stage and intra-stage research.

The main contributions of this work are summarized as follows. We first establish the scope of the security risks by integrating the CIA principles with elements of ML-based MD systems. Subsequently, we dissect the pipeline of the ML-based MD system into operational stages and propose a stage-based framework to holistically examine their

security risks. Based on the proposed framework, we systematize the technical progress and discuss the limitations of existing attack and defense proposals in ML-based MD systems at each stage. Finally, we conduct two case studies from the perspective of both inter-stage and intra-stage analyses, resulting in novel insights. Our analyses provide a comprehensive understanding of the security risks in ML-based MD systems and offer promising directions for future research.

2 Background

2.1 Components of ML-based MD Systems

MD systems are designed to detect software files exhibiting malicious behaviors, such as trojans, spyware, and ransomware. Over the past two decades, the landscape of MD has seen considerable advances. Traditionally, MD systems primarily rely on signature-based approaches, utilizing patterns and heuristics to identify malware. These approaches necessitate a regularly updated malware signature database [40]. However, such traditional systems are poorly generalized, especially in balancing the precision and recall of malware signatures. To solve these challenges, SOTA MD systems employ ML techniques to enhance malware detection [64, 112, 162]. Consequently, this research primarily focuses on ML-based MD systems.

Terminology. In this paper, we represent an ML-based MD system as $\mathbb{F} = \{\mathbf{D}, \mathcal{F}, C\}$, with the goal of classifying an unknown software sample u as either benign or malicious (i.e., detection result o). The components are as follows:

- *File dataset (D)*: The dataset used for training and evaluating ML-based MD systems. It is derived from a raw file database ($\hat{\mathbf{D}}$) through a data preprocessing function (\mathcal{P}), which selects the appropriate ratio and quantity of benign samples (x_b) and malicious samples (x_m).
- *File feature extractor (F)*: A mapping that converts file samples into feature vectors, representing the discernible characteristics of the file samples. Typically, three types of feature extraction methods are used: static analysis, dynamic analysis, and hybrid analysis.
- *ML model (C)*: A classifier that categorizes the features of a file, determining whether it is benign or malicious.

We remark that the terminology describing ML-based MD systems can vary across the literature. The terms used in this paper are selected to facilitate a comprehensive analysis of the security risks associated with ML-based MD systems.

2.2 The Pipeline of ML-based MD Systems

In this section, we divide the pipeline of ML-based MD systems into four operational stages from a practical lifecycle perspective. These stages encompass both the development and deployment of ML-based MD systems, offering a comprehensive framework for analyzing security risks. Figure 1 illustrates the four operational stages of ML-based MD systems, denoted as $\mathbb{F} = \{\mathbf{D}, \mathcal{F}, C\}$.

Stage 1: building a software file dataset \mathbf{D} through \mathcal{P} . In contrast to benchmark datasets for the image and text domains, such as MNIST [92], ImageNet [51], IMDB Reviews [117], and Yelp Reviews [211], creating a software benchmark dataset presents unique challenges. Software data is highly dynamic due to frequent updates in programming languages, build tools, and associated libraries [142, 147]. Additionally, the computational cost of constructing a software file dataset is significantly higher than that for image or text datasets, as software files are often larger and require more computational and storage resources. Furthermore, labeling software data is more costly, as it requires specialized

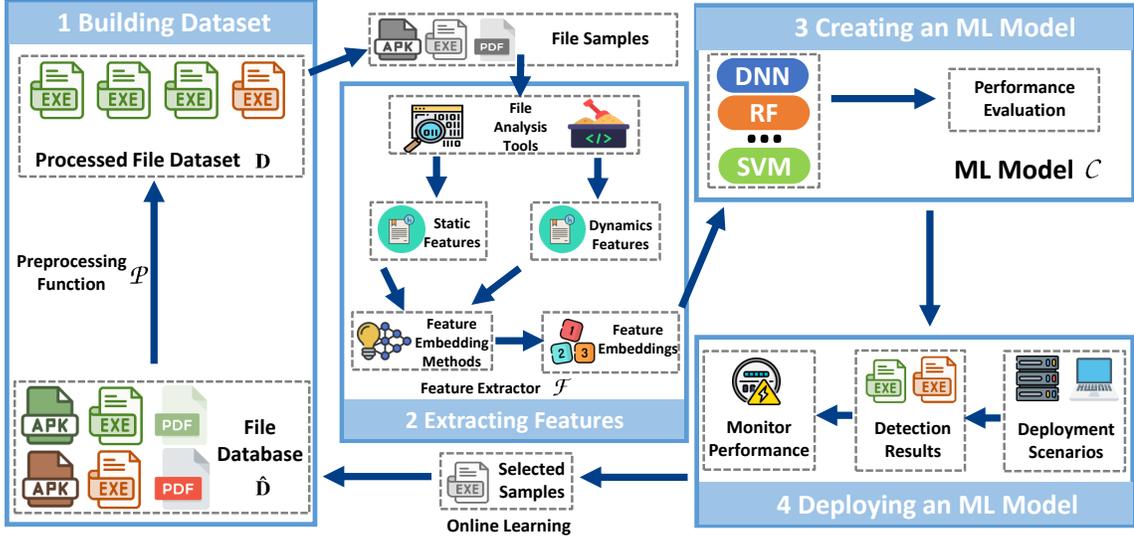


Fig. 1. The pipeline of ML-based MD systems. It has four operational stages: (1) building a software file dataset, (2) extracting the software features, (3) creating an ML model, and (4) deploying an ML model.

domain knowledge[10]. As a result, well-labeled software datasets are rare and valuable, thus usually regarded as proprietary [107].

The raw data used to construct the dataset primarily originates from online open-source software file repositories, such as VirusTotal [178], VirusShare [177], and Google Play [69], or is privately collected by security products [161]. We summarize the existing open-source datasets, as presented in Table 1. Despite their utility, these datasets often contain extensive software files that exceed the processing capabilities of contemporary computing systems. Moreover, the accessibility of software files from these sources tend to be limited and imbalanced, e.g., the BODMAS dataset [203] exclusively offers malicious samples for analysis. Besides, recent studies [12, 142] have emphasized the importance of considering the practical constraints when creating software datasets for training and evaluation, including temporal consistency and space constraints, etc. In this regard, an ML-based MD system \mathbb{F} utilizes a preprocessing function \mathcal{P} to strategically select the samples suitable for training and evaluation purposes.

Stage 2: extracting the software embedding features using \mathcal{F} . The software data in stage 1 is inherently file-based, characterized by complex structures and low-level semantics, presenting substantial challenges in training the ML model C . To enhance model training effectiveness, an ML-based MD system \mathbb{F} incorporates a dedicated software feature extractor \mathcal{F} , which utilizes file analysis tools and/or feature embedding models to derive numerical embedding features from the software file data.

Static analysis methods are predominantly utilized in the feature extraction process due to their comparative efficiency over dynamic and hybrid analysis methods [44, 112]. Static analysis examines the software file’s structure and code without executing the file, providing a safer and quicker analysis. In contrast, dynamic analysis methods execute the software sample to monitor its runtime behavior, identifying suspicious function calls and cryptographic operations. Hybrid analysis combines static and dynamic approaches, offering a more comprehensive but more costly embedding [112].

Table 1. Open-source Malware Datasets. “○, ◐, ●” = only features are available, only malware files are available, all files are available.

Dataset	Type	Size	Time	Maintain	Availability
VirusTotal [178]	Hybrid	3B+	✓	✓	◐
VirusShare [177]	Hybrid	41M+	✓	✓	◐
AndroZoo [6]	Andorid	24M+	✓	✓	●
Drebin [13]	Andorid	5,560	✗	✗	○
MalRadar [182]	Andorid	4,534	✓	✗	◐
MARVIN [105]	Andorid	150,000	✗	✗	●
BODMAS [203]	WinPE	134,435	✓	✗	◐
Ember [8]	WinPE	1.1M	✓	✗	○
SOREL20M [75]	WinPE	20M	✓	✗	○
Microsoft [153]	WinPE	10,868	✗	✗	●

The extraction process typically extracts features from three categories of software information: syntax information (e.g., permissions, strings, and file status), semantic information (e.g., function calls, data flow, and control flow), and raw bytes information (e.g., gray-scale images). These different types of information are extracted using various program analysis techniques. For instance, function calls can be identified through static analysis but may also be detected via dynamic analysis.

Stage 3: creating the ML model C . The features extracted in stage 2 may have diverse types of software information. These features, due to their varied nature, are not tailored for a specific ML model C . For instance, syntax features, typically represented by binary values, are well-suited for an SVM model [13]. Conversely, features based on function call graphs inherently possess graph structures and thus align better with graph-based models such as graph neural networks [106]. Therefore, an ML-based MD system \mathbb{F} may employ ensemble models to integrate heterogeneous features, with each child model processing a specific kind of feature [87]. In cases where \mathbb{F} is designed with additional properties, more advanced ML model structures and training algorithms are employed. For example, [191] applied the attention mechanism to encode interpretability, and [55] applied federated learning to improve the effectiveness and preserve data privacy.

Stage 4: deploying the ML model C at run-time. At run-time, an ML-based MD system \mathbb{F} extracts features from the unknown software sample using \mathcal{F} and applies the trained ML model C to classify the software sample. Once identified and confirmed as malware, the sample can be incorporated into the file database for subsequent analysis, such as exploring its relationship with other samples [179].

In practice, the ML model C may have diverse deployment scenarios. It can be deployed on user devices or centralized servers, representing on-device and off-device deployment cases, respectively. In the on-device deployment case [56], computational demands and user privacy concerns are critical considerations. Conversely, off-device deployment typically benefits from extensive computational resources and time budgets, allowing ML-based MD systems to utilize more complex ML models. Additionally, ML-based MD systems may employ a hybrid deployment strategy, operating as a distributed system consisting of both user-end devices and centralized servers. In this case, light computational tasks are executed locally, while intensive computations are transferred to remote servers.

Furthermore, the detection performance of an ML-based MD system is continuously monitored during deployment. As malware evolves and adversaries develop new variants capable of bypassing existing detection mechanisms, newly identified malware samples are selectively incorporated into the malware database to enhance detection capabilities.

Table 2. Representative ML-based MD systems according to the four stages.

Proposals	Stage 1	Stage 2		Stage 3	Stage 4	
	Dataset Preprocessing	File Analysis	Feature Encoding	ML Model	Detection Task	Deploy Environment
Drebin [13]	None	Static	Categorical	SVM	Malware Detection	Endpoint
Hidost [165]	None	Static	Tree	Decision Tree/SVM	Malware Detection	Endpoint
MaMadroid [126]	None	Static	Numerical	Random Forest/kNN	Malware Detection	Server
MalwareExpert [36]	Time-aware	Static	Graph	GNN	Malware Detection	Server
MSDroid [77]	None	Static	Graph	GNN	Malware Detection	Server
AndroAnalyzer [66]	None	Static	Graph	GNN	Family Attribution	Server
MalConv [147]	None	Static	Bytes	CNN	Malware Detection	Server
Visual-AT [110]	None	Static	Image	CNN	Malware Detection	Server
IMCEC [176]	None	Static	Image	CNN	Family Attribution	Server
LensDroid [129]	None	Static	Hybrid	Hybrid Model	Malware Detection	Server
FEDROID [55]	Random	Static	Categorical	ResNet	Malware Detection	Endpoint
Neurlux [83]	Random	Dynamic	Behavior (Text)	CNN/LSTM	Malware Detection	Server
Rabadi <i>et al.</i> [146]	None	Dynamic	API Hash	Ensemble	Family Attribution	Server
APIChecker [67]	None	Dynamic	Categorical (API)	Nine ML Models	Malware Detection	Server
XTrace+ [219]	Random	Dynamic	Behavior (Text)	Transformer	Malware Detection	Server
Chaulagain <i>et al.</i> [27]	None	Hybrid	Function Calls (Text)	LSTM	Malware Detection	Server

This process can be achieved through periodic retraining of the machine learning model or by employing online learning techniques, allowing the model to adapt to emerging threats in real-time.

Different ML-based MD systems adopt different methods for detecting different malware formats. We summarize the existing ML-based MD systems in Table 2 according to the methods used in four stages.

Table 3. CIA risks for each component of ML-based MD systems.

	D	\mathcal{F}	C	o
C	dataset leaking	extraction function leaking	model leaking	/
I	dataset poisoning	feature obfuscation	model hijacking	high false negative
A	dataset corruption	analysis blocking	/	high false positive

3 Stage-Based Taxonomy

In this section, we first present the motivation behind the stage-based taxonomy. Next, we establish the scope of security risks, utilizing the goals in the threat model. Subsequently, we detail the strategies for each attack and defense proposal at each stage. Lastly, an overview of the current field is provided according to our stage-based taxonomy.

3.1 Motivation

In practice, the adversary often has multifaceted objectives encompassing all aspects of the CIA principles. Additionally, a single system may be targeted by multiple attackers concurrently. This complexity implies that the attacks against ML-based MD systems can occur at any stage in their operational pipeline and may involve various attacks simultaneously. Conversely, defenders aim to secure the system comprehensively against all attacks at all stages. Nevertheless, current surveys [107, 200] in this field predominantly utilize taxonomy based on threat models, which fall short in capturing these practical complexities. More discussion of the related surveys can be found in Section 10.

Recognizing the necessity for a more comprehensive understanding, we propose a holistic stage-based taxonomy utilizing the pipeline of ML-based MD systems. This stage-based taxonomy is from the perspective of the ML-based MD system’s developer, who aims to understand the security risks. To reduce ambiguity, we identify the stage by the earliest time an attack influences the system, as developers prefer to eliminate risks at their entry point. Although some attacks span multiple stages to take effect, our taxonomy decouples security risks for the developer, because each attack has an initial contaminated stage. For example, while dataset poisoning spans all stages to finally affect the detection results, we identify it as a Stage 1 attack, as it initially contaminates the dataset. Hence, this stage-based taxonomy allows for effective assessment of potential security risks at each stage and investigation of interconnections between different stages in terms of attacks and defenses.

3.2 Security Risk Scopes

We establish the scope of the security risks in ML-based MD systems by referencing the goals within the threat model. These goals pertain directly to the triad of information security principles: **confidentiality**, **integrity**, and **availability**, collectively known as the CIA principles. As discussed in Section 2.1, an ML-based MD system comprises three main elements: the file dataset D , the file feature extractor \mathcal{F} , and the ML model C , along with the detection results o . Thus, the adversary’s goals are undermining these principles within the three components and the detection results, while the defender’s goals are to uphold them. For instance, during deployment (Stage 4), the confidentiality of the ML model can be compromised by model extraction attacks, and the integrity of the detection results can be compromised by adversarial attacks. Correspondingly, defenders can preserve confidentiality by employing model obfuscation techniques and maintain integrity by improving the robustness of ML-based MD systems. We define the concrete goals of CIA principles at each stage in Section 3.4 and further categorize all current attack and defense proposals according to these goals in Section 3.5.

Table 4. Taxonomy of representative attack and defense proposals. “Know. Assu.” = Knowledge Assumption. “PK, LK, ZK” = Perfect Knowledge, Limited Knowledge, Zero Knowledge. “Con., Int., Ava.” = Confidentiality, Integrity, Availability. “ , ” = Attack, Defense. “ ,  , ” = Android, Windows PE, PDF.

Stage	Proposal	Year	Proposal Properties								
			Know. Assu.	CIA Principles				Open Source	Real World	Proposal Type	Malware Format
				D	\mathcal{F}	C	o				
I	Biggio <i>et al.</i> [21]	2014	PK	Int.	—	—	Ava.				  
	StingRay [169]	2018	LK	Int.	—	—	Int.				
	Severi <i>et al.</i> [159]	2021	PK	Int.	—	—	Int.				  
	Li <i>et al.</i> [93]	2022	PK	Int.	—	—	Int.				
	JP [202]	2023	LK	Int.	—	—	Int.				
	SBV [171]	2023	LK	Int.	—	—	Int.				  
II	Moser <i>et al.</i> [132]	2007	ZK	—	Ava.	—	Int.				 
	DroidChameleon [149]	2014	ZK	—	Int.	—	Int.				
	DexHunter [213]	2015	PK	—	Int.	—	—				
	Mystique [128]	2016	ZK	—	Int.	—	Int.				
	Mystique-S [199]	2017	ZK	—	Int.	—	Int.				
	MRV [205]	2017	ZK	—	Int.	—	Int.				
	SecureDroid [29]	2017	PK	—	Int.	—	Int.				
	Malton [198]	2017	PK	—	Ava.	—	Int.				
	Hammad <i>et al.</i> [73]	2018	ZK	—	Int.	—	Int.				
	RevealDroid [62]	2018	PK	—	Ava.	—	Int.				
	FARM [74]	2020	PK	—	Int.	—	Int.				
	API-Xray [39]	2021	PK	—	Int.	—	Int.				
	MalGraph [106]	2022	PK	—	Int.	Int.	Int.				
RLOBF [181]	2022	ZK	—	Int.	—	Int.					
CorDroid [60]	2023	PK	—	Ava.	—	Int.					
III	Smutz <i>et al.</i> [163]	2016	PK	—	—	Int.	Int.				
	Wang <i>et al.</i> [183]	2017	PK	—	Int.	Int.	Int.				
	Sec-SVM [49]	2017	PK	—	—	Int.	Int.				
	KuafuDet [33]	2018	PK	—	—	Int.	Int.				
	Fleshman <i>et al.</i> [57]	2018	PK	—	—	Int.	Int.				
	Chen <i>et al.</i> [37]	2020	PK	—	—	Int.	Int.				
	Li <i>et al.</i> [102]	2021	PK	—	—	Int.	Int.				
	Lucas <i>et al.</i> [113]	2023	PK	—	—	Int.	Int.				
	PAD [95]	2023	PK	—	—	Int.	Int.				
Gibert <i>et al.</i> [65]	2023	PK	—	—	Int.	Int.					

Continued on next page...

Table 4. (continued)

Stage	Proposal	Year	Proposal Properties								
			Know. Assu.	CIA Principles				Open Source	Real World	Proposal Type	Malware Format
				D	\mathcal{F}	C	o				
	RS-Del [80]	2023	PK	—	—	Int.	Int.	✓	✗	🛡️	📄
	MaskDroid [215]	2024	PK	—	—	Int.	Int.	✓	✗	🛡️	📄
	AutoRobust [173]	2024	PK	—	—	Int.	Int.	✓	✓	🛡️	📄
IV	Mimicus [166]	2014	LK	—	—	—	Int.	✓	✗	🔪	📄
	EvadeML [193]	2016	ZK	—	—	—	Int.	✓	✗	🔪	📄
	EvadeHC [45]	2017	ZK	—	—	—	Int.	✗	✗	🔪	📄
	Grosse <i>et al.</i> [71]	2017	PK	—	—	—	Int.	✗	✗	🔪	📄
	Transcend [84]	2017	PK	—	—	—	Ava.	✓	✗	🛡️	📄
	Pierazzi <i>et al.</i> [143]	2020	PK	—	—	—	Int.	✓	✗	🔪	📄
	Android HIV [34]	2020	PK	—	—	—	Int.	✗	✗	🔪	📄
	Rosenberg <i>et al.</i> [154]	2020	LK	—	—	—	Int.	✗	✗	🔪	📄
	HRAT [214]	2021	LK	—	—	—	Int.	✓	✗	🔪	📄
	OFEI [192]	2021	LK	—	—	—	Int.	✗	✗	🔪	📄
	GAMMA [47]	2021	ZK	—	—	—	Int.	✗	✓	🔪	📄
	Lucas <i>et al.</i> [114]	2021	LK	—	—	—	Int.	✗	✓	🔪	📄
	Li <i>et al.</i> [103]	2021	ZK	—	—	—	Int.	✗	✗	🔪	📄
	CADE [204]	2021	PK	—	—	—	Ava.	✓	✗	🛡️	📄
	Li <i>et al.</i> [99]	2021	PK	—	—	—	Int.	✗	✗	🛡️	📄
	MAB [164]	2022	ZK	—	—	—	Int.	✓	✓	🔪	📄
	BagAmmo [100]	2023	LK	—	—	—	Int.	✗	✓	🔪	📄
	Zhang <i>et al.</i> [209]	2023	LK	—	—	—	Int.	✗	✗	🔪	📄
	URET [54]	2023	LK	—	—	—	Int.	✓	✗	🔪	📄
	AdvDroidZero [76]	2023	ZK	—	—	—	Int.	✓	✓	🔪	📄
	Sun <i>et al.</i> [170]	2023	PK	—	—	—	Int.	✓	✓	🔪	📄
	Rigaki <i>et al.</i> [152]	2023	LK	—	—	Con.	—	✗	✗	🔪	📄
	TRANSCENDENT [19]	2023	PK	—	—	—	Ava.	✓	✗	🛡️	📄
MalProtect [148]	2023	PK	—	—	—	Int.	✗	✗	🛡️	📄	
MalGuise [108]	2024	ZK	—	—	—	Int.	✓	✓	🔪	📄	

3.3 Security Risk Assumptions

The threat model also contains knowledge and capabilities, according to previous works [11, 24, 76, 143]. We utilize the knowledge and capabilities to represent the assumption of the security risks.

Knowledge. Drawing from an ML-based MD system $\mathbb{F} = \{D, \mathcal{F}, C\}$ described in Section 2.1, the knowledge can be categorized into perfect knowledge, limited knowledge, and zero knowledge.

In the perfect knowledge setting, the adversary has all the knowledge of the three components, which means that the adversary fully accesses the software dataset, feature space, and classifier. In the limited knowledge setting, the adversary can only access part of the three components. For instance, the adversary may be aware of the feature space but remains oblivious to the training dataset and the ML classifier. In the zero knowledge setting, the adversary remains uninformed about all three components. However, the adversary can acquire certain open-source knowledge, such as access to publicly available benign software from software markets or datasets. In practice, the adversary usually has zero knowledge about the target ML-based MD system.

In contrast, the defender is usually the developer of ML-based MD systems. Hence, the defender has the perfect knowledge of the three components of ML-based MD systems. However, as attack strategies continue to evolve, defenders might be abreast of known tactics but could remain uninformed about emerging threats. Additionally, dedicated defenders may be aware of a small group of samples introduced by the adversary.

Capabilities. It is generally assumed that adversaries can manipulate the software samples. In some contexts, they might even have the capability to poison the training dataset of the target ML-based MD system. This assumption is based on the fact that some real-world antivirus solutions (e.g., VirusTotal [178]) involve human votes to label the software samples. In practice, the adversary usually has a limited ability to query the target model as more queries imply higher financial costs and an increased probability of detection. Moreover, the feedback of the target ML-based MD system is available to the attacker in some cases, e.g., VirusTotal provides the detection results of every antivirus engine. However, there might also be some cases in which the attacker cannot easily inspect the output of the target ML-based MD system [11].

In contrast, the defender can conduct extensive analysis, including static analysis and dynamic analysis of the samples. Being typically involved in the development phase of ML-based MD systems, defenders can allocate significant computational resources during this phase, even if it entails ramping up training computational demands. However, it is vital for defenders to minimize resource-intensive computations during deployment. This is because the high computational operations may influence the detection efficacy of the ML-based MD systems, which probably influences the user experience.

3.4 Decomposing Attack & Defense Strategies

We now briefly summarize the general strategies used by existing attack and defense proposals at each stage. Then, we provide a more holistic view of the attack and defense strategies under our stage-based taxonomy.

Stage 1. In the building dataset stage, raw software samples are processed by \mathcal{P} to construct the dataset \mathcal{D} , which is essential for developing \mathbb{F} . These selected software samples can be used for training the ML classifier or evaluating its performance. Attack proposals targeting this stage focus on injecting manipulated software samples into the training dataset. Although there are no existing defense proposals at this stage, the theoretical objective would be to purify the dataset to maintain the system’s integrity and availability without compromising utility.

Stage 2. In the extracting features stage, \mathbb{F} employs program analysis to obtain the program semantics and behavior of the software sample. Following program analysis, \mathcal{F} is applied to extract software features for further processing. Attacks at this stage typically employ obfuscation techniques to prevent program analysis from accurately identifying malware features. In response, defense proposals aim to extract robust feature representations through feature selection, minimizing irrelevant and potentially harmful features.

Stage 3. In the creating an ML model stage, \mathbb{F} employs ML algorithms (e.g., SVM) to construct C . This stage, primarily under the developer’s control (typically the defender), focuses on training C for \mathbb{F} . Therefore, the defense proposals

at this stage aim to enhance the robustness of the ML model by designing robust model architectures and training methods. Conversely, while there are no existing attack proposals, the theoretical objective would be to disrupt the training process, leading to crashes in training or reduced performance of the trained model.

Stage 4. In the deploying an ML model stage, \mathbb{F} deploys C in specific scenarios, such as on user devices, to classify unknown software samples. Attack proposals at this stage utilize adversarial malware example generation methods to perturb malware samples or craft specific queries to extract the model. On the defense side, current efforts focus on preserving the integrity of detection results through adversarial malware example detection and maintaining the availability of detection results through concept drift detection.

We remark that we only summarize the existing strategies at each stage above. However, under our stage-based taxonomy, every CIA principle can be targeted at each stage. Thus, we further provide a more holistic view of the attack and defense strategies under our stage-based taxonomy in Table 3. For the dataset, leaking dataset (e.g., inference attack) breaks confidentiality, poisoning the dataset to inject a backdoor or degrade system performance, breaks integrity, and corrupts the dataset to forbid up-to-date malware dataset breaks availability. For the feature extractor, leaking extraction function breaks confidentiality, obfuscating features to hide malicious features breaks integrity, and blocking analysis to prevent feature extraction breaks availability. For the classifier, leaking model architecture and parameters breaks confidentiality, hijacking model (e.g., rowhammer attack [175]) breaks integrity, and we are not aware of any availability risks other than traditional attacks such as DDoS. For the detection results, a high false negative rate reduces recall, thus breaking integrity, and a high false positive rate raises false alarm, which prevents practical use of the system, thus breaking availability. There is no confidentiality risks at the detection results stage, as the detection results are usually shared with the user. Due to the stage-based taxonomy, this discussion serves as a guideline for developers to understand the potential risks at each stage and design corresponding defenses against existing and future attacks.

3.5 Existing Attack & Defense Proposals

3.5.1 Paper Review Methodology. We start the paper review methodology by identifying seed papers published at top security computer security venues (i.e., IEEE S&P, ACM CCS, USENIX Security, NDSS, TDSC, and TIFS) from 2019 to 2024, and lightweight snowballing [86] the reference lists. Then, we adopt the paper selection criteria to select the relevant and qualified studies in our preliminary paper list followed by Liu *et al.* [112]. We employ the same first 5 exclusion criteria of paper selection in Liu *et al.* [112] together with the exclusion criterion which papers are not related to the security risks of ML-based MD system.

3.5.2 Representative Proposal Overview. We now present a comprehensive taxonomy of existing attack and defense proposals for ML-based MD systems with the stage-based taxonomy established in Section 3.2 and 3.4. As depicted in Table 4, we categorize these proposals by target stage, year of release, and proposal properties. The proposal properties are further broken down by knowledge assumption, CIA principles in the components and detection results, open-source status, real-world evaluation, attack or defense classification, and malware format. We summarize 33 representative attack proposals and 26 representative defense proposals, totaling 59 proposals in Table 4.

Research concerning each stage of ML-based MD systems focuses on specific targets of the CIA principles, as well as showing a preference towards attacks or defenses in some stages. For example, Stage 1 attacks compromise the integrity of the file dataset and detection results through dataset poisoning techniques, while in other domains such as text, dataset poisoning is found effective to compromise confidentiality as well [186]. In contrast, Stage 3 only has defense proposals aimed at enhancing the integrity of the ML model and detection results using robust architectures

and training methods, while in other domains, Stage 3 attacks are also found effective [116]. Therefore, developers should prioritize specific types of attacks and defenses at each stage but also be aware of potential future risks of other types.

While attack and defense proposals across stages share the goal of influencing the integrity of detection results, they differ in stages. For instance, Stage 1 attacks achieve this through dataset poisoning, while Stage 4 attacks generate adversarial malware examples. Similarly, Stage 2 defenses preserve integrity through feature transformation, while Stage 3 defenses focus on robust model training. Therefore, ML-based MD systems may face attacks at multiple stages simultaneously and can benefit from combined defenses at multiple stages.

Most attack and defense proposals target the integrity of detection results, as misclassifying malware as benign directly impacts system performance. Recent works also consider the confidentiality risk of the ML model and the availability risk of the detection results. Attack proposals in different stages require varying knowledge assumptions: Stage 1 attacks requires at least partial knowledge of the system and Stage 2 attacks usually requires zero knowledge. In addition, while certain attacks are evaluated against real-world systems, none of the defense proposals have undergone real-world evaluation, leaving their effectiveness uncertain.

Finally, we remark that many proposals are not open-sourced, which hinders the reproducibility and evaluation of their effectiveness. This is particularly harmful as Table 4 has shown increased interest in this area recently. We encourage future works to release their code and datasets to facilitate the reproducibility and evaluation of their proposals and ask the community to take action to address this issue.

4 Security Risks in Building Datasets

4.1 Technical Progress

Dataset Poisoning Attacks. Stage 1 attacks primarily target the integrity of the file dataset D and the detection results o , instantiated via dataset poisoning. Such dataset poisoning attacks typically manipulate the raw data by injecting malicious samples or perturbing existing samples to compromise the classifier C . The most common objective for existing dataset poisoning attacks is to inject malicious behaviors into the classifier, thereby causing misclassification when a specific trigger is present in the input, with the focus of misclassifying malware as benign. Such attacks are also widely known as *backdoor attacks* and these triggers are called *backdoor triggers*. Unlike backdoor attacks in other domains such as images, backdoor triggers in malware detection are often generated using specialized algorithms [93, 159, 171, 202, 212] or ML model explanation methods [59, 72, 115, 130].

Severi *et al.* [159] propose a backdoor attack method that utilize the ML model explanation method to generate the trigger to attack MD systems using categorical features. In this attack, they utilize the SHAP value [115] to identify the most vulnerable feature and propose a greedy method for trigger computation within the training dataset for the target model, indicating a perfect knowledge setting. They also evaluate their method in the transferability attack scenarios and the experimental results show that they achieve a large accuracy drop on watermarked malware within a small poison rate.

SBV attack [171] is a clean label backdoor attack that exploits the sparsity of feature subspaces of malware samples. This attack proposes a dissimilarity metric-based candidate selection method that utilize nonconformity measures with P-values and a variation ratio-based trigger construction strategy. The SBV attack is evaluated under both data-agnostic and data-and-model-agnostic scenarios and the experimental results show that the proposed attack can reduce the detection accuracy on watermarked malware to nearly 0%, even when poisoning as little as 0.01% of the benign class.

Li *et al.* [93] present a stealthy backdoor attack that uses a Genetic Algorithm-based trigger selection to identify minimal and semantically plausible feature modifications that can be injected into apps. To achieve stealthiness, this method proposes a label-reserving method to ensure the the poisoned samples are mislabeled as benign. The experimental results show that the method achieves 99% evasion rate with a poisoning rate as low as 0.3%, and minimal impact on clean sample classification.

JP attack [202] introduces a selective clean-label backdoor attack designed to increase the stealthiness of backdoors against ML-based MD systems. Unlike prior universal backdoors that aim to misclassify all triggered malware, JP specifically targets a selected malware family or set controlled by the attacker, leaving other malware and benign samples unaffected. JP makes the trigger more stealthy by limiting its effective domain to a specific target family of malware samples using alternate optimization [139]. Experimental results on a large Android malware dataset show that JP achieves high attack success rates (e.g., over 90% for many families) on the attacker’s malware with negligible false positives and no degradation to main-task performance.

Except for the backdoor attacks, dataset poisoning can also degrade the performance of the ML-based MD system by manipulating the decision boundary of the classifier [21]. We remark that while recent works on dataset poisoning in the malware domain have focused on backdoor attacks, degrading the performance of the ML-based MD system via dataset poisoning worths further investigation, since works towards this direction are outdated and underrepresented while the potential risks are significant as well.

Dataset Poisoning Defenses. As discussed in Section 3.1, eliminating the security risks as soon as possible is crucial. However, our analysis reveals a significant gap in defense proposals compared to the attack proposals at Stage 1, as there is no defense about dataset processing to protect ML-based MD systems. In principle, potential Stage 1 defenses aim to purify the dataset, thus removing samples that lead to malicious behaviors or degrade the performance of the system from the training set. Nevertheless, defenses in later stages can also help to defend attacks in earlier stages. It is particularly notable that existing defenses against dataset poisoning attacks at Stage 1 belong to Stage 3 and Stage 4, although they are mostly taken from other domains and evaluated in attack proposals. For example, Severi *et al.* [159] discuss activation clustering [28] and isolation forest [109], and Yang *et al.* [202] evaluate STRIP [61], Neural Cleanse [180], and MNTD [194]. Additionally, Tian *et al.* [171] evaluate 10 different defense strategies against backdoor attacks originally developed for other domains, but find that these methods are not satisfying in the malware domain. Later, Qi *et al.* [145] propose a proactive machine learning approach for detecting backdoor poison samples in datasets. They propose confusion training which trains a model on both the potentially poisoned dataset and a small set of clean samples with randomized labels. This process breaks normal feature-label associations for clean data, so only poison samples with backdoor triggers remain memorized by the model. Experiments results demonstrate the defense effectiveness of cofusing training against attacks proposed by Severi *et al.* [159] in the Windows PE malware domain.

4.2 Discussion & Open Problems

Regarding data poisoning attacks, the current focus is on injecting backdoors, but the degradation of the detection performance is also a significant threat to the availability of ML-based MD systems. However, such attacks are underrepresented, with only one early work (published in 2014) discussing such threats. Therefore, the community could discuss such risks more modernly since ML-based MD systems have evolved significantly since then. In addition, the backdoor attacks in the malware domain are particularly effective, with extremely low poisoning rates, indicating a high stealthiness. However, such strong attack performance is established on an impractical assumption that the dataset is randomly split. In practice, the malware samples have the timestamps in the malware dataset. Thus, recent ML-based

MD systems split the dataset according to the timestamps [12, 142]. This temporal nature may affect the effectiveness of the backdoor attacks, and the community should investigate this aspect further.

Regarding defense against dataset poisoning attacks, one remarkable observation is that existing defenses all belong to later stages, i.e., there is no defense about dataset processing to protect ML-based MD systems at Stage 1. According to the discussion in Section 3.4, the earlier the security risks are eliminated, the better. Therefore, the community could investigate potential defense strategies at Stage 1, such as dataset purification and dataset selection, to protect ML-based MD systems. Further, according to the empirical evidence, existing defenses against backdoor attacks are ineffective in the malware domain. This indicates a need for further adaptation and development of backdoor defense methods for malware detection tasks.

5 Security Risks in Extracting Features

5.1 Technical Progress

Feature Extraction Attacks. ML-based MD systems heavily depend on program analysis (static and/or dynamic) to extract features from software, thus any disruption in this extraction process poses a critical threat to the entire system. There are two main avenues to attack feature extraction, namely *program obfuscation* and *variants generation*. The former aims to conceal malicious features, while the latter aims to rearrange malicious and benign features such that the extracted features of the malware variant have the same distribution as a benign sample. For example, consider a music player malware with the malicious behavior of reading locations; obfuscation aims to hide the location reading behavior, while variants generation aims to generate a variant for which reading location data seems benign, e.g., it has the same feature distribution as benign map applications.

Program obfuscation typically exploits the limit of static analysis in solving opaque constants by program control flow obfuscation [132] and incompleteness of static analysis by code transformation [9, 73, 119, 149, 181]. Early attacks on static analysis feature extraction use traditional obfuscation strategies [119, 149] such as repacking, reflection, string encryption and class encryption, etc., while recent attacks [181] employ reinforcement learning to generate code obfuscation sequences.

Rastogi *et al.* [149] evaluate the adversarial robustness of anti-virus engines by applying three trivial transformations, eight transformations detectable by static analysis and two transformations non-detectable by static analysis. The experimental results on 10 popular commercial anti-virus solutions demonstrate that malware can easily evade detection by applying trivial transformations. Hammad *et al.* [73] utilize 29 obfuscation techniques to evaluate the effectiveness of the top anti-malware products and find that the majority of anti-malware products are severely impacted by even trivial obfuscations.

In addition, runtime packing [4, 121, 136, 174], another obfuscation method, compresses/encrypts the software and decompresses/decrypts the code at runtime. Mantovani *et al.* [121] conduct a large scale evaluation about 50K low-entropy Windows malware. The experimental results show that ML-based models cannot identify low-entropy packed samples using static analysis features. Aghakhani *et al.* [4] train the ML model to detect the packed malware samples. They find that the features extracted from the packed malware are not enough for ML-based models to identify other unseen packers and resist adversarial examples.

Regarding dynamic analysis which relies on sandbox execution, the focus is on sandbox detection, which aims to identify whether the software is running in a sandbox environment, thus enabling the malware to evade detection. Miramirkhani *et al.* [131] finds that the wear-and-tear artifacts in the dynamic execution environments can be used as

features for the sandbox detection. Kondracki *et al.* [90] utilize the Android API to extract environment-related features to train the classifier to detect the Android sandbox. The experimental results show that the ML classifier can achieve 98.54% accuracy in distinguishing between real Android devices and well-known mobile sandboxes.

Variants generation typically exploits the incompleteness of the extracted features. Early works [128, 199] resemble the defined modularized attack features and evasion features using evolutionary algorithms, while recent works [134, 135, 158] further employ code transformation methods to generate malware variants more efficiently [134, 135, 158]. They either apply functionality-preserved assembly code transformation [134, 135] or insert benign code segments [158]. In particular, Sen *et al.* [158] investigate the coevolutionary arms race between mobile malware and anti-malware, introducing a fully automated framework based on genetic programming for both generating evasive malware variants and the malware detectors.

Feature Extraction Defenses. All existing defenses against feature extraction attacks are employed at Stage 2, in contrast with defenses against Stage 1 attacks. There are two main types of defenses against program obfuscation attack, namely *program deobfuscation* and *robust representation*.

Program deobfuscation tries to extract malicious behaviors from obfuscated malware. Early works [52] traces program execution with hardware virtualization extensions, while later works focus on interested system calls and their arguments identified by dependence and control flow analysis [42] or data mining [88], thus improving recall of malicious behaviors by ignoring irrelevant code.

More recent works [3, 198] further monitor malicious behaviors at runtime in an on-device and non-invasive way. Xue *et al.* [198] introduce Malton, an on-device, non-invasive dynamic analysis platform designed for comprehensive behavior analysis of Android malware running under the ART runtime. Malton provides multi-layer monitoring and information flow tracking, bridging semantic gaps between Java and native code, and tracking sensitive data flows even across JNI and reflective calls. Additionally, it integrates a path exploration engine using concolic execution to trigger conditionally executed behaviors.

To defend against runtime packing, defense proposals typically adopt unpacking mechanisms to extract the hidden code within the packed malware. Early works extract hidden code with hybrid analysis [157] or execution monitoring [127], while later works [195, 196, 213] identify collection points to recover the packed code. More recent works [38, 39, 197] unpack the packed malware via hardware-assisted approaches.

API-Xray [39] reconstructs a working import table from an unpacked malware's process memory at its original entry point, making the sample executable and its API calls analyzable for further static/dynamic inspection. It utilizes Intel's Branch Trace Store and NX bit hardware features to track the real target APIs. Extensive experiments demonstrate that API-Xray can recover import tables for complex commercial packing products.

Happer [197] unpacks the Android apps by leveraging ARM hardware tracing and hardware breakpoints to non-invasively monitor the actual runtime behaviors on real devices. Based on the detected behaviors, Happer then adaptively selects the best strategy to extract all hidden Dex data from memory, even when packers release code just-in-time or modify runtime objects, and reconstructs valid Dex files suitable for further static analysis. Extensive experiments with 12 commercial packers and over 24,000 Android apps show that Happer reveals 27 distinct packing behaviors and achieves much higher unpacking coverage and fidelity than existing tools.

LoopHPCs [38] investigate the feasibility of using hardware performance counters (HPCs) for malware unpacking, addressing recent skepticism about the non-deterministic nature of HPCs for security uses. Specifically, it utilizes the Precise Event-Based Sampling and Last Branch Record offered by Intel CPUs to unpack malware. Extensive experiments show that LoopHPCs achieves high unpacking accuracy and outperforms state-of-the-art software-based unpackers.

Instead of extracting obfuscated malicious behaviors in a post-hoc manner, robust representation aims to design a feature space that can hardly be obfuscated. One way to achieve this is by transforming features into a more robust space. Early works use feature selection strategies based on manipulation cost [29] and transform the original binary features into continuous probabilities [30], while more recent works capture robust API features against Android system updates [210] and transform the feature space into a more robust representation using an ensemble of feature transformations [74].

APIGraph [210] builds a robust feature space by incorporating the evolution of the Android APIs. It clusters the features with semantic similarity information among Android APIs by constructing an API relation graph from official Android documentation. The experimental results demonstrate that APIGraph’s clusters capture semantic similarity within malware families and that semantically-close APIs are grouped together beyond package-level abstraction.

Han *et al.* [74] propose FARM that uses irreversible feature transformations to defend against evasion attacks targeting machine learning classifiers. FARM begins by extracting standard static, dynamic, and API package call features from Android apps, then applies three novel classes of feature transformations to map these features into a new space.

Another way is to use obfuscation-resilient representation directly. Early works [1, 62, 118] use reflection-based features or features that are prohibitively difficult to manipulate, while more recent works [60, 79, 106] use graph-related features that are resilient to most obfuscation methods. These feature transformations make it difficult for attackers to manipulate features in a way that will fool the classifier.

CorDroid [60] leverages the complementarity of multiple semantic features to withstand a wide variety of bytecode-level obfuscation techniques. It employs the Enhanced Sensitive Function Call Graph and Opcode-based Markov transition Matrix as features. Experimental results demonstrate that it achieves defense effectiveness in detecting obfuscated Android application samples.

MaskDroid [215] introduces a masking-based self-supervised learning approach within a GNN framework. In the training process, MaskDroid is tasked with reconstructing the full graph from the remaining nodes and incorporates a proxy-based contrastive module to compress intra-class variance and sharpen the decision boundary. Extensive experiments on a large AndroZoo-sourced dataset demonstrate that MaskDroid reduces attack success rates from 41.5% to 32% under the white-box attacks setting, while maintaining competitive detection effectiveness and efficiency.

5.2 Discussion & Open Problems

Regarding feature extraction attacks, existing program obfuscation techniques have been shown to be effective against common feature extractors, especially in the wild, since obfuscation rarely targets specific feature extractors. However, as obfuscation adds many instructions to the original code, some even being executed, this brings memory and computation overhead to the original code. This problem is particularly important when program obfuscation is used for good, e.g., protecting copyrights. The other attack direction, malware variant generation, requires prior knowledge about the malicious features, which limits its generalization to malware families. Therefore, whether the automatic creation of malware variants is possible without human-designed attack features is an open question.

Regarding defense proposals, the SOTA deobfuscation works are limited in practice since they typically require specialized hardware and software. For instance, Malton is based on the Valgrind framework, which can be evaded by anti-emulator techniques; LoopHPCs does not support AMD processors due to the lack of hardware tracing mechanisms like LBR. Robust feature space methods, especially those that utilize function call graph-based features, are no longer robust because recent attack proposals have leveraged adversarial machine learning techniques to manipulate function

call graph-based features, thus breaking their underlying assumption. Therefore, better robust feature space methods that are not based on function call graphs are needed.

6 Security Risks in Creating ML Models

6.1 Technical Progress

Unlike other stages, while attacks at Stage 3 (model hijacking) are possible via injecting malicious code into the training framework (malicious wheel builder, e.g., open-source project contributors) and manipulating the training hardware (malicious hardware provider, e.g., cloud platforms), these attacks are not yet specialized for ML-based MD systems. Therefore, we focus on the defense proposals at this stage. Instead of defending against model hijacking, Stage 3 defenses defend against various attacks at different stages, namely backdoor attacks (Stage 1) and adversarial malware attacks (Stage 4). Backdoor defenses have been discussed in Section 4. Thus, we focus on defenses against adversarial malware attacks in the following. Such defenses can be roughly categorized into *robust model architecture* and *robust model training*. In the following, we illustrate each approach in more detail.

Robust Model Architecture. Enhancing the robustness of ML-based MD systems can be achieved through inherently robust model architectures. This involves using ensemble learning techniques [96, 97, 163, 167] by employing voting mechanisms [96, 97] or the diversity of an individual classifier to identify the evasion samples [163]. Additionally, the defender could also utilize another model as a complement to the vanilla model to capture the features of adversarial malware examples.

KuafuDet [33] introduces a two-phase, self-adaptive learning framework containing offline training and online detection modules. The online detection module employs a camouflage detector to identify and re-label suspicious false negatives for iterative retraining. Evaluations on a large dataset show that KuafuDet achieves higher detection accuracy than Drebin and is robust under severe data imbalance.

Li *et al.* [102] utilized a variational autoencoder to measure the reconstruction error of the input samples to filter out adversarial malware examples. The system makes a final decision by aggregating the outputs of both the VAE and MLP, declaring an app benign only if both models agree. Extensive experiments on a large real-world dataset demonstrate that this method remains effective against both white-box and black-box adversarial attacks.

The recent SOTA work [218] maintains a dynamic pool of heterogeneous models, each trained on distinct data partitions and further diversified through adversarial training with multiple perturbation strategies, thereby reducing the exposure and transferability risks associated with single-model detectors. To optimize detection robustness, MTDroid employs a two-stage sub-classifier selection algorithm for ensemble learning and introduces a hybrid update strategy that dynamically refreshes the model pool based on query and failure statistics. Experimental results demonstrate that MTDroid achieves state-of-the-art performance in both detection accuracy and robustness.

Another approach [49, 94, 104, 183] to enhance the ML classifier robustness involves designing constraints to process the feature using part of the model architecture. Wang *et al.* [183] proposed an extra layer for random feature nullification between the input and the first hidden layer in DNN to achieve the robustness by making the DNN model non-deterministic. RFN introduces stochasticity in both training and inference by randomly nullifying input features, making the model non-deterministic and significantly hindering an adversary’s ability to craft effective adversarial perturbations.

Sec-SVM [49] starts from the intuition of bounding classifier sensitivity and constrains the distribution of feature weights, effectively enforcing more evenly-distributed (dense) weights via ℓ_∞ -norm regularization. This approach

significantly increases the number of feature manipulations required for a successful evasion. Experiments on large-scale real-world Android datasets show that Sec-SVM achieves state-of-the-art robustness across a variety of attack scenarios, outperforming both standard SVM and previously proposed bagging-based approaches.

Monotonic classifiers [57, 81] present another intriguing avenue for the integrity preserving. These classifiers maintain or increase the confidence in a malware label, irrespective of any additional features added to a sample. Most of the current adversarial malware example attacks only add benign features to the malware samples due to the problem space constraints, which cannot bypass the defense of the monotonic classifier.

Robust Model Training. The robustness of an ML classifier can also be achieved by utilizing robust model training methods. Adversarial training is the most common solution to train a robust model in the image domain [68]. The key intuition of adversarial training is incorporating adversarial examples in the process of training an ML classifier. To adapt the adversarial training techniques to the malware domain, current defense proposals [5, 31, 32, 58, 95, 144, 150, 160] propose the customized training loss and adversarial malware example attacks in the feature space for adversarial training. The earliest works [31, 32] introduce the extra regularization term in the training loss to enhance the training robustness. The later works [5, 58, 150, 160] focus on generating adversarial examples for adversarial training. For instance, Al-Dujaili *et al.* [5] proposed the Bit Gradient Ascent method designed for the discrete feature. Galovic *et al.* [58] proposed the adversarial strings generation method from the perturbed latent space. The recent SOTA work [95] designs the Stepwise Mixture of Attacks with convergence guarantees for searching adversarial perturbations in the feature space.

To explore the effectiveness of adversarial training in the domain of raw-binary malware detectors, Lucas *et al.* [113] utilize adversarial training for raw-binary malware detection by generating adversarial examples through In-Place Replacement, Displacement, and Kreuk attacks. Their experiments reveal that standard data augmentation with random, functionality-preserving binary transformations provides little to no robustness against guided adversarial attacks. In contrast, adversarial training using guided, attack-specific adversarial examples significantly increases model robustness.

AutoRobust [173] utilizes a reinforcement learning-based adversarial training methodology to enhance the robustness of the ML-based MD systems based on dynamic analysis reports. It formulates the generation of adversarial examples as a Markov Decision Process, where an RL agent learns functionality-preserving transformations directly on dynamic analysis reports, ensuring all modifications are realistic and feasible for actual malware binaries.

However, adversarial training offers only empirical robustness of the ML classifier and may be susceptible to stronger attacks [43]. Thus, certified training [18, 63, 122, 125, 208] has been proposed to provide certified robustness of ML classifiers in the image domain [124, 133, 138] and text domain [53]. Current defense proposals [65, 80] in the malware domain also adapt certified defenses.

Chen *et al.* [37] propose a novel subtree distance metric on the PDF parse tree and specify two main classes of robustness properties: subtree insertions and subtree deletions under the PDF root. It leverages the verifiable robust training methods [184, 185] to build neural network classifiers that are formally guaranteed to be robust against any attacker bounded by these properties.

RS-Del [80] extends randomized smoothing [41, 201] to the edit distance setting by applying random deletions to input sequences. It only requires random deletions to certify robustness against all three edit operations, leveraging a proof that relies on longest common subsequence analysis rather than the standard Neyman-Pearson approach. It achieves a certified accuracy of 91% in the MalConv model at an edit distance radius of 128 bytes, with only a marginal drop in clean accuracy.

6.2 Discussion & Open Problems

Similar to other domains, increasing the adversarial robustness of ML-based MD systems via architecture design or specialized training methods often comes at the cost of clean accuracy and computational overhead. While this is more acceptable since attacks on MD systems are more common, it is unclear whether this trade-off can be resolved. For instance, FD-VAE [102] requires extensive hyperparameter tuning to maintain clean accuracy at the cost of multiplied training time. Moreover, robust training typically relies on specific train-time attacks, but the adversarial robustness is poorly generalized against other attacks. In particular, feature-space attacks are more efficient and frequently adopted in training, but such attacks often yield mismatches with realistic problem-space attacks due to different attack specifications. This often brings difficulty in having more practical certified robustness, as the input specification in the feature space is not norm-based, but current certified defenses only consider norms such as L_0 (patch) distance because such normed specifications are more established in other domains. In addition, while defenses against backdoor and adversarial malware attacks are extensively discussed, the confidentiality of the ML model is largely overlooked since no existing defense in the malware domain protects confidentiality, e.g., differential privacy.

7 Security Risks in Deploying ML Models

7.1 Technical Progress

This section is divided into four parts: model extraction attack (targeting confidentiality), adversarial malware attack (targeting integrity), adversarial malware defense (preserving integrity) and concept drift detection (preserving availability). This is because currently no specialized defense is designed to counter the model extraction attack. The availability risks at this stage are mainly caused by concept drift (similar to out-of-distribution in other domains), a phenomenon in which malware distribution shifts over time. This shift leads to high false positive rates and influences the availability of the detection results.

Model Extraction. Because detection results are shared with users in Stage 4, the ML models are exposed to model extraction attacks [78, 82, 137, 172, 206]. Model extraction attacks in other domains mainly extract the model functionality, model architecture and model parameter. However, in the malware domain, the feature extraction function \mathcal{F} is worth extraction as well. Early works [23, 190] extract the malware signatures and the detection logic of the anti-virus engines, and more recent works [152] focus on stealing ML model functionality.

Rigaki *et al.* [152] propose a model stealing attack to extract the model functionality of commercial antivirus engines. They design the dualFFNN, which leverages true labels with a skip connection for stable training, and the FFNN-TL, which combines transfer learning with active learning for improved surrogate accuracy.

Adversarial Malware Attack. To subvert the integrity of the detection results, current attacks utilize adversarial malware generation methods by perturbing the malware to evade detection. Most of the adversarial malware generation methods [2, 20, 34, 70, 71, 89, 111, 114, 168, 214] assume the attacker has the perfect knowledge or limited knowledge of ML-based MD systems. These assumptions are supported by the Kerckhoffs' principle [189] and represent the worst cases of the defender [143]. Under these scenarios, the adversarial malware generation algorithms may leverage optimization methods utilizing gradient-based optimization [34, 70, 71, 166, 214] or gradient-free optimization methods [91, 100, 154, 192]. In gradient-based optimization methods, the attacker may select perturbations with the largest gradients of the objective function. For example, Android HIV [34] refines the C&W attack [25] and JSMA attack [140] to generate adversarial malware examples in the feature space. HRAT [214] selects the manipulation type and manipulation targets using the gradient descent method. In gradient-free optimization methods, the attacker may leverage the hill-climbing

method [45], generative adversarial networks [101, 154], and genetic algorithms [100, 193]. The recent SOTA work [100] designs the generative adversarial network combined with a multi-population co-evolution algorithm to generate the desired perturbation. Moreover, the attacker [46] could also find the most vulnerable features in the feature space directly under the perfect knowledge setting. For instance, Pierazzi *et al.* [143] leveraged the weights of the SVM classifier to assess feature importance and subsequently searched for the most benign features to inject. Wang *et al.* [170] employed the SHAP values [115] to compute the accrued malicious magnitude of features for manipulation.

In practical scenarios, the internals of ML-based MD systems typically remain unknown to the attackers. Under the zero knowledge setting, existing attacks utilize extra knowledge inside the program semantics [22, 47, 54, 155, 205, 207, 209, 216] and query feedback [76, 103, 108, 151, 164]. They may involve mimicking benign samples [205], incorporating benign program slices [22, 47] or leveraging the malware program structure [209]. Moreover, they could also define the malware perturbations and utilize the query feedback to adjust the selection probability of these perturbations. The perturbation selection process can be modeled as the reinforcement learning process [7, 103, 151, 164]. The recent SOTA works [76, 108] utilize the perturbation selection tree, incorporating the perturbation semantics or Monte Carlo tree search for the effective perturbation selection.

He *et al.* [76] propose AdvDroidZero, generating adversarial Android malware samples under the zero knowledge setting. AdvDroidZero designs a perturbation selection tree that clusters the semantically similar perturbations together to improve the attack effectiveness and efficiency. The experimental results demonstrate that AdvDroidZero can achieve about 90% attack success rate against real-world anti-virus engines.

Ling *et al.* [108] propose MalGuise, a practical black-box adversarial attack framework targeting learning-based Windows malware detection systems. MalGuise introduces a semantics-preserving transformation called call-based redividing, which manipulates both nodes and edges of a malware's control-flow graph and leverages a Monte Carlo Tree Search algorithm to efficiently search for optimized sequences of transformations. Experimental results demonstrate that MalGuise achieves 74.97% against real-world anti-virus solutions.

Properties of adversarial malware attacks have gained attention as well. Prior works [50, 123] explore the transferability of adversarial examples, showing that similar conclusions hold for adversarial malware examples as well. Demetrio *et al.* [48] benchmark 7 adversarial malware example attacks in the context of Windows PE malware. They suggest that the size of injected bytes correlates with stealthiness, and model architecture has a greater impact on model robustness than training dataset size and activation function. Recent works [26, 207] generate universal adversarial perturbations and patches to understand perturbation adaptability across malware samples.

Adversarial Malware Defense. Defenses against adversarial malware examples either try to design robust models at Stage 3 (discussed in Section 6) or purify and abstain from adversarial malware examples at Stage 4. Rashid *et al.* [148] present MalProtect, which employs a suite of diverse threat indicators, including statistical and autoencoder-based anomaly scores, to analyze sequences of user queries before they reach the prediction model. These indicator scores are aggregated by a learned decision model to detect ongoing attacks. Experimental results across Android and Windows malware datasets demonstrate that MalProtect can reduce the evasion rate of query attacks by over 80%. To purify queries, Zhou *et al.* [217] employ a denoising autoencoder framework to reconstruct the input, thus removing adversarial perturbations.

Concept Drift Detection. Concept drift is a phenomenon where the dynamics of malware in the wild lead to performance degradation of the ML-based MD system. Hence, the current defense proposals [19, 84, 204] address this problem by detecting the concept drift samples to maintain the detection effectiveness.

Transcend [84] employs a conformal evaluator to compute non-conformity measures and p-values for incoming samples, assessing the statistical fit of each prediction relative to the training data. By monitoring algorithm credibility and confidence, Transcend can flag untrustworthy predictions and identify drifting samples in both binary and multi-class settings. TRANSCENDENT [19] builds upon this, refining the conformal evaluation theory and improving the efficiency of Transcend by calibration splits. To explain the concept drift samples, CADE [204] leverages autoencoder with contrastive learning to project the sample into the monitoring space and uses distance changes to explain the set of important features.

Another approach to address concept drift is the fine-tuning of the ML model. Chen *et al.* [35] propose a novel continuous learning framework that combines hierarchical contrastive learning and a new pseudo loss uncertainty measure to guide active learning and sample selection. The hierarchical contrastive learning explicitly models the similarity among malware families and between benign/malicious apps, improving generalization and detection of new malware families under severe class imbalance. Experimental results on the dataset spanning 7+ years show that this approach reduces the false negative rate from 14% to 9% and the false positive rate from 0.86% to 0.48%.

7.2 Discussion & Open Problems

Stage 4 attacks are the most extensively discussed, potentially due to the direct interaction between the ML-based MD system and the users. Current model extraction attacks are constrained by high query costs in order to be effective. In addition, the literature only discusses extraction attacks for certain malware types, which makes its generalization to other malware types questionable. Such extraction attacks also make strong knowledge assumptions, e.g., the knowledge about the model architecture. Future research could explore hardware-based model extraction attacks, such as side-channel reverse engineering, to reduce the cost and improve the effectiveness of these attacks. Among all the possible attacks, adversarial malware examples receive the most attention. However, SOTA adversarial malware attacks often disregard stealthiness, as they insert unlimited pieces of benign codes for evasion. Such attacks can hardly bypass monotonic classifiers in principle, because the inserted benign codes cannot reduce the maliciousness of the malware. Future research could discuss enhancing the stealthiness of adversarial malware examples by eliminating existing malware features without impacting the malicious functionality.

Arguably, the deployment stage is the most vulnerable to attacks, where the ML-based MD system is directly exposed to the users. However, the literature on defense at Stage 4 is limited, while defenses against attacks at this stage focus on Stage 3. In particular, no defense can identify adversarial malware examples based on individual queries, with few defenses relying on monitoring the continuous query history and purifying the input. Such defenses may be easily evaded if the attacker is aware of the defense mechanism. In addition, methods used to detect concept drifts are similar to those used to detect adversarial malware examples. Future research could explore the possibility of unifying and generalizing these two detection methods.

8 Delving into Stage-wise Interconnections

We present a stage-based taxonomy, which decomposes the security risks in ML-based MD systems into operational stages. In this section, we will focus on the interconnections between these stages to provide new empirical insights. Since the taxonomy is designed to be comprehensive, such insights cannot be exhaustive. Thus, we conduct two specific case studies: one inter-stage study connecting poisoning and evasion attacks (Stages 1 and Stage 4) and one intra-stage study connecting evasion and concept drift detection (Stage 4). These case studies aim to provide a deeper understanding of the interconnections between different stages and to inspire future research directions.

Table 5. The attack performance of adversarial example attacks measured by attack success rate under poisoned and benign models. “Feature, Problem, Non Problem” represent the trigger constraints.

Family	Feature		Problem		Non Problem	
	Poison Model	Benign Model	Poison Model	Benign Model	Poison Model	Benign Model
mobisec	0.38	0.28	0.40	0.28	0.23	0.29
tencentprotect	0.33	0.22	0.41	0.34	0.41	0.20
anydown	0.46	0.36	0.37	0.33	0.23	0.30
leadbolt	0.43	0.31	0.47	0.31	0.37	0.29
eldorado	0.28	0.29	0.18	0.32	0.34	0.34
jiagu	0.33	0.29	0.29	0.32	0.30	0.22
kuguo	0.13	0.31	0.31	0.31	0.21	0.32

Dataset. Our dataset comprises 134,759 benign applications and 14,775 malware samples, totaling 149,534 applications between January 2015 and October 2016. This dataset is the same as the previous work [202]. We choose this dataset because the malware family information is directly available, which is required by the JP attack.

Implementation Details. To provide case studies, we have developed an evaluation framework grounded in the stage-based taxonomy approach. This framework implements the ML-based MD methods following the stage-based design, integrating the SOTA attack and defense proposals [19, 76, 202]. Our case studies focus on the Android malware domain, a choice motivated by the abundance of data resources available in this area. For example, AndroZoo [6] has over 24 million APKs with rich analysis results. Besides, the current SOTA proposals [19, 76, 202] primarily focus on the Android domain. However, our evaluation framework can be easily extended into other malware formats due to the stage-based design. Other potential users only need to replace the software feature analysis module with the corresponding feature extractor, which requires relatively small workloads.

8.1 Poisoning and Evasion Attack Interactions

This subsection studies the interaction between backdoor (poisoning) and adversarial malware attacks (evasion) in ML-based MD systems. This study is inspired by previous works in the image domain [139, 187], which find these two attacks are strongly correlated and boost each other. However, software, characterized by its discrete format and unique problem space constraints, fundamentally differs from images. Consequently, such conclusions need to be revisited in the malware domain. To instantiate our study, we choose AdvDroidZero [76] as the adversarial example attack and JP [202] as the poisoning attack, because they are the SOTA methods in their respective fields.

We first train the poisoned models utilizing JP with different trigger constraints and the benign models in our dataset. Then, we conduct the adversarial malware attack utilizing AdvDroidZero under a 10-query budget on 100 malware samples, selected uniformly randomly from malware identified by both the poisoned and the benign models. We re-implement JP and integrate it into our evaluation framework using PyTorch [141], following the descriptions and configurations provided in the original paper and their open-source code. To be specific, we use the same hyperparameters with JP to compute the trigger and set the poisoning rate as 0.001, which corresponds to 100 benign samples. As for the target model, we use an MLP binary classifier with three hidden layers of 1,024 neurons and a dropout rate of 0.5. As suggested by their original implementation, we use LinearSVM regularizer to select the top 10,000 features and randomly split the dataset according to the ratio of about 7:3. For our experiment, we randomly choose 7 different malware families with various sample sizes as the targeted family. We also extend the JP attack to various constraints,

Table 6. The attack results of the JP attack under different trigger spaces and different target families. “AT, AR, FB, F1” = Attack Success Rate in samples with target family, Attack Success Rate in samples without target family, False Positive Rate in benign samples, F1 scores in malware detection task.

Target Family	Apps Num	Space	Trigger Size	AT	AR	FB	F1
mobisec	57	Feature	22	1.00	0.14	0.00	0.90
		Problem	14	0.64	0.02	0.04	0.92
		Non-problem	25	1.00	0.35	0.07	0.92
tencentprotect	157	Feature	46	0.98	0.51	0.00	0.92
		Problem	37	0.98	0.42	0.00	0.92
		Non-problem	16	0.74	0.23	0.00	0.91
anydown	192	Feature	20	0.95	0.03	0.02	0.91
		Problem	19	1.00	0.24	0.00	0.92
		Non-problem	18	0.98	0.09	0.00	0.92
leadbolt	222	Feature	34	0.91	0.13	0.00	0.91
		Problem	28	0.56	0.06	0.00	0.92
		Non-problem	24	0.93	0.68	0.00	0.91
eldorado	335	Feature	32	0.88	0.32	0.00	0.92
		Problem	51	0.92	0.37	0.00	0.91
		Non-problem	26	0.72	0.61	0.00	0.92
jiagu	668	Feature	38	0.84	0.16	0.00	0.91
		Problem	42	0.96	0.48	0.00	0.92
		Non-problem	22	0.90	0.80	0.00	0.92
kuguo	2,897	Feature	28	0.90	0.26	0.00	0.92
		Problem	37	0.83	0.33	0.00	0.92
		Non-problem	31	0.85	0.30	0.00	0.92

including optimizing triggers in the problem space and feature space, as well as optimizing triggers in the non-problem space. The JP attack’s effectiveness is detailed in Table 6, demonstrating successful backdoor injection and verifying our implementation’s fidelity.

To conduct adversarial example attacks, we utilize AdvDroidZero [76], reconstructing its malware perturbation set by re-extracting the code perturbation in our test set. We then select 100 malware samples, identifiable by both poisoned and benign models, and apply AdvDroidZero with the 10 query budget to generate adversarial malware examples.

Table 5 reveals that the poisoned model does not consistently exhibit higher vulnerability to adversarial malware examples compared to the benign model. It can be observed that there are some cases in which the attack success rates in the poisoned model are not higher than the benign model whenever the constraints of the trigger. Notably, in cases where triggers are optimized across the entire feature space, certain families like kuguo and eldorado exhibit no increase in attack success rates. We hypothesize that such triggers might not be effective in problem space due to problem space constraints. To validate our assumption, we truncate the trigger into the problem space and validate its backdoor effectiveness in the poisoned model. The attack success rates in samples with the kuguo family and eldorado family of truncated triggers are both nearly 0, indicating no effect in problem space. The attack success rates of remaining families, i.e., mobisec, tencentprotect, anydown, leadbolt, jiagu, are 0.250, 0.238, 0.438, 0.114, 0.208, respectively, indicating their effectiveness in problem space.

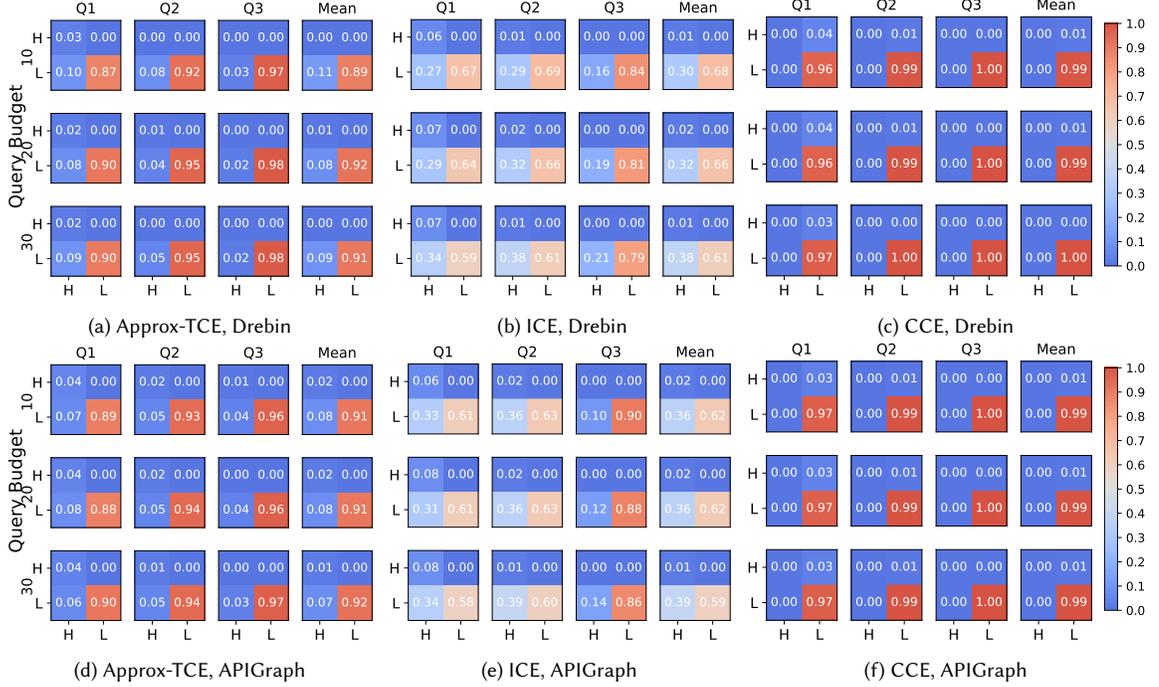


Fig. 2. The detection performance of TRANSCENDENT (three instantiations: Approx-TCE, ICE and CCE) against adversarial malware examples generated by AdvDroidZero with different query budget (10, 20 and 30) on different ML-based MD systems (Drebin, APIGraph). Given a predefined threshold (25%, 50% and 75%, denoted by Q1, Q2 and Q3, respectively), an input is classified based on algorithm confidence (High / Low) and credibility (High / Low). In each heatmap, the x-axis represents the algorithm confidence, the y-axis represents the algorithm credibility, and each region represents the percentage of samples.

To further analyze the impact of problem space constraints, we take into consideration the cases of constraining the trigger in problem space. In this case, about half of the families show no increase in attack success rates, indicating that a straightforward combination of poisoning and adversarial example attacks does not necessarily enhance evasion effectiveness. This can be attributed to the discrete nature of the feature spaces, where the adversarial example attacks may struggle to find an optimal combination of perturbations to form the trigger exactly.

Additionally, we explore the scenario where the trigger is restricted to the non-problem space. In this case, the injected trigger cannot be realized in the problem space, meaning that it will not introduce backdoor vulnerabilities in practice. In this setting, 3 out of 7 families show a significantly higher attack success rate, possibly due to the interplay between non-problem and problem space features. Conversely, 3 families exhibit a substantially lower attack success rate. This suggests the potential for defenders to design the defense utilizing the poisoning attack by injecting the trigger in the non-problem space to enhance the adversarial robustness of the model.

In summary, the poisoned model may not necessarily be more susceptible to adversarial examples in the malware domain, probably due to the problem space constraints and the discrete feature space. It is even possible for the defenders to design adversarial example defense methods employing poisoning attacks by leveraging the problem space constraints.

Table 7. Detection performance of the selected MD-based MD systems and the number of adversarial malware examples successfully generated by AdvDroidZero under different query budgets.

Methods	Performance		Query Budget		
	TPR	F1	10	20	30
Drebin	0.91	0.93	502	586	600
APIGraph	0.91	0.92	392	503	551

8.2 Connections between Evasion and Concept Drift Detections

Recall that concept drift detection, which aims to identify the distribution shift in the input data, is strongly similar to the detection of adversarial examples. However, whether these methods can be unified is in question. Therefore, this subsection studies the interaction between evasion attacks and concept drift detection in ML-based MD systems, particularly whether concept drift detection methods can detect adversarial malware examples. We choose TRANSCENDENT [19] as the concept drift detection algorithm and AdvDroidZero [76] as the adversarial example attack algorithm, due to their SOTA performance and practical applicability.

TRANSCENDENT [19] leverages conformal prediction theory to assess each test sample, offering two key metrics: algorithm credibility and algorithm confidence. It then determines thresholds for these metrics, considering samples with metrics below these thresholds as having low algorithm credibility or algorithm confidence. As TRANSCENDENT rejects concept drift samples with low algorithm credibility or low algorithm confidence, this policy requires adaptation for effective adversarial malware detection because it involves a broader range of cases, such as samples being similar to more classes. To adapt it to detect adversarial malware examples, we only consider the samples with both low algorithm credibility and low algorithm confidence as the detected adversarial malware example, indicating the sample seems to be more similar to another label. We re-implement the three conformal evaluators: approximate Transductive Conformal Evaluator (approx-TCE), Inductive Conformal Evaluator (ICE), and Cross-Conformal Evaluator (CCE) proposed in the TRANSCENDENT and integrate them into our evaluation framework. Furthermore, we use different thresholds based on quartiles and consider only the correct samples to compute the threshold. Notably, our implementation follows the latest version of TRANSCENDENT¹, which rectifies the data snooping error [12].

For generating adversarial malware examples, we select Drebin [13] and APIGraph [210] as target ML-based MD systems due to their various kinds of feature types, including syntax features and semantic features. To ensure fidelity to their original implementations, we strictly follow the configurations provided in their respective publications. Therefore, we employ SVM with the linear kernel as the target classifier, which is consistent with the previous work [76]. To simulate a real-world ML-based MD system, we utilize the time-aware split for training. The detection performance of these methods is shown in Table 7. For instance, the TPR values of Drebin and APIGraph are 0.914 and 0.912, respectively, indicating high detection effectiveness. Then, we randomly select 1,000 malware samples that can be successfully identified by the selected methods and employ AdvDroidZero with different query budgets to generate adversarial malware examples. The actual number of successfully generated adversarial malware examples is presented in Table 7. Subsequently, the three conformal evaluators are employed to detect these adversarial examples.

As depicted in Figure 2, TRANSCENDENT exhibits high true positive rates across various settings, confirming its effectiveness in identifying adversarial malware examples. This success is likely due to the adversarial generation process, which typically targets vulnerable features, reducing the confidence in the malware label. These features differ

¹The version can be accessed via the GitHub: <https://github.com/s2labres/transcendent-release>.

from the features that make the malware samples similar to samples in the training set. Therefore, TRANSCENDENT can be effective in detecting adversarial malware examples. Furthermore, the CCE achieves the most strong detection effectiveness among the three conformal evaluators, which achieves nearly 100% of the true positive rate. This may be attributed to the ensemble design in the CCE increasing the difficulty for the adversarial malware examples. In practice, the false positive rate, influenced by true concept drift samples, is also a critical consideration. As the false positive rate is introduced because of the true concept drift samples, the concept drift detection methods can be deployed in the initial days when there are fewer concept drift samples. It also opens up the future design to adapt concept drift detection methods in adversarial malware detection by distinguishing the adversarial malware examples and the true concept drift samples.

In summary, concept drift detection methods in the malware domain show high true positive rates in detecting adversarial malware examples. To minimize false positives caused by real concept drift samples, it is advisable to deploy these detection methods early on, reducing the likelihood of encountering such samples.

9 Future Directions

Building upon our stage-based taxonomy, we have identified significant technical advancements and limitations in current attack and defense proposals about the security risks of ML-based MD systems. However, our exploration is merely the beginning of a more extensive journey. This section outlines potential future research directions, aiming to deepen our understanding of security risks and develop secure and reliable ML-based MD systems. We propose these directions within our stage-based framework, dividing them into two primary categories: inter-stage future directions and intra-stage future directions. We remark that these directions are by no means exhaustive and only serve as a starting point for future research.

9.1 Inter-Stage Future Directions

Understanding the Attack Connections. In all current attack proposals within ML-based MD systems, a prevailing assumption is that a singular attack vector is in play [76, 143, 202]. However, this simplification might not accurately capture the complexity of real-world threats, where attackers are frequently motivated to amplify their impact by simultaneously deploying multiple types of attacks. For instance, the IMC attack [139] leverages both input perturbation and model poisoning techniques to enhance its attack effectiveness in the image domain. This example is an inspiration to investigate the potential for similar multifaceted attacks within the malware domain, as showcased in Section 8.1. Consequently, there is a clear opportunity for further research to deepen our understanding of stronger attacks that combine multiple attack vectors.

Holistic Defense Mechanism. As shown in Table 4, each stage of ML-based MD systems exhibits susceptibility to specific attack types, with some attacks not having countering defenses at the same stage. Furthermore, in practice, defenders aim to secure ML-based MD systems against all potential security threats, requiring that the defense mechanism to incorporate defense methods across different stages against various attacks. Additionally, minimizing the deployment costs of such defenses is a crucial consideration. Therefore, one promising direction is the development of a holistic defense mechanism. Such approach would entail a dynamic incorporation of existing defense methods tailored to address the full range of potential attacks across different stages of ML-based MD systems.

9.2 Intra-Stage Future Directions

Robust Dataset Preprocessing Method. In ML-based MD systems, the availability of massive software datasets from open-source platforms presents both opportunities and challenges. The constraints of computational cost and labeling demand necessitate efficient dataset preprocessing methods. Additionally, these methods must accurately reflect the real-world temporal and spatial distribution of malware data to avoid experimental bias. Thus, preprocessing methods that are robust against both time and space constraints are required yet missing. Moreover, open-source datasets are potentially vulnerable to poisoning attacks. Currently, such attacks often overlook the impact of dataset preprocessing methods. Evaluating the effectiveness of poisoning attacks in the context of robust preprocessing will provide deeper insights into the practical availability risks associated with ML-based MD systems. This understanding is instrumental in developing and refining preprocessing methods to mitigate such risks effectively. This direction would enhance the robustness of ML-based MD systems against data poisoning and ensure their effectiveness and reliability in real-world dynamic scenarios.

Practical Problem Space Solutions. To understand the semantics of software data, ML-based MD systems rely on program analysis tools to map the input data from the problem space to the feature space. From the attack side, the attackers need to perform problem space modification to conduct practical attacks. While there are some existing problem space modification solutions, they have many restrictions, such as the dependency on try-catch mechanisms, thereby constraining their ability to fully exploit problem space vulnerabilities. This limitation highlights the need for more sophisticated and flexible problem space modification techniques that can uncover and leverage a broader range of vulnerabilities. From the defensive perspective, there are inherent challenges in fully understanding the program semantics of malware data, often due to limitations in program analysis tools. For instance, static analysis can suffer from the path explosion problem, hindering comprehensive analysis. Additionally, the diverse nature of malware necessitates various program analysis methods for effective detection across different scenarios. As such, designing a better mapping from the problem space to the feature space would improve the detection effectiveness and robustness of ML-based MD systems.

Reliable Model Building Methods. As discussed in Section 6, attacks at Stage 3 of the ML-based MD systems remain largely unexplored. Drawing parallels from the image domain [17], one potential security risk in the malware domain involves attackers injecting vulnerabilities into ML models through open-source ML libraries. This emerging threat underscores the need to explore such security risks within the malware domain. Beyond awareness of code reliability, most of the existing robust model architectures and model training methods lack provable robustness guarantees. Although certified training methods have started to gain interest in the malware domain [37, 65, 80], they focus on limited input specifications, e.g., norm-based input constraints, while the malware domain requires more sophisticated specifications due to diverse discrete feature spaces and different semantic constraints. Future research may focus on developing novel metrics for constraints more tailored to the malware domain, alongside enhancing the adaptability and applicability of certified training methods.

Unified Anomaly Input Detection. At Stage 4 of ML-based MD systems, the defender can potentially unify concept drift detection methods and adversarial malware samples detection, as showcased in Section 8.2. Therefore, this raises a critical challenge for adversarial example attacks, which must now evade both malware detection and concept drift detection. This complexity also necessitates a more thorough exploration of the efficacy of concept drift detection methods in identifying adversarial malware, which can be pivotal in enabling defenders to develop more effective methods to detect adversarial malware. Moreover, the scope of anomaly detection may be further extended to include other

types of abnormal inputs, such as backdoor input samples. Such possibility calls for development of a comprehensive, unified anomaly input sample detection method. By integrating different forms of anomaly detection into a unified framework, defenders can significantly improve the utility and robustness of their security measures.

10 Related Surveys

Adversarial ML has recently gathered significant attention in various domains, such as image, text, audio, etc. In the malware detection domain, prior surveys [48, 98, 107, 120, 200] summarize the existing attack and defense proposals in ML-based MD systems based their assumptions and methodologies. Maiorca *et al.* [120] categorized the threats specifically targeted against ML-based PDF malware detectors based on the threat model. Yan *et al.* [200] summarized the existing adversarial example attack and defense methods for ML-based MD methods based on the approaches. Li *et al.* [98] proposed a concept systematization framework based on the attributes of threat models. Ling *et al.* [107] surveyed the adversarial example attack methods on ML-based Windows PE malware detection methods based on the adversary knowledge.

However, these surveys do not provide a holistic and comprehensive security analysis framework from the practical system development viewpoint to understand the landscape and new potential attacks and defense of ML-based MD systems. For instance, they may concentrate on one kind of threat, e.g., adversarial example attacks [107, 200]. Although such concentration may help to design better adversarial example attack methods, it hardly helps to understand connections between security risks, e.g., the connections between attacks. Additionally, they may not stand in the viewpoint of the ML-based MD system pipeline, which limits their practical implications.

11 Conclusion

This paper provides a stage-based taxonomy for a holistic understanding of the security risks of ML-based MD systems. The stage-based taxonomy is derived by dissecting an ML-based MD system into its operational stages. Leveraging the stage-based taxonomy, we summarize the technical progress of the related attacks and defenses with their limitations in each stage. In addition, we provide two case studies from the perspective of inter-stage and intra-stage with empirical insights. Furthermore, based on the taxonomy and analyses, we discuss the potential inter-stage and intra-stage future directions.

References

- [1] Yousra Aafer, Wenliang Du, and Heng Yin. 2013. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. In *SecureComm*.
- [2] Zainab Abaid, Mohamed Ali Káafar, and Sanjay Jha. 2017. Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers. In *NCA*. <https://doi.org/10.1109/NCA.2017.8171381>
- [3] Vitor Monte Afonso, Anatoli Kalysch, Tilo Müller, Daniela Oliveira, André Grégio, and Paulo Lício de Geus. 2018. Lumus: Dynamically Uncovering Evasive Android Applications. In *ISC*.
- [4] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When Malware is Packin' Heat: Limits of Machine Learning Classifiers Based on Static Analysis Features. In *NDSS*.
- [5] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. 2018. Adversarial Deep Learning for Robust Detection of Binary Encoded Malware. In *IEEE S&P Workshops*.
- [6] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: collecting millions of Android apps for the research community. In *MSR*.
- [7] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. *CoRR* (2018).
- [8] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *CoRR* (2018).

- [9] Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, and Alessio Merlo. 2020. Obfuscapk: An open-source black-box obfuscation tool for Android apps. *SoftwareX* (2020).
- [10] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. 2023. Humans vs. machines in malware classification. *USENIX Security Symposium* (2023).
- [11] Giovanni Apruzzese, Hyrum S. Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin A. Roundy. 2023. "Real Attackers Don't Compute Gradients": Bridging the Gap Between Adversarial ML Research and Practice. *IEEE SaTML* (2023).
- [12] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *USENIX Security Symposium*.
- [13] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *NDSS*.
- [14] AV-ATLAS. 2025. The Amount of Malware. <https://portal.av-atlas.org/malware/statistics>. [Accessed on March 10, 2025].
- [15] Avast. 2025. Avast Blog. <https://www.avast.com/technology/ai-and-machine-learning#pc>. [Accessed on March 10, 2025].
- [16] Avira. 2025. Avira White Paper. <https://oem.avira.com/en/technology/machine-learning>. [Accessed on March 10, 2025].
- [17] Eugene Bagdasaryan and Vitaly Shmatikov. 2021. Blind backdoors in deep learning models. In *USENIX Security Symposium*.
- [18] Stefan Balauca, Mark Niklas Müller, Yuhao Mao, Maximilian Baader, Marc Fischer, and Martin T. Vechev. 2024. Overcoming the Paradox of Certified Training with Gaussian Smoothing. *CoRR* (2024).
- [19] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending TRANSCEND: Revisiting Malware Classification in the Presence of Concept Drift. In *IEEE S&P*.
- [20] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srdic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2017. Evasion Attacks against Machine Learning at Test Time. *CoRR* (2017).
- [21] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2014. Poisoning behavioral malware clustering. In *AISec*.
- [22] Hamid Bostani and Veelasha Moonsamy. 2021. EvadeDroid: A Practical Evasion Attack on Machine Learning for Black-box Android Malware Detection. *CoRR* (2021).
- [23] Zhenquan Cai and Roland H. C. Yap. 2016. Inferring the Detection Logic and Evaluating the Effectiveness of Android Anti-Virus Apps. In *CODASPY*.
- [24] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. 2019. On Evaluating Adversarial Robustness. *CoRR* (2019).
- [25] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE S&P*.
- [26] Raphael Labaca Castro, Luis Muñoz-González, Feargus Wendlebury, Gabi Dreo Rodosek, Fabio Pierazzi, and Lorenzo Cavallaro. 2021. Universal Adversarial Perturbations for Malware. *CoRR* (2021).
- [27] Dewan Chaulagain, Prabesh Poudel, Prabesh Pathak, Sankardas Roy, Doina Caragea, Guojun Liu, and Ximeng Ou. 2020. Hybrid Analysis of Android Apps for Security Vetting using Deep Learning. In *CNS*.
- [28] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. 2018. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. *CoRR* (2018).
- [29] Lingwei Chen, Shifu Hou, and Yanfang Ye. 2017. SecureDroid: Enhancing Security of Machine Learning-based Detection against Adversarial Android Malware Attacks. In *ACSAC*.
- [30] Lingwei Chen, Shifu Hou, Yanfang Ye, and Shouhuai Xu. 2018. DroidEye: Fortifying Security of Learning-Based Classifier Against Adversarial Android Malware Attacks. In *ASONAM*.
- [31] Lingwei Chen and Yanfang Ye. 2017. SecMD: Make Machine Learning More Secure Against Adversarial Malware Attacks. In *Australasian Conference on Artificial Intelligence*.
- [32] Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. 2017. Adversarial Machine Learning in Malware Detection: Arms Race between Evasion Attack and Defense. In *EISIC*.
- [33] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Comput. Secur.* (2018).
- [34] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. 2020. Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection. *IEEE Trans. Inf. Forensics Secur.* (2020).
- [35] Yizheng Chen, Zhoujie Ding, and David A. Wagner. 2023. Continuous Learning for Android Malware Detection. In *USENIX Security Symposium*.
- [36] Yi-Hsien Chen, Si-Chen Lin, Szu-Chun Huang, Chin-Laung Lei, and Chun-Ying Huang. 2023. Guided Malware Sample Analysis Based on Graph Neural Networks. *IEEE Trans. Inf. Forensics Secur.* 18 (2023), 4128–4143.
- [37] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. 2020. On Training Robust PDF Malware Classifiers. In *USENIX Security Symposium*.
- [38] Binlin Cheng, Erika A. Leal, Haotian Zhang, and Jiang Ming. 2023. On the Feasibility of Malware Unpacking via Hardware-assisted Loop Profiling. In *USENIX Security Symposium*.
- [39] Binlin Cheng, Jiang Ming, Erika A. Leal, Haotian Zhang, Jianming Fu, Guojun Peng, and Jean-Yves Marion. 2021. Obfuscation-Resilient Executable Payload Extraction From Packed Malware. In *USENIX Security Symposium*.
- [40] Mihai Christodorescu, Somesh Jha, Douglas Maughan, Dawn Song, and Cliff Wang (Eds.). 2007. *Malware Detection*.
- [41] Jeremy Cohen, Elan Rosenfeld, and J. Zico Kolter. 2019. Certified Adversarial Robustness via Randomized Smoothing. In *ICML*.

- [42] Kevin Coogan, Gen Lu, and Saumya K. Debray. 2011. Deobfuscation of virtualization-obfuscated software: a semantics-based approach. In *ACM CCS*.
- [43] Francesco Croce and Matthias Hein. 2020. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*.
- [44] Savino Dambra, Yufei Han, Simone Aonzo, Platon Kotzias, Antonino Vitale, Juan Caballero, Davide Balzarotti, and Leyla Bilge. 2023. Decoding the Secrets of Machine Learning in Malware Classification: A Deep Dive into Datasets, Feature Extraction, and Model Performance. In *ACM CCS*.
- [45] Hung Dang, Yue Huang, and Ee-Chien Chang. 2017. Evading Classifiers by Morphing in the Dark. In *CCS*.
- [46] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2019. Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries. In *ITASEC*.
- [47] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware. *IEEE Trans. Inf. Forensics Secur.* (2021).
- [48] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial EXEMples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection. *ACM Trans. Priv. Secur.* (2021).
- [49] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2019. Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. *IEEE Trans. Dependable Secur. Comput.* (2019).
- [50] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In *USENIX Security Symposium*.
- [51] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*.
- [52] Artem Dinaburg, Paul Royal, Monirul Islam Sharif, and Wenke Lee. 2008. Ether: malware analysis via hardware virtualization extensions. In *ACM CCS*.
- [53] Tianyu Du, Shouling Ji, Lujia Shen, Yao Zhang, Jinfeng Li, Jie Shi, Chengfang Fang, Jianwei Yin, Raheem Beyah, and Ting Wang. 2021. Cert-RNN: Towards Certifying the Robustness of Recurrent Neural Networks. In *ACM CCS*.
- [54] Kevin Eykholt, Taesung Lee, Douglas Lee Schales, Jiyong Jang, Ian M. Molloy, and Masha Zorin. 2023. URET: Universal Robustness Evaluation Toolkit (for Evasion). In *USENIX Security Symposium*.
- [55] Wenbo Fang, Junjiang He, Wenshan Li, Xiaolong Lan, Yang Chen, Tao Li, Jiwu Huang, and Linlin Zhang. 2023. Comprehensive Android Malware Detection Based on Federated Learning Architecture. *IEEE Trans. Inf. Forensics Secur.* (2023).
- [56] Ruitao Feng, Sen Chen, Xiaofei Xie, Guozhu Meng, Shang-Wei Lin, and Yang Liu. 2021. A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. *IEEE Trans. Inf. Forensics Secur.* (2021).
- [57] William Fleshman, Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. 2018. Non-Negative Networks Against Adversarial Attacks. *CoRR* (2018).
- [58] Marek Galovic, Branislav Bosanský, and Viliam Lisý. 2021. Improving Robustness of Malware Classifiers using Adversarial Strings Generated from Perturbed Latent Representations. *CoRR* (2021).
- [59] Yuyou Gan, Yuhao Mao, Xuhong Zhang, Shouling Ji, Yuwen Pu, Meng Han, Jianwei Yin, and Ting Wang. 2022. "Is your explanation stable?": A Robustness Evaluation Framework for Feature Attribution. In *ACM CCS*.
- [60] Cuiying Gao, Minghui Cai, Shuijun Yin, Gaozhun Huang, Heng Li, Wei Yuan, and Xiapu Luo. 2023. Obfuscation-Resilient Android Malware Analysis Based on Complementary Features. *IEEE Trans. Inf. Forensics Secur.* (2023).
- [61] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Chinthana Ranasinghe, and Surya Nepal. 2019. STRIP: a defence against trojan attacks on deep neural networks. In *ACSAC*.
- [62] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware. *ACM Trans. Softw. Eng. Methodol.* (2018).
- [63] Timon Gehr, Matthew Mirman, Dana Drachler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *IEEE S&P*.
- [64] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* (2020).
- [65] Daniel Gibert, Giulio Zizzo, and Quan Le. 2023. Certified Robustness of Static Deep Learning-based Malware Detectors against Patch and Append Attacks. In *AISeC*.
- [66] Jiacheng Gong, Weina Niu, Song Li, Mingxue Zhang, and Xiaosong Zhang. 2024. Sensitive Behavioral Chain-Focused Android Malware Detection Fused With AST Semantics. *IEEE Trans. Inf. Forensics Secur.* (2024).
- [67] Liangyi Gong, Zhenhua Li, Feng Qian, Zifan Zhang, Qi Alfred Chen, Zhiyun Qian, Hao Lin, and Yunhao Liu. 2020. Experiences of landing machine learning onto market-scale mobile malware detection. In *EuroSys*.
- [68] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*.
- [69] Google Play. 2025. Google Play. <https://play.google.com/>. [Accessed on March 10, 2025].
- [70] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2016. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *CoRR* (2016).
- [71] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2017. Adversarial Examples for Malware Detection. In *ESORICS*.

- [72] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. LEMNA: Explaining Deep Learning based Security Applications. In *ACM CCS*.
- [73] Mahmoud Hammad, Joshua Garcia, and Sam Malek. 2018. A large-scale empirical study on the effects of code obfuscations on Android apps and anti-malware products. In *ICSE*.
- [74] Qian Han, V. S. Subrahmanian, and Yanhai Xiong. 2020. Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations. *IEEE Trans. Inf. Forensics Secur.* (2020).
- [75] Richard E. Harang and Ethan M. Rudd. 2020. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. *CoRR* (2020).
- [76] Ping He, Yifan Xia, Xuhong Zhang, and Shouling Ji. 2023. Efficient Query-Based Attack against ML-Based Android Malware Detection under Zero Knowledge Setting. In *ACM CCS*.
- [77] Yiling He, Yiping Li, Lei Wu, Ziqi Yang, Kui Ren, and Zhan Qin. 2023. MsDroid: Identifying Malicious Snippets for Android Malware Detection. *IEEE Trans. Dependable Secur. Comput.* (2023).
- [78] Péter Horváth, Dirk Lauret, Zhuoran Liu, and Lejla Batina. 2024. SoK: Neural Network Extraction Through Physical Side Channels. In *USENIX Security Symposium*.
- [79] Shifu Hou, Yujie Fan, Yiming Zhang, Yanfang Ye, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, and Fudong Shao. 2019. α Cyber: Enhancing Robustness of Android Malware Detection System against Adversarial Attacks on Heterogeneous Graph based Model. In *CKM*.
- [80] Zhuoqun Huang, Neil G Marchant, Keane Lucas, Lujo Bauer, Olga Ohrimenko, and Benjamin IP Rubinstein. 2023. RS-del: Edit distance robustness certificates for sequence classifiers via randomized deletion. In *NeurIPS*.
- [81] Inigo Incer, Michael Theodorides, Sadia Afroz, and David A. Wagner. 2018. Adversarially Robust Malware Detection Using Monotonic Classification. In *IWSPA@CODASPY*.
- [82] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High Accuracy and High Fidelity Extraction of Neural Networks. In *USENIX Security Symposium*.
- [83] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. 2019. Neurlux: dynamic malware analysis without feature engineering. In *ACSAC*.
- [84] Roberto Jordaney, Kumar Sharad, Santanu Kumar Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *USENIX Security Symposium*.
- [85] Kaspersky. 2025. Kaspersky Lab White Paper. <https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf>. [Accessed on March 10, 2025].
- [86] Staffs Keele et al. 2007. Guidelines for performing systematic literature reviews in software engineering.
- [87] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. 2019. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE Trans. Inf. Forensics Secur.* (2019).
- [88] Dhilung Kirat and Giovanni Vigna. 2015. MalGene: Automatic Extraction of Malware Analysis Evasion Signature. In *ACM CCS*.
- [89] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *EUSIPCO*.
- [90] Brian Kondracki, Babak Amin Azad, Najmeh Miramirkhani, and Nick Nikiforakis. 2022. The Droid is in the Details: Environment-aware Evasion of Android Sandboxes. In *NDSS*.
- [91] Yunus Kucuk and Guanhua Yan. 2020. Deceiving Portable Executable Malware Classifiers into Targeted Misclassification with Practical Adversarial Examples. In *CODASPY*.
- [92] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998).
- [93] Chaoran Li, Xiao Chen, Derui Wang, Sheng Wen, Muhammad Ejaz Ahmed, Seyit Camtepe, and Yang Xiang. 2022. Backdoor Attack on Machine Learning Based Android Malware Detectors. *IEEE Trans. Dependable Secur. Comput.* (2022).
- [94] Deqiang Li, Ramesh Baral, Tao Li, Han Wang, Qianmu Li, and Shouhuai Xu. 2018. HashTran-DNN: A Framework for Enhancing Robustness of Deep Neural Networks against Adversarial Malware Samples. *CoRR* (2018).
- [95] Deqiang Li, Shicheng Cui, Yun Li, Jia Xu, Fu Xiao, and Shouhuai Xu. 2023. PAD: Towards Principled Adversarial Malware Detection Against Evasion Attacks. *CoRR* (2023).
- [96] Deqiang Li and Qianmu Li. 2020. Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection. *IEEE Trans. Inf. Forensics Secur.* (2020).
- [97] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2021. A Framework for Enhancing Deep Neural Networks Against Adversarial Malware. *IEEE Trans. Netw. Sci. Eng.* (2021).
- [98] Deqiang Li, Qianmu Li, Yanfang (Fanny) Ye, and Shouhuai Xu. 2023. Arms Race in Adversarial Malware Detection: A Survey. *ACM Comput. Surv.* (2023).
- [99] Deqiang Li, Tian Qiu, Shuo Chen, Qianmu Li, and Shouhuai Xu. 2021. Can We Leverage Predictive Uncertainty to Detect Dataset Shift and Adversarial Examples in Android Malware Detection?. In *ACSAC*.
- [100] Heng Li, Zhang Cheng, Bang Wu, Liheng Yuan, Cuiying Gao, Wei Yuan, and Xiapu Luo. 2023. Black-box Adversarial Example Attack towards FCG Based Android Malware Detection under Incomplete Feature Information. In *USENIX Security Symposium*.
- [101] Heng Li, ShiYao Zhou, Wei Yuan, Jiahuan Li, and Henry Leung. 2020. Adversarial-Example Attacks Toward Android Malware Detection System. *IEEE Syst. J.* (2020).

- [102] Heng Li, ShiYao Zhou, Wei Yuan, Xiapu Luo, Cuiying Gao, and Shuiyan Chen. 2021. Robust Android Malware Detection against Adversarial Example Attacks. In *WWW*.
- [103] Xintong Li and Qi Li. 2021. An IRL-based malware adversarial generation method to evade anti-malware engines. *Comput. Secur.* (2021).
- [104] Yakang Li, Yikun Hu, Yizhuo Wang, Yituo He, Haining Lu, and Dawu Gu. 2023. RGDroid: Detecting Android Malware with Graph Convolutional Networks against Structural Attacks. In *SANER*.
- [105] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. 2015. MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis. In *COMPSAC*.
- [106] Xiang Ling, Lingfei Wu, Wei Deng, Zhenqing Qu, Jiangyu Zhang, Sheng Zhang, Tengfei Ma, Bin Wang, Chunming Wu, and Shouling Ji. 2022. MalGraph: Hierarchical Graph Neural Networks for Robust Windows Malware Detection. In *INFORCOM*.
- [107] Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Yaguan Qian, Chunming Wu, Shouling Ji, Tianyue Luo, Jingzheng Wu, and Yanjun Wu. 2023. Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art. *Comput. Secur.* (2023).
- [108] Xiang Ling, Zhiyu Wu, Bin Wang, Wei Deng, Jingzheng Wu, Shouling Ji, Tianyue Luo, and Yanjun Wu. 2024. A Wolf in Sheep's Clothing: Practical Black-box Adversarial Attacks for Evading Learning-based Windows Malware Detection in the Wild. In *USENIX Security Symposium*.
- [109] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *ICDM*.
- [110] Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. 2020. A novel method for malware detection on ML-based visualization technique. *Comput. Secur.* (2020).
- [111] Xinbo Liu, Jiliang Zhang, Yaping Lin, and He Li. 2019. ATMPA: attacking machine learning-based malware visualization detection methods via adversarial examples. In *IWQoS*.
- [112] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. 2023. Deep Learning for Android Malware Defenses: A Systematic Literature Review. *ACM Comput. Surv.* (2023).
- [113] Keane Lucas, Samruddhi Pai, Weiran Lin, Lujo Bauer, Michael K. Reiter, and Mahmood Sharif. 2023. Adversarial Training for Raw-Binary Malware Classifiers. In *USENIX Security Symposium*.
- [114] Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K. Reiter, and Saurabh Shintre. 2021. Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes. In *ACM AsiaCCS*.
- [115] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NeurIPS*.
- [116] Peizhuo Lv, Chang Yue, Ruigang Liang, Yunfei Yang, Shengzhi Zhang, Hualong Ma, and Kai Chen. 2023. A Data-free Backdoor Injection Approach in Neural Networks. In *USENIX Security Symposium*.
- [117] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL*.
- [118] Aravind Machiry, Nilo Redini, Eric Gustafson, Yanick Fratantonio, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. 2018. Using Loops For Malware Classification Resilient to Feature-unaware Perturbations. In *ACSAC*.
- [119] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. 2015. Stealth attacks: An extended insight into the obfuscation effects on Android malware. *Comput. Secur.* (2015).
- [120] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. 2019. Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks. *ACM Comput. Surv.* (2019).
- [121] Alessandro Mantovani, Simone Aonzo, Xabier Ugarte-Pedrero, Alessio Merlo, and Davide Balzarotti. 2020. Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem. In *NDSS*.
- [122] Yuhao Mao, Stefan Balauca, and Martin T. Vechev. 2024. CTBENCH: A Library and Benchmark for Certified Training. *CoRR* (2024).
- [123] Yuhao Mao, Chong Fu, Saizhuo Wang, Shouling Ji, Xuhong Zhang, Zhenguang Liu, Jun Zhou, Alex X. Liu, Raheem Beyah, and Ting Wang. 2022. Transfer Attacks Revisited: A Large-Scale Empirical Study in Real Computer Vision Settings. In *IEEE S&P*.
- [124] Yuhao Mao, Mark Niklas Mueller, Marc Fischer, and Martin Vechev. 2023. Connecting Certified and Adversarial Training. In *NeurIPS*.
- [125] Yuhao Mao, Mark Niklas Müller, Marc Fischer, and Martin T. Vechev. 2024. Understanding Certified Training with Interval Bound Propagation. In *ICLR*.
- [126] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J. Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *NDSS*.
- [127] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. 2007. OmniUnpack: Fast, Generic, and Safe Unpacking of Malware. In *ACSAC*.
- [128] Guozhu Meng, Yinxing Xue, Mahinthan Chandramohan, Annamalai Narayanan, Yang Liu, Jie Zhang, and Tieming Chen. 2016. Mystique: Evolving Android Malware for Auditing Anti-Malware Tools. In *AsiaCCS*.
- [129] Zhaoyi Meng, Jiale Zhang, Jiaqi Guo, Wansen Wang, Wenchao Huang, Jie Cui, Hong Zhong, and Yan Xiong. 2025. Detecting Android Malware by Visualizing App Behaviors From Multiple Complementary Views. *IEEE Trans. Inf. Forensics Secur.* (2025), 2915–2929.
- [130] Jaron Mink, Hadjer Benkraouda, Limin Yang, Arridhana Ciptadi, Ali Ahmadzadeh, Daniel Votipka, and Gang Wang. 2023. Everybody's Got ML, Tell Me What Else You Have: Practitioners' Perception of ML-Based Security Tools and Explanations. In *IEEE S&P*.
- [131] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. 2017. Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts. In *IEEE S&P*.
- [132] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of Static Analysis for Malware Detection. In *ACSAC*.
- [133] Mark Niklas Mueller, Franziska Eckert, Marc Fischer, and Martin Vechev. 2023. Certified Training: Small Boxes are All You Need. In *ICLR*.

- [134] Ritwik Murali, T. Palanisamy, and C. Shunmuga Velayutham. 2023. Evolving malware variants as antigens for antivirus systems. *Expert Syst. Appl.* (2023).
- [135] Ritwik Murali and C. Shunmuga Velayutham. 2022. Adapting novelty towards generating antigens for antivirus systems. In *ACM GECCO*.
- [136] Trivikram Muralidharan, Aviad Cohen, Noa Gerson, and Nir Nissim. 2023. File Packing from the Malware Perspective: Techniques, Analysis Approaches, and Directions for Enhancements. *ACM Comput. Surv.* (2023).
- [137] Tushar Nayan, Qiming Guo, Mohammed Alduniawi, Marcus Botacin, A. Sencuk Uluagac, and Ruimin Sun. 2024. SoK: All You Need to Know About On-Device ML Model Extraction - The Gap Between Research and Practice. In *USENIX Security Symposium*.
- [138] Alessandro De Palma, Rudy Bunel, Krishnamurthy (Dj) Dvijotham, M. Pawan Kumar, Robert Stanforth, and Alessio Lomuscio. 2024. Expressive Losses for Verified Robustness via Convex Combinations. In *ICLR*.
- [139] Ren Pang, Hua Shen, Xinyang Zhang, Shouling Ji, Yevgeniy Vorobeychik, Xiapu Luo, Alex X. Liu, and Ting Wang. 2020. A Tale of Evil Twins: Adversarial Inputs versus Poisoned Models. In *ACM CCS*.
- [140] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *IEEE EuroS&P*.
- [141] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.
- [142] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordane, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *USENIX Security Symposium*.
- [143] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *IEEE S&P*.
- [144] Robert Podschwadt and Hassan Takabi. 2019. On Effectiveness of Adversarial Examples and Defenses for Malware Classification. In *SecureComm*.
- [145] Xiangyu Qi, Tinghao Xie, Jiachen T. Wang, Tong Wu, Saeed Mahloujifar, and Prateek Mittal. 2023. Towards A Proactive ML Approach for Detecting Backdoor Poison Samples. In *USENIX Security Symposium*.
- [146] Dima Rabadi and Sin G. Teo. 2020. Advanced Windows Methods on Malware Detection and Classification. In *ACSAC*.
- [147] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. 2018. Malware Detection by Eating a Whole EXE. In *AAAI Workshops*.
- [148] Aqib Rashid and Jose M. Such. 2023. MalProtect: Stateful Defense Against Adversarial Query Attacks in ML-Based Malware Detection. *IEEE Trans. Inf. Forensics Secur.* (2023).
- [149] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2014. Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks. *IEEE Trans. Inf. Forensics Secur.* (2014).
- [150] Hemant Rathore, Animesh Sasan, Sanjay K. Sahay, and Mohit Sewak. 2022. Defending malware detection models against evasion based adversarial attacks. *Pattern Recognit. Lett.* (2022).
- [151] Hemant Rathore, Sujay C. Sharma, Sanjay K. Sahay, and Mohit Sewak. 2022. Are Malware Detection Classifiers Adversarially Vulnerable to Actor-Critic based Evasion Attacks? *EAI Endorsed Trans. Scalable Inf. Syst.* (2022).
- [152] Maria Rigaki and Sebastian Garcia. 2023. Stealing and evading malware classifiers and antivirus at low false positive conditions. *Comput. Secur.* (2023).
- [153] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft Malware Classification Challenge. *CoRR* (2018).
- [154] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. 2020. Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers. In *ACSAC*.
- [155] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2018. Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. In *RAID*.
- [156] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. 2012. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *IEEE S&P*.
- [157] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. 2006. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *ACSAC*.
- [158] Sevil Sen, Emre Aydogan, and Ahmet Ilhan Aysan. 2018. Coevolution of Mobile Malware and Anti-Malware. *IEEE Trans. Inf. Forensics Secur.* (2018).
- [159] Giorgio Severi, Jim Meyer, Scott E. Coull, and Alina Oprea. 2021. Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers. In *USENIX Security Symposium*.
- [160] Kamran Shaikat, Suhuai Luo, and Vijay Varadharajan. 2022. A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. *Eng. Appl. Artif. Intell.* (2022).
- [161] Yun Shen, Pierre-Antoine Vervier, and Gianluca Stringhini. 2022. A Large-scale Temporal Measurement of Android Malicious Apps: Persistence, Migration, and Lessons Learned. In *USENIX Security Symposium*.
- [162] Jagvir Singh and Jaswinder Singh. 2021. A survey on machine learning-based malware detection in executable files. *J. Syst. Archit.* (2021).
- [163] Charles Smutz and Angelos Stavrou. 2016. When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors. In *NDSS*.

- [164] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2022. MAB-Malware: A Reinforcement Learning Framework for Blackbox Generation of Adversarial Malware. In *ACM AsiaCCS*.
- [165] Nedim Srndic and Pavel Laskov. 2013. Detection of Malicious PDF Files Based on Hierarchical Document Structure. In *NDSS*.
- [166] Nedim Srndic and Pavel Laskov. 2014. Practical Evasion of a Learning-Based Classifier: A Case Study. In *IEEE S&P*.
- [167] Jack W. Stokes, De Wang, Mady Marinescu, Marc Marino, and Brian Bussone. 2017. Attack and Defense of Dynamic Analysis-Based, Adversarial Neural Malware Classification Models. *CoRR* (2017).
- [168] Octavian Suciuc, Scott E. Coull, and Jeffrey Johns. 2019. Exploring Adversarial Examples in Malware Detection. In *IEEE S&P Workshops*.
- [169] Octavian Suciuc, Radu Marginean, Yigitcan Kaya, Hal Daumé III, and Tudor Dumitras. 2018. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In *USENIX Security Symposium*.
- [170] Ruoxi Sun, Minhui Xue, Gareth Tyson, Tian Dong, Shaofeng Li, Shuo Wang, Haojin Zhu, Seyit Camtepe, and Surya Nepal. 2023. Mate! Are You Really Aware? An Explainability-Guided Testing Framework for Robustness of Malware Detectors. In *FSE*.
- [171] Jianwen Tian, Kefan Qiu, Debin Gao, Zhi Wang, Xiaohui Kuang, and Gang Zhao. 2023. Sparsity Brings Vulnerabilities: Exploring New Metrics in Backdoor Attacks. In *USENIX Security Symposium*.
- [172] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *USENIX Security Symposium*.
- [173] Ilias Tsingnopoulos, Jacopo Cortellazzi, Branislav Bosanský, Simone Aonzo, Davy Preuveneers, Wouter Joosen, Fabio Pierazzi, and Lorenzo Cavallaro. 2024. How to Train your Antivirus: RL-based Hardening through the Problem Space. In *RAID*.
- [174] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo García Bringas. 2015. SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers. In *IEEE S&P*.
- [175] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *ACM CCS*.
- [176] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Zheng Qin. 2020. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Comput. Secur.* (2020).
- [177] VirusShare. 2025. VirusShare. <https://virusshare.com/>. [Accessed on March 10, 2025].
- [178] VirusTotal. 2025. VirusTotal. <https://www.virustotal.com/gui/home/upload>. [Accessed on March 10, 2025].
- [179] VirusTotal. 2025. VirusTotal Graph. <https://www.virustotal.com/graph/>. [Accessed on March 10, 2025].
- [180] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *IEEE S&P*.
- [181] Huaijin Wang, Shuai Wang, Dongpeng Xu, Xiangyu Zhang, and Xiao Liu. 2022. Generating Effective Software Obfuscation Sequences With Reinforcement Learning. *IEEE Trans. Dependable Secur. Comput.* (2022).
- [182] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying Android Malware in the New Era. *Proc. ACM Meas. Anal. Comput. Syst.* (2022).
- [183] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia II, Xinyu Xing, Xue Liu, and C. Lee Giles. 2017. Adversary Resistant Deep Neural Networks with an Application to Malware Detection. In *KDD*.
- [184] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient Formal Safety Analysis of Neural Networks. In *NeurIPS*.
- [185] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *USENIX Security Symposium*.
- [186] Yuxin Wen, Leo Marchyok, Sanghyun Hong, Jonas Geiping, Tom Goldstein, and Nicholas Carlini. 2024. Privacy Backdoors: Enhancing Membership Inference through Poisoning Pre-trained Models. *CoRR* (2024).
- [187] Cheng-Hsin Weng, Yan-Ting Lee, and Shan-Hung Wu. 2020. On the Trade-off between Adversarial and Backdoor Robustness. In *NeurIPS*.
- [188] Wikipedia. 2025. Creeper Virus. https://en.wikipedia.org/wiki/Creeper_and_Reaper. [Accessed on March 10, 2025].
- [189] Wikipedia. 2025. Kerckhoffs' Principle. https://en.wikipedia.org/wiki/Kerckhoffs%27s_principle. [Accessed on March 10, 2025].
- [190] Christian Wressnegger, Kevin Freeman, Fabian Yamaguchi, and Konrad Rieck. 2017. Automatically Inferring Malware Signatures for Anti-Virus Assisted Attacks. In *Asia CCS*.
- [191] Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R. Lyu. 2021. Why an Android App Is Classified as Malware: Toward Malware Classification Interpretation. *ACM Trans. Softw. Eng. Methodol.* (2021).
- [192] Guangquan Xu, Guohua Xin, Litao Jiao, Jian Liu, Shaoying Liu, Meiqi Feng, and Xi Zheng. 2021. OFEI: A Semi-black-box Android Adversarial Sample Attack Framework Against DLaaS. *CoRR* (2021).
- [193] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In *NDSS*.
- [194] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. 2021. Detecting AI Trojans Using Meta Neural Analysis. In *IEEE S&P*.
- [195] Lei Xue, Xiapu Luo, Le Yu, Shuai Wang, and Dinghao Wu. 2017. Adaptive unpacking of Android apps. In *ICSE*.
- [196] Lei Xue, Hao Zhou, Xiapu Luo, Le Yu, Dinghao Wu, Yajin Zhou, and Xiaobo Ma. 2022. PackerGrind: An Adaptive Unpacking System for Android Apps. *IEEE Trans. Software Eng.* (2022).
- [197] Lei Xue, Hao Zhou, Xiapu Luo, Yajin Zhou, Yang Shi, Guofei Gu, Fengwei Zhang, and Man Ho Au. 2021. Happer: Unpacking Android Apps via a Hardware-Assisted Approach. In *IEEE S&P*.

- [198] Lei Xue, Yajin Zhou, Ting Chen, Xiapu Luo, and Guofei Gu. 2017. Malton: Towards On-Device Non-Invasive Mobile Malware Analysis for ART. In *USENIX Security Symposium*.
- [199] Yinxing Xue, Guozhu Meng, Yang Liu, Tian Huat Tan, Hongxu Chen, Jun Sun, and Jie Zhang. 2017. Auditing Anti-Malware Tools by Evolving Android Malware and Dynamic Loading Technique. *IEEE Trans. Inf. Forensics Secur.* (2017).
- [200] Senming Yan, Jing Ren, Wei Wang, Limin Sun, Wei Zhang, and Quan Yu. 2023. A Survey of Adversarial Attack and Defense Methods for Malware Classification in Cyber Security. *IEEE Commun. Surv. Tutorials* (2023).
- [201] Greg Yang, Tony Duan, J. Edward Hu, Hadi Salman, Ilya P. Razenshteyn, and Jerry Li. 2020. Randomized Smoothing of All Shapes and Sizes. In *ICML*.
- [202] Limin Yang, Zhi Chen, Jacopo Cortellazzi, Feargus Pendlebury, Kevin Tu, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. 2023. Jigsaw Puzzle: Selective Backdoor Attack to Subvert Malware Classifiers. In *IEEE S&P*.
- [203] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2021. BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware. In *IEEE S&P Workshops*.
- [204] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *USENIX Security Symposium*.
- [205] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. 2017. Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps. In *ACSAC*.
- [206] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. 2020. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. In *NDSS*.
- [207] Dazhi Zhan, Yexin Duan, Yue Hu, Weili Li, Shize Guo, and Zhisong Pan. 2024. MalPatch: Evading DNN-Based Malware Detection With Adversarial Patches. *IEEE Trans. Inf. Forensics Secur.* (2024).
- [208] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *NeurIPS*.
- [209] Lan Zhang, Peng Liu, Yoon-Ho Choi, and Ping Chen. 2023. Semantics-Preserving Reinforcement Learning Attack Against Graph Neural Networks for Malware Detection. *IEEE Trans. Dependable Secur. Comput.* (2023).
- [210] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In *ACM CCS*.
- [211] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *NeurIPS*.
- [212] Yunchun Zhang, Fan Feng, Zikun Liao, Zixuan Li, and Shaowen Yao. 2023. Universal backdoor attack on deep neural networks for malware detection. *Appl. Soft Comput.* (2023).
- [213] Yueqian Zhang, Xiapu Luo, and Haoyang Yin. 2015. DexHunter: Toward Extracting Hidden Code from Packed Android Applications. In *ESORICS*.
- [214] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. 2021. Structural Attack against Graph Based Android Malware Detection. In *ACM CCS*.
- [215] Jingnan Zheng, Jiahao Liu, An Zhang, Jun Zeng, Ziqi Yang, Zhenkai Liang, and Tat-Seng Chua. 2024. MaskDroid: Robust Android Malware Detection with Masked Graph Representations. In *ASE*.
- [216] Fangtian Zhong, Pengfei Hu, Guoming Zhang, Hong Li, and Xiuzhen Cheng. 2022. Reinforcement learning based adversarial malware example generation against black-box detectors. *Comput. Secur.* (2022).
- [217] Yuyang Zhou, Guang Cheng, Zongyao Chen, and Shui Yu. 2023. MalPurifier: Enhancing Android Malware Detection with Adversarial Purification against Evasion Attacks. *CoRR* (2023).
- [218] Yuyang Zhou, Guang Cheng, Shui Yu, Zongyao Chen, and Yujia Hu. 2024. MTDroid: A Moving Target Defense-Based Android Malware Detector Against Evasion Attacks. *IEEE Trans. Inf. Forensics Secur.* (2024).
- [219] Huijuan Zhu, Xilong Chen, Liangmin Wang, Zhicheng Xu, and Victor S. Sheng. 2024. A Dynamic Analysis-Powered Explanation Framework for Malware Detection. *IEEE Trans. Knowl. Data Eng.* (2024).