Dissecting Malicious Behaviours of Mobile Applications

Wei Wang

A thesis submitted for the degree of
MASTER OF PHILOSOPHY
The University of Adelaide

May 11, 2022

Contents

Abstract **ix**

Declaration of Authorship **xi**

Acknowledgements **xiii**

Achievements **xv**

1 Introduction **1**
1.1 Background 1
1.2 Common Notations 2
1.3 Thesis Contribution 2
1.4 Thesis Organization 3

2 Literature Survey **5**
2.1 Introduction 5
2.2 Existing Literature Surveys and Related Work 6
2.3 Machine Learning-based Malware Detection 7
2.3.1 Feature Extraction 7
Windows Portable Executable (WinPE) 7
Android Application Package (APK) 8
2.3.2 Malware Detection 8
Machine Learning Models 8
Deep Learning 9
Concept Drift 9
2.4 Explanation on Machine Learning 9
2.4.1 Local Explanation 9
2.4.2 Global Explanation 11
2.5 Attacks on Machine Learning 12
2.5.1 Backdoor Attack 12
2.5.2 Adversarial Evasion & Other Attacks 13
2.6 Conclusion 14

3 A Semi-Automated Assessment of Android Clipboards **15**
3.1 Introduction 18
3.2 Clipboard Privacy Assessment 19
3.3 Preliminary Results 20
3.4 Conclusion and Future Work 20

4 Explaining and Measuring Functionalities of Malware Detectors **21**
4.1 Introduction 24
4.2 Related Work & Motivation 25
4.2.1 Related Work and Background 26
4.2.2 Motivating Example 27

4.3 Threat Model & Problem Definition . . . 28
4.3.1 Threat Model . . . 29
4.3.2 Problem Definition . . . 29
4.3.3 Ethical Considerations . . . 30
4.4 Methodology . . . 30
4.4.1 Step 1: Feature Selection . . . 30
4.4.2 Step 2: Adversarial Sample Generator . . . 32
4.4.3 Step 3: Malware Detector Evaluation . . . 34
4.5 Experiment Setup . . . 34
4.5.1 Experimental Environment . . . 35
4.5.2 Target Detectors . . . 35
4.5.3 Datasets and Model Training . . . 35
4.5.4 Benchmark Methods . . . 36
4.6 Evaluation & Results . . . 37
4.6.1 Evaluation of Detectors . . . 38
4.6.2 Transferability Analysis . . . 40
4.6.3 Generalizability Analysis . . . 42
4.6.4 Revisiting AMM with Improved Detectors . . . 43
4.7 Case Studies . . . 45
4.7.1 A Case Study on Evasion Capability . . . 46
4.7.2 A Case Study on Detection Efficiency . . . 47
4.8 Discussion & Limitations . . . 49
4.9 Conclusion . . . 50

5 Conclusions and Future Work **51**

A Appendix **53**
A Structures of WinPE Binaries . . . 53

Bibliography **55**

List of Figures

1.1 The growth of malware during the past 14 years. 2
1.2 The process of training a machine learning model. 3

2.1 Overview of the Literature Survey. 6

3.1 An overview of the semi-automated assessment method for clipboard security. 19

4.1 An example of antivirus engine evasion. Code in blue: dummy function calls. 28
4.2 The overview of our evaluation framework. 29
4.3 Comparison of detection rate among seed samples, adversarial samples (default AMM-based), adversarial samples (balanced AMM-based) and adversarial samples (Statistics-based). 37
4.4 Detection rates of manipulating different sizes of feature maps on APK. 39
4.5 AMM values of features in different datasets and models. Each heatmap contains normalized AMM values of 1,024 features that are selected according to the adversarial generation model. 41
4.6 Detection rate of proposed adversarial samples generated from LGBM, SVM and RF against three detectors. 42
4.7 Detection rates of WinPE seed malware and AMM-based adversarial samples detected by four detectors 43
4.8 Detection rates of seed malware and AMM-based adversarial samples detected by models excluding different amount of important features. . 44
4.9 Precisions and detection rates of seed malware S_m, adversarial samples S_a (AMM-based features) and adversarial samples S_i (important features) on the original model and two updated model m_i and m_a. . 45
4.10 SAGE and AMM value distributions of important features and AMM-based features. 46
4.11 SHAP values of two APK samples before and after adversarial generation. (a) and (b) are the original and adversarial samples of *Sample 1*; (c) and (d) are of *Sample 2*. Details of features can be found in Table 4.3. 47
4.12 Decision function values of samples with different numbers of features manipulated with AMM-based, balanced AMM-based and Statistics-based strategies 48
4.13 Distribution of weights of features in SVM models trained by samples generated from different strategies. 48

A.1 Structure of WinPE files. 54

List of Tables

1.1 Common notations 3

2.1 Malware Detection Approaches 7
2.2 Explanation approaches 10
2.3 Attack Approaches 12

4.1 Target detectors 36
4.2 Selected feature numbers of three strategies. 38
4.3 APK feature IDs and their description 45

University of Adelaide

Abstract

Dissecting Malicious Behaviours of Mobile Applications

by Wei WANG

This thesis dissects the behaviours of malicious software and unveils the internal mechanism of evading the detection by malware detectors. With the popularity of smartphones, malicious software has been one of the most severe risks to the public. Therefore, mobile security has been a critical and hot topic in security research. Various malware detection and antivirus methodologies have been proposed to defend the rapid evaluation and variation. These malware variants can camouflage themselves with complicated techniques, such as obfuscation and feature perturbation, to evade the detection by antivirus products. Machine learning-based malware detection techniques are introduced into these products to address this problem. Machine learning-based malware detectors leverage features extracted from malicious and benign software to train detection models to identify malware and its variants effectively. In this thesis, I will first conduct a literature review of state-of-the-art malware detection techniques to unveil how these techniques contribute to anti-malware research. The literature review covers state-of-the-art methodologies, including software static and dynamic analysis, malware detection and machine learning. Then, an explainability-guided measurement approach is proposed to measure malware detectors' functionalities and guide adversarial sample generation. In this approach, we introduce a novel measurement concept, **Accrued Malicious Magnitude (AMM)** to identify which malware features should be manipulated to maximize the likelihood of evading detection. The AMM is defined as the product of the magnitude of SHAP values in each feature and the number of samples that have malicious-oriented values in the corresponding feature. Compared with SHAP values representing features' importance to the prediction results, AMM values reflect how much the specific features can contribute to flipping the prediction result. Finally, I will conclude the thesis and discuss the future work.

Declaration of Authorship

I acknowledge the support I have received for my research through the provision of an Australian Research Council Grant Funded Short Term Scholarship.

Wei Wang

May 2022

Acknowledgements

I would like to express my most tremendous appreciation to my supervisors, Dr. Minhui Xue and Assoc. Prof. Markus Wagner. This thesis cannot be fulfilled without the guidance and help from my supervisors. I would like to thank my supervisor, Dr. Minhui Xue, for guiding me to conduct research and to think and work as a researcher. I would like to thank my principal supervisor Assoc. Prof. Markus Wagner, who allowed me to have a flexible and accessible topic selection with patient guidance and professional advice. Last, but not the least, I would like to thank my parents, my partner and all my friends for supporting and helping me in all these years.

Achievements

Publications and expected publications are listed as follows:

- **Wei Wang**, Ruoxi Sun, Tian Dong, Shaofeng Li, Minhui Xue, Gareth Tyson, Haojin Zhu. 2021. Explaining and Measuring Functionalities of Malware Detectors. arXiv preprint arXiv:2111.10085. (Submitted in January 2022).
- **Wei Wang**, Ruoxi Sun, Minhui Xue, Damith C. Ranasinghe. 2020. An automated assessment of Android clipboards. Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, the Late Bricking Results track. (Published in December 2020).

Chapter 1

Introduction

1.1 Background

Malware detection is a hot topic in security-related research. Malware, short for malicious software, refers to harmful computer software that causes system and network disruption and gains access to sensitive information. Generally, malware can be categorized as virus, worm, trojan and other malicious programs. Recalling a brief history of malware, the Creeper program, designed to test self-replicating mechanisms in 1971, is the earliest one [32]. With over 50 years of battling against malware, modern world still suffers from the risks and attacks of malware. Figure 1.1 shows the growth of malware during the past 14 years (*i.e.* 2008 - 2022) [97]. The number of malware and potential unwanted application (PUA) increases significantly every year. According to the statistics [97], over 6.5 million Android malware and 121 million Windows malware were detected and captured in 2021. Although public app stores (*e.g.* Google Play Store, APKPure and AppStore) have integrated malware detection mechanisms, they are still the dominant malware distribution sources [76].

Malware detection mechanisms involve rule-based, machine learning-based and hybrid approaches. Rule-based approaches leverage static rules (*e.g.* signatures, specific headers, function calls, network requests and file IO requests) to monitor malicious behaviours of malware and their processes. For instance, ClamAV [35], a widely-known opensource antivirus engine, detect malware by file signatures and YARA [169] rules. However, the rule-based malware detection mechanism requires maintainers to update rules and signature database frequently, and it is difficult to detect newly developed malware or variants. Machine learning-based malware detectors extract features from existing malware dataset to train machine learning models used to predict unknown potential malware. Feature extraction functions may extract and vectorize either static information [14, 8], dynamic information [56], or both from application samples, which are remarkably different from the counterparts of computer visions. Commercial antivirus products, *e.g.* AVAST [16] and Kaspersky [71], have integrated machine learning approaches into their engines [5, 94], making them hybrid malware detection products. This thesis focuses on security and functionality issues of machine learning-based malware classification and detection approaches. Machine learning is a statistic-based computer algorithm that automatically evolves itself through experience and data.The process of training a machine learning model is shown in Figure 1.2. First, we extract features from the training samples. Then, these features are used to train a machine learning model. We use part of samples to evaluate the trained model and then the evaluation results are used to improve the model training. Finally, we extract features from input samples to predict their classification by the machine learning model. Machine learning algorithms are traditionally categorized as supervised learning, unsupervised learning, semi-supervised learning, and reinforced learning. Supervised learning algorithms require training input data and corresponding labels

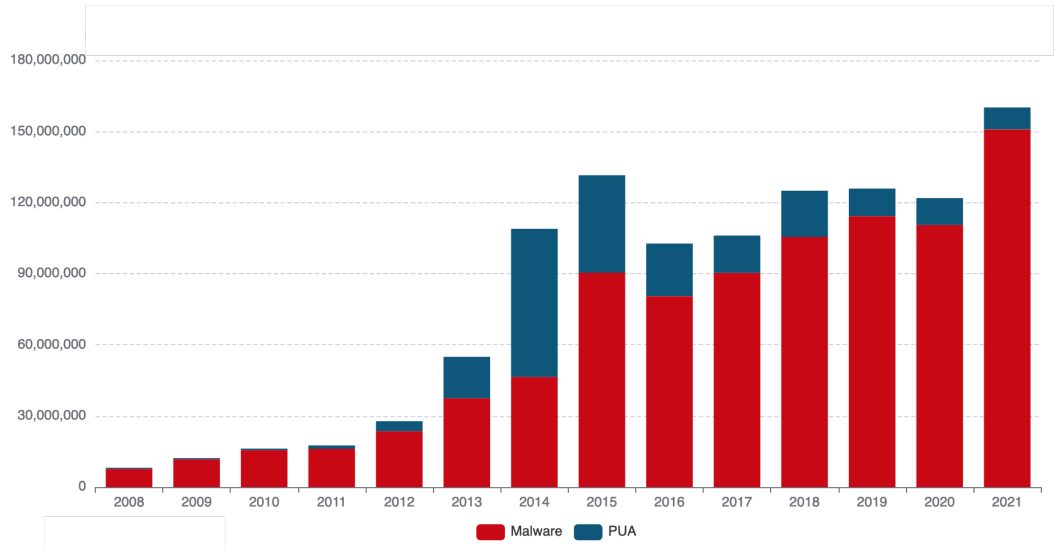


Figure 1.1. The growth of malware during the past 14 years.

while training a machine learning model. Semi-supervised learning algorithms do not require all training data to be labelled. In contrast, unsupervised learning algorithms do not require labels and cluster data based on commonalities. Reinforcement learning algorithms are concerned with how algorithms should take actions in an environment to maximize cumulative rewards. Typical usage of reinforcement learning is training gaming AI [141]. Deep learning is another class of machine learning based on artificial neural networks, inspired by information processing and communication mechanisms among neurons in biological systems.

Although machine learning algorithms can evolve themselves through data, it is crucial to understand why and how a machine learning model predicts a particular result since the internal structure of the model is complicated, especially deep learning neural networks. Therefore, many explaining and interpreting frameworks [91, 122, 59, 59, 37, 33] are proposed to address this issue. These frameworks help researchers to understand how each feature contributes to the results and to unveil potential risks (*e.g.* bias, over-fitting and potential attacks) in machine learning models [152].

Attacks on machine learning algorithms can be divided into two major categories: backdoor attacks and adversarial attacks. Backdoor attacks embed malicious triggers in the training dataset and activate in specific circumstances. It requires attackers to poison training datasets, which are usually public datasets contributed by users, so that generated models misclassify specific data. Adversarial attacks require attackers to generate perturbed features applying to the input so that the target model misclassifies the manipulated input. Note that malware-based machine learning attacks should consider the functionalities of original binaries [118, 152]. Therefore, features cannot be manipulated arbitrarily to fit the pure feature-space attacks.

1.2 Common Notations

The common notations are listed in Table 1.1.

1.3 Thesis Contribution

The contributions of this thesis can be summarised as follows:

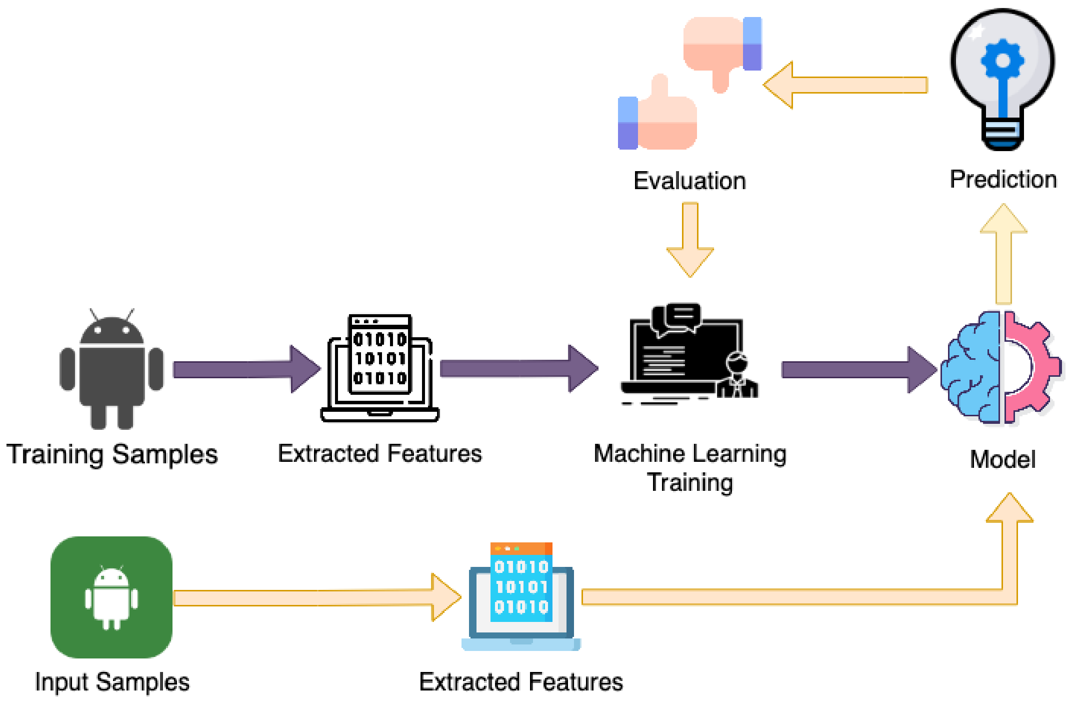


Figure 1.2. The process of training a machine learning model.

Table 1.1. Common notations

Symbol	**Description**
X, Y, Z	variable matrix
x, y, z	a variable or a variable vector
x_i	the i-th value of x
$f(x), g(x)$	functions; $f(x)$ usually refers to a machine learning function while $g(x)$ is a optimised or generation function of $f(x)$
$\mathbb{E}(X)$	expectation of X

- a synthesized literature survey to understand state-of-the-art research topics on machine learning-based malware security topics;
- a novel semi-automated static analysis framework to detect privacy-related malicious behaviours and security issues;
- a novel explainability-guided approach to evaluate functionalities and weaknesses of malware detectors.

1.4 Thesis Organization

The rest of this theses is organised as follows:

- Chapter 2: this chapter presents a comprehensive literature survey of machine learning-based malware detection research in three major aspects: detectors, explanation frameworks and attacks. The survey summarizes the state-of-the-art malware-related outcomes and helps us to reflect potential breakouts in security topics.
- Chapter 3: this chapter presents a rule-based static analysis framework to analyse data leakage issues from the clipboard of Android apps. The preliminary

evaluation of using this framework shows that some widely-installed Android apps can potentially leak sensitive data stored in the clipboard.

- Chapter 4: this chapter proposes an explainability-guided and model-agnostic malware detector measurement framework (Section 4.4). This framework exploits SHapley Additive exPlanations (SHAP) and introduces the concept of *Accrued Malicious Magnitude (AMM)* to guide the feature selection approach for feature-space manipulation. This chapter further generates adversarial Android malware variants and measures the functionalities and vulnerabilities of machine learning-based detectors and antivirus engines.
- Chapter 5: this chapter summarises this thesis and presents recommendations for future work.

Chapter 2

Literature Survey

2.1 Introduction

Although cybersecurity technologies have significantly developed and evolved in the past decades, malware is still not negligible to modern society. Malware developers, or attackers, rapidly utilize numerous techniques and tricks to evade malware detection. Meanwhile, they also conduct new attacks by continuously uncovering vulnerabilities to network systems. Security vendors and researchers update antivirus methodologies to deal with new attacks; however, we still have challenges to address.

One noteworthy challenge is that malware developers integrate compound techniques, *e.g.* obfuscation and encryption, to bypass malware detection. Attackers also leverage feature perturbation and generation techniques to evade malware detection by machine learning-based detectors. Another challenge is that the evolution of traditional rule-based malware detection always goes behind the evolution of malware. Security analysers need to find specific features of new malware or variants and update malware detection strategies, which is time-consuming; meanwhile, attackers can generate numerous variants in a short time. These challenges give attackers sufficient time to conduct massive attacks.

This literature survey reviews existing literature where machine learning is applied to analyse Windows Portable Executives (WinPE) and Android Application Package (APK) malware. WinPE analysis techniques differ slightly from APKs because of significant distinctions in operating systems and applications. However, we can still leverage similar detection algorithms of machine learning to detect malware. In this literature survey, we focus on three aspects of malware-related security research: (*i*) malware detection methodologies based on machine learning techniques; (*ii*) explanation and interpretation methodologies; and (*iii*) attacks on machine learning-based detectors. The first aspect considers machine learning-based detection algorithms on WinPE and APK malware, which are two dominant operating systems suffering from malware and ransomware attacks. The second aspect unveils the internal decision factors of machine learning-based classifiers and how classification results can be explained. The third aspect illustrates how attackers leverage vulnerabilities of existing malware detectors and classifiers and how researchers address potential attacks on existing malware detection mechanisms. These three aspects guide researchers to discover novel topics in malware security research.

This work includes the following contributions:

- a comparative analysis of the literature on machine learning-based security topics covering malware detection algorithms, explanation methods and attacks;
- reflections of malware related research.

Figure 2.1 illustrates the overview of this chapter. The rest of the chapter is organised as follows: Section 2.2 describes related literature surveys; Section 2.3

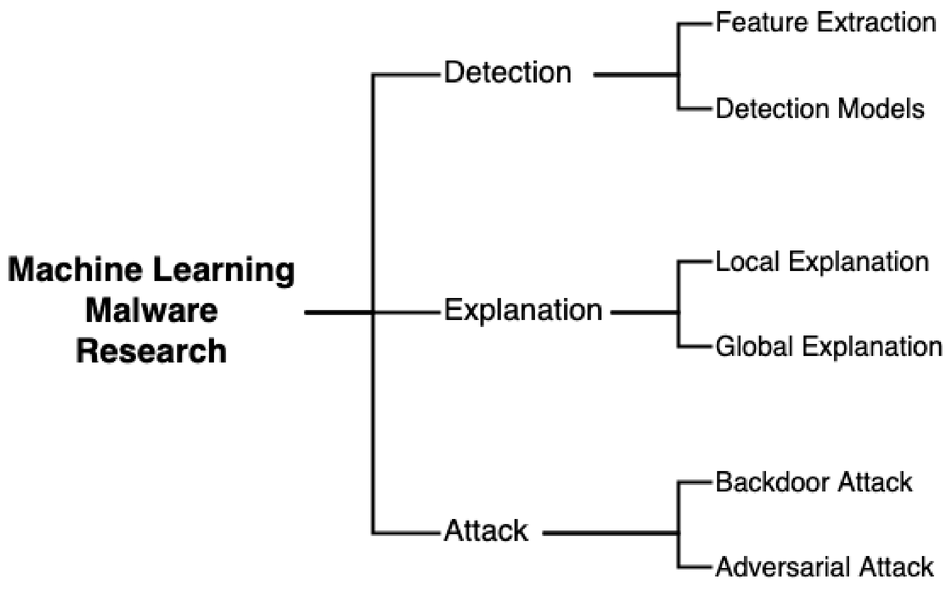


Figure 2.1. Overview of the Literature Survey.

presents state-of-the-art malware detection methodologies based on machine learning; Section 2.4 introduces explanation methodologies on machine learning models; Section 2.5 shows adversarial attacks on machine learning detectors; and finally Section 2.6 presents reflections and conclusions of this work.

2.2 Existing Literature Surveys and Related Work

Several academic literature surveys have addressed malware analysis and machine learning-based detection. Ucci *et al.* [142] wrote a literature review of machine learning frameworks for malware analysis. The authors proposed a definition of the taxonomy to synthesise off-the-shelf WinPE malware detection frameworks based on machine learning by reviewing 64 papers. The taxonomy of malware analysis techniques consists three dimensions: objective, feature and machine learning algorithm. The objective includes malware detection, similarity analysis and category detection; the feature includes feature extraction functions and WinPE feature analysis; and machine learning algorithm includes supervised, unsupervised and semi-supervised learning. Liu *et al.* [89] reviewed machine learning-based Android malware detection. This survey focuses on essential aspects of Android malware detection, including data processing, feature selection, machine learning algorithms, and detection evaluation.

Regarding explaining machine learning prediction, Vilone *et al.* [143] wrote a systematic review of eXplainable Artificial Intelligence (XAI) as a whole, and they tried to define the boundaries of the discipline of XAI. This survey reviews XAI method articles with four main aspects: review articles, theories and notions, methods, and evaluation. Linardatos *et al.* [88] compared and analyzed machine learning interpretability methodologies on both white-box and black-box machine learning detectors. They also tested the best use cases for each interpretability method and provided links to their programming implementations.

Akhtar *et al.* [6] wrote a systematic review of adversarial attacks on computer vision deep learning. They reviewed 12 adversarial attack methodologies on deep learning classification models. The authors also reviewed attacks on other scopes, including autoencoders and generative models, deep reinforcement learning, semantic

Table 2.1. Malware Detection Approaches

Authors	ML Algorithm	Features	Type	Dataset
Anderson *et al.* [8]	LightGBM [72]	parsed values, histogram	WinPE	EMBER2018 [8]
Harang *et al.* [61]	LightGBM, NN	parsed values, histogram	WinPE	SOREL-20M [61]
Raff *et al.* [112]	MalConv, Byte n-grams	raw byte codes	WinPE	Virus Share [147], Open Malware [102]
Graziano *et al.* [56]	Logistic Model Tree	dynamic features	WinPE	Anubis Sandbox
Zhang *et al.* [171]	CNN+LSTM	API calls	WinPE	private
Arp *et al.* [14]	SVM	8 sets of static features	APK	Drebin [14]
Xu *et al.* [161]	SVM	AndroidManifest	APK	Google Play, VirusTotal [148], Drebin
McLaughlin *et al.* [99]	CNN	n-grams features from opcode sequences	APK	Android Malware Genome [9], Google Play
Kwong *et al.* [165]	N/A	static & dynamic	APK	Android Malware Genome
Saxe *et al.* [116]	NN	strings	WinPE	VirusTotal
Vinayakumar *et al.* [145]	LSTM	static & dynamic	APK	Android Malware Genome
Naway *et al.* [100]	NN	8 sets of static features	APK	Drebin

segmentation and object detection, and face attributes [6]. Despite these "ad-hoc" scenarios, they also reviewed attacks in the real world, including cell-phone cameras, road signs, 3d objects, cyberspace and robotic vision. Martins *et al.* [98] reviewed adversarial attacks on intrusion and malware detectors.

The existing reviews focus on the individual aspect of malware detection, adversarial generation and explanation. This chapter will conduct a comprehensive survey that links all these aspects together to illustrate a wider picture of malware security research.

2.3 Machine Learning-based Malware Detection

This section introduces state-of-the-art machine learning-based malware detectors and classifiers. In this section, we will review two dimensions of machine learning-based detectors. The first dimension is feature extraction functions used to train machine learning-based malware detectors for WinPE and APK. In contrast, the second dimension covers structures of detectors, including ML models and neural networks. Papers and methodologies reviewed in this section are listed in Table 2.1.

2.3.1 Feature Extraction

This section addresses the feature extraction functions of samples used to analyse malicious behaviours and train machine learning detectors. The structure and content of malware features vary due to the significant difference between Windows and Android systems. Features can be extracted from either static analysis, dynamic analysis, or a combination of both methods.

Windows Portable Executable (WinPE)

Ember [8] is a WinPE dataset that is widely adopted in academic research [118, 144, 130, 166, 158, 66]. It offers a feature extraction function that extracts 2,351 raw features, composed of both parsed values and binary-related histograms, from WinPE binaries. The parsed values include general and string-related statistical data, section information, header information, imported and exported function information. The raw features are extracted and stored in a human-readable format that can be converted to 2,351-dimension feature vectors. Ember provides off-the-shelf raw features extracted from over 1 million samples. SOREL-20M [61] leverages the same feature extraction function and offers features from around 20 million samples with a set of pre-trained LightGBM models and feed-forward neural networks.

Raff *et al.* [112] leverage the static feature extraction function based on the byte-level content of a WinPE binary. They train machine learning models with the raw bytes of whole binaries, which alleviate a series of issues from other common byte n-gram approaches [3, 25], *e.g.* sensitive features and overreliance on the PE-Header. Generally, an n-gram is a series of n bytes of a binary where features are different combinations of these n bytes with various strategies. These n-gram features usually represent the counts of specific combinations existing in a binary.

Graziano *et al.* [56] introduce network activities as dynamic features, including statistics on protocols, TCP/UDP ports and network requests. They also capture file operations executed by WinPE binaries in sandboxes. Their proposed framework combines dynamic and static features as a whole to train a logistic model tree to detect malware. Zhang *et al.* [171] also leverage API calls extracted from dynamic analysis of WinPE dataset to train a neural network.

Android Application Package (APK)

Drebin [14] is a popular dataset [170, 105, 150] that provides both feature extraction function and samples. This dataset categorizes static features into eight groups, including hardware components, requested permissions, app components, filtered intents, restricted API calls, used permissions, suspicious API calls and network addresses [14]. Features in each group represent the existence of each feature (*i.e.* 1 represents existence while 0 is absence). The authors trained a Support Vector Machine (SVM) detector that has over 95% accuracy. Xu *et al.* [161] leverages Android manifest elements, including components, explicit intent, implicit intent and intent filter, as features to train machine learning models.

McLaughlin *et al.* [99] propose an n-gram based malware detection approach that leverages the opcode sequences from the entire decompiled source code classes as features to train a deep neural network. The proposed approach only focuses on 218 opcodes defined by the Android framework, making 218-dimension feature vectors. This feature extraction function has significantly fewer dimensions than Drebin, of which the dimension depends on the number of 8 types of features existing in the training dataset.

Kwong *et al.* [165] leverage both static analysis and dynamic features in the runtime. They trace API calls, native instructions, Dalvik instructions and taints as dynamic features, and analyze decompiled code to find vulnerabilities.

2.3.2 Malware Detection

This section introduces widely adopted machine learning-based malware detection approaches: regular machine learning models in Section 2.3.2 and deep learning models in Section 2.3.2.

Machine Learning Models

In the previous section we introduced that Drebin [14] leveraged the Support Vector Machine (SVM) algorithm to detect malware. SVMs are a set of supervised machine learning algorithms that are usually used in classification and regression analysis. Meanwhile, LightGBM [72] is another classification model used in malware detection [8, 61]. It is a gradient boosting framework based on decision tree learning algorithms, including variants of gradient boosting algorithms (*e.g.* GBDT, GBM and GBRT) and random forest.

Deep Learning

Deep learning is regarded as an evolution of machine learning by leveraging programmable neural networks to improve accuracy and reduce human interference. Saxe *et al.* [116] utilize neural networks to detect malware. They extract strings and WinPE file characteristics as features to train a neural network with two hidden layers. McLauhlin *et al.* [99] trained a convolutional neural network (CNN) with embedding projection to detect WinPE malware. Vinayakumar *et al.* [145], introduced long short-term memory (LSTM) into Android malware identification by using features extracted from static and dynamic analysis. Naway *et al.* [100] trained a deep neural network (DNN) classifier with features similar to Drebin [14]. Zhang *et al.* [171] trained sophisticated deep learning neural networks combining multiple gated-CNNs and a bidirectional long-short term memory (LSTM) networks.

Concept Drift

Although machine learning-based malware detectors can effectively detect malware based on existing features of dataset, they still face the challenge of concept drift. Concept drift is an online supervised learning scenario when the target inputs to be predicted change over time in anticipated manners [54, 122, 117]. Yang *et al.* [166] proposed CADE framework to address the challenges from concept drift and complement existing supervised training-based malware detectors. The proposed framework introduces contrastive learning to map input data into latent space, and leverages Median Absolute Deviation [80] to detect concept drift. Jordaney *et al.* [67] proposed a concept drift detection framework, Transcend, based on statistically evaluating the performance of a detector and filtering out unreliable detection results. The authors applied their framework on DroidScribe [41] to enhance the identification ability of concept drift [67].

2.4 Explanation on Machine Learning

Understanding why and how a machine learning model predicts a particular result is crucial as the internal structure of the model is complicated, especially deep learning neural networks. Researchers proposed various approaches to explain the connections between features and predictions. Explanation approaches are generally categorized into two types: local and global explanations. Local explanation interprets how features contribute to an individual sample's prediction result, while global explanation explains how much a model depends on each feature. However, these two types of explanation approaches are not strictly discriminated [36]. This section introduces explanation methodologies on machine learning frameworks, and the major works are listed in Table 2.2.

2.4.1 Local Explanation

The LIME [114] method interprets and explains individual predictions via local approximation on the given model around predictions, which is a model-agnostic manner.

Table 2.2. Explanation approaches

Approach	**Description**
LIME [114]	model-agnostic local estimation
DeepLIFT [122]	local explanation, designed for Deep Learning
LEMNA [59]	local explanation, designed for Deep Learning
SHAP [91]	a unified local explanation, model-agnostic & model-specific
TreeExplainer [90]	a tree-based explainer for the SHAP framework
CADE [166]	concept drift explanation framework
SAGE [37]	a model-agnostic global explanation framework
Chen *et al.* [33]	global robustness property-based explanation approach
Ilyas *et al.* [65]	global sensitivity-based explanation approach
Patel *et al.* [103]	differencial-privacy-based explanation approach

Generally, the explanation produced by LIME is obtained by the following:

$$\xi = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g), \tag{2.1}$$

where G is a class of explanation models, $\mathcal{L}$ is loss functions, and Ω is complexity measures.

DeepLIFT [122] is a recursive prediction explanation framework designed for interpreting deep learning neural networks. It interprets the prediction result of a deep learning neural network via backward propagating the contributions of neurons to each feature of the input data. Compared with model-agnostic explanation methods, DeepLIFT has a faster explanation speed and better computation performance on deep learning neural networks. LEMNA [59] is an explanatory framework that aims to explain deep learning malware classification results. It is specifically designed to address feature dependency on binary analysis and deal with nonlinear local boundaries to enhance explanation fidelity [59].

SHapley Additive exPlanations (SHAP) [91] is a unified explanation framework that ensembles multiple explanation approaches into a Shapley value-based coalitional game theory concept. This framework explains model predictions in either model-agnostic or model-specific manner by leveraging different explainers. SHAP aims to calculate the contribution of each feature in predicting the individual result. To accomplish this task, it generates a surrogate explanation model g of the form:

$$\begin{gathered} f(x) = g(x'), \\ g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j, \end{gathered} \tag{2.2}$$

where f is the original model, x' is the coalition vector of x. The Shapley value $\phi_j \in \mathbb{R}$ is the feature attribution for the feature x'_j to the sample's prediction result. Although the SHAP framework can explain an arbitrary model in a model-agnostic manner, the explanation process can be boosted via customizing its explainer. The authors further proposed the TreeExplainer [90] to interpret predictions by decision tree-based models.

CADE [166] explain concept drift by proposing a Distance-based Explanation, which is evaluated with the Boundary-based Explanation implemented from these approaches [24, 40, 50, 51]. Regular machine learning classifiers make predictions via dicision boundaries; in contrast, the concept drift detection model is based on the sample's distance to centroids of distribution clusters of training data [166]. Therefore, the proposed approach seeks a set of original features that push the drifting sample input toward the closest centroid.

2.4.2 Global Explanation

SHAP calculates how much each feature contributes to an individual prediction (local explanation). Shapley Additive Global importancE (SAGE) [37] calculates how much each feature contributes to the predictive power, *i.e.* importance, across the whole dataset (global explanation). The important features, which enhance the model's prediction performance, will have large values, while unimportant features will have small values. To accomplish this task, SAGE defines the restricted model f_S that only a part of the entire features set are chosen as:

$$f_S(x_S) = \mathbb{E}[f(X) \mid X_S = x_S], \tag{2.3}$$

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

$$v_f(S) = \underbrace{\mathbb{E}[\ell(f_\emptyset(X_\emptyset), Y)]}_{Mean\ prediction} - \underbrace{\mathbb{E}[\ell(f_S(X_S), Y)]}_{Using\ features\ X_S}, \tag{2.4}$$

where $v_f(S)$ represents the performance of f given features X_S. As well known in game theory, Shapley values reflect the credit allocation [37] and adopt this framework to attribute the model's prediction power on sets of features. Therefore, the expression is shown as follows:

$$\phi_i(v_f) = \frac{1}{d} \sum_{S \subseteq D \setminus \{i\}} \binom{d-1}{|S|}^{-1} \left(v_f(S \cup \{i\}) - v_f(S)\right), \tag{2.5}$$

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

Chen *et al.* [33] define global robustness properties to explain malware detection. The properties involve five attributes: monotonicity, stability, high confidence, redundancy and small neighborhood. The authors leverage these five properties to explain domain knowledge about suspicious factors, evasion strategies and the semantics and dependency among features.

Ilyas *et al.* [65] proposed the concepts of ρ-useful and γ-robustly features to help explain global sensitivity of features in machine learning models. The authors claimed that these useful but non-robust features are the causes of adversarial attacks, which is evaluated with empirical studies.

Patel *et al.* [103] proposed an adaptive differential privacy algorithm for explanation methodologies to prevent sensitive information about training datasets from leakage. The proposed algorithms can be adopted to any explanation approach based on local queries.

Table 2.3. Attack Approaches

Authors	Type	Description
Severi *et al.* [118]	Backdoor	a SHAP-guided backdoor attack
Ma *et al.* [92]	Backdoor	hiding malformed triggers into neural networks by leveraging model quantization in embedded frameworks
Li *et al.* [84]	Backdoor	a backdoor attack on NLP
Song *et al.* [127]	Adversarial	an adversarial attack guided by Reinforcement Learning (RL)
Zhao *et al.* [172]	Adversarial	an adversarial attack on Android detectors guided by RL
Li *et al.* [81]	Adversarial	an empirical study to evaluate the quality of predictive uncertainties of Android detectors with adversarial samples
Slack *et al.* [124]	Adversarial	fooling LIME and SHAP
Wang *et al.* [153]	Others	hidding malare in a neural network
Li *et al.* [107]	Adversarial	an adversarial attack on computer vision via active learning
Carlini *et al.* [21]	Others	extracting confidential data and secrets from a deep neural network

2.5 Attacks on Machine Learning

This section will illustrate state-of-the-art attacks on machine learning-based malware detectors. We generally introduce two dominant attacks: backdoor attacks and adversarial evasion attacks. However, attacks on machine learning are not limited to these two types. Papers introduced in this section are listed in Table 2.3.

2.5.1 Backdoor Attack

Backdoor attacks embed sophisticated malicious inputs into machine learning models, which only trigger and cause misclassifications on model inputs containing a specific activator [156]. Severi *et al.* [118] conducted a backdoor attack on malware detectors guided by the explanation approach. The authors proposed two feature selection and three value selection functions with a Greedy Combined Selection strategy guided by SHAP [91] to generate manipulable features and values. The proposed backdoor attack is based on a model-agnostic manner, does not require to access the labelling process and adopts to restrictive adversarial models, *e.g.* poisoning a small number of samples with manipulated features and values.

Ma *et al.* [92] proposed a novel backdoor attack that leverages model quantization empowered by commercial machine learning frameworks to hide malformed triggers into neural networks. Commercial machine learning frameworks, *e.g.* TensorFlow Lite [138] and PyTorch Mobile [111], post-quantize a large high-precision model (*i.e.* float-32) into a small low-precision model (*i.e.* int-8) to reduce the computation resource due to the limitation of embedded devices. The authors utilize such a feature to hide a sophisticated backdoor into a neural network model. The backdoor can be triggered when converting the precision of values without being identified by malware detectors.

Besides malware-related backdoor attacks, Li *et al.* [84] conducted backdoor attacks on Natural Language Processing (NLP) models by replacing several characters with mal-formed Unicode characters in training words and sentences. For instance, they replace the Latin small letter "e" (code 0065) with Cyrillic small letter "e" (code 0435). These backdoor characters are then used in training NLP models and triggered while processing natural language inputs, *e.g.* misclassifying or generating malicious words or sentences.

2.5.2 Adversarial Evasion & Other Attacks

Adversarial evasion attacks aim to manipulate malicious samples targeting specific models so that the manipulated samples are misclassified by the models. Reinforcement learning (RL) is a field of machine learning that focuses on the concept of how an intelligent agent should behave in an environment to maximize cumulative reward. Compared with the machine learning models mentioned above, RL does not require labelled data as input; instead, it focuses on finding a balance between exploration and exploitation [69]. Therefore, many researchers leverage RL to conduct adversarial attack on malware detectors. Song *et al.* [127] proposed an adversarial attack approach based on reinforcement learning. The proposed framework targets both machine learning detectors and commercial AV engines. It addresses the action selection problem with three processes: (*i*) converting the generation process to a stateless process, (*ii*) reusing successfully evaded payloads in modelling and (*iii*) minimising the changes on adversarial examples to assign rewards correctly. The framework's action minimiser can figure out ineffective actions for adversarial sample generation and only change minimal features, explaining the reason for evasion attacks.

Zhao *et al.* [172] also leverages RL to generate Android adversarial samples. The proposed approach maps graph modifications in vectors to semantic code-level manipulations on apps to generate final adversarial APK samples. Specifically, graph modifications include inserting and removing nodes and edges, while code-level manipulations include adding and removing methods, adding call relations and rewriting code instructions.

Li *et al.* [81] conducted an empirical study to evaluate the quality of predictive uncertainties of Android malware detectors with adversarial samples. They redesign 24 state-of-the-art Android malware detectors and measure their functionalities with nine methods, three of which address the data imbalance issue. Adversarial samples are generated via feature perturbation, which leverages PGDs+GDKDE attack and the Mimicry attack that are borrowed from vision-based feature perturbation methods, and obfuscation techniques.

Explanation approaches explains which features contribute to the classification, learned from the previous section. Therefore, explanation frameworks can be used on adversarial attacks. Slack *et al.* [124] conducted adversarial attacks based on LIME [114] and SHAP [91] and demonstrated how they can be defeated. The authors generate adversarial examples via perturbing features based on explanation frameworks to attack original biased detectors. Then they generate an unbiased detector, which may fool the explanation frameworks but can detect the distribution distances of inputs to training samples, to detect whether the input is adversarial examples. Such an approach can effectively defend adversarial attacks based on explanation methods; however, it can also be a target to exploit and decrease the accuracy of malware detection.

Despite adversarial attacks on machine learning-based malware detectors, Wang *et al.* [153] proposed an attack on deep neural networks by embedding malware in neurons. From the evaluation, the malware embedded in the neural networks can successfully evade the detection of antivirus detectors in VirusTotal, and the neural networks have around 1% accuracy loss. Li *et al.* [107] conducted a black-box adversarial attack on computer vision classifiers via active learning. Carlini *et al.* [21] proposed a framework to detect and extract confidential data and secrets by leveraging memorization mechanism in training a deep neural network. These research exposes different weaknesses of existing machine learning classifiers and detectors. Sun *et al.* [134] unveiled mobile advertisement fraud via assessing network traffics. Hu *et al.* [64]

evaluated TableGAN-MCA and discovered that GAN-synthesized table releasing can potentially leak privacy data. Many research[19, 26, 106, 46, 38, 129, 23, 160, 159, 93] evaluated data leakage and attacks in deep learning neural networks. Wen *et al.* [155] evaluated poisoning attacks on logistic regression models. Li *et al.* [85, 86] also evaluated backdoor attacks on deep learning models. Zhou *et al.* [173] evaluated profiling attacks on federal learning. Zhu *et al.* [174] unveiled clicking fraud attacks on Android apps. Much research [136, 27, 119] evaluated attacks on mobile systems and apps. Wen *et al.* [154] evaluated poisoning attacks and defenses for linear regression models.

2.6 Conclusion

This chapter conducts a comparative analysis of existing literature of malware-related security topics. Similar machine learning-based malware detection and classification algorithms can be applied on both WinPE and APK malware. However, feature extraction functions have significant differences between the two types of malware due to the differences in runtime and binary structures between Windows and Android systems. In addition, feature extraction functions may cover different aspects of malware binaries, *e.g.* static and dynamic features, which may lead to coverage and concept drift issues. Explanation methods expose the inner decision factors of machine learning classifiers, which guides attackers and researchers to discover vulnerabilities of malware detectors. Meanwhile, explanation frameworks also help researchers to address security issues and eliminate potential attacks. Existing adversarial attack methodologies highly relies on feature perturbation based on either explanation or reinforcement learning approaches, which consume remarkable computation resources. From existing literature, we try to find efficient approaches to attack malware detectors that are effective, explainable and economic. In the following chapters, we discuss how to discover privacy-related malicious behaviours and security issues via static analysis and propose an approach to explain and measure functionalities of malware detectors.

Chapter 3

A Semi-Automated Assessment of Android Clipboards

Statement of Authorship

Title of Paper	An Automated Assessment of Android Clipboards
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Wei Wang, Ruoxi Sun, Minhui Xue, and Damith C. Ranasinghe

Principal Author

Name of Principal Author (Candidate)	Wei Wang		
Contribution to the Paper	Performed experiments, analysed all samples, interpreted data and wrote manuscript		
Overall percentage (%)	85%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	06/03/2022

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Ruoxi Sun		
Contribution to the Paper	Helped in data interpretation and manuscript editing and evaluation		
Signature		Date	09/03/2022

Name of Co-Author	Minhui Xue		
Contribution to the Paper	Supervised development of work, helped in manuscript editing and evaluation		
Signature		Date	09/03/2022

Please cut and paste additional co-author

Statement of Authorship

Name of Co-Author	Damith C. Ranasinghe
Contribution to the Paper	Supervised development of work, helped in manuscript editing and evaluation
Signature	Date 11/03/2022

Please cut and paste additional co-auth

3.1 Introduction

A clipboard is a temporary buffer to allow short-term storage and data transfer within and between applications. It is regarded as a fundamental component of most operating systems and can be accessed either by users or by applications. Additionally, the functionality of the clipboard, such as copy and paste, is by default not constrained by the mobile operating system itself.

In June 2020, Apple published its most recent version of the iOS system, iOS 14. This version introduces a new privacy strategy: it prompts notifications while an app is attempting to access the clipboard. Consequently, numerous highly-used apps that access user clipboard data are exposed; Tik-Tok has been revealed to grab the contents of the clipboard every 1-3 keystroke(s), while LinkedIn has been shown to copy the contents of the clipboard every keystroke [157]. In contrast to iOS 14, the Android system does not include such a feature to inform its users of clipboard accesses. Hence, apps can monitor changes of the clipboard in both foreground and background without any user acknowledgement or consent.

Although, in 2019, Android 10 no longer allowed apps to access the clipboard in the background [110], the input method editor (IME) was excluded; such a restriction alone is too weak to protect user privacy. Malicious programs can arbitrarily read sensitive data from the clipboard and transfer the data to a remote service, hence, users are still exposed to severe threats from private and confidential data leaks. For example, since the wallet address of cryptocurrencies, *e.g.*, Ethereum [48] and Bitcoin [18], is usually long and hard to memorize, users copy the payee's address and paste it into the wallet app when making a transaction. Recently, it was substantiated that a malicious application found in the Google Play Store monitors the clipboard, obtains the private keys and seeds, and then forwards the data to the attacker's Telegram account to steal a user's cryptocurrency [2]. Recent work has proposed research related to privacy policy [133] and privacy leakage [83, 28, 132, 168, 62, 29, 151, 31] of mobile apps, but our work focuses on the problem of clipboard data leakage—the transfer of private user data to backend servers without user consent.

In order to distinguish privacy leakage behaviours from normal clipboard use, we define the analysis criteria as (*i*) programs read from and write to the clipboard, data, without user notifications or consent; and (*ii*) clipboard content is directly sent to backend services. This study, to the best of our knowledge, *performs the first security and privacy investigation of the clipboard access on Android apps.* Our key contributions are as follows:

- We analyze the mechanism of clipboard access in an Android system and develop an approach to determine whether an app obtains data from the clipboard and detect the data acquisition behaviour, *e.g.*, the timing and frequency of clipboard access.
- We architect the framework for a *first* tool, to the best of our knowledge, to identify potential privacy leakage emanating from the transfer of data from Android clipboards to backend servers and implement a preliminary version that combines manual and automated methods. The detection approach utilizes static taint analysis to expose the data flow from clipboard to risky sinks and integrates call graph backtracking to determine clipboard data leakages.
- We conduct our preliminary experiments on a popular app, *i.e.*, Sogou Input. The experimental results illustrate the effectiveness of our method by revealing potential clipboard privacy leakage, *i.e.*, direct data uploads to a backend server.

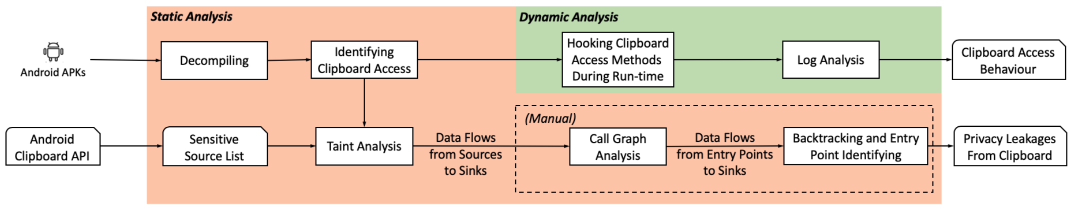

Static Analysis
Dynamic Analysis
Android APKs
Decompiling
Identifying Clipboard Access
Hooking Clipboard Access Methods During Run-time
Log Analysis
Clipboard Access Behaviour
Android Clipboard API
Sensitive Source List
Taint Analysis
Data Flows from Sources to Sinks
(Manual)
Call Graph Analysis
Data Flows from Entry Points to Sinks
Backtracking and Entry Point Identifying
Privacy Leakages From Clipboard

Figure 3.1. An overview of the semi-automated assessment method for clipboard security.

3.2 Clipboard Privacy Assessment

An overview of our semi-automated clipboard privacy analysis methodology is shown in Figure 3.1. To assess the clipboard privacy of Android apps, we perform (*i*) static code analysis to identify clipboard access; (*ii*) dynamic analysis to detect the access timing and frequency; and (*iii*) data flow analysis to determine privacy leaks (through a manual process). To assess the tool, we focus our study on the popular *e.g.*, Sogou Input [126], with more than 10 million downloads in the Google Play store and 1.5 billion in the Tencent App Store.

Clipboard access identification. After decompiling the Android Packages (APKs), we scan the source code of each app to detect the function calls of `ClipBoardManager.getPrimaryClip()` and `ClipData.Item.getItemAt(int)`, in order to determine whether an app accesses the clipboard or not. Once a clipboard-related function call is detected in an app, we will further investigate its clipboard access behaviour by dynamic analysis, and assess the privacy leakage through further static analysis.

Clipboard access behaviour detection. After clipboard access methods are identified in the source code, we run the corresponding app on a rooted device where a dynamic instrumentation framework, FRIDA [52], is installed. Using FRIDA, we are able to monitor the app's status, hook system APIs, change the app's behaviours, and trace a function call path through injecting JavaScript scripts which can be executed with full access to device memory [53]. To detect clipboard access behaviour, we hook and track the `ClipboardManager.getPrimaryClip()` method during run-time and extract the timing and frequency of the method calls from output logs.

Privacy leakage determination. From the Android clipboard API, we identify methods that access the clipboard, *e.g.*, `ClipboardManager.getPrimaryClip()` and `ClipData.Item.getItemAt(int)`, as sensitive Sources. These Sources acquire the sensitive data from the clipboard, and then they are transferred to Sinks that leak the data outside of the app. For example, the Sink `OkHttpClient.newCall()` leaks Sources to the backend service via `OkHttp` [101] library, and another Sink `Writer.write(String)` writes the Sources into either local files or HTTP connections. Then, we use FLOWDROID [15] to conduct a taint analysis on the apps identified of clipboard access to screen out high risk privacy leakages. The output of taint analysis are data flows from Sources to Sinks, which will be further analysed using call graph analysis.

In our call graph analysis, we apply ANDROGUARD [10] to track method invocations from Entry Points to Sources. We define the app code from the Android framework that directly or indirectly invokes Sources, *e.g.*, `View.OnClickListener.onClick()` and `Activity.onResume()`, as an Entry Point. By concatenating the call graph results with the data flows previously obtained, we are able to backtrack data flows from Sinks to Entry Points. If an Entry Point indicates a user-originated event, *e.g.*, a click event which is not an abuse of clipboard functionalities, we remove it and store the remaining results for manual inspection.

3.3 Preliminary Results

We evaluate the effectiveness of assessment method using a popular app abusing clipboard data.

Sogou Input is the most widely-used Chinese input method editor (IME) app with over 10 million downloads in Google Play and 1.5 billion in Tencent App Store. Although the Android system has restricted apps from accessing the clipboard in the background since Android 10, it still allows the app configured as a default input editor to monitor the clipboard [87].

We discovered that the `cyj.onPrimary-CLipChanged()` method monitors clipboard changes in the background and transfers clipboard contents to `CopyTranslateResultActivity`, and then it transfers the clipboard data to the backend server. Moreover, once users invoke the app's main screen, `Sougou-IMEHomeActivity`, the app copies the clipboard data and sends it to the backend server. This is a clear case of a user privacy breach whilst using this IME app as it acquires and sends clipboard data to remote servers whenever it is running in *foreground* or *background*, *without any user consent.*

3.4 Conclusion and Future Work

Since cross-platform development has become widely accepted in recent years, privacy data leakage more likely happens among apps, and its detection will be more complicated. Based on our preliminary findings, we plan to further extend our work to large-scale assessment in order to validate our clipboard privacy tool.

Chapter 4

Explaining and Measuring Functionalities of Malware Detectors

Statement of Authorship

Title of Paper	Explaining and Measuring Functionalities of Malware Detectors
Publication Status	☐ Published ☐ Accepted for Publication [✓ Submitted for Publication] ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Wei Wang, Ruoxi Sun, Tian Dong, Shaofeng Li, Minhui Xue, Gareth Tyson, Haojin Zhu

Principal Author

Name of Principal Author (Candidate)	Wei Wang		
Contribution to the Paper	Preformed experiments, analysed all samples, interpreted data and wrote manuscript.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	06/03/2022

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Ruoxi Sun		
Contribution to the Paper	Helped in data interpretation and manuscript editing and evaluation.		
Signature		Date	09/03/2022

Name of Co-Author	Tian Dong		
Contribution to the Paper	Helped in model training and manuscript editing		
Signature		Date	08/03/2022

Please cut and paste additional co-author panels here as required.

Statement of Authorship

Name of Co-Author	Shaofeng Li		
Contribution to the Paper	Helped in model training and manuscript editing		
Signature		Date	09/03/2022

Name of Co-Author	Minhui Xue		
Contribution to the Paper	Supervised development of work, helped in manuscript editing and evaluation		
Signature		te	09/03/2022

Name of Co-Author	Gareth Tyson		
Contribution to the Paper	Helped in manuscript editing and evaluation		
Signature		te	10/03/2022

Name of Co-Author	Haojin Zhu		
Contribution to the Paper	Supervised development of work. Helped in manuscript evaluation		
Signature		e	11/03/2022

Please cut and paste additional co-author panels here as required.

4.1 Introduction

Malware continues to be one of the most pressing security issues that users face today. Recent research has shown that the total number of malware infections has been rising for the last decade (2009 to 2018) [39]. In 2018, the number of malware infections was 812.6 million across mobile phones and computers, while during the first nine months of 2019, at least 7.2 billion malware attacks and 151.9 million ransomware attacks were reported. Thomas *et al.* [139] presents the risks of stolen credentials raised by malware, suggesting that 7–25% of exposed passwords match a victim's Google account. Furthermore, the attack rate has hit a new high during the COVID-19 pandemic [128]. These figures suggest that traditional signature-based methods cannot keep up with the rampant growth of novel malware. Hence, commercial antivirus companies have started using machine learning [4, 137] to enable detection without the need for signatures. However, research has demonstrated that attackers can evade machine learning-based detectors by manipulating the malware features that such detectors use [44, 45, 96, 108, 164, 167]. Because of this, commercial antivirus systems are susceptible to adversarial attacks [123]. Although there has been several works [55, 22, 49] looking at adversarial attacks in computer vision (where adversaries change specific pixels), adversarial attacks on malware are far less understood.

For the purposes of this chapter, we divide such attacks into two broad categories. The first group relies on *problem-space* obfuscation. Here we consider the problem space as a domain containing real-world objects (*e.g.* malware code, images, audio). Obfuscations in the problem-space change the semantic meanings of code snippets and further obfuscate the malicious signatures or patterns, thereby fooling rule-based malware detectors. Researchers have proposed a variety of such obfuscation techniques to generate adversarial malware that can evade detection by manipulating this domain [12, 149, 146, 68, 75, 113]. These include approaches such as hiding the control flow, inserting dummy code, and manipulating variable names.

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

Despite this, feature-space manipulation is more difficult than arbitrary code modification. This is because, after manipulating the feature space, it is necessary to map the modification onto the malware's code. However, a single byte change can break the program or damage the malware's original purpose. As a result, adversaries usually cannot directly modify the raw bytes of the program file. Instead, feature-space manipulation requires finding the correct action(s) on the problem-space that will influence the feature values (but without changing run-time functionality). These actions could be, for example, adding a redundant section (*e.g.* adding a new code section without linking its address in the section table) or injecting dead code that is unreachable (*e.g.* adding a file I/O request under an always-false condition, so that the dummy code will never be executed). Note, we are not the first to explore this topic. Similar techniques have been implemented by Demetrio *et al.* [43] in a black-box optimization of adversarial Windows malware. However, they focus on the problem-space, instead of feature-space manipulation.

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

- We propose an explainability-guided and model-agnostic malware detector measurement framework (Section 4.4). Our framework generates adversarial malware while preserving the malicious functions of the malware. We exploit SHapley Additive exPlanations (SHAP) and introduce the concept of ***Accrued Malicious Magnitude (AMM)*** to guide the feature selection approach for feature-space manipulation. We further project the manipulated feature back to problem-space with a binary builder that generates adversarial samples.
- We use AMM to measure the performance of state-of-the-art malware detectors protecting against adversarial attacks (Section 4.6.1). We show that commercial antivirus engines are vulnerable to AMM-based adversarial samples, while a detector with multiple different feature extraction functions reduces the impact of the adversarial attacks in a certain degree. Experimental results indicate that our approach has significant evasion capability, which decreases the detection rates of seven malware detectors by 56.47%, and bypasses an average of 25 out of the 60 antivirus engines in VirusTotal (VT). We also present the generalizability of our AMM approach by applying it on WinPE malware detectors (Section 4.6.3).
- We explain how manipulations trained on one detector can work on another detector (*i.e.* transferability) through our explainability-guided approach (Section 4.6.2). Our explainability-guided approach shows that the transferability relies on the overlaps of features with large AMM values between different machine learning models.
- We further explore the effectiveness of our proposed attack on improved machine learning-based detectors that exclude *important* features while training (Section 4.6.4). Results show that AMM values can effectively measure the importance of features and the capability of flipping classification results. We suggest that machine learning-based AV products should consider using the AMM values to improve their performance.

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

4.2 Related Work & Motivation

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

4.2.1 Related Work and Background

Malware detectors. Many modern antivirus engines utilize rule-based analysis, such as signature matching, static unpacking, heuristics matching, and emulation techniques [74, 63]. However, rule-based antivirus engines rely heavily on expert knowledge. With the advantage of feature extraction derived from machine learning techniques, there is a flurry of work that integrates machine learning models into malware detectors [14, 74, 73, 163, 162]. We focus our evaluation on detectors that use static features due to their prevalence in providing pre-execution detection and prevention for many commercial endpoint protection solutions, such as Kaspersky [71], Avast [16], and ESET [11].

A few studies have explored the effect of obfuscations on anti-malware products, utilizing off-the-shelf tools. Maiorca *et al.* [96] and Pomilia [109] evaluated several anti-malware products using code obfuscation by a single tool. Hammad *et al.* [60] conducted a large-scale empirical study that evaluates the effectiveness of the top anti-malware products, including 7 open-source, academic, and commercial obfuscation tools. Several studies [119, 30, 131] have evaluated machine learning-based malware classifier models with the adversarial samples generated by generative adversarial networks (GANs) or automated poisoning attacks. Chen *et al.* [34, 33] studied machine learning classifiers with global robustness properties. Barbero *et al.* [17] proposed a method to cope with concept drift which may lead to performance degradation of malware detectors. Li *et al.* [82] conducted an empirical study to detect dataset shift and adversarial examples in Android malware detectors. Compared to the evaluation conducted in this chapter, the scope of these studies only covers either the rule-based products or the machine learning-based models in isolation (rather than both).

Evasion attacks against malware detectors. The goal of the evasion attacks is to generate a small perturbation for a given malware sample that results in it being misclassified. This type of attack has been extensively explored in computer vision, and previous research efforts have also investigated the applicability of such techniques to malware classification. Xu *et al.* [164] proposed a genetic programming-based approach to perform a directed search for evasive variants for PDF malware. Demetrio *et al.* [42] demonstrated that genetic programming based adversarial attacks are applicable to portable executable (PE) malware classifier. Two recent works [7, 127] also applied deep reinforcement learning to generate adversarial samples for Windows PE malware to bypass machine learning models.

SHapley Additive exPlanations (SHAP). Research into explainable machine learning has proposed multiple systems to interpret the predictions of complex models. In this chapter, we rely on SHAP [91] (based on the coalitional game theory concept of Shapley values). Hence, we briefly describe its operation. The SHAP framework subsumes several earlier model explanation techniques together, including LIME [115] and Integrated Gradients [135]. SHAP has the objective of explaining the final value of a prediction by attributing a value to each feature based on its contribution to the final result. To accomplish this task, the SHAP frameworks train a surrogate linear explanation model g of the form:

$$\begin{aligned} f(x) &= g(x'), \\ g(x') &= \phi_0 + \sum_{j=1}^{M} \phi_j x'_j, \end{aligned} \tag{4.1}$$

where f is the original model, x is the input sample to be attributed, x' is the coalition vector of x. For each entry of x', its value is 1 if the corresponding feature is "present" and 0 if "absent". $\phi_0 = \mathbb{E}_X(f(X))$ is the average prediction of the original model on sampled dataset X. The Shapley value $\phi_j \in \mathbb{R}$ is the feature attribution for the j^{th} feature x'_j to the model's decision. Summing the effects of all feature attributions approximates the difference of prediction for x and the average of the original model. Further, the SHAP framework connects LIME and Shapley values to fit Equation 4.1 and explains any machine learning-based model without internal knowledge.

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

$$\xi = \underset{g \in G}{\arg\min}\ L(f, g, \pi_x) + \Omega(g), \tag{4.2}$$

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

Shapley Additive Global importancE (SAGE) SHAP calculates the contribution of each feature to an individual prediction (local interpretability). Shapley Additive Global importancE (SAGE) [37] summarizes each feature's importance based on the predictive power it contributes across whole dataset (global interpretability). The features that are most critical for the model to make good predictions will have large values, while unimportant features will have small values. To accomplish this task, SAGE defines the restricted model f_S that only a part of the entire features set are chosen as:

$$f_S(x_S) = \mathbb{E}[f(X) \mid X_S = x_S], \tag{4.3}$$

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

$$v_f(S) = \underbrace{\mathbb{E}[\ell(f_\emptyset(X_\emptyset), Y)]}_{Mean\ prediction} - \underbrace{\mathbb{E}[\ell(f_S(X_S), Y)]}_{Using\ features\ X_S}, \tag{4.4}$$

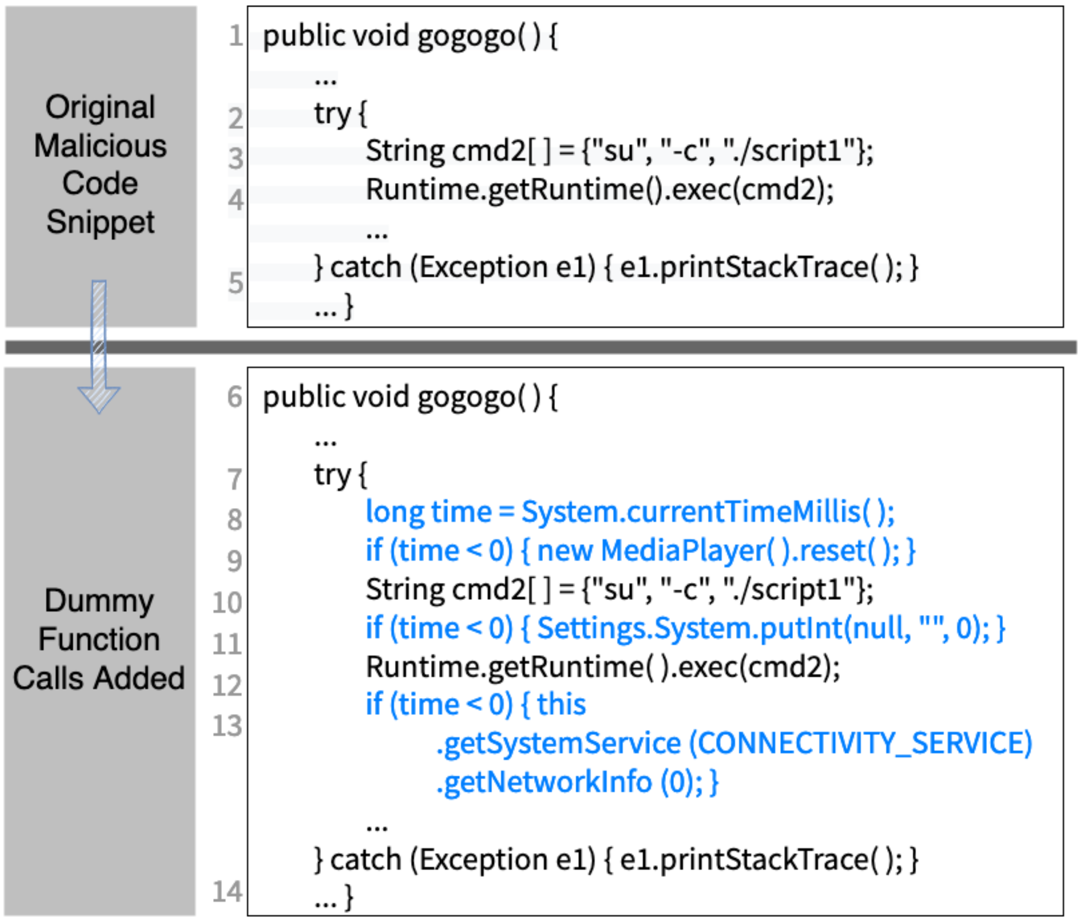
where $v_f(S)$ represents the performance of f given features X_S. As well known in game theory, Shapley values are the unique credit allocation scheme [120], and adopt this framework to attribute the model's prediction power on sets of features, we have the expression as follows:

$$\phi_i(v_f) = \frac{1}{d} \sum_{S \subseteq D \setminus \{i\}} \binom{d-1}{|S|}^{-1} \left(v_f(S \cup \{i\}) - v_f(S)\right), \tag{4.5}$$

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

4.2.2 Motivating Example

To motivate our adversarial attack, we analyze a sequence of source code decompiled from an Android malware, which is tagged as malicious by 39/67 detectors from



```
 1  public void gogogo( ) {
        ...
 2      try {
 3          String cmd2[ ] = {"su", "-c", "./script1"};
 4          Runtime.getRuntime().exec(cmd2);
            ...
 5      } catch (Exception e1) { e1.printStackTrace( ); }
        ... }
```

```
 6  public void gogogo( ) {
        ...
 7      try {
 8          long time = System.currentTimeMillis( );
 9          if (time < 0) { new MediaPlayer( ).reset( ); }
10          String cmd2[ ] = {"su", "-c", "./script1"};
11          if (time < 0) { Settings.System.putInt(null, "", 0); }
12          Runtime.getRuntime( ).exec(cmd2);
13          if (time < 0) { this
                .getSystemService (CONNECTIVITY_SERVICE)
                .getNetworkInfo (0); }
            ...
        } catch (Exception e1) { e1.printStackTrace( ); }
14      ... }
```

Figure 4.1. An example of antivirus engine evasion. Code in blue: dummy function calls.

VirusTotal (VT) [148]. From the source, we find a snippet of malicious code shown in the top part of Figure 4.1. As shown in lines 3 and 4, the malware executes a native scripts via root permissions by `su -c ./script1` command.

In order to bypass machine learning-based detectors, we must perturb its API-call-based feature space towards 'benign'. Hence, we insert several function calls with always-`false` condition closure (*e.g.* `time<0`) to ensure they are unreachable during run-time, preserving the original (malicious) functionality. These function calls are randomly selected from benign features, which involve a list of function calls extracted from benign apps, provided by an Android dataset, Drebin [14]. The inserted code is marked as blue in the middle part of Figure 4.1. After rebuilding the source code, the modified binary is identified by 33 scanners – 6 fewer than the originally, and it bypasses the machine-learning detector provided by Drebin. The remainder of this chapter develops an explainability-guided feature-space manipulation framework and problem-space rebuilding tool.

4.3 Threat Model & Problem Definition

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

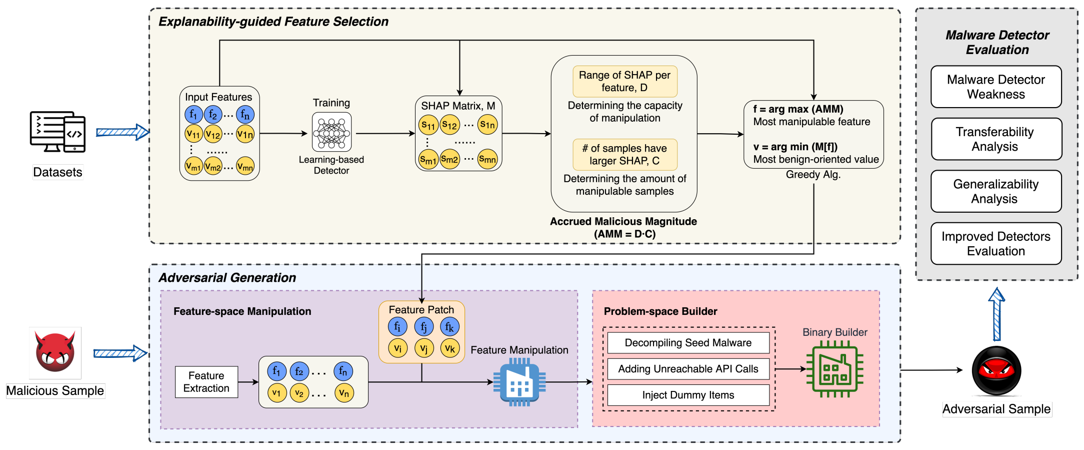

Explanability-guided Feature Selection
Datasets
Input Features
Training
Learning-based Detector
SHAP Matrix, M
Range of SHAP per feature, D
Determining the capacity of manipulation
# of samples have larger SHAP, C
Determining the amount of manipulable samples
Accrued Malicious Magnitude (AMM = D·C)
f = arg max (AMM) Most manipulable feature
v = arg min (M[f]) Most benign-oriented value
Greedy Alg.
Malware Detector Evaluation
Malware Detector Weakness
Transferability Analysis
Generalizability Analysis
Improved Detectors Evaluation
Adversarial Generation
Feature-space Manipulation
Feature Patch
Feature Manipulation
Feature Extraction
Malicious Sample
Problem-space Builder
Decompiling Seed Malware
Adding Unreachable API Calls
Inject Dummy Items
Binary Builder
Adversarial Sample

Figure 4.2. The overview of our evaluation framework.

4.3.1 Threat Model

We follow the methodology by Carlini *et al.* [20] and describe the threat model of evasion attacks against malware detectors from three aspects: the adversary's *goals*, *capabilities*, and *knowledge*.

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

Adversary capability. We assume that the adversary does not have access to the training phase or the model of the machine learning-based detectors. For instance, the adversary cannot inject poisoned data in the training dataset or manipulate any code or output of detectors. However, they will still have some basic knowledge about machine learning-based detectors, *e.g.* the access to open-source datasets, features extraction methods [118], or off-the-shelf machine-learning detectors. In addition, we introduce black-box detectors whose feature extraction functions and internal architecture are kept unknown to the adversary.

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

4.3.2 Problem Definition

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

$$F(x) = 1, x_a = \text{Gen}(x), F(x_a) = 0, \tag{4.6}$$

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

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

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

To establish such an evaluation strategy, we consider generating adversarial samples through perturbation the feature space (*e.g.* manipulating values in feature vectors) and converting the manipulation back to the problem space (*e.g.* modifying malware source code and rebuild the binary). To achieve our goal, the problem can be split into two sub-problems: (*i*) generating adversarial samples and (*ii*) evaluating them against malware detectors, both of which are detailed in Section 4.4.

4.3.3 Ethical Considerations

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

4.4 Methodology

Our research methodology consists of three key components (see Figure 4.2): (*i*) explainability-guided feature selection, to select the feature manipulation; (*ii*) an adversarial sample generator, to generate the evasive samples; and (*iii*) an evasion attack evaluation, to evaluate the proposed evasion attacks against four different machine-learning detectors, three antivirus engines and VirusTotal [148], to explain their transferability and the impact of *important* features on the accuracy. Note, transferability refers to the performance of evasion attacks when generating adversarial samples for one machine-learning model and then applying them to different detectors; *important* features refer to a set of features that a machine-learning model depends on the most.

4.4.1 Step 1: Feature Selection

In the first step of our methodology, we utilize SHapley Additive exPlanations (SHAP) to create an explainability-guided adversarial example. SHAP calculates how much one feature contributes to an individual prediction. In this step we generate SHAP values of the input dataset. The workflow of the explainability-guided feature selection is illustrated in Algorithm 1. For a set of seed malware, S, we aim to generate a corresponding adversarial sample set, A, such that they evade the target model only by modifying features.

Pre-processing. We extract features from the training samples X of a trained machine learning model m (line 2). Then the vectorized samples, X', and the model are input to shap() to calculate the SHAP value matrix M (line 3). The matrix is then used to select the most evasive features and the most benign-oriented values.

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

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

Update feature patch. After obtaining a pair of (f, v), if the selected feature f is manipulable, we add the pair into map P as the *Feature Patch* to be used in the feature-space manipulation (line 11). Although the SHAP framework can find features that impact the decision boundary, some of them cannot be manipulated directly. For example, consider the feature that counts the size of a binary, when we modify the value of another feature, the former will be modified indirectly. Therefore, the features and values we select to be manipulated follow two principles employed by the previous literature [118, 58, 57]. These principles are: (i) features are manipulable in the original problem space; and (ii) selected features have no dependencies or cannot be affected by other features.

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

Statistics-based Feature Selection Algorithm 2 illustrates the process of feature selection. First, after vecterizing the dataset, X' is divided into malicious set M' and benign set B'. Then we summarize all feature values across the samples by feature, representing the number of existence of each feature M_{sum} and B_{sum}. Then we choose the top and bottom 10% summary value as the threshold of *majority* and

Algorithm 1: AMM-based Feature-Space Selection

```
Input: Machine learning model $m$, dataset $X$, and the number of features to
       be selected $N$.
Output: Feature patch $P$.
1  $P = \text{map}(Feature, Value)$;
2  $X' \leftarrow \text{vectorize}(X)$;
3  $M \leftarrow \text{shap}(X', m)$;
4  while $size(P) < N$ do
5      $D \leftarrow \text{getRange}(M)$;
6      $C \leftarrow \text{countLarge}(M)$;
7      $AMM \leftarrow D \cdot C$;
8      $f \leftarrow \arg\max(AMM)$;
9      $v \leftarrow \arg\min(M[f])$;
10     if isManipulatable($f$) then
11         $P \leftarrow P \cup (f, v)$;
12     for each $x' \in X'$ do
13         if $x'[f] \neq v$ then
14             $idx \leftarrow \text{getIndex}(X', x')$;
15             $M \leftarrow M \setminus M[idx]$;
16             $X' \leftarrow X' \setminus x'$;
17 return $P$;
```

minority feature in benign and malicious dataset. Finally, we traverse M' and B' to find benign and malicious-oriented features B and N. In feature manipulating phase, benign-oriented features will be set as 1 in a sample while malicious ones will be set to 0.

4.4.2 Step 2: Adversarial Sample Generator

In the next step, the adversarial sample generator applies features manipulation from previous steps and generate adversarial samples. Explainability-guided feature-space manipulation involves changing features selected by Algorithm 1 to mislead the detector.

Feature-space manipulation. To train a machine learning model, the first step is to convert input data into vectors of features (*i.e.* the feature extraction process). In an evasion attack, we manipulate features to induce misclassifcations. However, not every feature has equal influence on the result of the detector, so the question becomes: how can we gain insight into a model's decision in a generic, model-agnostic way? Thus, we rely on SHAP to understand which features drive the model towards a benign classification. Guided by SHAP, we can manipulate the malware sample and cause a misclassification. Importantly, we must ensure the malware functionality is preserved. Equation 4.7 summarizes the feature-space manipulation:

$$\begin{aligned} x' &= \text{vectorize}(x), \\ a' &= \text{manipulateFeature}(x', P), \\ \text{Gen}'(x) &= \text{buildSample}(a', x), \end{aligned} \tag{4.7}$$

where x' is the result of applying feature extraction on sample x using vectorize(). The SHAP value matrix, M, is obtained through shap(), the SHAP algorithm. m

Algorithm 2: **Statistics-based Feature-Space Selection**

```
Input: Dataset $X$, labels $Y$ and the number of benign- and
       malicious-oriented features to be selected $N$.
Output: Benign-oriented features $B$ and malicious-oriented features $M$.
1  $B = \{\};$
2  $M = \{\};$
3  $X' \leftarrow \text{vectorize}(X);$
4  $M', B' \leftarrow devideDataset(X', Y);$
5  $M_{sum} \leftarrow sum(M', axis = 0);$
6  $B_{sum} \leftarrow sum(B', axis = 0);$
7  $M_{sort} \leftarrow sortDescend(M_{sum}, value > 0);$
8  $B_{sort} \leftarrow sortDescend(B_{sum}, value > 0);$
9  $m_{top} = M_{sort}[len(M') * 0.1];$
10 $m_{bottom} = M_{sort}[len(M') * 0.9];$
11 $b_{top} = M_{sort}[len(B') * 0.1];$
12 $b_{bottom} = M_{sort}[len(B') * 0.9];$
13 $d = X'.columns;$
14 for $i$ in $[0...d]$ do
15     if $B_{sum}[i] \geq b_{top}$ & $M_{sum}[i] \leq m_{bottom}$ & $size(B) < N$ then
16         $B \leftarrow B \cup (i);$
17     if $M_{sum}[i] \geq m_{top}$ & $B_{sum}[i] \leq b_{bottom}$ & $size(M) < N$ then
18         $M \leftarrow M \cup (i);$
19 return $M$, $B$;
```

represents the machine learning model. manipulateFeature() manipulates the sample in feature-space guided by SHAP. Note that, the sample generator Gen$'$() will take the manipulated feature-space sample a' and the original sample x as input, and implement the changes in feature-space back to problem-space to generate the adversarial sample, while keeping its malware functionality.

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

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

We take an Android APK as an example. Since features are a vector of boolean values representing the existence of a feature, proposed by Drebin [14], the feature value could only be modified from 0 (absence) to 1 (presence) to preserve original functionalities. We first leverage Apktool [13] to decompile an APK file into smali [125] code, a structured assembly language. API calls and network URLs are transformed to smali instruction code, which is wrapped by an unreachable disclosure, *e.g.* an always-false condition closure. The smali code is then inserted into the smali file of the main activity. Features representing Android manifest components are inserted into AndroidManifest.xml file directly. Finally, we utilize Apktool to assemble all decompiled and manipulated files into an adversarial APK sample. If a feature manipulation cannot be implemented in this way, we skip it and continue with the next most important feature.

4.4.3 Step 3: Malware Detector Evaluation

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

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

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

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

Evaluation of improved detectors. Inspired by recent research [65], a detection model can be improved by removing important features from the training phase, where the important features refer to the ones that are sensitive to the model prediction accuracy. Shapley Additive Global importancE [37] (SAGE) is a framework that measures how much a feature contributes to the prediction accuracy of a model. We utilize SAGE to improve the detection models and evaluate their performance against adversarial samples. Specifically, we first calculate SAGE values of each feature. Since the most important features are the ones with largest SAGE values, we sort the features by SAGE values in a descending order and select the top features. Then we remove the top features from samples and generate a new training set. Finally, an improved model m' is trained with the new training set. To establish a thorough comparison, we train another model that excludes top AMM-based features to compare and explain the performance of improved models.

4.5 Experiment Setup

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

4.5.1 Experimental Environment

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

4.5.2 Target Detectors

During the evaluation, we evaluate the performance of malware detectors using the proposed AMM approach, including 4 machine learning-based detectors, 3 open-sourced commercial antivirus engines, and 60 antivirus engines available at VirusTotal [148]. The detectors are summarized in Table 4.1.

To follow the convention of the prior studies [118, 14], we select 4 off-the-shelf machine learning-based malware detectors.

- **LightGBM (LGBM) [72]** is a free and open source distributed gradient boosting framework, based on the decision tree algorithm, originally developed by Microsoft.
- **Support Vector Machine (SVM)** is a set of supervised learning methods used for classification, regression and outliers detection.
- **Random Forests (RF)** is an ensemble learning method that combines a multitude of decision trees to provide classification.
- **Deep Neural Network (DNN).** In addition, we introduce a simple-structured DNN with one input layer and three fully-connected hidden layers that followed by ReLU activation function (the last one ends with a Softmax function).

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

We utilize the feature extraction function from Drebin [14] to train LGBM, SVM and RF, and we reference the feature extraction function from recent work [99] to train DNN. As the adversary knows the feature extraction function of LGBM, the SVM and RF match the *grey-box* scenario and the DNN model fits the *black-box* scenario.

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

4.5.3 Datasets and Model Training

In our experiments, we conduct evaluations on an Android dataset and machine learning-based detectors. The Android Application Package (APK) is the package file format used by the Android operating system for distribution of mobile apps. We use the well-studied Drebin [14] dataset which contains features extracted from 5,560 malicious and 123,453 benign samples. These features are represented by over 545,000 dimension `Boolean` vectors indicating whether a feature exists in an application or not. Features are categorized to 8 logical subsets representing hardware components, requested permissions, app components, filtered intents, restricted API calls,

Table 4.1. Target detectors

Name	Type	Description
LGBM	White-box	LightGBM, a tree-based classifier.
SVM	Grey-box	A linear support vector machine classifier.
RF	Grey-box	A random forest classifier.
DNN	Black-box	A feed-forward neural network with 3 hidden layers.
AV1	Black-box	An open-source antivirus engine.
AV2	Black-box	A commercial antivirus engine.
AV3	Black-box	A commercial antivirus engine.
VT	Black-box	VirusTotal, a free online service that integrates over 70 antivirus detectors. We use 60 of them.

permissions used in the source codes, suspicious API calls, and network addresses, respectively. Since the ratio of malicious and benign apps is biased, we randomly select 5,560 benign samples, making up 11,120 samples with 76,889 feature dimension, to balance the dataset. Further, we create a random 50:20:30 split of samples for training, validation, and testing set respectively to train a LightGBM model. We will discuss different models with the same dataset in Section 4.6.2. In our experiment, we evaluate adversarial samples on four machine learning models, shown in Table 4.1.

To train ML models mentioned above, we employ Androguard [10] to extract raw features from APK samples. Androguard is a python tool to analyze and manipulate Android files. It disassembles an APK file and converts its byte code and resource files into a readable and structured format. We further extract the first four types of features from the manifest file and the rest four from the Dalvik Executable (dex) file, and then these features are used to train and evaluate the machine learning based detectors mentioned above.

4.5.4 Benchmark Methods

In order to evaluate the performance of evasion attacks guided by AMM values, we introduce an intuitive *statistics-based* feature selection strategy according to the APK dataset in Section 4.5.3. In this feature selection strategy, we aim to find features (*i*) set as 1 in a major amount among *benign* samples but a minor amount among malicious samples; and (*ii*) set as 1 in a significant amount in *malicious* samples but a minority of benign samples. Features in the former represent benign-oriented features while those in the latter represent malicious-oriented features.

Basic Iterative Method [77] (BIM) and C&W [22] are two white-box setting adversarial attacks that require to access the gradients of machine learning models. They have outstanding performance of adversarial attacks in computer vision. However, they are not capable in the malware detection domain, because adversarial samples cannot be generated by perturbing each feature arbitrarily with noise, and thus the objective function of those attacks may not successfully converge. In addition, our evaluation aims to generate runnable adversarial malware samples that are effective on both machine learning-based detectors and antivirus engines. However, for antivirus engines, the attacker can only query the models without accessing the gradients of models. Therefore, we exclude BIM and C&W attacks from the evaluation.

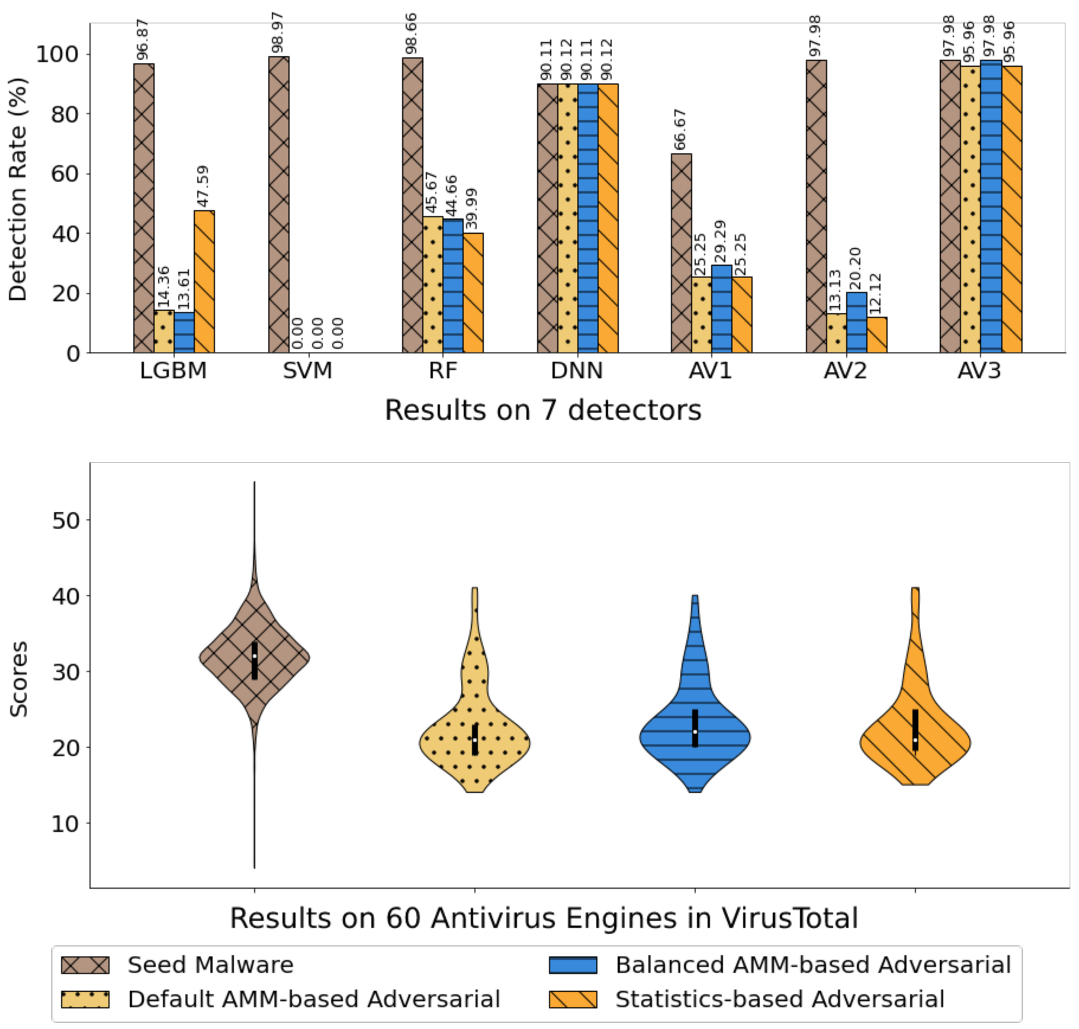


Figure 4.3. Comparison of detection rate among seed samples, adversarial samples (default AMM-based), adversarial samples (balanced AMM-based) and adversarial samples (Statistics-based).

4.6 Evaluation & Results

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

In APK machine-learning model training, the feature vector involves a sequence of 1 and 0, representing whether a specific feature exists or not. To find how SHAP-guided features affect the evasion result, we conducted another experiment to compare the detection rate of manipulating 10, 25, 50, 75, 100, 125 and 150 features on 20 sets of 100 seed samples, 2,000 samples in total. The result is shown in Figure 4.4. By manipulating more features, the detection rate decreases. The turning point of the trending comes to 75 features with 5.1% adversarial samples detected; by contrast, the rates of 100 features comes only 2.1% lower than that of 75 features. Therefore, we choose N as 75 in Algorithm 1 for APK binaries.

According to the experiment, the number of AMM-based features to manipulate is 75. However, the distribution of each feature set is not balanced, *i.e.* S_3 has the

Table 4.2. Selected feature numbers of three strategies.

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

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

largest amount of patching features as shown in Table 4.2. Therefore, to compare the performance of proposed evasion attacks, we introduce two different feature selection functions and generate three types of adversarial samples: (*i*) patching 75 features guided by the top AMM values (*i.e. default AMM-based strategy*); (*ii*) patching balanced 73 features guided by the top AMM values (*i.e. balanced AMM-based strategy*); and (*iii*) patching 300 statistics-based features(*i.e. Statistics-based strategy*).

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

4.6.1 Evaluation of Detectors

We start by evaluating our proposed evasion attack with three strategies against four machine learning-based detectors and VirusTotal (VT). As our performance metric, we focus on the difference in detection rates between the *original* samples and the *adversarial* samples. Figure 4.3 illustrates the detection rates of each strategy against each detector. The y-axis refers to the detection rates of samples, whereas each detector is presented on the x-axis. In our evaluation, we generate APK adversarial samples from the LGBM model with three strategies, listed in Table 4.2, and evaluate adversarial samples on four machine learning-based detectors, three antivirus engines and VT. All four machine learning-based detectors have reasonable accuracy (above 90%). Note that the detection rate of VT in Figure 4.3 represents the rate of how many detectors on average mark one sample as malicious out of 60 detectors in VT.
Performance against AMM-based Strategy. This strategy selects 75 features purely based on top AMM values. Comparing with the detection rate of seed malware, adversarial samples have a significant evasion performance on LGBM, SVM and RF, with detection rates of 14.36%, 0.00% and 45.67%. However, DNN, as a black-box detector with an absolutely different feature extraction function, detected 90.12% adversarial samples. Comparing with seed malware detection rate, the DNN detector

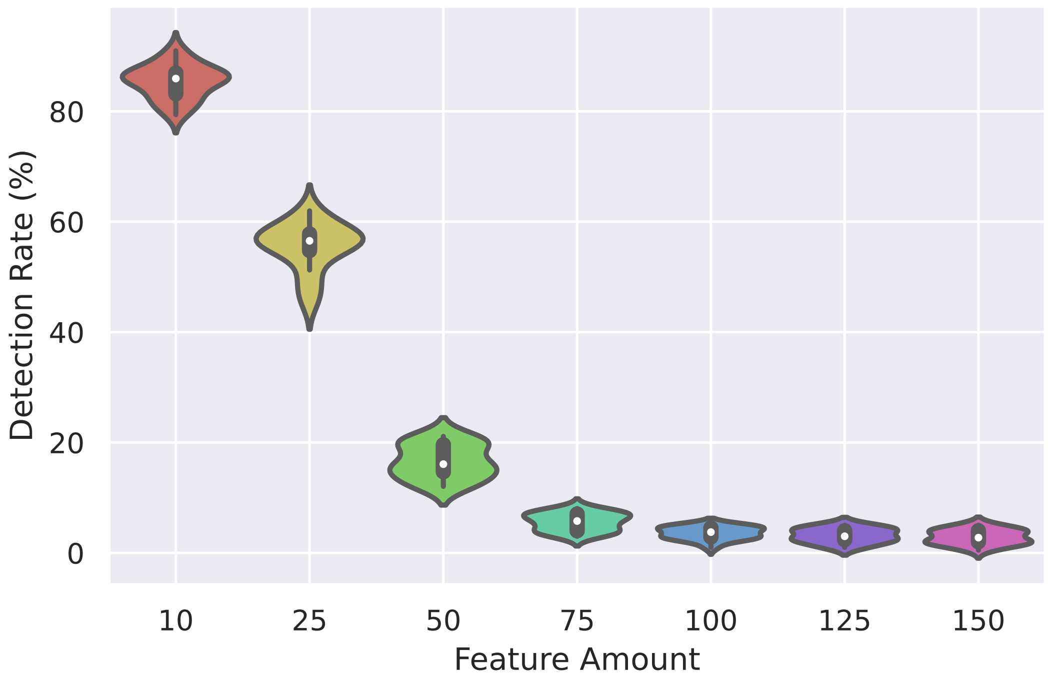


Figure 4.4. Detection rates of manipulating different sizes of feature maps on APK.

is not remarkably impacted by the adversarial attack with different feature extraction function.

Performance against Balanced AMM-based Strategy. The default AMM-based strategy selects an unbalanced amount of features in eight feature sets. Table 4.2 shows that very few features in S_1, S_3 and S_6 are selected. Therefore, we choose the top 10 features of each set from the top 1000 AMM features to compare the performance of AMM features with balanced number in different sets. However, only 7 features were found from S_1, making up 73 features in total.

Figure 4.3 illustrates that LGBM and RF detect fewer adversarial samples comparing with the default AMM-based strategy. The LGBM detects 13.61% adversarial samples while RF detects 44.66%. Meanwhile, SVM and DNN have a similar detection rate of the default AMM-based strategy (0.00% by SVM and 90.11% by DNN). The balanced AMM-based strategy has a slightly lower detection rate of adversarial samples comparing with the default AMM-based strategy.

Performance against Statistics-based Strategy. According to Algorithm 2, we select 300 features to manipulate and generate adversarial samples. From Figure 4.3, statistics-based adversarial samples have a significantly higher detection rate (47.59%) by LGBM comparing with the counterparts of other two strategies. However, RF can detect 39.99% statistics-based adversarial samples, fewer than other two counterparts. SVM and DNN detect a similar number of statistics-based adversarial samples compared with the other two strategies. Therefore, the statistics-based strategy has a better evasion performance on RF but worse on LGBM.

AV3 has the best performance on adversarial samples, with above 95% detection rates while AV1 and AV2 perform poorly. According to the scanning logs, AV3 scanned dalvik codes, manifest files, embedded resources and inner structures of native libraries while other two antivirus engines scanned the first three. This indicates that AV1 and AV2 can be impacted by our proposed adversarial strategies while AV3 may concentrate on aspects of malware other than what we have manipulated.

The average detection rate of seed malware is 56.15%, *i.e.* on average, 33.69 VT detectors mark a sample as malicious. In contrast, 37.25% of detectors mark AMM-based adversarial samples as benign, while balanced AMM-based and statistics based adversarial samples are 37.91% (23.31 detectors) and 37.32% (22.82 detectors), respectively. This result indicates that not all detectors in VT are reliable — our adversarial generation strategies effectively bypass their malware detection.

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

Summary. From the experimental results above, both white-box (LGBM) and grey-box (SVM and RF) detectors can be sufficiently evaded by adversarial samples from all three strategies. In contrast, the black-box (DNN) detector is not impacted by feature manipulation due to different feature extraction functions. This indicates that malware detectors can leverage multiple feature extraction functions and architectures to reduce the impact of adversarial evasion attacks. Noticeably, SVM has the poorest detection ability on adversarial samples, none of which are detected.

Takeaway 1:

- Both white-box and grey-box detectors are vulnerable to adversarial evasion attacks guided by AMM-based strategies.
- A detector with multiple different feature extraction functions can defend the adversarial attacks in a certain degree.
- Malware detectors in VirusTotal can be attacked by adversarial samples.

4.6.2 Transferability Analysis

Transferability is the ability for an attack to be effective against multiple learning-based detectors. To study this, we next generate AMM-based adversarial samples from SVM and RF and evaluate them on LGBM, SVM and RF, which have the same feature extraction functions. Here, we seek to understand how a manipulation guided by one detector performs against the other detectors. To unveil what causes the transferability, we evaluate LGBM, SVM and RF, since they share the same feature extraction functions.

We observe the earlier results in Figure 4.3 and find that the AMM-guided approaches have a notable performance of evading detection across LGBM, SVM and RF. Recall that, according to Algorithm 1, we select the features with highest AMM values as the most evasive features to conduct feature-space manipulation. In this section, we analyze how the attack transfer to other detectors in two aspects: feature overlaps and detection rates.

Feature overlap. To explore the reason why our proposed attack can transfer across detectors, we present the AMM values of the top features across each dataset in a heatmap, shown in Figure 4.5. As the number of features differs across each dataset, we only display the top 1024 features that have the highest AMM values in LGBM, the generation model. In each sub plot, we present the 1024 features as 32 rows by 32 columns of dots (normalized to $[0, 1]$), where the darker dots indicate higher values of AMM (which indicates a greater possibility to be selected a feature to be manipulated). Further, we sort the features in the original dataset according to the AMM values in descending order. Therefore, darker dots scattered in the upper zone

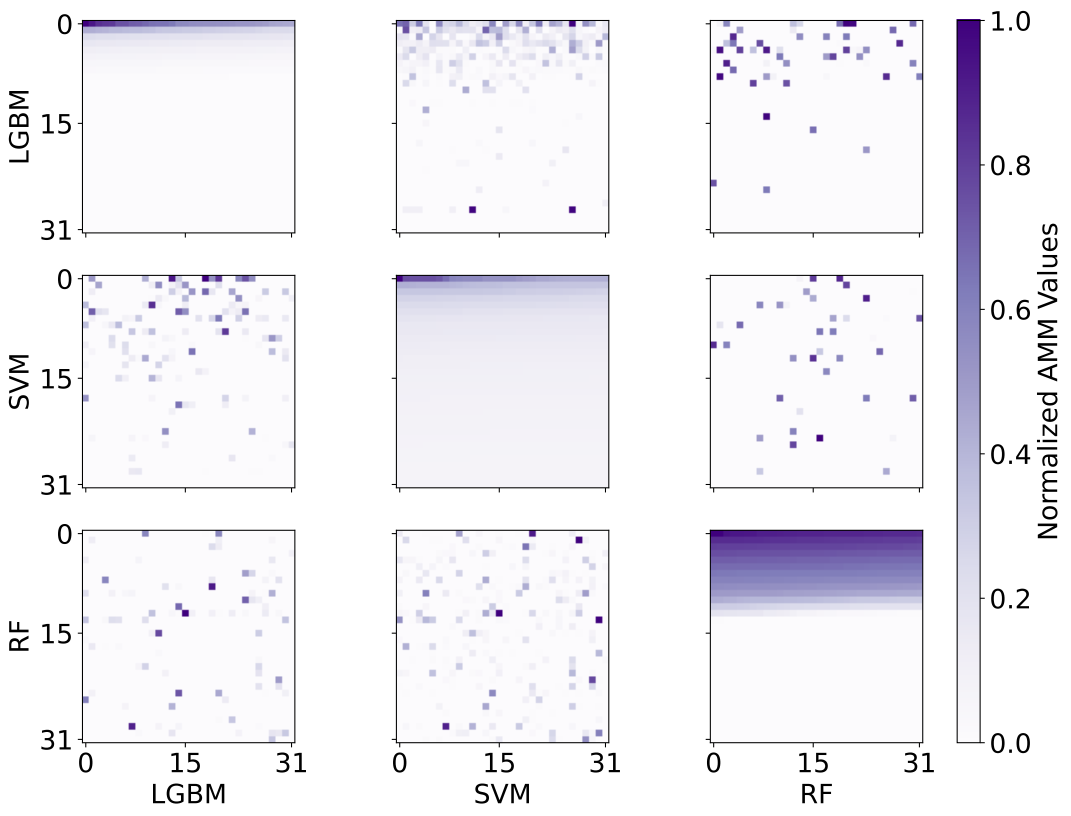


Figure 4.5. AMM values of features in different datasets and models. Each heatmap contains normalized AMM values of 1,024 features that are selected according to the adversarial generation model.

of the nine subplots indicate that there are more features having been selected across detectors.

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

Detection rate. We can also evaluate transferability of adversarial samples generated by inspecting the detection rates. In this experiment, we generated default AMM-based adversarial samples generated from LGBM, SVM and RF models. Figure 4.6 shows the detection rates of adversarial samples across machine learning detectors. The y-axis represents each dataset with three generating models, and the x-axis shows the target detectors. In the figure, a darker color indicates a higher detection rate (representing lower transferability).

Figure 4.6 shows that adversarial samples generated from LGBM and RF can effectively transfer the evasion attacks across three detectors. Adversarial samples from SVM have poor performance on the RF detector. Overall the SVM detector has the worst detection performance on adversarial samples (*i.e.* 0.00% of adversarial samples are detected), while the RF detector performs the best.

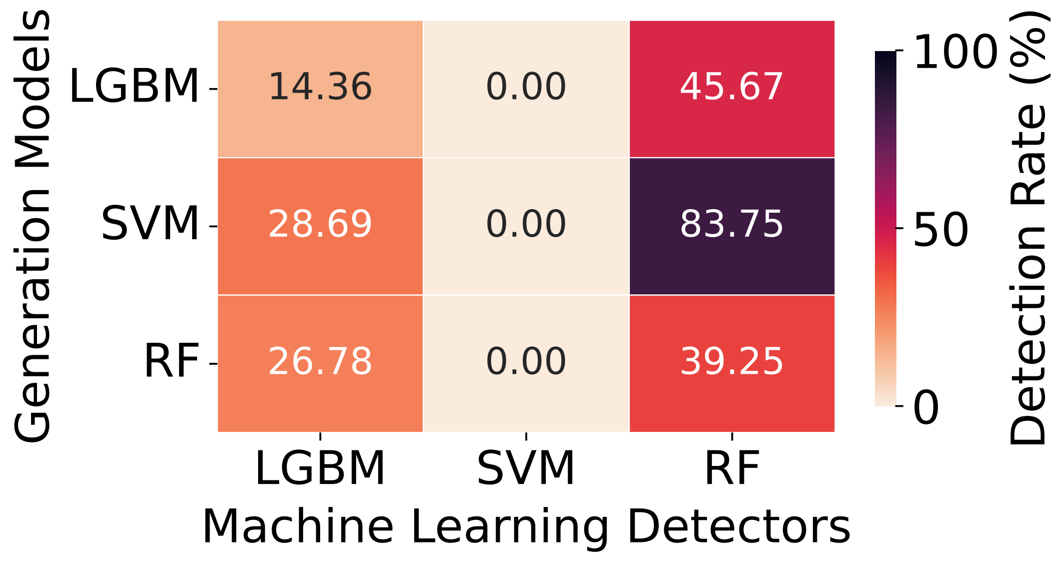


Figure 4.6. Detection rate of proposed adversarial samples generated from LGBM, SVM and RF against three detectors.

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

4.6.3 Generalizability Analysis

To examine if our evasion attack can generalize to other operating systems, we next test how effective our evasion attack is on Windows Portable Executable (WinPE).
WinPE. The Portable Executable (PE) format is the standard file format for executables, object code, and Dynamic Link Libraries (DLLs) used in 32- and 64-bit versions of the Windows operating systems. We use SOREL-20M [61] as the WinPE dataset in our experiment. SOREL-20M is a representative public dataset of malicious and benign WinPE samples used for malware classification, consisting of 2,381-dimensional feature vectors extracted from 9,470,626 benign and 9,919,251 malicious samples, as well as corresponding malicious binaries. It leverages the feature extraction function from Ember [8] and provides larger dataset. We randomly choose 10,000 benign and 10,000 malicious samples to train LGBM, SVM, RF and DNN with the same feature extraction function.
Feature manipulation. Previous work [118] shows that only 17 features can be modified directly and indirectly to preserve the functionality of WinPE binaries. Hence, we leverage LIEF [140] to extract features, and pefile [104] to apply feature manipulation on the WinPE binaries.
Results. In the experiment, we only evaluate the default AMM-based strategy. This is because the WinPE has fewer features to manipulate, and its feature structures are different from APK features in Drebin. The results are shown in Figure 4.7. Our proposed strategy has a remarkable evasion performance on LGBM, SVM and RF detectors. Since most WinPE features correlate with each other, the method of parsing and generating WinPE binaries (*i.e.* directly modifying values and adding empty sections) may negatively affect the performance of our proposed attack, illustrated

in DNN. In a nutshell, the test result shows that our proposed evasion attack is also effective on Windows.

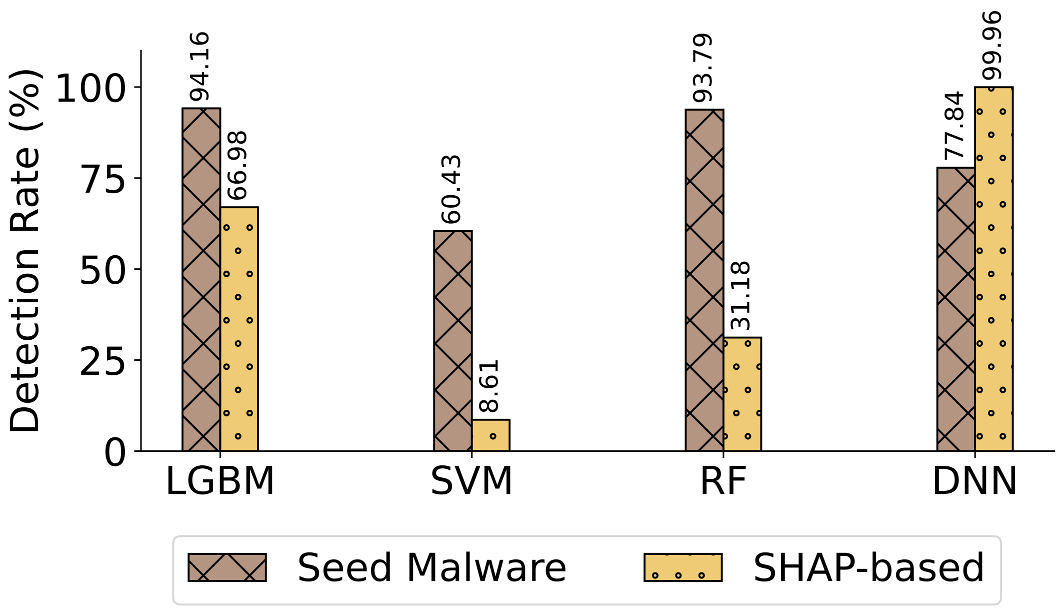


Figure 4.7. Detection rates of WinPE seed malware and AMM-based adversarial samples detected by four detectors

4.6.4 Revisiting AMM with Improved Detectors

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

Improving detectors. Our evaluation shows that machine learning-based detectors are vulnerable to adversarial attacks guided by AMM values. Therefore, we seek to improve the performance of existing detector so that adversarial samples can be identified. We follow the methodology in Section 4.4.3 and generate an improved LGBM detector m_i by excluding SAGE-based important features in the training phase. The improved model m_i aims to evaluate AMM-based adversarial samples S_a. In order to measure the effectiveness of AMM-based and SAGE-based features, we also generate another improved detector m_a by excluding AMM-based features and generate adversarial samples S_i by applying SAGE-based important features.

To understand how many important features can noticeably improve the detection ability against adversarial samples, we generated new machine learning detectors with different number of important features excluded from the training set. The result is shown in Figure 4.8. The y-axis refers to the model accuracy and detection rates of seed malware and AMM-based samples, whereas the numbers of features excluded from models are presented on the x-axis. From the result, we found that excluding 220 important features allows the new detector to perform similar detection rates of original malware (80.56%) and adversarial samples (80.56%). Therefore, we employ 220 important features in the following experiment.

Measuring detectors. Considering the ability for detectors to improve their accuracy by removing features, we next experiment with two new evasion attacks based on removing features using AMM: (i) manipulating the top AMM-based features to

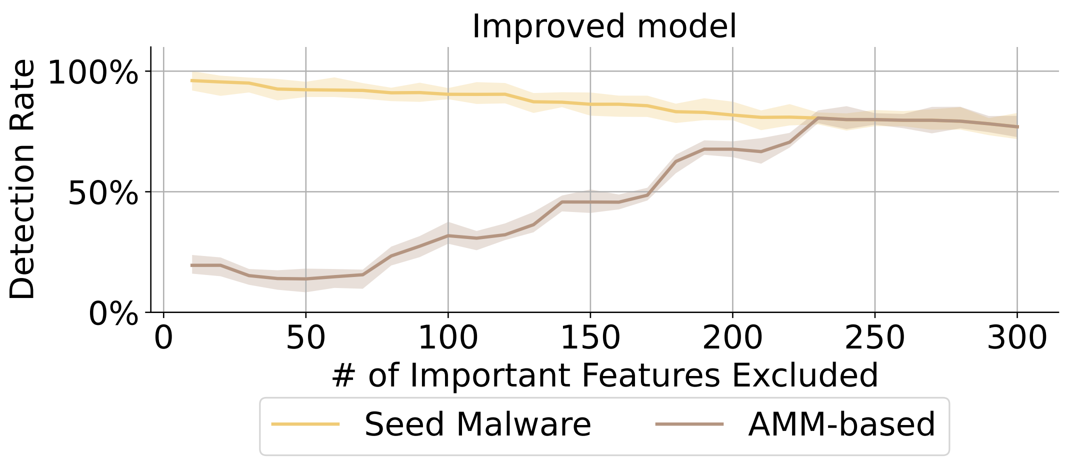


Figure 4.8. Detection rates of seed malware and AMM-based adversarial samples detected by models excluding different amount of important features.

generate adversarial samples S_a, and generating an improved LGBM detector m_i by excluding important features; (ii) manipulating the top important features to generate adversarial sample S_i, and generating an improved LGBM detector m_a by excluding AMM-based features. To evaluate AMM and SAGE in the same conditions, we use the same parameters: the top 75 features for adversarial samples and the top 220 features for improved detectors. Meanwhile we introduce the seed samples S_m and the original LGBM detector as a benchmark. In the experiment, we leverage 5,459 seed malwares to generate adversarial samples where we randomly select 300 samples for 20-round tests on each detector.

Figure 4.9 presents box plots of the detection rate and prediction rate. Recall, the prediction value defines the confidence that a sample is malware. We present results for three malware datasets: seed malware S_m, adversarial sample S_a and S_i. We define a sample as malicious when the prediction value is greater than or equals to 0.5. On the original detector, less than 15% of samples in both S_a and S_i are classified as malicious. On m_i, the average detection rate of the S_a samples increase to 67.3% while their prediction values range from 0.23 to 0.82. In contrast, the detection rates of S_i on m_a are around 78.2%, while their prediction values range from 0.69 to 0.91, which means that m_a can detect more adversarial samples than m_i.

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

Comparison of AMM and SAGE. Next, we compare how AMM-based and important features impact the detection. Figure 4.10 illustrates the feature distribution of AMM and SAGE values and the amount of evaded samples manipulating the corresponding features. The X-axis indicates normalized AMM values; the Y-axis is normalized SAGE values. As shown in the figure, AMM-based features are more centrally located on low SAGE-value area while important features are more likely to have small AMM values. They have a big portion of common features, most of which have small SAGE values but with large AMM values. This result indicates that evaded samples are more likely to have modified features with large AMM values. In addition, these features have small SAGE values so that they are less likely to downgrade the performance of detectors but more to flip prediction labels.

Summary. The results show that our proposed AMM framework can guide us to

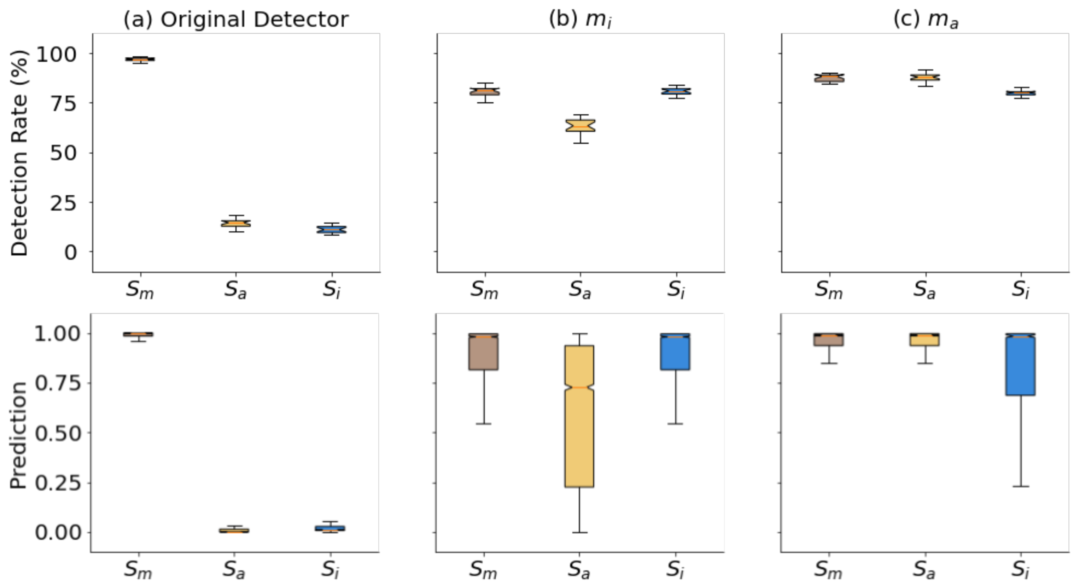


Figure 4.9. Precisions and detection rates of seed malware S_m, adversarial samples S_a (AMM-based features) and adversarial samples S_i (important features) on the original model and two updated model m_i and m_a.

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

Takeaway 3:

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

Table 4.3. APK feature IDs and their description

Feature ID	Description
46227	receiver::com.google.android.apps.analytics.AnalyticsReceiver
40271	call::Landroid/media/MediaPlayer;->reset()V
39950	api_call::Landroid/app/Service;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;
40960	intent::android.intent.action.VIEW
43959	url::maps.google.com

4.7 Case Studies

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

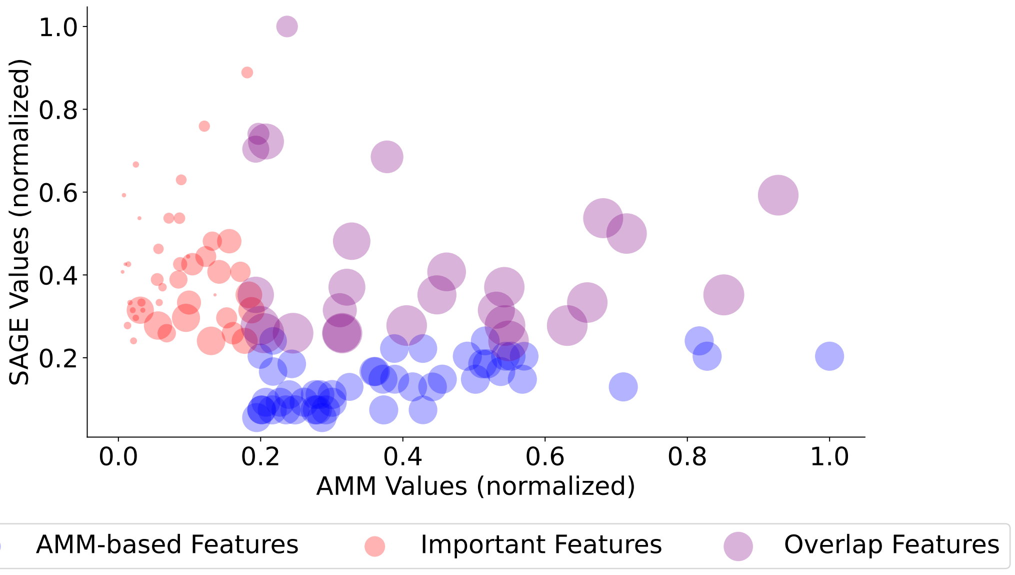


Figure 4.10. SAGE and AMM value distributions of important features and AMM-based features.

4.7.1 A Case Study on Evasion Capability

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

First we use *Sample 1* and *Sample 2* to manipulate different numbers of features (guided by Algorithm 1) and compare their prediction values. We consider a sample malicious when its prediction value is larger than 0.5 (benign otherwise). Initially, the prediction values of *Sample 1* and *Sample 2* are 0.99975447 and 0.9997253, and raw scores given by LGBM are 8.31183171 and 8.19956212. After manipulating 75 selected features, the prediction value of *Sample 1* turns to 0.28416445 (*i.e.* benign) with a raw score of -0.92389732 . In contrast, the prediction value of *Sample 2* remains positive (*i.e.* malicious) at 0.91948969 with raw score 2.43543351.

We further generate SHAP values of the original and adversarial samples (with $N = 75$ features selected) of *Sample 1* and *Sample 2* to analyze the impact of the manipulated features. The results are shown in Figure 4.11. The subfigures labeled (A) and (B) indicate the original and adversarial samples of *Sample 1*, respectively. Further, (C) and (D) show results for for *Sample 2*. The X-axis indicates the prediction value; the Y-axis is the feature ID. $f(x)$ and $f(x_a)$ are raw scores of the original and adversarial samples given by the LGBM detector. We use red bars to indicate postive SHAP values, and blue bars to highlight negative SHAP values of each feature (ϕ_j in Equation 4.1). Feature IDs shown in the figure are parts of manipulated features that had the greatest impact on *Sample 1*.

Figure 4.11(b) shows that the SHAP values of manipulated features change significantly towards negative (blue bars), thereby pushing the output, $f(x_a)$, towards negative. In contrast, the SHAP values of the features of *Sample 2*, shown in Figure 4.11(d), change far less. Specifically, only features 46227, 40271, 39950 and 73533

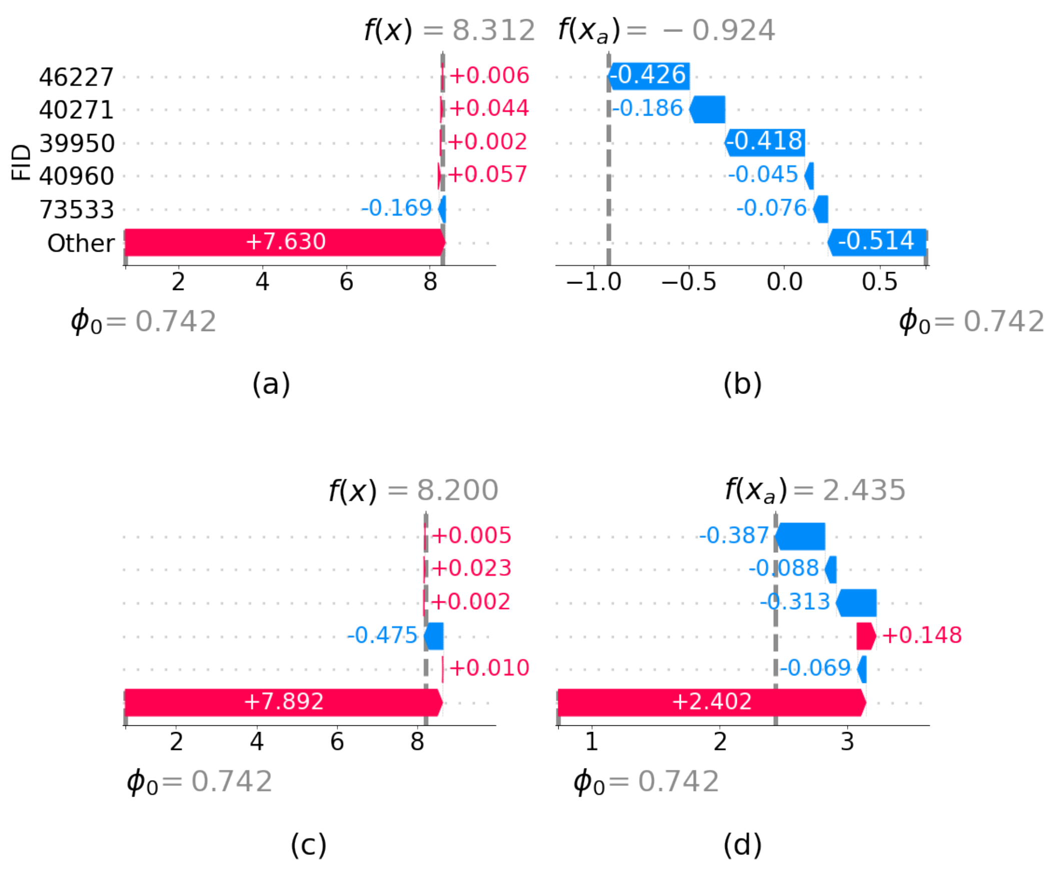


Figure 4.11. SHAP values of two APK samples before and after adversarial generation. (a) and (b) are the original and adversarial samples of *Sample 1*; (c) and (d) are of *Sample 2*. Details of features can be found in Table 4.3.

are manipulated towards negative while 40960 is not. This means that we cannot manipulate enough features to force the decision making towards benign for *Sample 2*. This result indicates that the manipulated features have limited impact on *Sample 2* to invert the result from malicious to benign. If we increase the number of selecting features to 290, *Sample 2* is then identified as benign. Therefore, the detection result depends on how many features have malicious-oriented values that we can manipulate in the sample. However, infinitely increasing the number of selecting features would also lead to heavily computational load and decrease the efficiency of attack.

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

4.7.2 A Case Study on Detection Efficiency

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

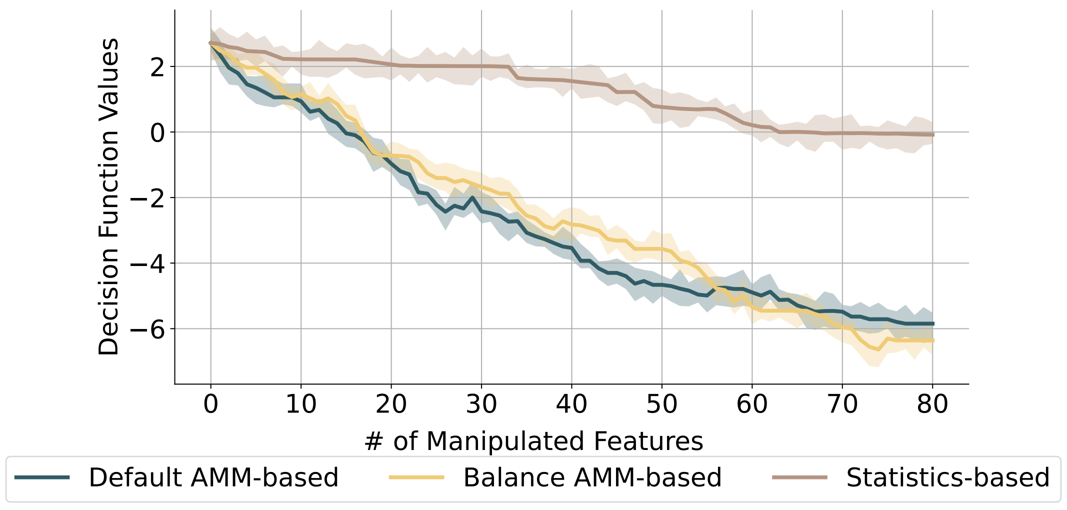


Figure 4.12. Decision function values of samples with different numbers of features manipulated with AMM-based, balanced AMM-based and Statistics-based strategies

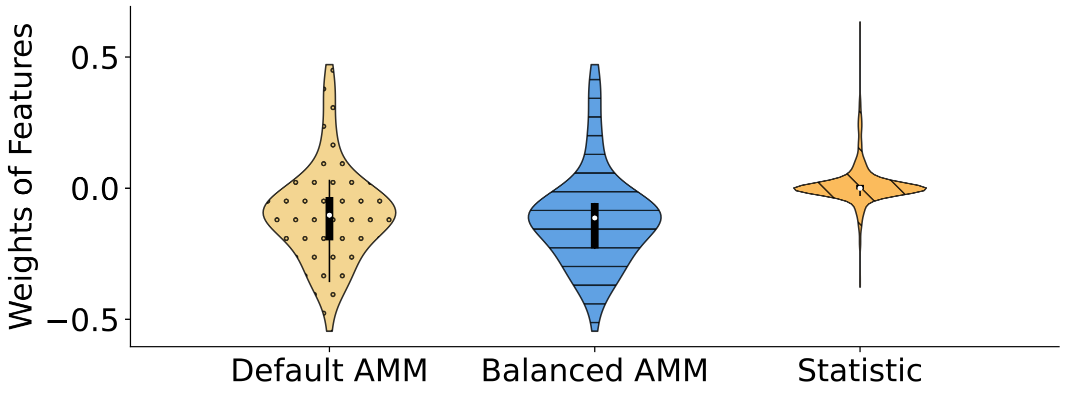


Figure 4.13. Distribution of weights of features in SVM models trained by samples generated from different strategies.

First we generate adversarial samples by manipulating from 1 to 80 features. We then calculate their decision function values from the SVM. The decision function value represents whether a predicted sample by the classifier occupies the side of and the distance to the Hyperplane. We still use features selected from LGBM and validate them on SVM. Results are shown in Figure 4.12, where the Y-axis represents the decision function values of samples guided-by AMM-based, Balanced AMM-based and Statistics-based strategies. The X-axis shows the number of features manipulated. Positive decision function values represent malicious results while negative ones represent benign results.

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

From the result, we speculate that AMM-guided features occupy large weights of SVM classification. To verify this, we export the weights to each feature in the SVM detector and compare weight values of selected features. Figure 4.13 illustrates weight values of selected features, where the y-axis represents the weight of each feature, and the x-axis shows the indexes of selected features. We can see that most weight values of features selected by the statistics-based strategy are close to 0, while most weight values of features selected by other two strategies have relatively large negative values, making the prediction decision values to be negative (*i.e.* benign). We also compared selected features from LGBM with SVM features from top AMM values and found that most of LGBM selected features has large AMM values in SVM, which matches the result in Figure 4.5.

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

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

4.8 Discussion & Limitations

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

Learning models. Our experiments are conducted on three existing models followed by the prior research [118, 14], including SVM, LightGBM and a feed-forward neural network. These models, however, are trained with the default configuration and parameters from each dataset. This does not preclude the possibility that alternative models and configurations may gain superior performance.

Adaptive defenses. Existing defenses against adversarial generation, for example, adversarial training [95] and differential privacy (DP) [1, 47, 79], may not be effective against our proposed evasion attack. For adversarial training, it is because that our attack is based on the transferability of important features among different learning-based detectors, in lieu of optimization tricks, *e.g.* FGSM [55]. On the other hand, DP-based robust machine learning techniques cannot defend against our attacks, because unbounded random perturbations may break the generated samples' functionality. One possible adaptive defense is to analyze feature distribution of queries before we carry out *de facto* malware detection. For example, Slack *et al.* [124] propose an adversarial classification approach to fool explanation methods, *e.g.* SHAP and LIME. They offer a different classifier, *e.g.* an extremely biased one, along with the original one to the input according to the perturbation distribution. However, this classification approach can be an attack target and cannot always boost the detection accuracy.

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

4.9 Conclusion

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

Chapter 5

Conclusions and Future Work

This thesis focuses on researching malware-related security issues, including malicious behaviours and bypassing detection by mobile malware. This thesis reviews three major topics of machine learning-based malware detection research, introduces a semi-automatic static analysis framework and proposes an explainability-guided adversarial malware variants generation and detector measurement approach. The semi-automatic static analysis framework unveiled potential privacy leakage issues of Android clipboard data. The explainability-guided approach offers the concept of *Accrued Malicious Magnitude (AMM)* to find the most vulnerable features for machine learning-based detectors, which can guide both adversarial attacks and functionality assessment.

The research finds that current malware detection approaches still remain unexplainable issues. For instance, the issues include how classifiers cluster malware samples into specific classes and find the attributes representing specific classes. Recent computer vision research has proposed style-based approaches to generate adversarial examples and conduct adversarial attacks [70, 78, 121]. However, due to the significant differences between computer vision and malware detection, it is difficult to apply a style-based approach on malware-related research directly. Due to the limited candidature time for the master's degree, this research remains a preliminary study. This thesis has already guided future research that a style-based explanation and feature perturbation approach is available in malware-related research.

Appendix A

Appendix

A Structures of WinPE Binaries

Figure A.1 illustrates the structure of a WinPE binary. Specifically, a WinPE binary involves five parts:

- **DOS Header** is a legacy header from DOS era to maintain compatibility with legacy Windows systems.
- **PE Header** involves general information including the target architecture, number of sections and symbols, timestamp, and the header size.
- **Optional Header** contains detailed information required by the system to load, such as the entry-point address, `dll` characteristics, size of file, and version information.
- **Section Table** is a list of section headers storing the section name, address, relocations, and other general information.
- **Sections** contains contiguous chunks of bytes hosting the real content of the executable. For example, `.text` stores the code, `.data` stores global variables, and `.rdata` stores read-only constants and counting.

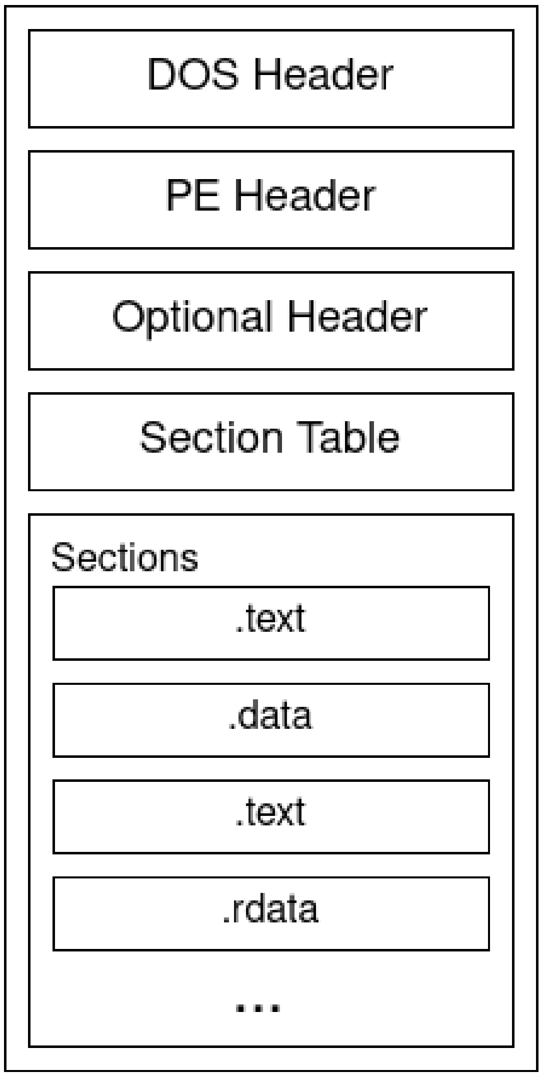


Figure A.1. Structure of WinPE files.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. “Deep learning with differential privacy”. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 2016.

[2] Lawrence Abrams. *First CryptoCurrency Clipboard Hijacker Found on Google Play Store*. 2019. URL: https://www.bleepingcomputer.com/news/security/first-cryptocurrency-clipboard-hijacker-found-on-google-play-store/.

[3] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. *Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification*. 2016. arXiv: 1511.04317 [cs.CR].

[4] *AI and machine learning*. URL: https://www.avast.com/en-us/technology/ai-and-machine-learning.

[5] *AI and Machine Learning | AVAST*. URL: https://www.avast.com/technology/ai-and-machine-learning#mac.

[6] Naveed Akhtar and Ajmal Mian. “Threat of adversarial attacks on deep learning in computer vision: A survey”. In: *IEEE Access* (2018).

[7] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. “Learning to evade static PE machine learning malware models via reinforcement learning”. In: *arXiv preprint arXiv:1801.08917* (2018).

[8] Hyrum S Anderson and Phil Roth. “Ember: An open dataset for training static PE malware machine learning models”. In: *arXiv preprint arXiv:1804.04637* (2018).

[9] *Android Malware Genome Project*. URL: http://www.malgenomeproject.org/.

[10] Anthony Desnos and Geoffroy Gueguen and Sebastian Bachmann. *AndroGuard*. URL: https://github.com/androguard/androguard.

[11] *Antivirus and Internet security | ESET*. URL: https://www.eset.com/.

[12] Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, and Alessio Merlo. “Obfuscapk: An open-source black-box obfuscation tool for Android apps”. In: *SoftwareX* (2020).

[13] *Apktool - A tool for reverse engineering Android APK files*. URL: https://ibotpeaches.github.io/Apktool/.

[14] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. “Drebin: Effective and explainable detection of Android malware in your pocket”. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2014.

[15] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps". In: *ACM SIGPLAN Notices* (2014).

[16] *Avast.* URL: https://www.avast.com.

[17] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. "Transcending transcend: Revisiting malware classification with conformal evaluation". In: *Proceedings of the IEEE Symposium on Security and Privacy (SP).* 2022.

[18] *Bitcoin - open source P2P money.* URL: https://bitcoin.org/en/.

[19] Olivia Byrnes, Wendy La, Hu Wang, Congbo Ma, Minhui Xue, and Qi Wu. "Data hiding with deep learning: A survey unifying digital watermarking and steganography". In: *arXiv preprint arXiv:2107.09287* (2021).

[20] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. "On evaluating adversarial robustness". In: *arXiv preprint arXiv:1902.06705* (2019).

[21] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. "The secret sharer: Evaluating and testing unintended memorization in neural networks". In: *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19).* 2019, pp. 267–284.

[22] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *Proceedings of the IEEE Symposium on Security and Privacy (SP).* 2017.

[23] Alvin Chan, Lei Ma, Felix Juefei-Xu, Yew-Soon Ong, Xiaofei Xie, Minhui Xue, and Yang Liu. "Breaking neural reasoning architectures with metamorphic relation-based adversarial examples". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[24] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. "Explaining image classifiers by counterfactual generation". In: *arXiv preprint arXiv:1807.08024* (2018).

[25] Lingwei Chen, Tao Li, Melih Abdulhayoglu, and Yanfang Ye. "Intelligent malware detection based on file relation graphs". In: *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015).* IEEE. 2015, pp. 85–92.

[26] Liuqiao Chen, Hu Wang, Benjamin Zi Hao Zhao, Minhui Xue, and Haifeng Qian. "Oriole: Thwarting privacy against trustworthy deep learning models". In: *Proceedings of the Australasian Conference on Information Security and Privacy.* Springer. 2021.

[27] Sen Chen, Lingling Fan, Chunyang Chen, Minhui Xue, Yang Liu, and Lihua Xu. "Gui-squatting attack: Automated generation of android phishing apps". In: *IEEE Transactions on Dependable and Secure Computing* (2019).

[28] Sen Chen, Lingling Fan, Guozhu Meng, Ting Su, Minhui Xue, Yinxing Xue, Yang Liu, and Lihua Xu. "An empirical assessment of security risks of global android banking apps". In: *arXiv preprint arXiv:1805.05236* (2018).

[29] Sen Chen, Ting Su, Lingling Fan, Guozhu Meng, Minhui Xue, Yang Liu, and Lihua Xu. "Are mobile banking apps secure? What can be improved?" In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018.

[30] Sen Chen, Minhui Xue, Lingling Fan, Lei Ma, Yang Liu, and Lihua Xu. "How can we craft large-scale Android malware? An automated poisoning attack". In: *Proceedings of the International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*. 2019.

[31] Sen Chen, Minhui Xue, Zhushou Tang, Lihua Xu, and Haojin Zhu. "Stormdroid: A streaminglized machine learning-based system for detecting Android malware". In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. 2016, pp. 377–388.

[32] Thomas M Chen and Jean-Marc Robert. "The evolution of viruses and worms". In: *Statistical methods in computer security* (2004).

[33] Yizheng Chen, Shiqi Wang, Yue Qin, Xiaojing Liao, Suman Jana, and David Wagner. "Learning security classifiers with verified global robustness properties". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021.

[34] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. "On training robust {PDF} malware classifiers". In: *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 2343–2360.

[35] *ClamAVNet*. URL: https://www.clamav.net/.

[36] Ian Covert. *Explaining machine learning models with SHAP and SAGE*. URL: https://iancovert.com/blog/understanding-shap-sage/.

[37] Ian Covert, Scott Lundberg, and Su-In Lee. *Understanding Global Feature Contributions With Additive Importance Measures*. 2020. arXiv: 2004.00668 [cs.LG].

[38] Matthew Crawford, Wei Wang, Ruoxi Sun, and Minhui Xue. "Statically Detecting Adversarial Malware through Randomised Chaining". In: *arXiv preprint arXiv:2111.14037* (2021).

[39] *Cyber security statistics*. URL: https://purplesec.us/resources/cyber-security-statistics/.

[40] Piotr Dabkowski and Yarin Gal. "Real time image saliency for black box classifiers". In: *Proceedings of the International Conference on Neural Information Processing Systems*. NIPS. 2017.

[41] Santanu Kumar Dash, Guillermo Suarez-Tangil, Salahuddin Khan, Kimberly Tam, Mansour Ahmadi, Johannes Kinder, and Lorenzo Cavallaro. "Classifying android malware based on runtime behavior". In: *Proceedings of the 2016 IEEE Security and Privacy Workshops*. 2016.

[42] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. "Efficient black-box optimization of adversarial windows malware with constrained manipulations". In: *arXiv preprint arXiv:2003.13526* (2020).

[43] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. "Functionality-preserving black-box optimization of adversarial windows malware". In: *IEEE Transactions on Information Forensics and Security* (2021).

[44] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. “Yes, machine learning can be more secure! A case study on Android malware detection”. In: *IEEE Transactions on Dependable and Secure Computing* (2017).

[45] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. “Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks”. In: *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*. 2019.

[46] Bao Gia Doan, Minhui Xue, Shiqing Ma, Ehsan Abbasnejad, and Damith C Ranasinghe. “TnT Attacks! Universal Naturalistic Adversarial Patches Against Deep Neural Network Systems”. In: *arXiv preprint arXiv:2111.09999* (2021).

[47] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. “Calibrating noise to sensitivity in private data analysis”. In: *Proceedings of the Theory of Cryptography, Third Theory of Cryptography Conference (TCC)*. 2006.

[48] *etherum.org*. URL: https://ethereum.org/en/.

[49] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. “Robust physical-world attacks on deep learning visual classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[50] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. “Understanding deep networks via extremal perturbations and smooth masks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.

[51] Ruth C Fong and Andrea Vedaldi. “Interpretable explanations of black boxes by meaningful perturbation”. In: *Proceedings of the 2017 IEEE international conference on computer vision*. 2017.

[52] *Frida - Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers*. URL: https://frida.re/.

[53] *Frida Overview*. URL: https://frida.re/docs/home/.

[54] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* (2014).

[55] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *International Conference on Learning Representations* (2015).

[56] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. “Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence”. In: *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. 2015.

[57] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. “Adversarial examples for malware detection”. In: *Proceedings of the European Symposium on Research in Computer Security*. 2017.

[58] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. “Adversarial perturbations against deep neural networks for malware classification”. In: *arXiv preprint arXiv:1606.04435* (2016).

[59] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. "LEMNA: Explaining Deep Learning Based Security Applications". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. 2018.

[60] Mahmoud Hammad, Joshua Garcia, and Sam Malek. "A large-scale empirical study on the effects of code obfuscations on Android apps and anti-malware products". In: *Proceedings of the International Conference on Software Engineering*. 2018, pp. 421–431.

[61] Richard Harang and Ethan M. Rudd. "SOREL-20M: A large scale benchmark dataset for malicious PE detection". In: *arXiv preprint arXiv:2012.07634* (2020).

[62] Yongzhong He, Xuejun Yang, Binghui Hu, and Wei Wang. "Dynamic privacy leakage analysis of Android third-party libraries". In: *Journal of Information Security and Applications* (2019).

[63] *How anti-virus software works*. URL: https://cs.stanford.edu/people/eroberts/cs201/projects/viruses/anti-virus.html.

[64] Aoting Hu, Renjie Xie, Zhigang Lu, Aiqun Hu, and Minhui Xue. "TableGAN-MCA: Evaluating Membership Collisions of GAN-Synthesized Tabular Data Releasing". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. 2021.

[65] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Adversarial examples are not bugs, they are features". In: *Advances in neural information processing systems* (2019).

[66] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. "Neurlux: Dynamic Malware Analysis without Feature Engineering". In: *Proceedings of the 35th Annual Computer Security Applications Conference*. 2019.

[67] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. "Transcend: Detecting concept drift in malware classification models". In: *Proceedings of the 26th USENIX Security Symposium (USENIX Security 17)*. 2017.

[68] Jinho Jung, Chanil Jeon, Max Wolotsky, Insu Yun, and Taesoo Kim. "AVPASS: Leaking and bypassing antivirus detection model automatically". In: *Black Hat USA Briefings (Black Hat USA)*. 2017.

[69] L. P. Kaelbling, M. L. Littman, and A. W. Moore. *Reinforcement Learning: A Survey*. 1996. arXiv: cs/9605103 [cs.AI].

[70] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[71] *Kaspersky cyber security solutions for home & business*. URL: https://www.kaspersky.com.au/.

[72] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "LightGBM: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems* (2017).

[73] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. "A multimodal deep learning method for Android malware detection using various features". In: *IEEE Transactions on Information Forensics and Security* (2018).

[74] Dhilung Kirat and Giovanni Vigna. “MalGene: Automatic extraction of malware analysis evasion signature”. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 2015.

[75] *Kiteshield*. URL: https://github.com/GunshipPenguin/kiteshield.

[76] Platon Kotzias, Juan Caballero, and Leyla Bilge. “How did that get in my phone? unwanted app distribution on android devices”. In: *Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 53–69.

[77] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2017. arXiv: 1607.02533 [cs.CV].

[78] Oran Lang, Yossi Gandelsman, Michal Yarom, Yoav Wald, Gal Elidan, Avinatan Hassidim, William T Freeman, Phillip Isola, Amir Globerson, Michal Irani, et al. “Explaining in Style: Training a GAN to explain a classifier in StyleSpace”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 693–702.

[79] Mathias Lécuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. “Certified robustness to adversarial examples with differential privacy”. In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. 2019.

[80] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median”. In: *Journal of experimental social psychology* (2013).

[81] Deqiang Li, Tian Qiu, Shuo Chen, Qianmu Li, and Shouhuai Xu. “Can We Leverage Predictive Uncertainty to Detect Dataset Shift and Adversarial Examples in Android Malware Detection?” In: *Proceedings of the Annual Computer Security Applications Conference*. 2021, pp. 596–608.

[82] Deqiang Li, Tian Qiu, Shuo Chen, Qianmu Li, and Shouhuai Xu. “Can we leverage predictive uncertainty to detect dataset shift and adversarial examples in Android malware detection?” In: *Annual Computer Security Applications Conference (ACSAC)* (2021).

[83] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. “Iccta: Detecting inter-component privacy leaks in android apps”. In: *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE. 2015, pp. 280–291.

[84] Shaofeng Li, Hui Liu, Tian Dong, Benjamin Zi Hao Zhao, Minhui Xue, Haojin Zhu, and Jialiang Lu. “Hidden backdoors in human-centric language models”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021.

[85] Shaofeng Li, Shiqing Ma, Minhui Xue, and Benjamin Zi Hao Zhao. “Deep learning backdoors”. In: *arXiv preprint arXiv:2007.08273* (2020).

[86] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. “Invisible Backdoor Attacks on Deep Neural Networks Via Steganography and Regularization”. In: *IEEE Transactions on Dependable and Secure Computing* (2021).

[87] *Limited access to clipboard data - Privacy Changes in Android 10*. URL: https://developer.android.com/about/versions/10/privacy/changes#clipboard-data.

[88] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable ai: A review of machine learning interpretability methods". In: *Entropy* (2021).

[89] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. "A Review of Android Malware Detection Approaches Based on Machine Learning". In: *IEEE Access* (2020). DOI: `10.1109/ACCESS.2020.3006143`.

[90] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. "From local explanations to global understanding with explainable AI for trees". In: *Nature machine intelligence* (2020).

[91] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Proceedings of the International Conference on Neural Information Processing Systems*. 2017.

[92] Hua Ma, Huming Qiu, Yansong Gao, Zhi Zhang, Alsharif Abuadbba, Anmin Fu, Said Al-Sarawi, and Derek Abbott. "Quantization Backdoors to Deep Learning Models". In: *arXiv preprint arXiv:2108.09187* (2021).

[93] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. "Deepct: Tomographic combinatorial testing for deep learning systems". In: *Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2019, pp. 614–618.

[94] *Machine Learning in Cybersecurity | Kaspersky*. URL: `https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity`.

[95] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards deep learning models resistant to adversarial attacks". In: *Proceedings of the International Conference on Learning Representations*. 2018.

[96] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. "Stealth attacks: An extended insight into the obfuscation effects on Android malware". In: *Computers & Security* (2015).

[97] *Malware Statistics & Trends Report | AV-ATLAS*. URL: `https://www.av-test.org/en/statistics/malware/`.

[98] Nuno Martins, José Magalhães Cruz, Tiago Cruz, and Pedro Henriques Abreu. "Adversarial machine learning applied to intrusion and malware scenarios: a systematic review". In: *IEEE Access* (2020).

[99] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaeisemnani, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail Joon Ahn. "Deep Android Malware Detection". In: *Proceeding of the ACM Conference on Data and Applications Security and Privacy (CODASPY) 2017*. 2016.

[100] Abdelmonim Naway and Yuancheng LI. *Using Deep Neural Network for Android Malware Detection*. 2019. arXiv: `1904.00736 [cs.CR]`.

[101] *OkHttp*. URL: `https://square.github.io/okhttp/`.

[102] *Open Malware*. URL: `http://openmalware.org/`.

[103] Neel Patel, Reza Shokri, and Yair Zick. "Model explanations with differential privacy". In: *arXiv preprint arXiv:2006.09129* (2020).

[104] *pefile*. URL: https://github.com/erocarrera/pefile.

[105] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. “DeepXplore: Automated Whitebox Testing of Deep Learning Systems”. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017.

[106] Zirui Peng, Shaofeng Li, Guoxing Chen, Cheng Zhang, Haojin Zhu, and Minhui Xue. “Fingerprinting Deep Neural Networks Globally via Universal Adversarial Perturbations”. In: *arXiv preprint arXiv:2202.08602* (2022).

[107] Li Pengcheng, Jinfeng Yi, and Lijun Zhang. “Query-efficient black-box attack by active learning”. In: *Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018.

[108] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. “Intriguing properties of adversarial ML attacks in the problem space”. In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. 2020.

[109] Matteo Pomilia. “A study on obfuscation techniques for Android malware”. In: *Sapienza University of Rome* (2016).

[110] *Privacy changes in Android 10*. URL: https://developer.android.com/about/versions/10/privacy/changes#data-ids.

[111] *PyTorch*. URL: https://pytorch.org/mobile/home/.

[112] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. “Malware detection by eating a whole EXE”. In: *arXiv preprint arXiv:1710.09435* (2017).

[113] *RelocBonus*. URL: https://github.com/nickcano/RelocBonus.

[114] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. 2016.

[115] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why should I trust you?”: Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016.

[116] Joshua Saxe and Konstantin Berlin. “Deep neural network based malware detection using two dimensional binary program features”. In: *Proceedings of the 2015 10th international conference on malicious and unwanted software (MALWARE)*. IEEE. 2015.

[117] Jeffrey C Schlimmer and Richard H Granger. “Incremental learning from noisy data”. In: *Machine learning* (1986).

[118] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. “Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers”. In: *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*. 2021.

[119] Maryam Shahpasand, Len Hamey, Dinusha Vatsalan, and Minhui Xue. “Adversarial attacks on mobile malware detection”. In: *Proceedings of the International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*. 2019.

[120] Lloyd S Shapley. *17. A value for n-person games*. Princeton University Press, 2016.

[121] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. "Interpreting the latent space of gans for semantic face editing". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9243–9252.

[122] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *Proceedings of the International Conference on Machine Learning*. 2017.

[123] SkyLight. *Cylance, I kill you!* URL: https://skylightcyber.com/2019/07/18/cylance-i-kill-you/.

[124] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. "Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. 2020.

[125] *Smali*. URL: https://github.com/JesusFreke/smali.

[126] *Sogou Input Editor - Google Play*. URL: https://play.google.com/store/apps/details?id=com.sohu.inputmethod.sogou.

[127] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. "MAB-Malware: A reinforcement learning framework for attacking static malware classifiers". In: *ACM ASIA Conference on Computer and Communications Security (ASIACCS)* (2021).

[128] *Sonicwall research malware attacks 2019*. URL: https://www.msspalert.com/cybersecurity-research/sonicwall-research-malware-attacks-2019/.

[129] Hamish Spencer, Wei Wang, Ruoxi Sun, and Minhui Xue. "Dissecting Malware in the Wild". In: *arXiv preprint arXiv:2111.14035* (2021).

[130] Octavian Suciu, Scott E Coull, and Jeffrey Johns. "Exploring adversarial examples in malware detection". In: *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2019.

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

[132] Ruoxi Sun, Wei Wang, Minhui Xue, Gareth Tyson, Seyit Camtepe, and Damith Ranasinghe. "Vetting Security and Privacy of Global COVID-19 Contact Tracing Applications". In: *arXiv preprint arXiv:2006.10933* (2020).

[133] Ruoxi Sun and Minhui Xue. "Quality Assessment of Online Automated Privacy Policy Generators: An Empirical Study". In: *Proceedings of the Evaluation and Assessment in Software Engineering*. 2020, pp. 270–275.

[134] Suibin Sun, Le Yu, Xiaokuan Zhang, Minhui Xue, Ren Zhou, Haojin Zhu, Shuang Hao, and Xiaodong Lin. "Understanding and Detecting Mobile Ad Fraud Through the Lens of Invalid Traffic". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. 2021.

[135] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic attribution for deep networks". In: *Proceedings of the International Conference on Machine Learning*. 2017.

[136] Zhushou Tang, Ke Tang, Minhui Xue, Yuan Tian, Sen Chen, Muhammad Ikram, Tielei Wang, and Haojin Zhu. "iOS, Your OS, Everybody's OS: Vetting and Analyzing Network Services of iOS Applications". In: *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 2415–2432.

[137] Microsoft Defender Security Research Team. *New machine learning model sifts through the good to unearth the bad in evasive malware*. 2019. URL: https://www.microsoft.com/security/blog/2019/07/25/new-machine-learning-model-sifts-through-the-good-to-unearth-the-bad-in-evasive-malware/.

[138] *TensorFlow Lite | ML for Mobile and Edge Devices*. URL: https://www.tensorflow.org/lite.

[139] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, et al. "Data breaches, phishing, or malware? Understanding the risks of stolen credentials". In: *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017, pp. 1421–1434.

[140] Romain Thomas. *LIEF - Library to instrument executable formats*. 2017. URL: https://lief.quarkslab.com/.

[141] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. "Deep reinforcement learning for general video game ai". In: *Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2018, pp. 1–8.

[142] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. "Survey of machine learning techniques for malware analysis". In: *Computers & Security* (2019). DOI: 10.1016/j.cose.2018.11.001. URL: http://dx.doi.org/10.1016/j.cose.2018.11.001.

[143] Giulia Vilone and Luca Longo. *Explainable Artificial Intelligence: a Systematic Review*. 2020. arXiv: 2006.00093 [cs.AI].

[144] R Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. "Robust intelligent malware detection using deep learning". In: *IEEE Access* (2019).

[145] R Vinayakumar, KP Soman, Prabaharan Poornachandran, and S Sachin Kumar. "Detecting Android malware using long short-term memory (LSTM)". In: *Journal of Intelligent & Fuzzy Systems* (2018).

[146] *Virbox protector*. URL: https://www.sense.com.cn/VirboxProtector.html.

[147] *VirusShare*. URL: https://www.virusshare.com.

[148] *VirusTotal*. URL: https://www.virustotal.com.

[149] *VMProject software protection*. URL: https://vmpsoft.com/.

[150] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". In: *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*. 2019.

[151] Rongrong Wang, Minhui Xue, Kelvin Liu, and Haifeng Qian. "Data-driven privacy analytics: A WeChat case study in location-based social networks". In: *Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications*. Springer. 2015, pp. 561–570.

[152] Wei Wang, Ruoxi Sun, Tian Dong, Shaofeng Li, Minhui Xue, Gareth Tyson, and Haojin Zhu. “Exposing Weaknesses of Malware Detectors with Explainability-Guided Evasion Attacks”. In: *arXiv preprint arXiv:2111.10085* (2021).

[153] Zhi Wang, Chaoge Liu, and Xiang Cui. “EvilModel: Hiding Malware Inside of Neural Network Models”. In: *Proceedings of the 2021 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2021.

[154] Jialin Wen, Benjamin Zi Hao Zhao, Minhui Xue, Alina Oprea, and Haifeng Qian. “With great dispersion comes greater resilience: efficient poisoning attacks and defenses for linear regression models”. In: *IEEE Transactions on Information Forensics and Security* (2021).

[155] Jialin Wen, Benjamin Zi Hao Zhao, Minhui Xue, and Haifeng Qian. “PALOR: Poisoning attacks against logistic regression”. In: *Proceedings of the Australasian Conference on Information Security and Privacy*. Springer. 2020.

[156] Emily Wenger, Josephine Passananti, Arjun Nitin Bhagoji, Yuanshun Yao, Haitao Zheng, and Ben Y Zhao. “Backdoor attacks against deep learning systems in the physical world”. In: *Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.

[157] Davey Winder. *Apple iOS 14 exposes Microsoft’s LinkedIn app reading clipboard data*. 2020. URL: https://www.forbes.com/sites/daveywinder/2020/07/04/apple-ios-14-catches-microsofts-linkedin-spying-on-clipboard-tiktok-apps-privacy-iphone-ipad-macbook/#70bf4e695896.

[158] Cangshuai Wu, Jiangyong Shi, Yuexiang Yang, and Wenhua Li. “Enhancing Machine Learning Based Malware Detection Model by Reinforcement Learning”. In: *Proceedings of the 8th International Conference on Communication and Network Security*. 2018.

[159] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. “Deephunter: a coverage-guided fuzz testing framework for deep neural networks”. In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2019, pp. 146–157.

[160] Jing Xu, Minhui Xue, and Stjepan Picek. “Explainability-based backdoor attacks against graph neural networks”. In: *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*. 2021, pp. 31–36.

[161] Ke Xu, Yingjiu Li, and Robert H. Deng. “ICCDetector: ICC-Based Malware Detection on Android”. In: *IEEE Transactions on Information Forensics and Security* (2016).

[162] Ke Xu, Yingjiu Li, and Robert H. Deng. “ICCDetector: ICC-based malware detection on Android”. In: *IEEE Transactions on Information Forensics and Security* (2016).

[163] Ke Xu, Yingjiu Li, Robert H. Deng, and Kai Chen. “DeepRefiner: Multi-layer Android malware detection system applying deep neural networks”. In: *Proceedings of the IEEE European Symposium on Security and Privacy*. 2018.

[164] Weilin Xu, Yanjun Qi, and David Evans. “Automatically evading classifiers”. In: *Proceedings of the Network and Distributed Systems Symposium*. 2016.

[165] Lok Kwong Yan and Heng Yin. “DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis”. In: *Proceedings of the 21st USENIX Security Symposium (USENIX Security 12)*. 2012.

[166] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. “CADE: Detecting and Explaining Concept Drift Samples for Security Applications”. In: *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*. 2021.

[167] Wei Yang, Deguang Kong, Tao Xie, and Carl A Gunter. “Malware detection in adversarial settings: Exploiting feature evolutions and confusions in Android apps”. In: *Proceedings of the Annual Computer Security Applications Conference*. 2017.

[168] Yufei Yang, Wenbo Luo, Yu Pei, Minxue Pan, and Tian Zhang. “Execution enhanced static detection of Android privacy leakage hidden by dynamic class loading”. In: *Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 149–158.

[169] *YARA - The pattern matching swiss knife for malware researchers*. URL: https://virustotal.github.io/yara/.

[170] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. “Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014.

[171] Zhaoqi Zhang, Panpan Qi, and Wei Wang. “Dynamic malware analysis with feature engineering and feature learning”. In: *Proceedings of the 2020 AAAI Conference on Artificial Intelligence*. 2020.

[172] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. “Structural Attack against Graph Based Android Malware Detection”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 3218–3235.

[173] Chunyi Zhou, Yansong Gao, Anmin Fu, Kai Chen, Zhiyang Dai, Zhi Zhang, Minhui Xue, and Yuqing Zhang. “PPA: Preference Profiling Attack Against Federated Learning”. In: *arXiv preprint arXiv:2202.04856* (2022).

[174] Tong Zhu, Yan Meng, Haotian Hu, Xiaokuan Zhang, Minhui Xue, and Haojin Zhu. “Dissecting Click Fraud Autonomy in the Wild”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 271–286.